Dr B. Diop
CINERI, 2025

Exercices avancés de Programmation Parallèle et Distribuée

Chimie Computationnelle

Notre troisième étude de cas, comme la première, provient de la science computationnelle. C'est un exemple d'application qui accède à une structure de données distribuée de manière asynchrone et qui se prête à une décomposition fonctionnelle.

Contexte en Chimie

Les techniques computationnelles sont de plus en plus utilisées comme alternative aux expériences en chimie. Ce domaine, appelé chimie quantique ab initio, utilise des programmes informatiques pour calculer des propriétés fondamentales des atomes et des molécules (telles que la force des liaisons et les énergies de réaction) à partir de principes fondamentaux, en résolvant diverses approximations de l'équation de Schrödinger qui décrit leur structure de base. Cette approche permet aux chimistes d'explorer des voies de réaction qui seraient dangereuses ou coûteuses à étudier expérimentalement.

Une des applications de ces techniques concerne l'étude des processus biologiques. Par exemple, l'illustration 6 montre un modèle moléculaire de la région du site actif de l'enzyme malate déshydrogénase, une enzyme clé dans la conversion du glucose en ATP (molécule riche en énergie). Cette image provient d'une simulation du transfert d'un anion hydrure du substrat (le malate) vers un cofacteur, le nicotinamide adénine dinucléotide. Les deux isosurfaces, colorées en bleu et en brun, représentent des densités électroniques plus faibles et plus élevées respectivement, calculées grâce à une méthodologie combinant la mécanique quantique et classique. Les atomes de carbone, d'oxygène, d'azote et d'hydrogène sont représentés respectivement en vert, rouge, bleu et blanc.

Dr B. Diop
CINERI, 2025

Exercices avancés de Programmation Parallèle et Distribuée

Un élément fondamental de plusieurs méthodes en chimie quantique est le calcul de la matrice de Fock, une matrice bidimensionnelle qui représente la structure électronique d'un atome ou d'une molécule. Cette matrice, notée F, est de taille N × N et est obtenue en effectuant la somme suivante pour chaque élément :

$$\mathbf{F}_{ij} = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} \mathbf{D}_{kl} \left(\mathbf{I}_{ijkl} - \frac{1}{2} \mathbf{I}_{ikjl} \right),$$

où D est une matrice N × N en lecture seule et où I représente des intégrales calculées à partir des éléments i, j, k et l d'un tableau A de taille M (en lecture seule). Une intégrale peut être vue comme une approximation de la force répulsive entre deux électrons.

Dr B. Diop
CINERI, 2025

Exercices avancés de Programmation Parallèle et Distribuée

```
procedure fock_build
begin
  for i = 1 to N
     for j = 1 to i
        for k = 1 to j
           for l = 1 to k
              integral(i ,j ,k ,l )
           endfor
        endfor
     endfor
  endfor
end

procedure integral(i ,j ,k ,l )
begin
  I = compute_integral(i, j, k, l)
  F_ij = F_ij + D_kl I
  F_kl = F_kl + D_ij I
  F_ik = F_ik + D_jl I
  F_jl = F_jl - (1/2) D_ik I
  F_il = F_il - (1/2) D_jk I
  F_jk = F_jk - (1/2) D_il I
end
```

Algorithm 2.3 Sequential Fock matrix construction algorithm. When symmetry is exploited in the computation, each of $N^4/8$ integrals can contribute to six elements of F.

L'équation ci-dessus implique qu'il faut en apparence N^4 intégrales pour calculer F, soit un total de N^4 intégrales. Cependant, en exploitant la redondance et la symétrie dans les intégrales et la matrice F, il est possible de réduire ce nombre à N^2. Cela conduit à l'algorithme 2.3, dont la logique peut sembler étrange mais qui est optimisée pour minimiser les calculs inutiles.

Dr B. Diop
CINERI, 2025

Exercices avancés de Programmation Parallèle et Distribuée

Dans les systèmes moléculaires d'intérêt pour les chimistes, la taille du problème N peut aller de 10^2 à 10^4. Comme l'évaluation d'une intégrale est une opération coûteuse ($O(N^4)$), la construction de la matrice de Fock nécessite jusqu'à $O(N^5)$ opérations. De plus, la plupart des méthodes nécessitent plusieurs matrices de Fock pour améliorer progressivement l'approximation de la structure électronique d'une molécule. Ces contraintes ont conduit au développement de nombreux algorithmes parallèles et à des méthodes améliorées visant à réduire le nombre d'intégrales à calculer.

Conception de l'Algorithme en Chimie

Partitionnement

Étant donné que le problème de la matrice de Fock est centré sur des matrices symétriques F et D, une approche évidente consiste à appliquer une décomposition de domaine. Une stratégie naïve consiste à diviser la matrice en $N(N+1)/2$ tâches, chacune correspondant à un élément de F et D. Chaque tâche aurait $O(N^2)$ données à traiter et serait responsable du calcul de N^2 intégrales.

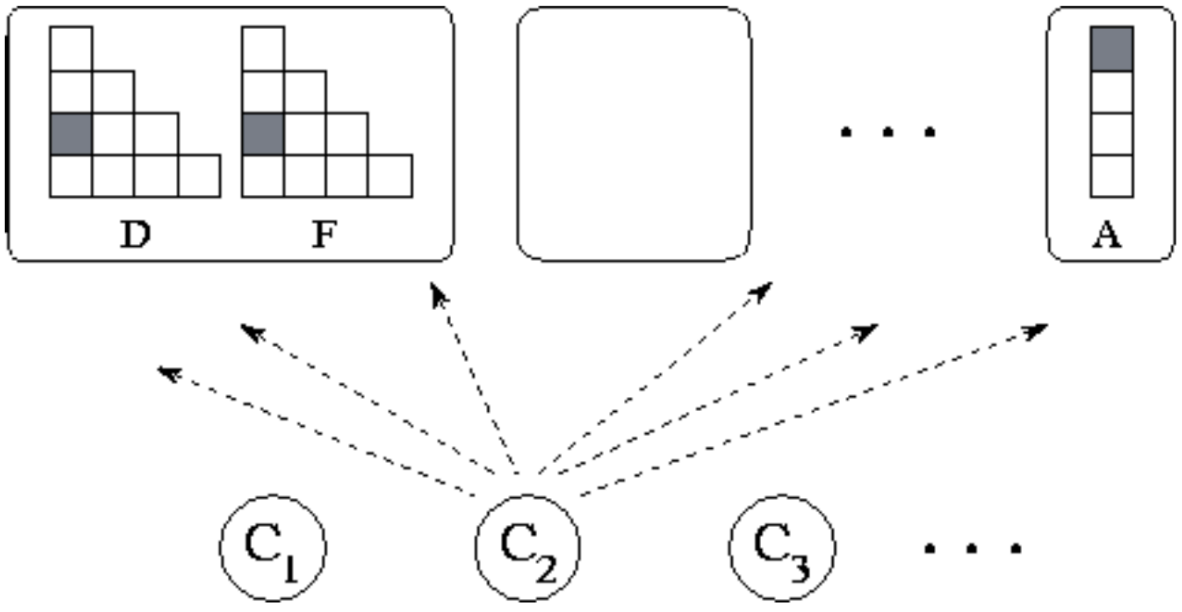

Figure 2.31: *Functional decomposition of Fock matrix problem. This yields about data tasks, shown in the upper part of the figure, and computation tasks, shown in the lower part of the figure. Computation tasks send read and write requests to data tasks.*

Cependant, cette méthode présente un inconvénient majeur : elle ne tire pas parti des redondances et des symétries, ce qui entraîne un surcoût d'un facteur 8 en calculs inutiles. Une approche alternative basée sur la décomposition fonctionnelle est bien plus efficace.

Plutôt que de découper les données, on peut se concentrer sur les opérations à effectuer. Chaque intégrale requiert 6 éléments de D et contribue à 6 éléments de F. Ainsi, nous pouvons définir N^4 tâches de calcul, chacune étant responsable du calcul d'une intégrale.

Puisque F, D et A sont de grandes structures de données, nous définissons également des tâches de données responsables uniquement de répondre aux requêtes de lecture/écriture. Ces tâches encapsulent des éléments des matrices D et F, ainsi que des éléments du tableau A. Au total, nous avons environ N^4 tâches de calcul et N^2 tâches de données.

Communication et Agglomération

Nous avons maintenant défini des tâches de calcul et des tâches de gestion des données. Chaque tâche de calcul doit effectuer seize communications : six pour obtenir les éléments de la matrice D, quatre pour obtenir les éléments de la matrice A et six pour stocker les éléments de la matrice F. Étant donné que les coûts de calcul des différentes intégrales peuvent varier de manière significative, il ne semble pas possible d'organiser ces opérations de communication selon une structure régulière, comme préconisé dans la section précédente.

Sur de nombreux ordinateurs parallèles, le coût d'une intégrale sera comparable au coût d'une communication. Par conséquent, les exigences en matière de communication doivent être réduites par un regroupement (agglomération). Nous décrivons deux stratégies alternatives pouvant être utilisées pour atteindre cet objectif. Leurs besoins en données sont illustrés dans la figure 2.32.

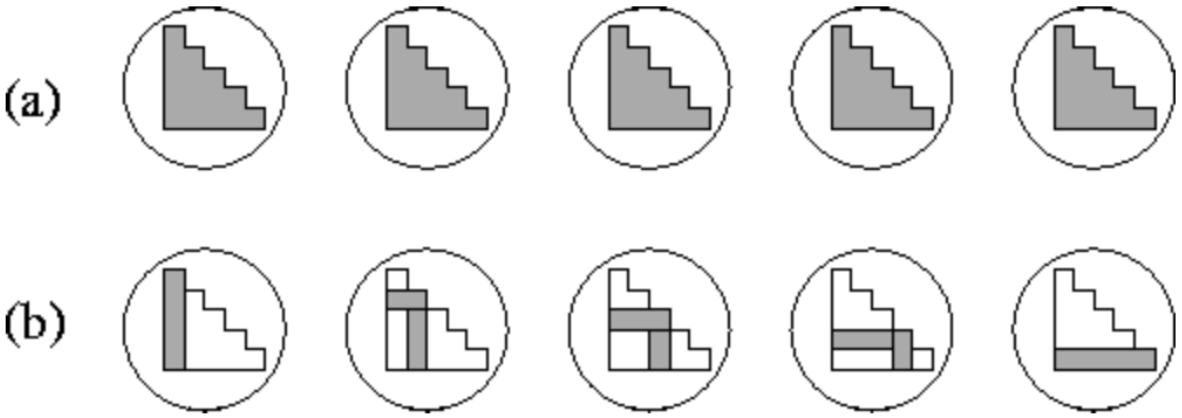

Figure 2.32 : Stratégies d'agglomération pour la construction de la matrice de Fock avec N=P=5N = P = 5N=P=5, pour (a) l'algorithme de réplication totale et (b) l'algorithme de réplication partielle.

Dans chaque cas, les cinq tâches sont représentées avec des ombrages indiquant la portion des matrices symétriques D et F allouée à chaque tâche.

- **(a)** : Chaque matrice est entièrement répliquée dans chaque tâche.
- **(b)** : Chaque tâche se voit attribuer une seule ligne et une seule colonne, ce qui correspond à un facteur de réplication de deux.

Réplication totale

Les coûts de communication peuvent être considérablement réduits en répliquant les matrices F et D dans chacune des PPP tâches, soit une par processeur dans un ordinateur parallèle. Chaque tâche est alors responsable de 1/P des intégrales. Le calcul peut ainsi s'exécuter dans chaque tâche sans aucune communication. La seule coordination nécessaire est une sommation finale pour accumuler les matrices FFF partielles. Cette opération peut être réalisée en utilisant un algorithme parallèle de réduction vectorielle, décrit dans la section 11.2.

La technique consistant à répliquer les structures de données sur chaque processeur d'un ordinateur parallèle est couramment utilisée en calcul parallèle pour réduire les coûts de développement logiciel. Elle permet d'adapter rapidement un code séquentiel existant à une exécution parallèle, sans qu'il soit nécessaire de modifier les structures de données. L'inconvénient principal de cette approche est son manque de passage

à l'échelle (scalabilité). En effet, les besoins en mémoire augmentent proportionnellement au nombre de tâches créées, ce qui signifie que la taille maximale du problème pouvant être résolu est limitée non pas par la mémoire totale de l'ordinateur parallèle, mais par la mémoire disponible sur un seul processeur.

Par exemple, sur un ordinateur à 512 processeurs, chacun doté de 16 Mo de mémoire, une implémentation du code de chimie quantique **DISCO** utilisant cette stratégie ne peut pas résoudre des problèmes avec N>400. En théorie, il serait intéressant de résoudre des problèmes où N est dix fois plus grand.

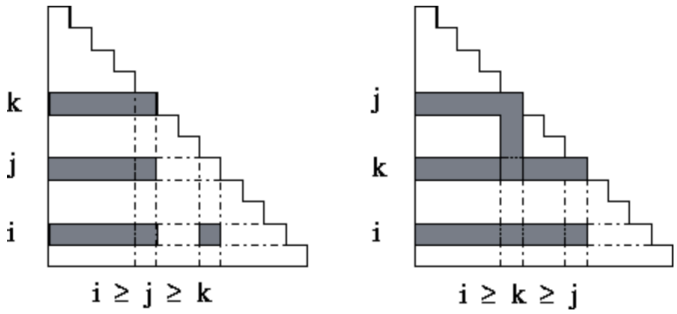

Figure 2.33: *Data requirements for integral clusters. Each task accesses three rows (and sometimes columns) of the D and F matrices.*

Figure 2.33 : Exigences en matière de données pour les regroupements d'intégrales.

Chaque tâche accède à trois lignes (et parfois colonnes) des matrices D et F.

2. Réplication partielle

Une autre approche consiste à procéder ainsi :
Tout d'abord, nous regroupons les calculs d'une manière qui semble évidente, à savoir en transformant la boucle interne de la procédure `fock_build` (voir l'algorithme 2.3) en une tâche distincte. Cela produit des tâches de calcul, chacune responsable d'un certain nombre d'intégrales. Ensuite, nous analysons les exigences en communication de chaque tâche. Il apparaît qu'il existe une **forte localité** dans les données requises par ces regroupements d'intégrales : chaque groupe d'intégrales accède aux i-ème, j-ème et k-ème lignes (et parfois colonnes) des matrices D et F (voir figure 2.33).

Pour exploiter cette localité, nous regroupons les données en créant N tâches de données, chacune contenant une paire ligne/colonne des tableaux bidimensionnels D et F, ainsi que l'intégralité du tableau unidimensionnel A. Dans ce schéma :

- Chaque élément des matrices D et F est répliqué **une seule fois**.

Dr B. Diop
CINERI, 2025

Exercices avancés de Programmation Parallèle et Distribuée

- A est répliqué N fois.
  Les besoins en stockage augmentent donc en moyenne de N à **3N par tâche**.

Grâce à cette réplication, chaque tâche de calcul ne nécessite désormais des données que de trois tâches de données. Ainsi, le **nombre de messages échangés** est réduit de $O(n^4)$ à $O(n^3)$. Toutefois, le **volume total communiqué** reste le même à $O(n^4)$. Comme le coût de communication d'un mot est généralement bien inférieur à celui du calcul d'une intégrale, cette approche permet de construire un algorithme parallèle **efficace**.

Affectation des tâches

L'algorithme de construction de la matrice de Fock par **réplication partielle** crée N tâches de données et un certain nombre de tâches de calcul. Nous utilisons la notation (i, j, k) pour identifier la tâche de calcul responsable du calcul des intégrales I ; cette tâche nécessite des données provenant des tâches de données i, j et k.

Pour compléter l'algorithme parallèle, nous devons définir une **affectation** (mapping) des tâches de calcul et des tâches de données aux processeurs.

Nous supposons disposer de P processeurs. Étant donné que chaque tâche de données recevra à peu près le même nombre de requêtes, nous affectons **une tâche de données par processeur**. Il reste donc à résoudre le problème de l'affectation des tâches de calcul. Plusieurs approches sont envisageables :

1. **Un mapping simple** :
   - La tâche (i, j, k) est affectée au même processeur que la tâche de données i.
   - Comme chaque tâche de calcul communique avec les tâches de données i, j et k, cette approche **réduit d'un tiers les communications inter-processeurs**.
   - Cependant, un inconvénient majeur est que le **nombre d'intégrales dans une tâche** ainsi que la **quantité de calcul par intégrale** peuvent varier, ce qui peut entraîner une répartition déséquilibrée du calcul entre les processeurs.
2. **Un mapping probabiliste** :
   - Les tâches de calcul sont affectées aux processeurs de manière **aléatoire** ou selon une **stratégie cyclique**.

Dr B. Diop
CINERI, 2025

Exercices avancés de Programmation Parallèle et Distribuée

---

3. **Un algorithme d'ordonnancement des tâches** :
    - Les tâches de calcul sont affectées aux **processeurs inactifs**.
    - Comme un problème est représenté par trois entiers (i, j, k) et que plusieurs problèmes peuvent être facilement regroupés en un seul message, un **planificateur centralisé** simple peut être utilisé.
    - **Des études empiriques** montrent qu'un tel planificateur fonctionne bien sur un **nombre allant jusqu'à quelques centaines de processeurs**.
4. **Des schémas hybrides** :
    - Par exemple, les tâches peuvent être affectées **aléatoirement** à des **groupes** de processeurs, dans lesquels un **planificateur maître-esclave** est utilisé.

Le **meilleur schéma** dépendra des **contraintes de performance**, ainsi que des caractéristiques du **problème** et de la **machine** utilisée.

Résumé en Chimie

Nous avons développé **deux algorithmes parallèles** pour la construction de la matrice de Fock.

1. **Première approche : réplication totale**
    - Les matrices **F** et **D** sont **répliquées** dans chacune des N tâches.
    - Les calculs d'intégrales sont répartis entre ces tâches.
    - Un algorithme de **sommation** est utilisé pour **additionner** les contributions à la matrice **F** accumulées dans les différentes tâches.
    - Cet algorithme est **simple**, mais **non évolutif** (non scalable).
2. **Deuxième approche : réplication partielle et agglomération**
    - Les matrices **F, D** et **A** sont **partagées** entre les N tâches, avec une **petite quantité de réplication**.
    - Les calculs d'intégrales sont **agglomérés** en un certain nombre de **tâches de calcul**, chacune contenant plusieurs intégrales.
    - Ces tâches sont ensuite **affectées aux processeurs**, soit de manière **statique**, soit via un **ordonnancement dynamique**.

Analyse des compromis

Dr B. Diop
CINERI, 2025

Exercices avancés de Programmation Parallèle et Distribuée

Cette étude de cas met en lumière certains **arbitrages** dans la conception des algorithmes :

- **Le premier algorithme** réduit **drastiquement** les coûts de communication et de développement logiciel, mais **n'est pas évolutif**.
- **Le second algorithme** a un coût de communication plus élevé, mais **il est très évolutif** : ses **besoins en mémoire augmentent uniquement avec la taille du problème**, et non avec le nombre de processeurs.

Pour choisir entre ces deux algorithmes, il est nécessaire de **quantifier leurs performances parallèles** et d'évaluer **l'importance de l'évolutivité**, en fonction des **exigences de l'application** et des **caractéristiques de l'ordinateur parallèle cible**.