Idée de sujet SAE

F.Hoguin d'après MagPi

TP Python

Ce projet est tiré du magazine **MagPi** et utilise une grappe de plusieurs Raspberry Pi.

Recommandations

> Attention, il est possible que vous soyez obligés d'adapter les commandes qui sont présentes dans ce syllabus

Renommer les noeuds

Il faut renommer (`hostname`) de votre grappe de raspberry-pi :

- node1 (noeud principal ou maître)
- node2 (noeud secondaire ou ouvrier)
- node3 (noeud secondaire ou ouvrier)
- etc..

Clés ssh

- Chaque noeud ouvrier doit pouvoir communiquer avec le noeud principal sans avoir à communiquer de mot de passe. comme, nous allons utiliser `ssh`. Chaque noeud ouvrier va générer une paire de cléfs (public, privée) à l'aide de la commande suivante : `ssh-keygen -t rsa`.
- Les informations demandées ensuite ne sont pas utiles, vous pouvez valider sans préciser de pass phrase.
- Copier ensuite les clefs au noeud maître avec la commande `ssh-copy-id @IP-noeudmaître`.
- Faire la même choses sur le noeud maître et copier les cléfs sur tous les noeuds ouvriers.

MPI

- Le fonctionnement de la grappe repose sur MPI (Message Passing Interface).
- Ce protocole permet à plusieurs ordinateurs de se déléguer une tache et de répondre avec des résultats.
- Il faudra installer MPI sur chaque noeud. `apt install mpich python3-mpi4py`.
- Vérifier ensuite que MPI fonctionne sur chaque noeud : `mpiexec -n 1 hostname`.
- En principe le paramètre hostname renvoie le nom du noeud surlequel a été exécuté la commande

Premier test en grappe

- a partir du noeud maître, lancer la commande suivante :

```
mpiexec -n 4 --hosts @ip1,@ip2,@ip3,@ip4 hostname
```

- Vous devriez voir la liste des 6 noeuds s'afficher si tout va bien.

Programme python

Enregistrer le code suivant sur le noeud principal

```
from mpi4py import MPI
import time
import sys

# Attach to the cluster and find out who I am and how big it is
comm = MPI.COMM_WORLD
my_rank = comm.Get_rank()
cluster_size = comm.Get_size()

# Number to start on, based on the node's rank
start_number = (my_rank * 2) + 1

# When to stop. Play around with this value!
end_number = int(sys.argv[1])

# Make a note of the start time
start = time.time()

# List of discovered primes for this node
primes = []

# Loop through the numbers using rank number to divide the work
for candidate_number in range(start_number,
                              end_number, cluster_size * 2):

    # Log progress in steps
    # print(candidate_number)

    # Assume this number is prime
    found_prime = True

    # Go through all previous numbers and see if any divide without remainder
    for div_number in range(2, candidate_number):
        if candidate_number % div_number == 0:
            found_prime = False
            break

    # If we get here, nothing divided, so it's a prime number
    if found_prime:
        # Uncomment the next line to see the primes as they are found (slower)
        # print('Node ' + str(my_rank) + ' found ' + str(candidate_number))
        primes.append(candidate_number)

# Once complete, send results to the governing node
results = comm.gather(primes, root=0)

# If I am the governing node, show the results
if my_rank == 0:

    # How long did it take?
    end = round(time.time() - start, 2)
```

```
print('Find all primes up to: ' + str(end_number))
print('Nodes: ' + str(cluster_size))
print('Time elasped: ' + str(end) + ' seconds')

# Each process returned an array, so lets merge them
merged_primes = [item for sublist in results for item in sublist]
merged_primes.sort()
print('Primes discovered: ' + str(len(merged_primes)))
# Uncomment the next line to see all the prime numbers
# print(merged_primes)
```

- Le code python précédent est tiré de `magpi.cc/EWASJx`. Le programme précédent ne prend qu'un seul argument.
- Pour voir si le programme fonctionne, commencer par tester le code python uniquement sur le noeud principal : `mpiexec -n1 python3 prime.py 1000`

Propagation

- Il faut maintenant faire une copie du programme sur chaque noeud `scp prime.py @ip1`, `scp prime.py @ip2`, etc...
- Pour vérifier que cela a fonctionné, chaque noeud peut exécuter en standalone le programme comme pour le noeud principal.

Finalement

Il faut distribuer le calcul aux RPI, Le RPI maître va donc lancer une commande qui fera calculer une partie par chaque RPI, une fois terminé, les RPI renverront le résultat au RPI maître qui affichera le résultat.

Commande à exécuter :

```
mpiexec -n 4 --host @ip1,@ip2,@ip3,@ip4 python3 prime.py 100000
```

Rapport

Faites votre rapport