Analyse de méthodes de résolution parallèles d'EDO/EDA raides

David Guibert

THESE
présentée
devant l'UNIVERSITE CLAUDE BERNARD - LYON 1
pour l'obtention
du DIPLOME DE DOCTORAT
(arrêté du 25 avril 2002)
en MATHEMATIQUES APPLIQUEES

présentée et soutenue publiquement le
10/09/2009

Analyse de méthodes de résolution parallèles d'EDO/EDA raides

par

David Guibert

Dirigée par Damien Tromeur-Dervout

JURY:
Mme Jocelyne Erhel, Rapporteur
M. Luc Giraud, Rapporteur
M. Bruno Lacabanne
M. Damien Tromeur-Dervout, Directeur de thèse

Table des matières

CHAPITRE 1

Introduction

La simulation numérique est devenue partie intégrante du processus de développement et de mise au point des systèmes mécaniques complexes. La description d'un système dans sa globalité conduit à une modélisation 1D (temporelle), ou modélisation système. Cette approche est nécessairement basée sur une modélisation plus ou moins idéalisée des composants formant le modèle global. Les besoins en innovation et les exigences de plus en plus fortes imposées aux produits en terme de poids, de consommations, d'émissions, etc, ne peuvent être atteints qu'en intégrant le plus en amont possible l'optimisation dans le processus de conception. Dans cette perspective, les modélisations 1D permettent de valider directement les impacts des modifications de conception. Elles permettent ainsi d'intégrer dans la modélisation des sous-modèles de plusieurs disciplines (thermique, hydraulique, traitement du signal,...).

Chaque sous-modèle consiste en un système d'équations différentielles soit ordinaires (EDO), soit le plus souvent algébriques (EDA). L'intégration de plus en plus de physique conduit sur un modèle ayant un nombre de plus en plus grand d'inconnues et dont la solution nécessite de devoir simuler un système d'équations différentielles non linéaires dont la raideur évolue avec les dynamiques des différents sous-modèles. L'augmentation de ces deux facteurs, raideurs et nombre d'inconnues, impactent directement sur le temps de calcul. L'ingénieur concepteur est donc confronté au dilemme d'avoir un modèle suffisamment riche en terme de physiques prises en compte pour être réaliste et celui d'obtenir la solution en un temps raisonnable pour pouvoir agir sur l'optimisation de la conception. Ces contraintes sont tant plus fortes dans les simulations en temps-réel comme par exemple la conception de composants embarqués sur des bancs d'essai (conception hardware-in-the-loop).

Cette augmentation des temps de calcul constitue donc un verrou majeur pour le développement de l'innovation et le gain de compétitivité qui en découle. Par conséquent il devient impératif de diminuer les temps de calculs des intégrateurs numériques par le développement de techniques d'intégrations numériques qui soient adaptées aux architectures de machines multiprocesseurs et extensibles en terme de nombre de processeurs,

tout en conservant leurs propriétés de consistance et de stabilité.

Ces éléments montrent la nécessité de développer de nouvelles techniques de calcul pour la parallélisation efficace de systèmes d'EDO/EDA non linéaires raides. Les recherches menées au sein de ce document ont donc pour ambition d'apporter quelques solutions à cette problématique de la diminution du temps de calcul des systèmes différentiels.

Citons dans ce domaine, les travaux précurseurs de décomposition des systèmes d'équations en sous systèmes proposés par Skelboe [91, 89, 87] avec un estimateur a priori du découpage. Les méthodes de "Waveform Relaxation" s'inscrivent dans ce cadre. B. Pohl et K. Burrage propose dans [14] une parallélisation par la résolution itérative indépendante des sous-systèmes par un processus de type Jacobi ou Gauss-Seidel. Des résultats intéressants ont été obtenus sur des problèmes particuliers où les dynamiques des solutions sont continues et non très raides. De même, les techniques parallèles $C(p, q, j)$ [41] développées par Garbey et Tromeur-Dervout sont un premier pas pour relaxer les communications entre les termes de couplage dans le cadre de système aux équations aux dérivées partielles (EDP) modélisant des physiques couplées. Ces méthodes s'appuient sur des prédictions des termes de couplage à partir d'une extrapolation d'ordre j prise avec un retard p. Elles offrent une alternative aux techniques de séparation du système entre variables lentes et variables rapides que l'on trouve dans les problèmes de chimie en permettant de ne négliger aucun des couplages, ce qui s'avère nécessaire pour traiter des problèmes très raides.

Plus particulièrement, les systèmes d'EDA, mis en jeux par exemple dans le secteur de l'automobile, nécessitent le développement de technologies logicielles numériques spécifiques. En effet, la gestion des discontinuités, la raideur des systèmes d'équations, l'adaptivité des pas de temps, les bifurcations éventuelles des solutions et les différences d'échelles de temps des solutions nécessitent des implémentations particulières comme celles rencontrées dans [76] sur des solveurs séquentiels. Ces particularités peuvent donc perturber non négligemment la parallélisation du schéma numérique et conduisent nécessairement à repenser les techniques d'intégration numérique.

Pour les EDP, de nombreuses parallélisation en espace existent. Allant du partitionnement de données comme la distribution par lignes ou colonnes des éléments d'un système linéaire à résoudre, jusqu'aux méthodes de décomposition de domaine comme les méthodes de Schwarz ou de Schur. Ces techniques s'appuient principalement sur les dépendances régulières entre les variables introduites par les schémas de discrétisation rendant la parallélisation relativement aisée. La difficulté majeure dans le présent travail sur les EDO/EDA est la dépendance non régulière des données qui ne permet pas une localisation aisée de la distribution des données et le caractère non linéaire des dynamiques qui peut entrainer des parties non actives dans le système global.

L'autre aspect motivant la recherche sur la parallélisation du schéma d'intégration en temps est l'évolution des architectures de calcul. En effet la mise à disposition de plusieurs centaines, voire milliers, de processeurs fait que la parallélisation en espace trouve sa limite en terme de ratio des temps de communication/calcul. Par exemple Intel annonce des processeurs à 80 cœurs pouvant atteindre 1 Tera opérations par secondes. Devant ces avancées, l'ingénieur ou le chercheur aura bientôt en sa possession une machine personnelle disposant d'un grand nombre de processeurs. Ceci rendra indispensable la prise en compte

d'algorithmes tirant parti des ressources parallèles. Pour une précision donnée, le nombre d'inconnues est fixé donc il est inutile d'augmenter artificiellement ce nombre pour garantir un bon équilibrage entre la charge de calcul et le volume des communications. De nouvelles façons de paralléliser doivent alors être envisagées, comme la décomposition en temps.

Cette problématique de la parallélisation des EDO/EDA va donc de plus en plus se poser et conduira à repenser les méthodes numériques dans le contexte des nouvelles technologies de calcul haute performance. Dans ce document, nous proposons quelques pistes en développant des techniques mathématiques pour le partitionnement dynamique des modèles et des techniques de décomposition aussi bien au niveau des matrices de résolution que de l'intervalle de temps de simulation.

De façon générale, notre problème s'écrit sous forme de systèmes d'équations différentielles ordinaires non linéaires (EDO, équations (1.1))

$$\begin{cases} \dot{y} = f(t, y) \quad \text{sur } [T_0, T] \\ y(T_0) = y_0 \end{cases} \tag{1.1}$$

ou bien d'équations algébriques (EDA, équations (1.2)).

$$\begin{cases} F(t, y, \dot{y}) = 0 \quad \text{sur } [T_0, T] \\ y(T_0) = y_0 \\ \dot{y}(T_0) = \dot{y}_0 \end{cases} \tag{1.2}$$

Mon développement se focalise sur les méthodes parallèles pour la résolution de ces systèmes différentiels suivant deux axes de recherche. Le premier concerne une parallélisation "externe" où l'intervalle du temps de simulation est décomposé en morceaux et le second une parallélisation "interne" où l'intégrateur est parallélisé sans connaissance a priori du modèle simulé.

Dans un premier temps, nous avons jugé nécessaire d'évaluer les parallélisations à travers les méthodes de Runge-Kutta introduites de façon théorique par K. Burrage. Ceci nous permet de disposer d'une base de référence pour évaluer les performances des développements faits par la suite. Dans la partie I, les résultats d'une implémentation à travers la méthode sont obtenus et permettent de mettre en évidence les limites de cette approche en terme d'accélération (ou speed-up).

Ceci nous conduit à investiguer, dans notre premier axe, les méthodes de décomposition en temps : elles proposent de découper l'intervalle d'intégration $[T_0, T]$ en morceaux (sous-intervalles) puis de résoudre sur chacun de ces intervalles et enfin de corriger l'erreur commise par la résolution découplée. Les principales méthodes entrant dans cette catégorie sont les méthodes Pita et Parareal. Nous montrons dans la partie II que ces méthodes peuvent se rassembler sous le formalisme des méthodes de tirs. Ces méthodes font appel à des notions de pas de temps fins et de pas de temps grossiers. Étant donné la nature raide des systèmes différentiels que nous adressons, nous avons dû redéfinir ces notions de grilles en temps fines et grossières. Cependant, les limites en terme d'efficacité parallèle de ces méthodes sur des problèmes raides nous conduisent à étudier les méthodes d'estimation de l'erreur globale du schéma d'intégration. Après un rappel des estimateurs d'erreur avec

l'estimateur de Richardson, nous introduisons une extrapolation des conditions initiales de chaque sous-domaine afin d'éviter la séquentialité de la correction de l'erreur des méthodes précédentes. Enfin nous étudions la méthode du résidu différé et nous proposons une version parallèle de cette méthode par une mise en cascade du calcul.

La partie III correspond à notre second axe. Nous cherchons à étendre les méthodes de décomposition de domaine aux systèmes différentiels. Les problèmes 'industriels' visés étant raides, la contrainte de stabilité des méthodes explicites conduirait à l'utilisation de pas de temps trop petits pour être efficace. Ces problèmes nécessitent donc l'intégration par des méthodes implicites en temps. Le système non linéaire en découlant peut être résolu par une méthode de point fixe ou par une méthode de Newton. Cette dernière est plus souvent privilégiée pour sa vitesse de convergence par rapport aux méthodes de point fixe. Une itération de Newton requière l'inversion d'un système linéaire mettant en jeu la matrice jacobienne. C'est sur cette inversion que nous introduisons la décomposition de domaine de type complément de Schur primal. Pour une décomposition de domaine en espace d'un système d'équations aux dérivées partielles discrétisé par éléments ou volumes finis, les dépendances de données sont explicites. Cependant, ces dépendances explicites ne sont pas disponibles entre les variables des systèmes différentiels construits automatiquement à partir de sous-modèles simulant des physiques différentes.

Une analyse a priori des dépendances est introduite pour permettre de construire la matrice d'adjacence du graphe de dépendances. Elle fournit ce que nous appellerons un masque. Ce masque nous autorise une réduction importante du coût de construction de la matrice Jacobienne par différentiation numérique en ne traitant que les éléments susceptibles d'être non nuls. Ensuite, au niveau de la résolution, le masque est utilisé pour partitionner les inconnues en inconnues internes et inconnues interface dans la méthode du complément de Schur. Nous montrons des gains en terme d'efficacité parallèle par cette approche par rapport aux méthodes directes type pivot de Gauss.

Cependant, les techniques Jacobian Free Newton-Krylov se montrent encore plus performantes. Le problème linéarisé est résolu par une méthode de Krylov qui ne nécessite que l'action du produit de la matrice jacobienne avec un vecteur. Cette action peut être réalisée sans la construction explicite de la matrice Jacobienne mais par la différentiation de la fonction du système dans la direction du vecteur. Nous avons alors développé le cadre théorique pour combiner la méthode du complément de Schur avec l'approche Matrix Free. Le long processus de validation en terme de stabilité et de consistance de ces méthodes du complément de Schur (avec ou sans construction explicite de la matrice) a été réalisé sur les cas tests fournis par le partenaire industriel.

Pour bien appréhender la difficulté de la parallélisation, il nous a paru important de faire une introduction à la problématique du calcul distribué.

Introduction : la problématique du calcul distribué

Les méthodes dites performantes numériquement sont généralement implicites et adaptatives. Elles font appel à des dépendances de données non locales et mettent en jeux des structures de données irrégulières pour limiter la taille des systèmes linéaires ou non linéaires à résoudre.

Cependant les machines parallèles sont malheureusement plus efficaces lorsqu'elles :
- traitent des données de nature régulières ;
- ont des relations de dépendances aussi locales que possibles ;
- traitent des problèmes de taille la plus importante possible de sorte d'occuper toute la mémoire disponible sans swaper.

Ceci a bien été mis en évidence dans les travaux de Gustafson et de ses collaborateurs [30]. Elles sont d'autant plus performantes que le traitement de ces données est explicite.

Pour obtenir le maximum de performances numériques, les méthodes de résolution de systèmes d'EDO/EDA ont été développées avec
- une intégration en temps implicite pour passer les contraintes de stabilité sur le pas de temps ;
- une adaptativité sur le pas de temps pour contrôler l'erreur sur la solution ;
- une taille limitée définie par le modèle.

On a donc un antagonisme entre la performance numérique et la performance parallèle. Les développements de l'analyse numérique parallèle ont donc pour objet de faire évoluer les algorithmes séquentiels performants, adaptatifs et précis en fonction des contraintes des machines parallèles.

Une machine parallèle peut se caractériser en première approximation par trois données fondamentales que sont :
- le temps de latence pour établir une requête sur une donnée ;
- la bande passante de communication du réseau reliant les processeurs et/ou mémoires qui indique la vitesse de transfert des données en nombre d'octets par seconde ;
- et enfin le nombre d'opérations flottantes par seconde des processeurs de la machine.

Ces trois caractéristiques de l'architecture influent sur les performances des algorithmes numériques. Ainsi, la mise en œuvre d'une parallélisation efficace d'une méthode numérique aura pour objectif de minimiser le coût de calcul combiné au coût de l'accès aux données : en conséquence, la méthode choisie ne sera pas forcément la même suivant que l'on dispose d'une machine parallèle intégrée avec un temps de latence court et une bonne vitesse de communication, ou bien d'une ferme de stations de travail reliées par un réseau local lent. Toutefois, on assiste à une homogénéisation des calculateurs parallèles vers une architecture hybride de type multicluster ; c'est-à-dire des serveurs multiprocesseurs à mémoire partagée, constitués de quelques processeurs, connectés par des réseaux de communication rapides. Le développement et la disponibilité des réseaux hauts débits, dédiés ou non dédiés, permettent une extension de ce concept en une ferme de serveurs distants où chaque serveur est un super-calculateur de plusieurs centaines de processeurs. Ces réseaux de communication ont par nature un temps de latence très important et une bande passante limitée [41].

Plus précisément, une machine parallèle peut être vue comme une hiérarchie de mémoires, cache primaire, cache secondaire, mémoire partagée, mémoire distribuée, adressage global de la mémoire, disques externes, réseaux de communications lents. Le coût d'accès aux données dépend du type de mémoires accédé comme le décrit le Tableau 1.1 inspiré de [75]. Chaque type de mémoire aura un temps de latence et une bande passante propre. Une des conséquences les plus connues de cette hiérarchie de mémoires est l'influence de la distribution des données sur les performances du code de calcul. En Fortran (resp. en

Niveau	1	2	3	4	5
Nom	Registres	Cache	Mémoire principale	Réseau	Sauvegarde disque
Taille	< 32 KB	< 8 MB	< 16 GB	-	> 1 TB
Technologie	CMOS ou BiCMOS	On-chip ou Off-chip	CMOS SRAM CMOS DRAM	ATM, FDDI, memory channel, Crossbar	Disque magnétique ou optique
Temps d'accès (ns)	2-5	3-10	80-400	1000 - 100000	5 000 000
Bande passante (MO/s)	4000-32 000	800-5000	400-2000	10-800	4-32

TABLE 1.1 – Hiérarchies mémoires, temps d'accès et bande passante.

C), les partitions où l'on distribue des blocs de colonnes (resp. lignes) donnent lieu aux calculs (et envois de messages) les plus performants car les matrices sont stockées par colonnes (resp. lignes) [30].

Le facteur essentiel de limitation des performances des algorithmes numériques n'est donc pas en général le temps de traitement de l'opération mais bien la contrainte des temps d'accès aux données. Les nouveaux algorithmes numériques pour machines parallèles sont essentiellement développés pour minimiser ces contraintes de coût d'accès aux mémoires.

Cette contrainte d'accès aux données est d'autant plus présente pour les algorithmes d'intégration en temps, car par nature les algorithmes actuels nécessitent la valeur de la solution sur le pas de temps précédemment calculé (ou plus en fonction de la méthode). Ainsi les accès sont séquentiels sur la donnée venant juste d'être calculée.

Un algorithme numérique de simulation d'un modèle plus ou moins complexe possède de nombreuses sources de parallélisme. On distingue des niveaux de parallélisme à gros grain ou à grain fin, selon que le degré de parallélisme croît avec la taille des données ou bien est fixé à priori. La façon la plus moderne d'obtenir des algorithmes numériques parallèles consiste donc à utiliser ces niveaux complémentaires de parallélisme et à définir une hiérarchie selon la granularité du parallélisme. Cette granularité doit si possible se calquer à la hiérarchie mémoire de l'ordinateur. Cette tâche peut être grandement simplifiée par l'utilisation de bibliothèques de calcul. Cependant, le problème de l'interfaçage entre ces bibliothèques peut être très délicat. En effet, elles sont souvent développées avec leurs propres hypothèses sur la structure et la distribution des données.

L'étude théorique de complexité d'un algorithme numérique est nécessaire, mais malheureusement insuffisante pour décider de la valeur de l'algorithme proprement dit. Les machines parallèles sont suffisamment sophistiquées pour que seule la réelle implémentation des algorithmes permette de juger des résultats. Cependant les performances des machines parallèles sont suffisamment variables d'un type de machine à un autre pour qu'il ne soit pas aisé d'établir une hiérarchie absolue entre les performances des différents algorithmes numériques.

Pour estimer les capacités d'un code parallèle sont introduites les définitions suivantes

Définition 1.1 (Speed-up)**.** Le speed-up est le rapport des temps d'exécution d'un programme entre la version séquentielle et la version parallèle.

Définition 1.2 (Efficacité)**.** L'efficacité est le rapport du speed-up par le nombre de processeurs.

Le speed-up mesure l'effet d'accélération introduit par l'exécution sur plusieurs processeurs. En théorie le speed-up est calculé par rapport à la version séquentielle la plus efficace connue.

Un code idéal est un code qui tourne p fois plus vite (ou plus) sur p processeurs, c'est à dire un code dont le speed-up vaut p (ou supérieur, on parle alors de speed-up superlinéaire), ou bien dont l'efficacité est égale (ou supérieure) à 1.

Les lois d'Amdhal et de Gustafson [30] constituent deux points de vue radicalement opposés de la mesure de performance d'une méthode. Ces lois montrent que, selon que l'on parte d'un code séquentiel dont on mesure la partie intrinsèquement séquentielle de l'algorithme ou que l'on parte d'un code parallèle dont on mesure la performance sur une machine parallèle en utilisant toute la mémoire disponible, on obtient une évaluation de l'algorithme numérique très différente.

CHAPITRE 2

Modélisation de systèmes complexes

Ce chapitre introduit plusieurs problèmes conduisant à des systèmes d'équations différentielles dont la raideur et la complexité croissent. Ce sont des problèmes nous servant de cas tests pour les méthodes parallèles que nous allons développer dans la suite.

Soit un système d'équations différentielles ordinaires, en abrégé EDO, de la forme

$$\dot{y}(t) = f(t, y(t)) \tag{2.1}$$

où écrit composante par composante :

$$\begin{cases} \dot{y}_1(t) = f_1(t, y_1(t), \ldots, y_n(t)) \\ \dot{y}_2(t) = f_2(t, y_1(t), \ldots, y_n(t)) \\ \vdots \\ \dot{y}_n(t) = f_n(t, y_1(t), \ldots, y_n(t)) \end{cases}$$

Afin d'obtenir une solution unique, le modèle différentiel doit être écrit

- soit comme un problème à valeur initiale (IVP, initial value problem) ou problème de Cauchy

$$y(t_0) = y_0 \tag{2.2}$$

  à l'instant initial t_0.
- soit comme un problème aux conditions limites (BVP, boundary value problem)

$$r(y(t_0), y(t_f)) = 0 \tag{2.3}$$

  sur l'intervalle $[T_0, T]$ pour une fonction frontière donnée r.

Lorsque le système modélisé présente des contraintes, le système différentiel obtenu est dit algébrique et s'écrit sous la forme

$$F(\dot{y}, y(t), t) = 0 \tag{2.4}$$

Définition 2.1. [53, p. 455] Le système $F(\dot{y}, y(t), t) = 0$ a un indice de différentiation id si id est le plus petit nombre de différentiations analytiques

$$F(\dot{y}, y) = 0, \qquad \frac{dF(\dot{y}, y)}{dt} = 0, \qquad \ldots, \qquad \frac{d^{id}F(\dot{y}, y)}{dt^{id}} = 0 \tag{2.5}$$

nécessaires pour extraire par manipulations algébriques le système différentiel ordinaire explicite $\dot{y} = f(y)$, appelé aussi système d'EDO sous-jacent.

Si l'équation (2.4) est d'indice 1, elle peut s'écrire

$$\begin{cases} \dot{y}_1(t) = f_1(t, y_1(t), \ldots, y_n(t)) \\ \dot{y}_2(t) = f_2(t, y_1(t), \ldots, y_n(t)) \\ \vdots \\ \dot{y}_m(t) = f_m(t, y_1(t), \ldots, y_n(t)) \\ \\ 0 = g_1(t, y_1(t), \ldots, y_n(t)) \\ 0 = g_2(t, y_1(t), \ldots, y_n(t)) \\ \vdots \\ 0 = g_{n-m}(t, y_1(t), \ldots, y_n(t)) \end{cases}$$

Afin d'éclairer ces définitions, plusieurs exemples vont être introduits. Ils serviront de base de comparaison pour les méthodes de résolution introduites par la suite.

Les problèmes sont de complexité croissante de part leur raideur et leur non-linéarité. Le premier problème est un problème linéaire. Le second est un problème non-linéaire classique, celui de l'évolution conjointe de deux populations, basée sur un modèle de prédation. Ce problème est non-linéaire avec des dynamiques fortes. Le suivant est un problème de chimie présentant des dynamiques très fortes. Ensuite les deux suivants sont issus de modélisation de phénomènes instationnaires bidimensionnels. La discrétisation spatiale permet facilement d'augmenter le nombre d'inconnues de ce système, facilitant les tests de performances sur un même problème. Enfin le dernier problème est un cas test fourni par notre partenaire industriel dans le but de tester les méthodes sur leurs problèmes qui sont souvent raides, avec de nombreuses discontinuités.

2.1 Système différentiel linéaire

Commençons par le problème de Cauchy le plus simple, le cas d'un problème à valeur initiale linéaire scalaire.

$$\dot{y} = ay + b(t) \tag{2.6}$$

avec $a \in \mathbb{R}$ et b une fonction continue, alors la solution est donnée par

$$y(t) = e^{a(t-t_0)} \left(y_0 + \int_{t_0}^{t} e^{-a(t-t_0)} b(s) ds \right) \tag{2.7}$$

Ce qui, pour b constante, se simplifie

$$y(t) = \left(\frac{b}{a} + y_0\right) e^{a(t-t_0)} - \frac{b}{a} \tag{2.8}$$

Ce problème peut être étendu aux systèmes différentiels linéaires

$$\begin{cases} \dot{y} = Ay, \quad A \in \mathbb{R}^{n\times n}, b \text{ continue} \\ y(t_0) = y_0 \end{cases} \tag{2.9}$$

Supposons la matrice A inversible. Pour les n valeurs propres λ_i de la matrice A, les n vecteurs propres v_i associés peuvent être construits linéairement indépendants. On a ainsi

$$A\,(v_1, \cdots, v_n) = (v_1, \cdots, v_n)\, diag(\lambda_1, \cdots, \lambda_n) \tag{2.10}$$

Si on note V la matrice des vecteurs propres de A

$$V = \Big(v_1, \cdots, v_n\Big) \qquad\qquad D = diag(\lambda_1, \cdots, \lambda_n) \tag{2.11}$$

alors

$$V^{-1}AV = D \tag{2.12}$$

En utilisant la solution transformée

$$z(t) = Vy(t) \tag{2.13}$$

On a

$$\begin{aligned} \dot{z}(t) &= V\dot{y}(t) \\ &= VAy(t) \\ &= diag(\lambda_1, \cdots, \lambda_n)z(t) \end{aligned}$$

dont l'intégration des n équations indépendantes donne

$$z(t) = diag(exp(\lambda_1 t), \cdots, exp(\lambda_n t)z_0 \tag{2.14}$$

ce qui aboutit à

$$y(t) = Vdiag(exp(\lambda_1 t), \cdots, exp(\lambda_n t)V^{-1}y_0 \tag{2.15}$$

$$= Vexp(Dt)V^{-1}y_0 \tag{2.16}$$

$$= exp(At)y_0 \tag{2.17}$$

où $exp(At)$ représente l'exponentielle matricielle, définie par

$$exp(At) = \sum_{k=0}^{\infty} \frac{(At)^k}{k!}$$

2.2 Modèle proie/prédateur

Le modèle se comporte de la façon suivante :
- lorsque la population des proies augmente, celle des prédateurs peut augmenter car elle a plus de proies pour se nourrir ;
- inversement lorsque la population des prédateurs devient trop importante, elle se nourrit plus vite que ne peut se renouveler la population de proies, conduisant au décroissement des deux populations.

Les populations de deux espèces sont notées x et y. Les paramètres de leur lois d'évolution naturelle pour les proies et les prédateurs sont respectivement les cœfficients μ_1 et μ_3. Et les paramètres μ_2 et μ_4, sont les cœfficients définissant les influences inter-populations.

L'équation qui régit ce comportement est la suivante :

$$\begin{cases} \frac{\partial x}{\partial t} = \mu_1 x - \mu_2 xy \\ \frac{\partial y}{\partial t} = -\mu_3 y + \mu_4 xy \end{cases} \tag{2.18}$$

Par exemple pour des paramètres μ égaux à $(1.5, 1, 3, 1)$, et des populations initiales de 10 et 5, les espèces évoluent de manière cyclique comme illustré dans la figure 2.1.

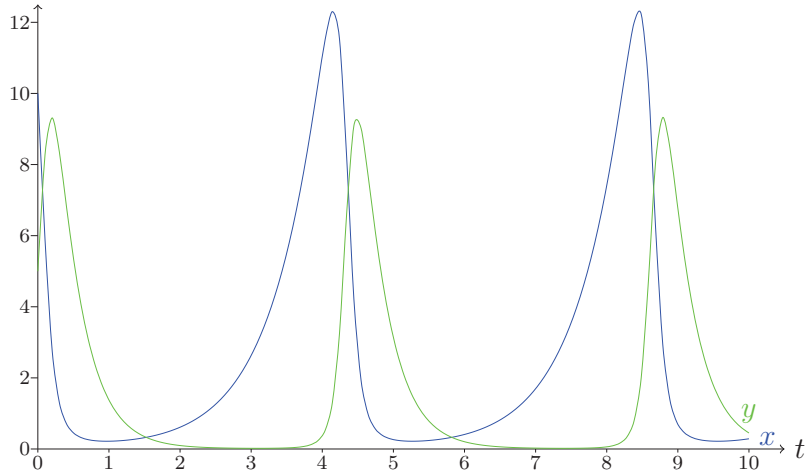


FIGURE 2.1 – Évolution des populations de proie/prédateur pour le problème Lotka-Volterra avec les paramètres $\mu = (1.5, 1, 3, 1)$

Proposition 2.1. *[53] La raideur d'un système de la forme $\dot{y} = f(t, y)$ se caractérise par le fait que la matrice jacobienne de f comprend au moins une valeur propre complexe λ, avec $|\lambda|$ grande.*

La matrice jacobienne de ce problème est

$$\begin{pmatrix} \mu_1 - \mu_2 y & -\mu_2 x \\ \mu_4 & -\mu_3 + \mu_4 x \end{pmatrix}$$

En modifiant les valeurs des paramètres, on influe sur les valeurs propres. En augmentant μ_1 et/ou μ_3 devant μ_2 et μ_4 on augmente la raideur du problème.

2.3 Cinétique de réactions chimiques

Ce problème, issu de la suite de problèmes tests de [66], provient de la célèbre réaction chimique de Belousov Zhabotinskii (BZ). Lorsque certains réactifs, comme l'acide bromique, l'ion bromate et l'ion cérium sont mélangés, ils révèlent une réaction chimique qui, après une période d'induction inactive, oscille avec des changements de structures et de couleurs allant du rouge au bleu et inversement.

Ce changement de couleur est dû à une alternance des oxydations dans lesquelles le cérium bascule entre ses états d'oxydations $Ce(III)$ et $Ce(IV)$.

Le procédé obéit aux trois réactions suivantes :

- la réduction du bromate ($br0_3^-$) en bromine (Br) par l'agent réducteur l'ion bromate (Br^-). L'acide bromomalonic ($BrMA$) est produit ;
- l'augmentation d'acide hypobromus ($HBrO_2$) et la production de $Ce(IV)$. Ce processus provoque un changement brutal de couleur du rouge au bleu ;
- la catalyse du $Ce(IV)$ en $Ce(III)$ ce qui se traduit par un changement graduel de couleur du bleu au rouge.

Les équations chimiques sont alors les suivantes

$$\begin{cases} A + Y \rightarrow X + P & r = k_3 AY \\ X + Y \rightarrow 2P & r = k_2 XY \\ A + X \rightarrow 2X + 2Z & r = k_5 AX \\ 2X \rightarrow A + P & r = k_4 X^2 \\ B + Z \rightarrow \frac{1}{2} fY & r = k_c BZ \end{cases}$$

avec les conventions

- l'acide hypobromus $[HBrO_2] = X$.
- l'acide bromique $[Br^-] = Y$.
- le cérium (IV) $[Ce(IV)] = Z$.
- le bromate $[BrO_3^-] = A$.
- les espèces organiques oxydables $[Org] = B$.
- $[HOBr] = P$.

Les équations chimiques sur les espèces X, Y et Z sont

$$\begin{cases} \mu_2 \dfrac{\partial X}{\partial t} = \mu_1 Y - XY + X(1 - X) \\ \mu_3 \dfrac{\partial Y}{\partial t} = -\mu_1 Y - XY + \mu_4 Z \\ \dfrac{\partial Z}{\partial t} = X - Z \end{cases} \tag{2.19}$$

De même que pour le système proie/prédateur, les paramètres μ ont une influence sur la raideur.

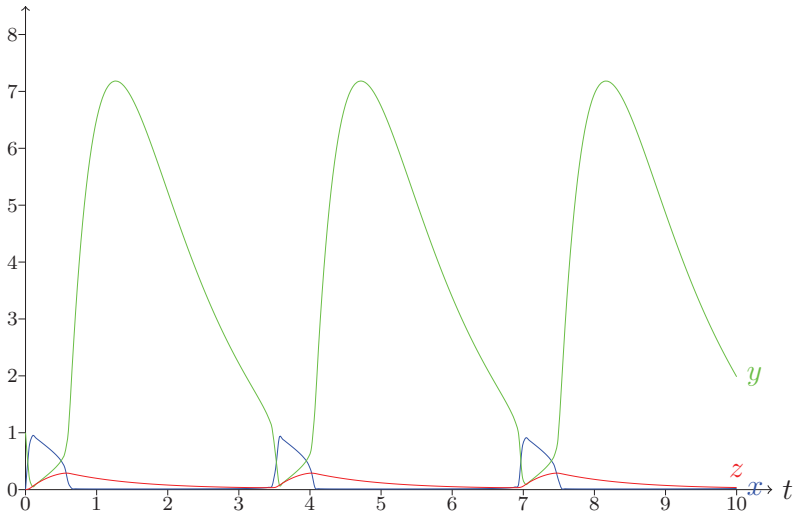


FIGURE 2.2 – Évolution des concentrations pour le problème BZ avec les paramètres $\mu = (0.01, 0.001, 0.01, 1.0)$

2.4 Poutre en flexion

Ce problème est issu de la suite de problèmes tests de [66]. Le problème de poutre en flexion provient de la description mécanique du mouvement d'une poutre élastique mince, supposée inextensible, de longueur 1. De plus la poutre est encastrée à une de ses extrémités et soumise à une force $F = (F_u, F_v)$ sur son extrémité libre. Le système de coordonnées utilisé est l'angle Θ, fonction de la longueur de l'arc s et du temps t. Ainsi la position de la poutre est décrite par les équations suivantes

$$u(s,t) = \int_0^s \cos\theta(\sigma,t)d\sigma \qquad\qquad v(s,t) = \int_0^s \sin\theta(\sigma,t)d\sigma \tag{2.20}$$

Par la théorie de Lagrange, l'énergie cinétique T et l'énergie potentielle U vérifient les relations

$$T = \frac{1}{2}\int_0^1 \left(\left(\frac{du(s,t)}{dt}\right)^2 + \left(\frac{dv(s,t)}{dt}\right)^2\right) ds \tag{2.21}$$

$$U = \frac{1}{2}\int_0^1 \left(\frac{d\theta(s,t)}{ds}\right)^2 ds - F_u(t)u(1,t) - F_v(t)v(1,t) \tag{2.22}$$

Les forces extérieures (F_u, F_v) sont

$$F_u = -\phi(t) \tag{2.23}$$

$$F_v = \phi(t) \tag{2.24}$$

avec

$$\phi(t) = \begin{cases} 1.5\sin^2(t), & 0 \le t \le \pi \\ 0, & t \ge \pi \end{cases}$$

En dérivant les équations du mouvement, on obtient

$$\begin{aligned}&\int_0^1 (1-\max(s,\sigma))\cos(\theta-s,t) - \theta(\sigma,t)\frac{d^2\theta(\sigma,t)}{dt^2}d\sigma \\ &= \frac{d^2\theta(s,t)}{ds^2} + \cos\theta(s,t)F_v(t) - \sin\theta(s,t)F_u(t) \\ &- \int_0^1 (1-\max(s,\sigma))\sin(\theta(s,t)-\theta(\sigma,t))\left(\frac{d\theta(\sigma,t)}{dt}\right)^2 d\sigma\end{aligned} \tag{2.25}$$

Les conditions aux limites sont :

$$\theta(0,t) = 0 \qquad\qquad \frac{d\theta}{ds}(1,t) = 0 \tag{2.26}$$

Enfin une discrétisation par la méthode des lignes — discrétisation des équations différentielles uniquement sur la(les) variable(s) spatiale(s) — est appliquée aux intégrales par la règle du point milieu.

$$\int_0^1 f(\theta(\sigma,t))d\sigma = \frac{1}{N}\sum_{k=1}^{n} f(\theta_k), \qquad \theta_k = \theta\left(\left(k-\frac{1}{2}\right)\frac{1}{n}, t\right), \qquad k = 1,\dots,n \tag{2.27}$$

Les équations (2.25) deviennent

$$\begin{aligned}\sum_{k=1}^{n} a_{lk}\ddot{\theta}_k =& n^4(\theta_{l-1} - 2\theta_l + \theta_{l+1}) + n^2(\cos\theta_l F_v - \sin\theta_l F_u) \\ &- \sum_{k=1}^{n} g_{lk}\sin(\theta_l - \theta_k)\dot{\theta}_k^2, \quad l = 1,\dots,n \\ \theta_0 = -\theta_1, &\quad \theta_{n+1} = \theta_n\end{aligned} \tag{2.28}$$

avec

$$a_{lk} = g_{lk}\cos(\theta_l - \theta_k), \qquad\qquad g_{lk} = n + \frac{1}{2} - \max(l,k) \tag{2.29}$$

Le problème s'écrit sous la forme d'un problème du second ordre

$$\begin{cases}\ddot{\theta} = f(t,\theta,\dot{\theta}) \\ \theta(0) = \theta_0 \\ \dot{\theta}(0) = \dot{\theta}_0 \qquad \text{avec } \theta \in R^n,\ t \geq 0\end{cases} \tag{2.30}$$

La fonction $f : R^n \to R^n$ est définie par $f(t,\theta,\dot{\theta}) = Cv + Du$. Où C est une matrice tridiagonale de taille $n \times n$ dont les éléments sont donnés par

$$\begin{cases}C_{1,1} = 1 & \\ C_{l,l} = 2, & l = 2,\dots,n-1, \\ C_{l,l+1} = -cos(\theta_l - \theta_{l+1}), & l = 1,\dots,n-1, \\ C_{l,l-1} = -cos(\theta_l - \theta_{l-1}), & l = 2,\dots,n \\ C_{n,n} = 3 & \end{cases} \tag{2.31}$$

et D est une matrice bidiagonale de taille $n \times n$ dont les éléments sur- et sous-diagonaux sont

$$\begin{cases} D_{l,l+1} = -sin(\theta_l - \theta_{l+1}), & l = 1, \ldots, n-1, \\ D_{l,l-1} = -sin(\theta_l - \theta_{l-1}), & l = 2, \ldots, n. \end{cases} \tag{2.32}$$

Le vecteur $v = (v_1, \ldots, v_n)^T$ est quant à lui défini par

$$v_l = n^4(\theta_{l-1} - 2z_l + \theta_{l+1}) + n^2(cos(\theta_l)F_v - sin(\theta_l)F_u), \quad l = 1, \ldots, n$$

avec comme précédemment $\theta_0 = -\theta_1$, $\theta_{n+1} = \theta_n$, et u est le vecteur colonne de taille n solution du système linéaire tridiagonal $Cu = g$, avec $g = Dv + ((\dot{\theta}_1)^2, \ldots, \dot{\theta}_n)^2)^T$.

Le problème est ensuite écrit sous la forme d'un problème du premier ordre en définissant $w = \dot{\theta}$, ce qui conduit à un système différentiel ordinaire non linéaire de taille $2n$ de la forme

$$\frac{\partial}{\partial t}\begin{pmatrix} \theta \\ w \end{pmatrix} = \begin{pmatrix} w \\ f(t, \theta, w) \end{pmatrix} \tag{2.33}$$

avec $(\theta, w)^T \in R^{2n}$, $t \geq 0$.. Les conditions initiales obéissent à l'équation suivante

$$\begin{pmatrix} \theta(0) \\ w(0) \end{pmatrix} = \begin{pmatrix} \theta_0 \\ \dot{\theta}_0 \end{pmatrix} \text{ avec } \begin{cases} \theta_0 = (0, \ldots, 0)^T \\ \dot{\theta}_0 = (0, \ldots, 0)^T \end{cases} \tag{2.34}$$

Ce problème est issu d'une discrétisation spatiale. Ceci présente l'avantage de pouvoir augmenter le nombre d'inconnues du système différentiel. Ainsi ce problème peut être utilisé pour tester le comportement parallèle des méthodes.

De plus, l'évaluation de la fonction f est coûteuse car elle requiert l'inversion d'un système linéaire. Ce problème est analogue aux problèmes traités par notre partenaire industriel dans le sens où ils ont des fonctions extrêmement coûteuses.

2.5 Problème Navier-Stokes 2D en formulation tourbillon/fonction de courant

Ce problème est issu de la modélisation de l'écoulement d'un fluide incompressible en 2D ([82]).

Notant la vitesse $\vec{\mathbf{c}} = \begin{pmatrix} U \\ V \end{pmatrix}$ et p la pression du fluide, $\nu = \dfrac{1}{Re}$ la viscosité cinématique du fluide, Re le Reynolds et ρ la masse volumique du fluide.

L'équation de continuité (ou de conservation de la masse) s'écrit

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho \vec{\mathbf{c}}) = 0 \tag{2.35}$$

L'équation de bilan de la quantité de mouvement est

$$\frac{\partial \vec{\mathbf{c}}}{\partial t} + \left(\vec{\mathbf{c}} \cdot \overrightarrow{\nabla}\right) \vec{\mathbf{c}} = -\frac{1}{\rho}\overrightarrow{\nabla} p + \nu \nabla^2 \vec{\mathbf{c}} + \vec{f} \tag{2.36}$$

On considère par la suite un fluide incompressible, c'est-à-dire avec ρ constante. L'équation de continuité devient

$$\overrightarrow{\nabla} \cdot \vec{\mathbf{c}} = 0 \tag{2.37}$$

après adimentionnement de la masse volumique ρ.

La fonction de courant ψ peut alors être définie telle que

$$\begin{cases} u = \dfrac{\partial \psi}{\partial y} \\ v = -\dfrac{\partial \psi}{\partial x} \end{cases} \tag{2.38}$$

Le tourbillon, dans le cas général (3D), est donné par le rotationnel de la vitesse :

$$\omega = \frac{1}{2} \begin{pmatrix} \dfrac{\partial w}{\partial y} - \dfrac{\partial v}{\partial z} \\ \dfrac{\partial u}{\partial z} - \dfrac{\partial w}{\partial x} \\ \dfrac{\partial v}{\partial x} - \dfrac{\partial u}{\partial y} \end{pmatrix} \tag{2.39}$$

En 2D, par l'invariance selon la troisième composante, le tourbillon est une fonction scalaire définie par

$$\omega = \frac{1}{2} \left(\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right) \tag{2.40}$$

La fonction de courant vérifie alors le problème de Poisson

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \Delta \psi = -\omega \tag{2.41}$$

L'équation qui gouverne le tourbillon s'obtient, quant à elle, en appliquant l'opérateur rotationnel à l'équation (2.36) :

$$\frac{\partial \omega}{\partial t} + \vec{\mathbf{c}} \nabla \omega = \nu \Delta \omega \tag{2.42}$$

Et en remplaçant $\vec{\mathbf{c}}$ par l'expression (2.38), cette équation devient :

$$\frac{\partial \omega}{\partial t} + \frac{\partial \psi}{\partial x} \frac{\partial \omega}{\partial y} - \frac{\partial \psi}{\partial y} \frac{\partial \omega}{\partial x} = \nu \Delta \omega \tag{2.43}$$

Finalement le fluide incompressible modélisé en 2D en formulation tourbillon-fonction de courant est décrit par le système d'équations

$$\begin{cases} \omega_t + \psi_x \omega_y - \psi_y \omega_x - \nu \Delta \omega = 0 & \\ \Delta \psi = -\omega & \\ \psi = 0 & \text{sur } \partial\Omega \\ \dfrac{\partial \psi}{\partial n} = v(t) & \text{sur } \partial\Omega \end{cases} \tag{2.44}$$

Pour mettre le système sous forme d'une EDO, la discrétisation par les différences finies n'est effectuée que sur les variables d'espace (méthode des lignes). Ceci conduit à un problème de Cauchy de la forme suivante lorsque des différences finies d'ordre 2 sont utilisées :

$$\begin{cases} \dfrac{d}{dt}\begin{pmatrix} \omega_{i,j} \\ \psi_{i,j} \end{pmatrix} = f\left(t, \omega_{i[\pm 1],j[\pm 1]}, \psi_{i[\pm 1],j[\pm 1]}\right) \\ \begin{pmatrix} \omega \\ \psi \end{pmatrix}(0) = \begin{pmatrix} \omega_0 \\ \psi_0 \end{pmatrix} \end{cases} \tag{2.45}$$

Comme exemple, on considère le problème de la cavité entrainée dans un carré avec un entrainement constant sur le bord supérieur. Ce problème s'écrit

$$\begin{cases} \dot{\omega}_{ij} = -(\psi_{y_{ij}}\omega_{xij} - \psi_{xij}\omega_{y_{ij}}) + 1/Re(\omega_{xxij} + \omega_{yy_{ij}}) & \left.\begin{matrix} i = 1, \ldots, nx-2, \\ j = 1, \ldots, ny-2 \end{matrix}\right\} \\ \dot{\psi}_{ij} = \psi_x x_{ij} + \psi_y y_{ij} + \omega_{ij} & \\ \dot{\omega}_{ij} = 0 & \\ \dot{\psi}_{ij} = 0 & sinon \\ \omega_{ij}(0) = 0 & \\ \psi_{ij}(0) = 0 & \end{cases} \tag{2.46}$$

avec

– sur le bord $i = 0$:

$$\omega_{ij} = -\frac{2}{\delta_y^2}\psi_{(i+1)j} - \frac{2}{\delta_y}$$

– sur le bord $i = nx - 1$:

$$\omega_{ij} = -\frac{2}{\delta_y^2}\psi_{(i-1)j}$$

– sur le bord $j = 0$:

$$\omega_{ij} = -\frac{2}{\delta_x^2}\psi_{i(j+1)}$$

– sur le bord $j = ny - 1$:

$$\omega_{ij} = -\frac{2}{\delta_x^2}\psi_{i(j-1)}$$

– à l'intérieur du domaine :

$$\begin{cases} \psi_{x_{ij}} = \dfrac{\psi_{i(j+1)} - \psi_{i(j-1)}}{2\delta_x} \\ \psi_{y_{ij}} = \dfrac{\psi_{(i+1)j} - \psi_{(i-1)j}}{2\delta_y} \\ \omega_{x_{ij}} = \dfrac{\omega_{i(j+1)} - \omega_{i(j+1)}}{2\delta_x} \\ \omega_{y_{ij}} = \dfrac{\omega_{(i+1)j} - \omega_{(i-1)j}}{2\delta_y} \\ \omega_{xx_{ij}} = \dfrac{\omega_{i(j-1)} - 2\omega_{ij} + \omega_{i(j+1)}}{\delta_x^2} \\ \omega_{yy_{ij}} = \dfrac{\omega_{(i-1)j} - 2\omega_{ij} + \omega_{(i+1)j}}{\delta_y^2} \\ \psi_{xx_{ij}} = \dfrac{\psi_{i(j+1)} + \psi_{i(j-1)} - 2\psi_{ij}}{\delta_x^2} \\ \psi_{yy_{ij}} = \dfrac{\psi_{(i+1)j} + \psi_{(i-1)j} - 2\psi_{ij}}{\delta_y^2} \end{cases}$$

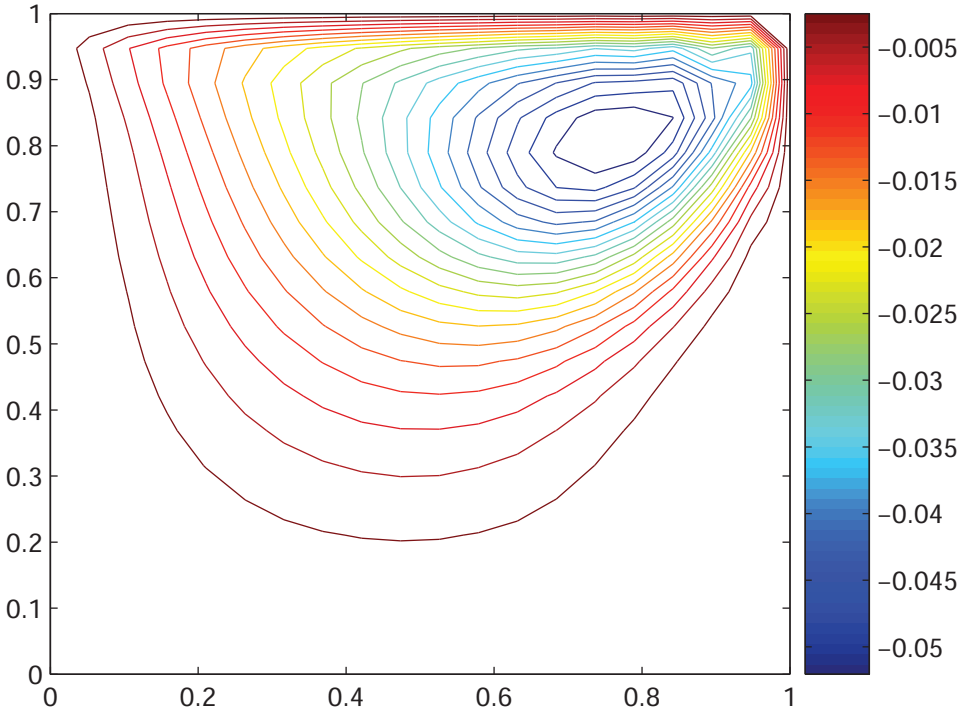


FIGURE 2.3 – isovaleurs de la fonction de courant pour la cavité entrainée avec $Re = 400$ au temps $T = 5$ avec une discrétisation uniforme 40×40

De même que le problème précédent, ce problème est issu d'une discrétisation spatiale. Le nombre d'inconnues, donc la taille du problème, dépend du nombre de points considérés dans la discrétisation. Ce problème, étant 2D, offre un nombre plus important de variables.

2.6 Injection d'un moteur

La modélisation 1D permet de modéliser des problèmes complexes par assemblage de composants simples.

Chaque composant est caractérisé par un ensemble d'équations différentielles ordinaires et/ou de relations algébriques. En couplant les entrées et sorties de chaque composant, le système d'équations est ainsi obtenu.

Le système global correspond à un système d'équations différentielles ordinaires à 287 inconnues. Une simplification de ce système global nous offre un modèle d'injection avec seulement 131 variables d'états. L'un ou l'autre de ces modèles seront utilisés pour nos tests.

Les valves sont soit ouvertes soit fermées. Lorsqu'une valve est fermée, toute une partie du sous-modèle se trouve isolée du reste. Lorsqu'elle s'ouvre, elle densifie les interactions entre les variables.

L'intérêt de ce modèle réside dans la complexité de son intégration car il présente de nombreuses discontinuités provenant de la détermination exacte des instants auxquels les différentes valves changent d'état.

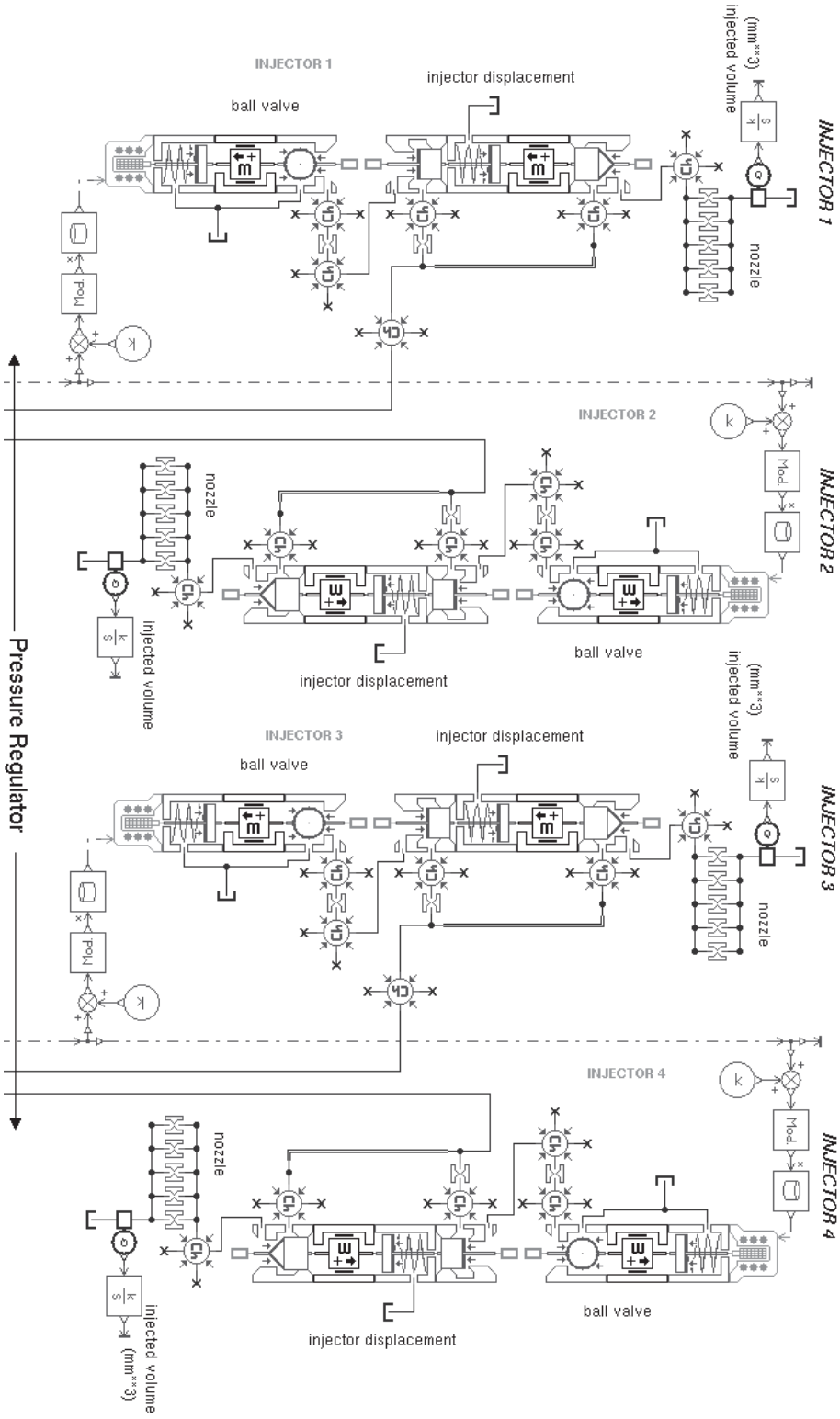


FIGURE 2.4 – Modélisation de la partie injection d'un moteur

Première partie

Parallélisation à travers la méthode

CHAPITRE 3

Les méthodes de Runge-Kutta implicites

Nous allons nous intéresser à la parallélisation à travers la méthode proposé par K. Burrage. Cette parallélisation se base sur les méthodes de Runge-Kutta. Ce chapitre rappelle la définition de ces méthodes et quelques unes de leurs propriétés.

La première méthode de Runge-Kutta (RK) *implicite* fut introduite par Cauchy (1824, [52, Section II, p 204]), en combinant la valeur moyenne dans le problème écrit sous la forme de Picard :

$$y(x_1) = y(x_0) + \int_{x_0}^{x_1} f(x, y(x))dx \tag{3.1}$$

pour obtenir

$$y(x_1) = y(x_0) + hf(x_0 + \theta h, y_0 + \Theta(y_1 - y_0)) \tag{3.2}$$

où $0 \leq \theta, \Theta \leq 1$. Cette méthode est appelée θ-méthode.

Remarque 3.1. Les choix suivants de θ et Θ permettent de retrouver les méthodes classiques :

$\theta = \Theta = 0$ **:** la méthode est la méthode d'Euler explicite.

$$y_1 = y_0 + hf(x_0, y_0)$$

$\theta = \Theta = 1$ **:** la méthode est la méthode d'Euler implicite.

$$y_1 = y_0 + hf(x_1, y_1)$$

$\theta = \Theta = \dfrac{1}{2}$ **:** et en définissant $k_1 = (y_1 - y_0)/h$, on obtient la méthode du point milieu.

$$\begin{cases} k_1 = f\left(x_0 + \dfrac{h}{2}, y_0 + \dfrac{h}{2}\right) \\ y_1 = y_0 + hk_1 \end{cases} \tag{3.3}$$

Une autre façon pour approximer l'équation (3.1) est d'utiliser la méthode des trapèzes — pour l'approximation de l'intégrale —

$$y_1 = y_0 + \frac{h}{2}\left(f(x_0, y_0) + f(x_1, y_1)\right)$$

Le schéma de Radau peut aussi être utilisé

$$\begin{aligned} y(x_1) - y(x_0) &= \int_{x_0}^{x_0+h} f(x, y(x))dx \\ &\approx \frac{h}{4}\left(f(x_0, y_0) + 3f\left(x_0 + \frac{2}{3}h, y\left(x_0 + \frac{2}{3}h\right)\right)\right) \end{aligned}$$

Il reste à approximer y en $x_0 + \frac{2}{3}h$, par exemple en utilisant une interpolation quadratique

$$y\left(x_0 + \frac{2}{3}h\right) \approx \frac{5}{9}y_0 + \frac{4}{9}y(x_1) + \frac{2}{9}hf(x_0, y_0)$$

En posant $y_1 = y_0 + \frac{h}{4}(k_1 + 3k_2)$ et en substituant y_1 dans l'équation précédente, on obtient la méthode de Hammer & Hollingsworth se résumant ainsi

$$\begin{cases} k_1 = f(x_0, y_0) \\ k_2 = f\left(x_0 + \frac{2}{3}h, y_0 + \frac{h}{3}(k_1 + k_2)\right) \\ y_1 = y_0 + \frac{h}{4}(k_1 + 3k_2) \end{cases} \tag{3.4}$$

Le schéma (3.3) est défini par une étape intermédiaire, le calcul de k_1, le schéma précédent (3.4) est défini par deux étapes, k_1 et k_2.

La construction de schémas peut s'étendre afin d'obtenir des schémas à s étapes et d'ordre $2s - 1$.

Définition 3.1. Soient b_i, a_{ij} $(i = 1, \cdots, s)$ réels, et c_i défini par

$$c_i = \sum_{j=1}^{i-1} a_{ij}$$

La méthode

$$\begin{cases} k_i = f\left(x_0 + c_i h, y_0 + \sum_{j=1}^{s} a_{ij}k_j\right), \quad i = 1, \ldots, s \\ y_1 = y_0 + h\sum_{i=1}^{s} b_i k_i \end{cases} \tag{3.5}$$

est appelée *méthode implicite de Runge-Kutta à s-étapes.*

Définition 3.2. Une méthode de RK (3.5) est d'ordre p si pour un problème suffisamment régulier

$$||y(t_0 + h) - y_1|| \leq Kh^{p+1} \tag{3.6}$$

Autrement dit la série de Taylor de la solution exacte $y(t_0+h)$ et y_1 coïncident jusqu'au terme h^p (inclus).

La méthode s'appuie sur $s+1$ formules de quadrature, les s premières sont dites internes et la dernière externe. Elle est généralement représentée par un triplet (A, b, c) où A est la matrice carrée $(a_{ij})_{i,j=1,\dots,s}$ de dimension $s \times s$, b et c sont respectivement les vecteurs de dimension s, $b^T = (b_1, \dots, b_s)$ et $c = (c_1, \dots, c_s)^T$.

Ces méthodes sont représentées sous forme d'un tableau ([18]) :

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} \tag{3.7}$$

Remarque 3.2. En posant $g_i = y_0 + \sum_{j=1}^{s} a_{ij} k_j$, les méthodes de RK sont généralement écrites

$$\begin{cases} g_i = y_0 + h \sum_{j=1}^{s} a_{ij} f(x_0 + c_j h, g_j), \quad i = 1, \dots, s \\ y_1 = y_0 + h \sum_{j=1}^{s} b_j f(x_0 + c_j h, g_j) \end{cases} \tag{3.8}$$

Ceci permet d'écrire les méthodes sous la forme tensorielle suivante

$$\begin{cases} G = I_s \otimes y_0 + h(A \otimes I_s) F(G) \\ y_1 = y_0 + h(b^T F(G)) \end{cases} \tag{3.9}$$

en posant

$$G = \begin{pmatrix} g_1 \\ \vdots \\ g_s \end{pmatrix}, \qquad F(G) = \begin{pmatrix} f(x_0 + c_1 h, g_1) \\ \vdots \\ f(x_0 + c_s h, g_s) \end{pmatrix}$$

avec l'opérateur tensoriel $\otimes$ défini par

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & \dots & a_{1m}B \\ a_{21}B & \dots & \dots & a_{2m}B \\ \vdots & & & \vdots \\ a_{n1}B & \dots & \dots & a_{nm}B \end{pmatrix}$$

3.1 Méthodes de type Radau

Définition 3.3 ([52])**.** Les méthodes de Runge-Kutta d'ordre $2s-1$ basées sur les formules de quadratures de Radau et Lobatto sont des méthodes de type I, II, ou III respectivement si les cœfficients $c_1, \dots, c_s$ sont les zéros des polynômes de degré s

$$I : \frac{d^{s-1}}{dx^{s-1}} \left(x^s (x-1)^{s-1} \right) \tag{3.10}$$

$$II : \frac{d^{s-1}}{dx^{s-1}} \left(x^{s-1} (x-1)^s \right) \tag{3.11}$$

$$III : \frac{d^{s-2}}{dx^{s-2}} \left(x^{s-1} (x-1)^{s-1} \right) \tag{3.12}$$

La méthode de type RadauIIA d'ordre 5 est représenté comme dans (3.7) par le tableau 3.1.

$c_1 = \frac{4-\sqrt{6}}{10}$	$a_{11} = \frac{88-7\sqrt{6}}{360}$	$a_{12} = \frac{296-169\sqrt{6}}{1800}$	$a_{13} = \frac{-2+3\sqrt{6}}{225}$
$c_2 = \frac{4+\sqrt{6}}{10}$	$a_{21} = \frac{269+169\sqrt{6}}{1800}$	$a_{22} = \frac{88+7\sqrt{6}}{360}$	$a_{23} = \frac{-2-3\sqrt{6}}{225}$
$c_3 = 1$	$a_{31} = \frac{16-\sqrt{6}}{36}$	$a_{32} = \frac{16+\sqrt{6}}{36}$	$a_{33} = \frac{1}{9}$
	$b_1 = \frac{16-\sqrt{6}}{36}$	$b_2 = \frac{16+\sqrt{6}}{36}$	$b_3 = \frac{1}{9}$

TABLE 3.1 – méthode RadauIIA d'ordre 5

3.2 Stabilité des méthodes de RK

Définition 3.4. [52] Soit $\dot{y}(t) = f(t, y(t))$ un système d'EDO avec $y(0) = 0$, la condition initiale est dite stable au sens de Liapunov pour une norme donnée si $\forall \epsilon > 0$ il existe $\delta > 0$ tel que $||y(t_0)|| < \delta$ implique $||y(t)|| < \epsilon$, $\forall t > t_0$.

Considérons maintenant un système linéarisé de la forme

$$\begin{cases} \dot{y}_i = \sum_{j=1}^{n} a_{ij} y_j \\ a_{ij} = \dfrac{\partial f}{\partial y_j}(0) \end{cases} \tag{3.13}$$

Théorème 3.1. *Le système linéarisé* (3.13) *est stable (au sens de Liapunov) si et seulement si*

1. *toutes les racines λ de l'équation caractéristique*

$$det(\lambda I - A) = a_0\lambda^n + a_1\lambda^{n-1} + \cdots + a_{n-1}\lambda + a_n = 0 \tag{3.14}$$

   ont une partie réelle négative, $\Re(\lambda) \leq 0$.

2. *toutes les racines multiples ont une partie réelle strictement négative, $\Re(\lambda) < 0$.*

3.2.1 Stabilité des méthodes de RK explicites

Soit $\varphi(t)$ une solution de $\dot{\bar{y}} = f(t, \bar{y})$. En linéarisant f au voisinage de φ, on obtient

$$\dot{\bar{y}}(t) = f(t, \varphi(t)) + \frac{\partial f}{\partial y}(t, \varphi(t))(\bar{y}(t) - \varphi(t)) + \dots \tag{3.15}$$

En notant $y(t) = \bar{y}(t) - \varphi(t)$,

$$\dot{y}(t) = \frac{\partial f}{\partial y}(t, \varphi(t))y(t) + \ldots = J(t)y(t) + \ldots \tag{3.16}$$

En première approximation, si on considère la jacobienne J de f constante et les termes d'erreur négligeables, l'équation s'écrit :

$$\dot{y} = Jy \tag{3.17}$$

En appliquant la méthode d'Euler explicite à (3.17)

$$y_{n+1} = R(hJ)y_n \tag{3.18}$$

avec

$$R(z) = 1 + z \tag{3.19}$$

Le comportement de (3.18) peut être étudié en transformant la jacobienne J. On procède comme pour le problème linéaire (2.9). On suppose que J est supposé diagonalisable avec pour vecteurs propres $v_1, \ldots, v_n$ de valeurs propres associées $\lambda_1, \ldots, \lambda_n$ et que y_0 dans cette base s'écrit

$$y_0 = \sum_{i=1}^{n} \alpha_i v_i$$

Alors

$$y_m = \sum_{i=1}^{n} \left(R(h\lambda_i)\right)^m \alpha_i v_i \tag{3.20}$$

L'expression de y_m dans (3.20) reste bornée pour $m \to \infty$ seulement si pour toutes les valeurs propres λ_i, $|R(h\lambda_i)| < 1$.

Définition 3.5 ([52])**.** La fonction $R(\lambda y)$ est appelée la fonction de stabilité de la méthode. Elle peut être interprétée comme étant la solution numérique après un pas de l'équation test de Dahlquist suivante

$$\begin{cases} \dot{y} = \lambda y \\ y_0 = 1 \end{cases} \tag{3.21}$$

L'ensemble $\mathcal{S}$

$$\mathcal{S} = \{z \in \mathbb{C}, |R(z)| < 1\} \tag{3.22}$$

est appelé domaine de stabilité de la méthode.

Pour la méthode d'Euler explicite, le domaine de stabilité $\mathcal{S}$ s'écrit :

$$\mathcal{S} = \{z \in \mathbb{C}, |R(z)| < 1\} = \{z \in \mathbb{C}, |z - (-1)| < 1\} \tag{3.23}$$

Donc $\mathcal{S}$ est le cercle de centre $(-1, 0)$ et de rayon 1.

3.2.2 Stabilité des méthodes de RK implicites

Commençons par la méthode d'Euler implicite sur le problème de Dahlquist (3.21)

$$y_1 = y_0 + h\lambda y_1 \tag{3.24}$$

Ce qui s'écrit

$$y_1 = R(h\lambda)y_0 \qquad \text{avec} \qquad R(z) = \frac{1}{1-z} \tag{3.25}$$

Pour la méthode d'Euler implicite, le domaine de stabilité $\mathcal{S}$ est l'extérieur du cercle de centre $(+1, 0)$ et de rayon 1. Le domaine de stabilité recouvre la totalité du demi-plan complexe $\Re(z) < 0$, ce qui explique que le méthode d'Euler implicite est inconditionnellement stable.

Proposition 3.1. *Considérons la méthode de Runge-Kutta implicite*

$$g_i = y_0 + h\sum_{j=1}^{s} a_{ij} f(x_0 + c_j h, g_j), \qquad i = 1, \dots, s \tag{3.26}$$

$$y_1 = y_0 + \sum_{j=1}^{s} b_j f(x_0 + c_j h, g_j) \tag{3.27}$$

appliquée à $\dot{y} = \lambda y$, la fonction de stabilité $R(z)$ associée est

$$R(z) = 1 + zb^T(I - zA)^{-1}\mathbb{1} \tag{3.28}$$

où $b^T = (b_1, \dots, b_n)$, $A = (a_{ij})_{i,j=1,\dots,s}$ et $\mathbb{1} = (1, \dots 1)^T$

Démonstration. Le système (3.26) devient linéaire pour le problème de Dahlquist.

$$\begin{cases} g_1 = y_0 + h\lambda \sum_{i=1}^{s} a_{ij} g_j \\ \vdots \\ g_s = y_1 + h\lambda \sum_{i=1}^{s} a_{ij} g_j \end{cases}$$

ce qui s'écrit encore :

$$\begin{pmatrix} g_1 \\ \vdots \\ g_s \end{pmatrix} = \left((I - h\lambda A)^{-1}\mathbf{1}\right) y_0$$

L'équation (3.27) est alors

$$y_1 = y_0 + h\lambda b^T \begin{pmatrix} g_1 \\ \vdots \\ g_s \end{pmatrix} = \left(1 + h\lambda b^T (I - h\lambda A)^{-1}\mathbf{1}\right) y_0$$

□

Proposition 3.2. *La fonction de stabilité $R(z)$ satisfait*

$$R(z) = \frac{\det(I - zA + z\mathbf{1}b^T)}{\det(I - zA)} \tag{3.29}$$

Démonstration. C'est une application directe de la règle de Cramer pour la résolution du système linéaire

$$\begin{pmatrix} I - zA & 0 \\ -zb^T & 1 \end{pmatrix} \begin{pmatrix} g \\ y_1 \end{pmatrix} = \begin{pmatrix} \mathbf{1} \\ 1 \end{pmatrix}$$

□

Définition 3.6. Une méthode dont le domaine de stabilité vérifie

$$\{z, \Re(z) \leq 0\} \subset S \tag{3.30}$$

est dite A-stable.

Définition 3.7. Une méthode est dite L-stable si
- elle est A-stable
- $\lim_{z\to\infty} R(z) = 0$

Proposition 3.3. *Si une méthode de RK implicite avec une matrice A non-singulière satisfait une des conditions suivantes*

$$a_{sj} = b_j, \quad j = 1, \ldots, s \tag{3.31}$$
$$a_{1j} = b_j, \quad j = 1, \ldots, s \tag{3.32}$$

alors $\lim_{z\to\infty} R(z) = 0$.

Les méthodes satisfaisant la condition (3.31) sont dites *stiffly accurate*, précises quelle que soit la raideur.

Les méthodes qui sont considérées dans la suite sont stables (au sens du théorème 3.1). Ce qui nous importe est leur parallélisation. Dans le chapitre suivant, la parallélisation des méthodes de RK implicites est abordée. Puis nous établirons une analogie avec les méthodes DIMSIM.

CHAPITRE 4

Parallélisation à travers les méthodes de RK implicites

Une première proposition de parallélisation de la résolution d'un système d'équations différentielles ordinaires a été introduite par les travaux de K. Burrage [12, 13] vers la fin des années 90. Ces travaux se basent sur une parallélisation "à travers la méthode" d'intégration. Elle s'applique principalement aux méthodes de type Runge-Kutta largement répandues pour la résolution de systèmes d'EDO. Cependant, à notre connaissance, ces travaux sont principalement restés au niveau des résultats de convergence, mais n'ont pas abordé l'aspect efficacité/performance parallèle faute d'une implantation concrète. Il nous a paru important, avant de développer de nouvelles méthodes de parallélisation, d'étudier les possibilités réelles de ces parallélisations à travers la méthode.

Nous montrons dans la section 4.1 l'approche de l'implantation des méthodes de Runge-Kutta. Puis l'extension par une parallélisation à travers la méthode est développée dans la section suivante. Ce chapitre se conclut par les résultats de notre implantation dans la méthode RadauIIA d'ordre 5. Cette méthode a été choisie car elle ne faisait pas partie des méthodes utilisées par notre partenaire industriel. Nous voulions tester cette méthode cumulant les deux avantages suivants. C'est une méthode *stiffly accurate* (cf 3.31). Et elle permet de résoudre les problèmes différentiels ordinaires comme algébriques.

Les développements introduits ici sont transposables à n'importe quelle autre méthode de Runge-Kutta.

4.1 Implémentation des méthodes de Runge-Kutta implicites

La résolution par une méthode de RK implicite à s-stage implique la résolution d'un système non linéaire de dimension $(n \times s) \times (n \times s)$. L'idée principale de la parallélisation à travers la méthode est d'utiliser la structure tensorisée sous jacente aux étapes de Runge-Kutta implicite, pour distribuer les calculs sur plusieurs processeurs. Le détail des calculs est donné ci-après pour la méthode à 3 étapes, RadauIIA d'ordre 5.

4.1.1 Reformulation du système non linéaire

Afin de réduire les instabilités dues aux erreurs d'arrondis, on choisit de traiter des quantités plus faibles ([52]) :

$$z_i = g_i - y_0$$

Le système à résoudre (3.8) devient alors :

$$\begin{cases} z_i = h \sum_{j=1}^{s} a_{ij} f\left(x_0 + c_j h, y_0 + z_j\right), & i = 1, \ldots, s \\ y_1 = y_0 + h \sum_{j=1}^{s} b_j f(x_0 + c_j h, y_0 + z_j) \end{cases}$$

D'un point de vue implémentation, si la matrice A est non singulière, le système peut être écrit sous la forme

$$Z = (A \otimes I) h F(Z) \tag{4.1}$$

où

$$Z = \begin{pmatrix} z_1 \\ \vdots \\ z_s \end{pmatrix} \qquad \text{et} \qquad F(Z) = \begin{pmatrix} f(x_0 + c_1 h, y_0 + z_1) \\ \vdots \\ f(x_0 + c_s h, y_0 + z_s) \end{pmatrix} \tag{4.2}$$

Les s évaluations de fonctions additionnelles pour le calcul de y_1 peuvent être évitées en posant $y_1 = y_0 + \sum d_i z_i$ avec $(d_1, \ldots, d_s) = (b_1, \ldots, b_s) A^{-1}$. Pour RadauIIA à 3-étape, $d = (0, 0, 1)$ convient puisque $b_i = a_{si}$ (cf tab. 3.1). Ainsi

$$y_1 = y_0 + z_s \tag{4.3}$$

4.1.2 Itérations de Newton simplifié

Le système à résoudre pour obtenir les s étapes est donc non linéaire et pourrait être résolu par une méthode du point fixe. Cependant ces méthodes du point fixe ne présentent pas toujours de bons taux de convergence lorsqu'il s'agit d'un problème raide [29]. C'est pourquoi les méthodes de type Newton sont préférées. L'équation (4.1) devient après une linéarisation au premier ordre

$$\begin{cases} \left(I - h(A \otimes J)\right) \Delta Z^k = -Z^k + h(A \otimes I) F(Z^k) \\ Z^{k+1} = Z^k + \Delta Z^k \end{cases} \tag{4.4}$$

où $J = \dfrac{\partial F}{\partial Z}$ est la matrice jacobienne de la fonction F.

C'est sur ce calcul que repose la parallélisation à travers la méthode comme nous allons le montrer.

4.1.3 Système linéaire dans Radau5

Cette parallélisation exploite la structure particulière de la matrice $I - hA \otimes J$. Si la matrice A est inversible, en prémultipliant par la matrice $(hA)^{-1} \otimes I$, l'équation (4.4) devient

$$\begin{cases} \Big((hA)^{-1} \otimes I - I \otimes J\Big)\Delta Z^k = \\ \qquad\qquad - ((hA)^{-1} \otimes I)Z^k + (I \otimes I)F(Z^k) \\ Z^{k+1} = Z^k + \Delta Z^k \end{cases} \tag{4.5}$$

Rappelons que la matrice A donne les poids de la formule d'intégration. Une transformation orthogonale va être effectuée de sorte que la matrice A^{-1} soit équivalente à une matrice Λ simple (diagonale, diagonale par blocs, triangulaire) (Théorème de Schur [95, p. 382]) : il existe une matrice T orthogonale telle que

$$T^{-1}A^{-1}T = \Lambda$$

En notant $W^k = (T^{-1} \otimes I)Z^k$, le système à résoudre devient finalement

$$\begin{cases} (h^{-1}\Lambda \otimes I - I \otimes J)\Delta W^k = \\ \qquad\qquad - h^{-1}(\Lambda \otimes I)W^k + (T^{-1} \otimes I)F\left((T \otimes I)W^k\right) \\ W^{k+1} = W^k + \Delta W^k \end{cases} \tag{4.6}$$

Pour la méthode de Radau à 3-étapes (d'ordre 5), la matrice A^{-1} possède une valeur propre réelle $\hat{\gamma}$ et deux valeurs propres complexes conjuguées $\hat{\alpha} \pm i\hat{\beta}$. Cette situation est typique de la méthode de Runge-Kutta à 3 étapes, et est généralisable aux méthodes de type RadauIIA d'ordre supérieur.

Remarque 4.1. En notant $\gamma = h^{-1}\hat{\gamma}$, $\alpha = h^{-1}\hat{\alpha}$ et $\beta = h^{-1}\hat{\beta}$, la matrice du système peut se mettre sous la forme

$$\left(\begin{array}{c|cc} \gamma I - J & 0 & 0 \\ \hline 0 & \alpha I - J & -\beta I \\ 0 & \beta I & \alpha I - J \end{array}\right)$$

La résolution peut ainsi s'effectuer de manière indépendante pour la partie réelle et partie imaginaire.

4.2 Parallélisation mise en œuvre

Dans le calcul de (4.6), plusieurs étapes du calcul peuvent être parallélisées :
– l'application de la fonction F

$$F\left((T \otimes I)W^k\right) = F\left(Z^k\right)$$

– le calcul de $(T^{-1} \otimes I)F\left((T \otimes I)W^k\right)$

4.2.1 Calcul de $F\left(Z^k\right)$

Les évaluations des fonctions $F\left(Z^k\right)$ pouvant être coûteuses dans les problèmes industriels, nous avons également cherché à les paralléliser. La valeur de $F(z^k_{me+1})$ est calculée sur le processeur me, $me = 0, \cdots, np-1$ où np, le nombre de processeurs, est égal à s le nombre d'étapes de la méthode.

Algorithme 1 Évaluation de $F\left(Z^k\right)$

1: **for** $i = 1$ à 3 ou en parallèle $i = me$ **do**
2: $\tilde{y} = y + z_i$
3: $\zeta_i = f(x + c_i h, \tilde{y})$
4: **end for**

4.2.2 Calcul de $(T^{-1} \otimes I)F\left(Z^k\right)$

Chaque processeur a évalué le vecteur $\zeta_{me+1} = f(y+z_{me+1})$. L'étape suivante nécessite le calcul de $T^{-1}F\left(Z^k\right)$ qui s'écrit

$$T^{-1}F\left(Z^k\right) = T^{-1}\begin{pmatrix} f(y+z_1) \\ f(y+z_2) \\ f(y+z_3) \end{pmatrix} = T^{-1}\begin{pmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{pmatrix}$$

La matrice T^{-1} est notée

$$\begin{pmatrix} ti_{11} & ti_{12} & ti_{13} \\ ti_{21} & ti_{22} & ti_{23} \\ ti_{31} & ti_{32} & ti_{33} \end{pmatrix}$$

Remarque 4.2. La matrice T est la matrice orthognale telle que $T^{-1}A^{-1}T = \Lambda$. Comme A est la matrice de la méthode d'intégration, cette dernière est constante. Ce qui signifie que par souci d'efficacité dans le code, la matrice T peut être a priori calculée puis inversée pour avoir les cœfficients ti_{ij}, $i, j = 1, \ldots, 3$.

Ainsi l'évaluation s'écrit

$$\begin{pmatrix} F_1 \\ F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} ti_{11}\zeta_1 + ti_{12}\zeta_2 + ti_{13}\zeta_3 \\ ti_{21}\zeta_1 + ti_{22}\zeta_2 + ti_{23}\zeta_3 \\ ti_{31}\zeta_1 + ti_{32}\zeta_2 + ti_{33}\zeta_3 \end{pmatrix}$$

Algorithme 2

1: $i = me + 1$
2: $f_1 = ti_{1i}\zeta_i$
3: $f_2 = ti_{2i}\zeta_i$
4: $f_3 = ti_{3i}\zeta_i$
5: MPI_Allreduce(f, F, 3*n, SUM), opération de sommation de tous à tous composante à composante.

4.2.3 Parallélisation du calcul de la jacobienne

Le calcul numérique de la jacobienne a également été distribué entre les processeurs, chacun en calculant une partie des colonnes et en les envoyant aux autres par une communication de chacun à tous.

Le programme du partenaire industriel construit la fonction du système différentiel par un assemblage de sous fonctions issues de librairies métier spécifiques, parfois uniques et confidentielles vis-à-vis du client. Dans ce cadre l'accès à la forme analytique n'est pas possible. Ceci rend donc inenvisageable l'utilisation de méthode de différentiation automatique à partir de code source [43].

C'est pourquoi dans tous les cas, la jacobienne $J = \dfrac{\partial F}{\partial y}$ est construite numériquement par la formule

$$Jv \approx \frac{F(u + \epsilon v) - F(u)}{\epsilon} \tag{4.7}$$

Pour construire entièrement la matrice J, chacune de ses colonnes est évaluée par

$$J_{|j} \approx \frac{F(u + \epsilon e_j) - F(u)}{\epsilon}$$

où e_j un le j-ième vecteur de la base canonique.

4.3 Performances de l'implémentation

Nous nous sommes limités à un problème test modélisant le circuit d'injection d'un moteur à explosion (pb 2.6). Cette modélisation conduit à devoir traiter un système d'équations différentielles algébriques de 131 inconnues. Les dynamiques de ce problème sont très différentes car les conditionnements des systèmes linéarisés varient de 10^{12} à 10^{16} sur l'intervalle de temps de la simulation.

	séquentiel	parallèle	speed up
calcul de la jacobienne		260	
communication jacobienne		28.6	
total jacobienne	847	288.6	2.93
calcul des alg. 1-2		43.5	
comm. dans alg. 1-2		44.7	
total de 1-2	143	88.2	1.62
total éxecution	1082	436	2.48

TABLE 4.1 – Comparaison entre RadauIIA séquentiel et la version parallèle "accros the method" (en secondes).

La méthode de RadauIIA d'ordre 5 comprend 3 étapes. Les tests de performances ont donc été effectués sur 3 processeurs. Les résultats montrent d'abord que le code parallélisé est presque 2.5 fois plus rapide que le code séquentiel.

Cependant, ce résultat doit être discuté. La parallélisation du code a eu lieu sur deux aspects :

1. le découpage de la construction de la matrice jacobienne ;
2. la distribution du calcul des étapes de la méthode Radau IIA.

Pour le découpage de la construction de la matrice jacobienne qui est une parallélisation à travers la fonction, on remarque dans le tableau 4.1 que le temps de construction de J est réduit (le speed-up est de 2.93 sur 3 processeurs).

Ce rapport relativement bon s'explique par le fait que les évaluations de la fonction f du modèle sont extrêmement coûteuses, ce qui relativise les poids des communications (28s contre 260s).

En ce qui concerne la parallélisation à travers la méthode, le speed-up n'est que de 1.62. Dans le détail, on constate que le temps des communications (44.7s) est aussi important que le temps nécessaire au calcul (43.5s).

Ce temps de communication ne peut être réduit par des calculs recouvrants. Avec une méthode d'ordre plus élevé, c'est à dire avec plus d'étapes ($s > 3$), le nombre de processeurs sera plus grand mais en contre partie les communications seront elles aussi plus importantes, les échanges étant plus volumineux.

Ce type de parallélisation "à travers la méthode" est limité par le nombre d'étapes de la méthode considérée et par le volume des communications bloquantes requises. C'est pourquoi les méthodes DIMSIM qui selon [25] ont un meilleur potentiel parallèle que les méthodes de RK implicites sont introduites dans le chapitre suivant, et l'analogie de leur parallélisation avec les développements de ce chapitre sont montrés.

CHAPITRE 5

Parallélisation des méthodes DIMSIM

Les méthodes DIMSIM (Diagonally-Implicit Mutlistage Integration Methods) font partie d'une classe particulière des méthodes linéaires générales.

5.1 Méthodes linéaires générales

Les méthodes linéaires générales furent introduites pour fournir une théorie commune pour les questions de convergence, stabilité et consistance entre les méthodes de RK et les méthodes linéaires multi-pas. Elles sont aussi utilisées dans le cadre d'étude de précision et des phénomènes de convergence non-linéaire ([20, 19, 53]). Ceci dépasse le cadre de ce chapitre dont le but est d'introduire ces méthodes, puis de montrer en quoi leur parallélisation est analogue à celle des méthodes de RK vue au chapitre précédent.

Une méthode linéaire générale est représentée par une matrice par bloc de taille $(s + r) \times (s + r)$:

$$\begin{bmatrix} A & U \\ B & V \end{bmatrix}$$

où A est la matrice carrée $(a_{ij})_{i,j=1,\dots,s}$ de dimension $s \times s$, B, U et V sont respectivement les matrices de dimension $r \times s$, $s \times r$ et $r \times r$ telles que $B = (b_{ij})$, $U = (u_{ij})$ et $V = (v_{ij})$.

En considérant le système différentiel

$$\begin{cases} \dot{y} = f(y) & \text{sur } [T_0, T] \\ y(T_0) = y_0 \end{cases}$$

la méthode calcule à chaque pas r valeurs $y_{j,n+1}$ $(j = 1, \dots, r)$ qui servent de valeurs d'entrée pour le pas suivant.

On note

$$y_n = \begin{pmatrix} y_{1,n} \\ \vdots \\ y_{r,n} \end{pmatrix}$$

De la même manière, les s valeurs des étapes intermédiaires sont notées

$$Y = \begin{pmatrix} Y_1 \\ \vdots \\ Y_s \end{pmatrix}$$

$$\begin{cases} Y_i = h_n \sum_{j=1}^{s} a_{ij} f(Y_j) + \sum_{j=1}^{r} u_{ij} y_{j,n} & i = 1, \dots, s \\ y_{i,n+1} = h_n \sum_{j=1}^{s} b_{ij} f\left(Y_j\right) + \sum_{j=1}^{r} v_{ij} y_{j,n} & i = 1, \dots, r \end{cases} \tag{5.1}$$

La méthode s'écrit alors

$$\begin{pmatrix} Y \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} A \otimes I & U \otimes I \\ B \otimes I & V \otimes I \end{pmatrix} \begin{pmatrix} hF(Y) \\ y_n \end{pmatrix} \tag{5.2}$$

en notant

$$F(Y) = \begin{pmatrix} f(Y_1) \\ \vdots \\ f(Y_s) \end{pmatrix}$$

Exemple 5.1. Pour la méthode RadauIIA d'ordre 5 vue précédemment dont le tableau caractéristique est (3.1)

$$\begin{array}{c|ccc} \frac{4-\sqrt{6}}{10} & \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\ \frac{4+\sqrt{6}}{10} & \frac{269+169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\ 1 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \\ \hline & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{array}$$

s'écrit sous le formalisme des méthodes linéaires générales par le tableau suivant

$$\left[\begin{array}{cccc|c} \frac{4-\sqrt{6}}{10} & \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} & 1 \\ \frac{4+\sqrt{6}}{10} & \frac{269+169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} & 1 \\ 1 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} & 1 \\ \hline 0 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} & 1 \end{array}\right]$$

Exemple 5.2 (Adams-Bashforth [20])**.** Pour la méthode d'Adams-Bashforth

$$y_n = y_{n-1} + h\left(\frac{3}{2}f(y_{n-1}) - \frac{1}{2}f(y_{n-2})\right)$$

La formulation linéaire générale s'écrit

$$\begin{array}{c|cc} 0 & 1 & \frac{3}{2} & -\frac{1}{2} \\ \hline 0 & 1 & \frac{3}{2} & -\frac{1}{2} \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}$$

Remarque 5.1. Les méthodes de Runge-Kutta ont toujours $r = 1$ puisque d'un pas à l'autre seul un y_n est utilisé. Pour les méthodes linéaires multipas (comme Adams-Bashforth), $s = 1$ parce que la fonction est évaluée uniquement une fois par pas.

5.2 Méthodes DIMSIM

Les méthodes d'intégration diagonalement implicites multi-étapes (DIMSIM) ont été introduites dans le but d'isoler une classe de méthodes linéaires générales potentiellement plus pratiques [19]. Dans le but de minimiser les coûts de l'implantation, il semble raisonnable de restreindre la matrice A à une matrice triangulaire inférieure comme dans les méthodes explicites ou les méthodes de Runge-Kutta diagonalement implicites (eq. (3.8) avec A triangulaire inférieure) :

$$A = \begin{bmatrix} \lambda & 0 & \cdots & 0 \\ \times & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \times & \cdots & \times & \lambda \end{bmatrix} \tag{5.3}$$

Dans [19], quatre sous-familles, appelées types, sont introduites :

1. Pour les problèmes non-raides pour une résolution séquentielle, A doit être triangulaire inférieure avec des 0 sur la diagonale.
2. Pour les problèmes raides pour une résolution séquentielle, A doit être triangulaire inférieure avec des valeurs non nulles sur la diagonale.
3. Pour les problèmes non-raides pour une résolution parallèle, A doit être la matrice nulle.
4. Pour les problèmes raides pour une résolution parallèle, A doit être diagonale. Dans un soucis de simplicité, on peut supposer les éléments de la diagonale égaux.

Le tableau 5.1 récapitule ces types.

Dans la suite, les méthodes de type 4 sont étudiées. Leur parallélisme potentiel réside dans le fait que la matrice A est diagonale [25].

type	λ	$a_{ij}, j < i$	Parallélisme	Classe de problème
1	0	arbitraires	non	non-raide
2	$\neq 0$	arbitraires	non	non-raide
3	0	0	oui	raide
4	$\neq 0$	0	oui	raide

TABLE 5.1 – Les 4 classes de DIMSIMs

5.3 Parallélisme potentiel des méthodes DIMSIM

Ce qui nous intéresse ici est plutôt l'efficacité possible par la mise en œuvre des DIMSIM sur une machine parallèle.

Pour cela, considérons par exemple la méthode issue de [25] dont le tableau est :

$$\left[\begin{array}{cccccc|cccccc}
\frac{2}{5} & 0 & 0 & 0 & 0 & 0 & 1 & -\frac{2}{5} & 0 & 0 & 0 & 0 \\
0 & \frac{7}{15} & 0 & 0 & 0 & 0 & 1 & -\frac{4}{15} & -\frac{11}{75} & -\frac{6}{125} & -\frac{1}{75} & -\frac{32}{9375} \\
0 & 0 & \frac{8}{15} & 0 & 0 & 0 & 1 & -\frac{2}{15} & -\frac{4}{15} & -\frac{24}{125} & -\frac{208}{1875} & -\frac{544}{9375} \\
0 & 0 & 0 & \frac{3}{5} & 0 & 0 & 1 & 0 & -\frac{9}{25} & -\frac{54}{125} & -\frac{243}{625} & -\frac{972}{3125} \\
0 & 0 & 0 & 0 & \frac{2}{3} & 0 & 1 & \frac{2}{15} & -\frac{32}{75} & -\frac{96}{125} & -\frac{1792}{1875} & -\frac{9728}{9375} \\
0 & 0 & 0 & 0 & 0 & \frac{11}{15} & 1 & \frac{4}{15} & -\frac{7}{15} & -\frac{6}{5} & -\frac{29}{15} & -\frac{8}{3} \\
\hline
0 & 0 & 0 & 0 & 0 & \frac{11}{15} & 1 & \frac{4}{15} & -\frac{7}{15} & -\frac{6}{5} & -\frac{29}{15} & -\frac{8}{3} \\
-\frac{2}{5} & \frac{35}{12} & -\frac{80}{9} & 15 & -\frac{50}{3} & \frac{1507}{180} & 0 & \frac{2}{3} & -\frac{2}{15} & -\frac{12}{5} & -\frac{92}{15} & -\frac{34}{3} \\
-\frac{25}{6} & \frac{2135}{72} & -\frac{260}{3} & \frac{535}{4} & -\frac{1925}{18} & \frac{275}{8} & 0 & 0 & \frac{1}{3} & -\frac{6}{5} & -\frac{34}{5} & -\frac{56}{3} \\
-\frac{175}{12} & \frac{7175}{72} & -\frac{2450}{9} & \frac{1475}{4} & -\frac{8875}{36} & \frac{4675}{72} & 0 & 0 & 0 & 0 & -\frac{44}{15} & -\frac{44}{3} \\
-\frac{125}{6} & \frac{9625}{72} & -\frac{1000}{3} & \frac{1625}{4} & -\frac{4375}{18} & \frac{1376}{24} & 0 & 0 & 0 & 0 & -\frac{1}{3} & -\frac{16}{3} \\
-\frac{125}{12} & \frac{4375}{72} & -\frac{1250}{9} & \frac{625}{4} & -\frac{3125}{36} & \frac{1375}{72} & 0 & 0 & 0 & 0 & 0 & -\frac{2}{3}
\end{array}\right] \tag{5.4}$$

C'est une méthode
- d'ordre 5, $p = q = 5$;
- avec 6 étapes, $r = s = 6$.

La méthode s'écrit (cf (5.2))

$$\begin{cases} Y = h(A \otimes I)F(Y) + (U \otimes I)y_n \\ y_{n+1} = h(B \otimes I)F(Y) + (V \otimes I)y_n \end{cases} \tag{5.5}$$

La seule différence avec la parallélisation vue pour RadauIIA est l'ajout des termes :
- $(U \otimes I)y_n$;

– $(V \otimes I)y_n$.

Remarque 5.2. Les méthodes de RK peuvent être écrites sous la forme des méthodes linéaires générales par la remarque 3.2 et l'équation (3.8), où U et V sont égales à I.

Comme pour Radau, posons

$$Z_i = Y_i - \sum_{j=1}^{6} u_{ij} y_{j,n} \tag{5.6}$$

Pour les méthodes de RK, la matrice A de la méthode est orthogonalisée de sorte que

$$T^{-1} A^{-1} T = \Lambda$$

Or puisque par construction des DIMSIM, A est diagonale

$$A^{-1} = diag(a_{11}^{-1}, \cdots, a_{ss}^{-1})$$

Remarque 5.3. Le possible gain en parallèle de la méthode se situe à ce niveau. Le calcul n'a pas besoin de la partie de réduction globale de l'algorithme 2 pour le calcul de $W = (T^{-1} \otimes I)Z$

Cette remarque permet d'écrire l'équation de Newton (4.6) (rappelée ci-dessous)

$$\begin{cases} (h^{-1}\Lambda \otimes I - I \otimes J)\Delta W^k = \\ \qquad\qquad - h^{-1}(\Lambda \otimes I)W^k \\ \qquad\qquad + (T^{-1} \otimes I)F\left((T \otimes I)W^k\right) \\ W^{k+1} = W^k + \Delta W^k \end{cases}$$

sous la forme suivante

$$\begin{cases} (h^{-1}A \otimes I - I \otimes J)\Delta W^k = \\ \qquad\qquad - h^{-1}(A \otimes I)W^k \\ \qquad\qquad + (I \otimes I)F\left((I \otimes I)W^k + (U \otimes I)y^m\right) \\ W^{k+1} = W^k + \Delta W^k \end{cases} \tag{5.7}$$

L'équation peut alors être résolue indépendamment sur s processeurs.

$$\begin{cases} (h^{-1}a_{ii}I - J)\Delta W_i^k = -h^{-1}a_{ii}W_i^k + F\left(W_i^k + \sum_{j=1}^{s} u_{ij} y_j^m\right) \\ W_i^{k+1} = W_i^k + \Delta W_i^k \end{cases} \tag{5.8}$$

Remarque 5.4. Lorsque les itérations de Newton sont terminés

$$W \equiv Z$$

.

L'intérêt d'une parallélisation des méthodes DIMSIM, par rapport à celles des méthodes de RK est qu'il n'est pas nécessaire d'effectuer le changement de base (algorithme 2), A étant dès le départ diagonale.

Cette suppression a l'avantage d'éviter une communication de tous à tous à chaque itération du Newton. En conséquence le processus de Newton est découplé.

Bien que potentiellement attrayantes, à notre connaissance, aucune implantation parallèle de ces méthodes n'existe à ce jour. Nous n'avons pas développé de code car comme dans le cas de RadauIIA5, le nombre de processeurs reste limité par le nombre d'étapes s de la méthode.

5.4 Conclusions

La parallélisation des DIMSIM est identique à celle proposée pour les méthodes de RK. La matrice A étant diagonale par construction, les calculs des étapes de la méthode sont découplés donc parallèles. Par rapport à une méthode de RK, une seule communication globale de tous à tous est requise à chaque pas. Cependant le nombre de processeurs reste limité par le nombre s d'étapes de la méthode. En pratique ce nombre est réduit, par exemple dans le code largement utilisé vode/cvode, le nombre d'étapes peut aller jusqu'à douze, mais par la présence de discontinuités, le critère de sélection de l'ordre du schéma conduit le plus souvent à utiliser les méthodes de 3 à 5 étapes.

Conclusions sur la parallélisation à travers la méthode

La parallélisation à travers la méthode de RadauIIA d'ordre 5 a montré un speed-up de 1.7 sur trois processeurs. Le speed-up monte à presque 2.5 lorsque la construction de la matrice jacobienne est conjointement parallélisée. L'implantation a également permis de mettre en évidence que ce genre de technique demande un volume important de communications. Ce qui peut représenter un speed-down lorsque le nombre de processeurs devient plus important.

Nous avons ensuite montré que la parallélisation à travers les méthodes de RK et DIMSIM est identique. Celle-ci repose sur la distribution des évaluations indépendantes de chaque étape de la méthode. Les communications bloquantes à chaque itération du Newton dans les méthodes de RK disparaissent dans les méthodes DIMSIM qui ne conservent qu'une seule communication bloquante après convergence des itérations de Newton sur chaque étapes. Elles sont potentiellement plus efficaces mais le nombre de processeurs reste limité par le nombre d'étapes s de la méthode.

La parallélisation à travers la méthode reste donc d'un intérêt limité pour la résolution de problèmes complexes avec un grand nombre d'inconnues. Car le nombre de processeurs sur lesquels la méthode est parallélisée correspond à son nombre d'étapes.

De plus les problèmes peuvent présenter de grandes variations dans leur complexité/-raideur au cours du temps. Un exemple de ce type de difficulté est la simulation d'un système hydraulique comportant des vannes ouvertes ou fermées en fonction de divers capteurs. Dans l'optique de résoudre de tels problèmes et lorsque le nombre de processeurs disponibles est nettement supérieur au nombre d'étapes de la méthode considérée, d'autres méthodologies de parallélisation sont à envisager. Nous allons présenter dans la partie suivante les méthodes de décomposition en temps. Ces méthodes s'inspirent du découpage de l'intervalle d'intégration en morceaux.

Deuxième partie

Parallélisation en temps

Introduction sur la décomposition en temps

Les méthodes de décomposition en temps consistent à résoudre de manière la plus indépendante possible le problème sur des sous-intervalles de l'intervalle de temps. Les méthodes mathématiques de décomposition en temps ont pour objet la mise à jour des conditions initiales de la solution sur chaque sous-intervalle de temps.

La difficulté est majeure car les méthodes d'intégration des systèmes différentiels ordinaires et algébriques sont par nature séquentielles. En effet, la ou les solutions des pas de temps précédents sont nécessaires pour l'intégration de la solution au pas de temps en cours.

Nous commencerons par introduire les méthodes basées sur une parallélisation à travers les pas. L'idée est de mettre à jour l'ensemble des pas de temps par une méthode de résolution non linéaire plutôt que d'avancer pas à pas la solution en devant résoudre exactement le problème non linéaire à chaque pas de temps. Cependant, elles nécessitent la connaissance a priori des différents pas de temps que va utiliser la méthode d'intégration.

Ensuite, nous présentons les méthodes de tirs. Dans ces dernières une première étape consiste à résoudre de manière indépendante la solution sur chaque sous intervalle, puis une deuxième étape corrige l'erreur commise entre les conditions initiales et les solutions à la fin du sous intervalle de temps précédent. Les algorithmes Parareal et Pita rentrent dans le cadre de ces méthodes et diffèrent par le traitement de l'étape de correction. Notre apport ici est de définir les notions de finesse de grille en temps pour adapter ces méthodes aux cas des équations différentielles raides.

Enfin, nous introduisons les estimateurs d'erreurs qui permettent de construire des schémas itératifs d'ordre élevé comme la deferred correction method. Nous avons vu le potentiel de cette méthode pour l'appliquer à la décomposition en temps. Nous obtenons alors une méthode à deux niveaux de parallélisme, l'un sur la décomposition en sous intervalles de temps, l'autre sur la mise en cascade des itérations.

CHAPITRE 6

Parallélisation à travers les pas

Dans cette partie, nous introduisons une parallélisation tirant partie de la connaissance a priori des pas de temps de la méthode d'intégration utilisée. La connaissance a priori du pas h permet d'écrire un système non-linéaire vérifié par la solution sur un intervalle $[t_n, t_{n+1}]$. Nous présentons l'approche de Bellen et al. dans [4, 6]. Communément cette approche résout le système non-linéaire séquentiellement par bloc à cause de la structure particulière du problème. Nous étendons cette approche par l'introduction d'un solveur global parallèle pour ce problème linéaire : soit une méthode de point fixe, soit un Newton. Nous allons montrer que nous pouvons introduire des techniques classiques de partitionnement de données pour résoudre en parallèle le système non-linéaire intervenant dans les techniques de parallélisation à travers les pas.

6.1 Équation aux différences

Le problème différentiel (1.1)

$$\begin{cases} \dot{y} = f(t, y) \text{ sur } [T_0, T] \\ y(T_0) = y_0^{init} \end{cases} \tag{1.1}$$

est résolu sur l'intervalle $[T_0, T]$ par une méthode d'intégration dont la séquence de pas de temps successifs h_i est connue a priori, telle que

$$\sum_{i=0}^{N-1} h_i = T - T_0$$

Cette séquence de pas de temps définit une partition de l'intervalle d'intégration

$$T_0 = t_0 < \cdots < t_i = t_0 + \sum_{j=0}^{i-1} h_j < \cdots < t_N = T \tag{6.1}$$

Remarque 6.1. Dans le cas de pas de temps constant, c'est à dire lorsque $\forall i, h_i = h$, on a

$$h = \frac{T - T_0}{N}$$
$$t_i = t_0 + ih$$

Définition 6.1. Le problème défini sur l'intervalle $[t_k, t_{k+1}]$ par

$$\begin{cases} \dfrac{dz}{dt} = f(t, z) \\ z(t_k) = y_k \end{cases} \tag{6.2}$$

admet comme solution $z(t; t_k, y_k)$.

Cette solution est aussi appelée *trajectoire*, ou flot.

Notons $F_{k+1}(y_k)$ l'approximation numérique de $z(t_{k+1}; t_k, y_k)$.

Ainsi sur la grille $t_0 < \cdots < t_i < \cdots < t_N$, n'importe quelle méthode d'intégration à un pas peut être vue comme une équation de récurrence d'ordre un (6.3) (cf [6])

$$y_{m+1} = F_{m+1}(y_m) \qquad\qquad m = 0, 1, \ldots, N-1 \tag{6.3}$$

Exemple 6.1 (F_{m+1} pour une méthode de RK implicite [6])**.** Rappelons la méthode de RK implicite à s étapes

$$\begin{cases} k_i = f\left(x_0 + c_i h, y_0 + \displaystyle\sum_{j=1}^{s} a_{ij} k_j\right) \\ y_{m+1} = y_m + h_m \displaystyle\sum_{i=1}^{s} b_i k_i \end{cases} \tag{3.5}$$

Cette méthode peut être utilisée pour obtenir la séquence y_{m+1} comme une approximation de la solution $y(t)$ aux instants t_{m+1}, $m = 0, \ldots, N-1$. Il est clair que $F_{m+1}(y_0) = y_0 + h_m \displaystyle\sum_{i=1}^{s} b_i k_i$ issue de la méthode de RK (3.5) définit l'ensemble de fonctions $\{F_{m+1}; m = 1, \ldots, N-1\}$. Les fonctions F_{m+1} dépendent de la fonction f du système.

Les N équations (6.3) peuvent être écrites sous forme d'un système :

$$\begin{cases} y_0 = y_0^{init} \\ y_1 = F_1(y_0) \\ y_2 = F_2(y_1) \\ \vdots \\ y_N = F_N(y_{N-1}) \end{cases}$$

Les valeurs $y_0, \ldots, y_N$ étant les inconnues du système non-linéaire, ce dernier peut être réécrit sous forme vectorielle, en notant $Y = (y_0, \ldots, y_N)^t$ le vecteur colonne des inconnues

$$G(Y) = \begin{pmatrix} y_0 - y_0^{init} \\ y_1 - F_1(y_0) \\ y_2 - F_2(y_1) \\ \vdots \\ y_N - F_N(y_{N-1}) \end{pmatrix} = 0 \tag{6.4}$$

Le système non-linéaire étant posé, une méthode de résolution parallèle peut être utilisée.

6.2 Parallélisation d'équations aux différences

La parallélisation d'un système non-linéaire fait intervenir des techniques différentes suivant la méthode de résolution du système non-linéaire. Une méthode de point fixe ou une méthode de Newton n'utiliseront pas les mêmes techniques de parallélisation. Nous allons décrire la parallélisation envisageable pour ces deux approches.

6.2.1 Méthode du point fixe

Pour une méthode de point fixe, le système non-linéaire est transformé en un processus itératif explicite. Le système non-linéaire (6.4) de dimension $(N+1) \times d$ est écrit sous la forme

$$Y = \mathbb{F}(Y) \tag{6.5}$$

où d est la taille du système différentiel (1.1) et

$$\mathbb{F}(Y) = \begin{pmatrix} y_0^{init} \\ F_1(y_0) \\ F_2(y_1) \\ \vdots \\ F_N(y_{N-1}) \end{pmatrix}$$

La nature explicite de cette méthode (algorithme 3) est attrayante car elle permet une parallélisation de type partitionnement de données.

Algorithme 3 Méthode du point fixe

1: définition d'un vecteur solution initial $Y^{(0)}$.
2: **repeat**
3: calcul de $Y^{(i+1)} = F\left(Y^{(i)}\right)$.
4: **until** $||Y^{(i+1)} - Y^{(i)}|| < \epsilon$, $i = 1, \ldots$

Les inconnues y_i peuvent être distribuées par un découpage en tranches [44]. Chaque processeur n'a plus qu'à gérer la tranche d'inconnues qui lui est attribuée. Notons np

le nombre de processeurs disponibles et $y_{me\times ns+k} = y_{me,k}$ pour $me = 0, \ldots, np-1$ et $k = 1, \ldots, ns$, on obtient $ns = \dfrac{N}{np}$ solutions intermédiaires gérées par chaque processeur :

$$\text{processeur } 0 \begin{cases} y_{0,1}^{(i+1)} = F_{0,1}(y_{0,1}^{(i)}) \\ \vdots \\ y_{0,ns}^{(i+1)} = F_{0,ns}(y_{0,ns-1}^{(i)}) \end{cases}$$
$$y_{1,0}^{(i)} \equiv y_{0,ns}^{(i)}$$

$$\text{processeur } 1 \begin{cases} y_{1,1}^{(i+1)} = F_{1,1}(y_{1,0}^{(i)}) \\ \vdots \\ y_{1,ns}^{(i+1)} = F_{1,ns}(y_{1,ns-1}^{(i)}) \end{cases}$$
$$\vdots$$
$$\vdots$$
$$y_{np-1,0}^{(i)} \equiv y_{np-2,ns}^{(i)}$$
$$\text{processeur } np-1 \begin{cases} y_{np-1,1}^{(i+1)} = F_{np-1,1}(y_{np-1,0}^{(i)}) \\ \vdots \\ y_{np-1,ns}^{(i+1)} = F_{np-1,ns}(y_{np-1,ns-1}^{(i)}) \end{cases}$$

Ce schéma peut être vu comme un Jacobi en temps. Le schéma des dépendances de données est alors simple :

– à l'itération $i+1$, le processeur $me+1$ a besoin de la solution $y_{me*\frac{N}{np}-1}^{(i)}$ du processeur me.

Des communications asynchrones peuvent être effectuées afin de tirer parti des calculs pour recouvrir les échanges de données entre les processeurs dès lors que $ns > 2$.

Algorithme 4 Across the Step par un point fixe (par processeur)

1: définition d'un vecteur solution initial $Y_{me}^{(0)} = \begin{pmatrix} y_{me,0} \\ y_{me,1} \\ \vdots \\ y_{me,ns} \end{pmatrix}$.
2: **repeat**
3: Réception non-bloquante de $y_{me,0}^{(i)}$.
4: Envoi non-bloquant de $y_{me,ns}^{(i)}$.
5: Calcul de $Y_{me,j}^{(i+1)} = \mathbb{F}\left(Y^{(i)}\right)_{me,j}$ pour $j = 2, \ldots, ns-1$ tel que $me \times ns + j > i$.
6: Attente de la réception
7: Calcul de $Y_{me,1}^{(i+1)} = \mathbb{F}\left(Y^{(i)}\right)_{me,1}$.
8: **until** $||Y^{(i+1)} - Y^{(i)}|| < \epsilon$, $i = 1, \ldots$

À chaque appel à la fonction $\mathbb{F}$, la solution numérique progresse d'un pas de temps. Ainsi la méthode aura convergé si N itérations du point fixe sont effectuées. Afin d'avoir un gain, il faut donc que l'algorithme converge en $n_{it} < N$ itérations. C'est pourquoi des techniques d'accélération de la convergence, comme celle de Steffensen, sont introduites dans la suite.

6.2.2 Accélération de Steffensen

La méthode de Steffensen est une méthode d'accélération basée sur la méthode Δ^2 d'Aitken.

La méthode Δ^2 d'Aitken est la méthode la plus simple pour accélérer la convergence des séries convergentes $y^{(i)}$ telles que

$$\lim_{i \to \infty} y^{(i)} = \xi \tag{6.6}$$

Ces deux méthodes transforment la suite $\{y^{(i)}\}$ en une suite $\{y^{(i)'}\}$ qui, en général, converge plus vite vers la limite ξ.

Rappelons la méthode Δ^2 ([10, 95]). Supposons que la suite $y^{(i)}$ converge géométriquement vers ξ, d'un facteur k tel que $|k| < 1$

$$y^{(i+1)} - \xi = k(y^{(i)} - \xi) \tag{6.7}$$

Les quantités k et ξ peuvent être déterminées avec trois valeurs successives de la série, $y^{(i)}, y^{(i+1)}$ et $y^{(i+2)}$.

$$y^{(i+1)} - \xi = k(y^{(i)} - \xi) \qquad y^{(i+2)} - \xi = k(y^{(i+1)} - \xi) \tag{6.8}$$

Ainsi après substitution et comme $k \neq 1$:

$$k = \frac{y^{(i)}y^{(i+2)} - {y^{(i+1)}}^2}{y^{(i+2)} - 2y^{(i+1)} + y^{(i)}} \tag{6.9}$$

$$\xi = \frac{y^{(i+1)} - ky^{(i)}}{1-k} = \frac{y^{(i)}y^{(i+2)} - {y^{(i+1)}}^2}{y^{(i+2)} - 2y^{(i+1)} + y^{(i)}} \tag{6.10}$$

En utilisant l'opérateur de différence $\Delta y^{(i)} = y^{(i+1)} - y^{(i)}$ et $\Delta^2 y^{(i)} = \Delta y^{(i+1)} - \Delta y^{(i)} = y^{(i+2)} - 2y^{(i+1)} + y^{(i)}$

$$\xi = y^{(i)} - \frac{\left(\Delta y^{(i)}\right)^2}{\Delta^2 y^{(i)}} \tag{6.11}$$

D'où le nom de la méthode.

La méthode se résume alors à construire la suite $\{y^{(i)'}\}$ définie par

$$y^{(i)'} = y^{(i)} - \frac{(\Delta y^{(i)})^2}{\Delta^2 y^{(i)}} \tag{6.12}$$

Théorème 6.1 ([95])**.** *Supposons qu'il existe k, $|k| < 1$ tel que pour la suite $\{y^{(i)}\}$, $y^{(i)} \neq \xi$*

$$y^{(i+1)} - \xi = (k + \delta^{(i)})(y^{(i)} - \xi) \qquad \lim_{i\to\infty} \delta^{(i)} = 0 \tag{6.13}$$

alors les termes $y^{(i)'}$ définis par (6.12) *existent tous à partir de i suffisamment grand et*

$$\lim_{i\to\infty} \frac{y^{(i)'} - \xi}{y^{(i)} - \xi} = 0 \tag{6.14}$$

Considérons maintenant que la suite $\{y^{(i)}\}$ provienne d'une méthode itérative $F(x)$ telle que

$$y^{(i+1)} = F(y^{(i)})$$

pour déterminer le zéro ξ d'une fonction $G(x) = x - F(x) = 0$. À partir de trois valeurs itérées successives, la suite $\{y^{(i)'}\}$ peut être construite par (6.12) :

$$\begin{aligned} w^{(i)} &= F(y^{(i)}), \quad z^{(i)} = F(w^{(i)}), \quad i = 0, 1, \ldots \\ y^{(i+1)} &= y^{(i)} - \frac{\left(w^{(i)} - y^{(i)}\right)^2}{z^{(i)} - 2w^{(i)} + y^{(i)}} \end{aligned} \tag{6.15}$$

Cette méthode est due à Steffensen.

L'équation (6.15) peut s'écrire sous forme itérative ψ

$$\begin{aligned} y^{(i+1)} &= \psi(y^{(i)}) \\ \psi(x) &= \frac{xF(F(x)) - F(x)^2}{F(F(x)) - 2F(x) + x} \end{aligned} \tag{6.16}$$

Théorème 6.2 ([95])**.**

- *$\psi(\xi) = \xi$ implique $F(\xi) = \xi$.*
- *Réciproquement, si $F(\xi) = \xi$ et $F'(\xi) \neq 1$ alors $\psi(\xi) = \xi$.*

Ce théorème assure qu'en général les fonctions F et ψ convergent vers les mêmes points fixes.

Théorème 6.3 ([95] th. 5.10.13)**.** *Soit F une fonction définissant une méthode d'ordre p pour calculer son point fixe ξ.*

- *Pour $p > 1$ la fonction ψ définie par* (6.16) *est une méthode d'ordre $2p - 1$ pour estimer ξ.*
- *Pour $p = 1$ la méthode est au moins d'ordre 2 si $F'(\xi) \neq 1$.*

Cas vectoriel

Dans [57, 73, 74], une généralisation de la méthode Δ^2 au cas multidimensionnel est introduite.

Matriciellement les équations (6.8) s'écrivent

$$y^{(i+1)} - \xi = P(y^{(i)} - \xi) \qquad y^{(i+2)} - \xi = P(y^{(i+1)} - \xi) \tag{6.17}$$

et l'équation (6.12) devient

$$y^{(i)\prime} = y^{(i)} - P(y^{(i+1)} - y^{(i)}) \tag{6.18}$$

Soit la formule suivante, dite d'Aitken-Steffensen

$$y^{(i)\prime} = y^{(i)} - \Delta Y^{(i)}(\Delta^2 Y^{(i)})^{-1}\Delta y^{(i)} \tag{6.19}$$

avec

$$\Delta y^{(i)} = F(y^{(i)}) - y^{(i)} \tag{6.20}$$

$$\Delta Y^{(i)} = \left(F(y^{(i)}) - y^{(i)}, F(F(y^{(i)})) - F(y^{(i)}), \ldots, F^{(n)}(y^{(i)}) - F^{(n-1)}(y^{(i)})\right) \tag{6.21}$$

$$\Delta^2 Y^{(i)} = \left(F^{(2)}(y^{(i)}) - 2F^{(1)}(y^{(i)}) + y^{(i)}, \ldots, F^{(n+1)}(y^{(i)}) - 2F^{(n)}(y^{(i)}) + F^{(n-1)}(y^{(i)})\right) \tag{6.22}$$

$$P = \Delta F = \Delta Y^{(i)}(\Delta^2 X)^{-1} \tag{6.23}$$

en notant

$$F^{(0)}(x) = x \qquad F^{(j)}(x) = F\left(F^{(j-1)}(x)\right)$$

Remarque 6.2. L'expression (6.21) nécessite pour le calcul de terme $F^{(j)}(y^{(i)}) = y^{(i+j)}$ d évaluations séquentielles de la fonction F.

La construction de la matrice ΔX présente deux inconvénients :

1. Les évaluations de la fonction F doivent s'effectuer séquentiellement, ce qui brise le parallélisme.
2. La construction de toutes les colonnes demande autant d'itérations de fonction F.

Remarque 6.3. L'équation (6.18) est analogue à la méthode de Newton. C'est pourquoi, dans [5, 6], la matrice P est considérée comme une approximation de la matrice jacobienne $DF(y^{(i)})$. Ils proposent le calcul suivant pour le cœfficient de la ligne k et colonne l de ΔF.

$$(\Delta F)_{kl} = \begin{cases} \dfrac{\left(w^{(i)} + (z^{(i)} - w^{(i)})e_k - w^{(i)}\right)_l}{\left(w^{(i)} - y^{(i)}\right)_k} & \text{si } \left(w^{(i)} - y^{(i)}\right)_k \neq 0 \\ \dfrac{\partial F}{\partial w_k}\left(y^{(i)}\right)_l \end{cases} \tag{6.24}$$

où e_k est le $k$ième vecteur de la base canonique, $e_k = (0, \ldots, 0, 1, 0, \ldots, 0)^T$ et $k, l = 1, \ldots, Nn$

$$\frac{\left(w^{(i)} + (z^{(i)} - w^{(i)})e_k - w^{(i)}\right)_l}{(w^{(i)} - y^{(i)})_k} = \begin{cases} G_k & \text{si } k = l-1 \\ 0 & \text{sinon.} \end{cases}$$

et lorsque $\left(w^{(i)} - y^{(i)}\right)_k \neq 0$

$$\frac{\partial F}{\partial w_k}\left(y^{(i)}\right) = \begin{cases} G_k & \text{si } k \bmod n = l-1 \bmod n \\ 0 & \text{sinon.} \end{cases}$$

La matrice ΔF est donc une matrice par bloc, ne contenant que la première diagonale inférieure.

$$\Delta F = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ G_1 & 0 & & & \vdots \\ 0 & G_2 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & G_{N-1} & 0 \end{pmatrix} \tag{6.25}$$

Parallélisation proposée par Bellen et al.

Bellen et al ([5, 6],) introduisent l'algorithme suivant 5 :

Algorithme 5 Parallélisation à travers les pas

1: **repeat**
2: calcul en parallèle de $w^{(i)} = F(y^{(i)})$.
3: calcul en parallèle de $z^{(i)} = F(z^{(i)})$.
4: construction parallèle de ΔF.
5: calcul séquentiel pour chaque $n = 0, \ldots, N-1$

$$y_{n+1}^{(i+1)} = w_{n+1}^{(i)} + G_j(y_n^{(i+1)} - y_n^{(i)})$$

6: **until** $||y^{(i+1)} - y^{(i)}|| < \epsilon$, $i = 0, \ldots$

Les auteurs obtiennent un speed-up de 11 sur 200 processeurs. Ce rapport relativement faible s'explique par la phase 5 pour laquelle la majorité du temps est perdue.

Le speed-up S est donné par

$$S = \frac{T_s + T_1}{T_s + T_p}$$

où T_s est le temps de la partie séquentielle de l'algorithme, T_p (resp T_1) le temps de la partie parallèle sur p processeurs (resp. sur 1 proc.).

Lorsque l'on réduit le temps parallèle $T_p \to 0$, le speed-up reste borné par $\frac{T_s + T_1}{T_s} = 1 + \frac{T_1}{T_s}$.

D'où l'intérêt de réduire le temps séquentiel T_s le plus possible, le mieux étant de supprimer les parties séquentielles des algorithmes. L'expression explicite de $y^{(i)'}$ peut être évaluée par un partitionnement par bloc des lignes de la matrice ΔF. Tout comme dans l'algorithme 4.

Nous développons dans la suite la méthode de Newton pour la résolution de l'équation (6.4).

6.2.3 Méthode de Newton

Pour la méthode de Newton, le système est linéarisé et la résolution s'effectue sur un incrément de la solution, δY.

$$\begin{cases} \left(\dfrac{\partial G}{\partial Y}(Y^{(i+1)})\right) \delta Y^{(i)} = -G(y^{(i)}) \\ Y^{(i+1)} = Y^{(i)} + \delta Y^{(i)} \end{cases} \tag{6.26}$$

Le système linéaire (6.26) peut ensuite être résolu par des méthodes de partitionnement de données dans les méthodes de Krylov parallèles ou bien par des méthodes de décomposition de domaines, comme par exemple la méthode Schwarz ou de Schur (qui sera traitée dans la partie III).

Considérons la méthode de Newton suivante (algorithme 6).

Algorithme 6 Méthode de Newton pour un schéma à pas constant

1: définition d'un vecteur solution initial $Y^{(0)}$.
2: **repeat**
3: calcul de $\delta Y^{(i)}$ par la résolution du système linéaire

$$\left(\frac{\partial G}{\partial Y}\left(Y^{(i+1)}\right)\right) \delta Y^{(i)} = -G(y^{(i)})$$

4: mise à jour de la solution $Y^{(i+1)} = Y^{(i)} + \delta Y^{(i)}$.
5: **until** $||\delta Y^{(i)}|| < \epsilon$

La matrice jacobienne $J = \frac{\partial G}{\partial Y}\left(Y^{(i+1)}\right)$ dans le système linéarisé de Newton s'écrit sous la forme suivante

$$J = \frac{\partial G}{\partial Y}\left(Y^{(i+1)}\right) = \begin{pmatrix} I & & & & \\ -G_0 & I & & & \\ & -G_1 & I & & \\ & & \ddots & \ddots & \\ & & & -G_{N-1} & I \end{pmatrix} \tag{6.27}$$

en notant $G_i = \frac{\partial F_i}{\partial y}(y_i)$, pour $i = 0, \ldots, N-1$.

Remarque 6.4. Cette structure est semblable à celle des méthodes de tirs qui vont être décrites au chapitre 7.

Les équations (6.26) et (6.27) nécessitent la résolution d'un système linéaire avec une matrice triangulaire inférieure par bloc.

Considérons l'intégration d'un problème linéaire

$$\begin{cases} \dot{y}(t) = Ay(t) \\ y(t_0) = y_0^{init} \end{cases}$$

par une méthode d'Euler implicite de pas constant h, on a

$$F_0(y) = F_1(y) = \cdots = F_{N-1}(y) = (I - hA)^{-1} y$$

ainsi

$$\begin{aligned} y_{n+1} - (I - hA)^{-1} y_n &= 0, && \forall n = 0, \ldots, N-1 \\ G_0 = G_1 = \cdots = G_{N-1} &= (I - hA)^{-1} \end{aligned}$$

La résolution du système linéaire intervenant dans l'étape 3 par une méthode directe conduit à

$$J^{-1}\delta Y^{(i)} = \begin{pmatrix} I & & & & \\ (I-hA)^{-1} & I & & & \\ (I-hA)^{-2} & (I-hA)^{-1} & I & & \\ & & \ddots & \ddots & \\ (I-hA)^{-N} & (I-hA)^{-(N-1)} & \cdots & (I-hA)^{-1} & I \end{pmatrix} \left(Y^{(i+1)} - Y^{(i)}\right) = Y^{(i)} - \begin{pmatrix} y_0^{init} \\ (I-hA)^{-1} y_0 \\ (I-hA)^{-1} y_1 \\ \vdots \\ (I-hA)^{-1} y_{N-1} \end{pmatrix}$$

Ce qui simplifie à

$$\begin{cases} y_0 = y_0^{init} \\ y_1 = (I - hA) y_0 \\ \vdots \\ y_N = (I - hA)^N y_0 \end{cases}$$

La résolution par une méthode directe revient alors à une résolution séquentielle.

Si on envisage de résoudre ce système par une méthode de Krylov, un soin particulier doit être apporter. Une méthode de Krylov est une méthode itérative s'appuyant sur la construction d'une base orthogonale pour la norme associée à la matrice. La matrice J n'étant pas symétrique, la méthode de Krylov doit être adaptée. Le Résidu Conjugué Généralisé peut, par exemple, être utilisé (algorithme 10 au chapitre suivant).

La parallélisation des méthodes de Krylov est aisée, car ces méthodes s'appuient sur la norme associée à la matrice J qui, dans le cas du RCG est évaluée par le produit scalaire $< J., J. >$. Les dépendances de données intervenant dans le calcul du produit de la matrice J par un vecteur sont les mêmes que précédemment.

L'algorithme 7 est la version parallèle pour le produit matrice-vecteur.

Algorithme 7 Produit matrice-vecteur $Jx = y$ (méthodes à travers les pas)

1: Réception non-bloquante de x_0.
2: Envoi non-bloquant de x_{ns}.
3: Calcul de $y_j = x_j - G_{j-1}x_{j-1}$, $j = 2, \ldots, ns$.
4: Attente de la réception
5: Calcul de $y_1 = x_1 - G_0 x_0$.

6.3 Conclusion

Les parallélisations à travers les pas de type point fixe et Newton sont intéressantes et aisées. De bonnes performances sont possibles, sans aucune limitation d'ordre théorique sur le nombre de processeurs. Dans les algorithmes 4 et 7, le problème est divisé par np et les communications sont recouvertes par le calcul. Ainsi sur np processeurs la résolution demandant n_{it} itérations parallèles, est $\dfrac{np}{n_{it}}$ fois plus rapide que l'algorithme séquentiel.

Remarque 6.5. Il n'est pas évident que la résolution parallèle de (6.4) soit plus rapide que l'évaluation successive sur un processeur des N équations (6.3). Néanmoins on peut supposer que pour N très grand, et np processeurs bien choisi, la résolution parallèle soit plus efficace. Le couple (N, np) va donc dépendre de

- La taille d du problème.
- La complexité de la fonction d'incrément $\mathbb{F}$.
- Le temps d'évaluation de la fonction f (parfois long à cause des multiples tests)

Cependant dans l'optique de résoudre des problèmes raides, cette approche présuppose la valeur des pas de temps successifs au cours de l'intégration.

C'est pourquoi les méthodes de tirs vont être présentées dans le chapitre suivant. Elles vont permettre d'utiliser deux grilles pour adapter au mieux le pas de temps au cours de l'intégration du problème.

CHAPITRE 7

Méthodes de tirs

7.1 Méthode de tirs multiples

La méthode de tirs multiples est une méthode de résolution de problème aux valeurs aux limites (*Boundary-Value Problem*). Ce problème peut se mettre sous la forme

$$\begin{cases} \dot{y} = f(t, y) \\ r(y(T_0), y(T)) = 0 \end{cases} \tag{7.1}$$

avec f une fonction d'évolution du problème de classe $\mathcal{C}^1$ sur $[T_0, T] \times \mathbb{R}^d$ à valeur dans $\mathbb{R}^d$ et r une fonction représentant les conditions aux limites.

Remarque 7.1. Les conditions aux limites, $r(y(T_0), y(T))$, s'écrivent le plus souvent sous la forme matricielle suivante

$$Ay(T_0) + By(T) = c \tag{7.2}$$

A et B sont des matrices carrées d'ordre d, et c un vecteur de $\mathbb{R}^d$. Généralement les conditions sont séparées, on a

$$\begin{pmatrix} A_1 \\ 0 \end{pmatrix} y(T_0) + \begin{pmatrix} 0 \\ A_2 \end{pmatrix} y(T) = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \tag{7.3}$$

Remarque 7.2. De plus, pour un problème à valeur initiale, la condition limite correspond à la valeur initiale.

$$Iy(T_0) + 0y(T) = y_0 \tag{7.4}$$

7.1.1 Cas général

Considérons la décomposition en temps (6.1) [95], $T_0 = t_0 < t_1 < \cdots < t_N = T$. On note λ_k les valeurs $\lambda_k = y(t_k)$ pour $k = 0, \ldots, N$ de la solution exacte $y(t)$ du problème (7.1).

Si les valeurs λ_k sont connues, alors les N problèmes de Cauchy définis sur les intervalles $[t_k, t_{k+1}]$ peuvent être résolus en parallèle.

$$\begin{cases} \dot{y} = f(t, y) \\ y(t_k) = \lambda_k \end{cases} \tag{7.5}$$

La solution de ces problèmes sera notée $y(t_{k+1}; t_k, \lambda_k)$, aussi appelée *trajectoire* (cf définition 6.1).

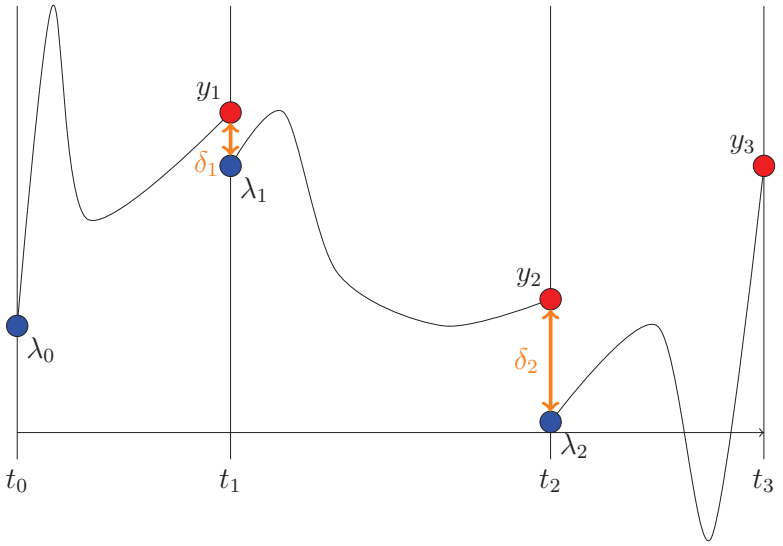


FIGURE 7.1 – Aperçu de la méthode de tirs multiples

Malheureusement les valeurs λ_k ne sont pas connues explicitement, leur détermination précise demanderait la résolution du problème originel.

Le problème revient alors à déterminer les vecteurs λ_k tels que la solution y définie par morceaux soit continue et vérifie la condition limite $r(y(t_0), y(t_N)) = 0$. Ainsi les $N + 1$ conditions suivantes doivent être vérifiées

$$\begin{cases} y(t_{k+1}; t_k, \lambda_k) = \lambda_{k+1} \text{ pour } k = 0, \dots, N-1 \\ r(\lambda_0, \lambda_N) = 0 \end{cases} \tag{7.6}$$

Ceci traduit le fait que pour chaque temps t_{k+1}, on cherche à annuler les éventuels sauts commis entre les approximations λ_{k+1} et les trajectoires $y(t_{k+1}; t_k, \lambda_k)$. Quant à la dernière condition, elle assure le respect des conditions aux limites.

Par conséquent $(N + 1)$ équations de dimension d sont imposées. On peut écrire ces conditions sous forme d'un système d'équations non-linéaires

$$F(\Lambda) = \begin{bmatrix} F_N(\lambda_0, \lambda_N) \\ F_0(\lambda_0, \lambda_1) \\ F_1(\lambda_1, \lambda_2) \\ \vdots \\ F_{N-1}(\lambda_{N-1}, \lambda_N) \end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix} r(\lambda_0, \lambda_N) \\ y(t_1; t_0, \lambda_0) - \lambda_1 \\ y(t_2; t_1, \lambda_1) - \lambda_2 \\ \vdots \\ y(t_N; t_{N-1}, \lambda_{N-1}) - \lambda_N \end{bmatrix} = 0 \tag{7.7}$$

Dont les inconnues sont

$$\Lambda = \begin{bmatrix} \lambda_0 \\ \vdots \\ \lambda_N \end{bmatrix} \tag{7.8}$$

La résolution de ce système non-linéaire s'effectue le plus souvent itérativement avec la méthode de Newton

$$\Lambda^{(i+1)} = \Lambda^{(i)} - J(\Lambda^{(i)})^{-1} F(\Lambda^{(i)}) \tag{7.9}$$

où $J(\Lambda^{(i)})$ est la matrice jacobienne de F.

À chaque pas, $F(\Lambda)$ et $J(\Lambda)$ doivent être calculées pour $\Lambda = \Lambda^i$. Pour l'évaluation de $F(\Lambda)$ (7.7), les valeurs des trajectoires $y(t_{k+1}; t_k, \lambda_k)$ sont à déterminer en résolvant les N problèmes de Cauchy (7.5).

La matrice Jacobienne J de ce problème possède une structure particulière

$$J(\Lambda) = \begin{pmatrix} A & 0 & & 0 & B \\ G_0 & -I & 0 & & 0 \\ 0 & G_1 & -I & \ddots & \\ & \ddots & \ddots & \ddots & 0 \\ 0 & & \ddots & G_{N-1} & -I \end{pmatrix} \tag{7.10}$$

Les matrices A, B et G_k, $k = 0, \ldots, N-1$ sont des matrices de taille $d \times d$

$$G_k = \partial_{\lambda_k} F_k(\Lambda) = \partial_{\lambda_k} y(t_{k+1}; t_k, \lambda_k) \tag{7.11a}$$
$$B = \partial_{\lambda_N} F_N(\Lambda) = \partial_{\lambda_N} r(\lambda_0, \lambda_N) \tag{7.11b}$$
$$A = \partial_{\lambda_0} F_N(\Lambda) = \partial_{\lambda_0} r(\lambda_0, \lambda_N) \tag{7.11c}$$

Dans la pratique, pour évaluer ces matrices, les termes différentiels sont remplacés par une formule aux différences — utilisation des formules de Taylor pour la différentiation numérique —.

$$\partial_{\lambda_k} F_k(\Lambda) v \approx \frac{y(t_{k+1}; t_k, \lambda_k + \epsilon v) - y(t_{k+1}; t_k, \lambda_k)}{\epsilon}$$

Ainsi pour construire la colonne j de la matrice G_k, on doit évaluer

$$\partial_{\lambda_k} F_k(\Lambda)|_j \approx \frac{y(t_{k+1}; t_k, \lambda_k + \epsilon e_j) - y(t_{k+1}; t_k, \lambda_k)}{\epsilon}$$

avec e_j le $j^{\text{ième}}$ vecteur canonique. Ce qui signifie que pour construire chaque matrice G_k, d problèmes de Cauchy doivent être résolus, donc $N \times d$ problèmes de Cauchy additionnels doivent être résolus.

En notant

$$\begin{bmatrix} \Delta\lambda_0^{(i)} \\ \vdots \\ \Delta\lambda_N^{(i)} \end{bmatrix} = \Lambda^{(i+1)} - \Lambda^{(i)} \qquad\qquad F_k = F_k\left(\lambda_k^{(i)}, \lambda_{k+1}^{(i)}\right) \tag{7.12}$$

le problème (7.9) est équivalent au système d'équations suivantes

$$A\Delta\lambda_0 = -B\Delta\lambda_N - F_N \tag{7.13a}$$
$$\Delta\lambda_1 = G_0\Delta\lambda_0 - F_0 \tag{7.13b}$$
$$\Delta\lambda_2 = G_1\Delta\lambda_1 - F_1 \tag{7.13c}$$
$$\vdots$$
$$\Delta\lambda_N = G_{N-1}\Delta\lambda_{N-1} - F_{N-1} \tag{7.13d}$$

En partant de la seconde équation (7.13b), toutes les autres valeurs de $\Delta\lambda_k$ peuvent être exprimées successivement en fonction de $\Delta\lambda_0$. De cette façon

$$\Delta\lambda_1 = G_0\Delta\lambda_0 - F_1 \tag{7.14a}$$
$$\vdots$$
$$\Delta\lambda_N = G_{N-1}G_{N-2}\dots G_0\Delta\lambda_0 - \sum_{j=0}^{N-1}\left(\prod_{l=0}^{j-1} G_{N-1-l}\right)F_{N-1-j} \tag{7.14b}$$

Et en considérant la première équation du système (7.13a) et (7.14b), l'équation suivante est obtenue

$$\left(A + BG_{N-1}G_{N-2}\dots G_0\right)\Delta\lambda_0 = w \tag{7.15}$$

où $w = -\left(F_N + BF_{N-1} + BG_{N-1}F_{N-2} + \dots + BG_{N-1}G_{N-2}\dots G_1F_1\right)$. Une fois que $\Delta\lambda_0$ est obtenue, on peut calculer successivement $\Delta\lambda_1, \cdots, \Delta\lambda_N$ à partir de (7.14), puis $\Lambda^{(i+1)}$ grâce à (7.12).

7.1.2 Cas des problèmes de Cauchy

Le problème (1.1) s'écrivant ainsi

$$\begin{cases} \dot{y} = f(t,y) \text{ sur } [T_0,T] \\ y(T_0) = y_0 \end{cases} \tag{7.16}$$

sera considéré à partir de maintenant.

Nous avons vu dans l'équation (7.4) que pour un problème de Cauchy les conditions aux frontières s'expriment uniquement sur y_0, par définition

$$r(y(T_0), y(T)) = y(T_0) - y_0 = 0$$

On en déduit par la première équation de (7.7), $r(\lambda_0, \lambda_N) = 0$, que

$$\lambda_0 = y_0 \tag{7.17}$$

Le système d'équations (7.9) est ainsi réarrangé en supprimant la première contrainte $F_N(\lambda_0, \lambda_N) = 0$ et l'inconnue λ_0

$$F(\Lambda) = \begin{pmatrix} F_0(y_0, \lambda_1) \\ F_1(\lambda_1, \lambda_2) \\ \vdots \\ F_{N-1}(\lambda_{N-1}, \lambda_N) \end{pmatrix} = \begin{pmatrix} y(t_1; t_0, y_0) - \lambda_1 \\ y(t_2; t_1, \lambda_1) - \lambda_2 \\ \vdots \\ y(t_N; t_{N-1}, \lambda_{N-1}) - \lambda_N \end{pmatrix} = 0 \tag{7.18}$$

La matrice jacobienne est comme suit

$$J(\Lambda) = \begin{pmatrix} -I & 0 & & 0 & 0 \\ G_1 & -I & 0 & & 0 \\ 0 & G_2 & -I & \ddots & \\ & \ddots & \ddots & \ddots & 0 \\ 0 & & 0 & G_{N-1} & -I \end{pmatrix} \tag{7.19}$$

Le problème se réduit à

$$\Big\{ \Delta\lambda_1 = -F_0 \Delta\lambda_k = G_{k-1}\Delta\lambda_{k-1} - F_k \quad k = 2, \ldots, N \tag{7.20}$$

7.1.3 Parallélisation classique des méthodes de tirs

Pour construire numériquement les $N-1$ matrices G_k nécessaires dans les calculs successifs des $\Delta\lambda_k$ (équation (7.18)), d problèmes de Cauchy doivent être résolus. Ces $N \times d$ problèmes peuvent être résolus en parallèle [27, 95].

Algorithme 8 Méthode de tirs multiples : Newton

1: **repeat**
2: calcul en parallèle des matrices G_k (algorithme 9).
3: calcul séquentiel des $\Delta\lambda_k = G_{k-1}\Delta\lambda_k - F_k, \quad k = 1, \ldots, N$.
4: mise à jour de la solution λ_k par $\lambda_k + \Delta\lambda_k$
5: **until** $||\Delta\lambda_k|| < \epsilon$

Algorithme 9 Méthode de tirs : Calcul parallèle des $G_k^{(i)}$

1: **for** $k = 0, \ldots, N$ **do**
2: **for** $j = 1, \ldots, d$ **do**
3: Calcul de $\tilde{\lambda}_{k+1} = y(t_{k+1}; t_k, \lambda_k + \epsilon e_j)$
4: $G_k|_j = \dfrac{\tilde{\lambda}_{k+1} - \lambda_{k+1}}{\epsilon}$
5: **end for**
6: **end for**

Remarque 7.3. Le gain potentiel de la méthode réside dans la résolution parallèle des problèmes de Cauchy pour la construction des matrices G_k. Cependant le calcul des incréments $\Delta\lambda_k$ est séquentiel et peut constituer le goulet d'étranglement pour l'efficacité de la méthode.

L'algorithme 8, bien que largement répandu ([27, 6, 4] et cf sections 7.2 et 7.5), souffre de l'étape 3 séquentielle.

7.1.4 Nouvelle parallélisation des méthodes de tirs

Pour briser la séquentialité de l'étape 3 de l'algorithme 8, le Newton est considéré dans sa globalité, le système linéarisé peut ainsi être résolu entièrement, plutôt que successivement.

Pour résoudre efficacement en parallèle le système linéarisé, une méthode itérative de Krylov est envisagée dans laquelle le produit matrice-vecteur sera parallélisé comme dans l'algorithme 7.

Rappelons la méthode du résidu conjugué généralisé, méthode de Krylov permettant la résolution des systèmes linéaires $Jx = b$ non-symétriques.

Algorithme 10 Algorithme de Résidu Conjugué Généralisé

1: Calcul de $r_0 = b - Jx_0$, $d_0 = r_0$.
2: Normalisation de Jd_0 et d_0.
3: **for** $j = 0, 1, \ldots$ tant que $||d_j|| > \epsilon$ fixé **do**
4: $\alpha_j = (r_j, Jd_j)$
5: $x_{j+1} = x_j + \alpha_j d_j$
6: $r_{j+1} = r_j - \alpha_j Jd_j$
7: Calcul de Ar_{j+1}
8: **for** $i = 0, \ldots, j$ **do**
9: Calcul de $\beta_{ij} = (Jr_{j+1}, Jd_i)$
10: **end for**
11: $d_{j+1} = r_{j+1} - \sum_{i=0}^{j} \beta_{ij} d_i$
12: $Jd_{j+1} = Jr_{j+1} - \sum_{i=0}^{j} Jd_i$
13: Normalisation de Jd_{j+1} et d_{j+1}
14: **end for**

Dans cette méthode, le produit de la matrice J par les directions de descente, les vecteurs de la base $d_0, \ldots, d_j$ est parallélisé en distribuant l'évaluation des lignes du vecteur résultat. En reprenant les notations de l'algorithme 4, le produit matrice-vecteur s'écrit (algorithme 11) :

Définissons le speed-up théorique comme le speed-up de la méthode sur un processeur par rapport à np processeurs. Ce n'est pas le speed-up réel qui est le rapport de la meilleure méthode séquentielle par rapport à la méthode parallèle.

Algorithme 11 Produit matrice-vecteur $Jx = y$ (méthodes de tirs)

1: Réception non-bloquante de $x_{me,0}$.
2: Envoi non-bloquant de $x_{me,ns}$.
3: Calcul de $y_{me,j} = G_{me,j-1}x_{me,j-1} - x_{me,j}$, $j = 2, \ldots, ns$.
4: Attente de la réception
5: Calcul de $y_{me,1} = G_{me,0}x_{me,0} - x_{me,1}$.

Proposition 7.1. *Le speed-up théorique est linéaire.*

Démonstration. Dans l'algorithme 11, le gros volume de communications provient des échanges dans l'évaluation parallèle de la matrice jacobienne. À chaque itération j, les $j + 2$ produits scalaires nécessitent une communication de tous à tous pour envoyer un double. Le volume total de ces communications ont beaucoup moins important que celui intervenant dans le calcul de Jx.

Le coût du produit de la matrice J par le vecteur x est distribué sur les np processeurs. La taille de la matrice J étant dN, ce produit a une complexité en

$$O\left(2dN\right)$$

La distribution par bande de la matrice réduit ce coût à

$$O\left(\frac{2dN}{np}\right)$$

par processeur.

Ainsi, sans considérer les communications, le speed-up est np.

Or les communications (non-bloquantes) sont recouvertes par le calcul des y_j. Ce qui assure que le temps des communications sera à peine visible dans le temps d'exécution total.

Donc sur np processeurs, le speed-up théorique est np par rapport à la résolution de la même méthode sur un processeur. □

De plus, la méthode de tirs nécessite l'évaluation de la dérivée de trajectoire vis-à-vis des conditions initiales. L'estimation de chacune de ces quantités demande la résolution de $d+1$ problèmes à valeur initiale. Comme dans l'algorithme 8, la construction des matrices $G_k^{(i)}$ peut être effectuée en parallèle.

La combinaison des algorithmes 11 et 9 devrait offrir un bien meilleur speed-up que l'approche 8 et 9.

On a vu que le goulot d'étranglement de la méthode de tir est la correction des sauts qui est séquentielle. Une approche, toujours séquentielle, mais qui a pour but de réduire les calculs est proposé dans Parareal.

7.2 Parareal

Dans cette section, nous allons introduire la méthode Parareal proposée par J.L. Lions dans [61] dans le cadre de problèmes scalaires. Une extension au problèmes non-linéaires

fut ensuite proposée en considérant une linéarisation. Par la suite [64, 65, 26] introduisent différentes formulations dans le cas non-linéaire. Notre introduction suivra ce formalisme. Nous montrerons ici que Parareal est un cas particulier de la méthode de tirs multiples.

La méthode Parareal introduit deux opérateurs $\mathcal{F}$ et $\mathcal{G}$ qui représentent la trajectoire avec une discrétisation temporelle respectivement fine et grossière.

– $\mathcal{F}$ est un opérateur défini sur une grille fine tel que

$$\mathcal{F}(t_{k+1}; t_k, \lambda_k) = y_h(t_{k+1}; t_k, \lambda_k) \tag{7.21}$$

– $\mathcal{G}$ est lui défini sur une grille grossière tel que

$$\mathcal{G}(t_{k+1}; t_k, \lambda_k) = y_H(t_{k+1}; t_k, \lambda_k) \tag{7.22}$$

Les grilles sont définies par une décomposition de l'intervalle de temps $[T_0, T]$ en sous-domaines de longueur δt pour l'opérateur $\mathcal{F}$ et Δt pour $\mathcal{G}$. Bien entendu la grille fine définie par δt et la grille grossière par Δt sont telles que $\delta t \leq \Delta t$.

Algorithme 12 Algorithme de Parareal

$\lambda_0^0 = y_0$
for $k = 0, \ldots, N-1$ **do**
  $\lambda_k^0 = \mathcal{G}\left(\lambda_k^0\right)$
end for
$i = 0$
repeat
  $i = i + 1$
  résolution de $\mathcal{F}$ en parallèle sur chaque processeur $k = 0, \ldots, N-1$.
  for $k = 0, \ldots, N-1$ **do**
    résolution de $\mathcal{G}\left(\lambda_k^{(i+1)}\right)$

$$\lambda_{k+1}^{(i+1)} = \mathcal{G}\left(\lambda_k^{(i+1)}\right) + \mathcal{F}\left(\lambda_k^{(i)}\right) - \mathcal{G}\left(\lambda_k^{(i)}\right) \tag{7.23}$$

  end for
until $||\lambda_{k+1}^{(i+1)} - \lambda_{k+1}^{(i)}|| < \epsilon$

Proposition 7.2. *[48] La méthode Parareal est une méthode de tirs multiples.*

Démonstration. Il suffit de montrer que Parareal est un cas particulier des méthodes de tirs.

Reprenant l'équation (7.9),

$$\Lambda^{(i+1)} = \Lambda^{(i)} - J(\Lambda^{(i)})^{-1} F(\Lambda^{(i)})$$

avec la matrice jacobienne 7.19, et en utilisant sa structure particulière, la forme de récurrence pour la méthode de tirs multiples s'écrit simplement

$$\begin{cases} \lambda_0^{(i+1)} = y_0 \\ \lambda_{k+1}^{(i+1)} = y\left(t_{k+1}; t_k, \lambda_k^{(i)}\right) + G_k(\lambda_k^{(i+1)} - \lambda_k^{(i)}) \end{cases} \tag{7.24}$$

Bien que l'équation (7.24) soit continûment vérifiée, afin de résoudre la méthode de tirs multiples, une discrétisation des équations différentielles sur chaque intervalle doit être choisie. Ce qui implique de disposer d'une méthode de résolution pour calculer $y(t_{k+1}; t_k, \lambda_k)$ ainsi que pour calculer les matrices G_k (7.11a).

Si on note $\mathcal{F}$ l'opérateur tel que

$$y\left(t_{k+1}; t_k, \lambda_k^{(i)}\right) = \mathcal{F}(t_{k+1}; t_k, \lambda_k^{(i)}) \tag{7.25}$$

et si on approxime le second terme $G_k(\lambda_k^{(i+1)} - \lambda_k^{(i)})$ par

$$G_k(\lambda_k^{(i+1)} - \lambda_k^{(i)}) = \mathcal{G}(t_{k+1}; t_k, \lambda_k^{(i+1)}) - \mathcal{G}(t_{k+1}; t_k, \lambda_k^{(i)}) \tag{7.26}$$

alors l'équation (7.24) n'est autre que l'équation de Parareal (7.23). □

D'après [94], plusieurs propriétés de Parareal peuvent être dressées, à savoir :

Propriété 7.1 (Convergence vers la solution séquentielle : [94])**.** *L'algorithme Parareal converge vers la solution séquentielle obtenue en utilisant le même solveur et la même discrétisation en temps, c'est à dire en utilisant l'opérateur $\mathcal{F}$.*

Propriété 7.2. *[94] Étant donnés les opérateurs $\mathcal{F}$ et $\mathcal{G}$ convergents et stables pour les paramètres δt et Δt choisis, alors après i itérations, la différence entre la solution séquentielle y_s et la solution parallèle y_p vérifie*

$$||y_s - y_p|| < \epsilon$$

sur l'intervalle $[T_0, i\Delta t]$, et ϵ est la précision machine. Ce qui signifie que

$$||y_s - y_p|| < \epsilon$$

sur $[T_0, T]$ en au plus $N = \frac{T - T_0}{\Delta t}$ itérations.

Parareal étant une méthode de tir (proposition 7.2) converge en au plus N itérations vers la solution de $y_h(t_{k+1}; t_k, \lambda_k)$. Il est important de garder à l'esprit que l'algorithme converge vers la précision demandée par $\mathcal{F}$, et ne peut en aucun cas être plus précis que la résolution séquentielle. Ceci signifie que les itérations successives après la tolérance recherchée sont superflues.

De plus la propagation de la solution par l'opérateur $\mathcal{F}$ n'offre aucun bénéfice lorsque la méthode est itérée N fois, car cela revient à résoudre séquentiellement le problème en appliquant le solveur $\mathcal{F}$.

Ainsi d'après [2], si les coûts calculatoires de $\mathcal{F}$ et $\mathcal{G}$ sont supposés identiques d'ordre 1, alors le speed-up [1] après i itérations est

$$S = \frac{\frac{T}{\delta t}}{i\frac{T}{\Delta T} + (i-1)\frac{\Delta T}{\delta t}} \tag{7.27}$$

1. voir la définition 1.1

Le nombre de processeurs est donné par $np = \frac{T}{\Delta T}$ de sorte que l'efficacité [2] s'écrit

$$E = \frac{1}{(i-1) + i\frac{T\delta t}{\Delta T^2}} \tag{7.28}$$

Le speed-up est maximal pour $i = 2$ et $\Delta T = \sqrt{2T\delta t}$. Ainsi

$$S = \frac{1}{2}\sqrt{T}2\delta t \qquad np = \sqrt{T}2\delta t \qquad E = \frac{1}{2} \tag{7.29}$$

Remarque 7.4. Une efficacité maximale de 50% dans le cas d'un calcul qui converge en deux itérations est un critère très contraignant. La classe de problèmes sur laquelle il peut être vérifié est extrêmement limitée. Ce résultat montre que Parareal est difficilement envisageable pour la résolution de problèmes industriels.

7.3 Pita

Dans la méthode de tirs multiples, la matrice Jacobienne peut-être calculée en résolvant les équations variationnelles associées ([8]), qui s'écrivent ainsi

$$\begin{cases} \partial_{\lambda_k} y(t_{k+1}; t_k, \lambda_k)' = f'(y_k)\partial_{\lambda_k} y(t_{k+1}; t_k, \lambda_k) \\ \partial_{\lambda_k} y(t_k; t_k, \lambda_k) = I \end{cases} \tag{7.30}$$

Ces équations linéaires sont en général des équations matricielles de taille d. Elles sont classiquement résolues en même temps que les équations de tirs et par la même méthode puisque les valeurs de $y(t_{k+1}; t_k, \lambda_k)$ sont nécessaires.

L'équation (7.30) est une équation matricielle mais seulement l'action de la matrice Jacobienne sur la différence des solutions (7.24) est nécessaire. Il est alors plus efficace de calculer le vecteur résultat sur la grille grossière.

$$\begin{cases} \delta\lambda_k^{(i)'} = f'\left(\lambda_k^{(i)}\right)\delta\lambda_k^{(i)} \\ \delta\lambda_k^{(i)}(t_k) = \lambda_k^{(i+1)} - \lambda_k^{(i)} \end{cases} \tag{7.31}$$

Algorithme 13 Algorithme de Pita

1: Calcul des valeurs initiales $\lambda_k^{(0)}$ pour $0 \leq k \leq N$
2: **repeat**
3: résolution fine $\mathcal{F}(\lambda_k^{(i)})$.
4: évaluation des sauts $\Delta_i = \lambda_k^{(i+1)} - \lambda_k^{(i)}$.
5: calcul séquentiel des termes correctifs $\delta\lambda_k^{(i)}$ et mise à jour de la solution $\lambda_{k+1}^{(i+1)}$.
6: **until** $||\delta\lambda_k|| < \epsilon$, $k = 1, \ldots, N$

Dans la suite, nous introduisons une adaptabilité dans la définition de la finesse des deux grilles basée sur la tolérance relative de l'intégrateur en temps. L'avantage est de pouvoir utiliser le choix automatique de pas de temps comme un indicateur sur la raideur du problème et ainsi pouvoir passer les non-linéarités fortes.

2. voir la définition 1.2

7.4 Algorithme adaptatif pour Pita

Une nouvelle notion de grilles fines et grossières est introduite ci-après. Ces grilles ne sont plus définies par la taille du pas de temps mais par la tolérance relative de l'intégrateur. L'avantage est que l'adaptivité du pas de temps choisi par le schéma d'intégration permet de mieux gérer les fortes non-linéarités du modèle.

L'adaptivité du pas de temps est basé sur une estimation de l'erreur entre la solution y_1 d'ordre p calculée par le schéma et la solution $\hat{y}_1$ d'un schéma d'ordre $\hat{p}$. Le pas de temps h sera choisi pour qu'il vérifie le critère $|y_1 - \hat{y}_1| \leq max(y_{0i}, y_{1i})tol$.

Le prochain pas de temps h_{new} est obtenu par la formule de relaxation suivante qui permet d'éviter de grande variation entre les choix de pas de temps successifs

$$h_{new} = h.min(facmax, max(facmin, fac.(1/err)^{1/(q+1)}))$$

où $q = min(p, \hat{p})$ et $fac, facmin, facmax$ sont les facteurs qui évitent une trop grande augmentation ou diminution du pas de temps, et $err = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_{1i} - \hat{y}_{1i}}{sc}\right)^2}$.

7.4.1 Algorithme parallèle pour Pita

Le premier algorithme implémente rigoureusement l'algorithme Pita. Étant donné que le nombre de sous-domaines nécessaires est plus important que le nombre de processeurs accessibles, chaque processeur gère plusieurs sous-domaines.

Algorithme 14 Algorithme parallèle de Pita

1: Calcul des valeurs initiales $\lambda_k^{(0)}$ pour $0 \leq k \leq N$ avec un intégrateur avec une tolérance grossière.
2: **repeat**
3: Sur chaque processeur k, résolution fine de $\mathcal{F}(\lambda_k^{(i)})$.
4: Évaluation des sauts $\delta\lambda_k^{(i)} = \lambda_k^{(i+1)} - \lambda_k^{(i)}$.
5: Réception du terme correctif précédent.
6: Calcul séquentiel du terme correctif $\delta\lambda_k^{(i)}$.
7: Envoi du terme correctif au processeur suivant.
8: Mise à jour de la solution $\lambda_{k+1}^{(i+1)}$.
9: **until** $||\delta\lambda_k^{(i)}|| < \epsilon$, $k = 1, \ldots, N$

Des difficultés apparaissent lors de l'implémentation et de l'exécution de l'algorithme Pita sur le problème test.

– pour des problèmes complexes, la matrice jacobienne pour le problème sur la correction doit être calculée numériquement car sa forme analytique n'est pas connue ou difficile à avoir. Ceci implique $d + 1$ produits matrice-vecteur chacun d'une complexité d^2. Et donc la fonction évaluée pour la correction f_y est environ d fois plus coûteuse que la fonction f du problème ;

– comme la correction est petite en valeur absolue, les pas de temps de l'intégrateur sont plus petits que pour l'initialisation. Ainsi l'intégrateur fait plus de pas pour la correction ;
– comme la phase de correction est séquentielle, seulement un processeur est actif à la fois. C'est la phase la plus coûteuse en terme de temps de calcul dans l'implémentation.

7.4.2 Algorithme parallèle relaxé pour Pita

En fonction de la dernière remarque, une solution doit être trouvée pour apporter du parallélisme dans la phase de correction. Nous proposons une relaxation dans la condition de transmission de l'erreur entre le dernier sous-domaine d'un processeur et le premier du suivant. L'algorithme où la phase de correction est remplacée par celle-ci s'écrit

Algorithme 15 Algorithme parallèle relaxé de Pita

1: Calcul des valeurs initiales $\lambda_k^{(0)}$ pour $0 \leq k \leq N$ avec un intégrateur avec une tolérance grossière.
2: **repeat**
3: Sur chaque processeur k, résolution fine de $\mathcal{F}(\lambda_k^{(i)})$.
4: Évaluation des sauts $\delta\lambda_k^{(i)} = \lambda_k^{(i+1)} - \lambda_k^{(i)}$.
5: Calcul séquentiel du terme correctif $\delta\lambda_k^{(i)}$.
6: Réception du terme correctif précédent.
7: Envoi du terme correctif au processeur suivant.
8: Mise à jour de la solution $\lambda_{k+1}^{(i+1)}$.
9: **until** $||\delta\lambda_k^{(i)}|| < \epsilon$, $k = 1, \ldots, N$

7.5 Résultats parallèles pour Pita

Pour l'implémentation et les tests numériques, l'algorithme de tirs écrit sous la forme Pita a été utilisé. Les problèmes focalisés dans la thèse sont les problèmes raides, ceux pour lesquels les méthodes explicites sont inadaptées. Ce qui explique le choix de la résolution d'un système de Cauchy des G_k par une méthode quelconque avec pas adaptatif dans Pita, plutôt que d'utiliser un schéma d'Euler implicite comme suggéré dans Parareal.

7.5.1 Résultats Numériques pour des problèmes test non-linéaires

Trois problèmes du chapitre 2 sont considérés. Ils sont de raideurs croissantes pour valider et montrer les limites des méthodes précédentes. Le premier exemple est le système proie-prédateur (2.2). Le second est le modèle Oregonator (2.3). Enfin le dernier est le système différentiel de la modélisation d'une poutre encastrée (2.4).

La figure 7.2 donne la convergence de la méthode en fonction du nombre d'itérations. La convergence est donnée par l'erreur entre la solution obtenue par le schéma et une

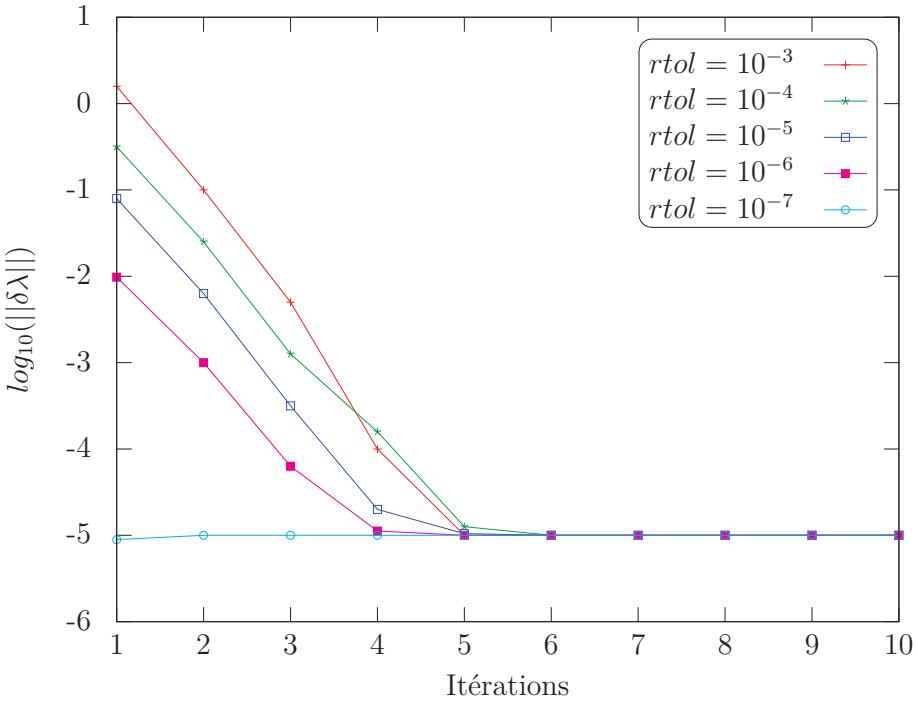


FIGURE 7.2 – Convergence de la méthode Pita en fonction de la tolérance sur la grille grossière, pour le problème Lotka-Volterra avec $\mu = (1.5, 1, 3, 1)$ et $N = 200$

solution de référence calculée par un solveur séquentiel à pas adaptatif avec une tolérance fixée à $rtol = 10^{-7}$.

La figure 7.3 montre la convergence de la méthode sur la correction pour le problème Lotka-Volterra. Les N sous-domaines sont soit définis de façon régulière avec une taille fixe $H = \frac{T-T_0}{N}$, soit définis de façon adaptative où la taille du sous-domaine i est la somme de pas de temps pris par le solveur sur la grille grossière. Pour 10 sous-domaines en temps, la méthode diverge pour des tolérances relatives $rtol = 10^{-3}, 10^{-4}$ et finit par converger à la dixième itération. Même en considérant une tolérance à $rtol = 10^{-7}$ la convergence n'est obtenue qu'au bout de 7 itérations. Pour un nombre de domaines de cet ordre la méthode n'a aucun intérêt. En effet, la méthode ne peut avoir un intérêt que lorsque le nombre d'itérations pour la version parallèle est moindre que le nombre de sous-domaines, en raison de la phase de correction qui est séquentielle et de la propagation de la solution exacte de proche en proche à chaque itération. Après 7 itérations, l'algorithme revient à avoir calculé l'algorithme séquentiel sur les 7 premiers sous-domaines.

Cependant, si le nombre de sous-domaines de même taille est augmenté jusqu'à 200, la convergence est atteinte entre 5 et 7 itérations pour $rtol = 10^{-6}$ et 10^{-3}, ce qui fournit un speed-up. Pour 1168 sous-domaines, la convergence est obtenue entre 2 et 7 itérations pour $rtol = 10^{-6}$ jusqu'à 10^{-3}.

Remarquons que la correction peut converger jusqu'à 10^{-14} mais la convergence de la solution est limitée par la tolérance du solveur sur la grille fine (ici $rtol = 10^{-7}$). Ce n'est donc pas nécessaire d'atteindre la précision de la machine pour avoir convergence sur la

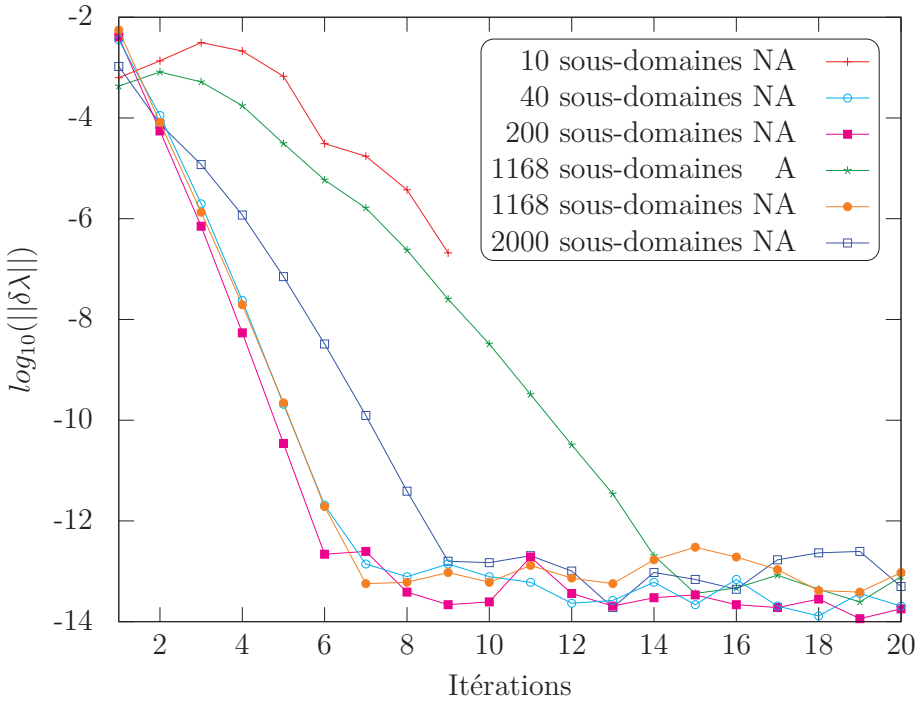


FIGURE 7.3 – Convergence de la correction en fonction du nombre de sous-domaines avec adaptivité (A) ou sans (NA) (pour une tolérance de $rtol = 10^{-5}$ sur la correction) pour le problème Lotka-Volterra avec $\mu = (1.5, 1, 3, 1)$

grille fine.

Dans la suite, on introduit une adaptativité dans la définition de la finesse des deux grilles basée sur la tolérance relative de l'intégrateur en temps. L'avantage est de pouvoir utiliser le choix automatique de pas de temps comme un indicateur sur la raideur du problème et ainsi pouvoir passer les non-linéarités fortes. Dans le code, la taille des sous-domaines est adaptée en fonction des variations du pas de temps en ne considérant qu'un certain nombre de pas de temps pour chacun des sous-domaines. La figure 7.3 montre que 1168 sous-domaines définis par l'adaptivité (A) donnent de meilleurs résultats. Néanmoins ce n'est pas le nombre optimal de sous-domaines, 2000 sous-domaines réguliers (NA) conduisent à une convergence plus rapide, en moins d'itérations, de plus le même nombre de sous-domaines NA, 1168 donne quasiment les mêmes résultats.

Pour le problème Oregonator, la convergence n'est pas atteinte même avec 1000 sous-domaines pour des tolérances de $rtol = 10^{-6}$ à 10^{-3} que ce soit avec ou sans la décomposition adaptative en temps.

7.5.2 Comparaisons entre les deux algorithmes

Une itération de Pita de l'algorithme 15 devient totalement parallèle. Cependant, cette approche introduit une erreur — un saut de la solution — entre les sous-domaines de deux processeurs voisins.

La figure 7.4 montre la convergence en norme infinie de la correction sur chaque processeur (125 sous-domaines par processeur) pour le problème de poutre encastrée en fonction du nombre d'itérations de Pita. La convergence est plus rapide pour les premiers processeurs gérant les premiers sous-domaines. Ceci permet de libérer dynamiquement les processeurs lorsque la convergence est atteinte. Remarquons que la solution est d'ordre 1 sur le dernier sous-domaine, et que la tolérance relative est défini à 10^{-5}, ainsi lorsque la correction est de l'ordre 10^{-6} alors la précision réquise pour la solution est atteinte.

Les résultats numériques présentés ici ont été réalisés sur une machine Compaq Sierra-cluster avec des processeurs alpha ev67/600Mhz de 8MB L2-cache et avec une bande passante de 800Mb/s pour le réseau de communication. La bibliothèque MPI constructeur est utilisé pour gérer les communications entre les processeurs.

Le tableau 7.1 montre le temps d'exécution et l'efficacité de l'algorithme 15 de Pita pour la poutre encastrée avec un nombre d'inconnues n égal à 200×2 pour seulement 3 itérations.

Problème de la poutre avec 200×2 inconnues				
temps d'exécution (s)	nombre de processeurs			
	2	4	8	16
Initialisation à $rtol = 10^{-2}$	1394	1366	1364	1366
Résolution sur la grille fine à $rtol = 10^{-5}$	2526	1080	587	312
Résolution de la correction à $rtol = 10^{-3}$	11260	4265	2184	1149
Total	15180	6711	4135	2827

TABLE 7.1 – Efficacité parallèle de l'algorithme 15 sur le problème de la poutre.

L'algorithme 14 offre de meilleures performances numériques pour trois itérations, tandis que l'algorithme 15 a besoin de plus d'itérations pour converger. La convergence dépend du nombre de processeurs pour l'algorithme 15 à cause de l'étape de relaxation. Cependant l'algorithme 15 est complètement parallèle et réduit d'autant le temps passé dans la phase de correction. La première communication peut-être recouverte par la linéarisation de la matrice jacobienne sur chaque sous-domaine.

L'efficacité parallèle est limitée ici par le petit nombre de processeurs accessibles. Cependant, de part la performance, l'algorithme 15 devient attractif pour réduire le temps d'exécution qui dépend fortement du nombre de processeurs accessibles et du nombre d'itérations de Pita nécessaire pour atteindre la tolérance demandée.

problème de la poutre avec 200×2 inconnues			
temps d'exécution (s)	nombre de processeurs		
	4	8	16
Initialisation à $rtol = 10^{-2}$	1135	2271	4550
Résolution de la grille fine à $rtol = 10^{-5}$	1484	1531	294 (1it)
Résolution de la correction à $rtol = 10^{-3}$	5946	6012	1198 (1it)

TABLE 7.2 – Extensibilité de l'algorithme 15 sur le problème de la poutre.

Le tableau 7.2 donne la scalabilité de l'algorithme 15. Chaque processeur gère 250 sous-domaines et 3 itérations de Pita sont effectuées. Les 16 processeurs convergent en

seulement 1 itération due au nombre important de sous-domaines globaux. De plus, le tableau montre que l'initialisation n'est pas très *extensible* alors que la phase de correction et la solution sur la grille fine le sont.

La méthode Pita est donc dans la pratique assujettie à des problèmes d'instabilités et pas totalement parallèle. Ceci nous a conduit à investiguer d'autres manières de paralléliser en temps.

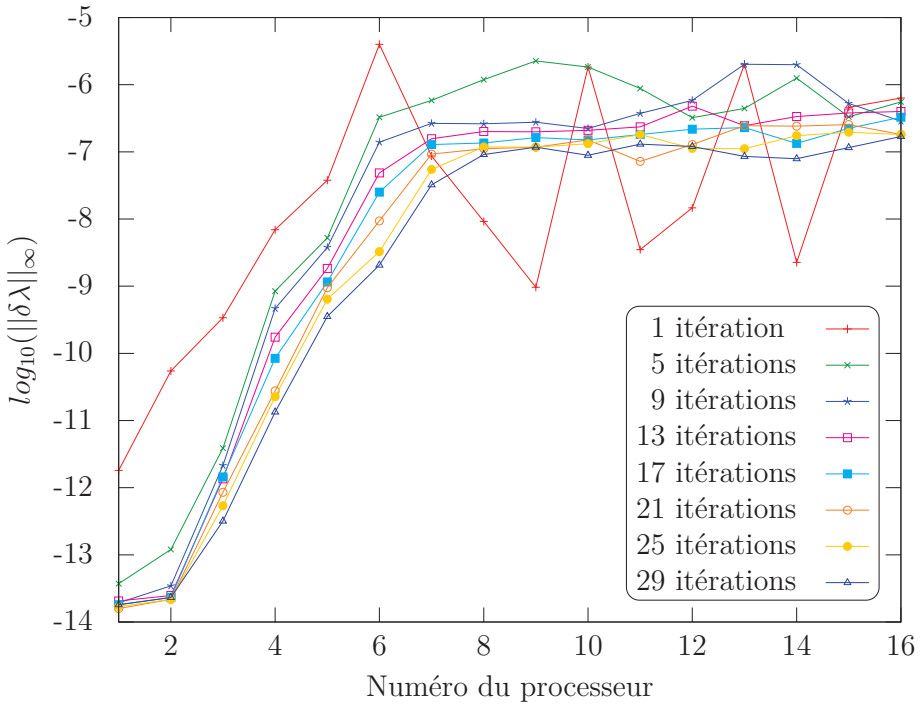


FIGURE 7.4 – Convergence pour le problème de poutre de l'algorithme 14 en fonction du nombre de processeurs pour un nombre différent d'itérations

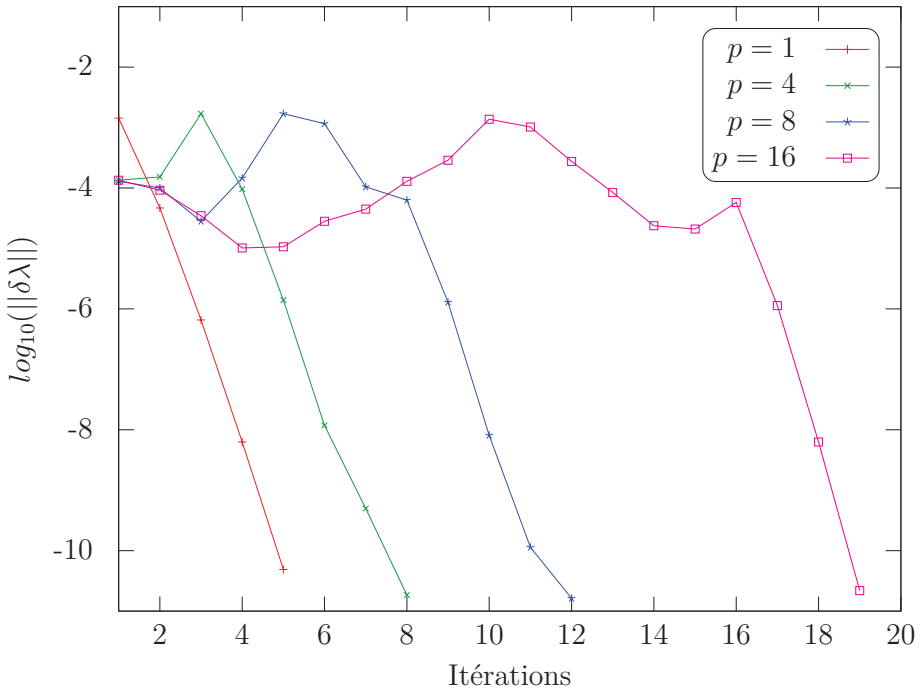


FIGURE 7.5 – Comparaison de la convergence pour l'algorithme 15 sur le problème de poutre en fonction du nombre de processeurs.

CHAPITRE 8

Méthodes basées sur une estimation de l'erreur

8.1 Introduction sur les estimateurs d'erreur

Définition 8.1. Considérant le problème (1.1), on note y la solution exacte et y_n l'approximation numérique obtenu par un schéma d'intégration d'ordre p (cf. def. 3.2).

L'erreur globale est définie par $E_n = y_n - y(t_n)$.

Soit y_n l'approximation de la solution au temps t_n par un schéma d'intégration ϕ. Entre deux instants t_n et t_{n+1} le schéma introduit une erreur appelée erreur de troncature donnée par la formule

$$\epsilon_n = y(t_{n+1}) - (y(t_n) - h_n\phi(y(t_n), h_n)) \tag{8.1}$$

Dans la pratique, la solution exacte y n'est pas connue, l'erreur globale du schéma d'intégration ne peut donc pas être estimée. C'est pour cela que l'on recherche un estimateur de l'erreur globale $\hat{E}_n \approx E_n$.

Définition 8.2 (Validité d'un estimateur)**.** Un estimateur $\hat{E}_n$ de E_n sera dit valide d'ordre relatif $r > 0$ si

$$E_n = (1 + O(H^r))\hat{E}_n \tag{8.2}$$

où $H = \max_n h_n$.

Remarque 8.1. La valeur de $\hat{y}_n = y_n - \hat{E}_n$ est alors d'ordre $p + r$.

Les estimateurs asymptotiques sont construits grâce au théorème suivant

Théorème 8.1. *[52] On suppose que le problème 1.1 est intégré avec la méthode d'intégration ϕ stable, d'ordre $p \geq 1$, à pas constant h. Alors il existe des fonctions e_k telles que l'on ait*

$$y_n - y(t_n) = h^p e_p(t_n) + \cdots + h^{p+q} e_{p+q}(t_n) + O(h^{p+q+1}) \tag{8.3}$$

uniformément sur $[T_0, T]$. *Les fonctions* e_k, $k = p, \ldots, p+q$, *sont solutions d'équations différentielles de la forme*

$$\begin{cases} \dot{e}_k(t) = f'(y(t))e_k(t) + \Psi_k(t) \\ e_k(0) = 0 \end{cases} \tag{8.4}$$

où f' *est le jacobien de* f *et* Ψ_k *un terme inhomogène dépendant de* ϕ *et de* f. *L'équation donnant* e_k *sera appelée* k*-ième équation variationnelle.*

Les méthodes introduites dans la suite sont basées sur la détermination d'un estimateur de l'erreur afin de corriger la solution préalablement calculée.

8.2 Éxtrapolation parallèle

Dans cette partie, l'estimateur de Richardson est rappelé. Il est communément utilisé pour calculer à partir de deux intégrations une solution plus précise. Puis en nous basant sur le comportement homothétique du pas de temps par rapport à la tolérance relative de la méthode d'intégration, nous construisons une méthode basée sur l'extrapolation de Richardson.

8.2.1 Rappel sur l'estimateur de Richardson

L'estimateur de Richardson est une méthode qui combine plusieurs approximations d'une certaine quantité et qui par une formule d'extrapolation en donne une meilleure approximation.

D'après [79], considérons une méthode $\mathcal{A}(h)$ disponible pour estimer une quantité α_0, quelque soit $h \neq 0$. De plus supposons qu'il existe un développement asymptotique de $\mathcal{A}$ à l'ordre k ($k \geq 0$)

$$\mathcal{A}(h) = \alpha_0 + \alpha_1 h + \cdots + \alpha_k (h)^k + \mathcal{R}_{k+1}(h) \tag{8.5}$$

où $\mathcal{R}_{k+1}(h) = O(h^{k+1})$ et où les cœfficients α_i sont indépendants de h.

Si l'équation (8.5) est écrite pour δh au lieu de h alors

$$\mathcal{A}(\delta h) = \alpha_0 + \alpha_1 \delta h + \cdots + \alpha_k (\delta h)^k + \mathcal{R}_{k+1}(\delta h) \tag{8.6}$$

En y soustrayant l'équation (8.5) multipliée par δ alors

$$\mathcal{B}(h) = \frac{\mathcal{A}(\delta h) - \delta \mathcal{A}(h)}{1-\delta} = \alpha_0 + \tilde{\alpha}_2 h^2 + \cdots + \tilde{\alpha}_k h^k + \tilde{\mathcal{R}}_{k+1}(h) \tag{8.7}$$

avec les $\tilde{\alpha}_i = \alpha_i \dfrac{\delta^i - \delta}{1-\delta}$ pour $i = 1, \ldots, k$ et $\tilde{\mathcal{R}}_{k+1} = \dfrac{\mathcal{R}_{k+1}(\delta h) - \delta \mathcal{R}_{k+1}(h)}{1-\delta}$.

Remarque 8.2. Notons que $\tilde{\alpha}_i \neq 0$ si et seulement si $\alpha_i \neq 0$.

En particulier si $\alpha_1 \neq 0$, alors $\mathcal{A}(h)$ est une approximation du premier ordre de α_0, alors que $\mathcal{B}(h)$ est au moins d'ordre 2.

Plus généralement, si $\mathcal{A}(h)$ est une approximation d'ordre p de α_0 alors

$$\mathcal{B}(h) = \frac{\mathcal{A}(\delta h) - \delta^p \mathcal{A}(h)}{1 - \delta^p}$$

est une approximation d'ordre $p+1$ (au moins).

Proposition 8.1 (Estimateur de Richardson appliqué à l'estimation de l'erreur d'un problème différentiel[52])**.** *Les développements asymptotiques de l'erreur entre une intégration de pas h et une autre de pas $2h$ en utilisant la même méthode sont*

$$y_n - y(t_n) = h^p e_p(t_n) + O(h^{p+1}) \tag{8.8}$$
$$y_{2n} - y(t_n) = 2^p h^p e_p(t_n) + O(h^{p+1}) \tag{8.9}$$

d'où l'erreur approchée

$$\hat{E}_n = \frac{y_n - \hat{y}_{2n}}{1 - 2^p} + O(h^{p+1}) \tag{8.10}$$

fournit une meilleure approximation de l'erreur E_n. Et

$$\hat{y}_{2n} = y_{2n} + \frac{y_n - \hat{y}_{2n}}{1 - 2^p} \tag{8.11}$$

est une meilleure approximation de y_{2n}.

8.2.2 Extrapolation parallèle

On s'est intéressé à combiner les méthodes de tirs et la méthode de Richardson.

L'extrapolation permet d'avoir un contrôle sur l'erreur sur la valeur initiale de chaque sous domaine en temps. La résolution sur chaque sous domaine est une résolution avec un solveur dont le choix du pas de temps est adapté en fonction de la raideur du problème comme retranscrit à la section 7.4.

Considérons le problème linéaire suivant

$$\begin{cases} \dot{y}(t) = Ay(t) \\ y(0) = y_0 \end{cases} \tag{8.12}$$

Les méthodes de tirs cherchent à résoudre N problèmes identiques définis sur les sous domaines $[t_i, t_{i+1}]$ pour $i = 1, \ldots, N$. Prenons par exemple la méthode d'Euler explicite pour la résolution de ces N problèmes avec un pas de temps δt tel que $\delta t = \dfrac{t_{i+1} - t_i}{3M} = h_M$

La solution en trois instants $t_{1,i+1}$, $t_{2,i+1}$ et $t_{3,i+1}$ sur la premier grille s'écrit

$$\begin{aligned} y^1(t_{1,i+1}) &= (I + A\delta t)^M y_i \\ y^1(t_{2,i+1}) &= (I + A\delta t)^{2M} y_i \\ y^1(t_{3,i+1}) &= (I + A\delta t)^{3M} y_i \end{aligned} \tag{8.13}$$

Pour la résolution sur une grille plus fine le pas $h_{M'}$ est plus petit que h_M. Comme pour l'extrapolation de Richardson, M' peut être pris égale à $\dfrac{M}{2}$, autrement dit $h_{M'} = \dfrac{h_M}{2}$.

Ainsi pour obtenir la solution au temps t_{i+1}, il faut deux fois plus de valeurs. La solution sur la seconde grille est

$$\begin{aligned} y^2(t_{1,i+1}) &= \left(I + A\frac{\delta t}{2}\right)^{2M} y_i \\ y^2(t_{2,i+1}) &= \left(I + A\frac{\delta t}{2}\right)^{4M} y_i \\ y^2(t_{3,i+1}) &= \left(I + A\frac{\delta t}{2}\right)^{6M} y_i \end{aligned} \tag{8.14}$$

Il en va de même sur une troisième grille où $M'' = \dfrac{M}{4}$, et $h_{M''} = \dfrac{\delta t}{4}$:

$$\begin{aligned} y^3(t_{1,i+1}) &= \left(I + A\frac{\delta t}{4}\right)^{4M} y_i \\ y^3(t_{2,i+1}) &= \left(I + A\frac{\delta t}{4}\right)^{8M} y_i \\ y^3(t_{3,i+1}) &= \left(I + A\frac{\delta t}{4}\right)^{12M} y_i \end{aligned} \tag{8.15}$$

La formule d'extrapolation est définie comme étant le polynôme d'ordre 3 qui à partir de ces neufs solutions coïncide avec la solution d'une grille encore plus fine ($M^{(3)} = \dfrac{M}{8}$).

Les cœfficients β_i, $i = 1, \ldots, 3$ du polynôme vérifient le système suivant

$$\begin{pmatrix} (I + A\delta t)^M y_i & \left(I + A\frac{\delta t}{2}\right)^{2M} y_i & \left(I + A\frac{\delta t}{4}\right)^{4M} y_i \\ (I + A\delta t)^{2M} y_i & \left(I + A\frac{\delta t}{2}\right)^{4M} y_i & \left(I + A\frac{\delta t}{4}\right)^{8M} y_i \\ (I + A\delta t)^{3M} y_i & \left(I + A\frac{\delta t}{2}\right)^{6M} y_i & \left(I + A\frac{\delta t}{4}\right)^{12M} y_i \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} = \begin{pmatrix} (I + A\delta t)^{8M} y_i \\ \left(I + A\frac{\delta t}{2}\right)^{16M} y_i \\ \left(I + A\frac{\delta t}{4}\right)^{24M} y_i \end{pmatrix} \tag{8.16}$$

$$\left(y_i \otimes I_3\right) \begin{pmatrix} (I + A\delta t)^M & \left(I + A\frac{\delta t}{2}\right)^{2M} & \left(I + A\frac{\delta t}{4}\right)^{4M} \\ (I + A\delta t)^{2M} & \left(I + A\frac{\delta t}{2}\right)^{4M} & \left(I + A\frac{\delta t}{4}\right)^{8M} \\ (I + A\delta t)^{3M} & \left(I + A\frac{\delta t}{2}\right)^{6M} & \left(I + A\frac{\delta t}{4}\right)^{12M} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} = \left(y_i \otimes I_3\right) \begin{pmatrix} (I + A\delta t)^{8M} \\ \left(I + A\frac{\delta t}{2}\right)^{16M} \\ \left(I + A\frac{\delta t}{4}\right)^{24M} \end{pmatrix} \tag{8.17}$$

Ce système est indépendant de la condition initiale y_i au temps t_i. Les cœfficients β ne dépendent donc que du choix des grilles via les pas de temps. Donc la détermination des cœfficients β à partir de la première grille, c'est-à-dire à partir de y_0 en t_0, garantie que les solutions extrapolées aux temps t_i, $i = 1, \ldots, N$ sont égales aux solutions calculées sur la grilles la plus fine.

Pour une méthode d'intégration d'ordre p, le pas de temps vérifie le critère suivant

$$|y_1 - \hat{y}_1| \leq Ch_n^p \leq sc = atol + max(y_0, y_1)rtol \tag{8.18}$$

entre la solution y_1 et $\hat{y}_1$ celle obtenue par schéma proche d'ordre $p-1$.

On montre ensuite que définir la finesse des grilles par rapport à la tolérance est équivalent à fixer le pas de temps. Fixer la tolérance a l'avantage de permettre le choix automatiquement sur le pas de temps donc de relaxer les coûts de calculs de la méthode d'intégration.

Lorsque les tolérances (relative $rtol$ et absolue $atol$) sont divisées par un facteur α, les pas de temps h_n de l'intégration à $rtol$ et $h_{\alpha n}$ à $rtol/\alpha$, sont reliés par la relation suivante

$$\begin{aligned} Ch_{\alpha n}^p &= \alpha \left(atol + max(y_0, y_1)rtol\right) \\ &= \alpha Ch_n^p \end{aligned} \tag{8.19}$$

Donc

$$h_{\alpha n} \equiv \alpha^p h_n \tag{8.20}$$

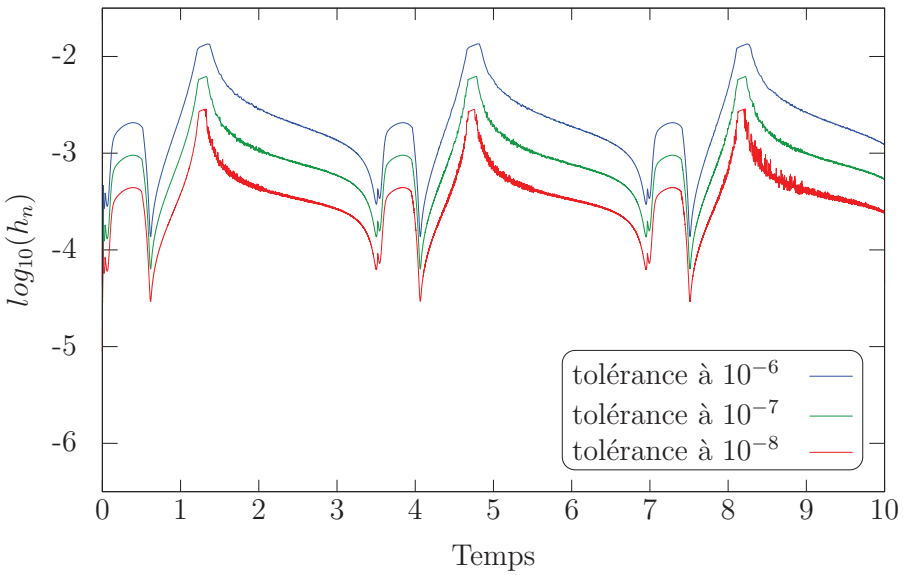


FIGURE 8.1 – Comparaison entre les comportements du pas de temps adaptatif en fonction de la tolérance relative $rtol$ (avec la tolérance absolue égale à $\frac{rtol}{1000}$) pour le problème Oregonator avec $\mu = (0.01, 0.001, 0.01, 1)$.

Nous avons appliqué l'extrapolation parallèle au problème non-linéaire 2.3. Tout d'abord les pas de temps sont retranscrits sur la figure 8.1. La résolution a été effectuée en utilisant le solveur ode23s de Matlab.

Les solutions obtenues pour les différentes tolérances ont un comportement similaire. On introduit alors le schéma de type "Richardson" pour reproduire ce comportement sur tous les sous domaines à partir du premier sous domaine.

Soit $y^k(t_1, j), 1 \leq j \leq l, 1 \leq k \leq l$ la solution calculée sur la grille $rtol_k$ au point de contrôle $t_{1,j}$. L'extrapolation peut être calculée à partir de la formule

$$\begin{pmatrix} y^1(t_{1,1}) & y^2(t_{1,1}) & \cdots & y^l(t_{1,1}) \\ y^1(t_{2,1}) & y^2(t_{2,1}) & \cdots & y^l(t_{2,1}) \\ \vdots & \vdots & \ddots & \vdots \\ y^1(t_{l,1}) & y^2(t_{l,1}) & \cdots & y^l(t_{l,1}) \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_l \end{pmatrix} = \begin{pmatrix} y^{l+1}(t_{1,1}) \\ y^{l+1}(t_{2,1}) \\ \vdots \\ y^{l+1}(t_{l,1}) \end{pmatrix} \tag{8.21}$$

Les cœfficients des matrices d'extrapolation β_i sont alors transmis aux autres processeurs pour qu'ils effectuent leur extrapolation.

Algorithme 16 Extrapolation parallèle adaptative

1: Définition de l points de contrôle t_{ki} sur chaque intervalle $[t_k, t_{k+1}]$.
2: Calcul de la solution sur l grilles grossières définies par une intégration avec une tolérance $rtol_1 > \cdots > rtol_l$.
3: Calcul de la solution sur $[t_0, t_1]$ avec $rtol_{l+1}$.
4: Calcul des cœfficients β_i de l'extrapolation.
5: Propagation de l'opérateur d'extrapolation β_i aux autres intervalles de temps.
6: Calcul des solutions extrapolées

$$y^{l+1}(t_{0,j}) = \sum_{k=1}^{l} \beta_k y^k(t_{l,j-1}), \quad 2 \leq j \leq np \tag{8.22}$$

7: Calcul en parallèle de la solution sur chaque sous domaine avec pour valeur initiale la valeur extrapolée.

La figure 8.2 montre l'erreur commise aux points de contrôle entre une solution de référence calculée à une tolérance de 10^{-10} et les solutions calculées par l'extrapolation parallèle.

L'extrapolation est construite sur le premier domaine avec deux grilles grossières obtenues avec les tolérances respectives 10^{-5} et 10^{-6}, et la solution d'une grille plus fine à 10^{-8}. Cette extrapolation sert à obtenir les conditions initiales pour chacun des sous domaines à partir desquelles la résolution parallèle est effectuée avec la même tolérance (10^{-8}).

On a d'abord cherché à comparer la solution de l'extrapolation parallèle avec une solution calculée séquentiellement à la même tolérance. Les deux erreurs sont quasiment identiques, globalement l'erreur de l'extrapolation est plus faible sauf en certains points.

Le processus est itéré, l'extrapolation est reconstruite à partir de la solution calculée sur le premier sous domaine avec une tolérance plus fine 10^{-9}. On remarque que la solution

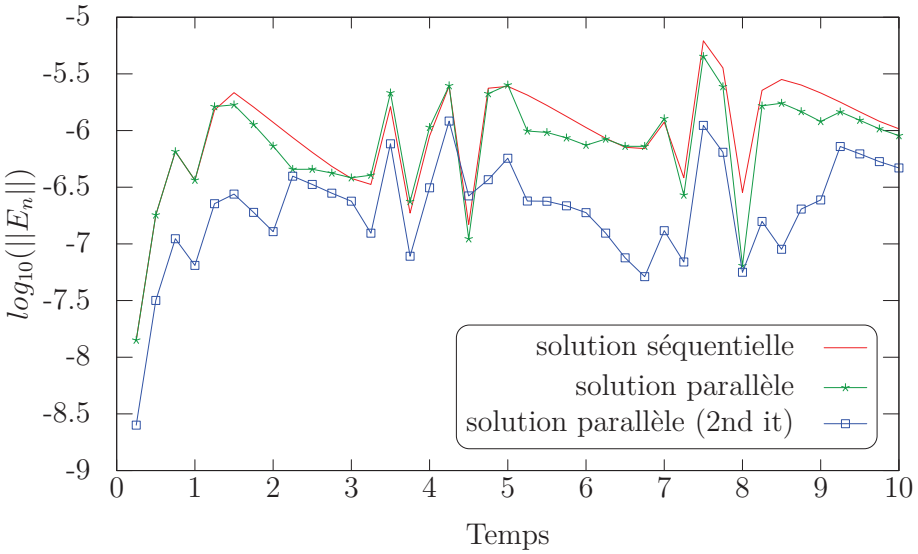


FIGURE 8.2 – Erreur entre une solution de référence à 10^{-10} et la solution séquentielle, et celles obtenues après une ou deux itérations de l'extrapolation parallèle pour le problème Oregonator avec $\mu = (0.01, 0.001, 0.01, 1)$.

obtenue à la seconde itération est globalement meilleure. Cependant l'erreur n'est pas uniformément améliorée.

Globalement la seconde solution est meilleure que la première, mais pas en certaines zones qui sont les zones de raideurs du problème.

8.3 Correction classique et spectrale du résidu

Les méthodes précédemment étudiées pour la décomposition en temps, ne fournissent pas d'estimation de l'erreur au niveau de la correction des sauts entre chaque domaine en temps.

Nous nous sommes intéressés à la méthode de correction du résidu ("defected correction method") sous sa forme simple (et spectrale). La forme simple est d'abord présentée.

8.3.1 Correction classique du résidu (DC)

On considère un problème différentiel classique sur l'intervalle $[T_0, T]$. Similairement aux méthodes précédentes et dans l'optique d'une décomposition en temps, cet intervalle est decoupé en $[t_n, t_{n+1}]$ avec $T_0 = t_0 < t_1 < \cdots < t_N = T$. Ainsi on souhaite approcher la solution du problème suivant :

$$\begin{cases} \dot{y} = f(t, y) \text{ sur } [t_n, t_{n+1}] \\ y(t_n) = y_n \end{cases} \tag{8.23}$$

Supposons qu'une approximation $y^0(\tilde{t}_m)$ de la solution ait été calculée aux points $\tilde{t}_m$ pour $m = 0, \cdots, M$ avec $t_n = \tilde{t}_0 < \tilde{t}_1 < \cdots < \tilde{t}_M = t_{n+1}$. En notant $y^0_m = y^0(\tilde{t}_m)$ et $p^0(t)$ l'unique polynôme d'ordre M qui interpole les y^0_m. On peut ainsi définir le résidu :

$$d(t) = (p^0)'(t) - f(t, p^0(t))$$

Ensuite l'approximation de l'erreur de discrétisation est donnée par

$$\bar{\delta}^0_m = p^0(\tilde{t}_m) - \psi^0_m$$

où ψ^0_m est solution du problème

$$\begin{cases} \dot{\psi}^0 = f(t, \psi^0) - d(t) \text{ sur } [t_n, t_{n+1}] \\ \psi^0(t_n) = y_n \end{cases}$$

Remarque 8.3. $p^0(t)$ est la solution exacte de ce problème.

De plus on a

$$y^1_m = y^0_m + p^0(\tilde{t}_m) - \psi^0_m = y^0_m + \bar{\delta}^0_m$$

qui est une meilleure approximation de $y(\tilde{t}_m)$ que y^0_m.

Cette procédure doit être itérée et se prête bien au pipelinage ce qui est étudié dans la section suivante.

8.3.2 Correction spectrale du résidu (SDC)

Soit le problème de Cauchy suivant

$$y(t) = y_0 + \int_0^t f(\tau, y(\tau))d\tau \tag{8.24}$$

Soit $y^0_m \simeq y(t_m)$ une approximation de la solution au temps t_m. Considérons ensuite le polynôme d'interpolation q_0 qui interpole la fonction $f(t, y)$ aux points $\tilde{t}_i$ i.e :

$$\exists! q_0 \in \mathbb{R}_M[x], \qquad q_0(\tilde{t}_i) = f(y^0(\tilde{t}_i)) \tag{8.25}$$

où $\left\{\tilde{t}_0 < \tilde{t}_1 < ... < \tilde{t}_M\right\}$ sont les points autour de t_m et $\mathbb{R}_M[x]$ est l'ensemble des polynômes sur $\mathbb{R}$ de degré M. Donc si l'erreur entre la solution approchée y^0_m et la solution approchée utilisant q_0 s'écrit

$$E(y^0, t_m) = y_0 + \int_0^{t_m} q^0(\tau)d\tau - y^0_m \tag{8.26}$$

Alors l'erreur entre la solution exacte et l'approximation y^0_m peut s'écrire

$$y(t_m) - y^0_m = \int_0^{t_m} \left(f(\tau, y(\tau)) - q^0(\tau)\right) d\tau + E(y^0, t_m) \tag{8.27}$$

Le défaut $\delta(t_m) = y(t_m) - y^0_m$ satisfait

$$\begin{aligned} \delta(t_{m+1}) = \delta(t_m) &+ \int_{t_m}^{t_{m+1}} \left(f(\tau, y^0(\tau) + \delta(\tau)) - q^0(\tau)\right) d\tau \\ &+ E(y^0, t_{m+1}) - E(y^0, t_m) \end{aligned} \tag{8.28}$$

Pour la *spectral deferred correction method*, l'approximation δ_m^0 de $\delta(t_m)$ est calculée en considérant par exemple un schéma d'Euler implicite

$$\begin{aligned}\delta_{m+1}^0 = \delta_m^0 + \Delta t(f(t_{m+1}, y_{m+1}^0 + \delta_{m+1}^0) - f(t_{m+1}, y_{m+1}^0)) \\ - y_{m+1}^0 + y_m^0 + \int_{t_m}^{t_{m+1}} q^0(\tau)d\tau\end{aligned} \tag{8.29}$$

Une fois que le résidu est calculé, l'approximation y_m^0 est mise à jour

$$y_m^1 = y_m^0 + \delta_m^0. \tag{8.30}$$

Un nouveau polynôme q_1 est calculé en utilisant les valeurs y_m^1 pour une nouvelle itération. Cette méthode est un processus séquentiel itératif se résumant comme suit

Algorithme 17 SDC

1: calcul d'une approximation y^0
2: **repeat**
3: calcul de δ_m
4: mise à jour de la solution y par ajout du terme correctif δ.
5: **until** $||\delta|| < \epsilon$ fixé

Dans la suite, nous proposons une version parallèle combinant la méthode SDC avec la décomposition en temps.

8.3.3 Développement d'une approche pipelinée pour la correction du résidu

Notre idée principale [45] pour introduire le parallélisme dans la SDC est de considérer la décomposition en temps et d'affecter un ensemble de sous-domaines en temps à chaque processeur. Alors le calcul peut être pipeliné où chaque processeur traite une itération différente de la SDC.

La principale difficulté est la transmission de valeurs de y_i aux temps $\tilde{t}_i, i = 1, M$ pour le calcul de la valeur du polynôme q_0 au temps t_m. Lorsque l'intervalle est découpé, à ses extrémités l'évaluation du polynôme requiert des valeurs des sous-intervalles voisins. Les valeurs de y_i requises peuvent naturellement constituer une zone de recouvrement entre les sous-intervalles, comme illustrée dans la figure 8.3.

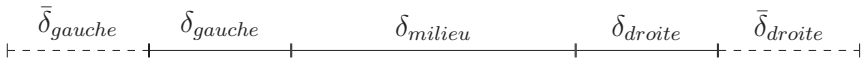


FIGURE 8.3 – Décomposition du sous-intervalle pour le calcul du polynôme q_0 : zones de recouvrement (en pointillé) et les trois zones internes

Algorithme 18 SDC pipelinée

1: Calcul des valeurs initiales $y^{(0)}$ pour $0 \leq k \leq N$ avec un intégrateur avec une tolérance grossière.
2: Distribution avec recouvrement des valeurs $y^{(0)}$ sur les processeurs.
3: **repeat**
4: Réception des valeurs dans le recouvrement à gauche, $\bar{\delta}_{gauche}$.
5: Réception non-bloquante des valeurs à droite, $\bar{\delta}_{droite}$.
6: Calcul de δ_{gauche} puis envoi au sous-domaine précédent.
7: Calcul de δ_{milieu}
8: Attente de la réception des valeurs à droite.
9: Calcul de δ_{droite} puis envoi au sous-domaine suivant.
10: Mise à jour de la solution $y_m^{i+1} = y_m^i + \delta_m^i$.
11: **until** $||\delta|| < \epsilon$

Remarque 8.4. L'étape 1, l'initialisation, ne peut pas être parallélisée. Car l'intégration doit respecter l'historique sur la solution. Ceci évite d'introduire une discontinuité de la dérivé à chaque instant t_i, $i = 1, \ldots, N$.

Remarque 8.5. L'algorithme sous sa forme séquentielle et parallèle effectue les mêmes opérations numériques. En effet, le découpage ne modifie pas l'algorithme car à un temps t_{k+1} on effectue les calculs sur l'intervalle $[t_k, t_{k+1}]$ sur un autre processeur et le processeur courant peut ainsi dès que possible faire son itération suivante.

#proc	1	2	4	8
étape 1 (s)	17.12	18.68	17.36	16.41
étape 4 (s)	1.35e-5	72.39	113	129
Euler	0.288	0.25	0.19	0.16
Total(s)	2062	1113	742	423
Efficacité	100%	92.6%	69.4%	60.9%
Speed-up	1	1.85	2.77	4.87
Speed up Corrigé	1	1.98	3.27	7.01

TABLE 8.1 – Temps, accélération et efficacité de la décomposition de domaine en temps SDC pipelinée

Le tableau 8.1 donne les temps d'exécution de la méthode SDC pipelinée en temps. Les tests ont été effectués sur une machines SGI-ALTIX350 avec 16 processeurs itanium 2 IA64 1.5Ghz/6Mo reliés par un réseau NumaLink. Les résultats sont obtenus avec le problème 2.5, la cavité entrainée avec un nombre de Reynolds égal à 10, un maillage uniforme de 20×20 points, le temps final à atteindre fixé à $T = 5$. Le nombre de sous-domaines est quant à lui fixé à 805. Le nombre d'itérations de SDC est de 25. L'étape 1 correspond à l'évaluation séquentielle de la solution sur le découpage en temps (avec une tolérance relative à 10^{-1}). Le temps de l'étape 2.2 est le temps d'attente du dernier processeur afin de commencer les calculs. Les résultats montrent que plus d'un quart du temps de calcul

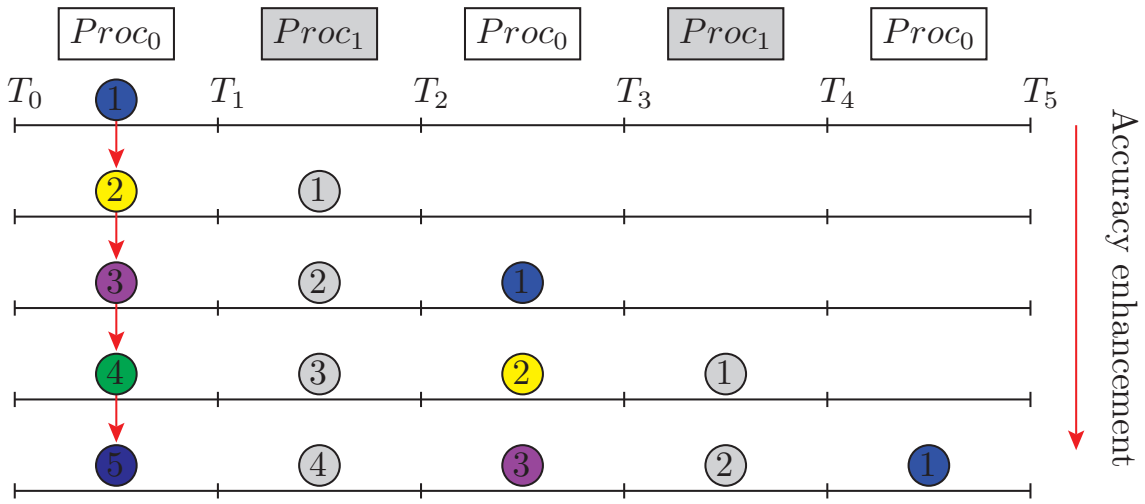


FIGURE 8.4 – SDC pipelinée avec distribution cyclique des sous-domaines par processeurs

est passé à attendre lorsque l'on a 8 processeurs. Le speed-up corrigé donne la capacité de l'implémentation parallèle de l'algorithme lorsque tous les processeurs sont en train de calculer.

Un speed-up raisonnable est obtenu sur 8 processeurs avec l'implémentation effectuée.

Dans cette approche, on constate que l'étape 4 occupe une place croissante pour les processeurs de la queue du pipe. Réduire ce temps d'attente permettrait d'avoir un speed-up proche du speed-up corrigé. C'est ce qui est développé dans la section suivante par une distribution cyclique des sous domaines sur chaque processeur.

8.3.4 Amélioration de l'approche proposée par une distribution cyclique

Dans [49], nous introduisons un distribution cyclique des sous-domaines par processeurs afin de diminuer les temps d'attente avant de débuter le calcul sur le dernier processeur. Cette distribution est illustrée dans la figure 8.4.

La difficulté technique, lors de l'implémentation du code parallèle, est l'activation des sous-domaines en fonction de l'état du pipe, afin de déterminer si le processeur doit envoyer et/ou recevoir des données. L'algorithme est retranscrit dans l'algorithme 19.

Algorithme 19 SDC cyclique

Calcul des valeurs initiales $y^{(0)}$ pour $0 \leq k \leq N$.
Distribution avec recouvrement des valeurs $y^{(0)}$ sur les processeurs.
repeat
  Rendre actif le sous-domaine i.
  for Sur chaque sous-domaine k actif **do**
    Réception des valeurs dans le recouvrement à gauche.
    Réception non-bloquante des valeurs à droite.
    Calcul de δ_{gauche} puis envoi au sous-domaine précédent.
    Calcul de δ_{milieu}
    Attente de la réception des valeurs à droite.
    Calcul de δ_{droite} puis envoi au sous-domaine suivant.
    Mise à jour de la solution $y_m^{k+1} = y_m^i + \delta_m^i$.
  end for
until $||\delta|| < \epsilon$

8.3.5 Résultats de la SDC cyclique sur le problème NS2D

La méthode de correction du résidu parallélisée a été appliquée au problème de la cavité entrainée (voir 2.5).

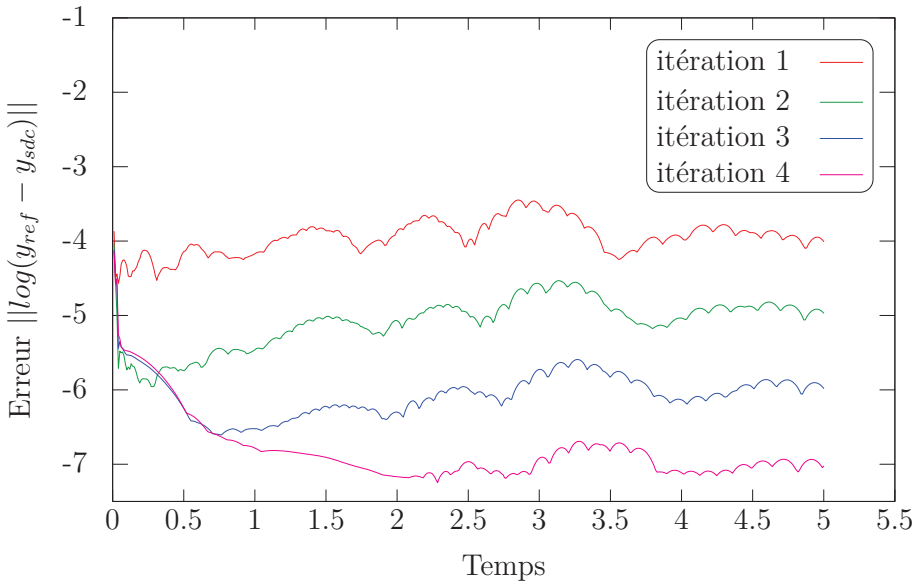


FIGURE 8.5 – Comparaison entre une solution de référence (calculée par une méthode BDF adaptative) et les solutions obtenues par la méthode spectrale de correction du résidu obtenues sur 500 intervalles pour le problème de NS2D ($Re = 400$ et un maillage régulier 40×40).

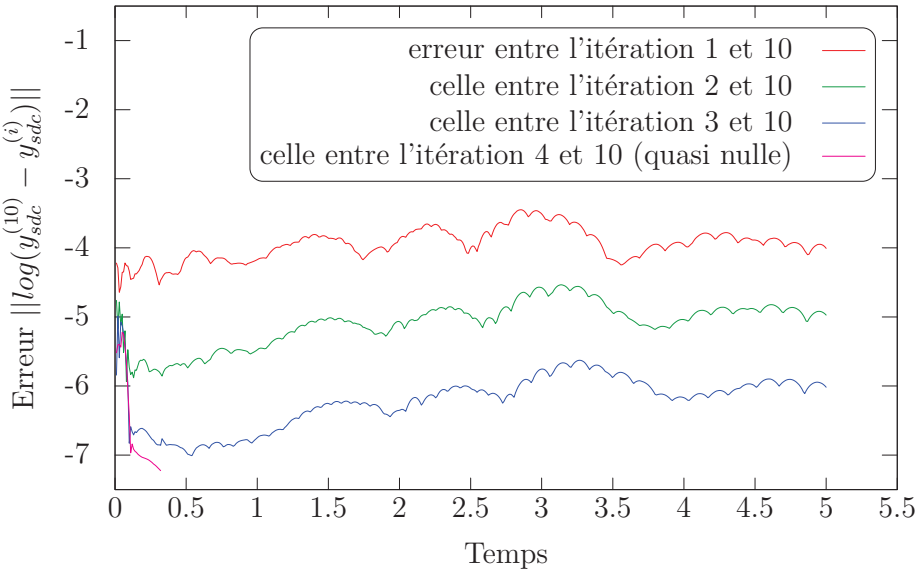


FIGURE 8.6 – Comparaison entre les solutions obtenues par la méthode spectrale de correction du résidu (erreur entre la solution convergée (à la 10e itération) et les solutions précédentes).

La figure 8.5 montre les résultats de convergence de la méthode parallèle pour le problème de la cavité entrainée pour un Reynolds de 400 et une grille de taille 40×40 sur $N = 500$ sous-domaine.

On remarque qu'entre 0 et 1s, l'erreur entre les itérations 3 et 4 est quasiment identique. Ceci s'explique par la nature instable de la solution du problème. Le contrôle automatique du pas de temps de BDF pour le calcul de la solution de référence adapte ce dernier pour gommer les oscillations numérique de la solution. Or la correction du résidu utilise un pas de temps fixe (ici $H = \frac{5}{500}$). Lorsque l'on compare le comportement de la correction du résidu (figure 8.6), on observe bien qu'à chaque itération on monte en ordre.

La figure 8.7 montre le temps d'exécution de la méthode en fonction du nombre de processeurs. On peut constater néanmoins que le temps d'attente du remplissage du pipe augmente avec le nombre de processeurs. Ce temps est le facteur limitant du code.

Afin de diminuer l'impact du temps d'attente sur le temps d'exécution, nous avons cherché à minimiser le nombre de sous-domaines gérés par le processeur avant qu'il ne passe l'information sur le processeur suivant. Le nombre de cycles par processeurs a été augmenté. C'est ce que nous observons dans la figure 8.8.

Nous remarquons que malgré une bonne réduction du temps de remplissage du pipe, le temps d'exécution n'est pas d'autant réduit. Ceci s'explique par un problème de charge, le problème de la cavité entrainée tend vers un état d'équilibre, ce qui déstabilise la complexité de l'intégration en fonction de l'intervalle de temps considéré.

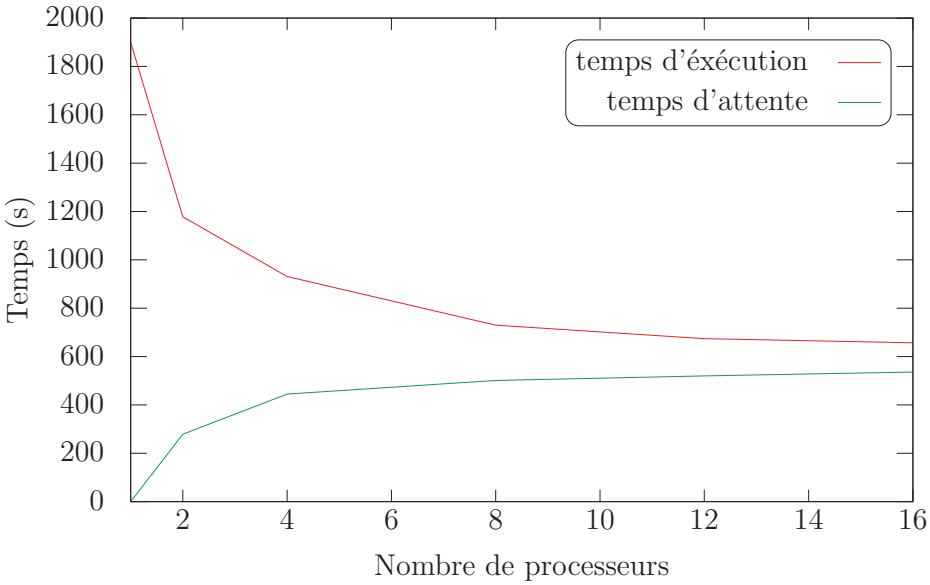


FIGURE 8.7 – Temps d'exécution et temps d'attente de la Pipelined SDC en fonction du nombre de processeurs

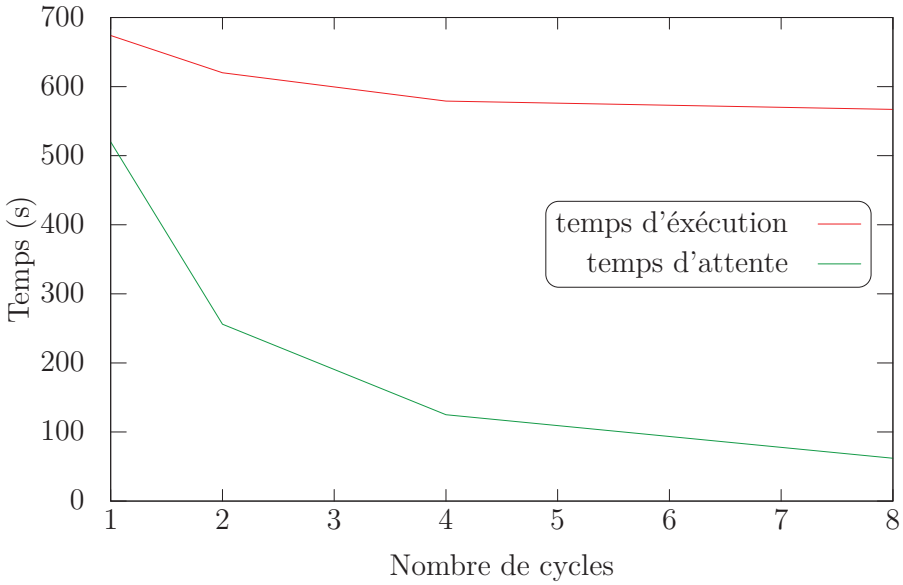


FIGURE 8.8 – Temps d'exécution et d'attente de la Pipelined SDC en fonction du nombre de cycles par processeur (pour 12 processeurs)

Conclusions sur la décomposition en temps

Dans cette partie, nous nous sommes intéressés aux méthodes mettant en œuvre une décomposition de l'intervalle d'intégration en sous-domaines.

Une première catégorie de méthodes est celle faisant intervenir une résolution globale sur tous les pas de temps. Nous proposons une parallélisation globale du point fixe et du Newton de ce problème non-linéaire (dépendant de f). Nous montrons la convergence de cette approche pour une EDO linéaire. On aboutit à une système bidiagonal par bloc dont la résolution par LU revient à résoudre la problème séquentiellement. Une résolution de type Krylov (parallèle) permet de gagner en temps lorsque le nombre d'itérations est inférieur au nombre de sous-domaines.

La seconde catégorie est celle des méthodes de tir (comme Parareal et Pita) dans lesquelles deux niveaux de grilles sont présents. Comme nous devons faire face à des classes de problèmes raides, nous avons dû redéfinir la notion de grille fine et grossière en temps de ces méthodes, en définissant la finesse de la grille par rapport à la tolérance relative de l'intégrateur. Nous avons ainsi la possibilité de passer les raideurs, mais la définition du gain par le rapport $\frac{\delta t}{\delta T}$ n'est plus disponible. Ceci nous a permis de valider la méthode sur certains problèmes raides. Cependant pour des problèmes extrêmement raides comme Oregonator, la méthode Pita échoue. Bien que la résolution sur chaque intervalle de temps soit complètement indépendante donc parallèle, le problème sur la correction des sauts commis entre les intervalles est un processus séquentiel qui limite le parallélisme. En effet, avec notre implémentation dans Pita de problèmes raides, bien que la tolérance soit plus grande sur la correction, l'intégrateur résout beaucoup plus difficilement ce problème sur la dérivée linéarisée de la fonction du modèle, rarement connue explicitement et construite par différentiation numérique ce qui n'apparaissait pas dans les papiers originaux de Pita où pour un problème d'EDO linéaire la complexité de la grille fine est la même que celle de la correction.

Nous avons introduit une relaxation numérique de type Jacobi sur les corrections entre les processeurs pour briser cette séquentialité introduite par la relation de Gauss Seidel. Nous avons ensuite analysé les méthodes d'estimation de l'erreur en s'inspirant de l'estimateur de Richardson. Nous montrons dans le cas d'EDO linéaires que l'opérateur

d'extrapolation ne dépend que de l'équation et des points d'intégration et est indépendant des conditions initiales. Ce dernier a pu être appliqué sur des problèmes non-linéaires pour accélérer la convergence en se basant sur le comportement de l'adaptation du pas de temps. Une amélioration de la solution parallèle sur Oregonator a pu être obtenue alors que Pita échoue. Cependant, l'utilisation d'un solveur à ordre variable ne garantit plus le comportement analogue des pas de temps.

Les corrections dans Parareal et Pita sont séquentielles. Nous avons souhaité introduire du parallélisme dans la correction sans relaxation numérique. Nous avons vu la possibilité de parallélisation en sous-domaines de la méthode de correction du résidu. Notre approche nous permet d'avoir un parallélisme à deux niveaux : le premier, la décomposition de l'intervalle de temps en sous-domaine et le second, la mise en cascade des itérations correctives de cette méthode sans altérer les propriétés numériques comparativement à la méthode appliquée séquentiellement. Une validation numérique sur les équations de Navier-Stokes écrites sous forme d'équations différentielles non-linéaires a été réalisée. Des performances par rapport à la correction du résidu sur un processeur ont été obtenues.

Troisième partie

Décomposition en sous-systèmes

Introduction sur la décomposition en sous-systèmes

Dans cette partie nous proposons une parallélisation de la résolution d'un système d'EDO/EDA fondée sur une décomposition de type Schur.

Dans le chapitre 9, nous introduisons la problématique, ou comment résoudre le système non-linéaire du Newton dans l'intégration du problème raide par un complément de Schur.

La difficulté pour introduire à ce niveau la décomposition "automatique" de type Schur est de définir les inconnues interface du système et les inconnues intérieures. Dans le chapitre 10, une solution à cette difficulté est apportée. En se basant sur une analyse a priori des dépendances des sous-modèles entre eux, un masque de dépendance entre les variables est automatiquement construit.

Ce masque présente également un autre avantage, il permet de réduire les temps de calcul de l'évaluation numérique de la matrice jacobienne puisqu'on peut restreindre les évaluations de fonctions aux éléments actifs.

Dans le chapitre 11, le complément de Schur préconditionné est introduit. Son developpement a été fait dans le code du partenaire industriel, ainsi que dans la librairie SUNDIALS. Les résultats obtenus sont retranscrits en fin de chapitre.

Enfin, le dernier chapitre présente un nouvelle approche introduisant le complément de Schur dans les méthodes de type Krylov-Newton sans construction de la matrice.

CHAPITRE 9

Problématique

9.1 Intégrateur

Dans ce chapitre nous considérons :
– soit un système d'équations différentielles ordinaires

$$\begin{cases} y'(t) = f(t, y(t)), \\ y(T_0) = y_0. \end{cases} \tag{1.1}$$

– soit un système d'équations différentielles algébriques

$$\begin{cases} F(t, y(t), y'(t)) = 0, \\ y(T_0) = y_0, \\ y'(T_0) = y'_0. \end{cases} \tag{1.2}$$

Nous nous plaçons dans la situation où le problème est raide. La résolution de ce type de problème se fait par des méthodes Backward Differential Formulae (BDF) implicites. Elles sont communément utilisées dans l'industrie.

Le principe [52] de ce type de schéma est de considérer le polynôme qui interpole les k valeurs précédentes de la solution et tel que la valeur de la dérivée du polynôme au dernier point corresponde à la valeur de la fonction f. Si le dernier point correspond au point précédent, on a une méthode explicite. Si c'est le point en cours la méthode est implicite.

La mise en œuvre des BDF implicites s'intègrent dans un schéma de type *prédicteur-correcteur*. Une méthode explicite est utilisée pour prédire une solution $y_{n(0)}$. Cette prédiction est ensuite corrigée par BDF implicite où un système non-linéaire G_{EDO} (respectivement G_{EDA}) pour les systèmes aux EDO (respectivement EDA) est à résoudre.

$$G_{EDO}(y_n) = y_n - \beta_0 h_n f(t_n, y_n) - \sum_{i>0}^{k} \alpha_{n,i} y_{n-i} = 0 \tag{9.1}$$

$$G_{EDA}(y_n) = F\left(t_n, y_n, h_n^{-1}\sum_{i=0}^{k}\alpha_{n,i}y_{n-i}\right) = 0 \tag{9.2}$$

où β_0 est une constante calculée par le schéma d'intégration, h_n le pas de temps et $\alpha_{n,i}$ sont les paramètres de la méthode utilisée.

La solution y_n est alors calculée de la manière suivante

- La solution prédite $y_n^{(0)}$ est calculée afin d'être utilisée comme condition initiale pour la phase de correction. En notant $\alpha_{n,i}^p$ et β_0^p les paramètres de la formule prédictive).

$$y_n^{(0)} = \sum_{i=1}^{k}\alpha_{n,i}^p y_{n-i} + \beta_0^p h_n f(t_{n-1}, y_{n-1}) \tag{9.3}$$

- S'ensuit la phase de correction qui nécessite une résolution du système non linéaire de type $G(y) = 0$, avec un processus de Newton

$$\begin{cases}\left(\dfrac{\partial G}{\partial y}\left(y_n^{(m)}\right)\right)\delta y_n = -G(y_n^{(m)}) \\ y_n^{(m+1)} = y_n^{(m)} + \delta y_n\end{cases} \tag{9.4}$$

  où la fonction G prend une des deux formes ci-dessous suivant le type de système traité EDO ou EDA

$$\frac{\partial G_{EDO}}{\partial y} = I - \gamma\frac{\partial f}{\partial y} = J_{EDO} \tag{9.5}$$

$$\frac{\partial G_{EDA}}{\partial y} = \frac{\partial F}{\partial y} + \alpha\frac{\partial F}{\partial y'} = J_{EDA} \tag{9.6}$$

  où J est la matrice Jacobienne, $\gamma = \beta_0 h_n$ et $\alpha = \alpha_{n,0}h_n^{-1}$.
- Lorsque la convergence du Newton est atteinte la solution au temps t_n est prise comme $y_n = y_n^{(m+1)}$.

Dans la suite, en fonction du système considéré, la matrice Jacobienne J_{EDO} ou J_{EDA} sera notée J.

9.2 Matrice bloc diagonale bordée

L'approche de la décomposition de domaine de type Schur consiste à mettre la matrice sous une forme spéciale appelée bloc diagonale bordée dont la définition est la suivante :

Définition 9.1 (Matrice bloc diagonale bordée)**.** Une matrice A est dite bloc diagonale bordée si elle s'écrit sous la forme

$$A = \begin{pmatrix} B_1 & & & & F_1 \\ & B_2 & & & F_2 \\ & & \ddots & & \vdots \\ & & & B_p & F_p \\ E_1 & E_2 & \cdots & E_p & C \end{pmatrix} \tag{9.7}$$

Cette structuration du système linéaire $Ax = b$ conduit à la définition de différents types d'inconnues :

$$\begin{pmatrix} B_1 & & & & F_1 \\ & B_2 & & & F_2 \\ & & \ddots & & \vdots \\ & & & B_p & F_p \\ E_1 & E_2 & \cdots & E_p & C \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \\ x_\gamma \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \\ b_\gamma \end{pmatrix} \tag{9.8}$$

Les différents éléments du système linéaire sont interprétés comme ci-dessous.

- les inconnues x_i, $i = 1, \ldots, p$, sont les inconnues intérieures au sous-domaine i.
- les inconnues x_γ sont les inconnues interfaces des sous-domaines. Ce sont celles qui pour un sous-domaine subissent une influence d'un autre sous-domaine.
- les matrices B_i sont les matrices associées à chacun des sous-domaines $i = 1, \ldots, p$.
- la matrice C correspond à la matrice des inconnues interfaces.
- les matrices F_i sont les interactions des inconnues interface sur les inconnues du sous-domaine i.
- les matrices E_i sont les matrices représentant les interactions entre le sous-domaine i et les inconnues interface.

Cette structure de matrice est intéressante car elle permet à partir de la connaissance des valeurs des inconnues interface x_γ de calculer indépendamment, en parallèle, les valeurs x_i des inconnues intérieures des sous domaines i.

Les inconnues intérieures x_i satisfont le système linéaire suivant

$$B_i x_i = b_i - F_i x_\gamma \tag{9.9}$$

Le problème est donc ramené à calculer les inconnues interface. En considérant la dernière équation de (9.8) et en substituant les inconnues intérieures x_i données en fonction de x_γ par (9.9), l'équation sur x_γ s'écrit :

$$\left(C - \sum_{i=1}^{p} E_i B_i^{-1} F_i \right) x_\gamma = b_\gamma - \sum_{i=1}^{p} E_i B_i^{-1} b_i \tag{9.10}$$

La difficulté de cette approche est de mettre les matrices (9.5) (respectivement (9.6)) issues de l'intégrateur sous la forme d'une matrice bloc diagonale bordée. En effet, nous n'avons pas de dépendances de données régulières dans le système différentiel, contrairement à la décomposition de domaine (en espace) d'une EDP où la discrétisation du problème par des méthodes volumes finis ou éléments finis donne un motif régulier de la dépendance des variables. Pour les problématiques de la modélisation 0D de systèmes complexes, les dépendances de données sont irrégulières et la fonction $f(t, y)$ est constituée d'une cascade de modèles physiques qui agissent comme des "boites noires".

Dans la section suivante, nous proposons une démarche pour d'une part extraire automatiquement les dépendances de données entre les variables d'un système différentiel,

ensuite réordonner les variables afin de mettre la matrice sous la forme bloc diagonale bordée. Enfin nous montrons comment implanter la résolution par le complément de Schur dans la méthode d'intégration.

CHAPITRE 10

Décomposition d'un système différentiel

Un système différentiel est souvent constitué par une collection de blocs élémentaires qui reliés entre eux permettent de définir un système dans sa globalité. C'est le point de vue des approches à bases de modèles à port développées dans AMESim, ou pour le cas particulier du signal dans Simulink de MathWorks. Contrairement aux EDP, cette façon de construire un modèle n'induit aucune connaissance a priori entre le couplage des variables.

Nous nous plaçons dans la perspective d'application de nos méthodes sur les modèles à port car c'est la façon dont sont construits les systèmes dans AMESim.

L'assemblage des sous-modèles en systèmes d'équations fournira les informations nécessaires pour construire un masque de dépendance des variables au sens d'une matrice d'adjacence pour les graphes. Puis l'emploi des algorithmes de partitionnement nous conduit à une décomposition du système en sous-systèmes.

10.1 Construction du masque de dépendance

Deux points de vue peuvent être envisagés pour extraire les dépendances entre les variables d'un système différentiel :

1. Soit le système global est construit à partir de blocs élémentaires dont les équations sont connues explicitement.
2. Soit le système global est construit en partie à l'aide de blocs dont on ne connait pas explicitement les équations. La seule information accessible reste les entrées et sorties du sous-modèle. Ceci arrive souvent avec le modèle d'AMESim, dans le cadre de la confidentialité des conceptions, les clients cryptent leurs sous-modèles, ne laissant visible que les informations permettant de les intégrer dans un modèle plus important.

Nous allons étudier le deuxième point de vue car il est plus pertinent dans la perspective d'intégration de notre travail dans AMESim et il englobe le premier point de vue.

Les fonctions des sous-modèles sont donc vues comme des boites noires dont uniquement les entrées et les sorties seront connues.

Par conséquent dans l'exemple suivant

$$\begin{cases} \dot{y_1} = f_1(y_1) \\ \dot{y_2} = f_2(y_1, y_2, y_3) \\ \dot{y_3} = f_3(y_1, y_3) \\ \dot{y_4} = f_4(y_4) \\ \dot{y_5} = f_5(y_4, y_5) \end{cases} \tag{10.1}$$

les fonctions f_i sont des boites noires dont seulement leurs variables d'entrée sont connues. En notant I_{f_i} l'ensemble des indices des variables d'entrée de la fonction f_i, les inconnues y_k, $k \in I_{f_i}$ peuvent avoir une influence sur la variation de la variable y_i. La structure du graphe de dépendance peut être construite directement à partir de cette description.

Remarque 10.1. La vision des fonctions f_i en boîte noire peut introduire des dépendances superflues entre certaines variables. Une variable d'entrée peut ne pas influer réellement sur calcul de la dérivée ce qui définit une dépendance dite pessimiste.

Le graphe de dépendance du système 10.1 est représenté par la figure 10.1.

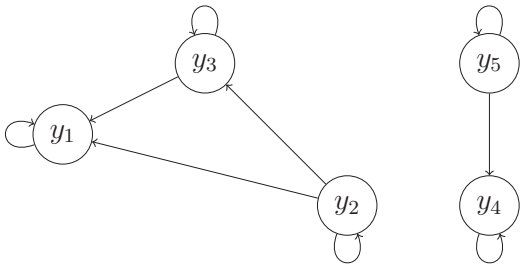


FIGURE 10.1 – Graphe de dépendance de l'exemple

Le graphe est un graphe orienté. La liaison (a, b) entre un nœud a et b signifie que la variable a a une influence dans le calcul de la valeur $\dot{b}$.

Un graphe orienté peut aussi être représenté par sa matrice d'incidence M. La matrice d'incidence est une matrice booléenne comportant un 1 pour l'élément de la ligne a et de la colonne b si la liaison (a, b) existe, 0 sinon.

La matrice d'incidence du système 10.1 est donc

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

L'implémentation de notre décomposition de Schur se base sur une méthode pour construire automatiquement cette matrice d'incidence lors de la construction du système

d'équations. Cette opération est interne à l'outil d'assemblage des blocs représentant les sous-modèles dans le diagramme bloc qui constitue le savoir-faire d'AMESim et n'est donc pas développée ici.

Cette construction fut développée et introduite dans une version de développement par les ingénieurs d'AMESim sur nos conseils. Cette version modifiée nous a été fournie afin que nous puissions développer directement les méthodes de décomposition en sous-systèmes et que nous puissions les évaluer sur des problèmes d'intérêt industriel. Grâce à cette approche, les développements informatiques effectués sont généraux et peuvent aisément être appliqués à n'importe quel système modélisé sous AMESim.

La matrice d'incidence introduite automatiquement lors de la mise en équations du modèle permet à elle seule d'obtenir un gain de temps dans l'évaluation numérique de la matrice jacobienne commme nous le montrons dans la section qui suit.

10.2 Évaluation numérique économique de la matrice Jacobienne

La matrice d'incidence, aussi appelée masque, fournit par construction l'ensemble des valeurs potentiellement non nulles, et donne ainsi une information précieuse dans l'évaluation "économique" de la matrice jacobienne. .

Remarque 10.2. La dérivée d'une inconnue y_i dépend a priori de toutes les inconnues I_{f_i}. Par conséquent l'ensemble des valeurs non nulles de la ligne i de la matrice Jacobienne $J = \partial_y f$ sont donnés par les éléments d'indices I_{f_i}.

Les valeurs identifiées comme étant nulles peuvent donc ne pas être évaluées. Parfois pour certains systèmes le rapport entre le nombre de 1 et le nombre de 0 du masque est très petit. Par exemple pour le problème d'injection le nombre de variables est de 281, le masque n'a que 1731 éléments non nuls sur $281^2 = 78961$.

L'évaluation numérique de la matrice Jacobienne se fait colonne par colonne à partir de la différentiation suivante

$$(Jv)_{\cdot j} \approx \frac{G(u + \epsilon e_j) - G(u)}{\epsilon} \tag{10.2}$$

où j est l'indice de la colonne considérée.

De part l'application du masque, l'évaluation de la colonne peut être restreinte à tous les éléments influant sur la dite colonne, ce sont les éléments d'indices $I_{f_{i|_j}}$, $\forall i = 1, \ldots, d$.

Gains de l'évaluation économique de la jacobienne

L'évaluation économique a été testée sur le problème d'injection à 131 variables d'état.

Le tableau 10.1 condense une vérification effectuée pour s'assurer que numériquement les intégrations par les deux méthodes d'évaluation de la Jacobienne sont identiques. Cette vérification porte sur le pas de temps, l'ordre de la méthode utilisée, le nombre d'évaluations de la fonction f et de la Jacobienne J.

Les statistiques obtenues sont identiques sur le modèle raide d'injection. Le tableau 10.1 permet de déduire que 95% des évaluations de fonctions f ont lieu lors de

pas de temps minimum (log)	-17.7096 s
pas de temps maximum (log)	-5.36592 s
nombre total de pas	89245
nombre d'évaluations de fonction f	5283200
nombre d'évaluations de jacobienne J	38248
nombre de discontinuités	362
nombre total de pas d'Adams	6337
nombre total de pas de BDF	82908

TABLE 10.1 – Statistiques d'exécution pour le problème d'injection à 131 variables d'état

l'évaluation de la matrice jacobienne J. Le temps d'exécution d'un appel à la fonction f est de l'ordre de $0.0026s$. Pour la version économique, les évaluations de fonctions dans la construction de la matrice jacobienne sont de l'ordre de $0.00023 = \frac{(1874 - (5283200 - 131 \times 38248) \times 0.0026)}{5010408}$.

Le tableau 10.2 contient une comparaison de cette version économique de la jacobienne par rapport à la version standard.

temps d'exécution (s)	13723
temps moyen par évaluation de f	0.00264222
temps d'exécution (s) (version économique)	1874
temps moyen par évaluation de f (version économique)	0.000354726
temps d'évaluation de f économique dans J	0.00023
speed-up	7.32

TABLE 10.2 – temps d'exécution et speed-up de l'évaluation économique de la jacobienne pour le problème d'injection à 131 variables

Le speed-up ainsi obtenu est de 7.32. Ce résultat est très encourageant.

Il sera d'autant plus intéressant lorsque le problème considéré sera creux, c'est à dire lorsque le rapport entre les valeurs non nulles et les zéros de la matrice jacobienne sera petit. Un speed-up de 60 a été observé chez un client.

10.3 Partitionnement du modèle en sous-modèle

Une fois le masque (ou la matrice d'incidence) défini, nous sommes en mesure de réordonner les inconnues du système, afin de faire apparaître une matrice bloc diagonale bordée. Nous devons cependant satisfaire la contrainte supplémentaire de minimiser les interactions entre les différents blocs B_i de la matrice . Dans l'optique d'une décomposition en sous-systèmes, cette contrainte exprime le fait d'avoir le plus grand découplage possible entre les sous-systèmes. D'un point de vue d'algorithmes parallèles, elle exprime la réduction des communications lors de l'échange des informations entre les processus gérant chacun un sous-système.

La partitionnement d'un graphe orienté, comme non orienté, soumis à une contrainte est une technique pour le moins classique mais pas forcément triviale. L'algorithme le

plus utilisé est une méthode de partitionnement en voie K (K-way). C'est une méthode de descente et remontée qui fournit un partitionnement optimal dont la contrainte est la minimisation du nombre d'arrêtes coupées par la décomposition du graphe.

Une librairie Métis [1], utilise cette méthode de partitionnement pour les graphes non-orientés. Il faut se reporter à la librairie HMétis pour la version sur les graphes orientés, elle repose sur la définition d'hypergraphe.

La librairie Métis a été utilisée car ses sources étaient disponibles au commencement de ce travail sur la décomposition en sous-systèmes.

Remarque 10.3. Les sources de HMétis ne sont pas disponibles. Leur investigation n'a pas été jusqu'à son terme car nous avons souhaité tester cette approche de décomposition de Schur dans un premier temps. Notre travail ayant un impact dans le développement industriel d'AMEsim les tests de non-régression du logiciel ont du être appliqués pour s'assurer que la méthode est stable sur toutes les modèles d'exemple proposés et qu'elle donne des résultats extrêmement proches du(des) solveur(s) préexistant(s). Dans un second temps seulement, viennent les soucis d'un développement performant. Le parallélisme est intégré dès le début afin de s'assurer du bon comportement de l'algorithme.

L'utilisation de Métis a un impact sur le développement du code. La méthode de partitionnement considère des graphes non-orientés or le masque retourné par le générateur d'équations correspond à la matrice d'incidence (c'est à dire aux liaison d'un graphe orienté). La solution consiste à transformer le graphe orienté en un graphe non-orienté. Ceci revient à dire que si une liaison (a, b) existe dans le graphe orienté alors les liaisons (a, b) et (b, a) existent dans le graphe non-orienté. Dans le cas des graphes non-orientés, les couples (a, b) et (b, a) sont remplacés par l'ensemble $\{a, b\}$.

La matrice d'adjacence M^{adj} n'est autre que la symétrisée de la matrice d'incidence M. Une liaison entre une inconnue a et une inconnue b doit devenir non orientée, c'est dit que ${M^{adj}}_{ab} = {M^{adj}}_{ba} = 1$.

Résultats du partitionnement du masque

L'application de la librairie Métis sur le masque issu du générateur d'équations permet de compartimenter les variables par sous-graphes. Chaque sous-graphe est donc un groupe de variables.

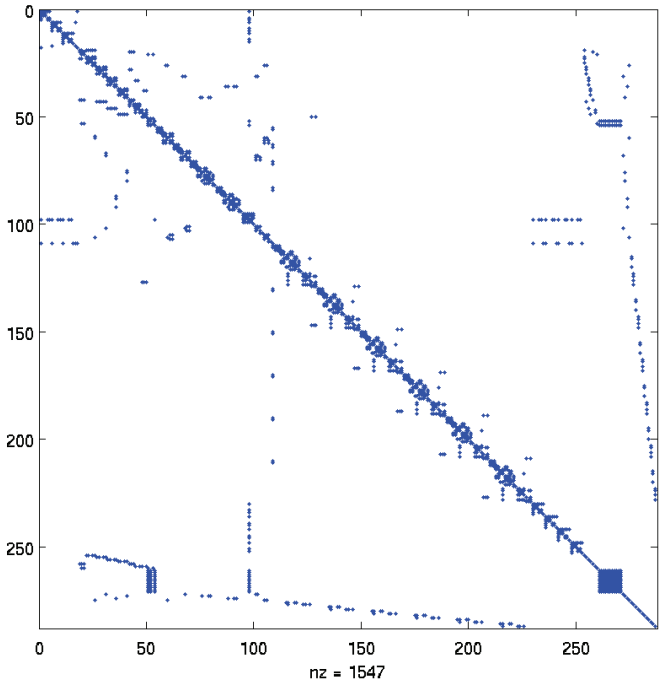
Un post traitement du partitionnement effectué par Métis doit être effectué pour différencier les variables dites internes de celles dites externes suivant qu'elles ont des dépendances ou non avec avec des variables du sous graphe.

Algorithme 20 Partitionnement Métis et post-traitement

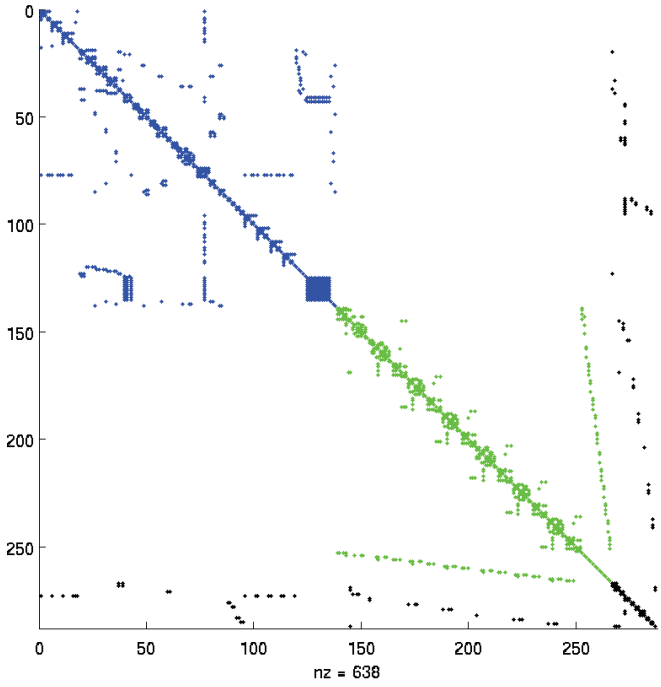
1: Récupération du masque depuis le système d'équation.
2: Pré-traitement du masque : Transcription au format d'entrée de Métis, stockage morse de la matrice.
3: Appel de la méthode METIS_PartGraphKway.
4: Post traitement : Définition de l'appartenance de chaque variable à l'intérieur ou l'extérieur d'un sous graphe.

1. http ://glaros.dtc.umn.edu/gkhome/views/metis

Le résultat de la décomposition permet d'isoler les sous-sytèmes et l'interface (fig 10.2).



(a) masque original



(b) après partitionnement en deux sous-systèmes

FIGURE 10.2 – Exemple : matrices jacobiennes pour un problème d'injection fourni par AMESim

CHAPITRE 11

Complément de Schur

Le complément de Schur permet de résoudre un système en deux étapes. Dans un premier temps, un système linéaire sur les inconnues interface est résolu. Puis les systèmes linéaires locaux peuvent être résolus en parallèle connaissant les inconnues interface. Au passage les systèmes linéaires à résoudre sont de tailles plus petites et donc nécessitent un coût algorithmique plus faible.

Dans cette section nous allons reprendre la décomposition de Schur avec la notation de matrice diagonale bordée pour montrer comment en terme d'implantation le conditionnement est pris en compte. En effet les problèmes que nous devons traiter ont des nombres de conditionnement extrêmement élevés, et le préconditionnement du système interface s'avère nécessaire.

Ainsi si pour un intégrateur quelconque, nous considérons le système linéarisé :

$$Ax = b$$

avec

$$A = \begin{pmatrix} B_1 & & & & F_1 & & & \\ & B_2 & & & & F_2 & & \\ & & \ddots & & & & \ddots & \\ & & & B_p & & & & F_p \\ E_1 & & & & C_1 & E_{12} & \dots & E_{1p} \\ & E_2 & & & E_{21} & C_2 & \ddots & E_{2p} \\ & & \ddots & & \vdots & \ddots & \ddots & \\ & & & E_p & E_{p1} & \dots & E_{pp-1} & C_p \end{pmatrix}$$

alors localement (ie pour chaque processus) le système à résoudre s'écrit (selon le formalisme de [83])

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{voisindei} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}$$

en notant u_i les inconnues intérieures et y_i celles ayant des liaisons avec des inconnues de domaines voisins, f_i et g_i les restrictions respectives de b aux points intérieurs et interface du domaine i. En identifiant la $i^{\text{ième}}$ ligne de la matrice C à

$$C_{|ligne\ i} = (E_{i1} \cdots C_i \cdots E_{ip})$$

et avec $A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix}$

Remarque 11.1. Les E_{ij} sont éventuellement nuls si les domaines i et j ne sont pas couplés.

Ainsi le système local à résoudre est le suivant

$$\begin{cases} u_i = B_i^{-1}(f_i - F_i y_i) \\ y_i + S_i^{-1} \sum E_{ij} y_j = \bar{g}_i = S_i^{-1}(g_i - E_i B_i^{-1} f_i) \end{cases}$$

où S_i est le complément de Schur local.

$$S_i = C_i - E_i B_i^{-1} F_i$$

Globalement le système à l'interface s'écrit

$$\bar{S} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \bar{g}$$

où

$$\bar{S} = \begin{pmatrix} I & S_1^{-1}E_{12} & \cdots & S_1^{-1}E_{1p} \\ S_2^{-1}E_{21} & I & \cdots & S_2^{-1}E_{2p} \\ \vdots & & \ddots & \vdots \\ S_p^{-1}E_{p1} & \cdots & \cdots & I \end{pmatrix} \qquad \bar{g} = \begin{pmatrix} \bar{g_1} \\ \vdots \\ \bar{g_n} \end{pmatrix}$$

Remarque 11.2. Ce système est résolu par une méthode itérative de type Krylov comme l'algorithme de RCG 10. Ce qui nécessite de réaliser le produit matrice vecteur de $\bar{S}$ par un vecteur d.

Remarque 11.3. S_i^{-1} est un calcul local obtenu par décomposition LU (sans pivotage) de la matrice A_i. En effet

$$A_i = L_i U_i = \begin{pmatrix} L_{iB} & 0 \\ E_i U_{iB}^{-}1 & L_{iS} \end{pmatrix} \begin{pmatrix} U_{iB} & L_{iB}^{-1} F_i \\ 0 & U_{iS} \end{pmatrix}$$

ce qui implique que

$$S_i = L_{iS} U_{iS}$$

11.1 Implémentation fine

La méthode présentée ci-dessus est notre seconde approche pour le complément de Schur. La première mise en œuvre, consistait à résoudre

$$\begin{cases} u_i = B_i^{-1}(f_i - F_i y_i) \\ y = S^{-1} \sum P_i^T (g_i - E_i B_i^{-1} f_i) \end{cases}$$

où l'interface y était vue de façon globale et la matrice P_i telle que l'interface locale $y_i = P_i y$.

Cette approche donne de bon résultats pour un problème de diffusion classique dans un carré, mais est sujette à des problème de précision numérique pour les problèmes issus d'AMESim en raison du conditionnement de la matrice $A = I - \gamma J$ bien supérieur à 10^{12} qui nécessite de préconditionner la matrice lorsqu'une méthode de Krylov est utilisée.

Par ailleurs nous ne souhaitons pas construire explicitement la matrice complément de Schur. Ce problème est donc résolu par une méthode de Krylov dont les directions de descentes peuvent être stockées. Nous disposons dès lors de la possibilité d'effectuer l'initialisation de la solution interface du pas de temps suivant par projections sur cet espace de directions de descente, suivant la dynamique de J, c'est à dire si le système évolue peu.

Remarque 11.4 (difficultés). Soulignons que le processus de validation, assez délicat à implémenter, consistant en la comparaison de cette méthode avec la méthode séquentielle utilisée dans AMESim, nous a permis de comprendre et d'expliquer les mauvaises performances numériques de la première implémentation.

Algorithme 21 Code pour l'utilisation de Schur

```
partition = SchurPartition(me, d, p, NULL) ;
schur = SchurAllocMat(me, d, p, partition) ;
schur→pivot = (long int*)malloc(d*sizeof(long int)) ;
for(j = 0 ; j < ni ; j++)
schur→pivot[j] = j ; /* by default no pivoting */
for(j = 0 ; j < ne ; j++)
schur→pivot[ni+j] = j ; /* by default no pivoting at interface */
SchurGETRF(schur, schur→pivot) ;
SchurGETRS(schur, schur→pivot, iteration, b) ;
```

11.2 Résultats

Résultats sur le problème d'injection dans AMESim

L'efficacité parallèle et les temps d'exécution sont donnés dans le tableau 11.2. Ils concernent la résolution du problème d'injection d'un moteur V10. Nous avons augmenté le nombre de partitions, ce qui correspond aux nombres de processeurs sur lequel l'exécution a lieu, de 1 à 4.

#proc	CPU time	speed-up	#Jac	#discont	#steps
1	6845	1	65355	1089	311115
2	4369	1.56	66131	1061	315357
3	1820	3.76	65787	1059	313064
4	1513	4.52	65662	1043	313158

TABLE 11.1 – Speed-up obtenu sur le problème d'injection (n=287)

#proc (p)	1	2	3	4	8	16
# inconnues interface (ne)	0	21	31	47	80	126
ratio interface/intérieur (%)	0	13	26	43	75	92

TABLE 11.2 – Pourcentage du nombre d'inconnues interfaces ne en fonction du nombre de processeurs $\dfrac{ne}{ne + \dfrac{d-ne}{np}}$ pour le problème d'injection ($d = 287$ inconnues).

Le tableau 11.2 montre une efficacité supra-linéaire sur 3 processeurs (de 3.76) en utilisant la décomposition de Schur ce qui est plus important qu'avec la parallélisation à travers la méthode. De plus, ici le nombre de processeurs (qui est égal au nombre de partitions) n'est limité que par les considérations de performances parallèles. C'est à dire le ratio entre les calculs et les communications.

Pour le problème d'injection comportant 287 inconnues, le nombre de partition optimal est 4. Comme on peut le voir dans le tableau 11.2.

Cette limitation provient du ratio entre le nombre d'inconnues interface et la complexité numérique pour la résolution des sous-domaines.

Pour ce cas (le problème d'injection), l'efficacité supra-linéaire peut s'expliquer par deux aspects. Le premier est que le partitionnement permet d'effectuer des décompositions LU de tailles plus petites. Dès que $p \geq 2$, on a pour tout sous-domaine, le nombre d'inconnues locales $ni + ne < d$, pour une décomposition quasi uniforme le nombre d'inconnues $ni + ne$ est uniformément réparti sur les partitions. Ainsi la complexité de la décomposition LU du système devient de l'ordre de $p \times (ni + ne)^3 \approx p \times \left(\dfrac{d}{p}\right)^3 = \dfrac{d^3}{p^2}$. Le second aspect est l'effet de cache, en résolvant des systèmes plus petits pour chaque sous domaine, les données restent d'avantage dans le cache évitant ainsi les échanges entre les différents niveaux de mémoire.

Cependant notre code utilise une décomposition LU sur des matrices au format dense. Nous pensons que l'efficacité attendue sur une décomposition LU en stockage morse ne sera que linéaire.

Résultats dans l'extension Schur dans SUNDIALS

Nous avons introduit le complément de Schur sous forme d'une extension de système linéaire dans la suite SUNDIALS. Sur un des problèmes d'exemple de SUNDIALS, un problème de réaction-diffusion moléculaire sur une surface plane, les performances obtenues sont retranscrites dans le tableau 11.3. La discrétisation de ce problème étant obtenue par

	temps d'exécution	nombre de pas	nombre d'évaluations de fonctions
Sundials avec stockage dense	1h29'	2080	2494
Schur sur 1 processeur	1h34'	2080	2494
Schur sur 2 processeurs	37'	2138	2680
Schur sur 3 processeurs	31'	2212	2791

TABLE 11.3 – Comparaison de l'efficacité de Sundials en stockage dense et de l'implémentation du Schur dans Sundials sur 1,2 et 3 processeurs sur un problème de diffusion-réaction chimique de 7200 inconnues.

la méthode des lignes, nous pouvons augmenter le nombre de points sur la surface ainsi que le nombre d'espèces. Pour une grille de 60×60, et 2 espèces, l'efficacité du schur sur 2 processeurs est très bonne.

Sur 3 processeurs, l'efficacité est moindre. Le temps gagné n'est que de 6 minutes, soit un gain de 19% par rapport au temps sur deux processeurs. Ceci s'explique par le détail de l'implantation de l'extension de Schur dans SUNDIALS. Contrairement aux résultats 11.2, la construction des matrices locales fait intervenir la construction de la jacobienne qui ne prend pas en compte la réduction du coût par l'utilisation du masque comme vu dans la section 10.2.

Ces résultats utilisent le stockage dense de SUNDIALS. À stockage dense équivalent, notre approche avec la décomposition de Schur donne des performances. Cependant, SUNDIALS offre la possibilité d'utiliser des méthodes de Krylov sans matrice dans les méthodes d'intégration (c'est à dire qu'il n'y a pas de construction explicite de la matrice du système linéarisé). Dans ce cas, le temps d'exécution tombe à $10s$, soit 180 fois plus rapide que notre complément de Schur avec stockage dense sur 3 processeurs. C'est pourquoi nous avons cherché à combiner le Schur avec ce genre de technique afin de pouvoir traiter des problèmes plus grands et d'être compétitif.

CHAPITRE 12

Complément de Schur sans matrice, application aux méthodes de Newton-Krylov

L'inconvénient des méthodes directes dans notre approche du complément de Schur est la construction explicite de la matrice jacobienne. Pour les problèmes concrets considérés. La construction de J est de l'ordre de $d+1$ évaluations de fonctions, quant à son stockage il est borné par d^2. Les $d+1$ évaluations de la fonction f accroissent les temps d'exécution de façon non négligeable. Cependant ce temps peut être réduit, dans le contexte des méthodes itératives de résolution de type Krylov où uniquement l'action de la matrice jacobienne sur un vecteur est considérée. C'est cette approche qui définit les méthodes de type Newton-Krylov sans matrice Jacobienne (Jacobian-Free Newton-Krylov) (voir [60] et ses références). Les méthodes considérées combinent donc les méthodes de type Newton pour la résolution des équations non-linéaires avec les méthodes de Krylov pour la résolution du système linéaire sur la correction. Le lien entre les deux est le produit de la matrice jacobienne avec un vecteur. Ce dernier peut être effectué approximativement sans avoir à construire explicitement et à stocker la matrice jacobienne J.

Nous souhaitons dans cette section combiner les avantages de cette méthode en terme de temps d'exécution avec les avantages du complément de Schur pour une résolution parallèle, pour définir une méthode de type *Schur Jacobian-Free Newton-Krylov*.

12.1 Méthodes Jacobian-Free Newton-Krylov

On cherche à résoudre le problème non linéaire (9.1) ou (9.2), considérons que ce problème écrit sous la forme

$$G(u) = 0 \tag{12.1}$$

Les systèmes linéarisés associés sont

$$\begin{aligned} J\delta u^{(k)} &= -G(u^{(k)}) \\ u^{(k+1)} &= u^{(k)} + \delta u^{(k)} \end{aligned} \tag{12.2}$$

Dans les méthodes JFNK, une méthode de Krylov est utilisée pour la résolution du système linéarisé. Soit un résidu initial r_0, étant donnée une valeur initiale δu_0 pour la correction de Newton

$$r_0 = -F(u) - J\delta u^{(0)} \tag{12.3}$$

Remarque 12.1. L'indice k de l'itération non-linéaire a été omis, parce que l'itération de Krylov est faite pour un k fixé. L'indice de l'itération de Krylov sera noté j.

Comme la solution de la méthode itérative est solution de la correction de Newton, et que l'algorithme de Newton fait tendre $G(u)$ vers 0. Il est légitime de prendre $\delta u^{(0)} = 0$.

Les méthodes de Krylov construisent la solution par projection sur des espaces K_j définis par

$$K_j = vect(r_0, Ar_0, \ldots, A^{j-1}r_0) \tag{12.4}$$

où $r_0 = -G(u) - J\delta u^{(0)}$. Elles ne requièrent que l'action du produit de la matrice J sur un vecteur. C'est la clé de leur utilisation dans la méthode de Newton.

Ce dernier produit est approché par

$$Jv \approx \frac{G(u + \epsilon v) - G(u)}{\epsilon} \tag{12.5}$$

où ϵ est une petite perturbation.

12.2 Méthode Schur Jacobian-Free Newton-Krylov

L'algorithme d'une méthode JFNK repose sur le remplacement dans les méthodes itératives du produit matrice-vecteur par l'approximation (12.5).

Le complément de Schur a été introduit sur la matrice J (ou similaire $I - \gamma J...$). Il peut tout aussi bien s'écrire à l'aide de l'équation (12.5).

L'algorithme du complément de Schur reste le même, à ceci près que deux méthodes itératives sont utilisées : une pour l'inversion de la matrice locale B_i et l'autre pour la matrice S. Pour la version préconditionnée une autre méthode itérative devra aussi être mise en place pour l'inversion locale de S_i.

Pour illustrer, considérons la matrice J suivante, correspondant à un réordonnancement en deux partitions.

$$J = \begin{pmatrix} J_{11} & 0 & J_{13} \\ 0 & J_{22} & J_{23} \\ J_{31} & J_{32} & J^1_{33} + J^2_{33} \end{pmatrix} \tag{12.6}$$

Le système linéaire à résoudre s'écrit alors

$$J\delta u = \begin{pmatrix} J_{11} & 0 & J_{13} \\ 0 & J_{22} & J_{23} \\ J_{31} & J_{32} & J^1_{33} + J^2_{33} \end{pmatrix} \begin{pmatrix} \delta u_1 \\ \delta u_2 \\ \delta u_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{12.7}$$

Le système local s'écrit ainsi

$$\begin{cases} J_{ii}\delta u_i = b_i - J_{i3}\delta u_3 \\ \sum_{i=1}^{2}(J_{33}^i - J_{3i}J_{ii}^{-1}J_{i3})\delta u_3 = b_3 - \sum_{i=1}^{2} J_{3i}J_{ii}^{-1}b_i \end{cases} \tag{12.8}$$

Remarque 12.2. Pour un bloc donné i, quatre actions différentes de la matrice J sont à définir :

- $J_{ii}v_i \approx \dfrac{f_{ii}(u_i + \epsilon v_i, u_3) - f_{ii}(u_i, u_3)}{\epsilon}$
- $J_{i3}v_3 \approx \dfrac{f_{i3}(u_i, u_3 + \epsilon v_3) - f_{i3}(u_i, u_3)}{\epsilon}$
- $J_{3i}v_i \approx \dfrac{f_{3i}(u_i + \epsilon v_i, u_3) - f_{3i}(u_i, u_3)}{\epsilon}$
- $J_{33}v_3 \approx \dfrac{f_{33}(u_i, u_3 + \epsilon v_3) - f_{33}(u_i, u_3)}{\epsilon}$

Dans la suite, $\tilde{J}_{..}$ représentera l'approximation de $J_{..}$.

Dans l'application du complément de Schur vu à la section précédente, le point délicat est le calcul du produit matrice vecteur appliqué au vecteur interface v_3

$$\sum_{i=1}^{2}(\tilde{J}_{33}^i - \tilde{J}_{3i}\tilde{J}_{ii}^{-1}\tilde{J}_{i3})v_3 \tag{12.9}$$

Par blocs, les étapes du calcul sont les suivantes

1. $w = \tilde{J}_{i3}v_3$
2. résolution du système linéaire $\tilde{J}_{ii}\hat{w} = w$ réalisée par une méthode itérative dont la base de Krylov pourra être réutilisée pour obtenir la solution des futures évaluations de $\tilde{J}_{ii}^{-1}v_i$.
3. $w = \tilde{J}_{3i}\hat{w}$
4. $w_i = \tilde{J}_{33}v_3 - w$

12.3 Résultats sur le problème de Bratu

La méthode Schur JFNK a été testée sur le problème de Bratu suivant

$$\begin{cases} G(y) = \lambda e^y - \Delta y = 0 \\ y(0) = 0 \\ y(1) = 0 \end{cases} \tag{12.10}$$

La figure 12.1 montre la convergence de l'implantation de l'algorithme en fonction du nombre de processeurs. On remarque que l'introduction du Schur dans la méthode JFNK ne change en rien le profil de convergence du Newton. Il faut cependant souligner que la stagnation de la convergence à 10^{-8} pour les résolutions sur plusieurs processeurs. Elle peut s'expliquer par le fait que la résolution du problème interface est faite à tolérance constante fixée à 10^{-12}.

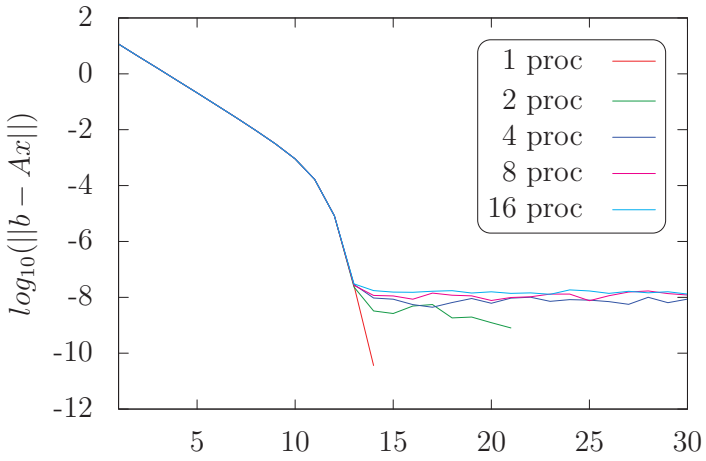


FIGURE 12.1 – Convergence de la méthode de Schur JFNK ($\lambda = 1, d = 1000$)

proc	1	2	4	8
temps (s)	24	14	5	3
efficacité	1	1.71	4.8	8

TABLE 12.1 – Efficacité du Schur Jacobian Free sur Bratu ($\lambda = 1, d = 1000$, 13 itérations)

En ce qui concerne les performances parallèles du prototype, les résultats sont condensés dans le tableau 12.1. Bien que les temps d'exécution soient courts, quelques secondes, l'efficacité est bonne, 8 sur 8 processeurs.

Pour obtenir ces résultats, la résolution du système linéarisé à chaque itération du Newton est résolu par la méthode de Krylov GMRES. Le critère d'arrêt de cette méthode itérative est fixé inconditionnellement à 10^{-12} avec éventuellement un redémarrage. Pour les tests mentionnés ci-dessus, nous avons observés que sur un processeur, sur les premières itérations, il y avait pratiquement un redémarrage à chaque fois. Par contre lorsque l'on augmente le nombre de processeurs, la fréquence des redémarrages diminue. De plus la complexité du GMRES est réduite par p^2.

Conclusion sur la décomposition en sous-systèmes

Cette partie s'est attachée à introduire du parallélisme dans les méthodes de résolution qui ne soit limité que par la taille du problème traité et non par le nombre d'étapes de l'intégrateur numérique. Cette approche repose sur la définition d'un masque lors de la mise en équations du système différentiel, nous permettant d'introduire les dépendances de données non régulières entre les variables. Ceci nous permet de développer un processus automatique de décomposition du système en sous systèmes internes et un système interface qui se prête à l'utilisation de la décomposition de domaine de type Schur sur les systèmes différentiels. Cette approche nous permet des gains substantiels sur l'évaluation numérique de la matrice jacobienne avec des speed-up pouvant atteindre 60 sur un seul processeur.

L'utilisation de méthode du complément de Schur dans le cadre de l'intégration des équations différentielles est nettement plus délicat que pour la résolution d'EDP où les schémas de discrétisation en espace et en temps sont fixés. Dans le cas présent, la structure de la matrice Jacobienne change presque à chaque pas de temps et des parties du système peut devenir inactives et/ou découplées du reste du système. Ce qui défavoris les méthodes directes basées sur une factorisation symbolique des éléments non nuls de la matrice (comme MUMPS).

Le processus de validation et de vérification du bon comportement du solveur est souvent long. La vérification des aspects numériques et le comportement parallèle, nous a amené à définir plusieurs stratégies de résolution des systèmes interface et des problèmes internes.

Ces développements de type décomposition de Schur "automatique" pour la résolution de systèmes linéaires issus d'intégrateurs numériques ont été implémentés sous forme d'une extension dans la suite SUNDIALS. Nous avons obtenu de bonnes performances par rapport au solveur linéaire par défaut à stockage équivalent. Mais par rapport à une méthode de Krylov sans construction explicite de matrice, les méthodes où la matrice jacobienne est stockée sont nettement moins rapides.

C'est pourquoi nous proposons une méthode combinant les avantages du complément de Schur et ceux des méthodes de Krylov sans construction explicite de matrice. D'un point de vue pratique cette combinaison de méthodes permet de résoudre en parallèle

de grands systèmes en utilisant deux niveaux de méthodes de Krylov à matrice libre. Le premier pour l'action de la jacobienne dans le produit matrice vecteur, le second pour le préconditionnement local. Des résultats de performance ont été obtenus pour la résolution du problème non-linéaire de Bratu. Afin de réduire encore le temps d'exécution de l'implantation parallèle, le Newton peut être résolu de façon inexacte. Ceci revient à résoudre le système linéaire à la tolérance de l'itération du Newton en cours [29]. L'introduction du Schur sans matrice dans un Newton et Newton inexacte est en cours dans une version sans matrice du Schur dans une extension de Sundials.

CHAPITRE 13

Conclusions et perspectives

Cette étude a porté sur le développement et l'implantation de méthodes numériques parallèles pour les systèmes différentiels sur des systèmes multiprocesseurs. Deux axes de parallélisation ont été développés au travers de 11 méthodes présentant chacune des avantages et des inconvénients. Nous avons cherchés à les mettre en œuvre pour étudier leur comportement réel. Cette démarche nous à conduit à améliorer certaines d'entre elles ou à innover, amenant des approches originales dans

- la relaxation de type Jacobi dans Pita ;
- la résolution globale de l'ensemble des pas de temps ;
- le calcul pipeliné dans la correction du résidu ;
- l'extrapolation parallèle pour fournir une meilleure condition initiale pour chaque sous-domaine ;
- l'utilisation du Schur dans le contexte des EDO ;
- en particulier l'introduction du Schur sans Matrice.

Le premier niveau de parallélisme se situe dans la méthode d'intégration. Nous avons repris les travaux théoriques de K. Burrage sur la parallélisation à travers les méthodes de Runge-Kutta en mettant en œuvre cette méthode dans le code Radau5. Ceci nous a permis de mettre en avant que le nombre de processeurs utilisables est forcément égal au nombre d'étapes de la méthode, mais aussi que l'efficacité parallèle obtenue n'est pas optimale en raison des communications globales nécessaires au solveur Runge-Kutta. De plus cette étude montre l'efficacité de notre distribution des calculs des colonnes de la matrice jacobienne lors de sa construction. Cette efficacité est bien meilleure que celle de la distribution d'étapes du Runge-Kutta et sa limitation en nombre de processeurs ne dépend que du nombre de variables dans le système afin de garder un ratio correct entre les coûts des évaluations de la fonction et le coût des communications pour rassembler les colonnes.

Cette distribution de la matrice jacobienne nous a conduit naturellement à envisager les techniques de décomposition de domaine. D'un point de vue algébrique le système linéaire a été résolu en parallèle par un complément de Schur. Cette méthode construit

un système interface et des systèmes locaux qui peuvent être résolus indépendamment et donc en parallèle. Contrairement aux EDP pour lesquelles le schéma de discrétisation en espace fournit un couplage régulier entre les variables, les sous-modèles qui composent le système différentiel sont souvent vus comme des boites noires avec des interconnexions non régulières. Nous avons introduit la construction d'un masque de dépendance à partir de la génération des équations du système.

Le masque des dépendances ainsi introduit permet de part sa construction d'identifier les éléments non nuls de la matrice jacobienne et de n'évaluer que ces éléments lors de la différentiation numérique pour le calcul de la matrice jacobienne. Comme dans les systèmes testés, les évaluations de la fonction du modèle sont coûteuses, des gains parfois importants (en séquentiel) ont été obtenus par rapport au code original.

Le second niveau de parallélisme intervient sur le découpage de l'intervalle d'intégration avec les méthodes de décomposition en temps. Les méthodes de type Parareal ou Pita ont été mises sous le formalisme plus général des méthodes de tirs. Une première adaptation de ces méthodes a été de redéfinir la notion de grille fine et grossière en temps pour pouvoir appréhender la classe des problèmes raides, en définissant la finesse de la grille par rapport à la tolérance relative de l'intégrateur. Bien que la résolution sur chaque intervalle de temps soit complètement indépendante donc parallèle, nous avons observé que le problème sur la correction des sauts commis entre les intervalles est un processus séquentiel et qu'il est aussi beaucoup plus difficile à résoudre pour l'intégrateur car il fait intervenir dans Pita le jacobien de la fonction du modèle, rarement connu explicitement et obtenu par différentiation numérique. Nous avons ensuite étudié les méthodes d'estimation de l'erreur en s'inspirant de l'estimateur de Richardson. Ce dernier a pu être appliqué sur la décomposition en temps pour accélérer la convergence en se basant sur le comportement de l'adaptation du pas de temps. La méthode de correction du résidu a été étudiée. Ses propriétés de convergence permettent d'accroître la précision en temps de la solution. Pour tirer parti de la correction itérative de la solution, un processus de calcul parallèle en cascade (calcul pipeliné) a été développé avec succès.

Ce travail a ouvert de nouvelles perspectives de développement : la décomposition de type Schur Matrix Free pour les systèmes différentiels. Des méthodes sans construction de la matrice jacobienne sont plus rapides pour de gros problèmes et elles sont moins gourmandes en terme de mémoire. Le développement de la combinaison de cette technique à matrice libre et du complément de Schur a été introduit dans la thèse. Sa concrétisation dans un code fait l'objet d'une extension de la suite SUNDIALS, dans le cadre du projet ANR PARADE.

En ce qui concerne la décomposition en temps, la voie des méthodes de correction du résidu est intéressante. Le calcul en cascade pose quelques difficultés techniques sur l'équilibrage de la charge de calcul. Dans notre travail nous nous sommes focalisés sur la méthode de type spectral pour ses propriétés de meilleure convergence théorique lorsqu'elle est utilisé avec des méthodes d'ordre un. Une extension de ce travail serait de considérer la forme classique dont l'avantage serait de permettre l'utilisation de n'importe quel solveur adaptatif (en ordre et en pas de temps).

De plus la nouvelle approche qui consiste à considérer les méthodes de tirs globalement avec un Newton parallélisé est prometteuse. Son développement est en cours afin de

confirmer le speed-up théorique.

Bibliographie

[1] W. Auzinger. Defect-based a-posterior error estimation forimplicit odes and daes. Workshop on Innovative Integrators for Differential and Delay Equations, 2006.

[2] G. Bal and Y. Maday. A "parareal" time discretization for non-linear pdes with application to the pricing of an american put. *Lecture Notes in Computational Science and Engineering*, 23 :189–202, 2002.

[3] A. Bellen. Parallel algorithms for initial-value problems for difference and differential equations. *Journal of Computational and Applied Mathematics*, 25(3) :341 – 350, 1989.

[4] A. Bellen, R Vermiglio, and M Zennaro. Parallel ode-solvers with stepsize control. *Journal of Computational and Applied Mathematics*, 31 :277–293, 1990.

[5] Alfredo Bellen. Parallelism across the steps for difference and differential equations.

[6] Alfredo Bellen and Marino Zennaro. Parallel algorithm for intial-value problems for difference and differential equations. *Journal of Computational and applied Mathematics*, 25 :341–350, 1989.

[7] A. Bourlioux, A. T. Layton, and M. L. Minion. High-order multi-implicit spectral deferred correction methods for problems of reactive flow. *J. Comput. Phys.*, 189(2) :651 – 675, 2003.

[8] Liz Bradley. The variational equation. Notes, Departement of Computer Science, University of Colorado, 1999.

[9] K.E. Breman, S.L. Campbell, and L.R. Petzold. Numerical solution of initial-value problems in differential-algebraic equations. 1989.

[10] Claude Brezinski and Michela Redivo Zaglia. *Extrapolation Methods, Theory and Pratice*. Number 2 in Studies in Computational Mathematics. Elsevier Science, 1991.

[11] L. Brugnano and C. Magherini. The bim code for the numerical solution of odes. *J. Comp. and Applied Math.*, 2003.

[12] K. Burrage and H. Suhartanto. Parallel iterated method based on multistep Runge-Kutta of Radau type for stiff problems. *Adv. Comput. Math.*, 7(1-2) :59–77, 1997. Parallel methods for ODEs.

[13] K. Burrage and H. Suhartanto. Parallel iterated methods based on multistep Runge-Kutta methods of Radau type. *Adv. Comput. Math.*, 7(1-2) :37–57, 1997. Parallel methods for ODEs.

[14] Kevin Burrage and Bert Pohl. Implementing an ode code on distributed memory computers. *Computers Math. Applic*, 28 :235–252, 1994.

[15] J. C. Butcher. On the convergence of numerical solutions to ordinary differential equations, October 2007.

[16] J. C. Butcher and P. Chartier. Parallel general linear methods for stiff ordinary differential and differential algebraic equations. *Applied Numerical Mathematics*, 17 :213–222, 1995.

[17] J. C. Butcher, P. Chartier, and Z. Jackiewicz. Nordsieck representation of dimsims. *Numerical Algorithm*, 16 :209–230, 1997.

[18] J.C. Butcher. Implicit Runge-Kutta processes. *Math. Comp.*, 18 :50–64, 1964.

[19] J.C. Butcher. Diagonally-implicit multi-stage integration methods. *Applied Numerical Mathematics*, 11 :347–363, 1993.

[20] J.C. Butcher. General linear methods. *Computers Math. Applic.*, 31(4/5) :105–112, 1996.

[21] Yang Cao, Shengtai Li, Linda Petzold, and Radu Serban. Adjoint sensitivity analysis for differential-algebraic equations : The adjoint dae system and its numerical solution. *SIAM Journal on Scientific Computing*, 24(3) :1076–1089, 2003.

[22] Yang Cao and Linda Petzold. A posteriori error estimation and global error control for ordinary differential equations by the adjoint method. *SIAM Journal on Scientific Computing*, 26(2) :359–374, 2004.

[23] R. P. K. Chan, P. Chartier, and A. Murua. Post-projected Runge-Kutta methods for index-2 differential-algebraic equations. *Applied Numerical Mathematics*, 42 :77–94, 2002.

[24] Chartier and Philippe. A parallel shooting technique for solving dissipative ode's. *Computing*, 51 :209–236, 1993.

[25] P. Chartier. The potential of parallel mutli-value methods for the simulation of large real-life problems. *CWI Quarterly*, 11(1) :7–32, Mars 1998.

[26] Philippe Chartier. *Parallélisme dans la résolution des problèmes de valeur initiale pour les équations différentielles ordinaires et algébriques*. Informatique, Université de Rennes 1 : Institut de Formation Supérieure en Informatique et Communication, juin 1993.

[27] Philippe Chartier and Bernard Philippe. A parallel shooting technique for solving dissipative ode's. *Computing*, 51(3-4) :209–236, 1993.

[28] A. Combescure and A. Gravouil. a time-scale multi-scale algorithm for transient structural nonlinear problems. *Mec. Ind.*, 2 :43–55, 2001.

[29] P. Deuflhard. *Newton Methods for Nonlinear Problems*. Springer Series in Computational Mathematics, 2000.

[30] J. Dongarra, F. Gustavson, and A. Karp. Implementing linear algebra algorithms for dense matrices on a vector pipeline machine. *SIAM Review*, 1984.

[31] J. R. Dormand, R. R. Duckers, and P. J. Prince. Global error estimation with Runge-Kutta methods. *MA Journal of Numerical Analysis*, 4 :169 – 184, 1984.

[32] J. R. Dormand and P. J. Prince. Global error estimation with Runge-Kutta methods II. *IMA Journal of Numerical Analysis*, 5 :481 – 497, 1985.

[33] W.H. Enright. Verifying approximate solutions to differentail equations. *J. Comp. Methods*, 2(3) :1–3, 2003.

[34] Jocelyne Erhel and Stéphanie Rault. Algorithme parallèle pour le calcul d'orbites. *Techniques et Sciences Informatiques*, 19 :649–673, 2000.

[35] C. Farhat and M. Chandersis. Time-decomposed parallel time-integrators : theory and feasability studies for fluid, structure, and fluid structure applications. *Int. J. for Num. Methods in Engineering*, (58) :1397–1434, 2003.

[36] I. Foster and C. Kesselman. Globus : A metacomputing infrastructure toolkit. *Int. J. Supercomputer Applications*, 11(2) :115–128, 1997.

[37] R. Frank and C. W. Ueberhuber. Iterated defect correction for Runge-Kutta methods. *Technical Report*, 14, 1975.

[38] R. Frank and C. W. Ueberhuber. Iterated defect correction for differential equations, Part I : Theoretical results. *Computing*, 20 :207 – 228, 1978.

[39] E. Gabriel, M. Resch, T. Beisel, and R. Keller. Distributed computing in a heterogenous computing environment. *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science*, pages 180–188, 1998.

[40] M. J. Gander and S. Vandewalle. Analysis of the Parareal Time-Parallel Time-Integration Method. 29(2) :556 – 578, 2007.

[41] M. Garbey and D. Tromeur-Dervout. Méthodes Numériques et Couplage de codes pour le Calcul Distribué Distant. *Calculateur Parallèle*, 2002.

[42] C. W. Gear and Xu Xuhai. Parallelism across time in ODEs. *Appl. Numer. Math.*, 11(1-3) :45–68, 1993. Parallel methods for ordinary differential equations (Grado, 1991).

[43] Andreas Griewank and George F. Corliss. *Automatic Differentiation of Algorithms, theory, implementation, and application*. SIAM, 1991.

[44] William Groop, Ewing Lusk, and Anthony Skjellum. *Using MPI*. Scientific and engineering computation. The MIT Press, Cambridge, Massachussetts, second edition, 1999.

[45] David Guibert and D. Tromeur-Dervout. Parallel deferred correction method for cfd problems. In J.H. Kwon, A. Ecer, N. Satofuka, J. Periaux, and P. Fox, editors, *Proc. Int. Conf. Parallel CFD 2006 : Computing and Its Applications*, pages 131–138, Busan city Korea, 2007. Elsevier. ISBN : 978-0-444-53035-6.

[46] David **Guibert** and Damien Tromeur-Dervout. Parallelism results on time domain decomposition for stiff odes systems. In A. Deane et al., editor, *Parallel Computational Fluid Dynamics – Theory and Application*, pages 301–308. Elsevier, 2006.

[47] David **Guibert** and Damien Tromeur-Dervout. Schur ddm for stiff dae/ode systems for complex mechanical system 0d modelling. In *Proc. Int. Conf. on Domain Decomposition Methods*, 2006. submitted.

[48] David **Guibert** and Damien Tromeur-Dervout. Adaptive parareal for system of odes. In O. Widlund and D. Keyes, editors, *Domain Decomposition Methods in Science anf Engineering XVI*, volume 55 of *Lecture Note in Computational Science and Engineering*, pages 587–594. Spinger Verlag, 2007.

[49] David **Guibert** and Damien Tromeur-Dervout. Cyclic distribution of pipelined parallel deferred correction method for ode/dae. 2007.

[50] David **Guibert** and Damien Tromeur-Dervout. Parallel adaptive time domain decomposition for stiff systems of ODEs/DAEs. *Computers & Structures*, 85(9) :553 – 562, 2007.

[51] Michael Günther. Numerical analysis ans simulation of ordinary differential equations. Course winter term 2005/06, 2005.

[52] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving ordinary differential equations. I*, volume 8 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 1993. Nonstiff problems.

[53] E. Hairer and G. Wanner. *Solving ordinary differential equations. II*, volume 14 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 1996. Stiff and differential-algebraic problems.

[54] Anders Christian Hansen and John Strain. Convergence theory for spectral deferred correction. preprint 00, 2005.

[55] Anders Christian Hansen and John Strain. On the order of deferred correction. preprint 00, Kluwer academic Publishers, 04 2005.

[56] P. Henrici. *Discrete Variable Methods in Ordinary Differential Equations*. 1962.

[57] P. Henrici. *Elements of Numerical Analysis*. John Wiley, New York, 1964.

[58] Imagine. The integration algorithms used in amesim. Technical Bulletin n°102.

[59] C.P. Jeannerod and J. Visconti. Global error estimation for index 1 and 2 daes. *Numerical Algorithms*, 1998.

[60] D.A. Knoll and D.E. Keyes. Jacobian-free newton-krylov methods : a survey of approaches and applications. In *Journal of Computational Physics*, volume 193, pages 357–397, 2004.

[61] Jacques-Louis Lions, Yvon Maday, and Gabriel Turinici. Résolution d'EDP par un schéma en temps "pararéel". *C. R. Acad. Sci. Paris Sér. I Math.*, 332(7) :661–668, 2001.

[62] Y. Maday and G. Turicini. A parareal in time procedure for the control of partial differential equations. *C. R. Acad. Sci. Paris*, I(335) :387–392, 2002.

[63] Y. Maday and G. Turicini. The parareal in time iterative solver : a futher direction to parallel implementation. 2003.

[64] Yvon Maday and Gabriel Turinici. A parareal in time procedure for the control of partial differential equations. *C. R. Math. Acad. Sci. Paris*, 335(4) :387–392, 2002.

[65] Yvon Maday and Gabriel Turinici. The parareal in time iterative solver : a further direction to parallel implementation. In *Domain decomposition methods in science and engineering*, volume 40 of *Lect. Notes Comput. Sci. Eng.*, pages 441–448. Springer, Berlin, 2005.

[66] F. Mazzia and F. Iavernaro. Test Set for Initial Value Problem Solvers. Technical report, University of Bari, 2003.

[67] E. Messina, J. J. B. de Swart, and W. A. van der Veen. Parallel iterative linear solvers for multistep runge-kutta methods. *J. Comp. and Applied Math.*, 85 :145–167, 1997.

[68] Metis. http ://glaros.dtc.umn.edu/gkhome/views/metis. Karypis Lab.

[69] M. L. Minion. Semi-Implicit Spectral Deferred Correction Methods For Ordinary Differential Equations. 2003.

[70] Kyoung-Sook Moon, Anders Szepessy, Raúl Tempone, and Georgios E. Zouraris. Convergence rates for adaptive approximation of ordinary differential equations. *Num. Math.*, 96 :99–129, 2003.

[71] Kyoung-Sook Moon, Anders Szepessy, Raúl Tempone, and Georgios E. Zouraris. A variational principle for adaptive approximation of ordinary differential equations. *Num. Math.*, 96 :99–129, 2003.

[72] P. H. Muir, R. N. Pancer, and K. R. Jackson. Pmirkdc : a parallel mono-implicit runge-kutta code with defect control for boundary value odes. *Parallel Computing*, 29 :711–741, 2003.

[73] Tatsuo Noda. The steffensen iteration method for systems of nonlinear equations. *Proc. Japan. Acad.*, 60(Ser. A.) :18–21, 1984.

[74] Tatsuo Noda. The steffensen iteration method for system of nonlinear equations ii. *Proc. Japan Acad.*, 63(Ser. A.) :186–189, 1987.

[75] D. Patterson and J. Hennessy. Computer Architecture a Quantitative Approach. *Morgan Kaufmann*, 1996.

[76] L.R. Petzold. Description of dassl : a differential/algebraic system solver. *Mathematics Computation*, 43(168) :473–482, 1982.

[77] H. Podhaisky, B. A. Schmitt, and R. Weiner. Desing, analysis and testing of some parallel two-step w-methods for stiff systems. *Applied Numerical Mathematics*, 42 :381–395, 2002.

[78] H. Podhaisky, B. A. Schmitt, and R. Weiner. Parallel 'peer' two-step w-methods and their application to mol-systems. *Applied Numerical Mathematics*, 2003.

[79] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*. Number 37 in Tests in Applied Mathematics. Second edition, 2007.

[80] P. S. Rao and G. Mouney. Data communication in parallel block predictor-corrector methods for solving odes. *Parallel Computing*, 23 :1877–1888, 1997.

[81] Stéphanie Rault. *Algorithmes parallèles pour le calcul d'orbites*. Informatique, Université de Rennes : École doctorale Informatique, Traitement du signal et Télécommunications, septembre 1998.

[82] Michel A. Saad. *Compressible fluid flow*. Prentice Hall, Englewood Cliffs, New Jersey 07632, second edition, 1993.

[83] Yousef Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2003.

[84] J. Sand and Stig Skelboe. Stability of backward euler multirate methods and convergence of waveform relaxation. *BIT*, 32, 1992.

[85] R. D. Skeel. a theorical freamework for proving accuracy results for deferred corrections. *Siam J. Num. Anal.*, 19(1) :171–196, 1981.

[86] Robert D. Skeel. Thirteen ways to estimate global error. *Num. Math.*, 48 :1–20, 1986.

[87] S. Skelboe. Parallel algorithms for a direct circuit simulator. In Erik Lindberg, editor, *Proceedings of the 10th European Conference on Circuit Theory and Desing*, Copenhagen, Denmark, 1991.

[88] S. Skelboe. Methods for parallel integration of stiff systems of odes. *BIT*, 32, 1992.

[89] S. Skelboe. Integration of partitioned stiff systems of ordinary differential equations. In *Proceedings of the Third International Workshop on Applied Parallel Computing in Industrial Computation and Optimization (PARA'96)*, volume 1184, pages 621–630, Lyngby, Denmark, 1996. Springer lecture Notes in Computer Science.

[90] S. Skelboe. Accuracy of decoupled implicit integration formulas. *Siam J. Sci. Comput.*, 21(6) :2206–2224, 2000.

[91] S. Skelboe and Z. Zlatev. Exploiting the natural partioning in the numercial solution of ode systems arising in atmospheric chemistry. *Springer Lecture Notes in Computer Science*, 1996.

[92] L. Smarr and C. Catlett. Metacomputing. *Communications of the ACM*, 35(6) :45–52, 1992.

[93] G. A. Staff and E. M. Ronquist. Stability of the parareal algorithm.

[94] Gunnar A. Staff. The parareal algorithm, a survey of present work. Summer 2003.

[95] J. Stoer and R. Burlirsch. *Introduction to Numerical Analysis*, volume 12 of *Tests in Applied Mathematics*. Springer, third edition, 2000.

[96] J. Stoer and R. Burlish. *Introduction to Numerical Analysis*. Text in Applied Mathematics. Springer, third edition, 2002.

[97] J. J. B. De Swart and J. G. Blom. Experiences with sparse matrix solvers in parallel ode software. *Computers Math. Applic.*, 31(9) :43–55, 1996.

[98] M. Valorani and D. A. Goussis. Explicit time-scale splitting algorithm for stiff problems : Auto-ignition of gaseous mixtures behind a steady shock. *J. Comp. Phyiscs*, 169 :44–79, 2001.

[99] P. J. van der Houven, B. P. Sommeijer, and W. A. van der Veen. Parallel iteration across the steps of high-order runge-kutta methods for nonstiff initial value problems. *J. Comp. and Applied Math.*, 60 :309–329, 1995.

[100] P. J. Van der Houwen and B. P. Sommeijer. Parallel iteration of high-order Runge-Kutta methods with stepsize control. *J. Comput. Appl. Math.*, 29(1) :111–127, 1990.

[101] W. A. van der Veen. Step-parallel algorithms for stiff initial value problems. *Computers Math. Applic.*, 30(11) :9–23, 1995.

[102] W. A. van der Veen, J. J. B. de Swart, and P. J. van der Houven. Convergence aspects of step-parallel iteration of runge-kutta methods. *Applied Numerical Mathematics*, 18 :397–411, 1995.

[103] P. E. Zadunaisky. On the estimation of error propagated in the numerical solution of a system of ordinary di erential equations. *Num. Math.*, 27(21), 1976.

Résumé : La simulation numérique de systèmes d'équations différentielles raides ordinaires ou algébriques est devenue partie intégrante dans le processus de conception des systèmes mécaniques à dynamiques complexes. L'objet de ce travail est de développer des méthodes numériques pour réduire les temps de calcul par le parallélisme en suivant deux axes : interne à l'intégrateur numérique, et au niveau de la décomposition de l'intervalle de temps.

Nous montrons l'efficacité limitée au nombre d'étapes de la parallélisation à travers les méthodes de Runge-Kutta et DIMSIM.

Nous développons alors une méthodologie pour appliquer le complément de Schur sur le système linéarisé intervenant dans les intégrateurs par l'introduction d'un masque de dépendance construit automatiquement lors de la mise en équations du modèle. Finalement, nous étendons le complément de Schur aux méthodes de type "Krylov Matrix Free".

La décomposition en temps est d'abord vue par la résolution globale des pas de temps dont nous traitons la parallélisation du solveur non-linéaire (point fixe, Newton-Krylov et accélération de Steffensen). Nous introduisons les méthodes de tirs à deux niveaux, comme Parareal et Pita dont nous redéfinissons les finesses de grilles pour résoudre les problèmes raides pour lesquels leur efficacité parallèle est limitée. Les estimateurs de l'erreur globale, nous permettent de construire une extension parallèle de l'extrapolation de Richardson pour remplacer le premier niveau de calcul. Et nous proposons une parallélisation de la méthode de correction du résidu.

Mots clé : Complément de Schur, Correction du résidu, Décomposition de domaine en temps, Équations différentielles ordinaires algébriques. Jacobian Free Newton-Krylov, Newton-Krylov, Parallélisation, Parareal, Pita, Runge-Kutta,

MSC : 49M27, 65B05, 65L05, 65L06, 65L10, 65L80, 65N10, 65Y05, 65Y20, 68W10.

Analysis of parallel methods for solving stiff ODE and DAE

Abstract : This PhD Thesis deals with the development of parallel numerical methods for solving Ordinary and Algebraic Differential Equations. ODE and DAE are commonly arising when modeling complex dynamical phenomena.

We first show that the parallelization across the method is limited by the number of stages of the RK method or DIMSIM.

We introduce the Schur complement into the linearised linear system of time integrators. An automatic framework is given to build a mask defining the relationships between the variables. Then the Schur complement is coupled with Jacobian Free Newton-Krylov methods.

As time decomposition, global time steps resolutions can be solved by parallel nonlinear solvers (such as fixed point, Newton and Steffensen acceleration). Two steps time decomposition (Parareal, Pita,...) are developed with a new definition of their grids to solved stiff problems. Global error estimates, especially the Richardson extrapolation, are used to compute a good approximation for the second grid. Finally we propose a parallel deferred correction.