Exemples d'applications de la programmation parallèle

Exercices pour le Module 10

INF8601 Systèmes informatiques parallèles

Michel Dagenais

École Polytechnique de Montréal

Département de génie informatique et génie logiciel

Gain parallèle avec MapReduce

Un système MapReduce sur une grappe de 100 noeuds est utilisé pour compter la fréquence des mots dans une très grosse base de donnée de 1000Gio. Chaque noeud de la grappe contient 3/100 de la base de donnée, de sorte que chaque fichier est présent sur 3 noeuds pour des fins de redondance. Chaque noeud peut calculer la fréquence des mots sur un morceau de n Gio en $2 + 30n$ secondes. La phase de réduction dans ce cas-ci prend un temps négligeable. La probabilité qu'un noeud tombe en panne est de 0.01 pendant le temps requis pour traiter 1Gio. Trois scénarios sont possibles, travailler sur des morceaux de 1Gio, de 5Gio ou de 10Gio. Lorsqu'un système tombe en panne pendant le calcul d'un morceau, ce calcul doit être repris par la suite. Quel scénario est le plus efficace?

Gain parallèle avec MapReduce

Pour un morceau de 1Gio, le temps est de 32 secondes. Avec un taux de panne de 0.01, cela donne 1Gio x 0.99 / 32s = 0.0309375 Gio/s. Pour un morceau de 5Gio, on a une probabilité de succès de $0.99^5 = 0.95099$, ce qui donne 5Gio x 0.95099 / 152s = 0.031282 GiO/s. Pour un morceau de 10Gio, on a une probabilité de succès de $0.99^{10} = 0.904382$, ce qui donne 10Gio x 0.904382 / 302s = 0.029946 Gio/s. Le plus efficace est 5Gio. Ceci ne calcule que le temps d'utilisation des noeuds et non pas le temps total écoulé. Si la probabilité d'avoir un noeud parmi 100 en panne est grande, et qu'il faut reprendre une tâche, il sera beaucoup moins dommageable de reprendre une tâche de 1Gio. Si chaque noeud fait 10 morceaux de 1Gio en 320s, et un noeud en fait un 11ème pour reprendre la panne, ceci donne 352s. A la place, s'il y a un morceau de 10Gio à reprendre, le temps sera de 1 morceau de 10Gio par noeud pour 302s et un 2ème morceau pour reprendre la panne, ce qui donne 604s.

Calcul préfixe

Un calcul préfixe est réalisé par la méthode en 2 phases. Avec un calcul sériel, il faut n additions pour faire la somme préfixe d'une suite de n nombres et donc n unités de temps. Que deviendra ce nombre d'unités de temps écoulé si le problème est parallélisé en OpenMP sur m coeurs?

Calcul préfixe

Un calcul préfixe est réalisé par la méthode en 2 phases. Avec un calcul sériel, il faut n additions pour faire la somme préfixe d'une suite de n nombres et donc n unités de temps. Que deviendra ce nombre d'unités de temps écoulé si le problème est parallélisé en OpenMP sur m coeurs?

Dans la première phase, chaque coeur fera n/m additions en parallèle, ce qui prend n/m unités de temps. Ensuite, la phase de réduction peut prendre entre m et log(m) unités de temps, selon si elle est sérielle ou parallèle en arbre. La dernière phase prendra aussi n/m additions en parallèle, pour un total de $2n/m + \log(m)$, ce qui donne un facteur d'accélération proche de m/2.

Multiplication de matrices

Pour effectuer la multiplication de deux matrices de taille n en sériel, il faut n^3 additions et multiplications et donc autant d'unités de temps. Que deviendra ce nombre d'unités de temps écoulées si le problème est parallélisé en OpenMP sur m coeurs, chaque coeur s'occupant de n / m rangées?

Multiplication de matrices

Pour effectuer la multiplication de deux matrices de taille n en sériel, il faut n^3 additions et multiplications et donc autant d'unités de temps. Que deviendra ce nombre d'unités de temps écoulées si le problème est parallélisé en OpenMP sur m coeurs, chaque coeur s'occupant de n / m rangées?

Pour chaque rangée, il faut n^2 additions et multiplications. Avec m coeurs, le temps deviendra de $n/m \times n^2 = n^3/m$ soit un facteur d'accélération de m. Ceci néglige cependant la phase de lecture en entrée et d'écriture en sortie des matrices, et ne tient pas compte de l'interférence entre les coeurs pour les accès en mémoire.

Puissance de matrices

On veut mettre une matrice de taille n à la puissance 200. Comment peut-on paralléliser cette opération. Quel sera le facteur d'accélération sur m coeurs si on ne tient compte que du nombre d'additions et multiplications requises?

Puissance de matrices

On veut mettre une matrice de taille n à la puissance 200. Comment peut-on paralléliser cette opération. Quel sera le facteur d'accélération sur m coeurs si on ne tient compte que du nombre d'additions et multiplications requises?

La mise à la puissance 200 de A peut se faire en calculant successivement $A^2, A^4, A^8, A^{16}, A^{32}, A^{64}, A^{128}, A^{128+64=192}, A^{192+8=200}$ en 9 multiplications de matrices. L'accélération sera la même que pour la multiplication de matrices, soit m.

Inversion de matrices

Appliquez la méthode itérative de Jacobi à la solution du système d'équations linéraires ($b = Ax$) suivant:

```
A = np.array([
    [5, 2, 1, 1],
    [2, 6, 2, 1],
    [1, 2, 7, 1],
    [1, 1, 2, 8]
])
b = np.array([29, 31, 26, 19])
```

Inversion de matrices

```
def jacobi(A, b, x_init, epsilon=1e-10, max_iterations=500):
    D = np.diag(np.diag(A))
    LU = A - D
    x = x_init
    D_inv = np.diag(1 / np.diag(D))
    for i in range(max_iterations):
        x_new = np.dot(D_inv, b - np.dot(LU, x))
        print("x:", x_new)
        if np.linalg.norm(x_new - x) < epsilon:
            return x_new
        x = x_new
    return x

x_init = np.array([1, 1, 1, 1])
x = jacobi(A, b, x_init)
print("computed b:", np.dot(A, x))
print("real b:", b)
```

Inversion de matrices

```
x: [5.         4.33333333 3.14285714 1.875     ]
x: [3.06309524 2.13988095 1.49404762 0.42261905]
x: [4.56071429 3.57718254 2.60493197 1.35111607]
x: [3.57791738 2.5529319  1.84768637 0.7065299 ]
x: [4.26798399 3.2403771  2.3728127  1.14672225]
... 10 ...
x: [3.98829709 2.94959415 2.15843325 0.96331198]
x: [3.99581329 2.95720456 2.1641718  0.96815528]
... 46 ...
x: [3.99275362 2.95410628 2.16183575 0.96618357]
x: [3.99275362 2.95410628 2.16183575 0.96618357]
x: [3.99275362 2.95410628 2.16183575 0.96618357]
x: [3.99275362 2.95410628 2.16183575 0.96618357]
computed b: [29. 31. 26. 19.]
real b: [29 31 26 19]
```

Tri par fusion

Dans le tri par fusion, comment peut-on efficacement paralléliser les dernières étapes sur un système à mémoire partagée?

Tri par fusion

La décomposition d'un vecteur de longueur n, à trier sur m coeurs, en m partitions égales de n/m éléments, est triviale. Ensuite, chaque partition peut être triée par un fil d'exécution en parallèle. La difficulté est de paralléliser la phase de fusion. Si on fait une fusion récursive, deux partitions à la fois, en log(m) étapes, il y aura la moitié des coeurs inactifs à la première étape, puis les trois quarts... alors que la durée augmente avec le nombre de coeurs libres. Différentes alternatives sont possibles, certaines demandent de choisir un pivot et déplacer les éléments comme pour le tri rapide. Une variante est d'échanger ses éléments (chacun partage ses n/m éléments avec un autre, l'un retient les n/m plus petits et l'autre les n/m plus gros) avec un voisin, en suivant un schéma comme la barrière en papillon.

```
 1  2| 3  4| 5  6| 7  8| 9 10|11 12|13 14|15 16  valeurs initiales
 2  1| 4  3| 6  5| 8  7|10  9|12 11|14 13|16 15  tri des partitions
 4  3| 2  1| 8  7| 6  5|12 11|10  9|16 15|14 13  partage avec i + 1
 8  7| 6  5| 4  3| 2  1|16 15|14 13|12 11|10  9  partage avec i + 2
16 15|14 13|12 11|10  9| 8  7| 6  5| 4  3| 2  1  partage avec i + 4
```

Tri par paquets

Comment peut-on paralléliser le tri par paquets sur un système à mémoire partagée?

Tri par paquets

Comment peut-on paralléliser le tri par paquets sur un système à mémoire partagée?

Dans la première phase, il faut répartir les n éléments entre m coeurs. Ensuite, chaque coeur trie ses éléments et le travail est terminé. Le défi, comme pour le tri rapide, est d'avoir une répartition équitable des éléments. Si on connait l'intervalle et que la distribution est uniforme, on peut diviser l'intervalle en m. Si on ne connait pas l'intervalle mais que la distribution est uniforme, on peut faire une recherche parallèle pour le minimum et le maximum, avec une réduction, et ensuite diviser l'intervalle en m. Une bonne solution est de prendre un échantillon uniforme des n éléments (e.g. 1/50), de trier ces échantillons, et de diviser le vecteur d'échantillons en m partitions. L'élément à la frontière de chaque partition est un des m - 1 pivots requis.