Metodología de la Programación Paralela

Silvia Perez Ruiz
E-mail: silvia.perezr@um.es, DNI: 49215974E
Raúl Hernández Martínez
E-mail: raul.hernandezm@um.es, DNI: 24423648V
Eduardo Gallego Nicolás
E-mail: eduardo.gallegon@um.es, DNI: 48844607J

Práctica 2. Paralelización con MPI

Grado en Ingeniería Informática
Metodología de la Programación Paralela
Profesor: Jose Matias Cutillas Lozano (josematias.cutillas@um.es)

UNIVERSIDAD DE MURCIA

Índice

Silvia Pérez Ruiz
Raúl Hernández Martínez
Eduardo Gallego Nicolás

1 Cuestión 1.

Enunciado: Reemplazar las llamadas a la función rand por rand_r (que es "thread safe") para evitar, por un lado, que se genere la misma secuencia de valores aleatorios en diferentes hilos y, por otro, que se pierda paralelismo durante la generación de dichos valores debido a la secuencialidad que introduce la función rand.

Este apartado lo implementamos en la práctica 1. Para poder juntarlo con esta lo que vamos hacer es partir primero del código de la práctica 3 y fusionar el apartado del r_rand con el código de la práctica 3.

2 Cuestión 2.

Enunciado: Implementar un algoritmo híbrido utilizando directivas OpenMP y primitivas MPI tomando como referencia las funciones utilizadas en las prácticas 1 y 2.

Para poder realizar la fusión con mpi y omp, lo que hemos hecho ha sido implementar la comunicación síncrona ya que al revisar los tiempos de la práctica 2 nos dimos cuenta de que era nuestro mejor tiempo con diferencia.

Partiendo de esta información, tenemos ahora que elegir que partes de OMP tenemos que poner en nuestro código. Primero, elegimos que implementación de fitness poner y nuestro mejor caso con diferencia era cuando lo implementábamos con **Reduction** asi que elegimos esa. Luego para los schedules el mejor que nos funciona es ponerlo en el primer bucle donde se crean los individuos y hemos elegido ponerlo con guided y con tamaño 12 ya que dentro de todas las posibilidades era la que mejor nos iba. Por último, hemos decidido no poner la paralelización que teníamos en la función **cruzar** ya que siempre nos empeoraba y no mejoraba nunca.

Con todos estos cambios, nos salen los siguiente tiempos:

N	N_GEN	TAM_POB	N_Procesos	N_THREADS	T_EJEC	SPEEDUP	EFICIENCIA
2500	50	800	1	1	9,69 seg	1	1
2500	50	800	1	2	11,03 seg	0,8785	0,4392
2500	50	800	1	4	11,81 seg	0,8204	0,2051
2500	50	800	1	8	12,67 seg	0,7647	0,0955
2500	50	800	2	1	9,64 seg	1	1
2500	50	800	2	2	11,72 seg	0,8225	0,4112
2500	50	800	2	4	11.29 seg	0,8538	0,2044
2500	50	800	2	8	11.73 seg	0,8218	0,1027
2500	50	800	3	1	9,48 seg	1	1
2500	50	800	3	2	4,97 seg	1,9074	0,9537
2500	50	800	3	4	2,54 seg	3,7322	0,9330
2500	50	800	3	8	3,09 seg	3,0679	0,3834
2500	50	800	4	1	9,55 seg	1	1
2500	50	800	4	2	4,93 seg	1,9371	0,9685
2500	50	800	4	4	2,73 seg	3,4981	0,8745
2500	50	800	4	8	8,37 seg	1,1409	0,1426

N	N_GEN	TAM_POB	N_Procesos	N_THREADS	T_EJEC	SPEEDUP	EFICIENCIA
2500	50	800	5	1	9,75 seg	1	1
2500	50	800	5	2	5,03 seg	1,9383	0,9691
2500	50	800	5	4	2,60 seg	3,75	0,9375
2500	50	800	5	8	11,26 seg	0,8658	0,1082
2500	50	800	6	1	9,77 seg	1	1
2500	50	800	6	2	5,00 seg	1,954	0,9770
2500	50	800	6	4	3,99 seg	2,4486	0,6121
2500	50	800	6	8	28,65 seg	0,3410	0,0426

A simple vista, observamos que, independientemente del número de procesos, el tiempo es prácticamente constante con un solo hilo. Sin embargo, al analizar los tiempos para uno y dos procesos, notamos similitudes notables. A partir de tres procesos, los tiempos son consistentes entre sí, excepto en el caso de seis procesos y ocho hilos, donde se observa un tiempo significativamente alejado en comparación con las demás configuraciones.

En comparación con las prácticas anteriores, observamos que el tiempo de ejecución es casi idéntico para uno y cuatro hilos. Sin embargo, se nota una mejora al utilizar ocho hilos en las ejecuciones de las prácticas anteriores.

3 Cuestión 3.1.

3.1 Apartado a.

Enunciado: Paraleliza el bucle for correspondiente a la generación inicial de individuos (FOR_INI) que incluye el cálculo del fitness para cada individuo dentro del mismo bucle for.

Como en la cuestión anterior hemos decidido dejar justo en ese bucle la paralelización con guided, hemos tenido que modificarlo acorde a esta cuestión que pide memoria compartida con nivel 1 de paralelismo. El código se queda así:

```
//Ejercicio 3.1.a)
    #pragma omp parallel for fisrtprivate(i)
    for (i = 0; i < tam_pob; i++)
    {
        //QUITAR MALLOC
        //sub_poblacion[i] = (Individuo *)malloc(sizeof(Individuo));
        int *array = crear_individuo(n);
        for (int j = 0; j < n; j++) {
            isla_poblacion[i].array_int[j] = array[j];
        }
        free(array);

        // calcula el fitness del individuo
        fitness(d, &isla_poblacion[i], n);
    }
```

Listing 1: Primer bucle (FOR_INI).

3.2 Apartado b.

Enunciado: Modifica el código de la Práctica 1 para que el programa reciba dos nuevos parámetros: el número de hilos (N_HILOS_INI) para el bucle FOR_INI y el número de hilos (N_HILOS_FIT) para el bucle for (FOR_FIT) dentro de la función que realiza el cálculo del fitness.

Para este apartado lo que hemos hecho ha sido lo primero de todo en el main.c crear la dos nuevas variables N_HILOS_INI y N_HILOS_FIT.

```
  int n = atoi(argv[1]);
  int n_gen = atoi(argv[2]);
  int tam_pob = atoi(argv[3]);
  double m_rate = atof(argv[4]);
  int n_hilos_ini = atof(argv[5]);
  int n_hilos_fit = atof(argv[6]);
```

Listing 2: Agregación de nuevas variables.

Tras esto modificamos el aplicar_ga para que en el primer bucle se ejecute con los hilos que indique la variable N_HILOS_INI.

```
  #pragma omp parallel for private(i) num_threads(n_hilos_ini)
  for(i = 0; i < tam_pob; i++) {
    poblacion[i] = (Individuo *) malloc(sizeof(Individuo));
    poblacion[i]->array_int = crear_individuo(n);

    // calcula el fitness del individuo
    fitness(d, poblacion[i], n, n_hilos_fit);
  }
```

Listing 3: Bucle FOR_INI.

Por último, también tenemos que modificar el fitness agregando los hilos con la variable nueva N_HILOS_FIT.

```
#pragma omp parallel for reduction(+:temp_fitness) numthreads(n_hilos_fit)
  for(int i = 0; i < n-1; i++) {
  double distancia = distancia_ij(d, individuo->array_int[i], individuo->array_int[i + 1],
   n);
      temp_fitness += distancia;
  }
```

Listing 4: Bucle FOR_FIT.

3.3 Apartado c.

Enunciado: Instrumentaliza el código para que permita medir el tiempo de ejecución del bucle FOR_INI al variar el número de hilos (por ejemplo, desde 1 hasta 20, con incrementos de 2: 1, 2, 4, 6, 8, . . .). En este caso no hay que paralelizar el bucle FOR_FIT.

El tiempo lo podemos medir modificando el código tal que así:

```
//Cuestion 3.1.c)
double t_ini = mseconds();

//Cuestion 3.1.a)
//#pragma omp parallel for private(i)
//Cuestion 3.1.b
#pragma omp parallel for private(i) num_threads(n_hilos_ini)
for(i = 0; i < tam_pob; i++) {
  poblacion[i] = (Individuo *) malloc(sizeof(Individuo));
  poblacion[i]->array_int = crear_individuo(n);

  // calcula el fitness del individuo
  fitness(d, poblacion[i], n, n_hilos_fit);
}

//Cuestion 3.1.c)
double t_fin = mseconds();
  t_total_time += (t_fin - t_ini) / 1000;
printf("Tiempo de FOR_INI = %.4lf\n", t_total_time);
```

Listing 5: Medida tiempo.

Como la función fitness debemos no pararelizarla quedaría de esta manera:

```
// Cuestion 3.1.c) "En este caso no hay que paralelizar el bucle FOR_FIT"
//#pragma omp parallel for reduction(+:temp_fitness) numthreads(n_hilos_fit)
  for(int i = 0; i < n-1; i++) {
  double distancia = distancia_ij(d, individuo->array_int[i], individuo->array_int[i + 1],
   n);
      temp_fitness += distancia;
  }
```

Listing 6: Función fitness.

3.4 Apartado d.

Enunciado: Aplicar el algoritmo sobre una instancia del problema de tamaño n = 20000. Fija el valor del parámetro NE en la ecuación 1, por ejemplo, NE = 6 y el número de iteraciones del algoritmo a 0 para que sólo se ejecute la parte de generación inicial de la población.

En las ejecuciones ponemos m_rate a 0 ya que no afecta nada para el tiempo que veremos en la tabla de a continuación. Como nos dice en el apartado anterior que no hay que paralelizar el bucle de fitness, entonces obviamos el último parámetro de N_HILOS_FIT ya que no afecta para nada

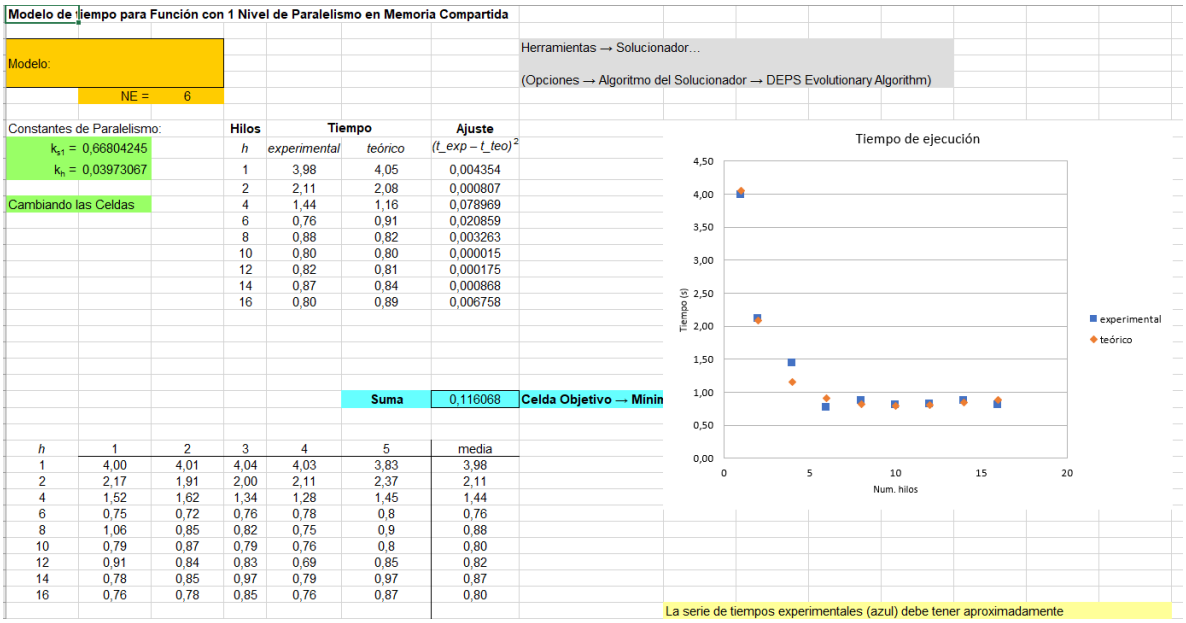
N	N_GEN	TAM_POB	N_HILOS_INI	T_EJECUCION
20000	0	6	1	5,8050 seg
20000	0	6	2	3,3580 seg
20000	0	6	4	2,2070 seg
20000	0	6	6	0,9900 seg
20000	0	6	8	1,2510 seg
20000	0	6	10	1,0890 seg
20000	0	6	12	0,9580 seg
20000	0	6	14	1,0530 seg
20000	0	6	16	1,0830 seg

4 Cuestión 3.2.

4.1 Apartado a.

Enunciado: Analiza la hoja de cálculo "Modelo 1-Nivel Paralelismo" del documento datos/solver.ods. En ella se presentan tiempos experimentales (t_experimental) frente al número de hilos, h, así como los correspondientes al modelo teórico (t_teorico) dado por la ecuación 1. Obviamente, debes reemplazar estos datos por tus tiempos obtenidos en el Apartado 3.1.

Aquí podemos ver los resultados obtenidos al introducir los datos experimentales y teóricos:



Modelo de tiempo para Función con 1 Nivel de Paralelismo en Memoria Compartida

Modelo:

NE = 6

Herramientas → Solucionador…

(Opciones → Algoritmo del Solucionador → DEPS Evolutionary Algorithm)

Constantes de Paralelismo:	Hilos	Tiempo		Ajuste
k_{s1} = 0,66804245	*h*	*experimental*	*teórico*	$(t\_exp - t\_teo)^2$
k_h = 0,03973067	1	3,98	4,05	0,004354
	2	2,11	2,08	0,000807
Cambiando las Celdas	4	1,44	1,16	0,078969
	6	0,76	0,91	0,020859
	8	0,88	0,82	0,003263
	10	0,80	0,80	0,000015
	12	0,82	0,81	0,000175
	14	0,87	0,84	0,000868
	16	0,80	0,89	0,006758
		Suma	0,116068	Celda Objetivo → Minin

h	1	2	3	4	5	media
1	4,00	4,01	4,04	4,03	3,83	3,98
2	2,17	1,91	2,00	2,11	2,37	2,11
4	1,52	1,62	1,34	1,28	1,45	1,44
6	0,75	0,72	0,76	0,78	0,8	0,76
8	1,06	0,85	0,82	0,75	0,9	0,88
10	0,79	0,87	0,79	0,76	0,8	0,80
12	0,91	0,84	0,83	0,69	0,85	0,82
14	0,78	0,85	0,97	0,79	0,97	0,87
16	0,76	0,78	0,85	0,76	0,87	0,80

La serie de tiempos experimentales (azul) debe tener aproximadamente

Figura 1: Hoja de cálculo.

Se observa cómo los datos experimentales se ajustan a los teóricos por lo que podemos pensar que los resultados obtenidos son bastantes acertados, la suma que obtenemos es de 0,116068.

4.2 Apartado b.

Enunciado: Instalar la Rutina con 1 Nivel de Paralelismo (obtener valores de constantes del sistema) a partir de los datos experimentales obtenidos en el Apartado 3.1.

Observamos que las constantes del sistema dan un valor de Ks1=0,668042448774879 y kh=0,0397306714201495. Lo que hemos hecho para conseguir esto ha sido ejecutar el solver tras insertar nuestros datos en el libreoffice, de esta forma conseguimos estas constantes.

Podemos observar estos valores en la cuestión anterior en la imagen.

5 Cuestión 3.3.

5.1 Apartado a.

Enunciado: Despeja el número de hilos óptimo (hopt) de la ecuación 3 en función de NE, ks1 y kh, donde los valores de ks1 y kh han sido obtenidos en la instalación (Apartado 3.2).

Para poder hacer este cálculo necesitamos usar esta ecuación:

$$\frac{dt}{dh} = \frac{-k_{s1} \cdot NE}{h^2} + k_h = 0$$

Para poder sustituir los valores, solo necesitamos los valores obtenidos en el excel. Despejamos h:

$$h = \sqrt{\frac{k_{s1} \cdot NE}{k_h}} = \sqrt{\frac{k_{s1} \cdot 6}{k_h}} = \sqrt{\frac{0{,}668042448774879 \cdot 6}{0{,}0397306714201495}} = 10{,}04$$

5.2 Apartado b.

Enunciado: Calcula el tiempo de ejecución que se obtendría con varios valores de h (p.e: 3, 5, 7, . . .) distintos a los usados en la obtención del modelo (Apartados 3.1 y 3.2).

Con la siguiente fórmula obtenemos los tiempos teóricos con varios hilos de h.

$$t_{un\_nivel} = \frac{k_{s1} \cdot NE}{h} + k_h \cdot h$$

Figura 2: Fórmula para sacar el tiempo de ejecución.

N_GEN	TAM_POB	N_HILOS	T_EJECUCION
0	6	3	1,4552 seg
0	6	5	1,0003 seg
0	6	7	0,8507 seg
0	6	9	0,8029 seg
0	6	11	0,8014 seg
0	6	13	0,8248 seg
0	6	15	0,8631 seg

5.3 Apartado c.

Enunciado: Realiza las correspondientes ejecuciones para ese número de hilos con NE = 6 (o el valor que hayas elegido para tu sistema) y comprueba la bondad del ajuste del modelo a los datos experimentales.

Sacamos los tiempos de as ejecuciones para los correspondientes hilos.

N_GEN	TAM_POB	N_HILOS	T_EJECUCION
0	6	3	1,5430 seg
0	6	5	1,2940 seg
0	6	7	0,7010 seg
0	6	9	0,9450 seg
0	6	11	0,9000 seg
0	6	13	0,7500 seg
0	6	15	0,7430 seg

En la siguiente captura de pantalla podemos apreciar la bondad del ajuste del modelo a los datos experimentales, y vemos que

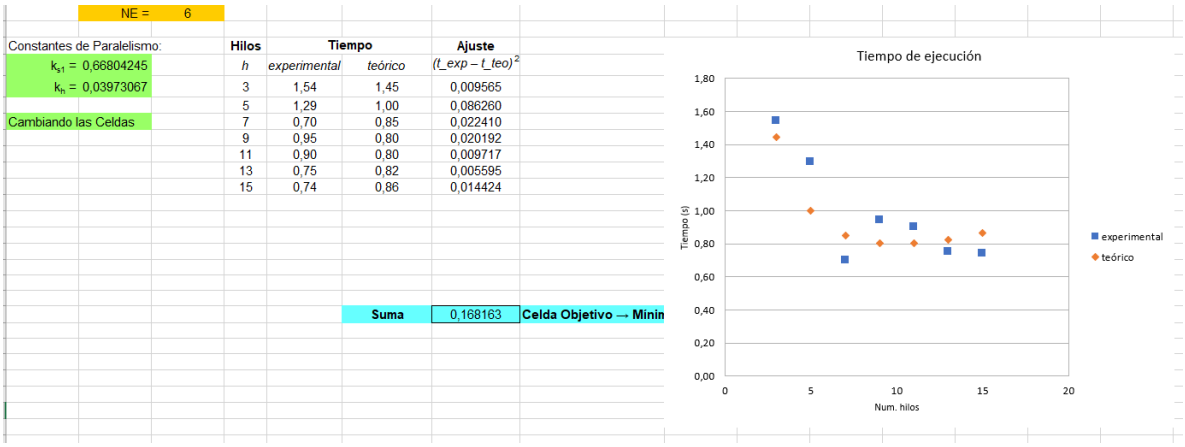


Figura 3: Bondad del ajuste del modelo.

5.4 Apartado d.

Enunciado: Varía el tamaño de la población dando valores a NE (por ejemplo 3, 10 u otros que estimes oportunos) y obtén el número de hilos óptimo (hopt) en estos casos. De esta forma, al variar el tamaño del problema no necesitamos volver a determinar experimentalmente el número de hilos óptimo, sino que podemos hacer el cálculo directamente gracias al modelo de tiempos.

Para poder calcularlo necesitamos esta fórmula otra vez:

$$\frac{dt}{dh} = \frac{-k_{s1} \cdot NE}{h^2} + k_h = 0$$

Vamos hacerlo con tres valores, el primero de todos será con NE = 3:

$$h = \sqrt{\frac{k_{s1} \cdot NE}{k_h}} = \sqrt{\frac{k_{s1} \cdot 3}{k_h}} = \sqrt{\frac{0{,}668042448774879 \cdot 3}{0{,}0397306714201495}} = 2{,}245$$

Ahora vamos hacerlo con NE = 10:

$$h = \sqrt{\frac{k_{s1} \cdot NE}{k_h}} = \sqrt{\frac{k_{s1} \cdot 10}{k_h}} = \sqrt{\frac{0{,}668042448774879 \cdot 10}{0{,}0397306714201495}} = 4{,}1005$$

Por último, vamos hacerlo con NE = 14:

$$h = \sqrt{\frac{k_{s1} \cdot NE}{k_h}} = \sqrt{\frac{k_{s1} \cdot 14}{k_h}} = \sqrt{\frac{0{,}668042448774879 \cdot 14}{0{,}0397306714201495}} = 15{,}342$$

5.5 Apartado e.

Enunciado: Ejecuta de nuevo la rutina FOR_INI con NE = 3, 10 y compara los resultados experimentales obtenidos con los arrojados por el modelo. ¿Son similares? ¿Ajusta bien el modelo?

N	N_GEN	TAM_POB	N_HILOS_INI	T_EJECUCION
20000	0	3	1	2,1070 seg
20000	0	3	2	1,3660 seg
20000	0	3	4	0,7430 seg
20000	0	3	6	0,7750 seg
20000	0	3	8	0,7330 seg
20000	0	3	10	0,7630 seg
20000	0	3	12	0,6960 seg
20000	0	3	14	0,7470 seg
20000	0	3	16	0,8780 seg

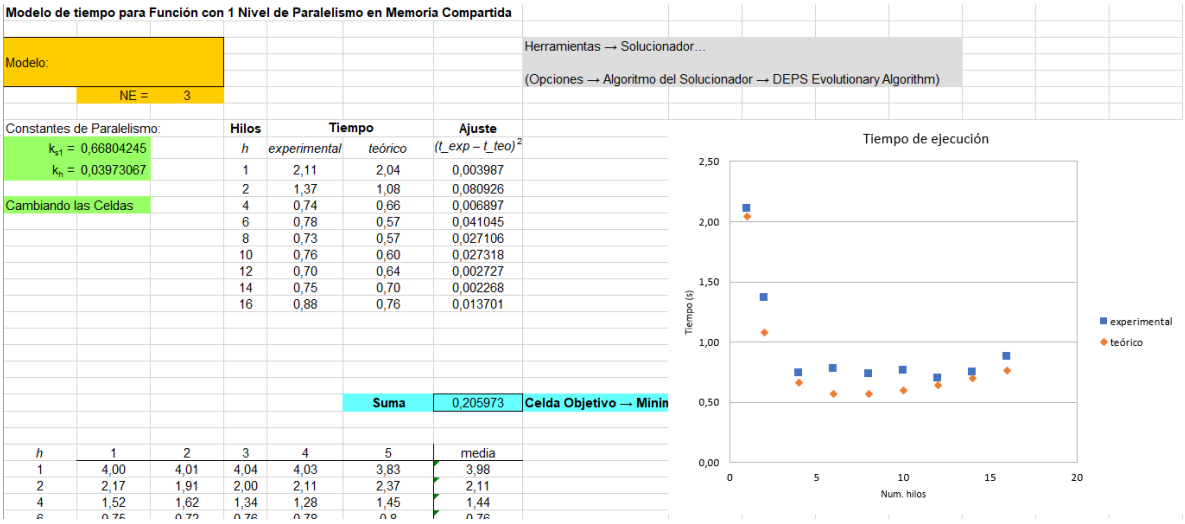


Figura 4: Bondad del ajuste del modelo para NE = 3.

N	N_GEN	TAM_POB	N_HILOS_INI	T_EJECUCION
20000	0	10	1	7,2680 seg
20000	0	10	2	3,4630 seg
20000	0	10	4	2,0660 seg
20000	0	10	6	1,5860 seg
20000	0	10	8	1,3660 seg
20000	0	10	10	1,0960 seg
20000	0	10	12	1,1250 seg
20000	0	10	14	0,9960 seg

N	N_GEN	TAM_POB	N_HILOS_INI	T_EJECUCION
20000	0	10	16	1,1170 seg

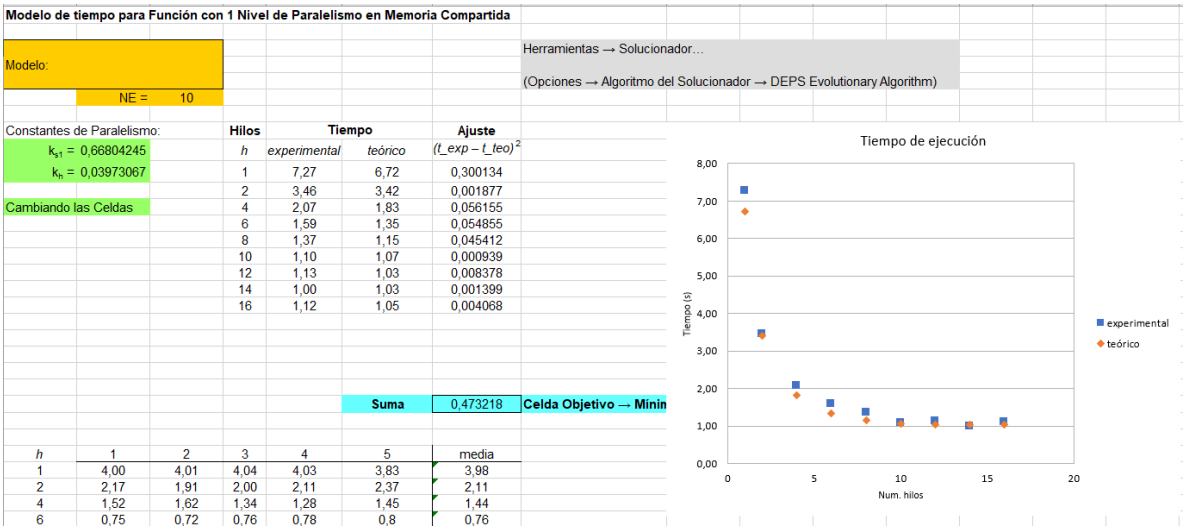


Figura 5: Bondad del ajuste del modelo para NE = 10.

Ambas son similares, pero podemos ver que el modelo se ajusta mejor cuando NE es igual a 10, aunque, cuando NE es igual a 3, el ajuste del modelo se acerca bastante igualmente.

6 Cuestión 4.

6.1 Apartado a.

Enunciado: Instalar la Rutina con 2 Niveles de Paralelismo, es decir, obtener los valores de las constantes del sistema ks2, hk1 y kh2 en la ecuación 2, donde se ha introducido el nuevo parámetro N, número de ciudades, que es el tamaño del individuo. Se debe partir de los datos experimentales obtenidos con la configuración de parámetros del Apartado 3.1(d), variando el número de hilos en ambos niveles de paralelismo (h1, h2) como se sugiere en la hoja de cálculo "Modelo 2-Niveles Paralelismo" del documento solver.ods.

El parámetro N será 20000 y NE será 6, estos datos los sacamos de la cuestión 3.1d. Insertamos los datos experimentales y los teóricos y volvemos a ejecutar el solver como en los anteriores apartados, el resultado es este:

Modelo de tiempo para Función con 2 Niveles de Paralelismo en Memoria Compartida

Modelo:

NE = 6
N = 20000

Constantes de Paralelismo:	Hilos		Tiempo		Ajuste
k_{s2} = 5,5559E-05	*h1*	*h2*	*experimental*	*teórico*	$(t\_exp - t\_teo)^2$
k_{h1} = 0,15892268	1	1	5,284	6,98	2,89307332033
k_{h2} = 0,15892269	1	2	5,087	3,81	1,62997002685
	1	4	5,38	2,46	8,5183544002
Cambiando las Celdas	1	8	5,452	2,26	10,1653433097
	1	16	5,238	3,12	4,49280257931
	2	1	2,779	3,81	1,06357390825
	2	2	2,867	2,30	0,318710714 77
	2	4	2,8	1,79	1,02633436081
	2	8	2,725	2,01	0,51707895192
	2	16	3,219	3,07	0,02251383564
	4	1	1,952	2,46	0,25946591839
	4	2	2,087	1,79	0,09004898603
	4	4	1,771	1,69	0,00687695479
	4	8	1,984	2,12	0,017270628
	4	16	1,674	3,28	2,58767907543
	8	1	1,074	2,26	1,41535370811
	8	2	1,065	2,01	0,88532655299
	8	4	0,923	2,12	1,4218600403
	8	8	0,945	2,65	2,8965850699
	8	16	0,969	3,87	8,39394614109
	16	1	1,155	3,12	3,85484791708
	16	2	0,964	3,07	4,4308303941
	16	4	1,105	3,28	4,74205655886
	16	8	0,982	3,87	8,31878676696
	16	16	1,014	5,11	16,7900716334

modificar/ampliar valores de *h1* y *h2* como se considere mejor

suma 30,6477554 Celda objetivo → mínimo

Figura 6: Datos del modelo 2 niveles de paralelismo.

Vemos el valor de las constantes, ks2=0,0000555588192951925, kh1=0,158922677013484 y kh2=0,158922685119861.

Los gráficos que muestra con nuestros datos son los siguientes:

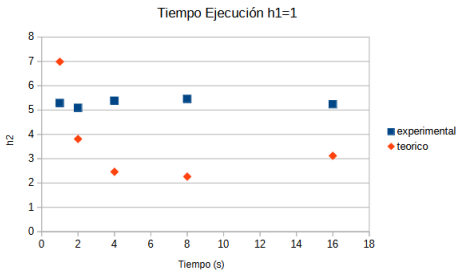


Figura 7: Tiempo de ejecucion h1=1.

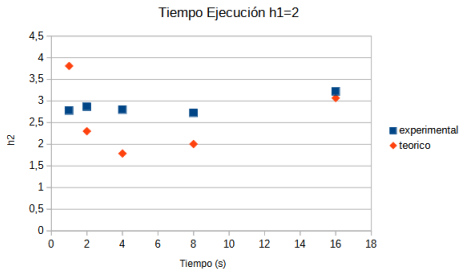


Figura 8: Tiempo de ejecucion h1=2.

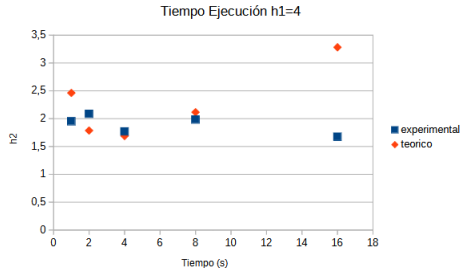


Figura 9: Tiempo de ejecucion h1=4.

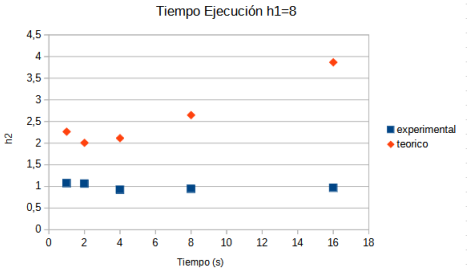


Figura 10: Tiempo de ejecucion h1=8.

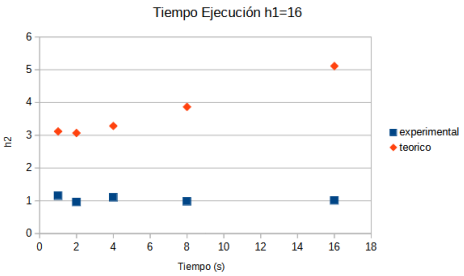


Figura 11: Tiempo de ejecucion h1=16.

6.2 Apartado b.

Enunciado: Despeja el número de hilos óptimo h1,opt y h2,opt de las ecuaciones 4 y 5 en función de NE, N, ks2, kh,1 y kh,2, donde las constantes han sido obtenidas en la instalación realizada en apartado anterior.

Para poder sacar estos dos valores tenemos que tener en cuenta estas dos fórmulas:

$$\frac{dt}{dh_1} = \frac{-k_{s2} \cdot NE \cdot NP}{h_1^2 \cdot h_2} + k_{h1} = 0$$

$$\frac{dt}{dh_2} = \frac{-k_{s2} \cdot NE \cdot NP}{h_1 \cdot h_2^2} + k_{h2} = 0$$

Para poder abordar este problema, nos hemos ayudado de un programa que hemos realizado en python usando la biblioteca `NumPy` y la función `fsolve()` de la biblioteca `SciPy`:

```python
import numpy as np
from scipy.optimize import fsolve


ks = 0.0000555588192951925
kh1 = 0.158922677013484
kh2 = 0.158922685119861
NE = 6
N = 20000


def equations(vars):
    h1, h2 = vars
    eq1 = (-ks * NE * N) / (h1 * h2**2) + kh2
    eq2 = (-ks * NE * N) / (h1**2 * h2) + kh1
    return [eq1, eq2]

initial_guess = [1, 1]


result = fsolve(equations, initial_guess)


h1_result, h2_result = result
print(f'h1 = {h1_result}, h2 = {h2_result}')
```

Poniendo los valores correspondientes nos da para h1 3.4746905622146613 y para h2 3.4746903849765474. Como podemos ver esto se aproxima a 4 hilos cada uno y si razonamos el resultado, en el apartado anterior en el excel vemos que nuestro tiempo teórico nos da el más bajo para ambos cuando h1 y h2 son 4 hilos. Con lo cual podemos concluir que esta bien ya que los hilos óptimos que saca son correspondientes con el teórico.

6.3 Apartado c.

Enunciado: Calcula el tiempo de ejecución que se esperaría para varios valores de h1 y h2 (distintos a los usados para instalar el modelo) con la configuración de parámetros utilizada en 4.1. Además, varía el tamaño de la población dando valores diferentes a NE y N y obtén el número de hilos óptimo h1,opt y h2,opt para cada configuración (no hay que lanzar ejecuciones).

Para este ejercicio hemos vuelto a utilizar el programa del ejercicio anterior pero hemos cambiado las constantes NE y N, obteniendo asi estos resultados:

1. NE = 3 y N = 10000 –>h1 = 2.188917890602637, h2 = 2.1889177789496506.
2. NE = 10 y N = 10000 –>h1 = 3.269809008256858, h2 = 3.2698088414694375.
3. NE = 3 y N = 5000 –>h1 = 1.7373452811071297, h2 = 1.7373451924884753.
4. NE = 10 y N = 5000 –>h1 = 2.595249129721018, h2 = 2.5952489973417503.

7 Cuestión 5.

7.1 Apartado a.

Enunciado: Añade una nueva hoja de cálculo al documento solver.ods, de forma que permita realizar los cálculos correspondientes a la fase de instalación en el modelo de tiempo para paso de mensajes.

En este apartado lo único que hay que hacer es crear una nueva hoja en el excel siguiendo el modelo de las anteriores páginas y poniendo las nuevas constantes que tenemos que encontrar, que son A, B, C, D y k_cmp. Ahora el tiempo no solo depende del tiempo de computación si no que también depende del tiempo de comunicación entre procesos. Es por ello que para calcular el tiempo teórico hay que usar la ecuación 9 descrita en el enunciado de la práctica:

$$t_{total} = \frac{k_{cmp}}{p} + A \cdot p^3 + B \cdot p^2 + C \cdot p + D$$

La hoja de excel quedaría así:

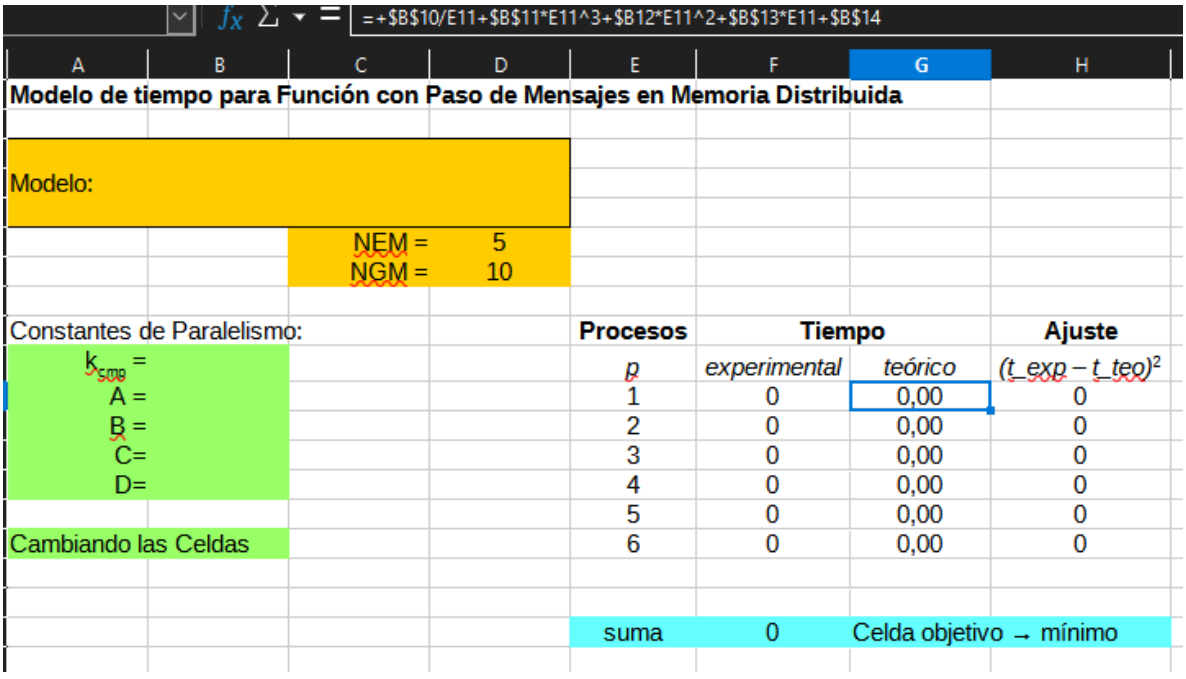


Figura 12: Memoria distribuida.

7.2 Apartado b.

Enunciado: Obtén los valores de las constantes del sistema en la ecuación 9 (kcmp, A, B, C y D) al variar el número de procesos con los datos de entrada proporcionados en el Makefile del directorio datos/ejecucion/pm. Consideraremos los siguientes parámetros de entrada: n, n_gen, tam_pob.

Al realizar todas las ejecuciones se queda así:

Modelo de tiempo para Función con Paso de Mensajes en Memoria Distribuida

Modelo:

NEM = 5
NGM = 10

Constantes de Paralelismo:

k_{cmp} = -2,22488258
A = -0,00275868
B = 7,99240116
C= 1,44776773
D= 0,37719866

Cambiando las Celdas

Procesos	Tiempo		Ajuste
p	*experimental*	*teórico*	$(t\_exp - t\_teo)^2$
1	7,59	7,59	7,4918051E-08
2	7,17	7,93	0,57652787044
3	7,23	7,30	0,0047855892
4	7,21	5,44	3,14887326789
5	7,22	6,83	0,15505797358
6	7,38	8,10	0,51425640243
suma	4,399501178	Celda objetivo → mínimo	

Figura 13: Datos experimentales y teóricos en memoria distribuida.

Como podemos ver en la imagen, nuestras constantes se quedarían:

1. kcomp = 0,00000000003355534485634277.
2. A = 0,000889448236324653.
3. B = 5,97739927094682.
4. C = 1,26312196637744.
5. D = 0,34858925199555.

No hay ningún problema en obtener variables negativas ya que en este caso no tienen significado físico.

7.3 Apartado c.

Enunciado: Calcula el tiempo de ejecución teórico para diferentes valores de p y resuelve la ecuación 10 para encontrar popt en función de las constantes del sistema (puedes usar algún método numérico para resolverla).

$$total = \frac{k_{cmp}}{p} + A \cdot p^3 + B \cdot p^2 + C \cdot p = 0$$

$$\frac{0,00000000003355534485634277}{p} + 0,000889448236324653 \cdot p^3 + 5,97739927094682 \cdot p^2$$

$$+1,26312196637744 \cdot p = 0$$

Resolviendo la ecuación de segundo grado obtenemos que el número de procesos óptimo en función de las constantes del sistema, y cogiendo el valor positivo, nos da p = 5,15389237*10⁻6.

8 Cuestión 6.

Enunciado: Realizar el informe final incluyendo las conclusiones generales y valoración personal sobre el trabajo realizado.

En este trabajo hemos repasado OpenMP y MPI de las dos prácticas anteriores recogiendo lo mejor de ambos además de enseñarnos a combinar estas dos estrategias.
Sentimos que la mayor parte de la práctica hemos estado calculando ecuaciones y fórmulas lo cual es muy engorroso y esa parte no la hemos sentido del todo satisfactoria, pero por otro lado a la hora de combinar las dos estrategias se nos hizo muy sencillo y ameno puesto que pensamos en un principio que iba a ser más difícil. La cuestión que más tiempo nos ha llevado ha sido la cuestión 4 por el tema de hacer el excel con todos los gráficos bidimensional, porque en un principio los íbamos a hacer tridimensionales pero al no controlar del todo la herramienta de LibreOffice no conseguimos el resultado esperado con los gráficos tridimensionales.
Los tres miembros del grupo hemos trabajado en todas las cuestiones de la práctica realizando cada uno una tarea, ejecución de código, diseño de los excels, instalación, documentación, etc. Nos ha permitido aprender mucho sobre OpenMP y MPI.