HAL
archives-ouvertes.fr

Génération de codes adjoints : Traitement de la trajectoire du modèle direct

Isabelle Charpentier

▶ **To cite this version:**

Isabelle Charpentier. Génération de codes adjoints : Traitement de la trajectoire du modèle direct. RR-3405, INRIA. 1998. <inria-00073285>

HAL Id: inria-00073285
https://hal.inria.fr/inria-00073285

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Génération de codes adjoints : Traitement de la trajectoire du modèle direct

Isabelle Charpentier

N° 3405

Avril 1998

THÈME 4

ISSN 0249-6399

INRIA
RHÔNE-ALPES

Génération de codes adjoints : Traitement de la trajectoire du modèle direct

Isabelle Charpentier

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet Idopt

Rapport de recherche n° 3405 — Avril 1998 — 28 pages

Résumé : Le problème principal de l'exécution des codes adjoints est la taille de la trajectoire créée par le modèle direct et nécessaire à l'évaluation des codes linéarisés. Cette trajectoire, stockée dans des fichiers ou recalculée localement, implique l'utilisation d'une grande quantité de mémoire-disque ou l'utilisation d'un grand nombre de variables locales (mémoire vive). Dans tous les cas, cette construction peut excéder les capacités mémoire de l'ordinateur utilisé. Pour résoudre ce probléme, A. Griewank a proposé d'utiliser des schémas de check-points qui permettent de diviser l'évaluation de l'adjoint en petit problèmes adjoints impliquant la construction locale de petits morceaux de trajectoire.
Dans ce rapport, nous présentons plusieurs algorithmes dont le schéma "Treeverse" de Griewank qui, dans certains cas, n'est pas le schéma optimal. Dans la seconde partie, les schémas de check-points sont utilisés pour calculer des gradients sur le modèle météorologique Meso-NH. Les performances numériques des schémas montrent que la plupart des simulations atmosphériques qui ont un sens physique sont réalisables : grand nombre d'itérations, discrétisation spatiale fine. Calculé de cette manière, l'adjoint est seulement 4/3 plus coûteux qu'un adjoint qui serait calculé en une seule fois.

Mots-clé : Code adjoint, sauvegarde de trajectoires, algorithmes de check-points, Meso-NH.

Remerciements : *L'étude a été financée successivement par le projet IDOPT (INRIA, CNRS, UJF, INPG) et l'action incitative Mode Inverse Opérationnel (INRIA). Le développement de l'adjoint de Meso-NH, réalisé au Laboratoire de Modélisation et de Calcul (UMR 5523, Grenoble), au Laboratoire d'Aérologie (Toulouse) et au CEMRACS (CIRM, Luminy), a bénéficié des moyens informatiques offerts par l'IDRIS.*

Adresse : *I.C., Projet IDOPT, LMC-IMAG, 51 rue des mathématiques, BP 53, F-38041 Grenoble Cedex 9.*

(Abstract: pto)

Generation of adjoint codes: The trajectory problem

Abstract: The main problem of running adjoint codes lies on the size of the trajectory generated by the direct code and which is required to evaluate the linearized codes. This trajectory, which is kept in files or recomputed locally, involves the use of a huge quantity of memory or the use of a large number of local variables. In both cases this may overflow the capacity of user's computer memory. In order to solve the problem, A. Griewank suggests the use of check-points algorithms that allow for dividing the computation of the adjoint into small adjoint sub-problems that imply the construction of small pieces of trajectory.
In this report we discuss about several algorithms including the famous Griewank's scheme named "Treeverse", in particular it appears that "Treeverse" is not always the optimal scheme. In the second part the check-points algorithms are used for computing gradients of the meteorological code Meso-NH. The numerical results show that most of the atmospheric simulations that have a physical meaning (large number of iterations, fine spatial discretisation) can be realized. Computed such a way, the adjoint is only 4/3 coster than an adjoint computed in a straight manner.

Key-words: Adjoint codes, trajectories saves, check-points algorithms, Meso-NH.

Thanks: *This work was supported by the IDOPT project (INRIA, CNRS, UJF, INPG) and the INRIA action for Operative Inverse Mode. The development of the adjoint of Meso-NH, realized in the Laboratoire de Modélisation et de Calcul (UMR 5523, Grenoble), in the Laboratoire d'Aérologie (Toulouse) and during the CEMRACS'97, is supported by the IDRIS computational center.*

Table des matières

1 Introduction

En théorie, le nombre d'opérations requises pour l'évaluation de l'adjoint d'un modèle numérique n'excède pas 5 fois le nombre d'opérations induites par la résolution du modèle lui-même. Cette borne ([17], [8] et [7]), qui est indépendante du modèle traité, indique que l'adjoint est en principe guère plus coûteux en temps que le modèle direct dont il est issu .
Pour les schémas temporels, l'évaluation de l'adjoint s'effectue de manière rétrograde (de T à 0) alors que l'évaluation du modèle direct se réalise dans le sens direct (de 0 à $T > 0$), il est donc nécessaire d'exécuter une fois le modèle direct et de préserver sa trajectoire ou de la recalculer. Quelque soit la méthode de gestion de la trajectoire, on se heurte à des problèmes de coûts (mémoire ou temps).

La trajectoire du modèle direct est construite, puis stockée (fichier ou variables locales) pour permettre l'évaluation de l'adjoint. Si la place mémoire est insuffisante pour conserver toute la trajectoire, l'adjoint ne peut être calculé directement. Basé sur l'idée classique "diviser pour régner", A. Griewank [9] a proposé le concept de calcul des adjoints par morceaux. De manière simplifiée, la méthode consiste à sauver l'état du modèle de temps en temps au cours d'une exécution du modèle direct sans sauvegarde de trajectoire. Ensuite, ces instants particuliers appelés check-points, permettent de reprendre l'exécution du modèle direct qui construit un petit morceau de trajectoire utilisable pour évaluer l'adjoint. Le calcul de l'adjoint complet est ainsi réalisé sur un espace mémoire limité.

L'avantage de ces techniques est de réduire la place mémoire occupée (complexité spatiale) du calcul de l'adjoint au prix d'éventuels recalculs (complexité temporelle). Par exemple, la méthode simplifiée décrite ci-dessus implique une exécution supplémentaire du modèle direct, mais construit une trajectoire qui est localement de taille acceptable. Par la suite, nous verrons que cette méthode à deux niveaux n'est pas toujours applicable.
Obtenir un bon adjoint ayant de faibles coûts en mémoire et en temps, revient à trouver une partition optimale du calcul de l'adjoint. La solution proposée par Griewank repose la loi binomiale et permet de déterminer un bon schéma pour les adjoints construits avec recalculs ; ce sont typiquement les adjoints générés par différentiation automatique. Mais il existe d'autres schémas.

L'objet de ce rapport est d'étudier quelques schémas de check-points d'un point de vue théorique (complexité spatiale et temporelle) tant pour les adjoints contenant des recalculs locaux de trajectoires, que pour les adjoints construits avec sauvegarde de trajectoire sur fichier. Cette différence est soulignée car le modèle direct choisi pour illustrer ce document est basé sur le code météorologique Meso-NH ([11]) dont l'adjoint a été récemment construit [1]. Obtenu par utilisation du logiciel de différentiation automatique O∂yssée, l'adjoint de Meso-NH a été optimisé (en temps) par modifications des recalculs locaux de trajectoire générés par O∂yssée : détection des parties linéaires et implantation d'une sauvegarde de trajectoire sur fichiers. Ces deux arguments réduisent notablement le coût des calculs de gradient, mais induisent l'utilisation d'un espace mémoire important (dépendant de la discrétisation et du nombre d'itérations).

Le rapport est organisé de la manière suivante. Après un bref rappel (chapitre 2 et 3) des méthodes de linéarisation des équations pour mettre en évidence l'importance des modèles adjoints et le problème de la trajectoire, les chapitres 4, 5 et 6 décrivent les principaux algorithmes de placement pour les check-points. On y expose notamment le schéma de Griewank, le schéma de bissection, et les schémas de sauvegarde à deux niveaux ; chaque algorithme y est présenté avec figure et propriétés. Après ces résultats généraux, on rappelle les caractéristiques des codes dérivés du modèle météorologique Meso-NH (chapitre 7), puis les algorithmes de check-points sont testés (chapitre 8) pour des simulations classiques.

2 Assimilation de données variationnelle

Le système d'équations permettant de modéliser le comportement de l'atmosphère (ou de l'océan) est un système d'évolution du premier ordre en temps que l'on peut écrire comme suit. Soit Ω un domaine borné représentant une partie de l'atmosphère, $[0, T]$ l'intervalle de temps étudié et X la variable appartenant à l'ensemble $\mathcal{X}$ des états possibles de l'atmosphère dans le domaine $\Omega \times [0, T]$. Si l'on néglige les conditions aux

limites, le problème d'évolution s'écrit sous la forme simplifiée :

$$\begin{cases} \dfrac{dX}{dt} = F(X) & \text{dans } \Omega \times [0,T], \\ X(0) = X_0 & \text{dans } \Omega, \end{cases} \tag{1}$$

système dans lequel F est un opérateur non linéaire différentiable de $\mathcal{X}$ dans lui-même décrivant la dynamique du modèle, et X_0 est la condition initiale du modèle. Dans ce qui suit, on suppose que ce premier système a une solution unique dans $\Omega \times [0,T]$.

Lorsque l'on souhaite prévoir l'état de l'atmosphère au delà du temps T, il est possible d'introduire dans le modèle des données observées ($X_{obs} \in \mathcal{O}$) pendant la période $[0,T]$ pour identifier la "condition initiale" du modèle. Le modèle, ainsi calé sur l'intervalle [0,T], est alors utilisable pour fournir une prévision.
Les techniques d'assimilation de données variationnelles décrites par F.-X. Le Dimet et O. Talagrand [12] permettent de prendre en compte ces observations afin de retrouver (d'approcher) l'état initial X_0^* du modèle. On choisit une fonction coût J mesurant les écarts entre la solution du système (1) et les observations qui peut être définie par :

$$J(X_0) = \frac{1}{2} \int_0^T \|C.X - X_{obs}\|^2 dt, \tag{2}$$

relation où C est un opérateur linéaire de $\mathcal{X}$ dans $\mathcal{O}$. Ensuite, on cherche X_0^* comme solution du problème de contrôle optimal suivant :

$$\begin{cases} Trouver \;\; X_0^* \in \mathcal{X}_0 \;\; tel \; que \\ J(X_0^*) = \inf_{X_0 \in \mathcal{X}_0} \; J(X_0). \end{cases} \tag{3}$$

Si J est différentiable, une condition nécessaire pour que X_0^* soit solution de (3) est que

$$\nabla J(X_0^*) = 0. \tag{4}$$

Connaissant ∇J, on emploie des méthodes d'optimisation pour résoudre le problème (3). De manière numérique, le gradient de la fonctionnelle peut être obtenu par linéarisation du système d'équations.

2.1 Linéarisation des équations

On différentie le couple d'équations ((1),(2)) par rapport à la variable d'état X. En notant $\bar{X}$ la variable linéaire tangente associée à X, le système (linéaire) tangent s'écrit :

$$\begin{cases} \dfrac{d\bar{X}}{dt} = \left[\dfrac{\partial F}{\partial X}(X)\right] \bar{X} & \text{dans } \Omega \times [0,T], \\ \bar{X}(0) = \bar{X}_0 & \text{dans } \Omega, \\ \nabla J(X_0).\bar{X} = \displaystyle\int_0^T C^*(C.X - X_{obs})\bar{X} dt. \end{cases} \tag{5}$$

Lorsque l'on travaille de manière discrète ($X_0 \in I\!R^n$), obtenir les composantes du gradient de la fonctionnelle J requiert le calcul de la solution du système tangent pour chaque vecteur de la base canonique de $I\!R^n$. La construction de ∇J impliquant n résolutions de (5), la méthode est trop onéreuse pour être utilisée sur des problèmes où n est grand.

On remarque que l'opérateur $\left[\dfrac{\partial F}{\partial X}\right]$ est évalué en X : la solution $\bar{X}$ dépend de l'état de la variable X au long de l'intégration du système d'équations (1). Comme les intégrations des systèmes direct (1) et tangent (5) sont toutes deux effectuées dans le sens direct ($0 \rightarrow T$), les deux systèmes sont résolus simultanément et le système direct (1) fournit à tout instant la trajectoire d'évaluation au système (5) qui n'a pas besoin d'être conservée.

Dans le contexte de la différentiation automatique, cette méthode est appelée mode linéaire tangent.

2.2 Mode adjoint

Une autre manière de calculer le gradient d'une fonctionnelle est de construire le problème adjoint. On montre ainsi [12] que la variable adjointe P, associée à X, est solution du problème :

$$\left\{ \begin{array}{ll} \dfrac{dP}{dt} + \left[\dfrac{\partial F}{\partial X}(X)\right]^* .P = C^*(C.X - X_{obs}) & \text{dans } \Omega \times [0,T] \\ P(T) = 0 & \text{dans } \Omega. \end{array} \right. \tag{6}$$

En résolvant ce système rétrograde [12], on obtient $P(0)$ qui est, au signe près, le gradient de la fonctionnelle J par rapport à la variable d'état X :

$$\nabla J = -P(0). \tag{7}$$

En discret ($X_0 \in I\!R^n$), obtenir le gradient de la fonctionnelle J nécessite donc une seule résolution du système (6). Puisqu'on obtient les composantes en utilisant la base canonique base de $I\!R^n$ dans la formule (7), cette méthode est nettement moins coûteux que le calcul par linéarisation directe.

À une transposition près, on observe que le terme $\left[\dfrac{\partial F}{\partial X}(X)\right]$ apparaît à nouveau dans le système adjoint. La différence essentielle avec la situation précédente est que le système adjoint s'intègre de manière rétrograde ($T \to 0$) alors que le modèle direct s'intègre dans l'autre sens ($0 \to T$). Il est donc nécessaire de calculer la solution du modèle direct sur l'intervalle $[0,T]$ en conservant *les valeurs de X à tout instant* (trajectoire du modèle) pour pouvoir résoudre le système (6).
Numériquement, ce détail a une importance considérable puisqu'il est a priori nécessaire de conserver toute la trajectoire : la gestion de la trajectoire est donc le point crucial de la construction d'un bon adjoint.

3 Fractionnement du calcul de l'adjoint, motivations et notations

3.1 Différentiation automatique et sauvegarde de trajectoire

Les différentiateurs automatiques ont été conçus pour construire les codes dérivés de programmes informatiques tant en mode linéaire tangent qu'en mode adjoint. Pour prendre en compte le calcul de la trajectoire apparaissant dans le système (6), les différentiateurs automatiques produisent des routines qui associent un calcul de trajectoire aux instructions adjointes : elles contiennent ainsi toute l'information (voire trop). Cette méthode par recalculs locaux, adoptée par la plupart des logiciels actuels, fonctionne "bien" pour les codes dont le nombre d'instructions est petit, l'arbre d'appel peu profond et comportant peu d'itérations.
Les schémas itératifs ne satisfont pas toujours à ces conditions puisque la longueur de la trajectoire dépend du nombre d'itérations à effectuer. Parmi ces schémas, on oppose

- les problèmes d'évolution dont coûts d'évaluation (mémoire et temps de calcul consommés) sont prévisibles car le nombre d'itérations est connu,
- aux schémas de point de fixe tels que méthode du gradient conjugué, ou méthode de quasi-Newton. Dans cette seconde classe, les coûts sont liés au nombre, parfois imprévisible, d'itérations permettant d'atteindre le critère d'arrêt. La solution proposée par les différentiateurs automatiques est souvent inefficace et quand cela est faisable, elle cède sa place au profit d'arguments mathématiques simples [4].

Dans ce rapport, nous ne nous intéressons qu'aux problèmes d'évolution.

Lorsque la trajectoire est construite par recalcul, les différentiateurs automatiques ne procèdent pas toujours à l'analyse permettant d'identifier les variables effectivement utilisées dans le calcul de l'adjoint. Cette lacune devient rapidement néfaste aux performances des codes adjoints ainsi générés car le surplus d'information créé mais non utilisé est abusivement conservé. Cette surcharge de l'espace mémoire est conséquente (excès de variables locales) et rend le code adjoint parfois inutilisable (exécutable trop gros). Un exemple illustrant ce problème est présenté dans [1]. Par ailleurs, si le graphe d'appel est profond, on construit plusieurs fois certaines parties de trajectoire.

Pour améliorer les performances en temps de calcul et/ou pour diminuer l'espace mémoire, il est possible d'utiliser les outils d'analyse décrits dans [1] et [2]. Par exemple, on utilise le code linéaire tangent pour détecter les parties linéaires ne nécessitant pas de sauvegarde. On détermine ainsi les valeurs de l'état X effectivement utilisées pour évaluer le terme $\left[\frac{\partial F}{\partial X}(X)\right]$ figurant dans le système tangent (5) i.e. la partie de trajectoire suffisant à l'évaluation de ce terme dans le système adjoint (6).
Il est aussi souhaitable de briser la construction récursive de l'adjoint (morceaux de trajectoires construits plusieurs fois). Pour ce faire, on exécute une fois le modèle direct et on sauve la trajectoire (sur fichier ou en mémoire) ; d'une certaine manière, cela revient à sauver la trajectoire dans des variables globales plutôt que dans des variables locales.
Appliquées à Meso-NH, ces techniques ont permis de produire un adjoint performant : une évaluation de l'adjoint de Meso-NH coûte environ 3 fois une évaluation du code direct (paragraphe 7.3).

L'approche par sauvegarde de trajectoire sur fichier a apporté une réponse au problème de la taille de l'exécutable, mais à reporter le problème sur la quantité de la mémoire requise pour accueillir les très gros fichiers de sauvegarde. Par exemple, la simulation météorologique bidimensionnelle (100 itérations) exposée au chapitre 7 implique la création d'une trajectoire utilisant 5% de la place mémoire autorisée (sur le Cray). Cela implique que, sans autres améliorations, les simulations plus gourmandes en mémoire telles que :

- grand nombre d'itérations,
- utilisation des phases de l'eau (6 variables tridimensionnelles supplémentaires),
- problèmes raides nécessitant le choix d'un pas de temps très petit (bulle convective),
- simulation tridimensionnelle en espace,

sont impossibles à réaliser.
Pour pallier au manque de mémoire, on décide de réintroduire de la récursivité et quelques recalculs.

Comme l'évaluation du code adjoint nécessite au moins une exécution du code direct, Griewank [9] a proposé de sauver l'état de temps en temps en jalonnant l'intervalle $[0, T]$ de "check-points" ; ces instants particuliers permettent de calculer des morceaux de trajectoire en évitant d'incessants retours à l'état initial. Lorsque l'on utilise ce type de stratégie de calcul, le problème fondamental est de trouver le bon schéma de placement des check-points.
Les schémas décrits dans ce rapport sont :

- "Treeverse" qui est le schéma optimal proposé par Griewank pour les adjoints avec recalculs,
- recalcul depuis l'état initial (paragraphe 5.1),
- optimalité en temps lorsque cela est possible (paragraphes 5.2 et 5.3),
- compromis entre temps et mémoire avec le schéma de bissection (paragraphe 5.4).

3.2 Notations

On suppose que le code direct est intégré sur l'intervalle $[0, T]$ par un schéma à P pas de temps. Il va de soit que les codes tangent et adjoint sont intégrés sur le même intervalle de temps et avec la même discrétisation temporelle.
Pour simplifier l'exposé, on considère que le nombre d'opérations effectuées à chaque itération est constant. Il en découle des coûts constants en temps de calcul et en place mémoire. Pour une itération, on les note respectivement :

- C_D^t et C_D^m pour le modèle direct,
- C_T^t et C_T^m pour le modèle tangent (qui contient la sauvegarde de trajectoire sur fichier),
- C_A^t et C_A^m pour l'adjoint (sans calcul de trajectoire),
- C_S^t et C_S^m pour les sauvegardes et lectures complémentaires (check-points).

Le temps de calcul total et la quantité totale de mémoire requis pour évaluer l'adjoint sur l'intervalle de temps $[0, T]$ sont respectivement notés C^T et C^M.

Dans la suite, on suppose que $C_S^t << C_A^t$; cette simplification est possible pour le calcul de l'adjoint du code météorologique Meso-NH sur Cray (paragraphe 7.1). On simplifie également le problème en choisissant $C_S^m = C_T^m = C_A^m$ car il est possible de construire l'adjoint d'un schéma temporel respectant cette contrainte (paragraphe 7.1).

Remarque 1 *Par la suite, la dénomination "modèle tangent" est indifféremment utilisée pour désigner le modèle linéaire tangent et le modèle direct écrivant la trajectoire sur fichier.*

4 "Treeverse" : la solution proposée par A. Griewank

La méthode "Treeverse" proposée par Griewank [9] est le schéma récursif permettant l'évaluation "optimale" des codes adjoints construits avec calcul local de la trajectoire. Nous verrons, sur exemple, que cette méthode se comporte aussi très bien dans le cas des adjoints avec sauvegarde sur fichier.

Appliquant l'adage commun "diviser pour régner", Griewank [9] propose de fractionner le calcul de l'adjoint en utilisant un schéma de placement basé sur une loi binomiale dont les paramètres sont K, le nombre de répertoires dans lesquels est sauvé l'état du système aux check-points, et τ, le nombre d'utilisation de la procédure récursive de calcul à partir d'un même check-point, il est possible de montrer les résultats suivants.

Propriétés du schéma 1

1. *L'adjoint d'un code discrétisé par P pas de temps peut être évalué par un schéma récursif de profondeur τ en utilisant K check-points si et seulement si*

$$P \leq P(K, \tau) = \frac{(K+\tau)!}{K!\tau!}.$$

2. *Le nombre minimal d'itérations en mode direct (sans sauvegarde) nécessaires pour intégrer les P pas de temps avec K check-points est*

$$N_{it} = \tau P - \frac{(K+\tau)!}{(K+1)!(\tau-1)!}, \tag{8}$$

relation où τ est choisi tel que

$$\frac{(K+\tau-1)!}{K!(\tau-1)!} < P \leq \frac{(K+\tau)!}{K!\tau!}.$$

3. *Parmi tous les schémas de check-points satisfaisant les points 1 et 2, il existe un schéma minimisant le nombre de sauvegarde des check-points.*

Des propriétés précédentes, on déduit les coûts temporel et spatial de l'évaluation de l'adjoint :

$$\begin{cases} C^T \simeq (N_{it} + P) * C_D^t + P C_A^t \\ C^M = (C_T^m + C_A^t) + K C_S^m \end{cases} \tag{9}$$

Démonstration 1 *La démonstration de ce théorème est proposée dans [9] et[10], les auteurs y précisent les propriétés concernant le choix du couple (K, τ) "optimal" minimisant le nombre de check-points, et le nombre d'itérations du schéma direct correspondant à ce choix.*

Comme le placement des check-points est dicté par la relation classique de la loi binomiale

$$\begin{aligned} P(K,\tau) &= \frac{(K+\tau)!}{K!\tau!} \\ &= \frac{(K-1+\tau)!}{(K-1)!\tau!} + \frac{(K+\tau-1)!}{K!(\tau-1)!} = P(K-1,\tau) + P(K,\tau-1), \end{aligned}$$

il est naturel d'écrire le schéma de check-points ("Treeverse") sous la forme suivante :

Algorithme 1 *Algorithme binomial avec recalculs locaux*

déterminer la position des check-points
(contacter A. Griewank pour obtenir ses routines)
appeler *Treeverse(1,P,τ, K)*

récursive routine *Treeverse (début, fin, t, k)*
sauver l'état en début
si *(((fin - début)> t) et (k>0))* **alors**
utiliser *le modèle direct jusqu'au check-point "p=début + $P(k,t-1)$"*
appeler *Treeverse (p+1,fin,t,k-1)*
appeler *Treeverse (début,p,t-1,k)*
sinon
boucle *f = fin-1, début, -1*
utiliser *le modèle direct de début à f*
utiliser *le modèle adjoint de f+1 à f*
fboucle
fsi

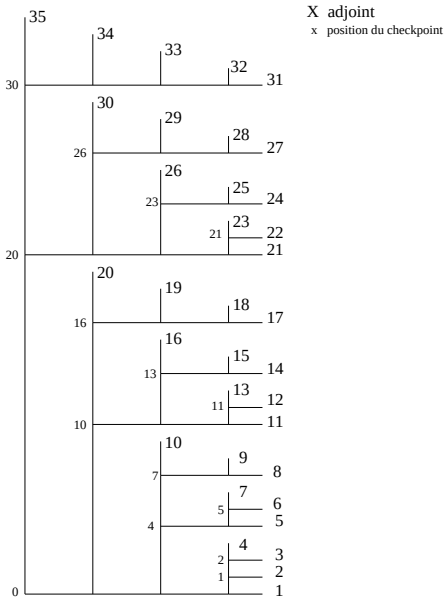


FIG. 1 – *Disposition optimale des check-points pour P=35, K = 3 et τ = 4.*

Remarque 2 *Les résultats énoncés par Griewank ne tiennent pas compte de la place mémoire effectivement disponible puisque c'est le nombre check-points qui est minimisé. Les routines fournies par les auteurs sont aisément modifiables pour optimiser le placement en fonction de la mémoire disponible : on choisit plus de check-points ce qui réduit le nombre d'itérations du direct.*

5 Algorithmes pour les adjoints avec sauvegarde sur fichier

Ce chapitre est dédié aux adjoints construits avec sauvegarde de la trajectoire sur fichier.
Lorsque l'on dispose de la place mémoire suffisante, cette approche, adoptée pour Meso-NH, permet de calculer le gradient d'une fonctionnelle par la méthode adjointe en effectuant une intégration du modèle direct (pendant laquelle on conserve la trajectoire), puis une intégration du modèle adjoint.

A l'instar du paragraphe précédent, on divise le problème physique en K sous problèmes de taille acceptable. Dans un premier temps, on suppose que le nombre de pas de temps P (représentant $[0,T]$) est divisé en K intervalles $[t_{k-1},t_k]$ $(k=1,..,K)$ de longueur inférieure ou égale à τ. On définit les "instants" t_k comme suit :

$$\left\{\begin{array}{ll} 0=t_0<t_1<...<t_{K-1}<t_K=P, & t_k\in IN, \quad \forall k\in IN, \\ t_k=k\tau, & \forall\{k,l\}\in\{1,..,K-1\}, \\ \sum_{k=1}^{K}(t_k-t_{k-1})=P. & \end{array}\right. \tag{10}$$

Les jalons t_k placés, les schémas de calcul de l'adjoint présentés ci-dessous fonctionnent tous avec le même principe : *connaître l'état au pas de temps t_k pour intégrer le modèle l'adjoint de t_{k+1} à t_k+1*. On remarque en particulier que la connaissance de la trajectoire de 0 à t_k devient inutile.

5.1 Recalculs depuis l'état initial

Après une unique sauvegarde de l'état initial, on intègre le modèle direct (sans conserver la trajectoire) de l'état initial jusqu'à un instant t_k donné. Arrivé en t_k, on dispose de l'état qui autorise l'exécution du modèle tangent sur l'intervalle (t_k+1,t_{k+1}) créant une trajectoire utilisable par le modèle adjoint. L'adjoint calculé sur ce morceau, on évalue l'adjoint sur $(0,t_{k-1})$.

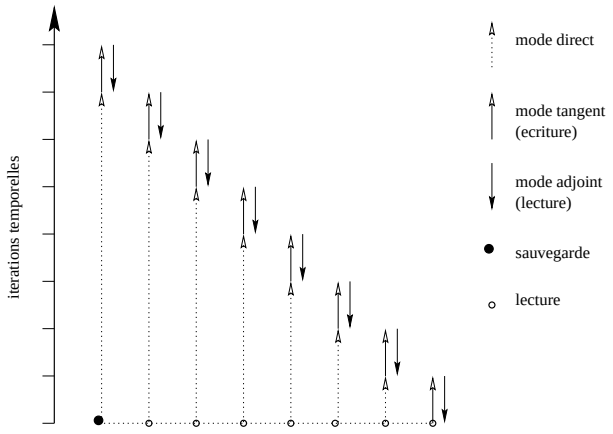


FIG. 2 – *Schéma temporel avec recalculs depuis l'état initial.*

Algorithme 2 *Recalcul depuis l'état initial*

déterminer K en fonction de P et τ
sauver l'état initial
appeler *Schéma2(P,τ,K-1)*

récursive routine *Schéma2 (fin,t,k) ! k guide la récursion*
lire l'état initial
utiliser *le modèle direct de 1 à kt*
utiliser *le modèle tangent de $kt+1$ à $min((k+1)t, fin)$*
utiliser *le modèle adjoint de $min((k+1)t, fin)$ à $kt+1$*
si $k > 0$ **alors**
    appeler *Schéma2 (fin,t,k-1)*
fsi

On a les résultats suivants.

Propriétés du schéma 2 *Soit P le nombre d'itérations à intégrer de manière rétrograde et Q la quantité de mémoire physique disponible pour sauver la trajectoire sur fichier. Le schéma décrit dans l'algorithme est le moins pénalisant en temps des schémas de ce type si et seulement si la taille $\tau \in IN$ est telle que*

$$\tau \leq \frac{Q - C_S^m}{C_T^m}.$$

Dans ce cas, le nombre $K \in IN$ d'appels du schéma récursif est tel que

$$K \geq \frac{P}{\tau}.$$

En choisissant K et τ de cette manière, les coûts de calcul de l'adjoint sont

$$\begin{cases} C^T \simeq \dfrac{K(K-1)}{2}(\tau C_D^t) + PC_T^t + PC_A^t, \\ C^M = \tau C_T^m + C_S^m. \end{cases} \tag{11}$$

Démonstration 2 *Classique.*

Du théorème précédent, on déduit que la complexité temporelle est en $O(P^2)$ car $K\tau = P$ et τ de longueur fixe.

Ce premier algorithme réduit la complexité spatiale du calcul de l'adjoint avec sauvegarde puisqu'il nécessite une place mémoire fixe, mais la complexité temporelle en $O(P^2)$ est mauvaise. En particulier, choisir $\tau = 1$ revient à effectuer un calcul d'adjoint classique privilégiant complètement les recalculs.

Remarque 3 *Les adjoints sans sauvegarde de trajectoire (i.e. $\tau = 1$) sont de même complexité, mais le coût de calcul est plus grand. En effet , on a*

$$C^T = \frac{P(P-1)}{2} C_D^t + PC_T^t + PC_A^t. \tag{12}$$

5.2 Sauvegardes périodiques de l'état direct (τ constant)

Dans ce second schéma, on décide de réduire le temps de calcul en introduisant plus de check-points. Lors du calcul de l'adjoint, on ne souhaite pas réutiliser le code direct hors du calcul des morceaux de trajectoire utiles. Puisque le nombre de pas de temps τ entre deux check-points est constant, on obtient le schéma 3 que l'on écrit algorithmiquement comme suit.

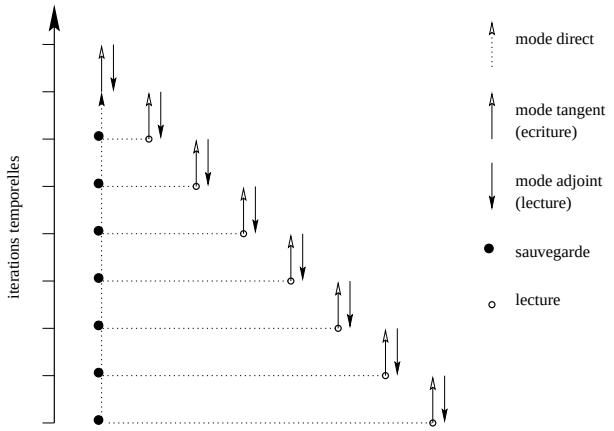


FIG. 3 – *Schéma temporel avec sauvegarde périodique de l'état direct.*

Algorithme 3 *Sauvegarde périodique de l'état direct*

déterminer K en fonction de P et τ
utiliser modèle direct de 0 à P et sauver l'état aux instants $t_k = k\dfrac{P}{\tau}$ $(k \in 0, .., K-2)$
utiliser le modèle tangent de $(K-1)\tau + 1$ à P
utiliser le modèle adjoint de P à $(K-1)\tau + 1$
appeler Schéma3 (τ,K-1)

récursive routine Schéma3 (t, k)
lire l'état en kt
utiliser le modèle tangent de $kt+1$ à $(k+1)t$
utiliser le modèle adjoint de $(k+1)t$ à $kt+1$
si k > 0 **alors**
    appeler Schéma3 (t, k-1)
fsi

Avec le choix τ constant, on montre les résultats suivants.

Propriétés du schéma 3 *Soit P le nombre d'itérations à intégrer de manière rétrograde et Q la quantité de mémoire physique disponible pour sauver la trajectoire sur fichier. Alors l'adjoint peut être intégré s'il existe un couple d'entiers (K, τ) tels que*

$$\tau C_T^m + K C_S^m \leq Q, P \leq K\tau.$$

Les coûts de calcul sont alors

$$\begin{cases} C^T = (K-1)\tau C_D^t + P C_T^t + P C_A^t + K C_S^t \quad (\simeq P(C_D^t + C_T^t + C_A^t) + K C_S^t), \\ C^M = \tau C_T^m + K C_S^m. \end{cases}$$

Démonstration 3 *Classique.*

Si on dispose de la place mémoire suffisante, le schéma (Fig. 3) est optimal en temps puisqu'on intègre le modèle direct :

- 1 fois pour poser les check-points et
- 1 fois pour construire les morceaux de trajectoire.

Toutefois, comme le nombre de check-points $K = P/\tau$ est une fonction linéaire croissante du nombre d'itérations et il ne sera pas toujours possible de sauver tous les check-points en mémoire. Par exemple, prendre $\tau = 1$ implique $K = P$, ce qui revient à sauver toute la trajectoire.

Remarque 4 *Ce même schéma construit avec recalcul entre les check-points n'est guère compétitif. En effet, on a*

$$C^T \simeq K\frac{\tau(\tau-1)}{2}C_D^t + P(C_T^t + C_A^t), \tag{13}$$

on en déduit toujours une complexité en $O(P)$, mais cette fois la distance entre check-points τ intervient de manière forte : on a multiplié le nombre d'itérations N_{it} par environ $\frac{\tau}{2}$.

Ce rapide calcul signifie que le schéma avec sauvegarde sur fichier est, ici, plus performant que le schéma avec recalcul, notamment lorsque la distance entre check-points est grande.

5.3 Régression arithmétique (τ variable)

Par abus de langage, ce schéma est également appelé schéma de sauvegarde périodique optimisée. On peut dès à présent remarquer (figures 3 et 4) que le comportement de ce nouvel algorithme est voisin du précédent, la seule différence résidant dans le placement des check-points : distribution régulière "vs" régression arithmétique.

Comme la place mémoire est limitée, il est important de prendre en compte le nombre de check-points de manière à gérer au mieux l'espace disponible. Suite à cette remarque, on relâche la contrainte τ_k *constant entre les check-points t_k et t_{k+1}*. On cherche les couples (k, τ_k) tels que $\tau_k C_T^m + kC_S^m$ soit proche de Q pour tout $k \in \{1, .., K\}$.

En supposant que $C_T^m = C_S^m$, on cherche donc (k, τ_k) vérifiant

$$\begin{array}{l} \tau_k \geq 1, \quad \forall\ k \in \{1, .., K\} \\ \dfrac{Q - C_T^m}{C_T^m} \leq \tau_k + k = \tau \leq \dfrac{Q}{C_T^m}. \end{array} \tag{14}$$

Les couples $\{(k, \tau_k)\}_{k\in\{0,..,K-1\}}$ satisfaisant ces relations sont

$$\left\{ \begin{array}{ll} \tau_1 = \tau - 1 & \text{pour } k = 1, \\ \tau_2 = \tau - 2 & \text{pour } k = 2, \\ \ldots & \\ \tau_K = \tau - K & \text{pour } k = K. \end{array} \right. \tag{15}$$

Ce choix, présenté dans la figure 4, place les check-points aux instants t_k définis par la formule ci-dessous :

$$\left\{ \begin{array}{ll} t_0 = 0 & \text{si } k = 0, \\ t_k = \displaystyle\sum_{l=1}^{k} \tau_l = \sum_{l=1}^{k} (\tau - l) & \text{si } k > 0. \end{array} \right. \tag{16}$$

Algorithme 4 *Sauvegarde optimisée de l'état direct*

choisir τ *et placer les check-points* t_k $(k \in \{0, .., K-1\})$,
utiliser *modèle direct de 0 à P et sauver l'état aux instants* t_k $(k \in 0, .., K-2)$
utiliser *le modèle tangent de* $t_{K-1}+1$ *à P*
utiliser *le modèle adjoint de P à* $t_{K-1}+1$
appeler *Schéma4 (K-1)*

récursive routine *Schéma4 (k) ! k est l'indice de check-point*
lire l'état en t_k
utiliser *le modèle tangent de* t_k+1 *à* t_{k+1}
utiliser *le modèle adjoint de* t_{k+1} *à* t_k+1
si $k > 0$ **alors**
  appeler *Schéma4 (k-1)*
fsi

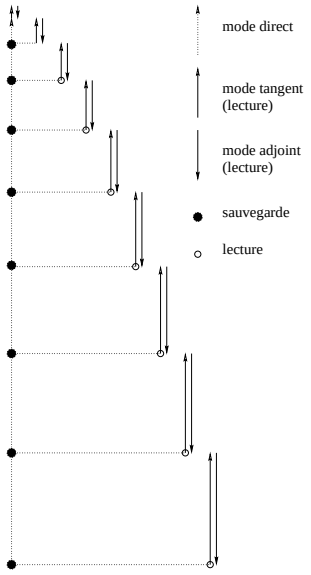


FIG. 4 – *Schéma temporel avec placement par régression arithmétique.*

On a les résultats suivants :

Propriétés du schéma 4 *Soit P le nombre d'itérations temporelles et la quantité Q de mémoire physique disponible pour sauver la trajectoire sur fichier. L'espace mémoire est géré au mieux si les couples d'entiers* $(k, \tau_k)_{k\in\{1,..,K\}}$ *définissant les positions des K check-points sont tels que*

$$\tau_k C_T^m + k C_S^m \leq Q, \quad \forall k \in \{1, .., K\}. \tag{17}$$

En supposant que $C_T^m = C_S^m$, *un adjoint à P pas de temps est intégrable par le schéma de check-points (Alg. 4) s'il existe un* τ *tel que :*

$$P \leq \frac{\tau(\tau-1)}{2}. \tag{18}$$

Dans ce cas, pour τ *donné, le nombre de check-points K est le plus petit entier vérifiant*

$$K \geq \frac{2\tau - 1 - \sqrt{(2\tau-1)^2 - 8P}}{2}, \tag{19}$$

et les coûts de calcul sont

$$\begin{cases} C^T = \left(\sum_{k=1}^{K-1} \tau_k\right) C_D^t + PC_T^t + PC_A^t + KC_S^t, \\ C^M = \tau C_T^m. \end{cases} \tag{20}$$

Démonstration 4

On choisit $\tau \in IN$ en fonction de la quantité de mémoire Q tel que

$$\tau C_T^m < Q. \tag{21}$$

Si $\tau \geq P$ alors on n'a pas besoin de mettre en place un schéma de check-points. Dans le cas contraire, on note $\tau_k = \tau - k$ l'espace entre les check-points t_k et t_{k+1} que l'on place de la manière suivante

$$\begin{cases} t_0 = 0, \\ t_1 = t_0 + \tau - 1 = t_0 + \tau_1 = \tau_1 \\ t_2 = t_1 + \tau - 2 = \tau_1 + \tau_2 = \sum_{l=1}^{2} \tau_l \\ \dots \\ t_k = t_{k-1} + \tau - k = \sum_{l=1}^{k-1} \tau_l + \tau_k = \sum_{l=1}^{k} \tau_l. \end{cases} \tag{22}$$

En choisissant les t_k de cette manière, on s'assure par récurrence que

$$\begin{cases} (\tau_k + k) C_T^m = \tau C_T^m < Q \quad \forall k, \\ t_{k+1} = t_k + \tau - (k+1) = \sum_{l=1}^{k} \tau_l + \tau_{k+1} = \sum_{l=1}^{k+1} \tau_l. \end{cases} \tag{23}$$

Cette construction indique les positions des check-points, mais n'implique pas que l'on puisse intégrer un adjoint à P pas. Pour y parvenir, on ajoute des contraintes de positivité pour τ_k, et on vérifie qu'une fois les check-points placés, il reste de la mémoire pour exécuter au moins un pas de l'adjoint : on impose

$$\begin{cases} \tau_k > 0 \quad \forall k \in \{0, .., K\}, \\ K < \tau, \end{cases} \tag{24}$$

et l'on doit avoir

$$\begin{aligned} P \quad &\leq \sum_{l=1}^{K} \tau_l = \sum_{l=1}^{K} \tau - l \\ &\leq \sum_{l=1}^{\tau-1} (\tau - l) = \sum_{l=1}^{\tau-1} l \\ &\leq \frac{\tau(\tau-1)}{2}. \end{aligned} \tag{25}$$

Le nombre minimum de check-points à utiliser est le plus petit entier K tel que

$$P \leq \sum_{l=1}^{K} \tau_l. \tag{26}$$

En étudiant cette inégalité, on s'aperçoit que le trinôme en K

$$P = \sum_{l=1}^{K} \tau_l = \sum_{l=1}^{K} \tau - l = K\tau - \frac{K(K+1)}{2} \tag{27}$$

i.e.

$$2P - K(2\tau - 1) + K^2 = 0 \tag{28}$$

n'admet une solution que si

$$(2\tau - 1)^2 - 8P \geq 0 \tag{29}$$

ce qui est cohérent avec l'inégalité (27). Dans ce cas, le nombre de check-points K est le plus petit entier tel que

$$\begin{cases} K \leq \tau - 1, \\ K \geq \dfrac{2\tau - 1 - \sqrt{(2\tau-1)^2 - 8P}}{2}. \end{cases} \tag{30}$$

Exemple : Pour intégrer $P = 10$ pas de temps, on remarque (figure 5) les check-points sont placés aux mêmes instant tant par la méthode de régression ($K = 3$) qu'avec le schéma de Griewank ($K = 3$ et $\tau = 2$) .

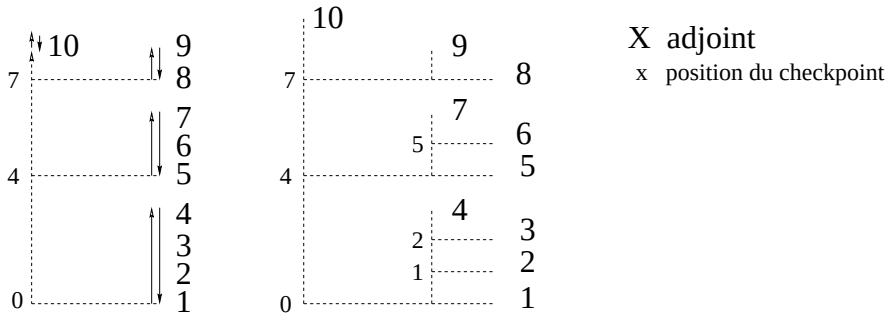


FIG. 5 – *Comparaison entre régression arithmétique et "Treeverse"*

Comme, le premier niveau de check-points est identique dans les deux cas, la différence se limite à la méthode de calcul de l'adjoint : recalcul pour "Treeverse" et sauvegarde sur fichier pour le schéma 4. Cette différence implique des coûts différents :

$$\begin{aligned} N_{it}^{schema4} &= 9 \\ N_{it}^{Treeverse} &= 2 \times 10 - \begin{pmatrix} 3+1 \\ 2-1 \end{pmatrix} = 15. \end{aligned} \tag{31}$$

Pour effectuer une comparaison plus juste, on choisit un schéma à $K = 5$ check-points pour "Treeverse" : le nombre d'itérations du direct diminue, on a

$$N_{it} = 2 \times 10 - \begin{pmatrix} 5+1 \\ 2-1 \end{pmatrix} = 13.$$

En fait, on peut tirer plus de bénéfices de l'algorithme 4 puisque les estimations théoriques ont été réalisées en supposant que la place requise pour sauver un check-point et pour sauver la trajectoire d'une itération ($C_S^m = C_A^m$) sont identiques. Cette hypothèse, qui n'est pas vraie pour tous les codes (paragraphe 7.1), peut être supprimée, ce qui réduit le coût mémoire (ou temporel) du schéma 4 (paragraphe 8.2).
Cette possibilité n'existe pas dans "Treeverse".

5.4 Compromis entre recalculs et sauvegardes

Une alternative classique dans de nombreux domaines (tri, partage de charge, ...) est d'employer une méthode de bissection qui divise un problème de taille P en $log_2(P)$ sous problèmes de tailles égales (ou presque).
Dans notre cas, le critère déterminant la partition est la place mémoire. Pour conserver la trajectoire sur des intervalles de longueur τ, il suffit de diviser le problème de taille P/τ en K sous-problèmes avec

$$K \geq log_2\left(\frac{P}{\tau}\right), \qquad K \in I\!N. \tag{32}$$

L'algorithme de calcul de l'adjoint s'écrit :

Algorithme 5 *Fractionnement récursif par bissection*

appeler Schéma5(1,P,τ,K)

récursive routine Schéma5 (début,fin,t,k)
$milieu = \dfrac{fin - début}{2}$
si (fin-début $> t$) **alors**
    sauver l'état en début
    utiliser le modèle direct de début à milieu
    appeler Schéma5 (milieu+1,fin,t,k)
    lire l'état en début
    appeler Schéma5(début,milieu,t,k)
sinon
    utiliser le modèle tangent de début à fin
    utiliser le modèle adjoint de fin à début
fsi

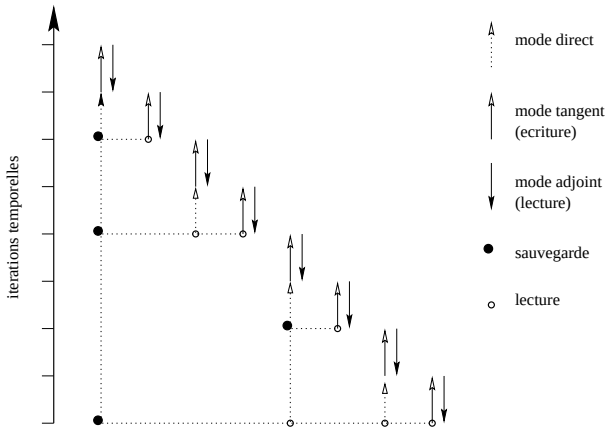


FIG. 6 – *Schéma avec méthode de bissection.*

Les résultats de complexité liés à l'algorithme (Fig. 5.4) sont les suivants.

Propriétés du schéma 5 *Soit P le nombre d'itérations à intégrer de manière rétrograde et Q la quantité de mémoire physique disponible pour sauver la trajectoire sur fichier. Le nombre de check-points K, espacés tous les τ pas, nécessaires pour intégrer un adjoint à P pas est*

$$K = log_2\Big(\frac{P}{\tau}\Big). \tag{33}$$

Si la place mémoire $KC_S^m + \tau C_A^m$ est inférieure à Q, alors l'adjoint peut être évalué aux coûts suivants

$$C^T \simeq \frac{KP}{2}C_D^t + PC_T^t + PC_A^t,$$
$$C^M = (K+\tau)C_T^m.$$

Démonstration 5 *Classique*

5.5 Propriétés des différents schémas lorsque $C_S^m = C_A^m$

Les résultats obtenus dans les paragraphes précédents sont résumés dans les deux tables suivantes.

TAB. 1 – *Caractéristiques des schémas de check-points en fonction P et τ*

Alg.	Schéma	Nombre de check-points K	N_{it}	Mémoire
1	Treeverse	(K,τ) donnés par (8)	$\tau P - \begin{pmatrix} K+1 \\ \tau-1 \end{pmatrix}$	K
2	recalcul de 0	1	$\frac{1}{2}\frac{P}{\tau}\left(\frac{P}{\tau}-1\right)$	$\tau+1$
3	sauv. per.	$\left(\frac{P-\tau}{\tau}\right)$	$P-\tau$	$\tau+K$
4	rég. arithm.	$\simeq \frac{2\tau-1-\sqrt{(1-\tau^2)-8P}}{2}$ (si $P \leq \frac{\tau(\tau-1)}{2}$)	$\left(P-(\tau-K)\right)$	τ
5	bissection	$log_2\left(\frac{P}{\tau}\right)$	$\left(\frac{KP}{2}\right)$	$\tau+K$

TAB. 2 – *Complexités des schémas (à la louche)*

Alg.	Schéma	Complexité temporelle	Complexité spatiale
1	Treeverse	$O(Plog(P))$	$O(log(P))$
2	recalcul de 0	$O(P^2)$	$O(1)$
3	sauv. per.	$O(P)$	$O(P)$
3	rég. arithm.	$O(P)$ (si possible)	$O(\sqrt{P})$
5	bissection	$O(Plog(P))$	$O(log(P))$

De ces tableaux, on déduit que :

1. le schéma optimisé par régression arithmétique est le plus performant en temps lorsque son emploi est possible. Ceci est essentiellement dû au choix de la méthode de calcul pour l'adjoint (sauvegarde contre recalculs).
2. "Treeverse" et le schéma de bissection sont de même complexité : ils ne différent l'un de l'autre que d'un facteur multiplicatif.
3. Présentées ainsi, "Treeverse" occupe beaucoup moins de mémoire, mais nécessite plus d'itérations que le schéma de bissection. Toutefois, à mémoire identique, "Treeverse" est meilleure que le schéma de bissection.

Ces observations sont numériquement confirmées dans le chapitre 7.

5.6 Autres applications des schémas de check-points

On peut citer deux applications importantes des schémas :

- le debbugage : il n'est pas nécessaire de refaire tous les calculs précédant l'erreur, et
- le calcul parallèle.

En effet, un programme séquentiel peut s'écrire sous la forme d'une composition de fonctions élémentaires, ou de fonctions composées (pas de temps par exemple) $f_i : IR^{n_{i-1}} \to IR^{n_i}$ $(i = 1, .., P)$ i.e.

$$\begin{array}{rccc} F: & IR^{n_0} & \longrightarrow & IR^{n_P} \\ & x & \longrightarrow & y = F(x) = f_P o..of_2 of_1(x) \end{array} \tag{34}$$

Dans ce cas, l'opérateur linéaire adjoint associé à F est

$$\left[\frac{\partial F}{\partial x}(x)\right]^* : IR^{n_P} \longrightarrow IR^{n_0}$$
$$\delta y \longrightarrow \delta x = \left[\frac{\partial f_1}{\partial x}(x)\right]^* \times \left[\frac{\partial f_2}{\partial x}(f_1(x))\right]^* \times \ldots \times \left[\frac{\partial f_P}{\partial x}(f_{P-1} o \ldots o f_2 o f_1)(x)\right]^* \times \delta y. \quad (35)$$

L'utilisation de calcul parallèle est immédiate puisqu'on peut calculer chacun des morceaux séparément et effectuer des produits entre matrices jacobiennes et vecteurs.

6 Choix de la méthode de check-point pour le calcul de l'adjoint lorsque $C_S^m = C_A^m$

Lors d'une simulation, quantité de mémoire et temps de calcul n'ont pas le même rôle. En effet, la mémoire est une quantité physique dépendant de la machine (parfois difficilement extensible) alors que le temps de calcul peut paraître plus "humain" puisqu'il dépend de la patience de l'utilisateur (ce n'est plus tout à fait vrai lorsqu'on utilise des moyens de calculs collectifs). En conséquence, **l'utilisateur demeure le seul à pouvoir gérer au mieux son espace mémoire et son temps de calcul**.

Les tableaux et figures présentés précisent les coûts de calcul des méthodes de check-point dans des situations particulières. Faciles à construire, ce sont de bons outils pour guider l'utilisateur vers l'une ou l'autre configuration.

6.1 Nombre de check-points K

Ce tableau indique le nombre de check-points (ou le couple (K, τ) pour "Treeverse") utilisés dans cet exemple.

TAB. 3 – *Nombre de check-points K en fonction de P ($\tau = 100$)*

Alg.	Schéma	P =500	1 000	2 000	4 000	8 000	16 000
1	Treeverse*	(6,6)	(7,6)	(7,7)	(8,7)	(8,8)	(9,8)
1'	≃Treeverse**	(103,2)	(104,2)	(102,2)	(106,2)	(107,3)	(108,3)
2	recalcul de 0	1	1	1	1	1	1
3	sauv. per.	4	9	19	39	79	159
4	reg. arithm.	5	10	22	55	—	—
5	bissection	3	4	5	6	7	8

* Les calculs avec "Treeverse" ont été réalisés à l'aide de la procédure de placement optimal des check-points gracieusement prêtée par ses auteurs.

** Ces calculs ont été réalisés à l'aide de "Treeverse", mais en utilisant la même mémoire (sur disque) que pour le schéma de bissection.

6.2 Nombre d'itérations N_{it} du direct

Dans le calcul de l'adjoint, l'intégration adjointe proprement dite n'est réalisée qu'une seule fois : la différence de temps de calcul réside dans le temps passé à utiliser le modèle direct pour passer d'un check-point à l'autre, ou d'un check-point à l'instant précédant un calcul en mode adjoint. Comme ce temps est proportionnel au nombre d'itérations, il est facile d'extraire de ce tableaux les schémas les plus performants.

TAB. 4 – *Nombre d'itérations du modèle direct en fonction de P ($\tau = 100$)*

Alg.	Schéma	P =500	1 000	2 000	4 000	8 000	16 000
1	Treeverse	2 208	4 713	10 997	22 995	52 560	108 552
1'	$\simeq$ Treeverse	897	1 895	3 894	7 893	18 005	41 895
2	recalcul	1 000	4 500	19 000	78 000	316 000	1 272 000
3	sauv. per.	400	900	1 900	3 900	7 900	15 900
4	reg. arithm.	405	910	1 922	3 955	—	—
5	bissection	750	2 000	5 000	12 000	28 000	64 000

Le coût temporel de chaque itération étant supposé constant, le temps de calcul nécessaire pour intégrer une fois le modèle adjoint se déduit de la formule

$$C^T = N_{it} C_D^t + P(C_T^t + C_A^t).$$

Ce tableau montre, qu'à mémoire identique, "Treeverse" modifié est moins coûteux que le schéma de bissection. On peut aussi observer le rapport, d'environ 2, entre le nombre d'itérations nécessaires pour "Treeverse" modifiée et le schéma de régression arithmétique ce qui met en évidence l'importance du choix la méthode de calcul de l'adjoint: recalculs ou sauvegarde.

6.3 Quantité de mémoire utilisée

Pour simplifier, on suppose que la quantité de mémoire utilisée pour la sauvegarde d'un check-point est identique à celle utilisée pour sauver la trajectoire du modèle durant une itération.

TAB. 5 – *Quantité de mémoire utilisée (unité arbitraire) en fonction de P ($\tau = 100$)*

Alg.	Schéma	P =500	1 000	2 000	4 000	8 000	16 000
1	Treeverse	6	7	7	8	8	9
1'	$\simeq$ Treeverse	103	104	105	106	107	108
2	recalcul de 0	101	101	101	101	101	101
3	sauv. per.	104	109	119	139	179	259
4	reg. arithm.	100	100	100	100	—	—
5	bissection	103	104	105	106	107	108

La quantité de mémoire effective s'obtient en multipliant ces nombres par la quantité de mémoire requise (C_A^m) pour stocker la trajectoire créée lors d'une itération.

6.4 Premières conclusions sur les schémas de check-points

Le choix du schéma repose essentiellement sur la méthode de construction de l'adjoint, le nombre d'itérations et la place mémoire disponible.

Si le critère est le temps d'exécution, alors

- les adjoints classiques avec recalcul sont mieux appréhendés avec le schéma binomial présenté par Griewank.
- Par contre, les adjoints avec sauvegarde sur fichier et espace mémoire suffisant sont plus performants lorsque l'on peut utiliser le schéma de régression arithmétique, essentiellement parce que l'on effectue pas de recalculs.

Dans le chapitre suivant, nous allons confirmer ces résultats par des tests numériques menés sur le modèle météorologique Meso-NH$^*_{adiab}$, puis approfondir la question des places mémoires respectivement nécessaires pour sauver check-points et morceaux de trajectoires.

7 Meso-NH$^*_{adiab}$, modèle adiabatique dérivé

Le modèle atmosphérique Meso-NH [11] est développé conjointement à Toulouse par le Centre National de Recherches Météorologiques (Météo-France) et le Laboratoire d'Aérologie (CNRS). Ce modèle est complètement décrit dans la documentation scientifique [14], algorithmique [15] et le manuel d'utilisation [16].

7.1 Leap-Frog de Meso-NH et sauvegarde de trajectoire

Les principales étapes de la résolution de la partie adiabatique du modèle sont :

model$n

Leap-Frog	(boucle temporelle)
$\rightarrow$	mise en place des conditions aux limites,
$\rightarrow$	advection des champs de vent,
	et des variables scalaires (température, ...),
$\rightarrow$	calcul de la force de Coriolis, de la flottabilité, ...,
$\rightarrow$	diffusion numérique (∇^4),
$\rightarrow$	résolution de l'équation de la pression,
$\rightarrow$	incrémentations temporelles.

Les équations correspondantes sont exposées dans [11].

Le schéma temporel (Leap-Frog) calcule la valeur des variables (XR) au temps $t+\delta t$ en fonction des variables (XM) à l'instant $t-\delta t$, et des variables XT à l'instant t, on a :

$$XR = \delta t F(XT) + XM,$$

relation où F est l'opérateur non linéaire (supposé différentiable) décrivant la dynamique du modèle. Dans cette équation, XM n'intervient que de façon linéaire.
A la fin de chaque pas, on échange les valeurs (étapes d'incrémentation temporelle), on a de manière simplifiée

$$\begin{array}{rcl} XM & \longleftarrow & XT, \\ XT & \longleftarrow & XR. \end{array} \tag{36}$$

Pour reprendre un calcul à partir d'un check-point, on a besoin des ensembles alors que, sur la simulation de référence (chapitre 7), il est apparaît que seule la connaissance de XT est nécessaire pour évaluer l'adjoint car seules ces variables ont un rôle non linéaire, les variables de l'ensemble XM n'interviennent pas (ce n'est pas toujours vrai).

Les remarques précédentes signifient que, pour cette simulation, le rapport entre $C^m_S = \{XM, XT\}$ (place mémoire pour conserver un check-point) et $C^m_A = \{XT\}$ (place mémoire pour conserver les éléments de trajectoire) est d'environ 2. Plus précisément, un check-point nécessite de conserver les variables tridimensionnelles $XM = \{XUM,\ XWM,\ XTHM,\ XPHIM\}$ et $XT = \{XUT,\ XWT,\ XTHT\}$ tandis que l'exécution d'un pas d'adjoint utilise seulement XT : le rapport C^m_S/C^m_A est de 7/3.

Les tableaux présentés au chapitre 6 ne prennant pas en compte la différence de coût au niveau de la mémoire, des résultats complémentaires sont proposés dans le paragraphe 8.2

Remarque : L'approximation anélastique est responsable de l'absence de $XPHIT$ dans l'ensemble XT.

7.2 Caractéristiques de la simulation de référence

Pour illustrer les possibles utilisations des codes dérivés, nous avons choisi l'étude d'un écoulement hydrostatique non-linéaire sur relief idéalisé [11], [18]. Ce type d'expérience simple, qui peut se faire avec la partie adiabatique de Meso-NH, permet notamment de tester des problèmes type de sensibilité au réglage de paramètres, ce point sera détaillé dans une prochaine étude [3].

Une montagne bidimensionnelle de demi largeur $a = 10\ km$ et de hauteur maximum $h_{max} = 500\ m$ est définie par

$$h(x) = \frac{h_{max}}{1 + (x/a)^2}.$$

Le flux imposé en amont de la montagne est homogène verticalement $U = 10\ m.s^{-1}$ pour la vitesse du vent et $N = 10^{-2}\ s^{-1}$ pour la stabilité de l'atmosphère.

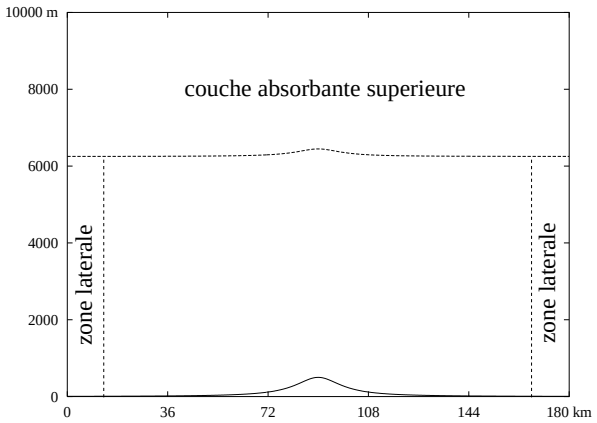
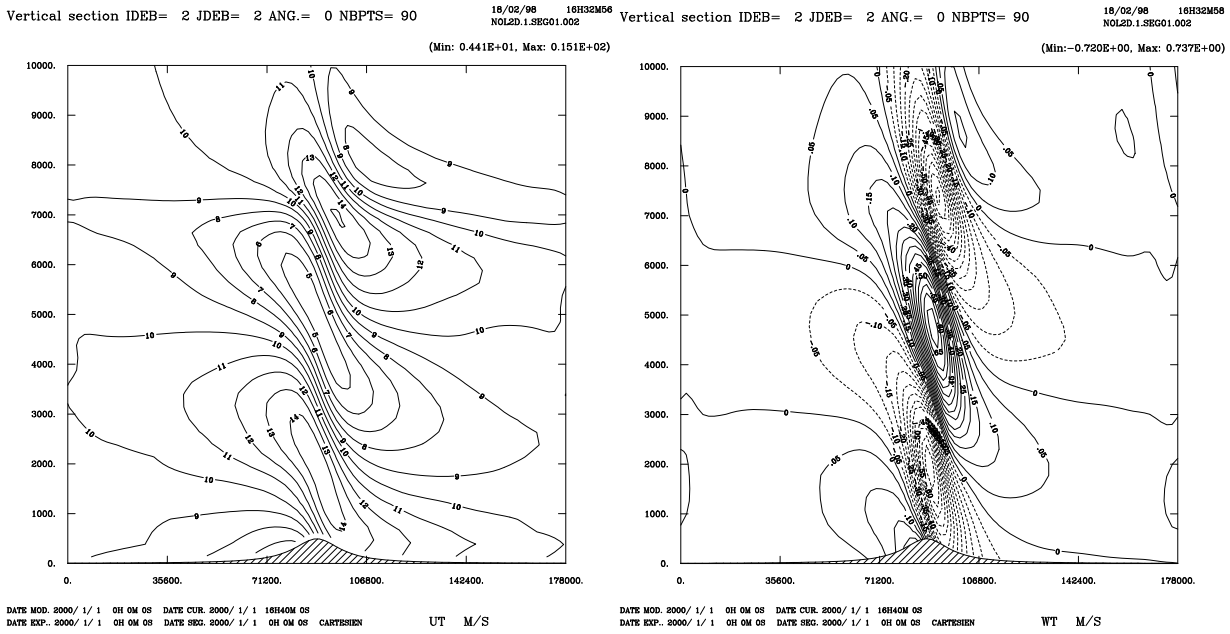


FIG. 7 – *Domaine de calcul et couche absorbante.*

Pour cette simulation, on emploie des conditions aux limites latérales périodiques avec une petite zone absorbante sur les bords latéraux pour amortir la perturbation initiale qui se propage vers le bord aval du domaine. Une zone absorbante est également définie au sommet du domaine pour éviter la réflexion de la composante verticale des ondes de gravité.

Du point de vue numérique, on utilise une grille de discrétisation de 90×60 dans les directions horizontales ($\Delta x = 2\ km$) et verticales ($\Delta z = 250\ m$). Le modèle est intégré avec un pas de temps de $\Delta t = 20\ s$ pendant 6×10^4 secondes. Cette durée correspond au temps adimensionnel $t^* = t\ U/a = 60$ qui permet d'atteindre un début de stationnarisation des champs produits par la simulation. Les champs de vents U et W et la température potentielle θ à $t^* = 60$ sont reproduits sur la figure 8.



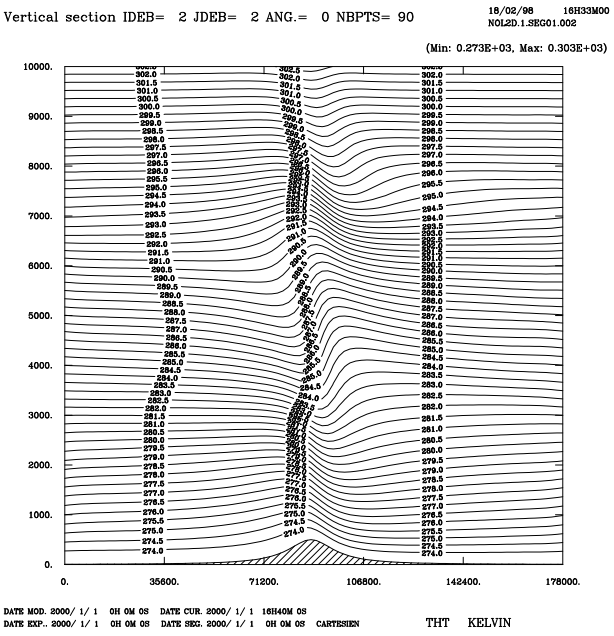


FIG. 8 – *Champs de U, W (en haut) et de température potentielle θ (en bas) à $t^\star = 60$.*

7.3 Coûts des codes dérivés exécutés sans check-points

Pour évaluer le coût temporel des codes dérivés, on intègre Meso-NH$^*_{adiab}$ sur 100 pas (simulation de 2000 secondes) qui ne nécessite pas de procédure de check-points. Les résultats, routine par routine, sont présentés dans le tableau 6 ; ils diffèrent de ceux annoncés dans l'article [2] car la trajectoire a été un peu plus optimisée : introduction de recalcul dans la routine advection, prise en compte de la nature de la simulation pour réduire le nombre des variables à sauver.

Le temps nécessaire à la sauvegarde d'une variable (tableau 3D) sur fichier est de $7.5.10^{-4}$ secondes. Comme on a supprimé des écritures dans la routine tangente de l'advection, on diminue le temps passé dans cette routine : 5.863 s. à la place de 6.082 s. Les variables nécessaires à la routine adjoint de l'advection n'étant plus disponibles sur fichier, on les recalcule. On passe ainsi de 7.697 s. (advection avec sauvegarde sur fichier) à 8.243 s. (advection avec recalculs). Le calcul de l'adjoint avec sauvegarde sur fichier utilise 13.779 s et sauve 300 tableaux alors que l'adjoint avec recalcul utilise seulement 14.100 s (pas de sauvegarde). Le choix d'effectuer quelques recalculs se justifie car le gain en mémoire est substantiel alors que la perte en temps est petite.

Les temps totaux d'évaluation permettent de calculer le surcoût d'évaluation du tangent et de l'adjoint par rapport au temps de calcul de l'adjoint, on obtient :

$$\frac{C_T^T}{C_D^T} = 1.99 \quad \text{(pour une évaluation du tangent)}$$

et

$$\frac{C_A^T}{C_D^T} = 2.02 \quad \text{(intégration rétrograde de l'adjoint).}$$

On note que ces rapports sont inférieurs aux bornes théoriques ([17], [8] et [7]) qui sont respectivement de $4n$ (où n est le nombre d'évaluation du tangent) pour le code linéaire tangent, et de 5 pour l'adjoint.

En retirant les instructions linéarisées du code tangent, et en négligeant le temps de sauvegarde, ces résultats indiquent qu'une intégration complète de l'adjoint (intégration directe + sauvegarde + intégration rétrograde) est seulement 3 fois plus chère qu'une itération du modèle direct.

TAB. 6 – *Mesures des temps de calcul (en secondes)*

routine	Meso-NH$_{adiab}$	Meso-NH$^*_{adiab}$ partie tangente avec écriture	Meso-NH$^*_{adiab}$ partie adjointe avec lecture
BOUNDARIES	.062	.121	.112
INITIAL_GUESS	.239	.294	.267
ADVECTION	3.185	5.863	8.243
DYN_SOURCES	.258	.357	.630
NUM_DIFF	1.291	2.261	2.375
RELAXATION	.375	.464	.329
RAD_BOUND	.002	.091	.020
PRESSURE	7.188	15.908	13.776
ENDSTEP_DYN	.316	.386	.385
ENDSTEP_SCALAR	.057	.102	.049
Total	$C_D^T = 12.973$	$C_T^T = 25.847$	$C_A^T = 26.186$

Ces résultats ne sont valables que lorsque la trajectoire tient entièrement en mémoire. Ici, la trajectoire nécessite 60 Mega-mots (400 tableaux 3D, 200 tableaux 1D non comptés).

Remarque 5 *Dans la dernière version de Meso-NH$^*_{adiab}$, les sauvegardes de XT sont effectuées au niveau de la routine model$n, Pour enlever les instructions tangentes, il suffit donc d'appeler les routines directes à la place des routines tangentes. Par commodité, il reste encore des variables locales (2 variables unidimensionnelles) sauvées sur fichier dans la routine p_abs.*

Remarque 6 *Le code direct Meso-NH$_{adiab}$ contiennent des routines de bilan pour chaque variable d'état (budget) qui sont destinées à un usage scientifique spécifique, elles n'ont été reproduites dans les codes dérivés.*

8 Calcul avec algorithmes de check-points dans Meso-NH$^*_{adiab}$

8.1 Comparaison entre les schémas avec sauvegarde sur fichier

Les deux figures présentées ci-dessous indiquent le temps de calcul (unité arbitraire) nécessaire à l'évaluation de l'adjoint en fonction du nombre d'itérations $P \in \{10, 20, 40, 80, 160, 320\}$. Les schémas de check-points utilisent $\tau = 10$ ce qui revient à découper l'intervalle $[0,T]$ respectivement en $\{1, 2, 4, 8, 16, 32\}$ sous-intervalles. Tous ces calculs sont réalisés en utilisant le code linéaire tangent (calculs entre check-points et construction de trajectoire) comme code direct.

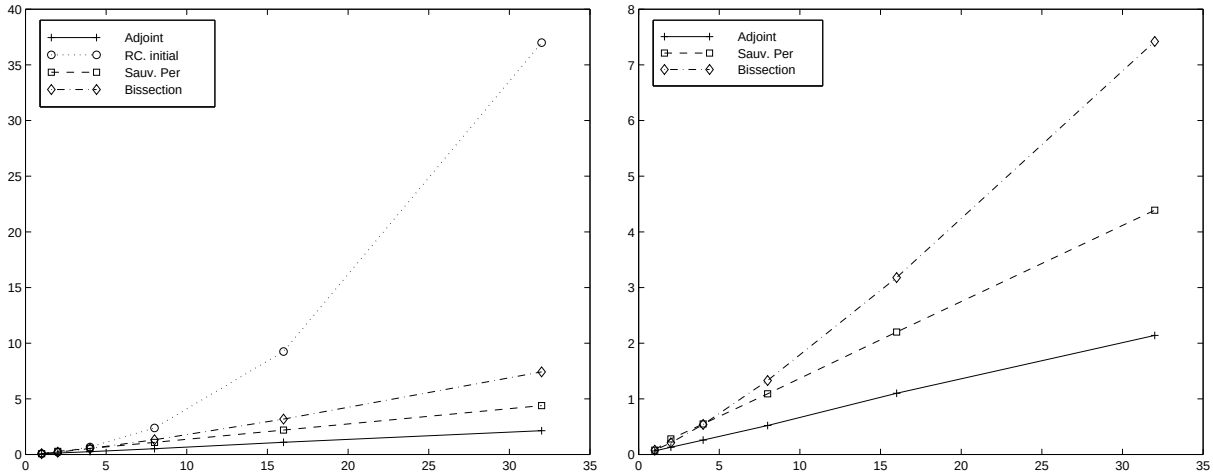


FIG. 9 – *Temps de calcul de l'adjoint en fonction du nombre de sous-intervalles.*

Le caractère quadratique du schéma avec recalcul depuis l'état initial est facilement observable sur la première figure. La seconde met en évidence le comportement affine des schémas de sauvegarde périodique (le schéma optimisé est utilisable jusqu'à $P = 40$). De plus, on vérifie que le rapport entre le temps d'utilisation du code tangent ($C_T^T = 2P - \tau$) et le temps d'utilisation du code adjoint ($C_A^T = P$) tend vers 2 lorsque le nombre d'itérations augmente.
En procédant au même type d'analyse, on note que le schéma de bissection a les coûts de calcul suivants

$$\begin{aligned} C_T^T &= \frac{1}{2}\log_2\left(\frac{P}{\tau}\right)P + P \\ C_A^T &= P, \end{aligned} \tag{37}$$

i.e. le rapport entre le temps de calcul avec le code tangent (C_T^T) et le temps de calcul avec le code adjoint est donné par la relation

$$\frac{C_T^T}{C_A^T} = \frac{1}{2}\log_2\left(\frac{P}{\tau}\right) + 1. \tag{38}$$

Dans le cadre de l'expérience ($P = 320,\ \tau = 10$), le rapport (C_T^T/C_A^T) est égal à 3.53 ce qui est proche de la valeur théorique 3.5. On retrouve numériquement le comportement logarithmique du schéma de bissection.

8.2 Comparaison entre régression arithmétique et Treeverse lorsque $C_S^m > C_A^m$

Pour établir les tableaux comparatifs proposés au chapitre 6, nous avions émis l'hypothèse que $C_S^m = C_A^m$, ce n'est plus le cas maintenant.
En effet, dans le paragraphe 7.1 nous avons mis en évidence que les quantités la place mémoire pour conserver un check-point pour reprendre l'exécution du modèle direct (C_S^m = mémoire pour XT"+"XM) et la place mémoire pour conserver les éléments de trajectoire pour l'évaluation de l'adjoint sur un pas de temps (C_A^m = mémoire pour XT) ne sont pas toujours identiques. En particulier, le rapport C_S^m/C_A^m est de 7/3 pour la simulation bidimensionnelle de référence (en 3D, ce rapport est de 9/4).

En utilisant cette remarque, les contraintes d'utilisation du schéma de régression arithmétique tenant compte du rapport $C_S^m/C_A^m = \gamma$ impliquent de choisir le nombre de check-points K et le nombre de pas de trajectoire maximum τ tels que :

$$\sum_{k=1}^{K}\tau_k = \sum_{k=1}^{K}(\tau - k\gamma) = K\tau - \gamma\frac{K(K+1)}{2} \geq P. \tag{39}$$

Pour exécuter la simulation de référence sur 16000 pas avec le schéma de régression arithmétique (γ =2), on peut donc choisir $\tau = 252$, $K = 126$, ce qui conduit à $N_{it}^{reg} = 15998$ itérations supplémentaires du modèle direct. Si l'on utilise "Treeverse" modifiée ($K = 126$, $\tau = 3$ i.e. même quantité de mémoire), $N_{it}^{Treev} = 39744$ sont nécessaires.

8.3 Exemples simplifiés d'application des schémas

Exemple 2D long : La simulation test sur une durée de 88 heures réelles, soit $P = 16000$ itérations, peut être entreprise avec le schéma de sauvegarde par régression arithmétique en choisissant $\tau = 200$ et 111 check-points (on considère que $C_S^m = C_A^m$). Ce schéma utilise alors 10% de la place mémoire disponible pour un coût de calcul total égal à 4 fois le coût de cette simulation en mode direct (paragraphe 7.3).

Exemple 3D : Pour sauver une fois l'état, il est nécessaire de disposer de 30 fois plus de place qu'en 2D (les variables bidimensionnelles sont codées sur trois points en y). On estime ainsi que la trajectoire d'une simulation sur 100 pas occuperait 1.8 Giga-mots; comme on ne dispose que de 1,2 Giga-mots, on ne peut sauver au plus que 66 check-points (ou 133 pas de trajectoire) . On en déduit aisément que faute de place le schéma de sauvegarde optimisé ne fonctionne pas toujours, on peut intégrer 4290 pas.
Dans de telles conditions, seuls les schémas de bissection ($\tau = 4$ convient pour $P = 16384$) ou binomial peuvent être utilisés.

Exemples type : (tableau extrait de l'article [11])

Sur chacune des expériences (Tab. 7) entreprises pour valider le modèle direct de Meso-NH, il est possible d'employer un schéma de check-points pour évaluer l'adjoint. On utilise le schéma avec placement par régression

TAB. 7 – *Expériences type*

cas	configuration	δx, δz, δt	discrétisation	durée	mémoire	CPU
ondes de relief (2D) (hydrostatique)	CL périod. + absorbant	2 km, 250 m, 20 s	90, 63, 3000	60000 s	2.4 Mw	167 s
ondes de relief (non-hydrostatique)	CL périod. + absorbant	133 m , 250 m, 2 s	180, 63, 100	4000 s	3.3 Mw	200 s
ondes de relief (3D)	CL périod. + absorbant	2 km, 100 m, 45 s	128,96,180, 889	40000 s	110 Mw	7400 s
ondes baroclines (non-hydrostatique)	CL périod. + réflexion	85 km , 500 m, 200 s	96, 48, 33, 4320	10 j	8.5 Mw	1886 s

arithmétique dans les cas 1 et 4, et "Treeverse" pour l'expérience 3.

TAB. 8 – *Expériences type* ($C_S^m = C_A^m$)

cas	discrétisation spatiale	P	mémoire (C_S^m)	schéma	(τ, K)	N_{it}
ondes de relief (2D)	90, 63	3000	2.4 Mw	rég. arithm.	(200,111)	2889
ondes de relief	180, 63	100	3.3 Mw	inutile		0
ondes de relief (3D)	128,96,180	889	110 Mw	"Treeverse"	(4,10)	3192
ondes baroclines	96, 48, 33	4320	8.5 Mw	rég. arithm.	(140,140)	4180

Dans l'expérience 3, le schéma par régression arithmétique ne permet d'intégrer que P=126 pas de temps (en supposant que C_S^m/C_A^m=9/4). En effet, on a

$$\begin{cases} Q = 1200 \text{ Mw}, \\ C_S^m = 110 \text{ Mw}, \quad C_A^m = 4C_S^m/9 = 48.889 \text{ Mw}, \end{cases} \tag{40}$$

on en déduit que $\tau = Q/C_A^m = 24$. En résolvant l'inégalité (39), on détermine le nombre d'itérations maximum P tel que le trinôme suivant ait une solution

$$\gamma K^2 - 2K\tau + (\gamma + 2P) = 0.$$

On en déduit que $P \leq \dfrac{1}{2\gamma}(\tau^2 + \gamma^2)$: P est égal à 126. Dans ces conditions, on choisit $K = 10$ ($\leq \dfrac{4}{9}\tau$).
Dans ce cas il faudrait $N_{it} < 126$ (rég. arithm.) contre 287 itérations pour "Treeverse".

Remarque 7 *Pour simplifier, on a considéré que la quantité de mémoire utilisée correspond à C_S^m. En fait la quantité de mémoire inclut un certain nombre de variables globales qui sont constantes au cours des itérations : elles n'ont pas à être sauvées. Les conditions d'utilisation des schémas sont donc exagérées, une analyse plus fine permettrait de choisir vraiment le meilleur schéma.*

9 Conclusions

Dans ce rapport consacré aux schémas de check-points, seuls les adjoints de schémas numériques temporels ont été abordés car il est facile de décomposer la trajectoire en sous-parties de même taille lorsque l'on suppose que la trajectoire d'une itération est de taille constante. Les schémas de check-points sont utilisables pour résoudre d'autres types de problèmes, mais les partitions peuvent être plus difficiles à établir.

Les propriétés théoriques décrites font apparaître que le schéma "Treeverse" de Griewank convient bien aux adjoints générés avec calculs locaux de trajectoire, mais qu'il n'est pas toujours le meilleur lorsque l'adjoint est

écrit avec gestion de trajectoire sur fichier.
En effet, lorsqu'il est utilisable, le schéma plaçant les check-points suivant une régression arithmétique (paragraphe 5.3) est toujours meilleur en nombre d'itérations du direct que le schéma "Treeverse". Cette constatation est d'autant plus vraie que le rapport entre la place mémoire requise pour un check-point, et la place utilisée pour la trajectoire créée en une itération est grand.

Sur Meso-NH, ce rapport est de l'ordre de deux car la résolution du système d'équations est basée sur un "Leap-Frog" (paragraphe 7.1) et on préfère le schéma par placement régressif, voire (quelque fois) le schéma de bissection, à "Treeverse". Cela étant, "Treeverse" est utilisable à des coûts corrects lorsque le placement par régression échoue.
Par exemple, l'adjoint de Meso-NH se calcule au prix de :

- 3 fois le direct lorsque la trajectoire tient entièrement sur disque ;
- 4 fois le direct lorsque le schéma par placement régressif est utilisable ;
- 5.6 fois le direct pour la simulation 3D du tableau 7 nécessitant l'utilisation de "Treeverse" (3.6 pour "Treeverse", 2 pour l'adjoint).

Les algorithmes de check-points autorisent les calculs de gradients avec la bibliothèque Meso-NH$^*_{adiab}$ et les simulations qui ont un sens physique à meso-échelle peuvent être traitées (3D, phases de l'eau, ...) : il est possible de calculer des gradients avec l'adjoint de Meso-NH i.e. d'employer les méthodes d'assimilation de données variationnelle pour faire de la prévision.

Références

[1] I. Charpentier et M. Ghemires, *Génération automatique de codes adjoints: Stratégies d'utilisation pour le logiciel Oðyssée, Application au code météorologique Meso-NH*, Rapport de Recherche INRIA RR-3251, 1997.

[2] I. Charpentier and M. Ghemires, *Efficient adjoint derivatives: Application to the atmospheric model Meso-NH*, submitted.

[3] I. Charpentier, J.-P. Pinty et M. Ghemires, *Premières expériences physiques avec les codes dérivés de Meso-NH.*

[4] F. Eyssette, C. Faure, J.C. Gilbert and N. Rostaing-Schmidt (1996) Applicabilité de la différentiation automatique à un système d'équations aux dérivées partielles régissant les phénomènes thermohydrauliques dans un tube chauffant, Rapport de Recherche INRIA RR-2745.

[5] C. Faure, *Documentation succinte d'Oðyssée version 1.6*, décembre 1996.

[6] C. Faure, and Y. Papegay, *Oðyssée, Version 1.6, The user's reference manual*, Rappor t Technique INRIA RT-0211.

[7] J.C. Gilbert, G. Le Vey et J. Masse (1991) La Différentiation automatique de fonctions representées par des programmes, Rapport de Recherche INRIA RR-1557.

[8] A. Griewank (1989) On automatic differentiation, Editors: M. M. Iri and K. Tanabe (Editors), Mathematical Progamming: Recent Developments and Applications (Kluwer Academic Publishers), 83-108.

[9] A. Griewank, *Achieving logarithmic growth of temporal and spatial complexity in Reverse Automatic Differentiation*, Optimization Methods and Software, 1(1):35-54, 1992.

[10] A. Griewank and A. Walther, *Treeverse: An implementation of the checkpointing for the reverse or adjoint mode of differentiation.*

[11] J.P. Lafore, J. Stein, N. Ascencio, P. Bougeault, V. Ducrocq, J. Duron C. Fischer, P. Héreil, P. Mascart, V. Masson, J.P. Pinty, J.L. Redelsperger, E. Richard and J. Vilà-Guerau de Arellano, *The Meso-NH atmospheric simulation system. Part I: adiabatic formulation and control system*, Ann. Geophysicae, 16: 90-109, 1998.

[12] F.-X. Le Dimet and O.Talagrand (1986) Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects, *Tellus*, **38A**, 97-110.

[13] J.-L. Lions (1971) *Optimal control of systems governed by partial differential equations* (Springer-Verlag, Berlin).

[14] Météo-France, CNRS *The Meso-NH Atmospheric Simulation System: Scientific Documentation*, version de septembre 1995.

[15] Météo-France, CNRS *The Meso-NH Atmospheric Simulation System: Algorithmic Documentation (MASDEV2_3)*, version de juillet 1996.

[16] Météo-France, CNRS *The Meso-NH User's Guide*, version de septembre 1996.

[17] J. Morgenstern (1985) How to compute fast a function and all its derivatives, a variation on the theorem of Baur-Strassen, SIGACT News, **16**, 60-62.

[18] J.P. Pinty, R. Benoit, E. Richard and R. Laprise, *Simple Tests of a semi-implicit semi-lagrangian model on 2D Mountain Wave problems*, Mon. Wea. Rev., 123, 10: 3042-3058, 1995.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399