Práctica 7: TOEFL.

Karina Flores García y Gabriel Alejandro Herrera Gandarela

***Abstract*— In this practice, the pipline architectural pattern is implemented to perform a functional decomposition of the tasks. The main task is a program that scores TOEFL tests.**

I. OBJETIVOS.

- Aplicar los métodos de comunicación colectiva.
- Implementar el patron arquitectonico de Pipeline.
- Realizar un procesamiento masivo de datos con técnicas de HPC.

II. INTRODUCCIÓN.

Estructura Pipline.

En esta técnica el problema se divide en una serie de tareas secuenciales. Cada tarea secuencial es realizada por un procesador independiente.
Cada procesador ejecuta una fracción del algoritmo.
Para llegar a su máxima eficiencia operacional se debe llenar la tubería para que a cada ciclo del reloj, salga un resultado.

Nota: La tubería está llena cuando cada nodo o proceso se encuentra ejecutando una tarea. Ninguno se encuentra en estado idle.

III. ANÁLISIS DEL PROBLEMA.

TOEFL ITP (Test of English as a Foreign Language Institutional Testing Program) es una prueba estandarizada de dominio del idioma inglés, específicamente orientada a hablantes no nativos de este idioma. Como estándar, esta prueba es aceptada por muchas instituciones académicas y profesionales de habla Inglesa alrededor del mundo. Tiene calificación máxima de 677 puntos.
Cada una de las secciones es calificada con una tabla de conversiones que depende del número de respuestas correctas. El puntaje final es un valor entero truncado y es obtenido con:

$$PF = \frac{PC1 + PC2 + PC3}{3} \times 100 \begin{cases} PF = Puntaje\,Final, \\ PC = Puntaje\,de\,Conversión. \end{cases}$$

Es necesario escribir un programa que califique los examenes respecto a las secciones y las escalas de evaluación para cada una. Existen dos forman de solucionar el problema, de forma secuencial y utilizando un pipline.

SCORE CONVERSION TABLE 1

Number Correct (Cs)	Converted Score Section 1	Converted Score Section 2	Converted Score Section 3
50	68	——	67
49	66	——	66
48	64	——	65
47	63	——	63
46	61	——	61
45	61	——	60
44	60	——	59
43	59	——	58
42	58	——	57
41	57	——	56
40	56	68	55
39	55	67	54
38	55	66	54
37	54	64	53
36	53	62	52
35	52	60	52
34	51	59	51
33	51	58	50
32	50	56	49
31	49	55	48
30	49	54	48
29	48	54	47
28	48	52	46
27	47	51	46
26	47	50	45
25	46	49	44
24	46	48	43
23	45	47	43
22	44	46	42
21	44	45	41
20	43	44	40
19	43	43	39
18	42	42	38
17	41	41	37
16	40	40	36
15	40	39	35
14	39	38	34
13	37	37	33
12	36	36	31
11	35	34	30
10	34	34	29
9	33	32	28
8	32	30	28
7	31	29	27
6	30	28	26
5	29	26	25
4	28	25	24
3	27	24	23
2	25	22	23
1	24	21	22
0	20	20	21

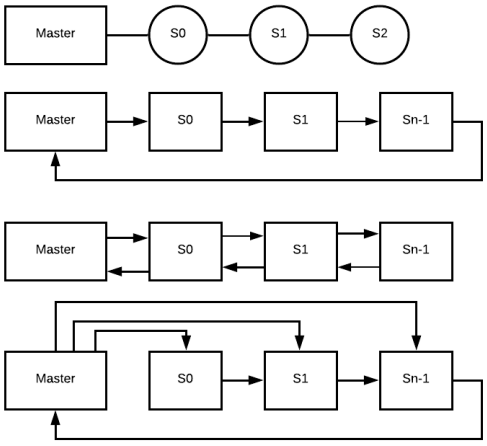
IV. IMPLEMENTACIÓN

A. Desarrollo

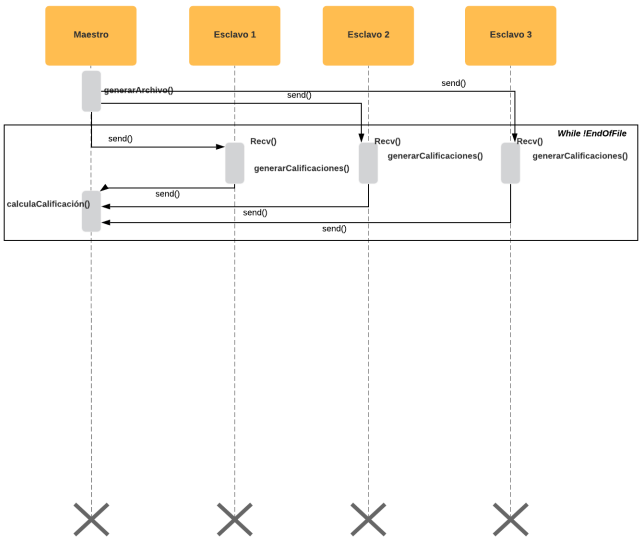
Para poder generar el archivo de registro de examenes se utiliza una variable que genere hasta 20000 id,para el puntaje de cada sección se debe utilizar una variable aleatoria que genere un numero con la función random(). Es necesario tener cuidado de que al momento de utilizar la función el puntaje para la sección uno y tres de números entre 0 y 50, para la sección dos, números entre 0 y 40.
Todos los registros son guardados en un archivo *registros.csv* manejado con las propiedades del lenguaje.
Para implementar la conversión en el caso del programa paralelo, para utilizar el patrón de *Pipline* se crea un maestro y tres esclavos, cada esclavo se encargara de la conversión de una sección del examen.
Se utiliza comunicación entre procesos con las funciones *send()* y *recv()* de MPI.
El maestro es el encargado de enviarle a todos los esclavos el arreglo que contiene los puntajes de cada sección para cada registro del archivo *registros.csv*, cada esclavo recibe su parte, hace la conversión y se lo envía al esclavo siguiente. Es importante recalcar que el último esclavo junta todas las

conversiones de los puntajes y el id en un arreglo, además de calcular el puntaje final con esta información. A su vez le regresa dicho arreglo al maestro para que este lo imprima en un nuevo archivo *toefl.csv* que contendra todos los examenes calificados.

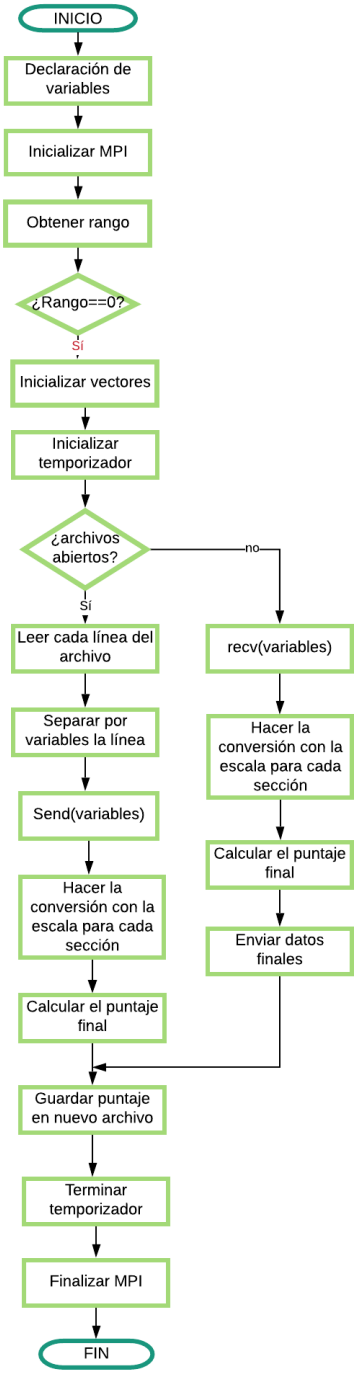
B. Diagrama de patrón arquitectónico



C. Diagrama UML de secuencia



D. Flujo del programa



V. CÓDIGO FUENTE

A. Código para generar registros

```
clock_t tiempo_inicio;
  //Calcula el tama o de los arreglos
FILE * archivo = fopen("registros.csv","w");
if(archivo == NULL){
  perror("Error en la apertura del archivo");
  return 1;
}else{
  tiempo_inicio = clock();
  //Crea 20000 nombres aleatorios y los guarda en
   un archivo
```

```
    for(int i=1;i<=total_nomb;i++){
      char registros[40];
      ID = i;
      numeroReg1 = rand() % 51;
      numeroReg2 = rand() % 41;
      numeroReg3 = rand() % 51;
      sprintf(ids, "%d", ID);
      sprintf(numero1, "%d", numeroReg1);
      sprintf(numero2, "%d", numeroReg2);
      sprintf(numero3, "%d", numeroReg3);

      strcpy(registros, ids);
      strcat(registros,", ");
      strcat(registros, numero1);
      strcat(registros,", ");
      strcat(registros, numero2);
      strcat(registros,", ");
      strcat(registros, numero3);
      strcat(registros,"\n");
      fputs(registros,archivo);
    }
    fclose(archivo);
    printf("El tiempo que tardo es: %f\n", (clock()
    -tiempo_inicio)/(double)CLOCKS_PER_SEC);
    printf("Se generaron %d registros\n",
    total_nomb);
  }
}

//te amo chicken jiji te extra o
//Y yo a ti mi amor <3
```

```
karina@karina-VirtualBox:~/Documentos/Practicas_SD/p7$ gcc regis
tros.c -o registros
karina@karina-VirtualBox:~/Documentos/Practicas_SD/p7$ ./registr
os
El tiempo que tardo es: 0.007164
Se generaron 20000 registros
karina@karina-VirtualBox:~/Documentos/Practicas_SD/p7$
```

B. *Código secuencial*

```
//Versi n secuencial del generador de registros
  if(archivo == NULL && calificaciones == NULL){
    perror("Error en la apertura del archivo");
    return 1;
  }else{
    tiempo_inicio = clock();
    //buffer para guardar lo del archivo
    char linea[1024];
    //lee linea por linea del archivo
    while(fgets(linea,1024,archivo)){
      //rompe la cadena con el delimitador y nos
    brinda el id
      char *token = strtok(linea,",");
      //Solo es null cuando no hay mas en la cadena
      if(token!=NULL){
        while(token!=NULL){
          if(contador==0){
            datos[0] = atoi(token);
            contador++;
            token = strtok(NULL,",");
          }else if(contador==1){
            datos[1] = atoi(token);
            contador++;
            token = strtok(NULL,",");
          }else if(contador==2){
            datos[2] = atoi(token);
            contador++;
            token = strtok(NULL,",");
          }else{
            datos[3] = atoi(token);
            contador = 0;
            token = strtok(NULL,",");
          }
        }
      }
      //printf("ID: %d\n",datos[0] );
      int pf = 0;
      int it1 = datos[1];
      int it2 = datos[2];
      int it3 = datos[3];
      char registros[40];
      ids = datos[0];
      cal1 = seccion1[it1];
      cal2 = seccion2[it2];
      cal3 = seccion3[it3];
      pf = (cal1+cal2+cal3)/3;
      sprintf(id, "%d",ids);
      sprintf(numero1, "%d",cal1);
      sprintf(numero2, "%d", cal2);
      sprintf(numero3, "%d",cal3);
      sprintf(numero4, "%d",pf);
      strcpy(registros,id );
      strcat(registros,", ");
      strcat(registros,numero1);
      strcat(registros,", ");
      strcat(registros,numero2);
      strcat(registros,", ");
      strcat(registros,numero3);
          strcat(registros,", ");
      strcat(registros,numero4);
      strcat(registros,"\n");
      fputs(registros,calificaciones);
    }
    fclose(archivo);
    fclose(calificaciones);
    printf("El tiempo que tardo es: %f\n", (clock()
    -tiempo_inicio)/(double)CLOCKS_PER_SEC);
  }

}
```

```
karina@karina-VirtualBox:~/Documentos/Practicas_SD/p7$ gcc secue
ncial.c -o secuencial
karina@karina-VirtualBox:~/Documentos/Practicas_SD/p7$ ./secuenc
ial
El tiempo que tardo es: 0.010404
karina@karina-VirtualBox:~/Documentos/Practicas_SD/p7$
```

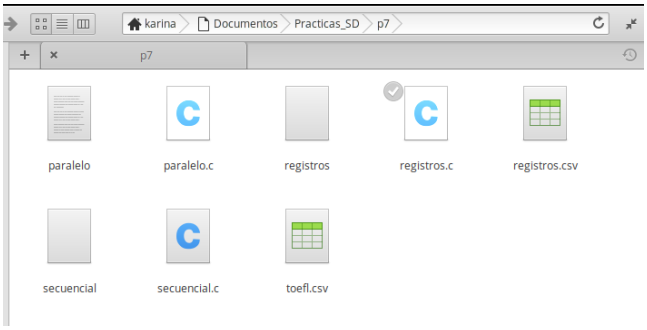
C. *Código paralelo*

```
  tiempo_inicio = MPI_Wtime();

  if(archivo == NULL){
    perror("Error en la apertura del archivo");
    return 1;
  }else{
    char linea[1024];
    while(fgets(linea,1024,archivo)){
      if(pid==0){
        char *token = strtok(linea,",");
        if(token!=NULL){
          while(token!=NULL){
            if(contador==0){
              datos[0] = atoi(token);
              contador++;
              token = strtok(NULL,",");
            }else if(contador==1){
              datos[1] = atoi(token);
              contador++;
```

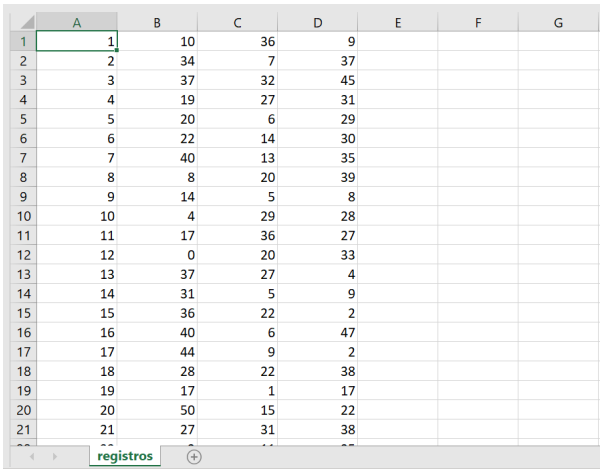
```
            token = strtok(NULL,",");
          }else if(contador==2){
            datos[2] = atoi(token);
            contador++;
            token = strtok(NULL,",");
          }else{
            datos[3] = atoi(token);
            contador = 0;
            token = strtok(NULL,",");

          }

        }
      }
      tag = 0;
      for(int i=1;i<size;i++){
        MPI_Send(&datos, 5,MPI_INT, i, tag,
MPI_COMM_WORLD);
      }
    }else if(pid==1){
      tag = 0;
      //Se omite la declaracion de todo el
arreglo
      //Corresponde a la conversi n para la
seccion1
      int seccion1[51] = {};
      int puntaje = 0;
      MPI_Recv(&datos,5,MPI_INT, 0, tag,
MPI_COMM_WORLD, &info);
      int conversion = seccion1[datos[1]];
      arrPuntaje[1] = conversion;
      MPI_Send(&arrPuntaje, 5,MPI_INT, 2, tag,
MPI_COMM_WORLD);
    }else if(pid==2){
      tag = 0;
      //Se omite la declaracion de todo el
arreglo
      //Corresponde a la conversi n para la
seccion2
      int seccion2[41] = {};
      MPI_Recv(&arrPuntaje, 5,MPI_INT, 1, tag,
MPI_COMM_WORLD, &info);
      int puntaje = 0;
      MPI_Recv(&datos, 5,MPI_INT, 0, tag,
MPI_COMM_WORLD, &info);
      int conversion = seccion2[datos[2]];
      arrPuntaje[2] = conversion;
      MPI_Send(&arrPuntaje,5,MPI_INT, 3, tag,
MPI_COMM_WORLD);
    }else if(pid==3){
      tag = 0;
      //Se omite la declaracion de todo el
arreglo
      //Corresponde a la conversi n para la
seccion3
      int seccion3[52] = {};
      MPI_Recv(&arrPuntaje, 5,MPI_INT, 2, tag,
MPI_COMM_WORLD, &info);
      MPI_Recv(&datos,5,MPI_INT, 0, tag,
MPI_COMM_WORLD, &info);
      int puntaje = 0;
      int total = 0;
      int conversion = seccion3[datos[3]];
      arrPuntaje[3] = conversion;
      arrPuntaje[0] = datos[0];
      for(int i=1;i<4;i++){
        total = total + arrPuntaje[i];
      }
      total = total/3;
      arrPuntaje[4] = total;
      MPI_Send(&arrPuntaje, 5,MPI_INT,0,tag,
MPI_COMM_WORLD);
    }
    MPI_Barrier(MPI_COMM_WORLD);
    if(pid==0){
      tag = 0;
      MPI_Recv(&arrPuntaje,5  ,MPI_INT, 3, tag,
MPI_COMM_WORLD, &info);
      if(calificaciones == NULL){
        perror("Error en la apertura del archivo"
);
        return 1;
      }else{
        char registros[40];
        sprintf(id, "%d",arrPuntaje[0]);
        sprintf(numero1, "%d",arrPuntaje[1]);
        sprintf(numero2, "%d", arrPuntaje[2]);
        sprintf(numero3, "%d", arrPuntaje[3]);
        sprintf(numero4, "%d",arrPuntaje[4]);
        strcpy(registros,id );
        strcat(registros,", ");
        strcat(registros,numero1);
        strcat(registros,", ");
        strcat(registros,numero2);
        strcat(registros,", ");
        strcat(registros,numero3);
        strcat(registros,", ");
        strcat(registros,numero4);
        strcat(registros,"\n");
        fputs(registros,calificaciones);
      }
    }
  }
  tiempo_fin = MPI_Wtime();
  tiempo_total = tiempo_fin - tiempo_inicio;
  printf("El tiempo de ejecuci n es: %f\n",
  tiempo_total);
}

fclose(archivo);
fclose(calificaciones);
MPI_Finalize();

return 0;
}
```

```
karina@karina-VirtualBox:~/Documentos/Practicas_SD/p7$ mpicc par
alelo.c -o paralelo
karina@karina-VirtualBox:~/Documentos/Practicas_SD/p7$ mpirun -n
p 4 paralelo
El tiempo de ejecución es: 0.126387
El tiempo de ejecución es: 0.126512
El tiempo de ejecución es: 0.126601
El tiempo de ejecución es: 0.126774
karina@karina-VirtualBox:~/Documentos/Practicas_SD/p7$
```

D. Archivos generados



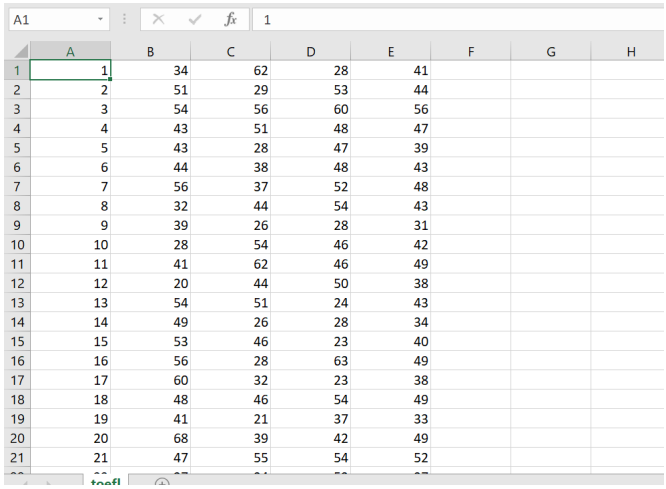
Archivo csv de registro de examenes



	A	B	C	D	E	F	G
1	1	10	36	9			
2	2	34	7	37			
3	3	37	32	45			
4	4	19	27	31			
5	5	20	6	29			
6	6	22	14	30			
7	7	40	13	35			
8	8	8	20	39			
9	9	14	5	8			
10	10	4	29	28			
11	11	17	36	27			
12	12	0	20	33			
13	13	37	27	4			
14	14	31	5	9			
15	15	36	22	2			
16	16	40	6	47			
17	17	44	9	2			
18	18	28	22	38			
19	19	17	1	17			
20	20	50	15	22			
21	21	27	31	38			

registros

Archivo csv de conversión de calificaciones para TOEFL



A1 | 1

	A	B	C	D	E	F	G	H
1	1	34	62	28	41			
2	2	51	29	53	44			
3	3	54	56	60	56			
4	4	43	51	48	47			
5	5	43	28	47	39			
6	6	44	38	48	43			
7	7	56	37	52	48			
8	8	32	44	54	43			
9	9	39	26	28	31			
10	10	28	54	46	42			
11	11	41	62	46	49			
12	12	20	44	50	38			
13	13	54	51	24	43			
14	14	49	26	28	34			
15	15	53	46	23	40			
16	16	56	28	63	49			
17	17	60	32	23	38			
18	18	48	46	54	49			
19	19	41	21	37	33			
20	20	68	39	42	49			
21	21	47	55	54	52			

toefl

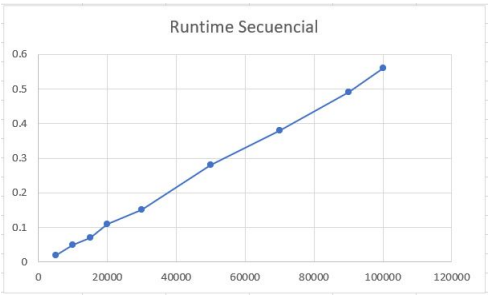
VI. COMPARACIÓN Y MÉTRICAS.

Runtime Secuencial

TABLE I

MÉTRICA DE RENDIMIENTO PARA *Runtime secuencial*.

Número de registros	Runtime (s)
5000	0.02
10000	0.05
15000	0.07
20000	0.11
30000	0.15
50000	0.28
70000	0.38
90000	0.49
100000	0.56

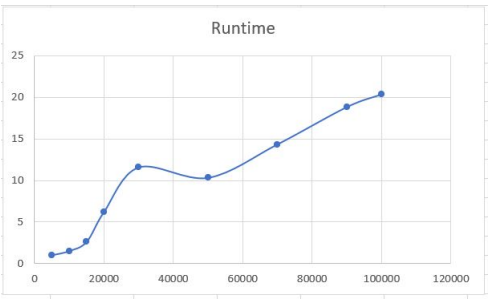


Runtime

TABLE II

MÉTRICA DE RENDIMIENTO PARA *Runtime paralelo*.

Número de registros	Runtime (s)
5000	1.01
10000	1.52
15000	2.62
20000	6.20
30000	11.56
50000	10.31
70000	14.33
90000	18.81
100000	20.34



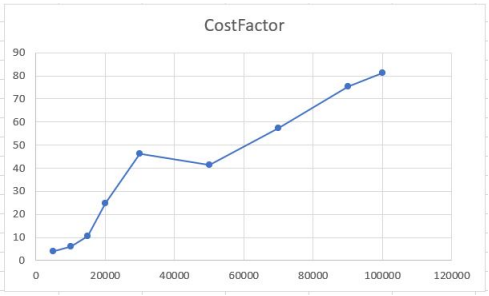
El runtime aumenta conforme los procesadores aumentan.

Cost Factor

TABLE III

MÉTRICA DE RENDIMIENTO PARA *Costfactor*.

Número de registros	Costfactor (s)
5000	4.04
10000	6.08
15000	10.48
20000	24.8
30000	46.24
50000	41.24
70000	57.32
90000	75.24
100000	81.36

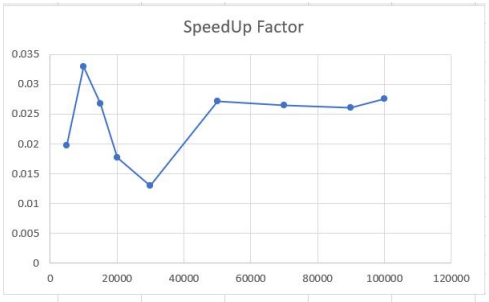


El costfactor aumenta conforme los procesadores aumentan.

Speedup Factor

TABLE IV
MÉTRICA DE RENDIMIENTO PARA *Speedup Factor*.

Número de registros	Speedup Factor (s)
5000	0.01980198
10000	0.032
15000	0.026
20000	0.017
30000	0.0129
50000	0.027
70000	0.026
90000	0.026
100000	0.027



El speedup disminuye conforme los procesadores aumentan.

Speedup Factor %

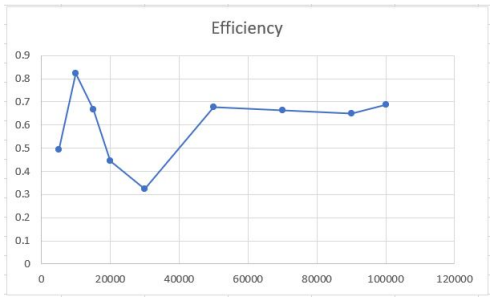
TABLE V
MÉTRICA DE RENDIMIENTO PARA *Speedup Factor porcentaje*.

Número de registros	Speedup Factor (s)
5000	-98.01%
10000	-96.71%
15000	-97.32%
20000	-98.22%
30000	-98.70.%
50000	-97.28%
70000	-97.34%
90000	-97.39%
100000	-97.24%

Efficiency

TABLE VI
MÉTRICA DE RENDIMIENTO PARA *Efficiency*.

Número de procesadores	Efficiency (s)
5000	0.49
10000	0.822
15000	0.667
20000	0.443
30000	0.324
50000	0.678
70000	0.662
90000	0.651
100000	0.688



El efficency disminuye conforme los procesadores aumentan.

VII. CONCLUSIONES

La comunicación entre procesos impletada con MPI sirve para utilizar diversos patrones arquitectónicos, en esta práctica utilizamos el de pipline.En este caso tuvimos especial cuidado para que la tubería se encontrara llena en todo momento.

También pusimos especial atención en no generar una condición de carrera, para esto dejamos que la lectura y la escritura en los archivos lo hiciera unicamente un procesador, el maestro.

Notamos como en prácticas pasadas que la implementación en paralelo resulta ser más tardada e implementa más recursos(procesadores) por lo que no conviene, el programa en secuencial es la mejor opción. Posiblemente si la cantidad de datos que se procesaran fuera mucho mayor, el programa paralelo convendría más.

Una vez más las funciones *send()* y *recv()* fueron las indicadas para la repartición de tareas ya que no necesitamos de la comunicación colectiva entre procesos.

El mayor problema al que nos enfrentamos fue la conversión de tipos de dato, debido a que las calificaciones se encontraban registradas como números enteros y al momento de escribir en el archivo debían ser convertidos a cadenas. Dicho proceso fue en el que invertimos más tiempo y lineas de código.

REFERENCES

[1] Ayala,J.A.(29 marzo 2020) Práctica 6:Generador aleatorio de nombres. Sistemas Distribuidos, 1.

[2] Reynoso, C. (marzo 2004) Estilos y Patrones en la estrategia de arquitectura de Microsoft. Buenos Aires, Argentina, 73.