Rapport de projet

Système distribué

Implémentation d'une version distribuée de *GNU make*

NGUYEN Kim Thuat - NGUYEN Dang Chuan
ENSIMAG - ISI3A

Grenoble, le 3 février 2013

I. Introduction

A l'époque ou le gros calcul devient de plus en plus fréquent, une seule machine ne peut plus supporter le travail. De plus, on peut perdre énormément de temps pour faire une grosse tâche avec une seule machine. Il est nécessite donc le fait de diviser une grosse tâche et distribuer les petites tâches sur plusieurs machines.

Dans ce document, nous allons présenter une version distribuée de l'utilitaire make GNU. Le programme se concentre essentiellement sur la partie performance et la répartition des tâches à distance au lieu de la lecture des *Makefile* complexes.

II. Architecture logiciel

1. Choix technique

1.1. Langage de programmation

Nous utilisons le langage C++ pour la programmation. L'objectif est pour avoir plus de performance dans le calcul. De plus, l'API pour gérer la communication entre les machines qu'on utilise dans le cadre de ce projet est très bien supportée en C++.

1.2 Gestion de communication entre machines

Pour les calculs parallèles, on utilise donc la bibliothèque MPI qui permet de réaliser facilement un programme distribué qui peut utiliser plusieurs machines pour le calcul. MPI permet alors de se passer de la gestion de la couche réseaux du programme, et gère automatiquement la création des processus répartis sur les machines.

2. Algorithme

2.1 Quelques définitions

Dans notre programme, on définit une structure qui s'appelle "*Rule*" pour décrire la *Makefile.* Chaque règle contient certaines propriétés comme vous pouvez voir dans l'image ci-dessous.

```
/* Struct Rule pour presenter une ligne (une regle) du MakeFile
 * Une regle depend aux autre regles et a aussi des dependances
 */
struct Rule {
        int idRule;
        string name;
        vector<string> command; // La ligne de la commande suivant une regle
        bool isExecute;
        bool isFinished;
        list<string> dpNames; // Enregistre les noms des fichiers necessaires
        list<Rule*> dependences; // Les regles que cette regle depend
        list<Rule*> dependants;  // Les regles qui depandent de cette regle
        Rule (string ruleName): isExecute(false), isFinished(false), name(ruleName) { }
};
```

Une règle décrit une ligne dans le fichier Makefile qui contient un identifié, un nom (la cible), une liste de dépendances ainsi que une liste des règles qui dépendent à elle. Il a aussi quelques variables booléens pour indiquer son état. Grâce à ces variables, on sait bien si une ligne est exécutée, ou bien si elle est terminée.

2.2 Architecture du programme

Notre programme utilise un algorithme simple de type maître/esclave qui assure l'équilibrage à charge statique. On peut décrire facilement l'algorithme ainsi que la façon que le maître communique avec les esclaves:

- Le programme tout d'abord parse le fichier *Makefile* a l'entrée selon la cible donnée. Il enregistre tous les règles dépendantes dans une liste. Le maître après utilise cette liste pour distribuer les tâches.

- Après avoir reçu tous les tâches à faire, le maître va donner dans un premier temps une tâche à chaque esclave. Au cas où le nombre de machine est plus grand que le nombre de tâches, certaines machines seraient étendues tout de suite car elles ne reçoivent aucune tâche.

- Le maître écoute tous les esclaves. Un esclave envoie une notification pour le maître des qu'il finit une tâche donnée. Le maître traite le message, détecte l'origine de message et envoie une autre tâche pour cet esclave en condition qu'il reste encore des tâches à faire.

- Si un esclave reçoit une tâche qui devrait faire après quelques tâches dépendantes, il va notifier le maître, le maître ensuite vérifie si toutes les tâches dépendantes de celui-ci sont déjà faites. Finalement, il renvoie les fichiers nécessaires à l'esclave pour qu'il puisse finir sa tâche.

- Quand il reste plus de tâche à faire, le maître va envoyer un message de type *"broadcast"* à tous les esclaves. Les derrières reçoivent le message et s'arrêtent tout de suite.

On vous donne ci-dessous l'algorithme simplifié du programme:

```
master() {
        // Le maitre envoie une tache a chaque esclave.
        // if (nbMachine>tasksTodo) alors on eteint certaines machines
        distributeTaskToSlave();

        while (tasksTodo>0) {
                // Écouter les messages des esclave
                // if (message = FINISHED_TAG) alors on lui attribute une nouvelle tache
                // if (message = NEED_FILE_TAG) le maître vérifie si les fichiers nécessaires sont //déjà
                disponibles en local. Si tous les fichiers demandés sont disponibles, il les renvoie une //
                fois a  l'esclave utilisant le protocole scp.
                receiveMessageFromSlave();
        }

        // Lorsque tous les taches sont envoyés, le maître attend les derrières messages des
        // esclaves
        receiveOtherMessages();

        // Le maître envoie un message pour informer aux esclaves que le travail est fini
        sendDieTaskToEsclave();
}

slave() {
        // Recevoir des messages du maître
        receiveMessageFromMaster();

        if (taskTodo==0) alors il sort de la boucle
        if (mpi_tag = DIE_TAG) le travail est fini. L'esclave s'arrête;

        // Si non, esclave fait la tache et revoie le résultat au maître
        do_the_work();
}
```

3. Manuel d'installation

3.1 Techonologies requises

- Open MPI 1.4.5+

3.2 Structure de l'archive d'installation

```
makefileENSI2013/
        sdmake.cpp
        sdmake.h
        TestProrams/
```

matrix/
premier/
blender_2.49
blender_2.59
myhosts
deploy.sh

- sdmake.cpp et sdmake.h : les deux fichiers principale du programme
- TestPrograms: contient les 4 programmes de test
- myhosts: presente la liste de hosts a tester
- deploy.sh: script pour préparer l'environnement de test dans chaque répertoire /tmp/makefile d'une hôte

3.3 Déploiement du test de performance

- Configurer le PATH:
  Lancer cette commande avant de faire le test ou l'ajoute dans le fichier .bashrc
  Supposons que make_dir est la place ou vous deposez le programme make distribué
  Dans notre cas: make_dir=/home/nguyend/partage/sysdis/makefilesENSI2013

PATH=make_dir/TestPrograms/matrix:make_dir/TestPrograms/premier:make_dir:$PATH

Ce qu'on fait ci-dessus est pour objectif d'ajouter tous les fichiers exécutables nécessaires dans le PATH partage. Ce qui nous permet de lancer le programme dans le répertoire /tmp/makefile de chaque hote sans ajouter le **./**

On ajoute aussi une autre fois le chemin vers le répertoire matrix parce qu'il existe aussi les deux programmes *split* et *multiply* dans les machines TX. Comme ça, on force l'utilisation des fichiers exécutables placés dans le répertoire matrix.

- Lancer le script deploy.h:
  ./deploy.sh

  Ex: voir le fichier deployTrace.txt
  Le script va tout d'abord créer le répertoire */tmp/makefile* dans chaque machine hôte. Il va ensuite copier tous les programmes de test dans ces répertoires. On copie également le fichier *myhosts* dans le répertoire de chaque programme de test.

Structure du répertoire /tmp/makefile sur chaque hôte:

/makefile
blender_2.49/
…….....
myhosts
blender 2.59/

…….....
myhosts
premier/
…….....
myhosts
matrix/
…….....
myhosts

- Lorsque l'environnement est prêt. Il est suffit d'aller dans la machine maitre (le premier hôte dans le fichier myhosts) et lancer la commande:

mpirun -x PATH -hostfile myhost sdmake -nkt Makefile > resultat.txt

- Pour compiler le programme, il faut aller dans le répertoire ou vous déposez le programme et lancer la commande ci-dessous:

mpic++ sdmake.cpp -o sdmake

Notes: s'il y a des erreurs pendant l'exécution, il est suffit de relancer la commande mpirun ci-dessus. Si tous va bien, vous pouvez voir le temps d'exécution dans le fichier temps_execution.txt qui est généré à la fin de calcul dans la machine maître.

III. Test de performance

1. Méthodologie

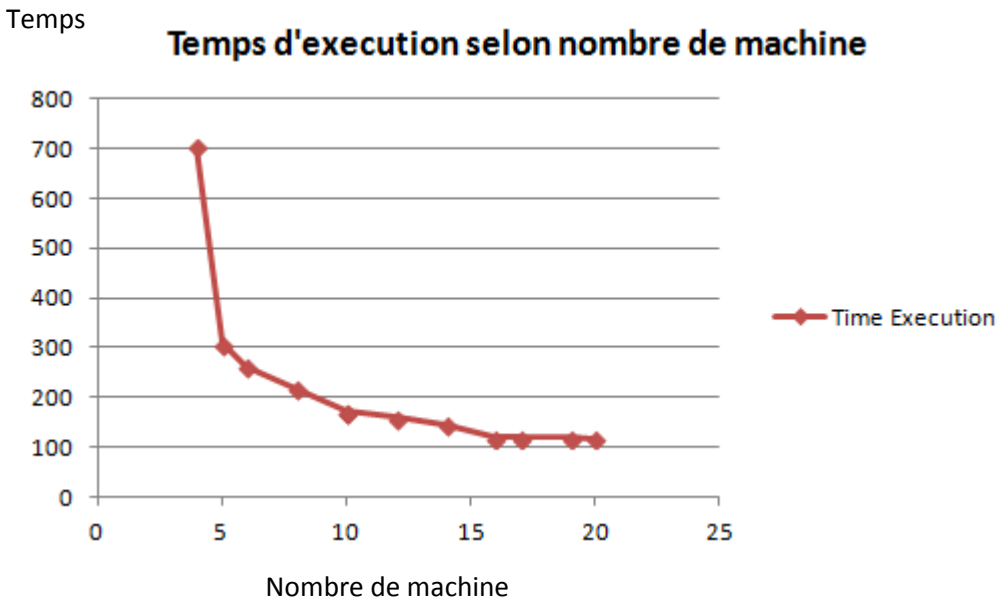
Pour apprendre bien l'efficacité du calcul parallèle, on observe le changement du temps de calcul en augmentant le nombre de machines et le volume des calculs.
Chaque fois, on fixe le nombre de machines et implémente quelques tests pour trouver l'intervalle de confiance. Autrement dit, on peut savoir approximativement le temps de calcul pour un nombre de machine précise.

2. Condition expérimentale

Afin d'expérimenter la nouvelle version de make distribuée, on a utilisé les machines dans la salle TX de notre école. Lorsqu'on augmente le nombre de machine, les machines ne sont pas vraiment stables. C'est à dire, le programme peut tomber des fois mais le problème n'est pas lie au programme lui-même. Il faut le relancer.

3. Résultats obtenus

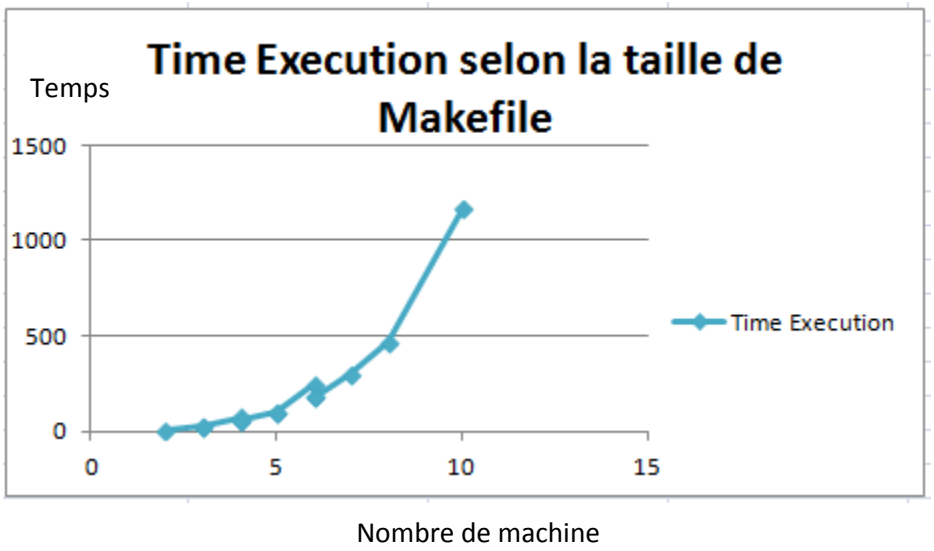
3.1 Premier



Le dessin ci-dessus présente le temps d'exécution pour finir le calcul des premiers selon le nombre de machines. On voit facilement que le temps de calcul diminue notablement quand le nombre de machines augmente.

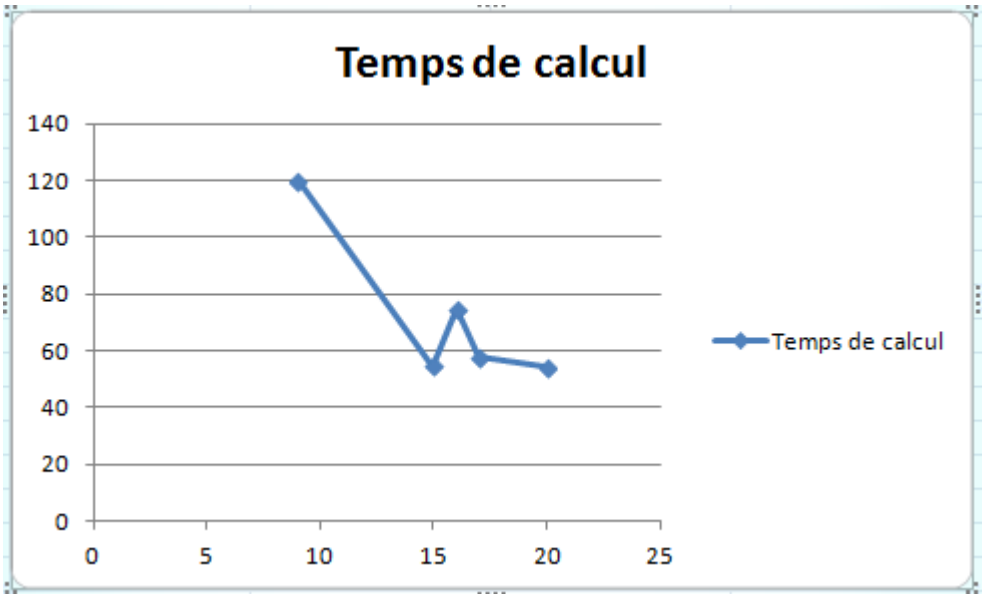
Dans ce cas, on teste ce programme jusqu'à 20 machines car le fichier Makefile contient seulement 20 lignes. Même si l'on utilise plus de 20 hôtes, il y aura uniquement 20 machines parmi eux seront utilisée. En faisant des expérimentations sur 20 machines, on conclut que l'intervalle de confiance à 95% vaut alors [118.5, 119.04]. Autrement dit, on est sur a environ 95% que le programme sera terminé entre 118.5 et 119.04 secondes.

3.2 Matrice



Dans ce graphe, on présente le changement de temps de calcul selon la taille de Makefile. Le nombre de machine dans ce cas est fixe de 15 à 17 hôtes. Le résultat qu'on a obtenu est correct. En effet, la taille de Makefile (généré par la commande: ./generate_makefile.pl "taille") augmente beaucoup après chaque modification. Il y aura énormément de dépendances à calculer pour chaque règle. Alors, la communication entre les machines devient plus fréquent qui augmente aussi le transfert des donnes entre eux.

Dans le deuxième graphe, on fixe la taille de Makefile. Le temps d'exécution sera observé selon le changement du nombre de machines.



On voit bien que le temps de calcul diminue dans la plupart de cas.

3.3 Blender

a. Blender 2.49

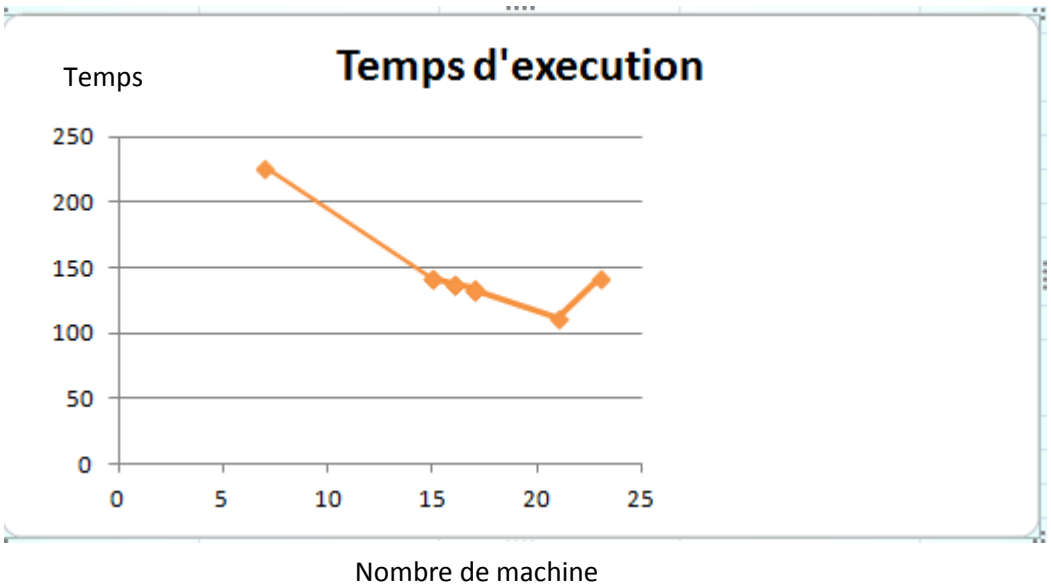
Pour ce programme, on a rencontré un problème avec le command *convert* qui est la derrière commande pour générer le fichier *cube.mpg*. On a arrive à calculer tous les fichiers dépendants mais le fichier *cube.mpg* est toujours vide comme indique dans l'image ci-dessous :

```
nguyend@ensipc42:/tmp/makefile/blender_2.49$ mpirun -x PATH -hostfile myhosts sdmake -nkt Makefile > result.txt
AL lib: pulseaudio.c:612: Context did not connect: Access denied
AL lib: pulseaudio.c:612: Context did not connect: Access denied
AL lib: pulseaudio.c:612: Context did not connect: Access denied
AL lib: pulseaudio.c:612: Context did not connect: Access denied
convert: Échec de la délégation `"ffmpeg" -v -1 -mbd rd -trellis 2 -cmp 2 -subcmp 2 -g 300 -pass 1/2 -i "%M%%d.jpg" "%u.%m" 2
> "%Z"' @ error/delegate.c/InvokeDelegate/1058.
```

```
nguyend@ensipc42:/tmp/makefile/blender_2.49$ ls
Makefile          frame_106.png  frame_19.png  frame_33.png  frame_48.png  frame_62.png  frame_77.png  frame_91.png
Makefile-recurse  frame_107.png  frame_2.png   frame_34.png  frame_49.png  frame_63.png  frame_78.png  frame_92.png
MakefileTest      frame_108.png  frame_20.png  frame_35.png  frame_5.png   frame_64.png  frame_79.png  frame_93.png
README            frame_109.png  frame_21.png  frame_36.png  frame_50.png  frame_65.png  frame_8.png   frame_94.png
cube.mpg          frame_11.png   frame_22.png  frame_37.png  frame_51.png  frame_66.png  frame_80.png  frame_95.png
cube_anim.blend   frame_110.png  frame_23.png  frame_38.png  frame_52.png  frame_67.png  frame_81.png  frame_96.png
cube_anim.zip     frame_111.png  frame_24.png  frame_39.png  frame_53.png  frame_68.png  frame_82.png  frame_97.png
file.sh           frame_112.png  frame_25.png  frame_4.png   frame_54.png  frame_69.png  frame_83.png  frame_98.png
frame_1.png       frame_113.png  frame_26.png  frame_40.png  frame_55.png  frame_7.png   frame_84.png  frame_99.png
frame_10.png      frame_12.png   frame_27.png  frame_41.png  frame_56.png  frame_70.png  frame_85.png  myhosts
frame_100.png     frame_13.png   frame_28.png  frame_42.png  frame_57.png  frame_71.png  frame_86.png  result.txt
frame_101.png     frame_14.png   frame_29.png  frame_43.png  frame_58.png  frame_72.png  frame_87.png
frame_102.png     frame_15.png   frame_3.png   frame_44.png  frame_59.png  frame_73.png  frame_88.png
frame_103.png     frame_16.png   frame_30.png  frame_45.png  frame_6.png   frame_74.png  frame_89.png
frame_104.png     frame_17.png   frame_31.png  frame_46.png  frame_60.png  frame_75.png  frame_9.png
frame_105.png     frame_18.png   frame_32.png  frame_47.png  frame_61.png  frame_76.png  frame_90.png
nguyend@ensipc42:/tmp/makefile/blender_2.49$ ls -la
total 41708
drwxr-xr-x 2 nguyend etudiants     2500 févr.  7 11:46 .
drwxr-xr-x 6 nguyend etudiants      120 févr.  7 11:45 ..
-rw-r--r-- 1 nguyend etudiants    13449 févr.  6 09:25 Makefile
-rw-r--r-- 1 nguyend etudiants    13677 févr.  6 09:25 Makefile-recurse
-rw-r--r-- 1 nguyend etudiants      487 févr.  7 11:40 MakefileTest
-rw-r--r-- 1 nguyend etudiants      155 févr.  6 09:25 README
-rw------- 1 nguyend etudiants        0 févr.  7 11:46 cube.mpg
-rw-r--r-- 1 nguyend etudiants   582388 févr.  7 11:45 cube_anim.blend
-rw-r--r-- 1 nguyend etudiants    54596 févr.  6 09:25 cube_anim.zip
-rwxr-xr-x 1 nguyend etudiants      187 févr.  7 04:35 file.sh
-rw-r--r-- 1 nguyend etudiants    13910 févr.  7 11:45 frame_1.png
-rw-r--r-- 1 nguyend etudiants    18537 févr.  7 11:45 frame_10.png
-rw-r--r-- 1 nguyend etudiants    24733 févr.  7 11:45 frame_100.png
-rw-r--r-- 1 nguyend etudiants    24733 févr.  7 11:45 frame_101.png
-rw-r--r-- 1 nguyend etudiants    24733 févr.  7 11:45 frame_102.png
-rw-r--r-- 1 nguyend etudiants    24733 févr.  7 11:45 frame_103.png
-rw-r--r-- 1 nguyend etudiants    24733 févr.  7 11:45 frame_104.png
-rw-r--r-- 1 nguyend etudiants    24733 févr.  7 11:45 frame_105.png
```

b. Blender 2.59

Le graphe ci-dessous montre le temps de calcul pour obtenir le vidéo out.avi avec le changement du nombre de machine.



On voit bien que le temps de calcul diminue rapidement si l'on augmente le nombre de machines. Cependant, quand il y a trop de machines, le temps peut-être augmente. Le résultat ne semble pas juste car on donne plus de ressource mais on prend plus de temps pour finir la tâche. En réalité, ce phénomène est explicable. Effectivement, le nombre de machines a un rapport avec l'augmentation du transfert des données. Le maitre doit peut-être traiter beaucoup plus de messages. C'est pour cela que le temps de calcul devient plus long.

4. Bilan

Après avoir expérimenté les tests de performance, nous voyons bien que l'augmentation du nombre de machines réduit le temps de calcul sensiblement. En revanche, ce temps ne diminue que lorsque le nombre de machines n'est pas trop grand en comparaison avec le nombre de tâches à faire. Effectivement, la communication et le transfert de données d'une part devient de plus en plus lourd quand le maître a trop de machine à gérer. D'autre part, s'il y a des machines qui tombent en panne au moment d'exécution, on devrait peut-être relancer tous les tâches. Il faut alors trouver le consensus entre le nombre de machines et la taille du travail pour économiser des ressources et diminuer le temps de calcul.

IV. Conclusion

En conclusion, on pourra donc affirmer que l'utilisation du calcul parallèle sera indispensable pour gagner le temps de calcul. Cependant, il faut parfois limiter le nombre de machine afin de ne pas dépenser trop de ressource et de diminuer le temps de calcul.