Ejercicios Capítulos 3 y 4

Lorena Xiomara Castillo Galdos

Abril 2017

1 Capítulo 3

- **Modify the trapezoidal rule so that it will correctly estimate the integral even if comm_sz doesn't evenly divide n. (You can still assume that n $\geq$ comm sz .)**

  Lo que haríamos en este caso es agregarle al último proceso el módulo del número de iteraciones entre el número de procesos, como se muestra en el siguiente código:

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

float trapecio(float local_a, float local_b, int local_n, float h)
{
    float integral;
    float x;
    int i;
    integral = ( ( f(local_a) + f(local_b) ) *1.0 )/2.0;
    x = local_a;

    for( i = 1; i <= (local_n -1); i++ )
    {
        x+=h;
        integral+=f(x);
    }
    integral*=h;
    return integral;
}

float f(float x)
{
  float return_val=x*x;
  return return_val;
}

void Get_data(int my_rank,int p,float *a_ptr,float *b_ptr,int *n_ptr)
{
      int source=0 , dest , tag;
      MPI_Status status;
```

```
      if(my_rank==0)
      {
          printf("Enter a, b, and n\n");
          scanf("%f %f %d",a_ptr,b_ptr,n_ptr);
          for(dest=1;dest<p;dest++) {
            tag=30;
            MPI_Send(a_ptr,1,MPI_FLOAT,dest,tag,MPI_COMM_WORLD);
            tag=31;
            MPI_Send(b_ptr,1,MPI_FLOAT,dest,tag,MPI_COMM_WORLD);
            tag=32;
            MPI_Send(n_ptr,1,MPI_INT,dest,tag,MPI_COMM_WORLD);
          }
      }
       else {
          tag=30;
          MPI_Recv(a_ptr,1,MPI_FLOAT,source,tag,MPI_COMM_WORLD,&status);
          tag=31;
          MPI_Recv(b_ptr,1,MPI_FLOAT,source,tag,MPI_COMM_WORLD,&status);
          tag=32;
          MPI_Recv(n_ptr,1,MPI_INT,source,tag,MPI_COMM_WORLD,&status);
        }
}


int main(int argc,char **argv)
{
   int  my_rank, p, n, local_n, source, dest=0, tag=50;
   float a, b, h, local_a, local_b, integral, total;
   //
   MPI_Status status;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
   MPI_Comm_size(MPI_COMM_WORLD,&p);
   //
   Get_data(my_rank,p,&a,&b,&n);
   h=(b-a)/n ; local_n=n/p;
   if(my_rank == (p-1)) local_n = n/p+(n%p);
   else{ local_n = n/p; }
   local_a=a+my_rank*local_n*h ;
   local_b=local_a+local_n*h;
   integral=trapecio(local_a,local_b,local_n,h);
   if(my_rank==0) {
     total=integral;
     for(source=1;source<p;source++) {
       MPI_Recv(&integral,1,MPI_FLOAT,source,tag,MPI_COMM_WORLD,&status);
       total+=integral;
       printf("proceso %d calculo un total de %f\n",source,total);
     }
      printf("con n= %d trapezoides\n la estimacion",n);
      printf("de la integral entre %f y %f\n es= %f \n",a,b,total);
    }
    else {
      MPI_Send(&integral,1,MPI_FLOAT,dest,tag,MPI_COMM_WORLD);
    }
    MPI_Finalize();
}
```

- **Determine which of the variables in the trapezoidal rule program are local and which are global**

  - Variables locales: número de trapecios, local_a y local_b, suma parcial, suma total, id. - Variables globales: a y b, tamaño de trapecios, número de procesos.

- **In a binary tree, there is a unique shortest path from each node to the root. The length of this path is often called the depth of the node. A binary tree in which every non leaf has two children is called a full binary tree, and a full binary tree in which every leaf has the same depth is sometimes called a complete binary tree. Use the principle of mathematical induction to prove that if T is a complete binary tree with n leaves, then the depth of the leavesis log 2 (n).**

  Suponiendo que tenemos un árbol completo con el que trabajar, podemos decir que a la profundidad k, hay 2^k nodos. Se puede probar esto mediante la inducción simple, basada en la intuición de que la adición de un nivel adicional al árbol aumentará el número de nodos en todo el árbol por el número de nodos que estaban en el nivel anterior dos veces.

  La altura k del árbol es log (N), donde N es el número de nodos, porque log de la base 2 de N es exactamente igual que escribir $N = 2^k$.

- **Suppose comm sz = 4 and suppose that x is a vector with n = 14 components. You should try to make your distributions general so that they could be used regardless of what comm_sz and n are. You should also try to make your distributions "fair" so that if q and r are any two processes, the difference between the number of components assigned to q and the number of components assigned to r is as small as possible.**

  - How would the components of x be distributed among the processes in a program that used a block distribution?

  Al utlizar distribución por bloques, los tres primeros procesos obtendrían los tres primeros valores del vector y el último proceso obtendría tres procesos más los faltantes que serían dos.

  - How would the components of x be distributed among the processes in a program that used a cyclic distribution?

  Al distribuír entre los procesos cíclicamente cada proceso iría obteniendo un valor del vector hasta que no queden valores, por lo que los dos primeros

procesos en este caso tendrían cuatro valores y los dos últimos tres valores.

- How would the components of x be distributed among the processes in a program that used a block-cyclic distribution with blocksize b = 2?

Al distribuír entre los procesos cíclicamente por bloque de dos, cada proceso iría obteniendo dos valores del vector hasta que no queden valores, por lo que los tres primeros procesos en este caso tendrían cuatro valores y el último dos valores.

- **What do the various MPI collective functions do if the communicator contains a single process?**

  Durante un broadcast, un proceso envía los mismos datos a todos los procesos en un comunicador. Uno de los principales usos de el bradcast es enviar entrada del usuario a un programa paralelo o enviar parámetros de configuración a todos los procesos, y al existir solamente un proceso en el comunicador, el programa sería ejecutado como uno serial y se realizaría broadcasting con ese mismo proceso.

- **In the Read vector function shown in Program 3.9, we use local_n as the actual argument for two of the formal arguments to MPI Scatter: send count and recv count. Why is it OK to alias these arguments?**

  El alias a utilizar puede ser el mismo en ambas funciones de recibir y enviar, dado que las funciones tratan este valor de manera diferente, ya que no es lo mismo enviar y recibir es que podemos utilizar local_n como alias para ambas.

2 Capítulo 4

- **When we discussed matrix-vector multiplication we assumed that both m and n, the number of rows and the number of columns, respectively, were evenly divisible by t, the number of threads. How do the formulas for the assignments change if this is not the case?**

  Al momento de realizar la asignación a los procesos, al último proceso se le debería añadir el módulo de la división.

- **The performance of the pi calculation program that uses mutexes remains roughly constant once we increase the number of threads beyond the number of available CPUs. What does this suggest about how the threads are scheduled on the available processors?**

  Si realmente no hay nada que distinguir entre la hebra B de la hebra C, entonces la respuesta en la mayoría de las implementaciones del mutex probablemente será podría ser B o C, y no se puede predecir cuál de irá primero, o el que intentó adquirir primero el mutex, es decir, el primero en llegar, primero en seguir.

  Sin embargo, puede haber algo que los distingue de alguna manera, si la implementación soporta la programación de prioridad, uno de los subprocesos puede tener una prioridad más alta que la otra, en cuyo caso la prioridad más alta debería ser programada primero.

- **If a program uses more than one mutex, and the mutexes can be acquired in different orders, the program can deadlock. That is, threads may block forever waiting to acquire one of the mutexes. As an example, suppose that a program has two shared data structures for example, two arrays or two linked lists each of which has an associated mutex. Further suppose that each data structure can be accessed (read or modified) after acquiring the data structure's associated mutex.**

  - Suppose the program is run with two threads. Further suppose that the following sequence of events occurs. What happens?

  Supongamos que la hebra 1 intenta adquirir el bloqueo del mutex B y la hebra 2 intenta adquirir el bloqueo del mutex A. La hebra 1 no puede continuar y se bloquea esperando el bloqueo del mutex B. La hebra 2 no puede continuar y se bloquea esperando el bloqueo del mutex A. Esto no se puede cambiar, por lo que este es un bloqueo permanente de los hilos, y un deadlock.

  - Would this be a problem if the program used busy-waiting (with two flag variables) instead of mutexes?

  Al utilizar busy waiting ocurriría lo mismo ya que estarían dentro de un bucle infinito hasta que la condición cambie, lo cual no sucederá.

- Would this be a problem if the program used semaphores instead of mutexes?

Al utilizar semáforos, el programa no caería en deadlock.

- **The linked list operations Insert and Delete consist of two distinct "phases". In the first phase, both operations search the list for either the position of the new node or the position of the node to be deleted. If the outcome of the first phase so indicates, in the second phase a new node is inserted or an existing node is deleted. In fact, it's quite common for linked list programs to split each of these operations into two function calls. For both operations, the first phase involves only read-access to the list; only the second phase modifies the list. Would it be safe to lock the list using a read-lock for the first phase? And then to lock the list using a write-lock for the second phase? Explain your answer.**

  Se podrían realizar el bloqueo de lectura para que solo una hebra tenga acceso a la función de lectura a la vez y el bloqueo de escritura para preveir que otra hebra intente escribir al mismo tiempo.