



UNIVERSIDAD DE COLIMA

Facultad de Telemática

Ingeniería en Software

Práctica #10

“Multiplicación de Matrices con MPI”

Ramírez García Gilberto Felipe

Valencia Sandoval Alfonso

Velasco Munguía Roberto Antonio

5 - “K”

Prof. Ramirez Alcaraz Juan Manuel

18 de Noviembre del 2019

Índice

Índice	2
Introducción	3
Desarrollo	3
Conexión con el cluster	3
Codificación del programa	3
Incluimos las bibliotecas de nuestro programa.	3
Función para generar matrices	4
Función para imprimir	4
Función para multiplicar las matrices	5
Función main() utilizando librería MPI	5
Condición para iniciar el programa	6
Validación del número de procesadores necesarios para ejecutar el programa	6
Proceso Maestro	7
Función BroadCast de mpi	8
Llamada a la función multiplicación de matrices	8
Función Gather de mpi	8
Imprimir resultados	9
Desplegando nuestro trabajo en el cluster	9
Ejecución del programa	10
Captura con 2 nodos, 2 procesadores por nodo = 4 procesadores	11
Captura con 10 nodos, 2 procesadores por nodo = 20 procesadores	11
Conclusión	13
Referencias	13

Introducción

- (Actividad en equipo de tres personas)
- Hacer un programa paralelo con MPI en C que calcule la multiplicación de 2 matrices, una de orden $M \times N$ y otra de $N \times P$.
- Los valores se pueden generar aleatoriamente o introducirlos por teclado.
- El proceso maestro debe distribuir los datos por bloques a cada proceso.
- Para mantener simple el algoritmo, genere el mismo número de procesos que renglones tenga la primer matriz.
- Para cada ejercicio probar la ejecución con al menos 2 diferentes número de procesadores.
- Agregar capturas de pantalla del proceso.
- Recuerden registrar los problemas que encontraron.
- El reporte debe estar en el formato de práctica acordado para la clase.

Desarrollo

Conexión con el cluster

El primer paso para realizar esta práctica será establecer conexión al cluster, tal como lo hemos visto en prácticas anteriores.

```
gilberto@gilberto-Satellite-U940:~$ ssh curso11@148.228.4.17
curso11@148.228.4.17's password:
Last login: Thu Nov  7 22:45:30 2019 from 187.204.176.177
[curso11@rogueone ~]$
```

Para mantener orden en la práctica, creamos un directorio de trabajo para almacenar todos nuestros archivos y códigos de programación de este programa.

```
communication hello-world ramirez sumaVector
[curso11@rogueone ~]$
```

Codificación del programa

Escribimos nuestro programa en un archivo de texto con extensión “.c”. En nuestro caso utilizamos un editor de código local para poder hacer uso de herramientas de corrección y resaltado de código, por lo que posteriormente subiremos el archivo al cluster utilizando *scp*.

Incluimos las bibliotecas de nuestro programa.

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

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

Función para generar matrices

```
void Generate_Matrix(int columns, int rows, int  
outM[rows][columns]) {  
    for (int i = 0; i < rows; i++) {  
        for (int j = 0; j < columns; j++)  
        {  
            outM[i][j] = (rand() % 15) + 1;  
        }  
    }  
  
    return;  
}
```

La función genera una matriz. Recibe como argumentos el número de columnas, el número de filas, y una matriz del tamaño de rows x columns.

Dentro de la función tenemos dos ciclos anidado que al iterarse van generando los números aleatorios con la función `rand()` en la matriz `outM[i][j]`.

Al llamar esta función en el método `main` creamos una variable la cual se pasa como parámetro al argumento `outM`, por lo que la variable en el método `main` y la que recibe `Generate_Matrix` apuntan al mismo espacio de memoria, por lo que podemos decir que `outM` es una variable de salida de la función.

Función para imprimir

```
void Print_Matrix(int rows, int columns, int  
matrix[rows][columns]) {  
    for (int i = 0; i < rows; i++) {  
        for (int j = 0; j < columns; j++) {  
            printf("%d ", matrix[i][j]);  
        }  
        printf("\n");  
    }  
    printf("\n");  
}
```

Esta función tiene como tarea imprimir las matrices que utilizaremos en el trabajo. Recibe como argumentos el número de filas, el número de columnas, y la matriz que debe corresponder a los tamaños dados en los dos argumentos anteriores.

Dentro de la función dos ciclos anidado se encarga de imprimir la matriz que recibe la función.

Función para multiplicar las matrices

```
void Multiply_Matrices(int m, int n, int p, int mA[m][n], int
mB[n][p], int outM[m][p]) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < p; j++) {
            outM[i][j] = 0;
            for (int k = 0; k < n; k++) {
                outM[i][j] += mA[i][k] * mB[k][j];
            }
        }
    }
    return;
}
```

Esta función tiene como tarea realizar la multiplicación de matrices. Recibe como argumentos m = el número de filas de la matriz A; n = el número de columnas de la matriz A y, a su vez, número de filas de la matriz B; p = el número de columnas de la matriz B; mA = a la matriz de número aleatorios $m \times n$; mB = la matriz de números aleatorios $n \times p$; y $outM$ = la matriz en la que se almacenará el resultante de la multiplicación de la $mA \times mB$.

Tres ciclos anidados realizan la tarea de la multiplicación. El ciclo más interno realiza el cálculo de sumar la multiplicación punto por punto de las filas de la matriz A por las columnas de la matriz B (también conocido como producto punto). El ciclo intermedio recorre las columnas de la matriz B. Mientras que el ciclo exterior, recorre las filas de la matriz A,

Al llamar esta función en el método main creamos una variable la cual se pasa como parámetro al argumento outM, por lo que la variable en el método main y la que recibe Multiply_Matrix apuntan al mismo espacio de memoria, por lo que podemos decir que outM es una variable de salida de la función.

Función main() utilizando librería MPI

```
int main(int argc, char* argv[]) {

    int proc_id, total_procs;
```

```

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &total_procs);
MPI_Comm_rank(MPI_COMM_WORLD, &proc_id);

```

En esta sección tenemos la inicialización de:

- Variables globales del programa: el id de proceso, y el total de procesos.
- La función de inicio de MPI: MPI_Init().
- Pasamos a la variable global de comunicación MPI_COMM_WORLD, el total de procesos, y el id de cada proceso que participa en esta tarea con las funciones MPI_Comm_size, y MPI_Comm_rank.

Condición para iniciar el programa

```

if (argc > 1) {

    int m, n, p;

    m = atoi(argv[1]);
    n = atoi(argv[2]);
    p = atoi(argv[3]);
    ...
} else if (proc_id == 0) {
    printf("Program usage: matrix-multiply <m> <n> <p>\n");
}

```

Si el programa recibe como argumentos > 1 , el programa continúa. De lo contrario, se sale de ejecución y lanza un mensaje para el usuario con instrucciones de cómo mandar a ejecutar el programa pasando los parámetros m , n y p .

Dentro del if están declaradas las variables m , n , y p .

La función atoi() convierte el argumento string que recibe el programa a un dato de tipo integer.

Validación del número de procesadores necesarios para ejecutar el programa

```

if (m >= total_procs && m % total_procs == 0) {
    int block_size = m / total_procs;
    int mA[m][n], mB[n][p], mC[m][p], pA[block_size][n],
    pC[block_size][p];
    ...
}

```

```

    } else if (proc_id == 0) {
        printf("Can't execute program: More processes than columns
of A or there is no precise block size!\n");
    }

```

En esta sección se validan las condiciones para realizar la multiplicación de matrices en paralelo. Son dos condiciones. El número de procesos debe ser igual o mayor al número de filas de la matriz A y, al mismo tiempo, el número total de procesos debe ser múltiplo del número de filas de dicha matriz.

Si se cumple la condición, se declaran las variables:

- block_size: que es el número de filas de la matriz A entre el total de procesos.
- mA: para la matriz A de m x n.
- mB: para la matriz B de n x p.
- mC: para la matriz resultante de multiplicar A x B.
- pA[block_size][n]: para el vector resultado de la división de la matriz A, al dividir los datos de esta entre el total de procesos.
- pC[block_size][p]: para el vector captura la suma de la fila de las multiplicaciones de los elementos de A por los correspondientes elementos de la columna de B.

De lo contrario, el programa termina arrojando el error: "Can't execute program: More processes than columns of A or there is no precise block size!\n"

Proceso Maestro

```

if (proc_id == 0) {
    // Initialize the matrixes
    Generate_Matrix(n, m, mA);
    Generate_Matrix(p, n, mB);
}

```

Condición para validar el trabajo que solo hace el proceso maestro. En este caso, nosotros decidimos que el proceso maestro es el proceso con el id 0. Este proceso, genera las matrices de números aleatorios A y B.

Función Scatter de MPI

```

MPI_Scatter(&mA, n * block_size, MPI_INT, &pA, n * block_size,
MPI_INT, 0, MPI_COMM_WORLD);

```

La función Scatter divide el trabajo entre los procesos participantes. Divide el número de filas de la matriz A entre el total de procesos, y esto por el número de columnas de A. Estos son los argumentos de:

- Envío: la dirección del buffer del proceso que envía los datos, el número de elementos que envía, y el tipo de datos de estos.
- Recepción: la dirección del buffer del proceso que recibe los datos, el número de elementos que recibe, y el tipo de datos de estos.
- Comunicador: el proceso que reparte el trabajo, en este caso, el maestro, y el comunicador global de mpi.

Función BroadCast de mpi

```
MPI_Bcast(&mB, p * n, MPI_INT, 0, MPI_COMM_WORLD);
```

La función broadCast comunica datos a todos los procesos involucrados en la tarea. Estos son sus argumentos: dirección del buffer de envío; número de elementos a enviar, en este caso, $n * p$; el tipo de datos enviados; el proceso que envía, en este caso, el maestro; y el comunicador global de mpi.

Llamada a la función multiplicación de matrices

```
// Performing multiplications
Multiply_Matrices(block_size, n, p, pA, mB, pC);
```

Esta es la llamada a la función de multiplicación de matrices, ya explicada arriba. Envía los argumentos de tamaño de bloque, número de columnas de A = número de filas de B, el número de columnas de B, el vector parcial de A, toda la matriz B, y el vector parcial de las sumas de las multiplicaciones de las filas de A x las columnas de B.

Función Gather de mpi

```
// Collecting results
```

```
MPI_Gather(&pC, block_size * p, MPI_INT, &mC, block_size * p,
MPI_INT, 0, MPI_COMM_WORLD);
```

La función Gather de mpi recolecta los datos de los procesos esclavos y los regresa al proceso maestro. Recibe como argumentos de:

- Envío: dirección del buffer que envía datos, el número de elementos enviados por proceso, y el tipo de dato de dichos elementos.
- Recepción: dirección del buffer que recibe datos, el número de elementos recibidos, y el tipo de dato de los mismos.

- Comunicación: id del proceso destino, en este caso, el maestro; y el comunicador usado.

Imprimir resultados

```
// Printing results
if (proc_id == 0) {
    printf("Printing A\n");
    Print_Matrix(m, n, mA);

    printf("Printing B\n");
    Print_Matrix(n, p, mB);

    printf("Printing C\n");
    Print_Matrix(m, p, mC);
}
```

Ya solo queda imprimir los resultados. Quien imprime es el proceso maestro. Imprimimos las tres matrices: A, B y C que es la matriz resultante de $A \times B$.

Función Finalize de mpi:

```
MPI_Finalize();
return 0;
```

La función Finaliza cierra la comunicación del programa paralelo.

Desplegando nuestro trabajo en el cluster

Cargamos la librería de MPI y compilamos el programa matrix-multiply.c

```
[curso13@rogueone matrix-multiply]$ module load Compilers/Parallel-Studio-XE-2018
[curso13@rogueone matrix-multiply]$ mpicc matrix-multiply.c
```

Adaptamos el script mpi-run.sh para este ejercicio:

```

[curso13@rogueone matrix-multiply]$ cat mpi-run.sh
#!/bin/bash

#PBS -l nodes=2:ppn=2,walltime=00:00:10
#PBS -N mpi_poncho_matrixmult
#PBS -q staff
#PBS -d /mnt/zfs-pool/home/curso13/Poncho/matrix-multiply/
#PBS -o matrix-mult.log
#PBS -j oe
#PBS -V
#PBS -S /bin/bash

source $MODULESHOME/init/bash
module purge
module load Compilers/Parallel-Studio-XE-2018

NPROCS=`wc -l < $PBS_NODEFILE`
cat ${PBS_NODEFILE} | sort -u > $PBS_O_WORKDIR/machines.LINUX

mpirun -np $NPROCS -machinefile machines.LINUX ./matrix-multiply 8 8 8 > matrix-
mult.out
[curso13@rogueone matrix-multiply]$ █

```

Ejecución del programa

Enviamos el script a la cola de ejecución, para seguir las políticas del clúster.

```

[curso13@rogueone matrix-multiply]$ qsub mpi-run.sh
7459.rogueone █

```

Captura con 2 nodos, 2 procesadores por nodo = 4 procesadores

m = 8, n = 8, p = 8

```
7459.rogueone
[curso13@rogueone matrix-multiply]$ cat matrix-mult.out
Printing A
14 2 13 11 9 11 2 13
10 2 3 8 6 5 9 2
1 7 8 2 12 9 13 10
3 6 3 14 8 11 15 13
13 4 15 13 14 2 2 8
10 4 7 15 15 8 9 15
6 1 9 2 2 6 12 4
3 6 2 2 1 1 15 13

Printing B
12 6 2 10 14 3 10 1
13 1 15 13 15 15 4 6
8 4 7 9 9 10 4 12
15 5 13 1 12 4 5 8
2 7 10 15 1 11 15 14
3 7 3 3 13 13 8 12
2 6 12 2 1 15 5 15
11 10 7 8 13 12 7 6

Printing C
661 475 530 570 798 674 579 636
357 265 372 300 399 412 349 413
384 380 562 488 487 760 467 686
570 459 683 436 662 797 510 734
649 411 586 614 679 607 563 619
690 532 711 617 766 804 658 769
277 251 326 260 348 459 276 440
338 276 420 280 372 532 261 408

[curso13@rogueone matrix-multiply]$
```

Captura con 10 nodos, 2 procesadores por nodo = 20 procesadores

m = 100, n = 10, p = 100

```
[curso13@rogueone matrix-multiply]$ cat matrix-mult.out
```

```
Printing A
```

```
14 2 13 11 9 11 2 13 10 2
3 8 6 5 9 2 1 7 8 2
12 9 13 10 3 6 3 14 8 11
15 13 13 4 15 13 14 2 2 8
10 4 7 15 15 8 9 15 6 1
9 2 2 6 12 4 3 6 2 2
1 1 15 13 12 6 2 10 14 3
10 1 13 1 15 13 15 15 4 6
8 4 7 9 9 10 4 12 15 5
13 1 12 4 5 8 2 7 10 15
1 11 15 14 3 7 3 3 13 13
8 12 2 6 12 2 1 15 5 15
11 10 7 8 13 12 7 6 10 8
13 2 3 4 7 6 2 9 15 15
14 7 3 15 4 15 1 4 14 6
11 10 7 9 9 11 12 7 2 6
15 14 15 9 2 6 14 11 15 5
10 5 3 12 4 7 3 11 10 2
8 12 3 14 6 3 10 2 9 11
15 8 1 14 9 9 4 7 4 10
12 5 14 14 9 9 12 11 5 14
4 4 10 13 3 7 15 4 1 9
6 15 8 6 5 1 6 15 15 2
10 3 13 15 8 13 1 5 9 12
3 4 15 4 2 9 11 8 12 11
1 2 2 1 14 6 1 12 5 7
13 6 9 2 13 9 15 13 13 15
1 7 3 15 10 11 9 12 4 12
7 11 14 15 11 4 5 12 15 2
10 4 14 4 6 11 12 12 15 1
11 15 7 13 7 8 1 7 5 4
3 3 14 8 10 10 12 6 13 3
7 7 14 13 2 11 15 5 7 15
5 2 6 3 6 4 3 6 10 14
```

```
Printing B
```

```
9 5 5 5 7 15 6 13 15 12 3 8 2 2 13 11 3 3 15 13 10 9 1 7 7 10 6 14 2 3 3 3 15 14 7 6 13 12 11 12 1 13 11 2 15 8 12 9
1 10 2 3 9 15 2 6 12 5 1 15 5 6 10 7 12 9 10 5 11 13 2 14 1 15 15 2 14 15 1 14 1 3 9 9 9 10 6 6 14
14 12 11 4 14 2 7 14 3 11 1 7 12 6 14 11 6 15 1 5 8 15 5 10 15 5 10 9 2 7 14 15 4 9 10 2 2 1 7 12 3 8 3 15 13 9 2 3
13 7 4 7 9 12 15 1 4 11 4 11 6 10 15 14 11 3 5 5 1 12 13 11 11 12 9 5 11 9 14 9 8 9 7 1 6 6 1
9 2 4 11 7 13 10 12 1 4 1 5 5 4 10 7 15 13 15 11 9 1 9 9 8 10 15 15 15 12 15 15 13 11 2 11 15 11 7 7 7 15 4 3 3 13 9 9 10 8 5 10 15 5 10 14 14 9
13 5 6 5 4 10 15 5 6 6 8 4 5 6 3 8 8 13 5 1 6 6 15 2 8 15 14 2 13 12 3 11 9 15 7 4 9 13 1 6 3 8
10 7 5 12 14 12 9 10 12 7 8 3 8 15 2 6 1 7 10 10 9 3 9 15 13 3 12 13 15 6 12 9 5 1 6 3 12 6 5 15 12 12 3 12 11 11 9 3 2 10 13 2 4 13 1 2 15 4 14
7 10 11 15 14 11 12 8 15 3 4 14 6 15 8 9 2 4 3 5 12 12 9 14 1 6 6 2 13 2 15 4 11 2 3 1 13 7 15 12 9
4 2 6 3 2 15 5 5 2 1 8 5 9 13 5 6 11 13 3 12 5 6 14 6 1 14 3 7 13 6 7 1 15 12 4 1 3 15 12 11 15 4 1 15 9 12 5 4 2 15 7 6 5 5 11 12 3 6 10 7 3 8
15 2 12 10 2 14 9 13 10 15 9 2 14 9 13 10 4 14 9 10 11 6 14 7 9 8 4 4 6 6 11 5 8 7 14 1 13 14
14 7 13 14 8 3 7 12 13 10 3 6 4 13 3 9 11 12 8 14 15 5 12 2 10 4 1 15 4 13 14 9 11 11 7 10 6 13 6 3 14 15 15 9 5 10 2 15 6 1 6 12 6 2 13 7 5 5 1
3 15 2 11 1 4 14 7 13 4 12 11 13 2 10 4 11 14 13 12 6 3 5 11 6 2 4 11 15 15 15 12 14 9 8 14 12 13 13 10 8 1
5 2 6 15 12 5 13 1 2 7 12 4 13 13 7 8 12 6 14 15 11 7 14 10 4 11 14 13 3 6 2 14 7 7 6 11 3 10 3 11 1 14 14 5 11 5 4 14 10 3 5 6 9 4 7 12 14 12
1 9 2 9 14 15 8 11 10 10 5 12 13 13 2 11 2 4 8 13 2 9 15 7 6 15 2 12 4 7 15 4 15 8 13 6 8 5 1 9 6
13 13 10 14 6 3 3 5 15 11 13 6 9 4 6 10 7 1 2 14 11 8 6 8 5 13 4 6 7 2 3 11 3 4 9 15 6 3 4 6 14 2 11 14 12 8 9 3 8 2 1 4 9 14 3 14 11 14 11 9
7 5 4 9 8 5 9 13 7 4 10 12 5 13 11 8 5 11 3 5 12 10 15 12 8 9 2 4 7 4 4 5 8 15 6 7 11 6 12
9 9 6 6 6 3 8 13 15 3 15 4 6 10 3 9 2 11 11 12 10 14 8 6 14 7 11 5 9 1 8 2 10 14 14 7 8 6 4 7 15 4 2 12 5 4 6 6 7 1 10 8 6 2 13 4 15 1 1 8 8 8 9
2 13 8 15 13 5 4 4 5 14 6 1 3 1 13 15 14 13 9 6 11 2 11 14 1 11 6 15 3 6 9 12 10 8 11 7 4
6 11 15 4 8 8 6 8 5 6 14 10 6 4 5 8 6 10 8 8 1 8 11 13 8 14 7 7 1 6 10 7 8 2 10 15 9 8 14 13 13 4 7 3 8 11 10 5 6 10 13 13 9 15 2 1 13 15 14 5 5
15 11 12 1 13 3 9 5 9 7 2 12 5 11 11 1 13 1 13 7 5 2 15 4 3 7 1 2 12 5 14 11 8 2 12 5 12 5 9
```

```
Printing C
852 583 620 744 765 831 590 864 749 679 585 588 488 768 558 684 658 812 783 897 896 632 718 668 774 654 784 924 749 112
921 817 615 734 808 958 507 760 790 878 649 678 443 562 607 649 566 648 863 640 951 697 1026 621 620 638 614 604 108 58
9 847 545 791 823 647 667 632 742 882 652 851 669 864 732 784 796 662 717 803 651 523 662 807 876 692 737 648 764
487 380 382 360 456 434 340 496 345 355 373 337 367 460 347 402 379 552 353 471 463 448 441 418 477 402 481 458 441 319 466 344 524 459 360 299
442 410 367 491 458 193 550 472 506 329 311 329 362 423 342 351 325 494 372 588 336 519 321 365 434 439 297 566 426 406 596 342 322 435 379
533 287 474 478 402 426 360 462 482 364 496 489 504 412 502 381 306 428 432 427 361 360 465 469 359 396 394 411
882 702 716 702 872 782 599 898 672 734 629 665 538 686 639 719 640 853 749 818 836 721 696 788 858 683 872 900 656 592 802 654 910 701 613 691
940 708 666 808 759 920 497 742 842 906 681 625 557 578 710 683 636 739 777 578 1088 781 1063 543 672 739 642 690 954 705 678 932 609 489 793 53
2 898 567 820 884 599 705 587 738 851 585 820 845 860 691 789 708 573 811 751 794 572 672 758 889 564 794 594 736
878 608 729 693 882 982 653 1082 618 660 524 723 598 836 865 822 814 1072 794 1093 943 778 817 852 849 811 779 1144 778 681 941 697 1136 945 650
652 859 884 891 859 900 879 738 920 849 1034 638 684 735 704 742 847 717 720 848 835 984 892 1183 513 656 749 728 759 1179 749 773 1031 711 637
1048 708 927 536 998 1013 844 876 592 771 921 690 915 755 1106 756 963 920 736 909 866 979 752 840 803 961 723 605 746 774
831 608 610 724 868 902 568 839 662 659 637 648 558 912 585 670 667 866 661 905 931 688 776 769 778 644 805 924 847 570 730 499 1004 730 547 531
897 781 663 773 880 874 505 944 856 946 628 602 477 707 629 555 523 706 812 629 965 727 1075 606 658 666 765 658 1089 730 671 1001 706 580 892
738 958 496 910 760 703 680 660 746 851 785 918 648 955 698 778 768 592 800 692 725 568 681 773 847 733 698 762 820
414 300 337 337 402 523 291 436 364 346 332 340 296 457 328 370 351 449 357 503 449 366 414 387 361 406 363 486 409 290 363 231 567 431 292 266
440 473 409 453 462 446 275 480 486 501 347 319 214 393 345 337 264 394 442 367 450 415 583 302 331 354 420 327 572 380 313 552 365 332 505 402
483 264 500 440 415 406 302 451 454 387 503 311 534 407 380 430 342 362 390 344 323 360 403 439 439 335 419 486
704 503 524 658 658 716 552 731 568 460 613 465 495 765 404 555 607 810 655 781 733 532 737 642 703 594 766 750 819 526 740 509 827 657 495 529
779 688 516 657 860 750 312 767 598 790 542 521 427 586 589 510 549 513 727 543 906 504 818 581 522 626 678 525 923 651 605 823 580 502 622 571
803 441 687 629 489 554 596 717 808 627 700 697 696 531 764 603 443 712 639 663 506 516 670 794 570 671 608 641
815 588 671 724 830 927 560 881 557 638 625 759 502 851 687 704 854 961 702 986 985 710 830 730 721 747 781 989 806 648 750 506 1159 834 544 696
938 871 766 653 916 865 668 869 766 1006 628 717 594 719 528 705 629 680 907 735 929 898 1138 625 600 629 654 670 1143 697 675 881 844 658 857
766 893 459 976 896 727 758 700 663 846 820 804 784 1048 653 946 733 672 821 752 818 686 809 868 875 742 600 699 817
798 619 634 662 743 697 548 822 709 603 664 573 500 780 493 654 583 822 667 839 834 696 717 651 784 614 738 797 685 515 714 478 895 736 606 581
801 710 584 687 858 787 459 813 726 801 569 599 469 528 630 603 521 582 809 535 936 613 890 604 584 671 646 536 968 668 702 887 624 524 736 564
895 473 739 726 578 720 649 733 794 679 749 692 736 689 805 631 616 714 755 635 543 667 766 815 659 719 636 662
692 551 637 578 627 716 523 785 634 557 599 561 407 558 519 630 568 732 744 800 670 585 643 640 675 692 667 789 538 503 684 497 863 680 557 679
804 720 653 690 744 745 498 542 651 764 619 586 436 497 602 709 565 651 687 520 868 732 911 491 518 629 545 595 836 596 560 753 551 468 673 432
740 480 679 768 465 687 496 722 775 504 623 708 694 592 699 594 566 651 718 651 535 614 641 809 559 670 502 664
797 634 674 667 793 663 629 907 606 532 617 516 562 716 536 662 576 914 724 815 709 636 735 801 887 627 831 856 713 574 927 717 755 643 637 639
778 617 614 794 865 760 419 758 645 826 571 518 613 545 794 660 666 607 635 516 1068 575 888 516 585 801 704 669 859 744 719 890 488 482 719 442
892 506 710 746 486 679 532 753 825 539 692 816 659 560 825 648 462 893 704 893 584 596 610 903 472 780 581 543
629 678 715 492 737 665 460 683 506 642 645 625 548 626 533 622 544 774 470 642 628 720 664 689 682 678 674 646 487 469 632 480 775 566 565 545
```

Conclusión

Observaciones

Tuvimos dificultad al tratar con funciones que regresan una matriz 2D. Pensamos que deberíamos trabajar con punteros. Sin embargo, al final el programa se ejecutó sin necesidad de ellos.

En conclusión, esta práctica nos sirvió para entender mejor los algoritmos en programación paralela, y repasar las funciones más importantes de la biblioteca MPI como son bCast, Scatter y Gather.

Referencias

CppNuts, How To Return 2D Array From Function In C & C++. (11-Ene-2018). Recuperado de <https://www.youtube.com/watch?v=g1Bu4FiJVyQ>

StackOverflow. (10-Ene-2017). Matrix multiplication using Mpi_Scatter and Mpi_Gather. Recuperado de <https://stackoverflow.com/questions/41575243/matrix-multiplication-using-mpi-scatter-and-mpi-gather>