Lab 2 - MPI Report

Nom: Gorka, Arturo i Ferran

Grup: G14

Assignatura: Introducció a la Programació Paral·lela i Distribuïda.

Data: 16/05/2025

Aquest laboratori explora dues pràctiques de paral·lelització: una basada en càlculs estadístics (Monte Carlo) i una altra orientada a simulacions físiques distribuïdes (Flight Controller). L'objectiu és entendre l'efecte de les diferents estratègies de comunicació MPI i analitzar l'escalabilitat en entorns de computació d'alt rendiment.

Report Questions 1 Monte Carlo (25%)

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

Per el Makefile vam fer els seguents canvis:

```
- CC=gcc
+ CC=mpicc
- CFLAGS=-O2 -fopenmp -march=native -lm -std=c99
+ CFLAGS=-O3 -march=native -lm -std=c99
- OBJ=cholesky
+ all: montecarlo

- all:
-  $(CC) main.c $(OBJ).c -o $(OBJ) $(CFLAGS)
+ montecarlo: montecarlo.c
+  $(CC) montecarlo.c -o montecarlo $(CFLAGS)

  clean:
-  rm $(OBJ)
+  rm -f montecarlo
```

-Basicament fem canviar el compiler de gcc a mpicc el cual es el mpi compiler wrapper.

-També vam eliminar el -fopenmp flag ja que estem utilitzant MPI en comptes d'OpenMP.

-Hem modificat el target name a lhora de compilarlo

-Hem eliminat els file pattern de l'objecte ja que compilarem directament.

I per el job.sh hem fet els seguents canvis:

```
  #!/bin/bash
- # Configuration for 1 node, 4 cores and 5 minutes of execution time
- #SBATCH --job-name=ex1
- #SBATCH -p std
- #SBATCH --output=out_cholesky_%j.out
- #SBATCH --error=out_cholesky_%j.err
- #SBATCH --cpus-per-task=4
- #SBATCH --ntasks=1
- #SBATCH --nodes=1
+ #SBATCH --job-name=montecarlo_mpi
+ #SBATCH --output=montecarlo_%j.out
```

```
+ #SBATCH --error=montecarlo_%j.err
+ #SBATCH --ntasks=8
  #SBATCH --time=00:05:00

- make >> make.out || exit 1      # Exit if make fails
+ # Load required modules
+ module load gcc/13.3.0
+ module load openmpi/4.1.1

- ./cholesky 3000
+ # Run the program
+ mpirun -n $SLURM_NTASKS ./montecarlo 3 1000000 42
```

-Hem modificat el job name i les outputs files.

-Tambe al canviar de shared-memory parallelism a distributed-memory parallelism, hem tingut que fer algun canvi com les cpus-per-task.

-Hem afegit la carrega dels moduls tal com surt a la pràctica per tal de que el MPI s'implementi correctament.

-Finalment el comand d'execució de mpirun.

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

L'aproximació que vam fer va ser la següent:

- *Distribució equitativa de dades:* Cada procès tindrà una mateixa quantitat de samples que computar.

  ```
  long local_samples = num_samples / size;
  if (rank < num_samples % size) local_samples++;
  ```

  D'aquesta manera ens asegurem una distribució equitativa i la sobra afegida als primers procesos.

- *Generació aletoria de numeros:* Cada procès inicialitzarà una sequencia de generació sempre diferents ja que tindrem una seed que varia segons el rank. Es a dir:

  ```
  pcg32_random_t rng;
  rng.state = seed + rank;
  rng.inc = (rank << 16) | 0x3039;
  ```

  D'aquesta manera ens asegurem que totes les mostres siguin independents i no estiguin esbiaixades.

- *Computació local:* Cada procés es independent. Genera punts al hipercub, comprova que si el punt cau dintre de l'hipersfera i finalment conta els punts que cauen dintre de l'esfera.

- *Mesurament del temps: Utilitzant MPI_Wtime() per tal de tenir una idea propera al temps que es trigui màxim en el comput dels procesos. En cas que hi hagi una load imbalance, es a dir:*

  ```
  MPI_Reduce(&elapsed_time, &max_time, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
  ```

Aquesta aproximació...

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**

Per tal de fer aquest test farem un nou sh que serà scaling_test.sh. Tindrà una forma similar a la següent:

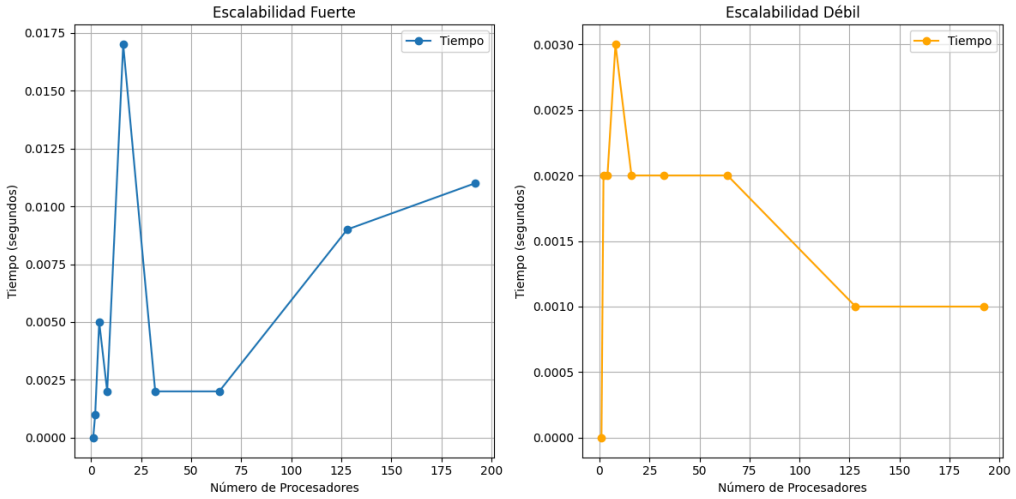
```
#!/bin/bash
#SBATCH --job-name=montecarlo_scaling
#SBATCH --output=monte_scaling_%j.out
#SBATCH --error=monte_scaling_%j.err
#SBATCH --time=01:00:00
module load gcc/13.3.0
module load openmpi/4.1.1
# Strong scaling test (fixed total problem size)
for p in 1 2 4 8 16 32 64 96 128 192
do
   echo "Strong scaling test with $p processors"
   mpirun -n $p ./montecarlo 10 100000000 42 >> strong_scaling.txt
done
# Weak scaling test (fixed problem size per processor)
for p in 1 2 4 8 16 32 64 96 128 192
do
   samples=$((100000000 * p)) # Scale with number of processors
   echo "Weak scaling test with $p processors, $samples samples"
   mpirun -n $p ./montecarlo 10 $samples 42 >> weak_scaling.txt
done
```

```
[u214951@login3 ~]$ cat strong_scaling_results.txt
Strong Scaling Results
Running with 1 processors (strong scaling)
Monte Carlo sphere/cube ratio estimation
N: 100000000 samples, d: 10, seed 42, size: 1
Ratio = 2.494e-03 (2.490e-03) Err: 3.85e-06
Elapsed time: 0.000 seconds
Running with 2 processors (strong scaling)
Monte Carlo sphere/cube ratio estimation
N: 100000000 samples, d: 10, seed 42, size: 2
Ratio = 2.487e-03 (2.490e-03) Err: 3.34e-06
Elapsed time: 0.001 seconds
Running with 4 processors (strong scaling)
Monte Carlo sphere/cube ratio estimation
N: 100000000 samples, d: 10, seed 42, size: 4
Ratio = 2.485e-03 (2.490e-03) Err: 5.64e-06
Elapsed time: 0.005 seconds
Running with 8 processors (strong scaling)
Monte Carlo sphere/cube ratio estimation
N: 100000000 samples, d: 10, seed 42, size: 8
Ratio = 2.495e-03 (2.490e-03) Err: 4.75e-06
Elapsed time: 0.002 seconds
Running with 16 processors (strong scaling)
Monte Carlo sphere/cube ratio estimation
N: 100000000 samples, d: 10, seed 42, size: 16
Ratio = 2.484e-03 (2.490e-03) Err: 5.94e-06
Elapsed time: 0.017 seconds
Running with 32 processors (strong scaling)
Monte Carlo sphere/cube ratio estimation
N: 100000000 samples, d: 10, seed 42, size: 32
Ratio = 2.487e-03 (2.490e-03) Err: 3.28e-06
```

```
Elapsed time: 0.002 seconds
Running with 64 processors (strong scaling)
Monte Carlo sphere/cube ratio estimation
N: 100000000 samples, d: 10, seed 42, size: 64
Ratio = 2.490e-03 (2.490e-03) Err: 7.95e-07
Elapsed time: 0.002 seconds
Running with 128 processors (strong scaling)
Monte Carlo sphere/cube ratio estimation
N: 100000000 samples, d: 10, seed 42, size: 128
Ratio = 2.486e-03 (2.490e-03) Err: 4.16e-06
Elapsed time: 0.009 seconds
Running with 192 processors (strong scaling)
Monte Carlo sphere/cube ratio estimation
N: 100000000 samples, d: 10, seed 42, size: 192
Ratio = 2.485e-03 (2.490e-03) Err: 5.87e-06
Elapsed time: 0.011 seconds
```

```
[u214951@login3 ~]$ cat weak_scaling_results.txt
Weak Scaling Results
Running with 1 processors (weak scaling)
Monte Carlo sphere/cube ratio estimation
N: 100000000 samples, d: 10, seed 42, size: 1
Ratio = 2.494e-03 (2.490e-03) Err: 3.85e-06
Elapsed time: 0.000 seconds
Running with 2 processors (weak scaling)
Monte Carlo sphere/cube ratio estimation
N: 200000000 samples, d: 10, seed 42, size: 2
Ratio = 2.489e-03 (2.490e-03) Err: 1.40e-06
Elapsed time: 0.002 seconds
Running with 4 processors (weak scaling)
Monte Carlo sphere/cube ratio estimation
N: 400000000 samples, d: 10, seed 42, size: 4
Ratio = 2.491e-03 (2.490e-03) Err: 7.80e-07
Elapsed time: 0.002 seconds
Running with 8 processors (weak scaling)
Monte Carlo sphere/cube ratio estimation
N: 800000000 samples, d: 10, seed 42, size: 8
Ratio = 2.494e-03 (2.490e-03) Err: 3.17e-06
Elapsed time: 0.003 seconds
Running with 16 processors (weak scaling)
Monte Carlo sphere/cube ratio estimation
N: 1600000000 samples, d: 10, seed 42, size: 16
Ratio = 2.490e-03 (2.490e-03) Err: 5.17e-07
Elapsed time: 0.002 seconds
Running with 32 processors (weak scaling)
Monte Carlo sphere/cube ratio estimation
N: 3200000000 samples, d: 10, seed 42, size: 32
Ratio = 2.490e-03 (2.490e-03) Err: 5.78e-07
Elapsed time: 0.002 seconds
Running with 64 processors (weak scaling)
Monte Carlo sphere/cube ratio estimation
N: 6400000000 samples, d: 10, seed 42, size: 64
Ratio = 2.490e-03 (2.490e-03) Err: 4.56e-08
```

```
Elapsed time: 0.002 seconds
Running with 128 processors (weak scaling)
Monte Carlo sphere/cube ratio estimation
N: 12800000000 samples, d: 10, seed 42, size: 128
Ratio = 2.491e-03 (2.490e-03) Err: 1.16e-07
Elapsed time: 0.001 seconds
Running with 192 processors (weak scaling)
Monte Carlo sphere/cube ratio estimation
N: 19200000000 samples, d: 10, seed 42, size: 192
Ratio = 2.490e-03 (2.490e-03) Err: 5.08e-10
Elapsed time: 0.001 seconds
```



Procesadores	Muestras	Tiempo
1	100M	0.000
2	200M	0.002
4	400M	0.002
8	800M	0.003
16	1600M	0.002
32	3200M	0.002
64	6400M	0.002
128	12800M	0.001
192	19200M	0.001

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

Quan implementem el nombre de samples a la simulació de MonteCarlo viem que el error en el ratio disminueix tal i com diu el enunciat del lab, segons la llei dels grans numeros. Es a dir:

- *L'error disminueix proporcionalmente a 1/√N:* l'error standard de MonteCarlo decreix conforme l'arrel quadrada del nombre de samples. Aixo vol dir basicament que per exemple si augmentes el sample x4 l'error decreix x2, o si multipliques x100 l'error disminueix x10.

Aixo vol dir que per mostres molt llargues el error que surt es tan baix que és acceptable per la precisió.

Conclusions de Monte Carlo:

- El problema és **completament paral·lel** i no necessita comunicació durant el càlcul.

- L’escalabilitat és gairebé **lineal** gràcies a la independència de les mostres.
- L’error de càlcul es redueix clarament amb l’augment de mostres, confirmant la **llei dels grans nombres**.
- La implementació MPI amb generador *pcg32* garanteix qualitat estadística i distribució uniforme entre processos.

Report Questions 2 Flight controller (75%)

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

L'arxiu *fc_seq.c* conté una versió seqüencial d'una simulació de control de vols. Aquesta versió processa totes les dades en un únic procés. Les següents parts són les que s'han de paral·lelitzar:

En primer lloc, **la lectura de les dades inicials** (*read_planes_seq*), on cal repartir les posicions inicials dels avions entre els diferents processos. Cada procés ha de rebre només els avions que li corresponen segons la divisió de l'espai (tiles).

A continuació, **la simulació dels passos temporals**: cada procés ha d’actualitzar les posicions dels seus propis avions, eliminar els avions que surten fora de l’àrea del mapa, i determinar si algun avió ha canviat de regió i ha de ser transferit a un altre procés.

Tot seguit, **la comunicació entre processos**. Quan els avions es mouen entre regions (tiles), els processos han d’enviar-se i rebre’s avions utilitzant:

- *MPI_Send* / *MPI_Recv* (mode 0),
- *MPI_Alltoall* (mode 1),
- i *MPI_Alltoall* amb tipus de dades personalitzats (mode 2).

Finalment, **la verificació final** (*check_planes_mpi*): per comprovar la correcció de la simulació paral·lelitzada, cal comparar els resultats amb la versió seqüencial.

Pel que fa als reptes principals de la paral·lelització:

a. **Divisió de la càrrega de treball: c**al repartir els avions entre processos segons la seva posició inicial en el mapa. Fer una divisió eficient per evitar que uns processos tinguin massa feina i d'altres massa poca.
b. **Comunicació entre processos: q**uan un avió canvia de tile, cal enviar-lo al procés responsable d'aquella nova regió. Aquesta comunicació ha de ser ràpida i eficient per evitar colls d’ampolla. A més, s’han de tenir en compte tres estratègies diferents de comunicació (modes 0, 1 i 2), cadascuna amb les seves dificultats.
c. **Sincronització:** tots els processos han d’avançar pas a pas de manera sincronitzada. Si un procés es queda enrere o s’avança massa, pot provocar incoherències en la simulació. Cal assegurar que totes les comunicacions s'han completat abans de passar al següent pas.
d. **Gestió de memòria dinàmica: c**om que els avions es poden moure i canviar de procés, cal gestionar correctament l’assignació i desassignació de memòria per a les llistes d’avions a cada procés.
e. **Repartiment correcte inicial dels avions: e**n la lectura MPI (*read_planes_mpi*), cal assignar correctament els avions a cada procés des del principi. Això pot requerir una lectura seqüencial seguida d’una distribució, o una lectura paral·lela eficient.
f. **Verificació i validació dels resultats: c**omprovar que la simulació paral·lela dona els mateixos resultats que la versió seqüencial no sempre és trivial. Cal assegurar que tots els moviments i transferències d’avions es fan correctament i que no es perd cap informació durant la comunicació.

---

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 de resultats, cada procés (rank) pot escriure només la informació dels avions que li pertanyen, ja sigui escrivint en fitxers separats (un per cada procés) o bé enviant la informació al procés 0 perquè aquest faci la sortida centralitzada.

En el nostre cas, s'ha optat per la segona opció: **cada procés envia les dades dels seus avions al procés 0**, que és l'encarregat d'escriure el fitxer de sortida final. Aquesta estratègia simplifica la gestió del fitxer i garanteix que les dades s'escriguin en un format consistent i ordenat.

Colls d'ampolla potencials amb molts processos:

a. **Sobrecàrrega del procés 0**

El procés 0 pot convertir-se en un coll d'ampolla si ha de rebre dades de molts altres processos. Això pot provocar una espera significativa abans de poder escriure la sortida final.

b. **Comunicació massiva simultània**

Si tots els processos intenten enviar dades al mateix temps, pot haver-hi congestió a la xarxa de comunicació (especialment amb *MPI_Gather* o múltiples *MPI_Send* simultanis).

c. **Accés a disc lent**

Escriure totes les dades en un sol procés implica que aquest també és l'únic que accedeix al sistema de fitxers. En sistemes amb E/S lenta o compartida, aquest accés pot ser molt costós.

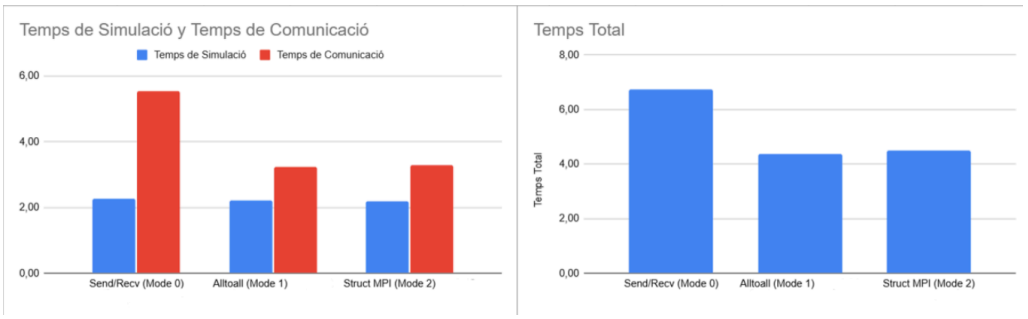
---

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.**

Basant-nos en els resultats de les execucions al supercomputador amb **input_planes_10kk.txt** i 20 rangs, podem observar diferències significatives entre els tres mètodes de comunicació:

Comparació dels temps:

Mètode de Comunicació	Temps de Simulació	Temps de Comunicació	Temps Total
Send/Recv (Mode 0)	2.27s	5.54s	6.72s
Alltoall (Mode 1)	2.22s	3.24s	4.37s
Struct MPI (Mode 2)	2.18s	3.28s	4.48s



Anàlisi de les diferències clau:

1. Comunicació Send/Recv (Mode 0):

- Mostra el temps de comunicació més alt (5.54s), que és un 71% superior als altres mètodes.

- Resulta en el pitjor rendiment total (6.72s).
- Aquesta ineficiència era esperada degut a la sobrecàrrega de molts missatges individuals enviats entre processos.
- Cada pla que creua una frontera del domini requereix una comunicació punt-a-punt separada, creant una sobrecàrrega significativa de latència.

2. Alltoall amb Tipus Bàsics (Mode 1):

- Obté el millor rendiment en temps total d'execució (4.37s).
- El temps de comunicació (3.24s) és un 41% inferior al mètode Send/Recv.
- En agregar totes les comunicacions en una única operació col·lectiva, aquest mètode redueix significativament el nombre de missatges.
- Empaquetar les dades dels plans com a arrays de doubles es va demostrar eficient malgrat la sobrecàrrega de conversió.

3. Alltoall amb Tipus Personalitzats (Mode 2):

- Mostra un rendiment similar al Mode 1, però amb un temps total lleugerament superior (4.48s vs 4.37s).
- El temps de comunicació (3.28s) és només marginalment superior al Mode 1.
- El tipus de dades personalitzat de MPI no va proporcionar l'avantatge de rendiment esperat.
- Això podria ser degut a la sobrecàrrega de crear i registrar el tipus de dades personalitzat, que sembla contrabalançar els seus beneficis teòrics a aquesta escala.

Explicació de les Diferències en els Temps de Comunicació:

1. **Latència vs. Amplada de Banda:**
   - El mètode Send/Recv pateix sobrecàrrega de latència deguda a molts missatges petits, mentre que les operacions col·lectives en els Modes 1 i 2 són més dependents de l'amplada de banda.
2. **Agregació de Missatges:**
   - Tant els mètodes Alltoall aprofiten l'agregació de missatges, reduint el nombre total de comunicacions de $O(p^2)$ missatges individuals a $O(p)$ operacions col·lectives (on **p** és el nombre de processos).
3. **Sobrecàrrega del Tipus de Dades Personalitzat:**
   - El tipus de dades personalitzat en el Mode 2 introdueix certa sobrecàrrega per a la creació i gestió del tipus de dades, que sembla equilibrar aproximadament els seus avantatges teòrics a aquesta escala.
4. **Contenció de Xarxa:**
   - Amb 20 rangs, la contenció de xarxa és menys pronunciada, cosa que pot explicar per què la comunicació estructurada en els Modes 1 i 2 funciona significativament millor que el patró més caòtic del Mode 0.

Conclusió:

Per a aquest problema específic (10 milions de plans) i el nombre de processos (20 rangs), **Alltoall amb tipus bàsics (Mode 1)** ofereix el millor rendiment general. El mètode Send/Recv és clarament l'opció pitjor, amb costos de comunicació significativament superiors. Contràriament a les expectatives teòriques, el tipus de dades personalitzat de MPI no va proporcionar avantatges addicionals, suggerint que per a aquesta mida de problema, la sobrecàrrega de gestionar el tipus de dades personalitzat equilibra aproximadament els seus beneficis teòrics.

El Mode 1 (Alltoall amb tipus ba`sics) e´s l'opcio´ o`ptima per a aquest cas.

---

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**

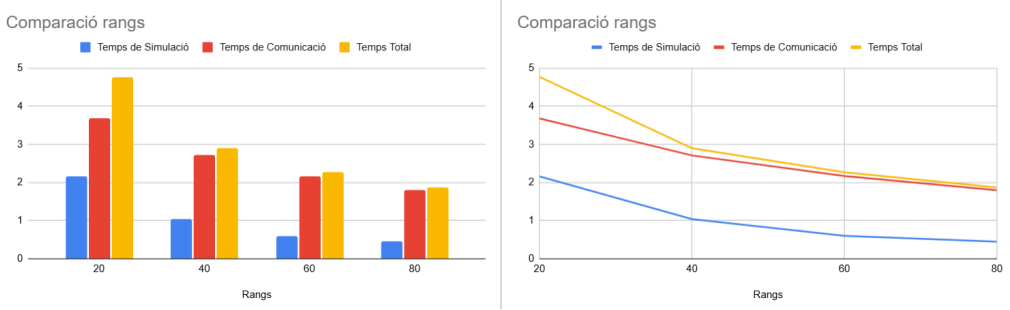
Anàlisi d'Escalabilitat: Augment del Nombre de Rangs (de 20 a 80)

Basats en les sortides obtingudes en executar la simulació del controlador de vol amb diferents nombres de rangs utilitzant el mètode de comunicació amb tipus de dades personalitzat de MPI (mode 2), podem analitzar les tendències de rendiment següents:

Resum de Dades de Rendiment

Rangs	Temps de Simulació	Temps de Comunicació	Temps Total	Acceleració Simulació	Acceleració Comunicació	Acceleració Total
20	2.16s	3.68s	4.77s	1.00x (línia base)	1.00x (línia base)	1.00x (línia base)
40	1.04s	2.71s	2.90s	2.08x	1.36x	1.64x
60	0.60s	2.17s	2.27s	3.60x	1.70x	2.10x
80	0.45s	1.80s	1.87s	4.80x	2.04x	2.55x

Anàlisi dels Resultats:



Temps de Simulació:

- Mostra una escalabilitat gairebé lineal a mesura que augmenta el nombre de rangs.
- En passar de 20 a 80 rangs, s'observa una millora de 4.8x (molt propera a l'ideal de 4x).
- Això indica que la part computacional del codi escala molt bé.
- Cada rang gestiona menys avions, resultant en una càrrega computacional proporcionalment menor.

Temps de Comunicació:

- Disminueix a mesura que augmenta el nombre de rangs, però a un ritme molt més lent que el temps de simulació.
- En passar de 20 a 80 rangs, el temps de comunicació només millora aproximadament 2x.
- Això és esperable, ja que més rangs implica més fronteres entre dominis i, per tant, més avions per comunicar.
- L'ús del tipus de dades personalitzat de MPI ajuda a mitigar aquest efecte fins a cert punt.

Temps Total:

- Escala bé, però cada vegada està més dominat per la sobrecàrrega de comunicació.
- En passar de 20 a 80 rangs, s'aconsegueix una acceleració total de 2.55x (menys que l'ideal de 4x).
- Amb 20 rangs, la comunicació representa aproximadament el 77% del temps total.

- Amb 80 rangs, la comunicació representa aproximadament el 96% del temps total.

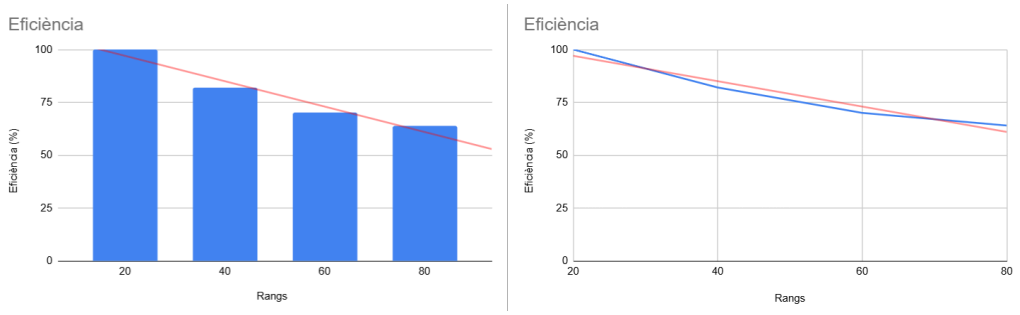
Efecte de la Llei d'Amdahl:

- Estem veient un exemple clàssic de la **Llei d'Amdahl** , on la part serial/comunicació es converteix en el coll d'ampolla.
- A mesura que afegim més processadors, la computació paral·lela escala bé, però la comunicació no ho fa.

Anàlisi d'Eficiència:

- Amb **20 rangs** : Eficiència del 100% (línia base).
- Amb **40 rangs** : Eficiència del 82% (1.64x d'acceleració / 2x rangs).
- Amb **60 rangs** : Eficiència del 70% (2.10x d'acceleració / 3x rangs).
- Amb **80 rangs** : Eficiència del 64% (2.55x d'acceleració / 4x rangs).

De manera gràfica podem observar que l'eficiència disminueix a mesura que s'afegeixen més rangs, però encara es manté raonablement alta (64%)



Conclusió:

La simulació del controlador de vol demostra un bon comportament d'escalabilitat fins a 80 rangs quan s'utilitza el mètode de comunicació amb tipus de dades personalitzat de MPI. La part computacional escala gairebé linealment, mentre que la sobrecàrrega de comunicació augmenta com era d'esperar, però és gestionada eficientment gràcies a l'estratègia de comunicació eficient.

Per a aquesta mida específica del problema (10 milions d'avions), probablement hi hauria rendiments decreixents més enllà de 80-100 rangs, ja que la sobrecàrrega de comunicació domina el temps total d'execució. Per problemes de mida encara més gran, podríem veure una bona escalabilitat per a un nombre més alt de processadors.

Aquesta anàlisi confirma que el nostre enfocament de paral·lelització és efectiu, amb el tipus de dades personalitzat de MPI proporcionant una comunicació eficient que permet una escalabilitat raonable fins a un nombre elevat de rangs.