

## Index

Objectius.....	2
Presentació de la pràctica.....	2
Restriccions de l'entorn.....	2
Material per a la realització de la pràctica.....	2
1. Entorn d'execució MPI.....	2
Pregunta 1.....	3
Pregunta 2.....	4
2. Processos MPI.....	4
Pregunta 3.....	4
3. Pas de missatges punt a punt.....	5
Pregunta 4.....	6
4. Ordre de l'escriptura de la sortida estàndard.....	7
Pregunta 5.....	7
Pregunta 6.....	9
5. Implementació d'un solver mitjançant MPI.....	9
Pregunta 7.....	10
Pregunta 8.....	13
6. Eines d'avaluació de rendiment (TAU).....	18
Pregunta 9.....	19
7. Eines d'avaluació de rendiment (Extrac/Paraver).....	24
Pregunta 10.....	25

## Objectius

Aquesta pràctica té com a objectius introduir els conceptes bàsics del model de programació MPI, i del seu entorn d'execució. Per a la realització es posaran en pràctica alguns dels conceptes presentats en aquesta assignatura.

## Presentació de la pràctica

Cal lliurar els fitxers amb el codi font realitzat i un document amb les respostes a les preguntes formulades i els comentaris que considereu. Tota la codificació es realitza exclusivament en llenguatge C amb les extensions associades a MPI.

## Restriccions de l'entorn

Tot i que el desenvolupament de la pràctica és molt més interessant utilitzant dotzenes de computadors, s'assumeix que inicialment tindreu accés a un nombre força reduït de computadors amb diversos nuclis per node.

## Material per a la realització de la pràctica

En els servidors de la UOC teniu el programari necessari per executar MPI.

De tota manera, si voleu fer el vostre desenvolupament i proves en el vostre sistema local haureu de tenir instal·lat algun programari que implementi MPI. Podeu utilitzar el que millor us sembli però us aconsellem OpenMPI o MPICH2, que és una distribució lliure, portable i molt popular. Podeu descarregar-la i trobar més informació sobre la instal·lació / utilització en la següent adreça:

<http://www.mcs.anl.gov/research/projects/mpich2/>

En qualsevol cas s'espera que realitzeu la vostra investigació i si cal que es debati en el fòrum de l'assignatura.

S'espera que es produeixi debat en el fòrum de l'assignatura per aprofundir en l'ús dels models de programació i els sistemes paral·lels proporcionats.

## 1. Entorn d'execució MPI

Aquesta part consisteix a familiaritzar-se amb l'entorn i ser capaç de compilar i executar programes MPI. Utilitzarem el següent programa MPI que implementa el típic programa que retorna «hello world» (hello.c):

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    MPI_Init(&argc, &argv);
    printf("Hello World!\n");
    MPI_Finalize();
}
```

podeu veure que cal incloure «mpi.h» i que és imprescindible començar el vostre programa MPI amb MPI\_init i al final acabar-lo amb MPI\_Finalize. Aquestes crides s'encarreguen de treballar amb el runtime de MPI per a la creació i destrucció de processos MPI.

Per tindre al clúster els fitxers de la PAC3, els copiem a la carpeta que he creat 'pac3' a meu '/home/capa20':

manelmdiaz@manelmdiaz-Desktop-Ubuntu:~/Nextcloud/Manel/Estudios/UOC/Computacions d'altres prestacions/PAC3\$ scp -P 55000 \*.\* capa20@eimtarqso.uoc.edu:/home/capa20/pac3/

A continuació es presenten els passos necessaris per compilar i executar un simple «hello world» utilitzant MPI:

- Compilar el programa helloc.c amb mpicc (més detalls a la documentació).

```
[capa20@eimtarqso pac3]$ mpicc hello.c -o hello
```

- Executar el programa MPI hello world mitjançant SGE. Un script d'exemples per a l'execució del programa hello utilitzant 8 processos MPI el podeu trobar a continuació:

```
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N hello
#$ -o hello.out.$JOB_ID
#$ -e hello.err.$JOB_ID
#$ -pe orte 8
mpirun -np 8 ./hello
```

L'opció -pe orte 8 li indica a SGE que necessiteu 8 nuclis (que s'assignaran en més d'un node al clúster de la UOC). LA comanda mpirun s'encarrega de cridar al runtime de MPI per realitzar l'execució del programa MPI. L'opció -np 8 li indica al runtime de MPI que utilitzi 8 processos per a l'execució del programa MPI i al final indiquem el nom del binari del programa. Podem estudiar altres opcions de mpirun.

## Pregunta 1

Quin és el resultat de l'execució del programa MPI? Per què?

```
[capa20@eimtarqso pac3]$ qsub hello.sge
Your job 452752 ("hello") has been submitted
```

El resultat ha sigut el següent:

```
[capa20@eimtarqso pac3]$ cat hello.out.452752
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

El codi del programa fa que es mostri per pantalla el text «Hello World!», en aquest cas com al fitxer SGE es crida amb el paràmetre '-np 8', es posen en marxa 8 processos en que tots executen el mateix codi. Per tant es mostra el text un cop per cada execució.

## Pregunta 2

Expliqueu què passa si utilitzeu l'opció `-np 4` en canvi de `-np 8` en el `mpirun`.

Es modifica el paràmetre al fitxer `'hello.sge'` de `'-np 8'` a `'-np 4'`

```
[capa20@eimtarqso pac3]$ qsub hello.sge
Your job 452757 ("hello") has been submitted
```

El resultat ha sigut el següent:

```
[capa20@eimtarqso pac3]$ cat hello.out.452757
Hello World!
Hello World!
Hello World!
Hello World!
```

Al reduir el paràmetre a 4, el mateix codi s'executa en quatre processos i per això mostre 4 cops `'Hello World!'`.

## 2. Processos MPI

A continuació introduïm un programa MPI que inclou crides bàsiques que trobarem en qualsevol programa MPI. Noteu que la variable `"rank"` és bàsica en l'execució de programes MPI ja que permet identificar el número de procés dins d'un comunicador (en aquest cas `MPI_COMM_WORLD`). Si us plau, consulteu els apunts per més detalls.

```
#include <mpi.h>
#include <stdio.h>
#include <unistd.h>
int main(int argc, char **argv)
{
    int rank, numprocs;
    char hostname[256];

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);

    gethostname(hostname,255);
    printf("Hello world! I am process number %d of %d MPI processes on host %s\n", rank, numprocs,
    hostname);

    MPI_Finalize();
    return 0;
}
```

## Pregunta 3

Quin és el resultat de l'execució del programa MPI? Per què?

Creem el fitxer `helloex3.c` amb el codi de l'enunciat i el compilem.

```
[capa20@eimtarqso pac3]$ mpicc helloex3.c -o helloex3
```

Creem un fitxer SGE a partir de l'anterior pregunta i ho modifiquem perquè cridi al programa helloex3 enlloc de hello

```
[capa20@eimtarqso pac3]$ cp hello.sge helloex3.sge
[capa20@eimtarqso pac3]$ cat helloex3.sge
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N helloex3
#$ -o helloex3.out.$JOB_ID
#$ -e helloex3.err.$JOB_ID
#$ -pe orte 8
mpirun -np 8 ./helloex3
```

```
[capa20@eimtarqso pac3]$ qsub helloex3.sge
Your job 452766 ("helloex3") has been submitted
```

El resultat obtingut és:

```
[capa20@eimtarqso pac3]$ cat helloex3.out.452766
Hello world! I am process number 0 of 8 MPI processes on host compute-0-0.local
Hello world! I am process number 1 of 8 MPI processes on host compute-0-0.local
Hello world! I am process number 2 of 8 MPI processes on host compute-0-0.local
Hello world! I am process number 3 of 8 MPI processes on host compute-0-0.local
Hello world! I am process number 4 of 8 MPI processes on host compute-0-2.local
Hello world! I am process number 5 of 8 MPI processes on host compute-0-2.local
Hello world! I am process number 6 of 8 MPI processes on host compute-0-2.local
Hello world! I am process number 7 of 8 MPI processes on host compute-0-2.local
```

Els processos MPI s'han assignat a dos nodes diferents del clúster, això es degut a que al executar mpirun sense especificar un fitxer amb els servidors on es poden executar els processos MPI, s'utilitzarà l'assignació per defecte que estigui establerta en el sistema. Per tant els processos MPI es poden assignar al mateix node o a diferents, com ha sigut el nostre cas.

### 3. Pas de missatges punt a punt

En aquest apartat us demanem el primer exercici de programació. Us facilitem un altre exemple d'aplicació MPI anomenat "hellompi.c" juntament amb l'anunciat de la PAC. Aquest programa està dissenyat per ser executat només amb 2 processos MPI i bàsicament s'encarrega de fer de pas de missatges entre els dos processos MPI.

Es demana que realitzeu una variant del programa "hellompi.c" que faci que cadascun dels processos MPI faci l'enviament d'un missatge MPI a la resta de processos.

Un exemple de sortida amb 3 processos MPI es mostra a continuació:

```
Proc #1 sending message to Proc #0
Proc #1 sending message to Proc #2
Proc #1 received message from Proc #0
Proc #1 received message from Proc #2
Proc #0 sending message to Proc #1
Proc #0 sending message to Proc #2
Proc #0 received message from Proc #1
Proc #0 received message from Proc #2
Proc #2 sending message to Proc #0
Proc #2 sending message to Proc #1
Proc #2 received message from Proc #0
Proc #2 received message from Proc #1
```

## Pregunta 4

Proporcioneu el codi del vostre programa.

He utilitzat `MPI_Isend` i `MPI_Irecv` que son enviaments i recepcions de missatges no bloquejants, conjuntament amb `MPI_Request_free` que allibera els recursos, així com `MPI_Wait` perquè s'espera a rebre el missatge.

El codi del programa es el següent:

```
[capa20@eimtarqso pac3]$ cat hellompiex4.c
#include "mpi.h"
#include <stdio.h>

int main(argc,argv)
int argc;
char **argv;
{

    int MyProc, tag=1;
    char msg='A', msg_recpt;
    int size;
    MPI_Status status;
    MPI_Request request;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &MyProc);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // printf("Process # %d started \n", MyProc);
    MPI_Barrier(MPI_COMM_WORLD);

    int i;
    for (i = 0; i < size; i++) {
        if (i != MyProc) {
            //printf("Inici Proc %d, i = %d\n",MyProc, i);
            MPI_Isend(&msg, 1, MPI_CHAR, i, tag, MPI_COMM_WORLD, &request);
            printf("Proc # %d sending message to Proc # %d\n",MyProc,i);
            MPI_Request_free(&request);
        }
    }
```

```

}

for (i = 0; i < size; i++) {
  if (i != MyProc) {
    //printf("Inici Proc %d, i = %d\n", MyProc, i);
    MPI_Irecv(&msg_recpt, 1, MPI_CHAR, i, tag, MPI_COMM_WORLD, &request);
    printf("Proc #%d received message from Proc #%d\n", MyProc, i);
    MPI_Wait(&request, &status);
  }
}

// printf("Finishing proc %d\n", MyProc);

MPI_Barrier(MPI_COMM_WORLD);
MPI_Finalize();
}

```

El resultat ha sigut:

```

[capa20@eimtarqso pac3]$ cat *453058
Proc #0 sending message to Proc #1
Proc #0 sending message to Proc #2
Proc #0 received message from Proc #1
Proc #0 received message from Proc #2
Proc #1 sending message to Proc #0
Proc #1 sending message to Proc #2
Proc #1 received message from Proc #0
Proc #1 received message from Proc #2
Proc #2 sending message to Proc #0
Proc #2 sending message to Proc #1
Proc #2 received message from Proc #0
Proc #2 received message from Proc #1

```

## 4. Ordre de l'escriptura de la sortida estàndard

En aquest apartat es demana que implementeu un programa MPI que realitzi pas de missatges entre processos MPI en forma d'anell, és a dir, que cada procés MPI faci l'enviament d'un missatge al següent rank i en rebi un del rank anterior (excepte el primer i l'últim que han de tancar l'anell).

Així doncs, donats N processos MPI, s'espera que el procés MPI amb rank 0 faci l'enviament d'un missatge al procés amb rank 1, el procés amb rank 1 rebi el missatge el procés amb rank 0 i faci l'enviament d'un missatge al procés amb rank 2, i així fins arribar al procés amb rank N-1 que rebrà un missatge del procés N-2 i farà l'enviament d'un missatge al procés amb rank 0.

Podeu utilitzar com a referència la següent línia com a exemple per a la sortida del vostre programa.

Proc 2: received message from proc 1 sending to 3

## Pregunta 5

Proporcioneu el codi del vostre programa.

```

[capa20@eimtarqso pac3]$ cat hellompiex5.c

```

```
#include "mpi.h"
#include <stdio.h>

int main(argc,argv)
int argc;
char **argv;
{

    int MyProc, tag=1;
    char msg='A', msg_recpt;
    int size;
    MPI_Status status;
    MPI_Request request;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &MyProc);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // printf("Process # %d started \n", MyProc);
    MPI_Barrier(MPI_COMM_WORLD);

    //Si el procés no és root (0) s'espera a rebre missatge del procés anterior
    if(MyProc != 0){
        MPI_Recv(&msg_recpt, 1, MPI_CHAR, MyProc -1, 0, MPI_COMM_WORLD, &status);
        printf("Proc %d: received message from proc %d sending to %d\n",MyProc, MyProc -1, MyProc +1);
    }

    //El procés envia missatge al procés següent
    MPI_Send(&msg, 1, MPI_CHAR, (MyProc + 1) % size, 0, MPI_COMM_WORLD);

    //Si és el procés 0, s'espera a rebre missatge de l'últim procés
    if(MyProc == 0){
        MPI_Recv(&msg_recpt, 1, MPI_CHAR, size -1, 0, MPI_COMM_WORLD, &status);
        printf("Proc %d: received message from proc %d\n", MyProc, size -1);
    }

    // printf("Finishing proc %d\n", MyProc);

    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Finalize();
}
```

El resultat es el següent:

```
[capa20@eimtarqso pac3]$ mpicc hellompiex5.c -o hellompiex5
[capa20@eimtarqso pac3]$ qsub hellompiex5.sge
Your job 453197 ("hellompiex5") has been submitted

[capa20@eimtarqso pac3]$ cat *453197
Proc 1: received message from proc 0 sending to 2
Proc 2: received message from proc 1 sending to 3
Proc 0: received message from proc 2
```



## Pregunta 6

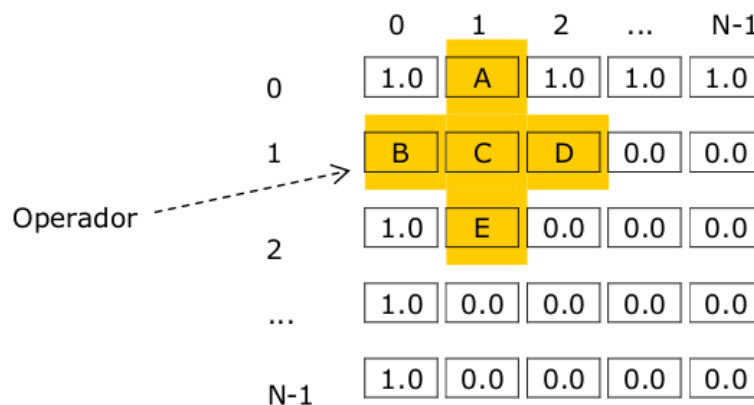
En quin ordre apareixen els missatges de sortida? Per què?

Si seguim el recorregut del codi, tenim que els processos inicien l'execució, i es troben la primera condició on tots els processos excepte el root crida a `MPI_Recv` i, es bloquegen fins no rebre un missatge del procés anterior, per tant el procés 0 seria l'únic que continuaria executant. El codi segueix amb un `MPI_Send` on el procés 0 envia un missatge al procés 1, per tant, l'ordre en que apareixen els missatges de sortida és degut a que tant `MPI_Send` com `MPI_Recv` bloqueja la execució de la resta del codi, i el primer procés a enviar és el procés 0 que determina que el primer missatge sigui que el procés 1 ha rebut missatge del procés 0.

## 5. Implementació d'un solver mitjançant MPI

Aquesta part d'aquesta PAC consisteix en implementar la versió paral·lela mitjançant MPI d'un programa seqüencial que implementa un solver. En primer lloc, el programa inicialitza una matriu (de tamany  $N \times N$ ) i després aplica un operador (també conegut com a stencil) sobre tota la matriu durant un determinat nombre d'iteracions (ITERS). Per a cada element, l'operador aplica la fórmula:

$C' = 0,3 * (C + A + B + D + E)$ , on  $C'$  és el valor de  $C$  en la propera iteració.



Es proporciona un programa seqüencial de referència que implementa el solver (solver.c)

Nota: Per implementar aquest problema es recomana definir la matriu de tamany  $(N+2) \times (N+2)$  per tal de poder aplicar l'operador sense sortir de rang - aquesta tècnica consisteix en envoltar la matriu (amb les també anomenades halo zones). També es proporciona un exemple addicional (stencil.c) que utilitza aquesta tècnica en un programa que implementa la propagació d'energia en un espai 2D. Podeu fer algunes proves i visualitzar el fitxer de sortida (heat.svg), per exemple:

```
./stencil 100 10 50
./stencil 100 10 150
./stencil 100 10 500
```

## Pregunta 7

Implementeu el solver.c mitjançant MPI. Expliqueu les vostres decisions/observacions (per exemple, si heu fet servir crides MPI síncrones o asíncrones/per què?

Tenint com a referència l'exemple stencil.c i solver.c que es una variació simplificada,

A la PAC2 vam veure que en llenguatge C els elements d'un array, s'emmagatzemen en l'ordre de les files de forma seqüencial i adjacent en la memòria, per tant la estratègia ha sigut segmentar la matriu en files per repartir la càrrega entre diversos processos i aprofitar que l'accés a dades sigui per files i millorar l'accés a memòria.

He fet servir crides MPI síncrones perquè a diferència de les asíncrones el codi espera a rebre per terminar la operació i seguir executant el codi i assegurar-me que les dades de la fila anterior i següent s'han rebut dels altres processos i es pot aplicar la fórmula. Podria haver utilitzat crides Send asíncrones per no haver d'esperar a que es finalitzi la crida.

```
[capa20@eimtarqso pac3]$ cat solver_ex7.c
//solver_ex7 N ITERS--> on N es la mida de la matriu NxN i ITERS les iteracions
//solver_ex7 100 20

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include "mpi.h"

// #define N 100
// #define ITERS 20
// N and ITERS might be input arguments

double **A;
int size, MyProc;
int tag = 1;
MPI_Status status;

void initialize (double **A, int n)
{
    int i,j;

    for (j=0;j<n+1;j++){
        A[0][j]=1.0;
    }
    for (i=1;i<n+1;i++){
        A[i][0]=1.0;
        for (j=1;j<n+1;j++) A[i][j]=0.0;
    }
}

void solve(double **A, int n, int iters)
{
    double diff, tmp;
    int i,j;
```

```

int for_iters;
int numRowsProc, firstRow, lastRow;
int N = n;
int ITERS = iters;

//Es calcula el número de files a repartir entre els processos
numRowsProc = (int) (N+2) / size;

//Es calcula quina serà la primera fila i la última per a cada procés
firstRow = numRowsProc * MyProc; //el número de files per cada procés multiplicat pel número del procés actual
lastRow = firstRow + numRowsProc; //la primera fila del procés actual més les files que s'encarrega cada proces

// Com la formula per calcula C necessita de la fila anterior i la següent, ens trobem que al primer procés (0)
// i a l'últim procés necessiten una fila anterior i una posterior, respectivament, per tant es defineixen
// matrius (N+2)*(N+2). Llavors al procés 0, assignem la primera fila al valor 1 i a l'últim procés assignem
// a l'última fila el valor de N+1
if (MyProc == 0 ) firstRow = 1;
if (MyProc == size -1) lastRow = N+1;

for (for_iters=0;for_iters<ITERS;for_iters++)
{
    diff = 0.0;

    // Enviarem i rebrem les dades entre els processos per tal de disposar de les dades per fer els calculs
    // Per evitar deadlocks al procés 0 primer enviarem i a la resta de processos primer rebrem

    // Si el procés actual es el primer (0):
    // 1.Envío la última del procés actual al procés següent
    // 2.Espera a rebre la primera fila del procés següent com la fila última+1 de l'actual procés

    if (MyProc == 0) {
        MPI_Send(A[lastRow], N+2, MPI_DOUBLE, MyProc+1, tag, MPI_COMM_WORLD);
        MPI_Recv(A[firstRow+1], N+2, MPI_DOUBLE, MyProc+1, tag, MPI_COMM_WORLD,&status);
        //printf("\nProc # %d/%d envia fila %d a %d i rep de %d\n", MyProc, size, lastRow, MyProc+1, MyProc+1);
    }

    // Si el procés actual es l'últim:
    // 1.Espera a rebre la última fila del procés anterior, com la fila-1 del procés actual
    // 2.Envía la primera fila del procés actual al procés anterior
    else if (MyProc == size-1){
        MPI_Recv(A[firstRow-1], N+2, MPI_DOUBLE, MyProc-1, tag, MPI_COMM_WORLD,&status);
        MPI_Send(A[firstRow], N+2, MPI_DOUBLE, MyProc-1, tag, MPI_COMM_WORLD);
        //printf("\nProc # %d/%d envia fila %d a %d i rep de %d\n", MyProc, size, lastRow, MyProc+1, MyProc+1);
    } else {
        // Si el procés es diferent a 0 o l'últim:
        // 1.Espera a rebre la última fila del procés anterior, com la fila-1 del procés actual
        // 2.Envía la primera fila del procés actual al procés anterior
        // 3.Envía la última fila del procés actual al procés següent
        // 4.Espera a rebre la primera fila del procés següent, com la última+1 de l'actual procés
        MPI_Recv(A[firstRow-1], N+2, MPI_DOUBLE, MyProc-1, tag, MPI_COMM_WORLD,&status);
        MPI_Send(A[firstRow], N+2, MPI_DOUBLE, MyProc-1, tag, MPI_COMM_WORLD);
        MPI_Send(A[lastRow], N+2, MPI_DOUBLE, MyProc+1, tag, MPI_COMM_WORLD);
    }
}

```

```

    MPI_Recv(A[lastRow+1], N+2, MPI_DOUBLE, MyProc+1, tag, MPI_COMM_WORLD, &status);
    //printf("\nProc # %d/%d envia fila %d a %d i rep fila %d de %d\n", MyProc, size, firstRow, MyProc+1, lastRow,
    MyProc+1);

}

for(i=firstRow; i<lastRow; i++)
{
    for (j=1; j<n; j++)
    {
        tmp = A[i][j];
        A[i][j] = 0.3*(A[i][j] + A[i][j-1] + A[i-1][j] + A[i][j+1] + A[i+1][j]);
        diff += fabs(A[i][j] - tmp);
        /*printf("partial dif is %f \n ", A[i][j] - tmp);*/
    }

}

iters++;

} /*for*/
}

long usecs (void)
{
    struct timeval t;

    gettimeofday(&t, NULL);
    return t.tv_sec*1000000+t.tv_usec;
}

int main(int argc, char * argv[])
{
    int i;
    long t_start, t_end;
    double time;

    int N = atoi(argv[1]); // Mida matriu
    int ITERS = atoi(argv[2]); // Número iteracions
    A = malloc((N+2) * sizeof(double *));
    for (i=0; i<N+2; i++) {
        A[i] = malloc((N+2) * sizeof(double));
    }

    initialize(A, N);

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &MyProc);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    t_start=usecs();

```

```

solve(A, N, ITERS);
t_end=usecs();

MPI_Barrier(MPI_COMM_WORLD);
MPI_Finalize();

time = ((double)(t_end-t_start))/1000000;
printf("Temps de computació = %d [sg].\n", time);

return 0;

}

```

## Pregunta 8

Avalueu l'execució del vostre programa MPI en relació a la versió seqüencial. En particular, es demana que estúdieu (i proporcioneu gràfiques) el speedup respecte el número de nuclis utilitzats, respecte la mida de la matriu i respecte el número d'iteracions. Nota: podeu parametritzar tamany/iteracions.

Per tal de estudiar el speedup necessitaré executar diverses vegades tant el programa seqüencial com la versió paral·lelitzada del exercici 7. Per automatitzar-lo, he creat dos scripts *solver\_ex8\_seq.sh* i *solver\_ex8\_par.sh* que de forma recursiva cridaran les cues SGE amb valors diferents de mida de la matriu, número d'iteracions, i en la versió paral·lela també per número de fils on executar. Aquets scripts accepten com entrada els paràmetres mida i iteracions, els quals s'utilitzen com a paràmetre al cridar el programa, en canvi el número de fils s'assigna a la variable OMP\_NUM\_THREADS. També es calcula el temps que triga en executar-se cada script i s'emmagatzema en un fitxer, per el posterior anàlisi.

També he modificat el codi solver.c i dessat com solver\_ex8\_seq.c perquè accepti com paràmetres la mida de la matriu i el número d'iteracions.

```

[capa20@eimtarqso pac3]$ cat solver_ex8_seq.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>

// #define N 1000
// #define ITERS 100
// N and ITERS might be input arguments

double **A;

void initialize (double **A, int n)
{
    int i,j;

    for (j=0;j<n+1;j++){
        A[0][j]=1.0;
    }
    for (i=1;i<n+1;i++){
        A[i][0]=1.0;
        for (j=1;j<n+1;j++) A[i][j]=0.0;
    }
}

```

```

    }
}

void solve(double **A, int n, int iters)
{
    double diff, tmp;
    int i,j;
    int for_iters;
    int N = n;
    int ITERS = iters;

    for (for_iters=0;for_iters<ITERS;for_iters++)
    {
        diff = 0.0;

        for (i=1;i<n;i++)
        {
            for (j=1;j<n;j++)
            {
                tmp = A[i][j];
                A[i][j] = 0.3*(A[i][j] + A[i][j-1] + A[i-1][j] + A[i][j+1] + A[i+1][j]);
                diff += fabs(A[i][j] - tmp);
                /*printf("partial dif is %f \n ", A[i][j] - tmp);*/
            }
        }
        iters++;
    } /*for*/
}

long usecs (void)
{
    struct timeval t;

    gettimeofday(&t,NULL);
    return t.tv_sec*1000000+t.tv_usec;
}

int main(int argc, char * argv[])
{
    int i;
    long t_start,t_end;
    double time;

    int N = atoi(argv[1]); // Mida matriu
    int ITERS = atoi(argv[2]); // Número iteracions

    A = malloc((N+2) * sizeof(double *));

```

```

for (i=0; i<N+2; i++) {
    A[i] = malloc((N+2) * sizeof(double));
}

initialize(A, N);

t_start=usecs();
solve(A, N, ITERS);
t_end=usecs();

time = ((double)(t_end-t_start))/1000000;
printf("Computation time = %f\n", time);

}

```

Scripts per executar codi seqüencial:

```

[capa20@eimtarqso pac3]$ cat solver_ex8_seq.sh
#!/bin/bash
for nthreads in 1; do
    for iters in 200 300 400 500; do
        for size in 2000 3000 4000 5000; do
            qsub -v nthreads=$nthreads,size=$size,iters=$iters solver_ex8_seq.sge
            echo "$nthreads - $size - $iters"
        done
    done
done

[capa20@eimtarqso pac3]$ cat solver_ex8_seq.sge
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N solver_ex8_seq
#$ -o solver_ex8_seq.out.$JOB_ID
#$ -e solver_ex8_seq.err.$JOB_ID
timeIni=$(date +%s%3N)
./solver_ex8_seq $size $iters
timeFin=$(date +%s%3N)
echo "$size, $iters, 1, $((timeFin-timeIni)) [ms]" >> solver_ex8_seq_result.out

```

Scripts per executar la versió paral·lelitzada del codi.

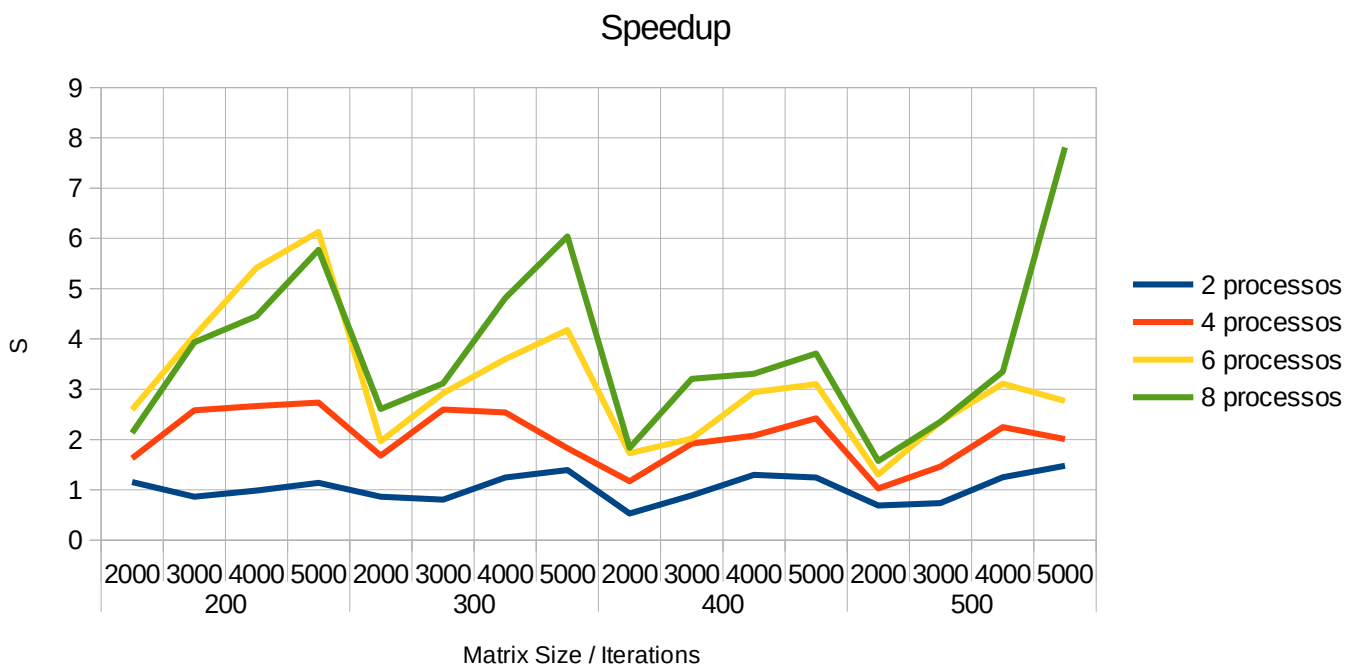
```

[capa20@eimtarqso pac3]$ cat solver_ex8_par.sh
#!/bin/bash
for nthreads in 2 4 6 8; do
    for iters in 200 300 400 500; do
        for size in 2000 3000 4000 5000; do
            qsub -v nthreads=$nthreads,size=$size,iters=$iters solver_ex8_par.sge
            echo "$nthreads - $size - $iters"
        done
    done
done

```

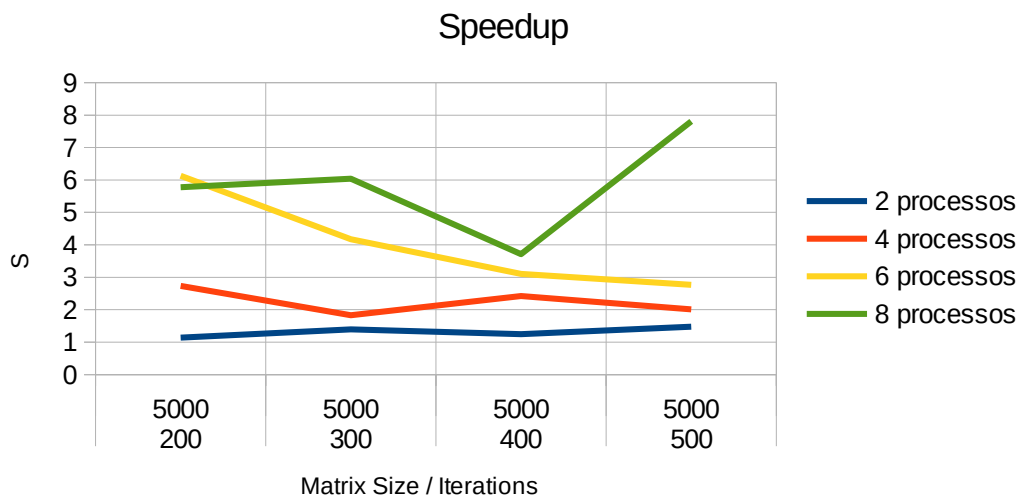
```
[capa20@eimtarqso pac3]$ cat solver_ex8_par.sge
#!/bin/bash
## -cwd
## -S /bin/bash
## -N solver_ex8_par
## -o solver_ex8_par.out.$JOB_ID
## -e solver_ex8_par.err.$JOB_ID
export OMP_NUM_THREADS=$nthreads
timeIni=$(date +%s%3N)
mpirun -np $nthreads ./solver_ex8_par $size $iters
timeFin=$(date +%s%3N)
echo "$size, $iters, $nthreads, $((timeFin-timeIni)) [ms]" >> solver_ex8_par_result.out
```

Els resultats de l'execució seqüencial i paral·lelitzada dels fitxers 'solver\_ex8\_seq\_result.out' i 'solver\_ex8\_par\_result.out', els he consolidat en la fulla de càlcul 'Pac3\_ex8\_grafica.ods', per analitzar el speedup respecte el número de nuclis (processos) utilitzats. La fórmula per calcular el speedup és  $S = T_{seq} / T_{par}$ .

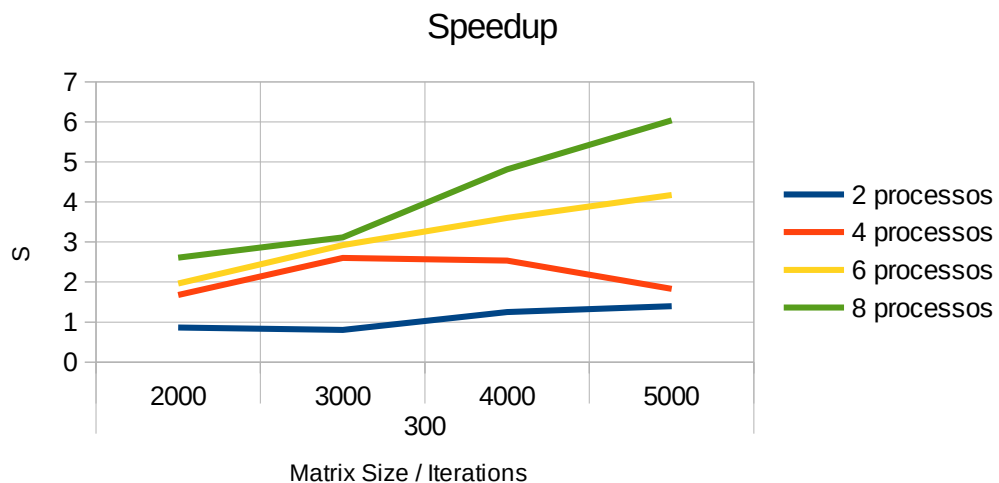
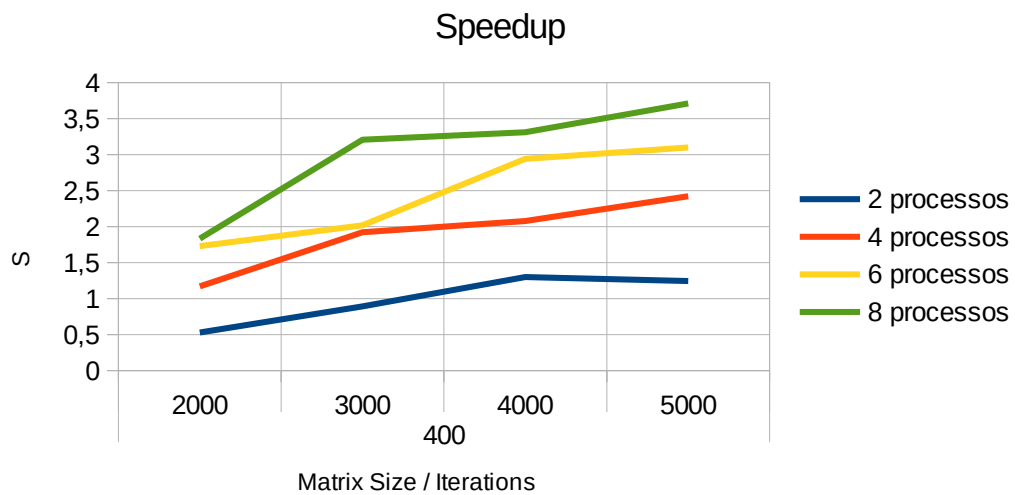


A mesura que augmenten les iteracions i es manté la mida de la matriu, el speedup disminueix al utilitzar més processos, a excepció d'un pic amb 8 processos que amb una matriu 5000 es desmarca de la tendència.





A mesura que augmenta la mida de la matriu l'ús de més processos fa que el speedup augmenti, encara que per sota de 300 iteracions el speedup amb menys de 6 processos no es mostra un guany significatiu o fins i tot disminueix.



Com a conclusió, la utilització de més processos (nuclis) augmenta el speedup a mesura que augmenta la mida de la matriu, però en canvi disminueix a mesura que augmentem el número d'iteracions.

## 6. Eines d'avaluació de rendiment (TAU)

Aquesta apartat consisteix en realitzar un estudi de rendiment del solver implementat.

Per a la realització d'aquest estudi utilitzarem el paquet d'anàlisi de rendiment TAU (<https://www.cs.uoregon.edu/research/tau/home.php>). Aquest paquet es pot utilitzar a nivell d'usuari i es troba en la següent ubicació al clúster de la UOC.

```
/export/apps/tau
```

Us caldrà també afegir el següent a \$PATH:

```
export PATH=$PATH:/export/apps/tau/x86_64/bin
```

El paquet de TAU porta tant les eines d'instrumentació i d'anàlisi com el visualitzador de traces jumpshot. Els passos bàsics per a la utilització de TAU i les eines relacionades es proporcionen a continuació:

### - Ús recomanat de TAU per a la PAC (amb instrumentació dinàmica)

En el vostre script SGE us caldrà utilitzar el següent:

```
export TAU_TRACE=1
(activates tracing)
```

```
mpirun -np NUM_PROCS tau_exec ./YOUR_MPI_PROGRAM
(dynamic instrumentation - recommended)
```

### - Anàlisi amb pprof (podeu explorar pel vostre compte l'eina paraprof si voleu aprofundir)

Només cal executar la comanda pprof en el directori on s'hagi executat el programa MPI i on s'hauran generat els fitxers «profile.\*.\*». Un exemple bàsic es mostra a continuació:

```
[ivan@eimtarqso p4]$ pprof
Reading Profile files in profile.*
```

```
NODE 0;CONTEXT 0;THREAD 0:
```

%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive usec/call	Name
100.0	3,000	3,015	1	12	3015921	.TAU application
0.3	10	10	1	0	10235	MPI_Init()
0.1	3	3	1	0	3450	MPI_Finalize()
0.0	0.863	0.863	2	0	432	MPI_Barrier()
0.0	0.545	0.545	3	0	182	MPI_Send()
0.0	0.018	0.018	3	0	6	MPI_Recv()
0.0	0.001	0.001	1	0	1	MPI_Comm_rank()
0.0	0.001	0.001	1	0	1	MPI_Comm_size()

```
NODE 1;CONTEXT 0;THREAD 0:
```

%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive usec/call	Name
100.0	3,000	3,015	1	12	3015923	.TAU application
0.3	10	10	1	0	10190	MPI_Init()
0.1	3	3	1	0	3470	MPI_Finalize()

Podeu trobar les opcions d'aquesta comanda a continuació:

```
usage: pprof [-c|-b|-m|-t|-e|-i|-v] [-r] [-s] [-n num] [-f filename] [-p] [-l] [-d] [node
numbers]
-a : Show all location information available
-c : Sort according to number of Calls
-b : Sort according to number of suBroutines called by a function
-m : Sort according to Milliseconds (exclusive time total)
-t : Sort according to Total milliseconds (inclusive time total) (default)
-e : Sort according to Exclusive time per call (msec/call)
-i : Sort according to Inclusive time per call (total msec/call)
-v : Sort according to Standard Deviation (excl usec)
-r : Reverse sorting order
-s : print only Summary profile information
-n <num> : print only first <num> number of functions
-f filename : specify full path and Filename without node ids
-p : suPpress conversion to hh:mm:ss:mmm format
-l : List all functions and exit
-d : Dump output format (for tau_reduce) [node numbers] : prints only info about all
contexts/threads of given node numbers
```

### - Visualització de traces amb jumpshot

Per poder entendre millor els possibles problemes relacionats amb el pas de missatges en MPI s'acostuma a visualitzar les traces de l'execució dels programes i fer-hi un estudi amb eines que treballin amb aquestes traces. Cal dir que l'estudi de rendiment mitjançant aquestes tècniques és un art i es requereix d'experiència per a fer anàlisi en profunditat. En aquesta PAC es demana que observeu les característiques més bàsiques i us familiaritzeu una mica amb aquestes eines.

El paquet de TAU porta l'eina jumpshot la qual podeu utilitzar mitjançant els següents passos:

```
tau_treemerge.pl
tau2slog2 tau.trc tau.edf -o tau.slog2
jumpshot tau.slog2
```

Si feu servir jumpshot directament al clúster de la UOC el temps de resposta serà molt gran i és possible que sigui impracticable. Per això es recomana que us descarregueu l'eina en el vostre entorn local. L'eina es troba en el següent directori del paquet de TAU:

```
/export/apps/tau/x86_64/bin/tau2slog2
```

Hi podreu trobar el fitxer jumpshot.jar

```
/export/apps/tau/x86_64/lib/jumpshot.jar
```

Només cal que tingueu instal·lat l'entorn d'execució de Java en el vostre sistema (ja sigui Linux, mac o Windows) per fer servir l'eina.

## Pregunta 9

Feu un breu estudi del comportament de la vostra implementació del solver mitjançant les eines introduïdes. Podeu incloure resums estadístics, captures de pantalla i els comentaris que considereu apropiats.

Executo a la consola

```
[capa20@eimtarqso pac3]$ export PATH=$PATH:/export/apps/tau/x86_64/bin/pprof
```

Executo el script SGE

```
[capa20@eimtarqso pac3]$ cat solver_ex9.sge
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N solver_ex8_par
#$ -o solver_ex8_par.out.$JOB_ID
#$ -e solver_ex8_par.err.$JOB_ID
export OMP_NUM_THREADS=$nthreads
export TAU_TRACE=1
export TAU_PROFILE=1
export PATH=$PATH:/share/apps/tau/x86_64/bin

timeIni=$(date +%s%3N)
mpirun -np $nthreads tau_exec ./solver_ex9 100 10
timeFin=$(date +%s%3N)
echo "$size, $iters, $nthreads, $((timeFin-timeIni)) [ms]" >> solver_ex9_result.out
%pprof
```

Al executar el programa, es generen diversos fitxers tautrace.X.X.X.trc, fitxers events.X.edf, un tau.edf, fitxers profile.X.0.0 i un altra tau.slog2.

### Analizem amb pprof:

Es mostra el resultat per a cada node on es veu que la major part de temps es troba en MPI\_Init i MPI\_Finalize.

```
[capa20@eimtarqso pac3]$ pprof
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
-----
%Time   Exclusive   Inclusive   #Call   #Subrs   Inclusive Name
      msec     total msec
-----
100.0         2         15         1       25   15576 .TAU application
 63.9         9         9         1         0   9947 MPI_Init()
 17.5         2         2         1         0   2723 MPI_Finalize()
  3.8       0.599     0.599        10         0    60 MPI_Recv()
  1.0       0.159     0.159        10         0    16 MPI_Send()
  0.7       0.109     0.109         1         0   109 MPI_Barrier()
  0.0       0.002     0.002         1         0    2 MPI_Comm_rank()
  0.0       0.001     0.001         1         0    1 MPI_Comm_size()
-----

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
-----
NumSamples  MaxValue  MinValue  MeanValue  Std. Dev.  Event Name
-----
      10      816      816      816         0  Message size received from all nodes
      10      816      816      816         0  Message size sent to all nodes
-----

NODE 1;CONTEXT 0;THREAD 0:
-----
%Time   Exclusive   Inclusive   #Call   #Subrs   Inclusive Name
      msec     total msec
-----
100.0         2         15         1       45   15618 .TAU application
 63.8         9         9         1         0   9965 MPI_Init()
 17.5         2         2         1         0   2728 MPI_Finalize()
  2.6       0.404     0.404        20         0    20 MPI_Recv()
  2.2       0.349     0.349        20         0    17 MPI_Send()
  0.4       0.069     0.069         1         0    69 MPI_Barrier()
  0.0       0.002     0.002         1         0    2 MPI_Comm_rank()
  0.0       0.001     0.001         1         0    1 MPI_Comm_size()
-----

USER EVENTS Profile :NODE 1, CONTEXT 0, THREAD 0
-----
NumSamples  MaxValue  MinValue  MeanValue  Std. Dev.  Event Name
-----
      20      816      816      816         0  Message size received from all nodes
      20      816      816      816         0  Message size sent to all nodes
-----
```

Al final es mostra les estadístiques total i la mitjana, que ens permet comparar els resultats dels diferents nodes i detectar diferències dels processos, per exemple el procés 0 i 4 que s'encarreguen de les primeres i últimes línies criden 3 vegades més a MPI\_Send i MPI\_Recv que la resta de processos.

FUNCTION SUMMARY (total):					
%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive Name usec/call
100.0	8	62	4	140	15636 .TAU application
63.9	39	39	4	0	9996 MPI_Init()
17.6	10	10	4	0	2748 MPI_Finalize()
3.4	2	2	60	0	36 MPI_Recv()
1.5	0.909	0.909	60	0	15 MPI_Send()
0.4	0.248	0.248	4	0	62 MPI_Barrier()
0.0	0.008	0.008	4	0	2 MPI_Comm_rank()
0.0	0.003	0.003	4	0	1 MPI_Comm_size()
FUNCTION SUMMARY (mean):					
%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive Name usec/call
100.0	2	15	1	35	15636 .TAU application
63.9	9	9	1	0	9996 MPI_Init()
17.6	2	2	1	0	2748 MPI_Finalize()
3.4	0.536	0.536	15	0	36 MPI_Recv()
1.5	0.227	0.227	15	0	15 MPI_Send()
0.4	0.062	0.062	1	0	62 MPI_Barrier()
0.0	0.002	0.002	1	0	2 MPI_Comm_rank()
0.0	0.00075	0.00075	1	0	1 MPI_Comm_size()

### Analizem amb Jumshot

Primer uneixo i converteixo les traces TAU al format slog2 amb les següents comandes:

```
[capa20@eimtarqso pac3]$ /export/apps/tau/x86_64/bin/tau_treemerge.pl
/state/partition1/apps/tau/x86_64/bin/tau_merge -m tau.edf -e events.0.edf events.1.edf
events.2.edf events.3.edf tautrace.0.0.0.trc tautrace.1.0.0.trc tautrace.2.0.0.trc tautrace.3.0.0.trc
tau.trc
tau.trc exists; override [y]? y
tautrace.0.0.0.trc: 138 records read.
tautrace.1.0.0.trc: 258 records read.
tautrace.2.0.0.trc: 258 records read.
tautrace.3.0.0.trc: 138 records read.

[capa20@eimtarqso pac3]$ /export/apps/tau/x86_64/bin/tau2slog2 tau.trc tau.edf -o tau.slog2
792 records initialized. Processing.
....
....
....
780 Records read. 98% converted
1521 enters: 0 exits: 0
1521 enters: 0 exits: 0
1521 enters: 0 exits: 0
1521 enters: 0 exits: 0
Reached end of trace file.
```

```

SLOG-2 Header:
version = SLOG 2.0.6
NumOfChildrenPerNode = 2
TreeLeafByteSize = 65536
MaxTreeDepth = 0
MaxBufferByteSize = 11094
Categories is FBInfo(902 @ 11202)
MethodDefs is FBInfo(0 @ 0)
LineIDMaps is FBInfo(68 @ 12104)
TreeRoot is FBInfo(11094 @ 108)
TreeDir is FBInfo(38 @ 12172)
Annotations is FBInfo(0 @ 0)
Postamble is FBInfo(0 @ 0)

```

```
1521 enters: 0 exits: 0
```

```

Number of Drawables = 328
timeElapsed between 1 & 2 = 68 msec
timeElapsed between 2 & 3 = 82 msec

```

Després em connecto al clúster per ssh amb el paràmetre -X que permetrà mostrar l'entorn visual al executar jumpshot.

```

manelmdiaz@manelmdiaz-Desktop-Ubuntu:~$ ssh -X -p55000 capa20@eimtarqso.uoc.edu
[capa20@eimtarqso pac3]$ /export/apps/tau/x86_64/bin/jumpshot tau.slog2

```

Al executar jumpshot triga molt temps a respondre i encara que es mostra finestra amb informació es fa impossible utilitzar-ho. Per tant, tal i com recomana l'encunciat de la PAC, copio a meu ordinador personal, fora del clúster de la UOC, els fitxers necessaris, així com tttau2slog2 com jumpshot de la carpeta *share/apps/tau/x86\_64/bin/*. Per copiar els fitxers del clúster al meu equip local on tinc emmagatzemada la pac he executat el següent:

```

sudo scp -P 55000 capa20@eimtarqso.uoc.edu:/home/capa20/pac3/tau* "/home/manelmdiaz/Documentos/PAC3"
sudo scp -P 55000 capa20@eimtarqso.uoc.edu:/home/capa20/pac3/event* "/home/manelmdiaz/Documentos/PAC3"
sudo scp -P 55000 capa20@eimtarqso.uoc.edu:/share/apps/tau/x86_64/bin/jump* "/share/apps/tau/x86_64/lib"
sudo scp -P 55000 capa20@eimtarqso.uoc.edu:/share/apps/tau/x86_64/lib/jumpshot.jar "/share/apps/tau/x86_64/lib"

```
















Al ordinador personal he instal·lat TAU → manelmdiaz@manelmdiaz-Desktop-Ubuntu\$ sudo apt install tau  
Tot seguit, he executat jumpshot

```
manelmdiaz@manelmdiaz-Desktop-Ubuntu:~$ ./jumpshot tau.slog2
```

Es mostren dues finestres una amb la línia de temps dels quatre processos i un altre amb la llegenda on es veu el significat dels colors i formes.

A la llegenda es pot veure que en l'execució van haver 60 missatges (message 1), 4 processos (MPI\_Comm\_rank) i es van enviar (MPI\_Send) i rebre (MPI\_Recv) 60 missatges.

Legend : tau.slog2

Topo	Name	V	S	count	incl	excl
	Preview_Arrow	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	0	0
	message 1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	60	0,072	0
	Preview_State	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	0	0
	.TAU application	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	4	3,982	0,875
	MPI_Barrier()	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	4	0,012	0,012
	MPI_Comm_rank()	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	4	0,001	0,001
	MPI_Comm_size()	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	4	0	0
	MPI_Finalize()	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	4	0,553	0,553
	MPI_Init()	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	4	2,39	2,39
	MPI_Recv()	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	60	0,105	0,105
	MPI_Send()	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	60	0,046	0,046
	Preview_Event	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	0	0
	Message size received from all nodes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	60	0	0
	Message size sent to all nodes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	60	0	0
	TauTraceClockOffsetEnd	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	4	0	0

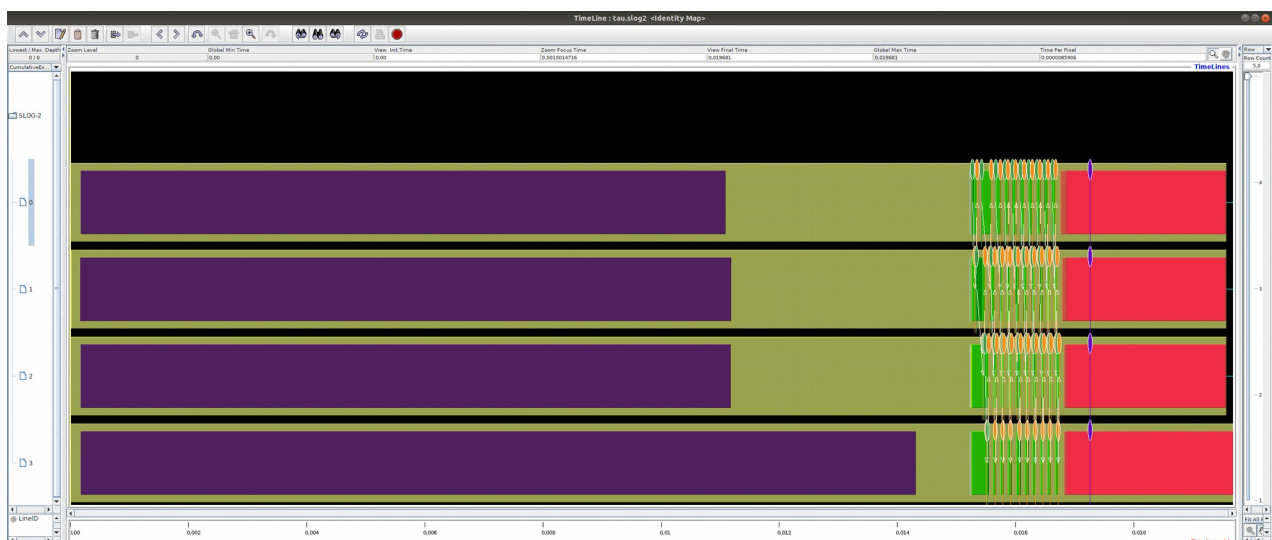
All

Select

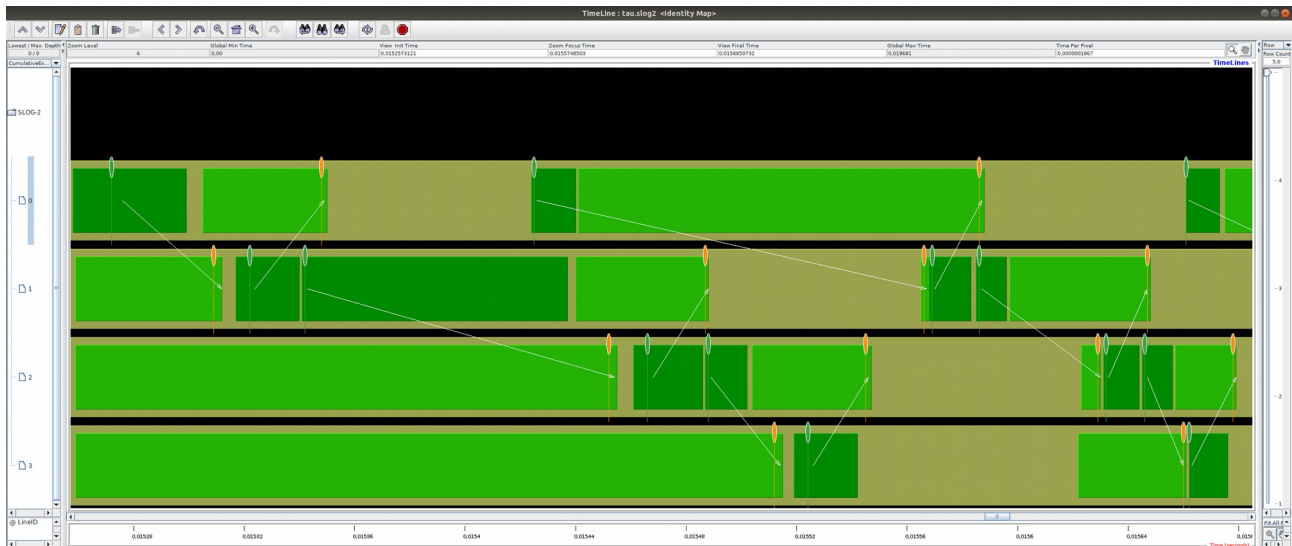
Deselect

close

Al gràfic (Timeline:tau.slog2) amb la línia de temps, es veu que el temps del procés MPI\_Init() i MPI\_Finalize són els que més consumeixen. També veiem que el quart procés triga més temps. Revisaria el codi per millorar probablement la creació de la matriu.



Si ampliem la part on els processos s'envien els missatges, l'enviament i recepció dels primers missatges és més gran perquè, per tant podríem millorar utilitzant crides asíncrones.



## 7. Eines d'avaluació de rendiment (Extræe/Paraver)

Aquesta apartat consisteix en realitzar un estudi de rendiment de algunes de les aplicacions dels NAS Parallel Benchmarks (NPB) - <https://www.nas.nasa.gov/publications/npb.html>. En concret es demana que estudeu els benchmarks EP, BT i CG (classe S). La motivació d'aquest exercici és entendre com fer un estudi de rendiment d'aplicacions no conegudes a priori i de les quals no es pugui disposar del codi font. Els binaris es troben als següents directoris:

```
/export/apps/aca/benchmarks/NPB3.2/NPB3.2-MPI/bin/ep.S.16
/export/apps/aca/benchmarks/NPB3.2/NPB3.2-MPI/bin/bt.S.16
/export/apps/aca/benchmarks/NPB3.2/NPB3.2-MPI/bin/cg.S.16
```

Aquests són de la classe S que és la més petita dels NPB i estan pensats per executar-los amb 16 processos MPI.

En aquest cas farem servir l'eina Extræe (<https://tools.bsc.es/extræe>) desenvolupada pel Barcelona Supercomputing Center.

Per fer servir extræe haureu de definir les següents variables i executar el següent:

```
export EXTRAE_HOME=/export/apps/extræe
cp /export/apps/extræe/share/example/MPI/extræe.xml YOUR_DIRECTORY
source /export/apps/extræe/etc/extræe.sh
export EXTRAE_CONFIG_FILE=YOUR_DIRECTORY/extræe.xml
export LD_PRELOAD=/export/apps/extræe/lib/libmpitrace.so
```

i fer les execucions com habitualment, per exemple:

```
mpirun -np 16 ./filename
```

Com a resultat de l'execució de l'aplicació instrumentada s'obté una traça que està composta de tres fitxers: filename.prv, filename.row i filename.pcf.

Per visualitzar i analitzar la traça obtinguda haureu d'utilitzar l'eina Paraver (<https://tools.bsc.es/paraver>),



la qual està disponible per múltiples plataformes i haureu d'instal·lar en el vostre terminal.

## Pregunta 10

Feu un breu estudi del comparatiu dels benchmarks NPB EP, BT i CG utilitzant Extrae/Paraver. Podeu incloure resums estadístics, captures de pantalla i els comentaris que considereu apropiats.

*Copiem els executables a la nostra carpeta.*

```
[capa20@eimtarqso pac3]$ cp /export/apps/aca/benchmarks/NPB3.2/NPB3.2-MPI/bin/ep.S.4 .
[capa20@eimtarqso pac3]$ cp /export/apps/aca/benchmarks/NPB3.2/NPB3.2-MPI/bin/bt.S.4 .
[capa20@eimtarqso pac3]$ cp /export/apps/aca/benchmarks/NPB3.2/NPB3.2-MPI/bin/cg.S.4 .
```

*Creem un script per a cada executable, només canvia ep per bt i cg i llancem els tres scripts.*

```
[capa20@eimtarqso pac3]$ cat extrae_ep.sge
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N extrae_ep
#$ -o extrae_ep.out.$JOB_ID
#$ -e extrae_ep.err.$JOB_ID
#$ -pe orte 4

export EXTRAE_HOME=/share/apps/extrae
cp /share/apps/extrae/share/example/MPI/extrae.xml .
source /share/apps/extrae/etc/extrae.sh
export EXTRAE_CONFIG_FILE=./extrae.xml
export LD_PRELOAD=/share/apps/extrae/lib/libmpitrace.so

mpirun -np 4 ep.S.4
```

*Executem els scripts:*

```
[capa20@eimtarqso pac3]$ qsub extrae_ep.sge
[capa20@eimtarqso pac3]$ qsub extrae_bt.sge
[capa20@eimtarqso pac3]$ qsub extrae_cg.sge
```

*Els fitxers generats es copien al ordinador local:*

```
sudo scp -P 55000 capa20@eimtarqso.uoc.edu:/home/capa20/pac3/extrae_* "/home/manelmdiaz/Documentos/PAC3"
```

*He descarregar Paraver de la web <https://tools.bsc.es/downloads>, però al instal·lar*

```
./configure
make
```

*mostra el següent error, que no he solucionat.*

```

manelmdiaz@manelmdiaz-Desktop-Ubuntu: ~/Documentos/wxparaver-4.8.0
Archivo Editar Ver Buscar Terminal Ayuda
manelmdiaz@manelmdiaz-Desktop-Ubuntu:~/Documentos/wxparaver-4.8.0$ make install
cd ./src && make install
make[1]: se entra en el directorio '/home/manelmdiaz/Documentos/wxparaver-4.8.0/src'
cd ./paraver-kernel && make install
make[2]: se entra en el directorio '/home/manelmdiaz/Documentos/wxparaver-4.8.0/src/paraver-kernel'
Making install in include
make[3]: se entra en el directorio '/home/manelmdiaz/Documentos/wxparaver-4.8.0/src/paraver-kernel/include'
make[4]: se entra en el directorio '/home/manelmdiaz/Documentos/wxparaver-4.8.0/src/paraver-kernel/include'
make[4]: No se hace nada para 'install-exec-am'.
make[4]: No se hace nada para 'install-data-am'.
make[4]: se sale del directorio '/home/manelmdiaz/Documentos/wxparaver-4.8.0/src/paraver-kernel/include'
make[3]: se sale del directorio '/home/manelmdiaz/Documentos/wxparaver-4.8.0/src/paraver-kernel/include'
Making install in src
make[3]: se entra en el directorio '/home/manelmdiaz/Documentos/wxparaver-4.8.0/src/paraver-kernel/src'
depbase=echo bplustree.lo | sed 's|[/]*$|.deps/&;s|\\.|.lo$||'";\
/bin/bash ../libtool --tag=CXX --mode=compile g++ -DHAVE_CONFIG_H -I. -I.. -I../include/ -I../api/ -I../ -I../.. -g -O2 -I/usr/lib64/boost/include -I/usr
/include/libxml2 -D_FILE_OFFSET_BITS=64 -D_LARGEFILE64_SOURCE -O2 -g -MT bplustree.lo -MD -MP -MF $depbases.Tpo -c -o bplustree.lo bplustree.cpp &&\
mv -f $depbases.Tpo $depbases.Plo
libtool: compile: g++ -DHAVE_CONFIG_H -I. -I.. -I../include/ -I../api/ -I../ -I../.. -g -O2 -I/usr/lib64/boost/include -I/usr/include/libxml2 -D_FILE_OFFSET_BIT
S=64 -D_LARGEFILE64_SOURCE -O2 -g -MT bplustree.lo -MD -MP -MF .deps/bplustree.Tpo -c bplustree.cpp -fPIC -DPIC -o .libs/bplustree.o
In file included from ../api/semantictcolor.h:33:0,
from ../api/trace.h:35,
from bplustree.cpp:36:
../api/paravertypes.h:36:10: fatal error: boost/serialization/string.hpp: No such file or directory
#include <boost/serialization/string.hpp>
^~~~~~
compilation terminated.
Makefile:562: recipe for target 'bplustree.lo' failed
make[3]: *** [bplustree.lo] Error 1
make[3]: se sale del directorio '/home/manelmdiaz/Documentos/wxparaver-4.8.0/src/paraver-kernel/src'
Makefile:584: recipe for target 'install-recursive' failed
make[2]: *** [install-recursive] Error 1
make[2]: se sale del directorio '/home/manelmdiaz/Documentos/wxparaver-4.8.0/src/paraver-kernel'
Makefile:569: recipe for target 'install' failed
make[1]: *** [install] Error 2
make[1]: se sale del directorio '/home/manelmdiaz/Documentos/wxparaver-4.8.0/src'
Makefile:752: recipe for target 'install' failed
make: *** [install] Error 2
manelmdiaz@manelmdiaz-Desktop-Ubuntu:~/Documentos/wxparaver-4.8.0$ sudo scp -P 55000 capa20@eintarqso.uoc.edu:/home/capa20/pac3/extrae_* "/home/manelmdiaz/Documen
tos/PAC3"

```