Rapport de Calculs Intensifs
Multiplication matrice-vecteur

Ayaz BADOURALY

24 février 2016

1 Introduction

Le problème posé est celui de la parallélisation et de la distribution de l'application d'une matrice à un vecteur. Il s'agit d'un problème courant à résoudre dans la vie d'un ingénieur ou d'un chercheur, notamment pour la résolution de problèmes différentiels.

Ce rapport a pour but de présenter le travail que j'ai effectué entre le vendredi 19 février et le mercredi 24 février. Il s'agit d'un support au code source lui-même. Les points les plus subtiles du code source sont expliqués sous forme de commentaires directement dans les fichiers sources.

2 Présentation du projet

Pour me rapprocher au plus près d'un cas réel avec distribution des données sur un cluster, j'ai choisi de séparer le projet en deux parties.

La première partie concerne la génération des données (matrice et vecteur à multiplier), et est exécutée par `bin/generator`. La seconde partie concerne le traitement des données et la multiplication en elle-même, exécutée par `bin/calculator`.

Concernant l'arborescence du répertoire, les fichiers sont séparés enre le dossier `src/` pour les sources principales (`src/generator.c` et `src/calculator.c`), le dossier `lib/` pour les librairies (`lib/file.c`, `lib/matrix.c` et `lib/vector.c`) et le dossier `tests/` pour les tests unitaires.

La compilation [1] est automatisée grâce à un `Makefile`.

3 Les librairies (`lib/`)

3.1 `lib/file.h`

Ce fichier contient toutes les fonctions utiles pour intéragir avec la mémoire dure : lecture du disque et écriture sur le disque.

3.2 `lib/matrix.h`

Ce fichier contient la définition de la structure `Matrix`. Elle contient deux entiers pour le nombre de lignes et le nombre de colonnes, et un tableau à deux dimensions alloué dynamiquement et qui contient les éléments de la matrice.

J'ai aussi écrit quelques méthodes, pour pouvoir intéragir plus facilement avec la structure de données.

1. *N.B. :* travaillant sous Linux, je garantis le fonctionnement sur cet OS — et plus généralement sur tout OS certifié POSIX — mais pas sur Windows (j'ai tout de même ajouté quelques instructions préprocesseurs concernant cet OS). Si besoin, je suis en mesure de fournir une machine Linux fonctionnelle.

3.3 `lib/vector.h`

Il s'agit d'une librairie analogue à `matrix.h`. Les structures sont proches et les méthodes quasiment identiques.

Ce fichier contient la définition de la structure `Vector`. Elle contient un entier pour le nombre composantes, et un tableau à une dimension alloué dynamiquement et qui contient les éléments du vecteur.
J'ai aussi écrit quelques méthodes, pour pouvoir intéragir plus facilement avec la structure de données.

3.4 À propos des éléments des matrices et des vecteurs

La façon la plus simple de procéder est de se limiter à des matrices et des vecteurs de `int`. C'est le type de données le plus courant et c'est d'ailleurs ce qui est choisi par défaut.

Néanmoins, j'ai écrit mon code de telle façon que l'on puisse choisir au moment de la compilation le type de données des matrices et des vecteurs. Le préprocesseur définit deux variables indiquant le type de données à utiliser : `TYPE_NUM` et `TYPE_ELMT`. Ces variables sont couplées suivant la règle :

— `TYPE_NUM = 0` $\Longleftrightarrow$ `TYPE_ELMT = int`
— `TYPE_NUM = 1` $\Longleftrightarrow$ `TYPE_ELMT = long`
— `TYPE_NUM = 2` $\Longleftrightarrow$ `TYPE_ELMT = float`
— `TYPE_NUM = 3` $\Longleftrightarrow$ `TYPE_ELMT = double`

On peut spécifier la macro `TYPE_NUM = <choix de l'utilisateur>` avec l'option `-D` du compilateur `gcc` (*cf.* `Makefile`).

À noter que conformément à ce qui est attendu, le benchmark montre que, à dimension équivalente, un problème sur des `int` est résolu beaucoup plus rapidement qu'on problème sur des `float`.

4 Les sources (`src/`)

Chacune des sources, contenue dans le dossier `src/`, utilise les librairies définies plus haut. Lorsqu'elles sont compilées, le `Makefile` crée les exécutables dans le dossier `bin/`.

J'ai choisi d'adopter une structure de communication maître/esclaves. À l'initiation de la session `MPI`, un processus `root` est défini et il agira en tant que maître. Il ne fait aucun calcul sur les données, il envoie seulement des paramètres et des ordres aux autres processus qui sont de fait esclaves.

4.1 `bin/generator`

Le fichier source `src/generator.c` génère un exécutable `bin/generator` charger de générer la matrice et le vecteur (que l'on multiplira dans un deuxième temps).

1. Le processus `root` calcule les tailles des données à traiter par chaque processus [2] (*cf. ligne 118*).
2. Puis il envoie un `int` contenant cette taille au processus chargé du calcul (*cf. ligne 119*).
3. Chaque processus esclave attend la taille des données qu'il aura à calculer ; une fois reçue, le processus crée une petite matrice aléatoire et un petit vecteur aléatoire (*cf. lignes 122 à 131*).
4. On écrit les données dans des fichiers dans `data/` (il y a une gestion des erreurs si on n'arrive pas à atteindre ce dossier, typiquement sous Windows). Ces fichiers sont communs à tous les processus (à la fin de l'exécution, on obtient trois fichiers `data/metadata`, `data/matrix` et `data/vector`). Pour assurer que les données seront écrites dans le bon ordre, on utilise le protocole suivant :
   — le processus `root` écrit les métadonnées dans `data/metadata` (*cf. lignes 146 à 150*) ;

2. Le calcul est fait de manière intelligente, pour distribuer le plus également possible des données. Par exemple, s'il y a 10 données à distribuer sur 4 processus esclaves, alors les tailles calculées seront : $3 - 3 - 2 - 2$.

— le processus `root` envoie un `ping` au processus `p0` (*cf. ligne 159*) et il attend un `pong` (*cf. ligne 160*) ;
— le processus `p0` attends un `ping` du processus `root` et le reçoit (*cf. ligne 171*) ;
— le processus `p0` ouvre les fichiers appropriés, écrit dedans et les referme (*cf. lignes 175 à 181*) ;
— le processus `p0` envoie un `pong` au processus `root` (*cf. ligne 184*) ;
— le processus `root` reçoit le `pong` (*cf. ligne 160*) ;
— le processus `root` envoie un `ping` au processus `p1` et il attend un `pong` ;
— le processus `p1` reçoit le `ping` ;
— le processus `p1` ouvre les fichiers appropriés, écrit dedans et les referme ;
— le processus `p1` envoie un `pong` au processus `root` ;
— le processus `root` reçoit le `pong` ;
— ...

Le programme peut être appelé avec plusieurs options :

`-h` affiche l'aide

`-n <dim>`
Prend en argument la taille de la matrice et du vecteur ;
Si non utilisée, la valeur `DEFAULT_DIM` est choisie

`-x <max_value>`
Prend en argument la valeur maximale des éléments de la matrice et du vecteur ;
Si non utilisée, la valeur `DEFAULT_MAX` est choisie

4.2 `bin/calculator`

Le fichier source `src/calculator.c` génère un exécutable `bin/calculator` charger de calculer le produit du vecteur `data/vector` par la matrice `data/matrix`.

1. Le processus `root` récupère la dimension de la matrice du fichier `data/metadata` (*cf. lignes 94 à 104*).
2. Le processus `root` calcule les tailles des données à traiter par chaque processus (dans la variable `recv_counts`) et l'offset associé (dans la variable `displs`) (*cf. lignes 138 à 153*).
3. Le processus `root` envoie toutes les données à tous les processus (*cf. lignes 165 à 168*).
4. Les processus esclaves extraient les données dont ils ont besoin des fichiers `data/matrix` et `data/vector` (*cf. lignes 179 à 184*).
5. Les processus esclaves se transmettent le vecteur avec `MPI_Allgatherv` (*cf. lignes 186 à 194*).
6. Les processus esclaves calculent chacun le produit de leur sous-matrice avec le vecteur (*cf. lignes 198 à 200*).
7. Les données calculées sont enregistrées en suivant le protocole décrit précédemment dans le fichier `data/result` (*cf. lignes 216 à 260*).
8. Si le mode *verbose* est activé, les processus esclaves envoient les données calculées au processus `root`, en les ordonnant avec `MPI_Gatherv`, et il se charge de l'afficher dans `stdout` (*cf. lignes 267 à 288*).

Le programme peut être appelé avec plusieurs options :

`-h` affiche l'aide

`-v` active le mode *verbose*

`-b <buffer_size>`
Prend en argument la taille du buffer [3] ; c'est notamment utile lorsque la taille de la matrice est importante et que le parsing du fichier `data/matrix` échoue ;
Si non utilisée, la valeur `BUFSIZ` est choisie

3. Elle est sauvegardée en mémoire dans la variable globale `buffer_size` — cette forme de variable est la plus simple à utiliser, car elle est utilisée dans plusieurs fichiers du projet et la passer en argument des fonctions l'utilisant s'avère être lourd en termes d'écriture.

4.3 Codes d'erreur

J'ai suivi une convention commune à tous les fichiers de `src/`. Si l'exécution se passe bien, une fonction va retourner `EXIT_SUCCESS`, c'est-à-dire `0` sur la majorité des OS. Sinon :

code d'erreur 1
une erreur a été détectée sur les paramètres envoyés au programme

code d'erreur 2
une erreur a été détectée lors d'un traitement sur un fichier (ouverture, lecture, etc...)

Dans tous les cas, l'erreur est accompagnée d'un message explicatif pour facilité le débuggage.

Une erreur provoque l'arrêt immédiat de tous les proccessus par un `MPI_Abort`.

5 Les data (`data/`)

Le dossier `data/` contient toutes les données écrites en dur sur le disque [4].

5.1 `data/metadata`

La première ligne de ce fichier contient le nombre de lignes et le nombre de colonnes de la matrice générée par `bin/generator`. Ils sont séparés par un espace.

Les deux lignes suivantes contiennent les chemins absolus dans l'arborescence du système de fichiers des fichiers `data/matrix` et `data/vector`.

5.2 `data/matrix`

Ce fichier contient la matrice générée aléatoirement par `bin/generator` au format brut. Chaque ligne correspond à une ligne de la matrice. Les éléments d'une même ligne mais de différentes colonnes sont séparés par un espace.

5.3 `data/vector`

Ce fichier contient la matrice générée aléatoirement par `bin/generator` au format brut. Chaque ligne correspond à une composante du vecteur.

5.4 `data/result`

Ce fichier est le seul à ne pas être généré par `bin/generator` mais par `bin/calculator`.

Il contient le vecteur, produit de la matrice `data/matrix` et du vecteur `data/vector`. Il est enregistré au format brut. Chaque ligne correspond à une composante du vecteur.

6 Les tests (`tests/`)

Le dossier `tests/` contient deux fichiers :

— `t_matrix.c` qui contient tous les tests unitaires de la librairie `lib/matrix.h`
— `t_vector.c` qui contient tous les tests unitaires de la librairie `lib/vector.h`

4. Ceci n'est vrai que pour un OS de type Linux. Sinon, il s'agit du dossier d'exécution des programmes et les fichiers sont suffixés avec `.txt`.

7 Conclusion

Le code présenté répond effectivement au cahier des charges. On pourrait apporter de nombreuses améliorations pour une faire une librairie de calcul matriciel complète. Notamment, on pourrait rajouter la génération de matrices non carrée (théoriquement possible avec la `lib/matrix.h` et utilisée dans `src/calculator.c`, mais non implémenté dans `src/generator.c`), ou la multiplication de deux matrices — et la multiplication d'un vecteur par une matrice ne serait alors plus qu'un cas particulier.