Universidad Nacional de San Agustin
Esc. Profesional de Ciencia de la Computación

Algoritmos Paralelos

Tema: Laboratorio 4

Profesor: Alvaro Henry Mamani Aliaga
Alumno: Diego André Ranilla Gallegos

10 de Marzo del 2017

1. Regla Trapezoidal

Se emplea la regla trapezoidal para aproximar el área entre la gráfica de una función $y = f(x)$, dos lineas verticales y el eje x.

La regla trapezoidal consiste en dividir el intervalo ubicado en el eje x en n sub-intervalos de igual longitud. Entonces debemos hallar el área aproximada entre el gráfico y cada sub-intervalo con la formula del trapecio como vemos en la siguiente imagen:

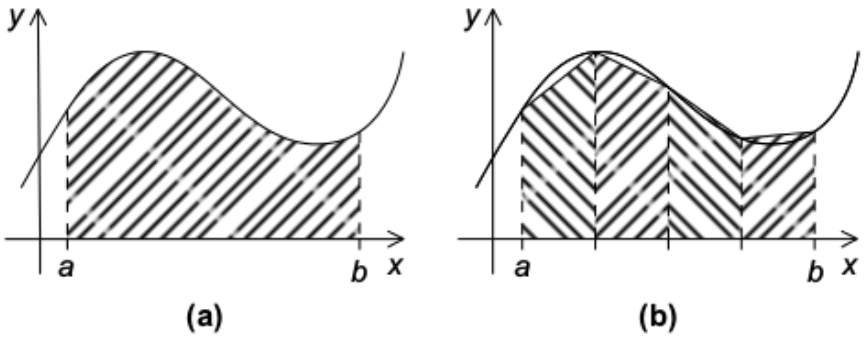

Figura 1: (a) Se estimará el área bajo la curva y (b) Se dividirán en sub-intervalos de igual longitud

Entonces para calcular el área de cada uno de los sub-intervalos vamos a tomar en cuenta estas cosas:

- La base será el sub-intervalo, si los extremos del intervalo es x_i y x_{i+1} la longitud del sub-intervalo que será llamado h será $h = x_{i+1} - x_i$.
- los lados verticales son aquellas que atraviesan los extremos del intervalo.
- El cuarto lado es la linea secante que une los puntos donde las lineas verticales o lados cortan el gráfico.
- Las longitudes de ambos lados verticales son $f(xi)$ y $f(x_{i+1})$.

Por lo tanto la formula general para hallar el área aproimada bajo la curva es:

$$\frac{h}{2}[f(x_i) + f(x_{i+1})]$$

Figura 2: Formula General bajo la curva

Finalmente dado que h tiene el mismo valor en cada intervalo desde el extremo inicial a y final b como vemos en la figura 1, h sería: $h = (b-a)/n$ donde n es el número de intervalos y los puntos a partir de x_0 a x_n se pueden interpretar como $x_0 = a, x_1 = a + h, x_2 = a + 2h, ..., x_{n-1} = a + (n-1)h. x_n = b$.

Podremos definir de otra manera la suma de áreas:

$$Área = h[f(x_0)/2 + f(x_1) + f(x_2) + ... + f(x_{n-1}) + f(x_n)/2]$$

Para implementar el cálculo aproximado del área podremos crear un algoritmo en serie como este:

```
#include <stdio.h>
...
int main(){
    int h = (b-a)/n;
    int aprox = (f(a) + f(b))/2.0;

    for(int i=1; i<=n-1; i++){
        int x_i = a + i*h;
        aprox += f(x_i);
    }
    aprox = h*aprox;

}
```

Listing 1: Programa serial de la regla trapezoidal

Para poder paralelizar la regla trapezoidal podemos dividir cada uno de los cálculos en procesos, pero si nos damos cuenta que si dividimos mucho más el intervalo entre a y b podemos obtener mucho más trapezoides y el calculo del área total sería mucho más preciso, por lo que sería más útil dividir el total de los cálculos de cada trapezoide n en cada uno de los procesos com_sz (tamaño del comunicador) tal que cada proceso calcule casi por igual el área de los trapezoides, para así enviar el resultado al proceso principal para calcular el resultado final, el cual lo veremos en este código:

```
int main(void) {
    int my_rank, comm_sz, n = 1024, local_n;
    double a = 0.0, b = 3.0, h, local_a, local_b;
    double local_int, total_int;
    int source;

    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

    h = (b      a)/n;
    local_n = n/comm_sz;

    local_a = a +  m y _ r a n k local_nh ;
    local_b = local_a + l o c a l _ n h ;
    local_int = Trap(local_a, local_b, local_n, h);

    if(my_rank != 0) {
        MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    }
    else {
        total_int = local_int;
        for (source = 1; source < comm_sz; source++) {
            MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0,
```

```
            MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            total_int += local_int;
        }
    }
    if (my_rank == 0) {
    printf("With n = %d trapezoids, our estimate\n", n);
    printf("of the integral from %f to %f = %.15e\n",
    a, b, total_int);
    }
    MPI_Finalize();
    return 0;
    }
}
```

Listing 2: Programa paralelo de la regla trapezoidal

* *Trap*() es una función en el cual recive como argumento el extremo izquierdo, extremo derecho, el número de trapecios y la longitud del intervalo para cada uno de los procesos que este programa.

2. Ejemplo con MPI_Scatter y MPI_Gather

```
#include <stdio.h>
#include <mpi.h>
#define N 3

int main(){
  int comm_sz;
  int my_rank;

  MPI_Init(NULL,NULL);
  MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

  if(my_rank==0){
    int A[] = {1,2,3,4,5,6};
    int B[2];
    MPI_Scatter(A,2,MPI_INT,B,2,MPI_INT,0,MPI_COMM_WORLD);

    int i;
    for(i=0;i<2;i++){
      printf("valor %d del proceso %d\n", B[i],my_rank);
      B[i]+=2;
    }


    int C[6];
    MPI_Gather(B,2,MPI_INT,C,2,MPI_INT,0,MPI_COMM_WORLD);

    for(i=0;i<6;i++){
      printf("valor %d\n", C[i]);
    }

  }

  else{
    int B[2];

    MPI_Scatter(0,0,MPI_INT,B,2,MPI_INT,0,MPI_COMM_WORLD);

```

```
39      int i;
40      for(i=0;i<2;i++){
41        printf("valor %d del proceso %d\n", B[i],my_rank);
42        B[i]+=2;
43      }
44      MPI_Gather(B,2,MPI_INT,0,0,MPI_INT,0,MPI_COMM_WORLD);
45    }
46
47    MPI_Finalize();
48  }
```

Listing 3: Uso de las funciones MPI_scatter() y MPI_Gather()

En este programa consiste en que el proceso *root* (proceso 0) reparte dos elementos a cada proceso (incluyendo asimismo), estos procesos lo reciben, suman los valores recibidos y los devuelven al proceso *root*, para que se almacenen en un vector.

3. Multiplicación de matriz-vector usando MPI_Allgather

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <mpi.h>
4
5
6  void mostrar_matriz(double local_y[],int m){
7
8    int i;
9    for (i=0;i<m;i++){
10     printf("%f ", local_y[i]);
11   }
12   printf("\n");
13 }
14
15 // la cantidad de valores de x de longitud n  deben ser acorde al
       numero de procesos osea n procesos
16
17 void multiplicacion(double local_A[],double local_x[],double
       local_y[], int local_m,int n,int local_n,MPI_Comm comm){
18
19   double *x;
20   int local_i,j;
21   int local_ok = 1;
22
23   x = malloc(n*sizeof(double));
24   MPI_Allgather(local_x,local_n,MPI_DOUBLE,x,local_n,MPI_DOUBLE,
       comm);//send/receive
25
26   for (local_i = 0; local_i < local_m; local_i++)
27   {
28     local_y[local_i] = 0.0;
29
30     for (j = 0; j < n; j++)
31     {
32       local_y[local_i] += local_A[local_i*n+j]*x[j];
33     }
34   }
35   //free(x);
36 }
37
38 int main(){
39   int comm_sz;
```

```
40   int my_rank;
41
42   MPI_Init(NULL,NULL);
43   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
44   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
45
46   double local_A
       [12]={1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0,12.0};
47   //double local_x[3]={2.0,5.0,3.0};
48   double local_x[1]={my_rank};
49   double local_y[4];
50
51   multiplicacion(local_A,local_x,local_y,4,3,1,MPI_COMM_WORLD);
52
53   mostrar_matriz(local_y,4);
54
55 }
```

Listing 4: Uso de la función MPI_AllGather()

4. Tomar el tiempo para obtener cuadros análogos a las tablas de libro: 3.5, 3.6 y 3.7

$comm_sz$	1024
1	
2	
4	
8	
16	

Tabla 1: Tabla muy sencilla.