Techniques d'optimisation de la parallélisation

Examen écrit

Master 1 MIHPS

Nom : _____

Prénom : _____

I – QCM

Ce QCM comporte en tout 20 questions. Il n'y a pas de pénalité associée aux mauvaises réponses. **Si une question accepte plusieurs réponses, elle le mentionne explicitement en précisant leur nombre.** Dans les questions suivantes **seq(n)** correspond à la partie séquentielle, **par(n)** à la partie parallèle et enfin **comm(n,p)** aux communications **n'**étant la taille du problème et **p** le nombre de cœurs. De plus la notation **S(n,p)** est associée au speedup.

Question n°1 :	Les principaux facteurs limitant le parallélisme d'une application sont (**deux réponses attendues**)	
❑ Les communications	❑ Le compilateur	❑ Les threads
❑ La météo	❑ La fraction parallèle	❑ La fraction séquentielle
Question n°2 :	Deux boucles *for* imbriquées ont une complexité en ...	
❑ $\Theta(nln_2(n))$	❑ $\Theta(n)$	❑ $\Theta(n^2)$
❑ $\Theta(1)$	❑ $\Theta(\sqrt{(n)})$	❑ $\Theta(n^{\log(n)})$
Question n°3 :	Une recherche dans une liste doublement chaînée à une complexité en ...	
❑ $\Theta(nln_2(n))$	❑ $\Theta(n)$	❑ $\Theta(n^2)$
❑ $\Theta(1)$	❑ $\Theta(\sqrt{(n)})$	❑ $\Theta(n^{\log(n)})$
Question n°4 :	En MPI les communications collectives ont une complexité en ...	
❑ $\Theta(1)$	❑ $\Theta(n)$	❑ $\Theta(\sqrt{n})$
❑ $\Theta(nln(n))$	❑ $\Theta(e^n)$	❑ $\Theta(n^2)$
Question n°5 :	Si le déroulement d'un algorithme peut se représenter sous la forme d'un arbre binaire alors sa complexité est probablement proche de ...	
❑ $\Theta(nln_2(n))$	❑ $\Theta(n^2)$	❑ $\Theta(n)$
❑ $\Theta(1)$	❑ $\Theta(\sqrt{(n)})$	❑ $\Theta(n^{\log(n)})$
Question n°6 :	$S(n,p) \leq \frac{1}{s+\frac{(1-s)}{p}}$ *avec* $s = \frac{seq(n)}{seq(n)+par(n)}$ c'est ...	
❑ L'efficacité	❑ La loi de Gustafson-Barsis	❑ La loi de Amdahl
❑ La métrique de Karp-Flatt	❑ La formule du speedup	❑ L'équation de Maxwell

Question n°7 :	L'efficacité d'un programme se calcule en faisant :	
❑ $\frac{1}{S(n,p)}$	❑ $\frac{p}{S(n,p)}$	❑ $\sqrt{\frac{seq(n)}{par(n)}}$
❑ $\frac{seq(n)}{p}$	❑ $\frac{e^{seq(n)+\ comm(n,p)}}{p}$	❑ $\frac{S(n,p)}{p}$
Question n°8 :	**En vous servant de la formule de la question 6** quel est le speedup maximal que l'on peut espérer si 20 % du temps d'exécution est séquentiel ?	
❑ 0,25	❑ 0,5	❑ 2
❑ 4	❑ 5	❑ 20
Question n°9 :	D'après la loi d'Amdahl, le speedup d'un programme est limité par (**deux réponses attendues**) ...	
❑ La vitesse de la lumière	❑ La fraction séquentielle	❑ La fraction parallèle
❑ Les communications	❑ $\frac{1}{s}$	❑ Le programmeur
Question n°10 :	Un programme passe à l'échelle d'autant plus que ….	
❑ La machine est grande	❑ Son problème est petit	❑ Son problème est grand
❑ La fraction séquentielle est importante	❑ Le réseau de communication est lent	❑ Les communications sont prépondérantes
Question n°11 :	Du point de vue optimisation un programme parallèle est idéal si … (**deux réponses attendues**)	
❑ Sa complexité est exponentielle	❑ La fraction séquentielle est importante	❑ Il n'y a pas de communications
❑ Le problème est facilement divisible	❑ Sa scalabilité est limitée	❑ Les communications sont prépondérantes
Question n°12 :	Quels éléments parmi les suivants peuvent nuire à une mesure (**quatre réponses attendues**)	
❑ Utiliser un compteur précis	❑ La répéter de nombreuse fois	❑ La faire une seule fois
❑ Utiliser un compteur peu précis	❑ Mesurer un temps trop court	❑ Les autres programmes s'exécutant sur la machine
Question n°13 :	La scalabilité faible se calcule pour une taille de problème ...	
❑ variable	❑ très petite	❑ constante

❑ très grande	❑ proportionnelle au nombre de cœurs	❑ inversement proportionnelle

Question n°14 :	Lequel de ces programmes permet d'obtenir un profil des communications MPI ?	
❑ gdb	❑ mpich	❑ Scalasca
❑ gprof	❑ gcc	❑ valgrind
Question n°15 :	Lors d'une décomposition de domaine en 2X2 cartésien avec des conditions de bord « nulles » combien de communications MPI il y a t'il au total par phase d'échange ? Chaque nœud échange avec ses 8 voisins.	
❑ 4	❑ 8	❑ 12
❑ 16	❑ 18	❑ 32
Question n°16 :	Quel est l'ordre de grandeur de la latence d'une communication intra-nœud ?	
❑ 2 ns	❑ 500 ns	❑ 2 µs
❑ 500 µs	❑ 2 ms	❑ 500 ms
Question n°17 :	L'analyse d'un programme MPI met en évidence trois MPI_Bcast (voir annexes pour manuel de MPI_Bcast) lors de la phase d'initialisation. Que dois-je faire pour optimiser ce programme ?	
❑ Les supprimer	❑ les remplacer par des MPI_Isend et MPI_Irecv	❑ les remplacer par des MPI_Send et MPI_Recv
❑ Fuir	❑ Déplacer l'initialisation à la fin	❑ Rien, ils ne sont pas un problème,
Question n°18 :	Un programme MPI s'exécute en 25 secondes sur 8 cœurs et pour un même problème en 100 secondes sur 1 cœur, L'efficacité de ce programme sur huit cœurs et de :	
❑ 0,1	❑ 0,125	❑ 0,25
❑ 0,5	❑ 1	❑ 2
Question n°19 :	On dit qu'il y a false-sharing entre deux variables lorsqu'elles sont accédées simultanément par deux threads et qu' ...	
❑ elles sont sur la même ligne de cache	❑ elles sont sur deux lignes de caches voisines	❑ elles sont dans le même tableau

❑ elles ont la même adresse	❑ elles ont la même valeur	❑ elles ont même nom
Question n°20 :	Un processeur hyper-threadé ...	
❑ peut exécuter deux threads sur un même cœur physique	❑ a physiquement deux fois plus de cœurs	❑ a deux fois plus d'unités flottantes.
❑ est deux fois plus rapide	❑ peut changer de fréquence	❑ fait pour OpenMP

II – Questions

1) Quelle est la différence entre un spinlock et un mutex ? Dans quel cas de figure les utilisez-vous ?

2) Donnez un exemple de false-sharing.

3) Implémentez une réduction MPI en utilisant un arbre binaire (code C)
values : Valeur à réduire en somme.
size : nombre de valeurs.
comm : communicateur sur lequel effectuer la réduction.
root : rang du processus racine.

void my_reduce(int *values, int size, int root, MPI_Comm comm)

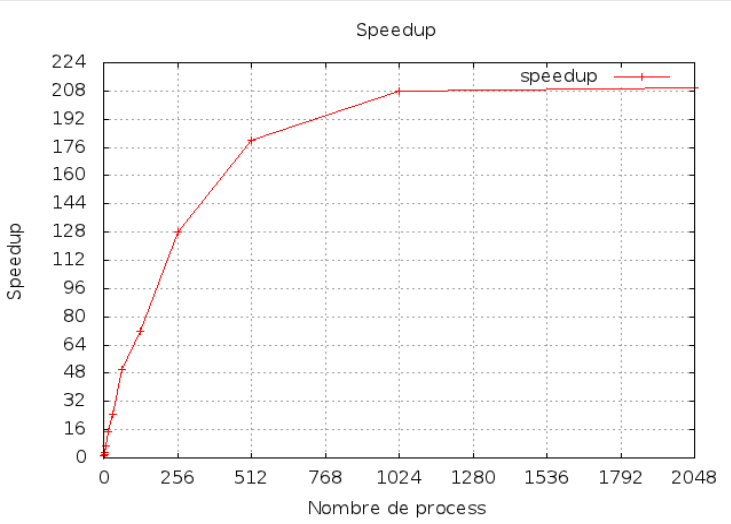
{

}

4)Définissez ce qu'est une architecture NUMA, quelles précautions doit-on prendre lorsqu'on les utilise en mode multi-threads ?

5) Quelle est la différence entre OpenMP et MPI ? Quel est l'intérêt pour les architectures modernes ?

5)



Répondez aux questions suivantes en utilisant cette courbe de speedup.

Quelle est l'efficacité de ce code sur 1024 cœurs ?

Comment pourriez-vous estimer la fraction séquentielle de ce programme ?

Que répondriez-vous au propriétaire de ce code vous expliquant que son code va plus vite sur 2048 cœurs que sur 512 ? Quelle métrique de performance utiliserez-vous donnez un exemple argumenté.

III – Critiques de codes

Dans cette partie vous allez devoir commenter des extraits de code en décrivant ce qui les rend peu performants et en proposant une correction améliorant leurs performances. Par souci de simplicité seules les parties intéressantes sont retenues.
Les manuels des fonctions utiles sont à la fin du sujet.

Vous ne devez pas écrire de code, une simple description de la correction suffit.

Code n°1 :

```
    if( rank == 0 )
    {
        for( i = 0 ; i < buffer_size  ;  i++ )
            MPI_Send( buffer + i, 1, MPI_FLOAT, 1, 0,
MPI_COMM_WORLD);
    }

    if( rank == 1 )
    {
        MPI_Status status;
        for( i = 0 ; i < buffer_size  ;  i++ )
            MPI_Recv( buffer + i, 1, MPI_FLOAT, 0,0,
MPI_COMM_WORLD, &status);
    }
```

Quels sont les défauts de ce code ? De quelle manière auriez-vous procédé ?

Code n°2 :

```
    MPI_Comm_rank( MPI_COMM_WORLD, &rank);
    MPI_Comm_size( MPI_COMM_WORLD, &size);

    if( rank == 0 )
    {
        for( i = 0 ; i < size, i++ )
            MPI_Send( &resultat_important, 1, MPI_FLOAT, i, 0,
MPI_COMM_WORLD);
    }
    else
    {
        MPI_Status status;
        MPI_Recv( &resultat_important, 1, MPI_FLOAT, 0,0,
MPI_COMM_WORLD, &status);
    }
```

Quels sont les défauts de ce code ? De quelle manière auriez-vous procédé ?

Code n°3 :

```
    MPI_Comm_rank( MPI_COMM_WORLD, &rank);
    MPI_Comm_size( MPI_COMM_WORLD, &size);

    if( rank == 0 )
    {
         resultat_partiel =  resultat_local;
         MPI_Send( &resultat_partiel, 1, MPI_FLOAT, (rank + 1) %
size, 0, MPI_COMM_WORLD);
    }
    else
    {
        MPI_Status status;
        MPI_Recv( &resultat_partiel, 1, MPI_FLOAT, rank-1,0,
MPI_COMM_WORLD, &status);
        resultat_partiel = resultat_partiel + resultat_local;
        MPI_Send( &resultat_partiel, 1, MPI_FLOAT, (rank + 1) %
size, 0, MPI_COMM_WORLD);
    }

    if( rank == 0 )
    {
        MPI_Status status;
        MPI_Recv( &resultat_total, 1, MPI_FLOAT, size - 1 ,0,
MPI_COMM_WORLD, &status);
    }
```

Quels sont les défauts de ce code ? De quelle manière auriez-vous procédé ?

Code n°4 :

```
#include <pthread.h>


void * calcul_thread( void *a )
{
   pthread_mutex_lock( &verrou_travaux );
     struct travail_thread *travail =
                        recuperer_travail( &liste_travaux );
     effectuer_travail( travail );
   pthread_mutex_unlock( &verrou_travaux );
   return NULL;
}


int main( int argc, char **argv )
{
pthread_t threads[10];

int i = 0;

for( i = 0 ; i < 10 ; i++ )
  pthread_create( &threads[i], NULL, calcul_thread, NULL);
for( i = 0 ; i < 10 ; i++ )
  pthread_join( threads[i], NULL);

return 0;
}
```

Quels sont les défauts de ce code ? De quelle manière auriez-vous procédé ?

MPI_Send

Performs a basic send

Synopsis

```
#include "mpi.h"
int MPI_Send( void *buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm comm )
```

Input Parameters

buf	initial address of send buffer (choice)
count	number of elements in send buffer (nonnegative integer)
datatype	datatype of each send buffer element (handle)
dest	rank of destination (integer)
tag	message tag (integer)
comm	communicator (handle)

Notes

This routine may block until the message is received.

MPI_Recv

Basic receive

Synopsis

```
#include "mpi.h"
int MPI_Recv( void *buf, int count, MPI_Datatype datatype, int source,
              int tag, MPI_Comm comm, MPI_Status *status )
```

Output Parameters

buf	initial address of receive buffer (choice)
status	status object (Status)

Input Parameters

count	maximum number of elements in receive buffer (integer)
datatype	datatype of each receive buffer element (handle)
source	rank of source (integer)
tag	message tag (integer)
comm	communicator (handle)

MPI_Reduce

Reduces values on all processes to a single value usign given MPI operation.

Synopsis

```
#include "mpi.h"
int MPI_Reduce ( void *sendbuf, void *recvbuf, int count,
                 MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm )
```

Input Parameters

sendbuf address of send buffer (choice)
count number of elements in send buffer (integer)
datatype data type of elements of send buffer (handle)
op reduce operation (handle)
root rank of root process (integer)
comm communicator (handle)

Output Parameter

recvbuf
address of receive buffer (choice, significant only at `root`)

Algorithm

This implementation currently uses a simple tree algorithm.

MPI_Bcast

Broadcasts a message from the process with rank "root" to all other processes of the group.

Synopsis

```
#include "mpi.h"
int MPI_Bcast ( void *buffer, int count, MPI_Datatype datatype, int root,
                MPI_Comm comm )
```

Input/output Parameters

`buffer` `starting address of buffer (choice)`
`count` `number of entries in buffer (integer)`
`datatype` `data type of buffer (handle)`
`root` `rank of broadcast root (integer)`
`comm` `communicator (handle)`

Algorithm

If the underlying device does not take responsibility, this function uses a tree-like algorithm to broadcast the message to blocks of processes. A linear algorithm is then used to broadcast the message from the first process in a block to all other processes. MPIR_BCAST_BLOCK_SIZE determines the size of blocks. If this is set to 1, then this function is equivalent to using a pure tree algorithm. If it is set to the size of the group or greater, it is a pure linear algorithm. The value should be adjusted to determine the most efficient value on different machines.