Universitat Pompeu Fabra Barcelona

REPORT LAB 2: MPI

Introducció a la Programació Paral·lela i Distribuïda

Berta Miguel Ninou (U232844)
Núria Esquius Bau (U233645)

1. Montecarlo

1. Explain the modifications you made in the Makefile and job.sh to make it work for an MPI program.

Makefile	job.sh

Makefile
`CC=mpicc`
`OBJ=montecarlo`
 	
`all:`
`    $(CC) -o $(OBJ) $(OBJ).c -lm`
 	
`clean:`
`    rm $(OBJ)`
	$ job.sh
	`#!/bin/bash`
	
	`# Configuration for 1 node, 4 cores and 5 minutes of execution time`
	`#SBATCH --job-name=ex1`
	`#SBATCH -p std`
	`#SBATCH --output=out_montecarlo_%j.out`
	`#SBATCH --error=out_montecarlo_%j.err`
	`#SBATCH --cpus-per-task=1`
	`#SBATCH --ntasks=32`
	`#SBATCH --nodes=1`
	`#SBATCH --time=00:05:00`
	`module purge`
	`module load gcc/13.3.0 openmpi/5.0.3`
	
	`make >> make.out || exit 1  # Exit if make fails`
	
	`mpirun -np 32 ./montecarlo 4 100000000 10`

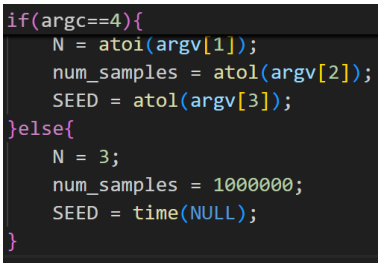
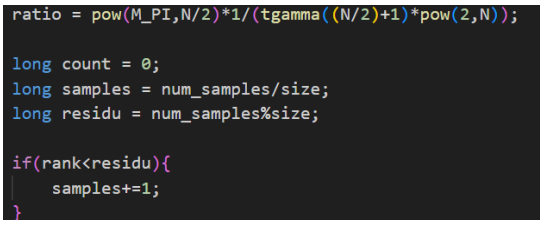
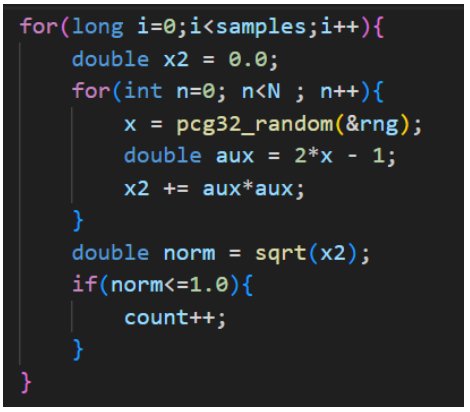
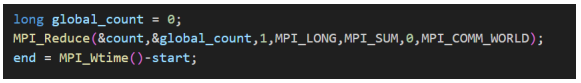
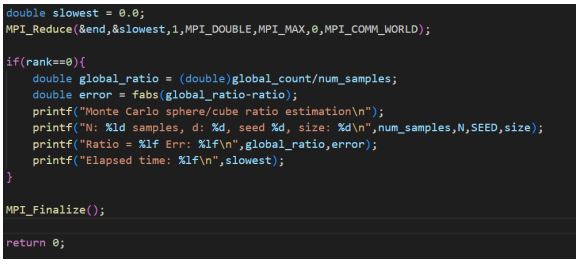
En el cas de Makefile, hem canviat el CC de gcc a mpicc (compilador). En el cas del job.sh, hem afegit les comandes *module purge i module load gcc/13.3.0 openmpi/5.0.3* perquè s'instal·li en cada execució i ens permeti compilar i utilitzar mpicc and mpirun. A més a més, hem de canviar el nombre de tasks en funció de quants ranks volem fer servir. Per executar el codi necessitem especificar quants processadors volem.

2. Describe your approach to designing the program from a parallel computing perspective

El mètode de Monte Carlo funciona generant punts aleatoris de manera independent i verificant si cauen dins de l'esfera. En el nostre codi es calcula el quocient entre el volum d'una esfera i un hipercub en un espai de dimensió N amb un càlcul paral·lel amb MPI per millorar l'eficiència.

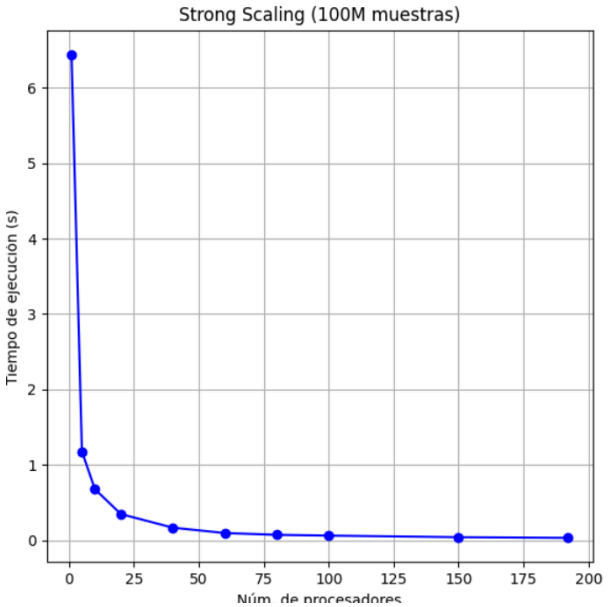
Descripció	Codi

Primer de tot, iniciem el sistema MPI i obtenim el rang del procés actual (rank) i el nombre total de processos (size).	`int rank,size;`
	
	`MPI_Init(&argc,&argv);`
	`MPI_Comm_rank(MPI_COMM_WORLD, &rank);`
	`MPI_Comm_size(MPI_COMM_WORLD, &size);`

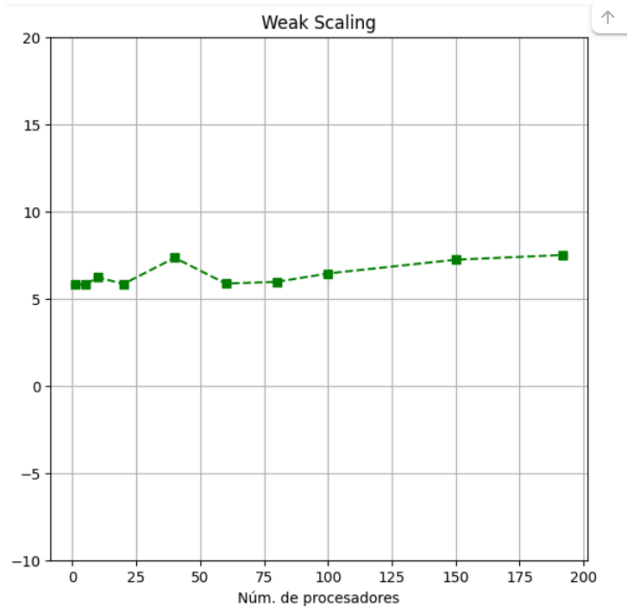
Definim els paràmetres d'entrada que passem com a arguments: N = dimensió de l'espai, num_samples = nombre de mostres i SEED=valor per inicialitzar el generador aleatori.	
Calculem la ratio amb la següent fórmula per calcular després l'error. També repartim el nombre de mostres de les quals s'encarregarà cada rank. Això ho fem dividint les mostres entre el nombre de ranks i calculem el residu per tal de repartir-lo entre els primers ranks.	
Per comptar quants punts cauen dins de la hiperesfera, primer de tot, generem un número random del tipus double de N dimensions que va del 0 a l'1 i fem servir la conversió 2x-1 perquè vagi de -1 a 1. D'aquest punt calculem la seva norma i si és menor a 1 sumem +1 al comptador.	
Per tal de saber quants punts del total han caigut dins l'esfera fem una Reduce per sumar en 'global_count' el resultat de cada comptador de cada rank. Especifiquem on s'ha de guardar el resultat, de quina variable s'ha de fer l'operació, quina operació, el tipus de dades, a quin rank i el comunicador.	
Per calcular el temps que es triga també hem de fer un Reduce amb els temps que han trigat cada rank i tria el màxim. Finalment, calculem la ratio, l'error i l'imprimim per pantalla.	

3. Setting d=10 and starting in 100 million sample points, plot its strong and weak scaling from 1 to 192 processors. Include the job script used to generate this data in the code zip

En strong scaling augmentem el nombre de processadors i mantenim constant el nombre de punts. Podem veure que a mesura que augmentem el nombre de processadors el temps d'execució disminueix ràpidament arribant a prop de 0 amb 192 processadors. Llavors el codi està ben paral·lelitzat i escala bé a més processadors.



En weak scaling augmentem el nombre de processadors i el nombre de punts. En el nostre cas comencem amb 100M de punts i, per exemple, si augmentem a 10 processadors, multipliquem el nombre de punts per 10. Podem veure que el temps d'execució es manté bastant constant, però quan arribem cap a 100 processadors el temps va creixent a causa de la sobrecàrrega de la sincronització i la comunicació.



4. What happens with the ratio computation error when you increase the number of samples?

A mesura que augmentem el nombre de mostres l'error del ratio disminueix de 0.000005 cap a pràcticament 0. Això és degut pel fet que com més punts tinguem més precisió hi haurà.

```
Monte Carlo sphere/cube ratio estimation
N: 100000000 samples, d: 10, seed 10, size: 1
Ratio = 0.002486 Err: 0.000005
Elapsed time: 5.832919
Monte Carlo sphere/cube ratio estimation
N: 500000000 samples, d: 10, seed 10, size: 5
Ratio = 0.002489 Err: 0.000001
Elapsed time: 5.852187
Monte Carlo sphere/cube ratio estimation
N: 1000000000 samples, d: 10, seed 10, size: 10
Ratio = 0.002491 Err: 0.000000
Elapsed time: 6.248625
Monte Carlo sphere/cube ratio estimation
N: 2000000000 samples, d: 10, seed 10, size: 20
Ratio = 0.002492 Err: 0.000001
Elapsed time: 5.879979
Monte Carlo sphere/cube ratio estimation
N: 4000000000 samples, d: 10, seed 10, size: 40
Ratio = 0.002490 Err: 0.000000
Elapsed time: 7.371809
Monte Carlo sphere/cube ratio estimation
N: 6000000000 samples, d: 10, seed 10, size: 60
Ratio = 0.002490 Err: 0.000001
Elapsed time: 5.890207
Monte Carlo sphere/cube ratio estimation
N: 8000000000 samples, d: 10, seed 10, size: 80
Ratio = 0.002490 Err: 0.000000
```

2. Flight controller ✈️

1. Analyze the sequential version of the simulation. What are the main parts that you need to parallelize? What are the challenges?

El codi ens mostra una versió seqüencial del moviment d'avions sobre una graella 2D. El codi realitza la lectura del fitxer d'entrada i inicialització dels avions, l'actualització de la posició de cada avió amb el pas del temps i el filtratge dels avions que surten dels límits.

Per paral·lelitzar aquest codi, cal canviar les operacions d'actualització de les posicions dels avions i gestió de la graella general per cada rank. El codi seqüencial tracta cada avió de manera independent, i el nostre objectiu és paral·lelitzar el codi dividint el mapa entre procesos MPI i assignant a cada procés la gestió dels avions dins d'una regió concreta de la graella. D'aquesta manera, podem paral·lelitzar l'actualització de les posicions dels avions i accelerar el procés i temps d'execució.

Tot i això, cal tenir en compte diversos reptes. Cal assegurar-se que la divisió de la graella sigui equilibrada, ja que alguns processos poden acabar amb més avions que altres. També cal assegurar una comunicació correcta i eficient entre processos. La comunicació dels avions entre ranks és el més important i difícil de gestionar, perquè tots els processos han d'estar sincronitzats després de cada pas de la simulació i han d'actualitzar bé els avions de la seva llista.

2. Regarding the output, how have you managed to parallelize it? What could be the bottlenecks for a large number of ranks?

Per paral·lelitzar la sortida, cada rank s'encarrega de certes zones de la graella. Això permet evitar que un sol procés s'encarregui de tota la sortida i l'actualització de tots els avions, millorant així l'eficiència i fer la feina paral·lelament. Per fer això, hem hagut de repartir l'espai entre els ranks que disposem i afegir a cada rank els avions que li corresponen a la seva zona. Així doncs, cada rank s'encarrega d'una quantitat d'avions en concret i de la seva actualització i filtratge. Això ho hem fet a la funció de read dels avions, on hem creat l'array tile_displacements per repartir la graella equilibradament. A partir d'aquest array, hem pogut afegir a la llista de cada rank els avions que li corresponen accedint a la posició de l'avió en la graella.

En la comunicació entre ranks hem fet servir 3 tipus diferents:

Send/Recv
Per a la comunicació send/recv hem creat dos arrays per saber quants avions hem d'enviar i rebre de cada rank. A més a més, hem creat dos altres buffers per enviar les dades i per rebre-les (en funció de counts buffers). Les dades que enviem són 5 doubles: id de l'avió, posició x, y i la velocitat vx, vy. A l'hora de comunicar els avions, primer de tot, fem les crides asíncrones Isend si hem de transmetre dades a altres ranks i després Recv per rebre de forma bloqueig. Aquesta forma de comunicació permet rebre dades mentre s'estan enviant d'altres i evitar deadlocks. És molt eficient en el nostre cas, ja que hem de transmetre moltes dades entre ranks.

Alltoall

En la comunicació alltoall hem creat també arrays per saber quants avions s'han d'enviar i rebre. A més a més, hem hagut de crear dos arrays de diplacement per saber quants avions hem de rebre i enviar a cada rank, perquè en alltoall fem servir un únic array amb tots els avions que hem d'enviar i, per tant, hem de saber quins són per cada rank.

Struct

En la forma struct creem una estructura de dades per enviar i rebre els avions. Aquesta estructura es basa en l'índex de l'avió, la posició x, y i la seva velocitat vx, vy. Per crear l'estructura necessitem definir quins tipus de dades conté, la quantitat d'aquestes i quants bytes ocupen. Tota la resta de codi és idèntic, menys pel fet que ara hem de crear arrays del tipus de l'estructura en comptes de 5 bytes.

Quan s'utilitza un gran nombre de processos, poden aparèixer colls d'ampolla. La necessitat de sincronització entre processos per evitar interferències o mantenir l'ordre de les dades en la comunicació entre ranks produeix un gran overhead. Aquest és el principal coll d'ampolla, ja que hem de comunicar moltes dades entre molts ranks. Podem veure-ho també a l'output, perquè el temps total d'execució és alt per la comunicació entre ranks. En canvi, la lectura d'avions és ràpida.

3. Discuss the different communication options. Check them input planes 10kk.txt with a moderate number of ranks of 20. What are the key differences between them? Do you see a communication time difference? Why? Include the job script used to generate this data in the code zip.

Entre les diferents comunicacions que hem implementat Alltoall és la més eficient, ja que té un alt rendiment i eficiència quan tots els ranks s'han de comunicar entre ells (com en el nostre cas). A més a més, és una comunicació col·lectiva que amb només una crida ja s'intercanvia tota la informació. En canvi, send/recv necessita més codi per gestionar bé l'enviament i rebuda dels paquets de cada rank i pot arribar a ser molt ineficient si tots necessiten comunicar-se, ja que pot congestionar la xarxa.

Podem veure que els temps d'execució són molt alts i que és probable que hi hagi algun error. Tot i això, el temps de send/recv és més alt que Alltoall, pel fet que alltoall és una comunicació col·lectiva que és molt eficient i més compacta quan tots els ranks s'han de comunicar entre ells. En el nostre cas és molt probable que en cada pas de l'execució hi hagi avions fora de la regió que li pertoca a cada rank i, per tant, s'hagi de comunicar amb els altres. En canvi, entre els temps de Alltoall i struct alltoall són bastant similars. El fet que struct trigui més pot ser pel cost de crear l'estructura de dades o pel soroll del cluster.

Mode 0 (Send)

```
Total planes read: 10000000
Flight controller simulation: #input input_planes_10kk.txt mode: 0 size: 20
Time simulation:     0.05s
Time communication:  174.16s
Time total:          174.18s
```

Mode 1 (Alltoall)

```
Total planes read: 10000000
Flight controller simulation: #input input_planes_10kk.txt mode: 1 size: 20
Time simulation:     0.03s
Time communication:  134.22s
Time total:          134.24s
```

Mode 2 (Struct)

```
Total planes read: 10000000
Flight controller simulation: #input input_planes_10kk.txt mode: 2 size: 20
Time simulation:     0.04s
Time communication:  148.71s
Time total:          148.72s
```

4. Using the same file, use the best communication strategy and increase the number of ranks. Use 20, 40 60, and 80 ranks. Analyze what you observe. Include the job script used to generate this data in the code zip

Com hem comentat anteriorment, la millor estratègia de comunicació és Alltoall. Podem veure que a mesura que augmentem el nombre de ranks el temps decreix molt. Per tant, fer el codi en paral·lel amb un gran nombre de ranks millora molt el temps d'execució i escala bé. Cada rank s'encarrega d'una quantitat de dades menor i la comunicació és més ràpida. A més a més, també millora la latència perquè cada rank ha d'enviar i rebre menys informació.

```
Total planes read: 10000000
Flight controller simulation: #input input_planes_10kk.txt mode: 1 size: 20
Time simulation:     0.08s
Time communication:  270.32s
Time total:          270.38s
```

```
Total planes read: 10000000
Flight controller simulation: #input input_planes_10kk.txt mode: 1 size: 40
Time simulation:     0.04s
Time communication:  121.97s
Time total:          121.99s
```

```
Total planes read: 10000000
Flight controller simulation: #input input_planes_10kk.txt mode: 1 size: 60
Time simulation:     0.03s
Time communication:  78.67s
Time total:          78.68s
```

```
Total planes read: 10000000
Flight controller simulation: #input input_planes_10kk.txt mode: 1 size: 80
Time simulation:     0.02s
Time communication:  41.73s
Time total:          41.74s
```