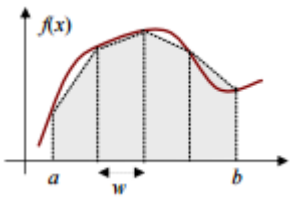
Practica 4 **10/23/2017**

Analice y realice los siguientes ejercicios. En su reporte debe aparecer el análisis de los programas línea por línea.

Ejercicio 1 integral_s.c / integral_p.c

El programa calcula la integral de una función, en este caso f(x) = 1.0 / (sen x * cos x + 2.0), mediante la suma del área de un cierto número de trapecios que cubren la curva de la función entre dos puntos. El programa solicita los dos extremos de la función y el número de trapecios a sumar. integral_s.c es el programa serie e integral_p.c la versión paralela MPI (broadcast para el reparto de datos y reduce para la recogida de resultados). El objetivo del ejercicio es medir el tiempo de ejecución de ambos programas, usando los siguientes parámetros: - límites de la integral: 0.0 y 100.0



- número de trapecios: 256 (2^8), 512, 1.024... hasta 134.217.728 (2^{27})

- número de procesadores (caso paralelo): 2, 4, 8, 16 y 32.

– Añade al programa un bucle que efectúe las integrales en función del número de trapecios y otro que efectúe las repeticiones de las que obtener el valor mínimo (elimina la petición de los límites de la integral —añádelos como constantes— y del número de trapecios). Representa gráficamente los tiempos de ejecución serie y paralelo en función del número de trapecios sumados, así como el factor de aceleración (speed-up) en relación al caso serie y la eficiencia obtenida. Utiliza las escalas más adecuadas para los gráficos, de manera que se observe correctamente el comportamiento del programa. Interpreta y justifica los resultados obtenidos.

Ejercicio 2 anillo.c

N procesos forman un anillo en el que la comunicación es i → (i+1) mod num_proc. Queremos medir cómo se comporta la red de comunicación en función del tamaño de los mensajes que se envían, para lo que vamos a ejecutar un programa que haga circular un vector de diferentes tamaños entre los procesadores del anillo, dando una vuelta al mismo. Al comienzo, se pide la longitud del vector que se va a transmitir. El objetivo del ejercicio es completar el programa y obtener el tiempo de ejecución del mismo en un anillo de 16 procesadores, siendo el tamaño del vector a transmitir de 2^0 , 2^1 , 2^2 , ..., 2^{22} enteros.

Se recomienda hacer un bucle para todos los experimentos, empezando con un mensaje de 2^0 elementos e incrementando la longitud del vector que se envía multiplicándola por 2. Igual que en el caso anterior, añade un bucle de repeticiones del experimento para obtener valores mínimos, medios, etc.

A partir de los resultados que obtengas, calcula el tiempo de transmisión i → i+1 y el ancho de banda conseguido en la transmisión (MB/s), en función de la longitud del vector transmitido. Dibuja ambas curvas de manera adecuada e interpreta los resultados.

Ejercicio 3 pladin.c

Se quiere repartir la ejecución de una determinada cola de tareas de manera dinámica entre n procesos. Uno de los procesos (manager) hace la veces de gestor de la cola de tareas; su trabajo consiste en ir distribuyendo las tareas a petición de los otros procesos. El resto de los procesos (workers) ejecuta la tarea encomendada, y, al finalizar la misma, solicita una nueva tarea, hasta terminar de esta manera con todas las tareas. Se trata de una estructura estándar de gestión dinámica de tareas y, como ejemplo de la misma, vamos a considerar el caso de procesamiento de una determinada matriz. El programa pladin_ser.c recoge la versión serie del trabajo que hay que realizar en paralelo. Escribe el programa pladin.c, que procesa en paralelo la matriz M repartiendo las filas de la matriz de manera dinámica, tipo self scheduling, es decir, fila a fila. Recuerda: uno de los procesos, P0, es el encargado de ir repartiendo los datos (filas) bajo demanda, y el resto, n–1 procesos, procesan las filas, devuelven resultados y solicitan una nueva tarea. Al final, la matriz M debe quedar actualizada en P0. Confirma los resultados de la versión paralela con los que obtienes en la versión serie. Finalmente, obtén los tiempos de ejecución y el speed-up para los casos de 1+1, 1+2, 1+4, 1+8, 1+16 y 1+31 procesos. Si quieres, prueba con otro tipo de scheduling y compara los resultados que obtengas.

Ejercicio 1: integral_s.c

```
/*****************************************************************************
     integral_s.c
     Integral de una funcion mediante sumas de areas de trapecios
 *****************************************************************************/

#include <stdio.h>
#include <sys/time.h>
#include <math.h>


/*************  FUNCION f: funcion a integrar  ***********************/
double f(double x)
{
  double y;

  y = 1.0 / (sin(x) + 2.0) + 1.0 / (sin(x)*cos(x) + 2.0);

  return y;
}


/*************  MAIN  ************************************************/
int main()
{
  struct timeval  t0, t1;
  double          tej;

  double      a, b, w;          // Limites de la integral
  long long   n, i;             // Numero de trapecios a sumar
  double      resul, x;
  float  aux_a, aux_b, aux_n;
```

```
  printf("\n  Introduce a, b (limites) y n (num. de trap.) \n");
  scanf("%f %f %f", &aux_a, &aux_b, &aux_n);

  a = (double) aux_a;
  b = (double) aux_b;
  n = (long long) aux_n;

  gettimeofday(&t0,0);

// CALCULO DE LA INTEGRAL

  w = (b-a) / n;                          // anchura de los trapecios

  resul = (f(a) + f(b)) / 2.0;
  x = a;

  for (i=1; i<n; i++)
  {
    x = x + w;
    resul = resul + f(x);
  }
  resul = resul * w;


  gettimeofday(&t1,0);
  tej = (t1.tv_sec - t0.tv_sec) + (t1.tv_usec - t0.tv_usec)/1e6;

   printf("\n Integral de f (a=%1.2f, b=%1.2f, n=%1.0f) = %1.12f\n",a,b,(double)n,resul);
  printf("  T. ejec. (serie) = %1.3f ms \n\n", tej*1000);

  return 0;
} /* main */
```

Ejercicio 1: integral_p.c

```
/*****************************************************************************
        MPI: integral_p.c
        Integral de una funcion mediante sumas de areas de trapecios
        Broadcast para el envio de datos y Reduce para la recepcion
*****************************************************************************/

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

double  t0, t1;


/*************  FUNCION f: funcion a integrar  ***************/
double f(double x)
{
  double y;

  y = 1.0 / (sin(x) + 2.0) + 1.0 / (sin(x)*cos(x) + 2.0);

  return y;
}


/*************  FUNCION Leer_datos  *************************/
void Leer_datos(double* a_ptr, double* b_ptr, long long* n_ptr, int pid)
{
  float aux_a, aux_b, aux_n;
  int   root;
```

```
  if (pid == 0)
  {
    printf("\n  Introduce a, b (limites) y n (num. de trap.)  \n");
    scanf("%f %f %f", &aux_a, &aux_b, &aux_n);

    t0 = MPI_Wtime();
  }

// castings para convertir a y b a double y n a long long (evitar overflow)

  (*a_ptr) = (double)aux_a;
  (*b_ptr) = (double)aux_b;
  (*n_ptr) = (long long)aux_n;

  root = 0;
  MPI_Bcast(a_ptr, 1, MPI_DOUBLE, root, MPI_COMM_WORLD);
  MPI_Bcast(b_ptr, 1, MPI_DOUBLE, root, MPI_COMM_WORLD);
  MPI_Bcast(n_ptr, 1, MPI_LONG_LONG, root, MPI_COMM_WORLD);

} /* Leer_datos */


/*************  FUNCION Integrar: calculo local   *********************/
double Integrar(double a_loc, double b_loc, long long n_loc, double w)
{
  double resul_loc, x;
  long long    i;               //  n_loc es un long de 8 bytes

  resul_loc = (f(a_loc) + f(b_loc)) / 2.0;
  x = a_loc;

  for (i=1; i<n_loc; i++)
  {
    x = x + w;
    resul_loc = resul_loc + f(x);
  }
  resul_loc = resul_loc * w;


   return (resul_loc);
 } /*  Integrar  */
```

```
/*************  MAIN    ***********************************************/
int main(int argc, char** argv)
{
  int          pid, npr, root;            // Identificador y numero de proc, y nodo
raiz.
  double      a, b, w, a_loc, b_loc;
  long long   n, n_loc;
  double      resul, resul_loc;         // Resultado de la integral


  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &pid);
  MPI_Comm_size(MPI_COMM_WORLD, &npr);


// Lectura y distribucion de datos a todos los procesadores

  Leer_datos(&a, &b, &n, pid);
  w = (b-a) / n;

  // por si n no es divisible entre npr; ojo overflow
  n_loc = (pid+1)*n/npr - pid*n/npr;
  a_loc = a + (pid)*n/npr * w;
  b_loc = a + (pid+1)*n/npr * w;

// Calculo de la integral local

  resul_loc = Integrar(a_loc, b_loc, n_loc, w);


// Suma de resultados parciales

  resul = 0.0; root = 0;
  MPI_Reduce(&resul_loc, &resul, 1, MPI_DOUBLE, MPI_SUM, root, MPI_COMM_WORLD);


// Impresion de resultados

  if (pid == 0)
  {
    t1 = MPI_Wtime();

     printf("\n  Integral de f (a=%1.2f, b=%1.2f, n=%f) = %1.12f", a,b,(double)n,resul);
    printf("\n  T. ejec. (%d proc.) = %1.3f ms \n\n",npr,(t1-t0)*1000);
  }

  MPI_Finalize();
  return 0;
} /*  main  */
```

Ejercicio 2: anillo.c

```
/*****************************************************************************
   MPI: anillo.c
   Se envia un vector de proceso a proceso en un anillo de npr procesos
   Calcula el tiempo de transmision y el ancho de banda.
   Pide la longitud maxima del vector

    -- por completar

   Repite el experimento variando la longitud del vector a enviar
   desde 1 hasta 2**lgmax
   En cada caso, repite el proceso NREP veces y obten el tiempo minimo
*****************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <mpi.h>

#define NREP 10                 // numero de repeticiones del experimento //

int main(int argc, char *argv[])
{
  int     pid, npr;
  int     siguiente, anterior;
  int     *vector, longi, lgmax, tam;
  int     i;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &pid);
  MPI_Comm_size(MPI_COMM_WORLD, &npr);
```

```
// direcciones de los vecinos en el anillo //
  siguiente = (pid + 1) % npr;
  anterior = (pid + npr - 1) % npr;

// lectura de parametros //
  if (pid == 0)
  {
    printf("\nLongitud maxima del vector a enviar (potencia de 2, maximo 22):   ");
    scanf("%d", &lgmax);
  }

// DISTRIBUCION DE DATOS A TODOS LOS PROCESOS //

/* Reserva de memoria, inicializacion del vector */
  tam = pow(2,lgmax);
  vector = (int *)malloc(sizeof(int)*tam);
  if (pid == 0) for(i=0; i<tam; i++) vector[i] = i % 100;

// COMIENZO DE LA TRANSMISION  //
// desde longi = 1 hasta el valor maximo solicitado (en potencias de 2)

// repetir el proceso NREP veces (para quedarse con el menor tiempo de transmision)

// transmitir vector de tamaino longi del 0 al 1, del 1 al 2, etc., dando 1 vuelta al anillo

// Finalmente, obtener como resultado el tiempo de una vuelta en el anillo,
// el tiempo de transmision i --> i+1
// y el ancho de banda de la transmision i-->i+1 = num_bits / tiempo   (en Mb/s)

  MPI_Finalize();
  free(vector)
  return 0;
}
```

Ejercicio 3: pladin_ser.c

```
/******************************************************************************
    pladin_ser.c
    version serie de un sistema de gestion de una cola de tareas
    (filas de una matriz) mediante planificacion dinamica
******************************************************************************/

#include <stdio.h>
#include <sys/time.h>

#define NF 3000
#define NC 500

int main(int argc, char **argv)
{
  struct timeval t0,t1;
  double tej;

  int   i, j, k, x;
  int   M[NF][NC], sum;


// inicializacion de la matriz

  for(i=0; i<NF; i++)
  for(j=0; j<NC; j++)
     M[i][j] = rand() % 1000 - 200;


  gettimeofday(&t0,0);
```

```
// Procesamiento de la matriz. CODIGO A PARALELIZAR
  for (i=0; i<NF; i++)
  for (j=0; j<NC; j++)
  {
    if (M[i][j] > 0)
    {
      x = 1;
      for (k=2; k <= M[i][j]; k++) x = (x * k) % (M[i][j]+1);
      M[i][j] = x % (M[i][j]+1);
    }
  }

  sum = 0;
  for (i=0; i<NF; i++)
  for (j=0; j<NC; j++)
    sum = (sum + M[i][j]) % 1000;

  gettimeofday(&t1,0);

// Imprimir resultados

  printf("\n Suma (mod 1000) de la matriz transformada = %d\n", sum);

  tej = (t1.tv_sec - t0.tv_sec) + (t1.tv_usec - t0.tv_usec)/1e6;
  printf("\n T. ejec. (serie) = %1.3f s\n\n", tej);

  return 0;
}
```