Compte rendu du projet de calcul parallèle

Résumé

Cette année, le devoir d'algorithmique parallèle a consisté à paralléliser un code de factorisation de matrices tri-diagonales par blocs. La méthode employée était celle de Jacobi par blocs et s'appuyait sur les bibliothèques scientifiques `Lapack` et `BLAS`. La parallélisation devait s'appuyer sur `MPI` (calcul distribué). Il était demandé aux étudiants de présenter leurs algorithmiques, leurs choix d'implantation ainsi qu'une étude de performance (sur `GDSDMI`). Il s'agissait d'un travail en binôme.

Ce document est accessible en ligne : http ://graal.ens-lyon.fr/ eagullo/algopar/tds/dm.corr.ps.

Rendus

Trois groupes se détachent par leur approche davantage scientifique que purement scolaire. Quatre groupes ont effectué un travail intéressant mais avec quelques points faibles concernant soit la rédaction, soit l'étude de performance. Deux groupes ont rendu un rapport très clair mais sans étude de performance. Deux groupes n'ont rendu que leur code (brut ou remis en forme) sans effort de rédaction.

Deux points ont régulièrement porté à confusion :

- Plusieurs groupes ont mélangé algorithme pour machine à mémoire distribuée et algorithme pour machine à mémoire partagée. En pratique, cela s'est traduit par l'inclusion de boucles *do ... par*.
- De même, plusieurs groupes ont confondu les topologies logique et physique du réseau et ont justifié l'efficacité pratique de leur algorithme en se basant sur certaines propriétés du réseau logique sous-jacent (anneau bi-directionnel).

Codes

De nombreux groupes ont à juste titre constaté que selon les paramètres d'entrée du problème la parallélisation du code n'améliorait pas nécessairement les performances sur la machine à leur disposition. En effet, `GDSDMI` dispose d'un switch 100 Mb Ethernet conduisant en pratique à des communications point à point de seulement environ 11 Mo/s.

L'évaluation du code a donc été effectuée sur une machine parallèle (http ://www.idris.fr/su/Scalaire-/zahir/) disposant d'un véritable réseau rapide. Voici ses principales caractéristiques [1] :

- 1024 processeurs Power4 et Power4+
- 3136 Goctets de mémoire
- 6,55 Tflops de performance crête totale
- Réseau Federation double lien (200 Mo/s par lien)

L'évaluation du code ne tient compte ni des avertissements ni des erreurs à la compilation. Seules les erreurs à l'exécution (erreurs de segmentation, deadlocks, nombre d'itération différent du cas séquentiel, norme finale fausse) ont été (légèrement) pénalisées. Néanmoins elles ont été corrigées afin de pouvoir procéder à l'évaluation de performances. Les (deux) codes effectuant une erreur de segmentation casi-systématique n'ont pas été corrigés.

Évaluation de performance

L'évaluation de performance a été effectuée sur la machine décrite précédemment. Plusieurs tests ont été effectués avec différents paramètres d'entrée. Quelque soit le nombre de processeurs utilisé, chaque processeur dispose de 3,5 Go de mémoire adressable au niveau utilisateur. Un processus `MPI` est utilisé par processeur.

Trois séries d'expérimentations ont été effectuées. Au sein de chaque série, les même paramètres d'entrée (conditionnement de la matrice, taille M d'un bloc) sont appliquées à tous les tests, et seul le nombre de processeurs (et donc le nombre de blocs) varie.

Chaque test correspond à une seule et même exécution. Pour ce faire, un seul exécutable génère une matrice (fonction des paramètres d'entrée), effectue un appel au code séquentiel (*jacseq*), puis un appel à un code parallèle naïf, et enfin une série d'appel à chacun des codes parallèles rendus par les étudiants (leur *jacpar*). L'appel à un code parallèle naïf avant l'appel aux codes des étudiants permet d'éviter de pénaliser le code exécuté en premier (il se peut que les premières communications aient un léger surcoût).

[1] http ://www.idris.fr/comp/scal/power4/zahir/index-zahir.html

	Nombre de processeurs				
Groupe	2	4	8	16	moyenne
NAIF	0.82	0.60	0.47	0.24	**0.53**
CHIS	0.92	0.90	0.88	0.64	**0.83**
PUJOL	1.03	0.99	0.98	0.86	**0.96**
HATAT	1.07	0.98	0.97	0.85	**0.97**
PALKA	0.97	0.93	0.91	0.77	**0.89**
BRAIBANT	1.01	0.99	0.97	0.86	**0.96**
FAWZI	0.90	0.63	0.42	0.22	**0.55**
PLANUL	0.90	0.82	0.59	0.29	**0.65**
COHEN	0.72	0.54	0.48	0.33	**0.52**
NIVOL	1.03	0.99	0.95	0.78	**0.94**

TAB. 1 – Efficacité (M=1000, Bon conditionnement)

D'autre part, l'algorithme naïf est un bon indicateur dans la mesure où il recentralise toute l'information sur le processeurs maître entre chaque itération. Une efficacité proche de celle l'algorithme naïf correspond donc à une mauvaise efficacité de la parallélisation.

Plusieurs indicateurs permettent de juger de l'efficacité d'un code. Ici sont présentés l'efficacité e et l'efficacité mémoire e_{mem}. Pour rappel :

$$e = \frac{T_{seq}}{p \times T_{par}} \tag{1}$$

Une efficacité de 1 avec 8 processeurs signifie que le code parallèle est 8 fois plus rapide que le code séquentiel (*i.e.* possède un facteur d'accélération égal à 8). Notons que certains groupes ont un code avec une *efficacité* proche de 1 sur cette machine (et ce quels que soient les paramètres d'entrée), tandis que sur `GDSDMI` plusieurs groupes n'ont réussi à obtenir un *facteur d'accélération* égal à 1 sur aucun problèmes ! Il y a machine parallèle et machine parallèle.

Nous définissons l'efficacité mémoire e_{mem} de même, en ce basant sur la valeur de consommation mémoire la plus élevée de tous les processeurs (celle qui est critique sur une architecture homogène) :

$$e_{mem} = \frac{M_{seq}}{p \times M_{par\text{-}max}} \tag{2}$$

Efficacité

Une première série d'expérimentations a permis de vérifier l'efficacité des codes sur des problèmes relativement petits ($M = 1000$) et bien conditionnés (la matrice étant "largement" à diagonale strictement dominante). Les résultats (voir Table 1) donnent une bonne idée de l'efficacité de la distribution des données étant donné que la convergence est très rapide (2 itérations sur 2 processeurs, 3 itérations sur davantage de processeurs).

Une deuxième série met davantage en avant la partie itérative de l'algorithme. Pour ce faire, la matrice générée est mal conditionnée et le critère de convergence plus exigeant. Le nombre d'itérations nécessaires pour converger est compris entre 15 (sur 2 processeurs) et 20 (16 processeurs). Les résultats (voir Table 2) montrent que certains codes se comportant très bien lorsque le nombre d'itérations est faible sont davantage mis en difficultés quand celui-ci augmente. Un groupe obtient une efficacité de 1.16 mais un résultat faux ....

Enfin une troisième série traite des problèmes relativement gros ($M = 2500$). Par exemple avec 16 blocs par ligne, la matrice A occupe une mémoire de 2.3 Go à elle seule. Les résultats sont présentés Table 3. Les matrices sont relativement mal conditionnées (elles ne sont pas à diagonale strictement dominante) mais en pratique la convergence est rapide (3 itérations).

Efficacité mémoire

L'efficacité mémoire est présentée Tables 4, 5 et 6 pour les trois séries évoquées ci-avant. Une efficacité mémoire de 0.25 sur 8 processeurs signifie que *jacpar* alloue sur un processeur une mémoire égale à 1/2 ($1/2 = 1/(0.25 \times 8)$) celle allouée par *jacseq*. De même, une efficacité mémoire de 0.25 sur 16 processeurs signifie que *jacpar* alloue sur le processeur le plus critique une mémoire égale à 1/4 ($1/4 = 1/(0.25 \times 16)$) celle allouée par *jacseq*.

	Nombre de processeurs				
Groupe	2	4	8	16	moyenne
NAIF	0.69	0.54	0.27	0.15	**0.41**
CHIS	0.86	0.73	0.66	0.65	**0.72**
PUJOL	1.02	0.95	0.83	0.82	**0.90**
HATAT	0.99	1.03	0.85	0.84	**0.93**
PALKA	0.90	0.96	0.83	0.71	**0.85**
BRAIBANT	0.89	0.90	0.85	0.75	**0.84**
FAWZI	0.98	0.76	0.58	0.37	**0.67**
PLANUL	1.38	1.48	1.10	0.67	**1.16**
COHEN	0.63	0.65	0.57	0.46	**0.58**
NIVOL	0.83	0.87	0.87	0.77	**0.84**

Tab. 2 – Efficacité (M=2000, Mauvais conditionnement)

	Nombre de processeurs				
Groupe	2	4	8	16	moyenne
NAIF	0.80	0.76	0.58	0.38	**0.63**
CHIS	0.93	0.86	0.89	0.81	**0.87**
PUJOL	1.02	0.93	0.90	0.84	**0.93**
HATAT	0.99	0.95	0.91	0.86	**0.93**
PALKA	0.92	0.90	0.87	0.82	**0.88**
BRAIBANT	1.00	0.90	0.89	0.87	**0.92**
FAWZI	0.98	0.77	0.53	0.33	**0.66**
PLANUL	0.92	0.85	0.64	0.41	**0.71**
COHEN	0.82	0.74	0.66	0.50	**0.68**
NIVOL	1.03	0.99	0.91	0.86	**0.95**

Tab. 3 – Efficacité (M=2500, Conditionnement moyen)

Notez que la véritable scalabilité mémoire de la méthode vient avec *jacpar_distentries* et consiste à appeler *jacpar* avec une matrice initialement distribuée. Tous les groupes n'ayant pas implanté cette fonction, aucun résultat à ce propos n'est présenté ici.

	Nombre de processeurs				
Groupe	2	4	8	16	moyenne
NAIF	0.33	0.25	0.25	0.25	**0.27**
CHIS	0.33	0.25	0.25	0.25	**0.27**
PUJOL	0.33	0.25	0.25	0.25	**0.27**
HATAT	0.33	0.25	0.25	0.25	**0.27**
PALKA	0.20	0.20	0.20	0.20	**0.20**
BRAIBANT	0.33	0.25	0.25	0.25	**0.27**
FAWZI	0.17	0.14	0.14	0.14	**0.15**
PLANUL	0.25	0.25	0.25	0.25	**0.25**
COHEN	0.14	0.14	0.14	0.14	**0.14**
NIVOL	0.33	0.25	0.25	0.25	**0.27**

Tab. 4 – Efficacité mémoire (M=1000, Bon conditionnement)

	Nombre de processeurs				
Groupe	2	4	8	16	moyenne
NAIF	0.33	0.25	0.25	0.25	**0.27**
CHIS	0.33	0.25	0.25	0.25	**0.27**
PUJOL	0.33	0.25	0.25	0.25	**0.27**
HATAT	0.33	0.25	0.25	0.25	**0.27**
PALKA	0.20	0.20	0.20	0.20	**0.20**
BRAIBANT	0.33	0.25	0.25	0.25	**0.27**
FAWZI	0.17	0.14	0.14	0.14	**0.15**
PLANUL	0.25	0.25	0.25	0.25	**0.25**
COHEN	0.14	0.14	0.14	0.14	**0.14**
NIVOL	0.33	0.25	0.25	0.25	**0.27**

Tab. 5 – Efficacité mémoire (M=2000, Mauvais conditionnement)

	Nombre de processeurs				
Groupe	2	4	8	16	moyenne
NAIF	0.33	0.25	0.25	0.25	**0.27**
CHIS	0.33	0.25	0.25	0.25	**0.27**
PUJOL	0.33	0.25	0.25	0.25	**0.27**
HATAT	0.33	0.25	0.25	0.25	**0.27**
PALKA	0.20	0.20	0.20	0.20	**0.20**
BRAIBANT	0.33	0.25	0.25	0.25	**0.27**
FAWZI	0.17	0.14	0.14	0.14	**0.15**
PLANUL	0.25	0.25	0.25	0.25	**0.25**
COHEN	0.14	0.14	0.14	0.14	**0.14**
NIVOL	0.33	0.25	0.25	0.25	**0.27**

Tab. 6 – Efficacité mémoire (M=2500, Conditionnement moyen)