



Universidad Nacional de Entre Ríos

FACULTAD DE INGENIERÍA

TRABAJO PRÁCTICO N° 1

Computación de Alto Rendimiento

Autor:

Justo Garcia

Agosto 2023

Tabla de contenidos

| | | |
|----------|----------------------------------|-----------|
| 1 | Introducción | 2 |
| 2 | Ejercicio 1 | 3 |
| 2.1 | Consigna | 3 |
| 2.2 | Resolución | 3 |
| 2.2.1 | Implementación con MPI | 3 |
| 2.2.2 | Ejecución en cluster | 5 |
| 2.2.3 | Análisis de resultados | 7 |
| 3 | Ejercicio 2 | 12 |
| 3.1 | Consigna | 12 |
| 3.2 | Resolución | 12 |
| 3.2.1 | Implementación con MPI | 12 |
| 3.2.2 | Ejecución en cluster | 14 |
| 3.2.3 | Análisis | 15 |
| 4 | Ejercicio 3 | 18 |
| 4.1 | Consigna | 18 |
| 4.2 | Resolución | 18 |
| 4.2.1 | Implementación con MPI | 18 |
| 4.2.2 | Ejecución | 21 |
| 4.2.3 | Análisis | 21 |
| 5 | Ejercicio 4 | 24 |
| 5.1 | Consigna | 24 |
| 5.2 | Resolución | 24 |
| 5.2.1 | Implementación con MPI | 24 |
| 5.2.2 | Ejecución | 28 |
| 5.2.3 | Análisis | 28 |

1 Introducción

En este informe desarrollaré las actividades propuestas para el trabajo práctico número 1 de Computación de Alto Rendimiento. Iré adjuntando los códigos necesarios, sin embargo recomiendo revisar el repositorio de GitHub [\[1\]](#) de este trabajo para obtener una mejor visualización de las soluciones propuestas.

2 Ejercicio 1

2.1 Consigna

Dados dos procesadores P_0 y P_1 el tiempo que tarda en enviarse un mensaje desde P_0 hasta P_1 es una función de la longitud del mensaje $T_{comm} = T_{comm}(n)$, donde n es el número de bytes en el mensaje. Si aproximamos esta relación por una recta, entonces

$$T_{comm} = l + \frac{n}{b}$$

donde l es la “latencia” y b es el “ancho de banda”. Ambos dependen del hardware, software de red (capa de TCP/IP en Linux) y la librería de paso de mensajes usada (MPI). Para determinar el ancho de banda y latencia de la red, escribir un programa en MPI que envíe paquetes de diferente tamaños y realice una regresión lineal con los datos obtenidos. Obtener los parámetros para cualquier par de procesadores en el cluster. Comparar con los valores nominales de la red utilizada (por ejemplo, para Fast Ethernet: $b \approx 100\text{Mbit/sec}$, $l = O(100\mu\text{sec})$).

2.2 Resolución

Adjunto la resolución del ejercicio uno debajo. Sin embargo, recomiendo visualizarlo en el repositorio de GitHub de este trabajo [\[2\]](#).

2.2.1 Implementación con MPI

```
1 #include <stdio.h>
2 #include <mpi.h>
3 #include <iostream>
4 #include <vector>
5 #include <cmath>
```

```
6
7 int main(int argc, char **argv) {
8
9     freopen("salida.txt", "w", stdout);
10
11     int ierror, rank, size;
12     int mtag = 0;
13
14     // Defino el máximo exponente al cual quiero que se eleve dos.
15     int max = 30;
16     double tiempo;
17     MPI_Init(&argc, &argv);
18     MPI_Status status;
19     MPI_Comm_rank(MPI_COMM_WORLD, &rank); //Paso por referencia rank y la
    modifica
20     MPI_Comm_size(MPI_COMM_WORLD, &size);
21     std::vector<int> vectorGrande;
22     double tInicio, tFin;
23     if(rank == 0)
24         printf("Tiempo, Tamaño\n");
25
26     // Implemento un ciclo para que se realice muchas veces y así \
    representar mejor
27
28     for (int repeticion = 0 ; repeticion < 10 ; repeticion++)
29         for(int i=0; i<=max; i++)
30             {
31                 // Se calculo el número de elementos del vector
32                 unsigned long tamaño = pow(2, i);
33                 vectorGrande.resize(tamaño);
34                 for(int i = 0; i<tamaño; i++)
35                     {
36                         vectorGrande[i] = i;
37                     }
38
39                 // Si es el proceso 0 se envía al otro proceso y recibe el
    tiempo \
40                 de fin para luego imprimirlo.
41                 if(rank == 0)
42                     {
```

```
43         MPI_Barrier(MPI_COMM_WORLD);
44         tInicio = MPI_Wtime();
45         MPI_Send(&vectorGrande[0], tamaño, MPI_INT, 1, mtag,
MPI_COMM_WORLD);
46
47         MPI_Recv(&tFin, 1, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD, &
status);
48         tiempo = tFin-tInicio;
49         std::printf("%f,%ld\n", tiempo, tamaño);
50     }
51     else if (rank == 1)
52     {
53         // El segundo proceso recibe el vector, calcula el tiempo
54         \
55         y lo envía
56         MPI_Barrier(MPI_COMM_WORLD);
57         MPI_Recv(&vectorGrande[0], tamaño, MPI_INT, 0, mtag,
MPI_COMM_WORLD, &status);
58         tFin = MPI_Wtime();
59
60         MPI_Send(&tFin, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
61     }
62 }
63
64
65 MPI_Finalize();
66
67 return 0;
68
69 }
```

2.2.2 Ejecución en cluster

Habiendo resuelto la codificación del ejercicio, me conecté al cluster por SSH y envié mi código. Una vez allí, lo compilé y establecí distintos parámetros para correrlo con SLURM (1).



Fig. 1: Script run.sh para SLURM

Habiendo establecido los parámetros procedo a correrlo con:

```
1 sbatch run.sh
```

Este comando me devuelve el identificador de la tarea lanzada y gestionará su ejecución en base a la disponibilidad de cómputo.

Luego observé el estado de mi tarea(2) con:

```
1 squeue -l
```

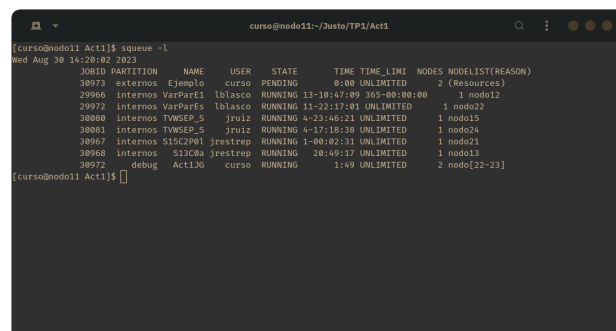


Fig. 2: Tarea corriendo

Finalizada su ejecución, copié la salida en mi dispositivo y realicé el análisis descripto a continuación

2.2.3 Análisis de resultados

Habiendo obtenido los datos de latencias realicé un procesamiento de datos con Python, análisis que recomiendo ver en el repositorio de GitHub [\[2\]](#).

La salida de mi código fue pensada como valores separados por coma para luego manipularlo facilmente como un archivo de tipo csv. Por ello simplemente hice uso de la librería *pandas* para cargarla como DataFrame y luego analizarla.

Al momento de la implementación decidí que la medición se realizara 10 veces para poder representar mejor los tiempos para cada tamaño. Por ello tuve que hacer un preprocesamiento para promediarlos. Habiendo realizado esto, obtuve la siguiente tabla:

| Tamano | Tiempo |
|------------|----------|
| 1 | 0.000023 |
| 2 | 0.000021 |
| 4 | 0.000021 |
| 8 | 0.000022 |
| 16 | 0.000022 |
| 32 | 0.000022 |
| 64 | 0.000022 |
| 128 | 0.000022 |
| .. | |
| 8192 | 0.000023 |
| 16384 | 0.000024 |
| 32768 | 0.000027 |
| 65536 | 0.000031 |
| 131072 | 0.000039 |
| 262144 | 0.000054 |
| 524288 | 0.000082 |
| 1048576 | 0.000139 |
| ... | |
| 1073741824 | 0.120231 |

Tabla 1: DataFrame obtenido tras la carga de datos

Con los datos ya cargados y promediados hice un análisis exploratorio de los datos con un ScatterPlot 3.

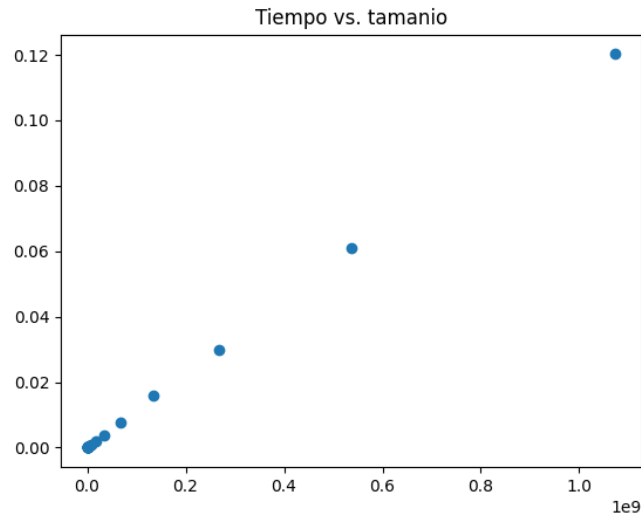


Fig. 3: Scatter plot de los datos

Se puede observar que los datos parecen tener una relación lineal. Luego, importé de la librería *SciKit Learn* los módulos necesarios para hacer una regresión lineal con mis datos.

```
1 from sklearn.linear_model import LinearRegression
2
3 reg = LinearRegression()
4
5 x = dfProm[["Tamaño"]]
6 y = dfProm["Tiempo"]
7
8 reg.fit(x, y)
```

Teniendo ya la recta que mejor se ajusta a mis datos, la plotee junto al Scatter Plot (4)

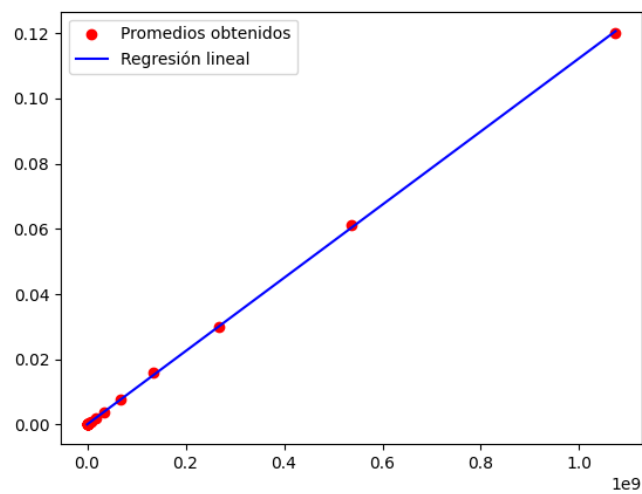


Fig. 4: Datos y la recta ajustada a ellos

Habiendo obtenido la regresión lineal, calculé la latencia como la ordenada:

```
print("l: ",reg.intercept_)  
[Out] l:  
5.690697517491978e-05
```

Fig. 5: Latencia de la red

$$l \approx 5,7$$

Luego, de la ecuación de la recta despejé el ancho de banda:

$$b = \frac{n}{T_{comm} - l}$$

Con los datos del DataFrame calculé este dato:

$$b \approx 8905586616.9$$

3 Ejercicio 2

3.1 Consigna

Para un dado número de procesadores (n) realizar una matriz de transferencia a los fines de detectar posibles heterogeneidades en la velocidad de transferencia de datos entre los procesadores. Analizar la matriz obtenida.

3.2 Resolución

3.2.1 Implementación con MPI

```
1 #include <stdio.h>
2 #include <mpi.h>
3 #include <iostream>
4 #include <vector>
5
6 using namespace std;
7
8 int main(int argc, char **argv) {
9
10     freopen("salida.csv", "w", stdout);
11
12     int ierror, rank, size;
13     MPI_Init(&argc, &argv);
14     MPI_Status status;
15     MPI_Comm_rank(MPI_COMM_WORLD, &rank); (MPI_COMM_WORLD, &size);
16     double tInicio, tFin;
17
18     // Se define un vector de tamaño {megas}.
19     int megas = 300;
20     int numElementos = megas * 1024 * 1024 / sizeof(int);
21     vector<int> mensaje;
22     mensaje.resize(numElementos);
23     for (int i = 0 ; i < numElementos ; i++)
24         mensaje[i] = i;
25 }
```

```
26     if(rank == 0)
27         printf("De,A,Tiempo\n");
28
29     // Se define un n mero de repeticiones para representar mejor el caso
30     for (int repeticiones = 0 ; repeticiones < 10 ; repeticiones++)
31         for (int i = 0 ; i<size; i++)
32             {
33                 MPI_Barrier(MPI_COMM_WORLD);
34
35                 if (rank == i)
36                     {
37                         // Se env a a todos los procesos y se miden los distintos
38                         tiempos \
39                         para luego conformar la matriz con todos los tiempos.
40                         for (int j = 0 ; j < size; j++)
41                             if (i!=j)
42                                 {
43                                     tInicio = MPI_Wtime();
44                                     MPI_Send(&mensaje[0], numElementos, MPI_INT, j, 0,
45                                     MPI_COMM_WORLD);
46                                     MPI_Recv(&tFin, 1, MPI_DOUBLE, j, 0,
47                                     MPI_COMM_WORLD, &status);
48                                     printf("%d,%d,%f\n", rank, j, tFin-tInicio);
49                                 }
50                             else
51                                 {
52                                     MPI_Recv(&mensaje[0], numElementos, MPI_INT, i, 0,
53                                     MPI_COMM_WORLD, &status);
54                                     tFin = MPI_Wtime();
55                                     MPI_Send(&tFin, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD);
56                                 }
57                             }
58
59     MPI_Finalize();
60
```

61 }

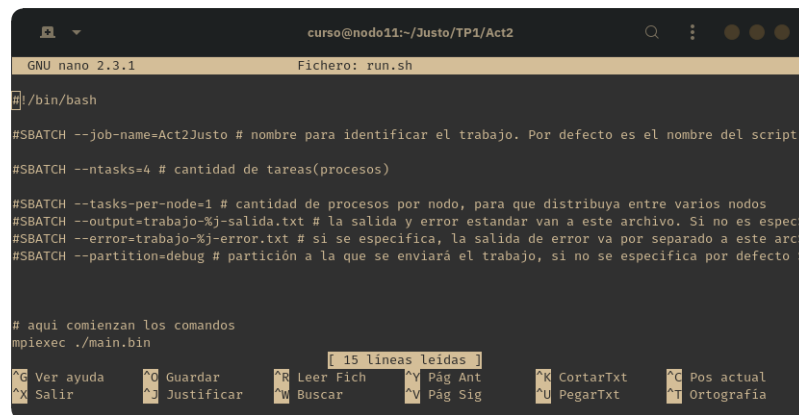
3.2.2 Ejecución en cluster

Tras haber llevado a cabo la implementación del código, lo pase al cluster y lo compile. Habiéndolo compilado pude observar que tenía 4 nodos disponibles para ejecutarlo. Por sugerencia del profesor decidí correrlo de dos maneras:

1. Con los 4 nodos y una tarea por nodo.
2. Con 4 nodos y dos tareas por nodo.

Caso 1

Para correr el primer caso definí el script de SLURM de la siguiente manera:



```
GNU nano 2.3.1          Fichero: run.sh
/bin/bash

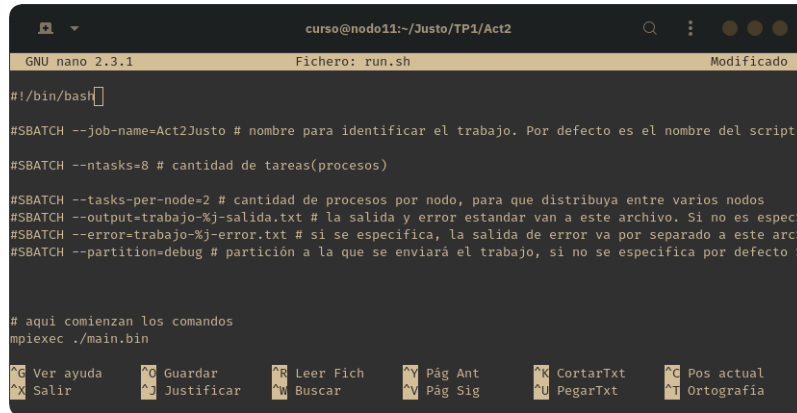
#SBATCH --job-name=Act2Justo # nombre para identificar el trabajo. Por defecto es el nombre del script
#SBATCH --ntasks=4 # cantidad de tareas(procesos)

#SBATCH --tasks-per-node=1 # cantidad de procesos por nodo, para que distribuya entre varios nodos
#SBATCH --output=trabajo-%j-salida.txt # la salida y error estandar van a este archivo. Si no es espec$
#SBATCH --error=trabajo-%j-error.txt # si se especifica, la salida de error va por separado a este arc$
#SBATCH --partition=debug # partición a la que se enviará el trabajo, si no se especifica por defecto $

# aqui comienzan los comandos
mpiexec ./main.bin
```

Fig. 6: Script para el caso 1

Caso 2 En él el script fue definido de la siguiente manera:



```

GNU nano 2.3.1          Fichero: run.sh          Modificado
#!/bin/bash

#SBATCH --job-name=Act2Justo # nombre para identificar el trabajo. Por defecto es el nombre del script
#SBATCH --ntasks=8 # cantidad de tareas(procesos)

#SBATCH --tasks-per-node=2 # cantidad de procesos por nodo, para que distribuya entre varios nodos
#SBATCH --output=trabajo-%j-salida.txt # la salida y error estandar van a este archivo. Si no es espec$
#SBATCH --error=trabajo-%j-error.txt # si se especifica, la salida de error va por separado a este arc$
#SBATCH --partition=debug # partición a la que se enviará el trabajo, si no se especifica por defecto $

# aqui comienzan los comandos
mpirun ./main.bin

^G Ver ayuda  ^O Guardar  ^R Leer Fich  ^Y Pág Ant  ^K CortarTxt  ^G Pos actual
^X Salir      ^J Justificar ^W Buscar    ^V Pág Sig  ^U PegarTxt   ^T Ortografia
  
```

Fig. 7: Script para el caso 2

Luego, corrí de igual manera que en el ejercicio 1 (2) y obtuve las salidas de ambos en mi computadora.

3.2.3 Análisis

Teniendo ambos archivos hice un procesamiento de los datos en Python siguiendo la metodología del ejercicio 1 (carga, cálculo del promedio, etc).

Habiendo obtenido un DataFrame para cada caso realicé ciertos cálculos estadísticos, que se ven reflejados en las siguientes tablas:

| | |
|---------------------------|----------|
| 4 nodos, 1 tarea por nodo | |
| Promedio | 0.032750 |
| Desvío | 0.003092 |

Tabla 2: Estadísticos para el caso 1

| | |
|----------------------------|----------|
| 4 nodos, 2 tareas por nodo | |
| Promedio | 0.006535 |
| Desvío | 0.001886 |

Tabla 3: Estadísticos para el caso 2

Como era de esperarse, el tiempo medio para el caso que tiene más de una tarea por nodo es menor. Esto se debe a que la comunidad entre procesos del mismo nodo es más rápida.

Luego, realicé mapas de calor para ambos casos de forma que pudiese visualizar gráficamente las diferencias en la latencia.

Caso 1

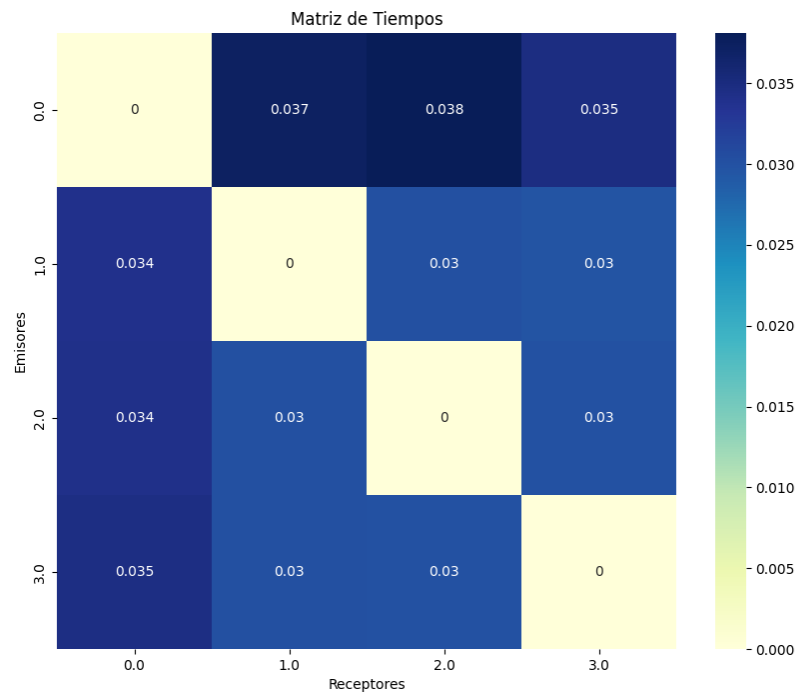


Fig. 8: Matriz para el caso 1

Se puede apreciar que cuando tomamos todos nodos distintos, las diferencias son mínimas, y la matriz obtenida es reflejada a través de la diagonal principal.

Caso 2

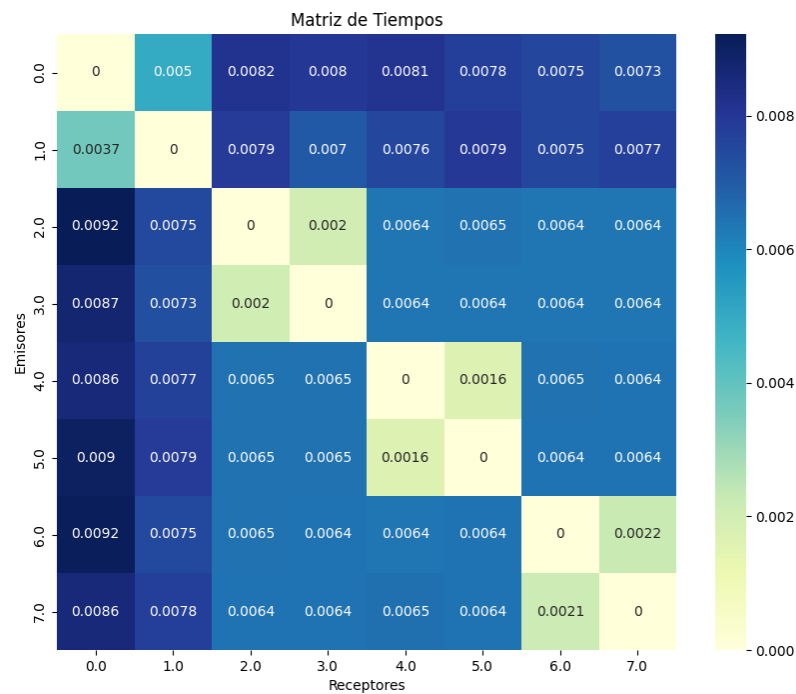


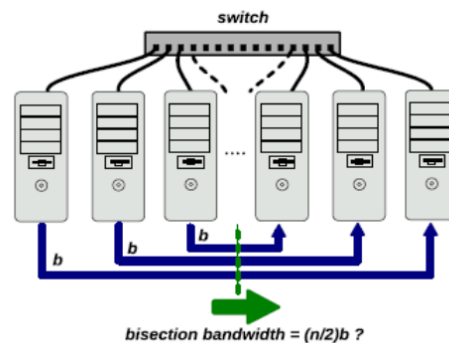
Fig. 9: Matriz para el caso 2

En este caso, si vamos tomando de a pares, se pueden ver tiempos muy bajos en comparación con el resto. Esto seguramente se deba a que son procesos que están corriendo en el mismo nodo.

4 Ejercicio 3

4.1 Consigna

El “ancho de banda de disección” de un cluster es la velocidad con la cual se transfieren datos simultáneamente desde $n/2$ de los procesadores a los otros $n/2$ procesadores. Asumiendo que la red es switchheada y que todos los procesadores están conectados al mismo switch, la velocidad de transferencia debería ser $\frac{n}{2} \cdot b$, pero podría ser que el switch tenga una máxima tasa de transferencia interna



Tomar un número creciente de n procesadores, dividirlos arbitrariamente por la mitad y medir el ancho de banda para esa partición. Comprobar si el ancho de banda de disección crece linealmente con n , es decir si existe algún límite interno para transferencia del switch.

4.2 Resolución

4.2.1 Implementación con MPI

```
1 #include <stdio.h>
2 #include <mpi.h>
3 #include <iostream>
4 #include <vector>
5
6 using namespace std;
```

```
7
8 int main(int argc, char **argv) {
9
10     freopen("salida.csv", "w", stdout);
11
12     int rank, size;
13     MPI_Init(&argc, &argv);
14     MPI_Status status;
15     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
16     MPI_Comm_size(MPI_COMM_WORLD, &size);
17     double tInicio, tFin;
18     int mtag = 0;
19
20     // Define un arreglo de 350 MB
21     int megas = 350;
22     int numElementos = megas * 1024 * 1024 / sizeof(int);
23     vector<int> paquete(numElementos);
24     for(int i = 0 ; i < paquete.size() ; i++)
25         paquete[i] = i;
26
27     if(rank == 0)
28         printf("Numero de procesadores,Tiempo\n");
29
30     // Declaro un ciclo para representar mejor la situaci n
31     int nroRepeticiones = 10;
32     for(int i = 0; i < nroRepeticiones ; i++)
33     {
34         int mitad;
35         int tama o = 2;
36
37         /* Implemento un ciclo hasta que una variable sea menor o igual al
38         n mero de procesos */
39         while (tama o <= size)
40         {
41             // Calcula la mitad
42             mitad = tama o/2;
43             MPI_Barrier(MPI_COMM_WORLD);
44             if (rank == 0)
45                 tInicio = MPI_Wtime();
```

```
46         if (rank <= tama o -1)
47         {
48             // Si es menor a la mitad env a y si no recibe.
49             if(rank <= mitad-1)
50             {
51                 // printf("\t%d envia a %d\n", rank, tama o -rank-1);
52                 MPI_Send(&paquete[0], numElementos, MPI_INT, tama o -rank
-1, mtag, MPI_COMM_WORLD);
53             }
54             else
55             {
56                 // printf("\t%d recibe de %d\n", rank, tama o -rank-1);
57                 MPI_Recv(&paquete[0], numElementos, MPI_INT, tama o -rank
-1, mtag, MPI_COMM_WORLD, &status);
58             }
59
60         }
61
62         MPI_Barrier(MPI_COMM_WORLD);
63         if(rank == 0)
64         {
65             tFin = MPI_Wtime();
66             printf("%d,%f\n", tama o , tFin-tInicio);
67         }
68         tama o += 2;
69     }
70 }
71
72
73 MPI_Finalize();
74
75 return 0;
76
77
78 }
```

4.2.2 Ejecución

A diferencia de los dos primeros su ejecución se llevó a cabo en uno de los clusters del CIMEC, por lo tanto no se cuenta con ciertos detalles informados en el resto de ejercicios.

4.2.3 Análisis

Habiendo finalizado la ejecución obtuve un archivo csv que cargue como DataFrame en python. Como corrí muchas veces el mismo proceso para representar mejor la situación, obtuve el promedio.

Con los datos ya promediados, calculé el ancho de banda de la siguiente forma:

$$b = \frac{\text{Tamaño de paquete} * \text{Nro de procesadores}}{t_{(\text{Nro de procesadores})}}$$

Con estos nuevos cálculos el DataFrame quedó de la siguiente forma:

| Nro de procesadores | Tiempo | b |
|---------------------|----------|---------------|
| 2.0 | 0.122060 | 5734.875086 |
| 4.0 | 0.118834 | 11781.140078 |
| ... | ... | ... |
| 42.0 | 0.120583 | 121907.428309 |
| 44.0 | 0.120362 | 127947.252499 |

Tabla 4: DataFrame obtenido

Con estos cálculos ya realizados procedí a graficar tanto el tiempo como el ancho de banda vs. el número de procesadores. (10, 11)

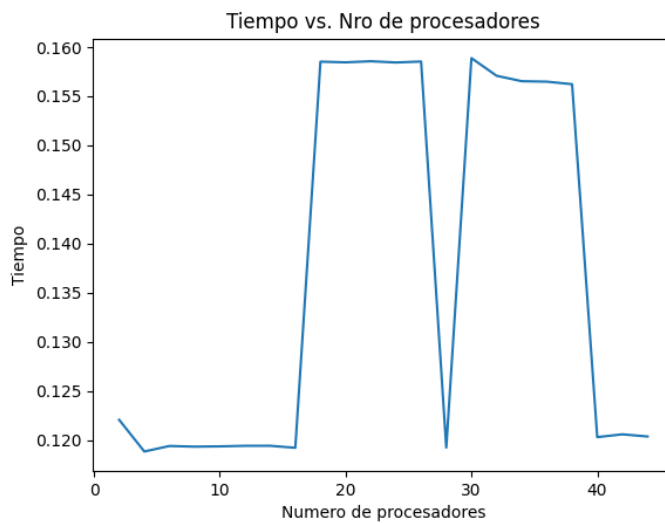


Fig. 10: Tiempo vs. el número de procesadores

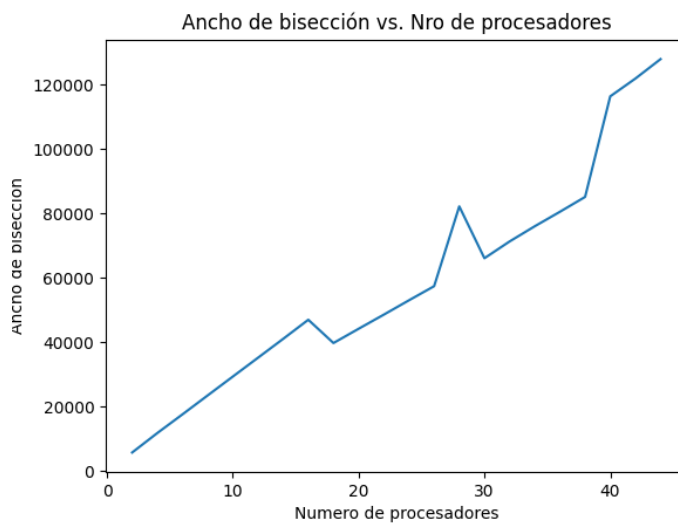


Fig. 11: Ancho de banda vs. el número de procesadores

Como se puede ver observar en la fig 10, el tiempo tiene ciertas irregularidades, pero observando los extremos, podemos ver que no aumenta significativamente. Estas irregulari-

dades pueden deberse a que la configuración de los nodos consta de dos switches distintos. Por otro lado, en la gráfica del ancho de banda no se logra apreciar que se alcance una cota. Este fenómeno se debe a que no se alcanzaron los límites del switch, en caso de disponer de más nodos o enviar un paquete más grande podría alcanzarse.

5 Ejercicio 4

5.1 Consigna

Escribir un programa que dado el archivo *homo_sapiens_chromosome_1.fasta*, lo particione en **n** partes (donde **n** es el número total del procesadores), envíe cada una de esas porciones a los respectivos procesos y busque la cantidad de veces que aparece la base A (adenina), devolviendo al master la cantidad de veces que cada proceso contabilizó. Finalmente calcule el porcentaje de A en el cromosoma.

5.2 Resolución

5.2.1 Implementación con MPI

```
1 #include <stdio.h>
2 #include <mpi.h>
3 #include <iostream>
4 #include <fstream>
5 #include <string>
6 #include <vector>
7
8 using namespace std;
9
10 /// @brief Funci n auxiliar para pasar por referencia un fragmento \
11     de secuencia y calcule el n mero de cada ocurrencia.
12 /// @param secuencia
13 /// @return [nro A, nro T, nro C, nro G, nro otro]
14 vector<int> analizarSecuencia(string *secuencia)
15 {
16     vector<int> recuento(5, 0);
17     for (int i = 0 ; i < secuencia->length() ; i++)
18     {
19         switch (secuencia->at(i))
20         {
21             case 'A':
22                 recuento[0]++;
```

```
23         break;
24     case 'T':
25         recuento[1]++;
26         break;
27     case 'C':
28         recuento[2]++;
29         break;
30     case 'G':
31         recuento[3]++;
32         break;
33     default:
34         recuento[4]++;
35         break;
36     }
37 }
38
39 return recuento;
40 }
41
42 int main(int argc, char **argv) {
43
44     int ierror, rank, size;
45     int entrada = 8;
46     int mtag = 0;
47
48     freopen("salida.csv", "w", stdout);
49     string ruta = "data/homo_sapiens_chromosome_1.fasta";
50
51     uint largo;
52
53     MPI_Init(&argc, &argv);
54     MPI_Status status;
55     MPI_Comm_rank(MPI_COMM_WORLD, &rank); //Paso por referencia rank y la
56     modifica
57     MPI_Comm_size(MPI_COMM_WORLD, &size);
58
59     vector<int> recuento;
60     vector<int> recuentoTotal(5);
```

```
61
62  /* Hago que el proceso 0 se encargue de cargar la secuencia,
63  fragmentarla y enviar estos fragmentos a qui n corresponda*/
64  if (rank == 0)
65  {
66      ifstream archivo(ruta);
67      string secuencia, linea;
68
69      while(getline(archivo, linea))
70      {
71          secuencia += linea;
72      }
73      string fragmentoAnalisis;
74
75      // Calcula el largo del fragmento.
76      int largoSecuencia = secuencia.length();
77      int largoFragmento = largoSecuencia / size;
78      printf(largoSecuencia);
79      for (int i = 0; i < size ; i++)
80      {
81          string fragmento;
82          int posInicio = i * largoFragmento;
83          int posFinal = (i == size - 1) ? largoSecuencia : (i+1) *
largoFragmento;
84          fragmento = secuencia.substr(posInicio, posFinal-posInicio);
85          largo = fragmento.size();
86
87          /* Env a primero el largo de la secuencia y luego el
fragmento.
88          Analizando el c digo en fr o podr a haber hecho un
broadcast del largo.
89          (o un Scatter, pero al momento de su realizaci n no lo vimos)
*/
90          if (i!=rank)
91          {
92              MPI_Send(&largo, 1, MPI_INT, i, mtag, MPI_COMM_WORLD);
93              MPI_Send(fragmento.c_str(), largo, MPI_CHAR, i, mtag,
MPI_COMM_WORLD);
94          }
```

```
95         else
96         {
97             fragmentoAnalisis = fragmento;
98         }
99     }
100
101     recuento = analizarSecuencia(&fragmentoAnalisis);
102
103 }
104 else
105 {
106     int largo;
107     MPI_Recv(&largo, 1, MPI_INT, 0, mtag, MPI_COMM_WORLD, &status);
108
109     char *buf = new char[largo];
110     MPI_Recv(buf, largo, MPI_CHAR, 0, mtag, MPI_COMM_WORLD, &status);
111     string fragmento(buf, largo);
112     delete [] buf;
113
114     recuento = analizarSecuencia(&fragmento);
115
116 }
117 /* Luego de haber analizado los fragmentos con la funci n auxiliar
118 hago un reduce para sumar los resultados*/
119 MPI_Reduce(&recuento[0], &recuentoTotal[0], 5, MPI_INT, MPI_SUM, 0,
120 MPI_COMM_WORLD);
121
122 /* Imprimo en formato csv para su posterior an lisis */
123 if (rank==0){
124     printf("Base,Ocurrencia\n");
125     printf("A,%d\n", recuentoTotal[0]);
126     printf("T,%d\n", recuentoTotal[1]);
127     printf("C,%d\n", recuentoTotal[2]);
128     printf("G,%d\n", recuentoTotal[3]);
129     printf("Otro,%d\n", recuentoTotal[4]);
130 }
131 MPI_Finalize();
132
```

133 }

5.2.2 Ejecución

Para compilar el código propuesto utilicé el comando:

```
1 mpicxx.mpich -o main.bin main.cpp
```

Habiendo compilado el código procedí a correrlo con el siguiente comando:

```
1 mpiexec.mpich -n n ./main.bin
```

Siendo n el número de procesos.

Tras finalizada su ejecución, la salida se guarda con formato csv para que luego sea fácilmente manipulado con la librería pandas.

5.2.3 Análisis

Habiendo obtenido el número de ocurrencias procedí a cargar los datos a un DataFrame(5) de la librería pandas [3]. Allí generé gráficas de barra con el número de ocurrencias para cada una de las bases (12). Luego realicé el cálculo de los porcentajes y, de la misma forma que las ocurrencias, las visualicé (13).

| Base | Ocurrencia | Porcentaje |
|------|------------|------------|
| A | 67070278 | 26.94 |
| T | 67244164 | 27.01 |
| C | 48055045 | 19.30 |
| G | 48111529 | 19.33 |
| Otro | 18475491 | 7.42 |

Tabla 5: DataFrame obtenido tras la carga de datos

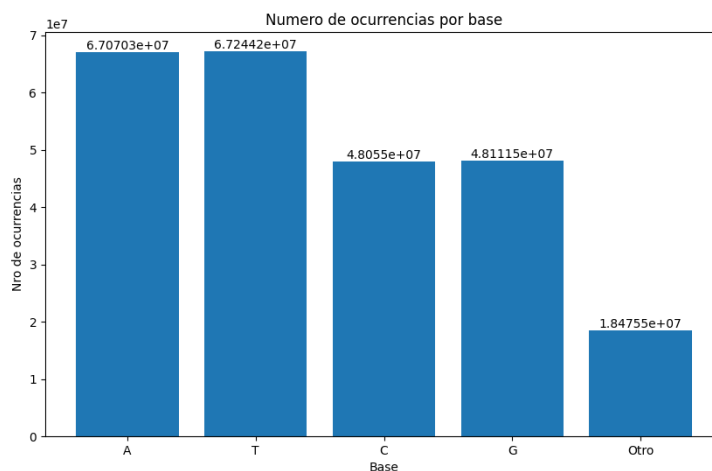


Fig. 12: Gráfico de barra del número de ocurrencia por base

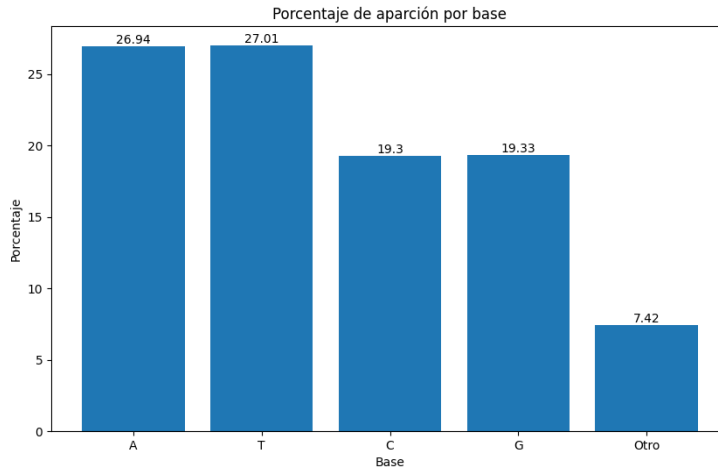


Fig. 13: Gráfico de barra del porcentaje de aparición de cada base

Por otro lado, decidí comprobar los resultados obtenidos con un script de bash. En él conte el número de ocurrencias con utilidades propias de este intérprete y obtuve los mismos resultados.

El script mencionado es el siguiente:

```
#!/bin/bash
ocurrencias=$(grep -o "A" data/*fasta | wc -l)
echo "El número de ocurrencias de A es: $ocurrencias"

ocurrencias=$(grep -o "T" data/*fasta | wc -l)
echo "El número de ocurrencias de T es: $ocurrencias"

ocurrencias=$(grep -o "G" data/*fasta | wc -l)
echo "El número de ocurrencias de G es: $ocurrencias"

ocurrencias=$(grep -o "C" data/*fasta | wc -l)
echo "El número de ocurrencias de C es: $ocurrencias"
```

Fig. 14: Script implementado

Tras ejecutarlo, obtuve la siguiente salida (15), que como se puede apreciar, concuerda con los datos obtenidos del procesamiento en paralelo con MPI:

A terminal window with a dark background and light-colored text. The window title bar shows the user 'justo' on a system 'pop-os' at the directory '~/Documentos/Facu/Segundo año/Co...'. The terminal content shows a prompt 'justo@pop-os:~/Documentos/Facu/Segundo año/Computación de alto rendimiento/Practica-CAR/TP 1/EJ 4\$' followed by the command './script.sh'. The output consists of four lines: 'El número de ocurrencias de A es: 67070278', 'El número de ocurrencias de T es: 67244164', 'El número de ocurrencias de G es: 48111529', and 'El número de ocurrencias de C es: 48055045'. The prompt is repeated at the bottom of the visible output.

```
justo@pop-os: ~/Documentos/Facu/Segundo año/Co...
justo@pop-os:~/Documentos/Facu/Segundo año/Computación de alto rendimiento/Practica-CAR/TP 1/EJ 4$ ./script.sh
El número de ocurrencias de A es: 67070278
El número de ocurrencias de T es: 67244164
El número de ocurrencias de G es: 48111529
El número de ocurrencias de C es: 48055045
justo@pop-os:~/Documentos/Facu/Segundo año/Computación de alto rendimiento/Practica-CAR/TP 1/EJ 4$
```

Fig. 15: Salida obtenida de script.sh

Referencias

- [1] Garcia Justo. *Repositorio del trabajo*. URL: <https://github.com/justog220/Practica-CAR/blob/main/TP%201/>.
- [2] justog220. *Practica-CAR*. original-date: 2023-08-26T20:44:22Z. Aug. 30, 2023. URL: <https://github.com/justog220/Practica-CAR> (visited on 08/30/2023).
- [3] *pandas.DataFrame* — *pandas 2.0.3 documentation*. URL: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html> (visited on 08/29/2023).