Méthode et Pratiques Scientifiques

Nahid Ehmad

Table des matières

1 Introduction

Technique de regroupement On cherche à partitionner les sommets V d'un graphe $G = (V, E)$ dans un ensemble de clusters $S_k \subseteq V$ tel que $\bigcup_{k=1}^{p} S_k = V$. La modularité est une mesure de la qualité d'un partitionnement des noeuds dans les communauté.

La plupart du temps, les solutions à ce problèmes sont NP-Complète : on les approxime par à l'aide d'algorithmes, qui sont principalement basées sur du calcul de valeurs propres. Ce dernier prend 90% du temps total ! De plus, 90% du temps passé dans le solveur s'effectue dans des multiplications (sparse) matrice-vecteur .

1.1 Principaux éléments

- Méthode itératives pour problèmes de grande taille
- Méthodes hybride synchrone/asynchrone
- Méthode numériques d'algèbre linéaire pour le traitement de masses de données (simulation de phénomènes physiques, analyse des réseaux sociaux, etc)
- Méthode de compression des structures creuses
- Modèles de programmation de graphe de Tâches, PGAS
- Métriques de performances

Quelques rappels sur la méthode de Gauss

Méthode directe vs Itérative Une méthode directe est une solution dont la solution peut être calculée avec un nombre fini d'opérations arithmétiques élémentaires. A contrario, une méthode itératives est un procédé qui part d'une information initiale arbitraire et renvoie un résultat approché. Pour cela, on réinjecte le résultat en entrée de l'algorithme afin de raffiner la solution.

Exemple Pour trouver les valeurs propres d'une matrice, il faut trouver les racines du polynôme caractéristique, ce qui est impossible dès que le polynôme dépasse le degrés 4 : on n'utilise donc jamais une résolution directe.

La plupart du temps, les matrices sont creuses : on n'utilise donc pas les mêmes méthodes, plus particulièrement pas de méthodes directes car celle-ci ne sont pas adaptées.

Une des problématiques de nos jours est la consommation électrique. La nouvelle frontière du *calcul exascale* impose des consommations énormes. La machine la plus puissante au 11/16 consommait plus de 15MW !

1.2 But du cours

Le principe de ce cours est la mise au point d'un paradigme de calcul pour la programmation à grande échelle et par la suite en déduire l'introduction de méthodes numériques modernes et "intelligentes" comme celles Krylov hybrides smart-tunées.

2 Quel paradigme de programmation ?

Certains chercheurs pensent directement à OpenMP/MPI lorsque l'on parle d'informatique parallèle. C'est *faux* : il ne s'agit que d'*outils de programmation*, qui ne sont donc pas représentatifs de la théorie derrière.

Par exemple, une multiplication matrice-vecteur puis réduction par addition du vecteur obtenu, il faut faire communiquer ensemble tous les processus : impossible. On rappel que le principal coût énergétique est le déplacement des données !

On cherche donc à :

— Minimiser le mouvement des données (amoindrir au maximum le coût des communications)

— Consommation énergétique

— Parallélisme multi-niveau

— Ordonnancement multi-niveau

— Équilibrage de charge multi-niveau

— Tolérance aux pannes

2.1 Une approche par composants...

Il faudrait un modèle de programmation laissant la possibilité de définir des blocs autonomes (cf GLCS) appelés *composants* réutilisables dans différents contextes. Par exemple, un composant pour le produit matrice-vecteur creux peut être réutilisé dans un solveur calculant le valeurs propres. Si le travail est bien fait, on peut même réutiliser un composant séquentiel dans un programme parallèle.

2.2 ... eux-même liés par un graph

On peut définir un problème comme un ensemble de composants liés entre eux par un graph. L'application est donc un ensemble de composants *de très gros grains*, car chaque composant peut être lui-même un graph de composants.

Un projet de recherche à ce sujet est YML (attention, encore expérimental).

Les principales caractéristiques du modèle de programmation recherché sont alors :

— Privilégier grandement les communication asynchrones sur les communications synchrones

— Prendre en compte l'hétérogénéité des unités de calcul

— Prendre en compte la tolérance aux fautes

— Encourager l'équilibrage de tâche

— Introduire l'auto-tuning (on choisit automatiquement des paramètres pendant l'exécution avec assistance de l'utilisateur final)

3 Unir et conquérir

On prend pour exemple la résolution d'un système linéaire $Ax = b$, où $A \in \mathcal{M}_n(\mathbb{C})$ et $b, x \in \mathbb{C}$ sont creux. On cherche quelques *paires de Ritz* $\Lambda_k = (\lambda_1, .., \lambda_k)$ et $U_k = (u_1, ..., u_k)$ tel que $\forall i \leq k, Au_i = \lambda_i u_i$.

Ce problème est au cœur de l'algorithme de recherches de données de Google.

Problème : Les algorithmes classiques ne sont pas adaptables aux grandes tailles (par exemple, la méthode QR pour trouver les valeurs propres a une précision dépendant de la taille de la matrice). On va donc chercher à réduire notre problème dans un sous-espace de taille moindre, chercher la solution dans ce sous-espace puis effectuer une transformation inverse afin d'obtenir un approximation des solutions recherchés. Le plus souvent, la précision n'est pas suffisante, et on va définir un sous-espace de meilleur qualité pour réitérer le processus.

Pour notre exemple, on va projeter notre matrice dans un sous-espace de Krylov défini par la base $(v, Av, ..., A^{m-1}v)$. On résous le problème de valeur propre dans ce sous-espace ; puis on réitère le processus avec le v trouvé en résolution du système de moins grande taille.

Attention, il ne s'agit pas vraiment d'un méthode itérative, mais plus d'un méthode "à redémarrage" (projection-résolution-retour à l'espace de départ).

La méthode peut être redémarrée explicitement (ERAM/Saad), c'est-à-dire en mettant à jour explicitement v puis recommençant le problème. Par contre, cette méthode n'est pas toujours stable. Un redémarrage implicite peut aussi être effectué (IRAM/Sorrensen) ; mais repose sur des bases mathématiques plus conséquentes.

Ces méthodes on par contre un inconvénient : si les valeurs propres sont trop approchées les unes des autres, l'algo peut les "confondre".

La méthode *Unite and conquer* consiste à utiliser plusieurs méthodes.

4 Multiple Krylov Subspace

On va non pas combiner un ensemble de méthode, mais combiner différentes instances de la même méthode. Par exemple, on va projeter-résoudre-retourner, puis on partage les résultats intermédiaires, et on choisit les meilleurs vecteurs retournés par les instantes. On peut également choisir des tailles de sous-espaces différentes de manière à recouvrer les temps des communications (asynchrones).