TUGAS 4

Nama : Abdul Rahman
NIM : 23/528808/SPA/00987
Prodi : Doktor Ilmu Komputer
MK : Komputabilitas dan Kompleksitas (MII227001)
Dosen : Bapak Faizal Makhrus, S.Kom., M.Sc., Ph.D.

Berikut adalah penjelasan kode C yang menggunakan **MPI (*Message Passing Interface*)** untuk menghitung integral numerik dari fungsi f(x)=cos(x) dalam interval tertentu dengan metode integrasi *Riemann* (*midpoint rule*):

Code	Penjelasan

Header
 	
`#include <mpi.h>`
`#include <math.h>`
`#include <stdio.h>`
`#include <time.h>`
	MPI.h: Digunakan untuk fungsi-fungsi MPI.
	math.h: Untuk fungsi matematika seperti acos (menghitung nilai π) dan cos (fungsi kosinus).
	stdio.h: Untuk input/output seperti printf.
	time.h: Untuk pengukuran waktu eksekusi.

Fungsi fct
 	
`double fct(double x) {`
`  return cos(x);`
`}`
	Fungsi ini mendefinisikan f(x)=cos(x) yaitu fungsi yang ingin diintegrasikan.

Fungsi Integral
 	
`double integral(double a, int n, double h) {`
`  int j;`
`  double h2, aij, integ;`
`  integ = 0.0;`
`  h2 = h / 2.;`
`  for (j = 0; j < n; j++) {`
`  aij = a + j * h;`
`  integ += fct(aij + h2) * h;`
`  }`
`  return (integ);`
`}`
	Parameter:
	• a: Batas bawah dari bagian integral.
	• n: Jumlah interval kecil (sub-interval) yang dihitung oleh satu proses MPI.
	• h: Panjang masing-masing interval kecil.
	Proses:
	• Menggunakan **Midpoint Rule**, yaitu mengambil titik tengah setiap interval hhh untuk menghitung nilai integral.
	• Integral dihitung sebagai integ $= \sum_{j=0}^{n-1} f(a + jh + h/2) \cdot h.$

Program Utama
 	
Inisialisasi MPI
 	
`MPI_Init(&argc, &argv);`
`MPI_Comm_rank(MPI_COMM_WORLD, &myid);`
`MPI_Comm_size(MPI_COMM_WORLD, &p);`
 	
Pengaturan Parameter
`pi= acos(-1.0);`
`a = 0.;`
`b = pi * 1. / 2.;`
`n = 100000000;`
	MPI_Init: Menginisialisasi lingkungan MPI.
	MPI_Comm_rank: Mengembalikan ID proses (0 hingga p−1p-1p−1).
	MPI_Comm_size: Mengembalikan jumlah total proses MPI yang berjalan.
	
	• **π** dihitung menggunakan fungsi acos(-1).
	• Batas integral:
	  ○ Bawah: a=0
	  ○ Atas: b=π2b
	• Jumlah interval: n=100,000,000.

Pembagian Kerja
 	
`h = (b - a) / n;`
`num = n / p;`
`my_range = (b - a) / p;`
`my_a = a + myid * my_range;`
 	
Perhitungan Parsial
 	
`start = clock();`
`my_result = integral(my_a, num, h);`
	h: Panjang setiap interval kecil.
	num: Jumlah interval kecil yang dihitung oleh setiap proses.
	my_range: Rentang yang dialokasikan untuk setiap proses.
	my_a: Batas bawah untuk proses saat ini.
	
	Setiap proses menghitung bagian integralnya masing-masing dengan fungsi integral.

Pengumpulan Hasil
 	
Proses 0 (Root)
 	
`if (myid == 0) {`
`  result = my_result;`
`  for (i = 1; i < p; i++) {`
`  MPI_Recv(&my_result, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD, &status);`
`  result += my_result;`
`  }`
`  printf("The result = %f\n", result);`
`}`
 	
Proses Lain
 	
`else {`
`  MPI_Send(&my_result, 1, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);`
`}`
	Proses dengan myid = 0 (root):
	• Menerima hasil parsial dari proses lain menggunakan MPI_Recv.
	• Menjumlahkan semua hasil parsial untuk mendapatkan hasil akhir.
	
	• Proses lain mengirim hasil parsial mereka ke proses root menggunakan MPI_Send.

Pengukuran Waktu
 	
`end = clock();`
`printf("durasi = %f \n", (double)(end - start) / CLOCKS_PER_SEC);`
	Mengukur waktu eksekusi dengan menggunakan fungsi clock.

Penyelesaian MPI `MPI_Finalize();`	Mengakhiri semua komunikasi MPI dan membersihkan sumber daya yang digunakan.

Alur Kerja Program

1. Program dibagi menjadi beberapa proses menggunakan MPI.
2. Rentang integral dibagi ke masing-masing proses.
3. Setiap proses menghitung integral parsialnya secara paralel.
4. Proses root (0) mengumpulkan semua hasil parsial dan menghitung hasil akhir.
5. Durasi eksekusi dicetak untuk analisis performa.

Manfaat dan Tujuan

1. **Tujuan**: Menghitung integral secara efisien dengan memanfaatkan komputasi paralel.
2. **Manfaat**:
    - Meningkatkan kecepatan penghitungan dengan membagi beban kerja ke beberapa proses.
    - Menunjukkan penggunaan fungsi MPI untuk komunikasi antar-proses.

Penjelasan Multiproses

```
/* C Example */
#include <mpi.h>
#include <math.h>
#include <stdio.h>
#include <time.h>

double fct(double x)
{
      return cos(x);
}

/* Prototype */
double integral(double a, int n, double h);

int main(argc, argv)
int argc;
char *argv[];
{
      /************************************************************************
       *                                                                      *
       * This is one of the MPI versions on the integration example           *
       * It demonstrates the use of :                                         *
       *                                                                      *
       * 1) MPI_Init                                                          *
       * 2) MPI_Comm_rank                                                     *
       * 3) MPI_Comm_size                                                     *
       * 4) MPI_Recv                                                          *
       * 5) MPI_Send                                                          *
       * 6) MPI_Finalize                                                      *
       * 7) MPI_allreduce                                                     *
       *                                                                      *
       ************************************************************************/
      long int n, i, j, ierr, num;
      double h, result, a, b, pi, global_sum;
      double my_a, my_range;
      time_t start, end;

      int myid, source, dest, tag, p, duration;
      MPI_Status status;
      double my_result;

      pi = acos(-1.0);  /* = 3.14159... */
      a = 0.;           /* lower limit of integration */
      b = pi * 1. / 2.; /* upper limit of integration */
      n = 100000000;    /* number of increment within each process */

      dest = 0;  /* define the process that computes the final result */
      tag = 123; /* set the tag to identify this particular job */

      /* Starts MPI processes ... */

      MPI_Init(&argc, &argv);               /* starts MPI */
      MPI_Comm_rank(MPI_COMM_WORLD, &myid); /* get current process id */
      MPI_Comm_size(MPI_COMM_WORLD, &p);    /* get number of processes */

      h = (b - a) / n; /* length of increment */
      num = n / p;     /* number of intervals calculated by each process*/
      my_range = (b - a) / p;
      my_a = a + myid * my_range;

      start = clock();
      my_result = integral(my_a, num, h);
      // printf("Process %d has the partial result of %f\n", myid, my_result);

      if (myid == 0)
      {
            result = my_result;
            printf("integral = %f\n", my_result);
            for (i = 1; i < p; i++)
            {
                  source = i; /* MPI process number range is [0,p-1] */
                  MPI_Recv(&my_result, 1, MPI_DOUBLE, source, tag,
                           MPI_COMM_WORLD, &status);
                  result += my_result;
                  printf("integral = %f\n", my_result);
            }

            printf("The result =%f\n", result);
            // printf("durasi = %f \n", (double)(end-start)/CLOCKS_PER_SEC);
      }
      else
      {
            MPI_Send(&my_result, 1, MPI_DOUBLE, dest, tag,
                     MPI_COMM_WORLD); /* send my_result to intended dest.
                                         */
      }
      end = clock();
      printf("durasi = %f \n", (double)(end - start) / CLOCKS_PER_SEC);
      // MPI_Reduce(&my_result, &global_sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
      // the parameters are = send_var, recv_var, no_array, data_type, operation, send_to,
world

      if (myid == 0)
      {
            end = clock();
            duration = end - start;
```

```
            printf("The result = %f and time = %d\n", result, duration);
      }

      MPI_Finalize();
      return 0; /* let MPI finish up ... */
}

double integral(double a, int n, double h)
{
      int j;
      double h2, aij, integ;

      integ = 0.0; /* initialize integral */
      h2 = h / 2.;
      for (j = 0; j < n; j++)
      {                       /* sum over all "j" integrals */
            aij = a + j * h; /* lower limit of "j" integral */
            integ += fct(aij + h2) * h;
      }
      return (integ);
}
```

Berikut adalah analisis kompleksitas waktu pada kode ini dan tabel yang menjelaskan komponen-komponennya:

Kompleksitas Waktu

1. **Penghitungan Parsial oleh Setiap Proses**
   - Fungsi integral menggunakan perulangan sebanyak $\frac{n}{p}$, yaitu jumlah interval yang dialokasikan untuk setiap proses.
   - Kompleksitas waktu fungsi integral: O($\frac{n}{p}$).
2. **Komunikasi Antar-Proses**
   - Proses root (ID = 0) mengumpulkan hasil dari p−1 proses lainnya.
   - Proses root menerima data dengan kompleksitas komunikasi O(p), di mana ppp adalah jumlah proses.
   - Proses lain hanya mengirim data sekali ke root, kompleksitas pengiriman untuk setiap proses non-root adalah O(1).
3. **Total Kompleksitas**
   - Total waktu komputasi (untuk semua proses): O($\frac{n}{p}$).
   - Waktu komunikasi untuk root: O(p).
   - **Total waktu keseluruhan program**: O($\frac{n}{p} + p$).

Bagian Code	Operasi Utama	Jumlah Iterasi/Kompleksitas
`MPI Initialization`	MPI_Init, MPI_Comm_rank, MPI_Comm_size	O(1)
`Pembagian Rentang`	Perhitungan my_range, my_a, num, dll. O(1)O(1)	O(1)
`Fungsi integral`	Perulangan menghitung integral parsial	O($\frac{n}{p}$)
`Pengumpulan Hasil (Root)`	Root menerima data dari p−1 proses	O(p)
`Pengiriman Hasil (Non-Root)`	Non-root mengirim data ke root	O(1)
`Total untuk Semua Proses`	**Komputasi + komunikasi**	**O($\frac{n}{p} + p$)**

Kompleksitas Multiproses = $O(\frac{n}{p} + p)$

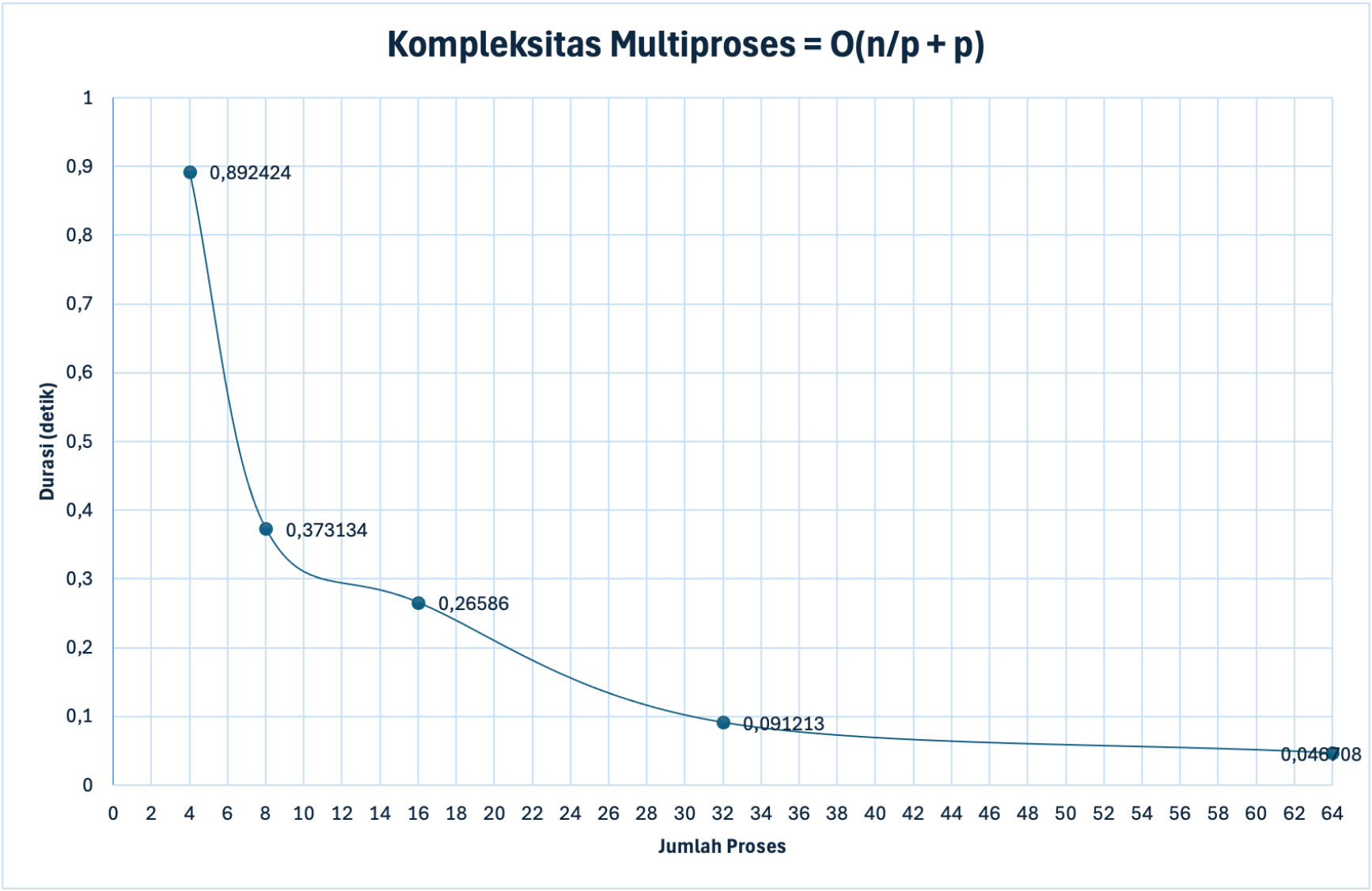
integral_mpi.c

integral_mpi.c > ...

```c
/* C Example */
#include <mpi.h>
#include <math.h>
#include <stdio.h>
#include <time.h>

double fct(double x)
{
      return cos(x);
}

/* Prototype */
double integral(double a, int n, double h);

int main(argc, argv)
int argc;
char *argv[];
{
      /************************************************************************
       *                                                                      *
       * This is one of the MPI versions on the integration example           *
       * It demonstrates the use of :                                         *
       *                                                                      *
       * 1) MPI_Init                                                          *
       * 2) MPI_Comm_rank                                                     *
       * 3) MPI_Comm_size                                                     *
       * 4) MPI_Recv                                                          *
       * 5) MPI_Send                                                          *
       * 6) MPI_Finalize                                                      *
       * 7) MPI_allreduce
       *                                                                      *
       ************************************************************************/
      long int n, i, j, ierr, num;
      double h, result, a, b, pi, global_sum;
      double my_a, my_range;
      time_t start, end;

      int myid, source, dest, tag, p, duration;
      MPI_Status status;
      double my_result;

```

Waktu yang dibutuhkan untuk menjalankan fungsi

Number of Process	Waktu (detik)
2^2	0,892424
2^3	0,373134
2^4	0,265860
2^5	0,091213
2^6	0,040708


Kompleksitas Multiproses = O(n/p + p)
Durasi (detik)
Jumlah Proses
0,892424
0,373134
0,26586
0,091213
0,046708
0
0,1
0,2
0,3
0,4
0,5
0,6
0,7
0,8
0,9
1
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64

LAMPIRAN PEMBUKTIAN

NP=4

```
(base) abdulrahman@192 code  % mpicc integral_mpi.c -o integral_mpi -lm

(base) abdulrahman@192 code  % mpirun -np 4 -oversubscribe ./integral_mpi
integral = 0.382683
durasi = 0.735134
integral = 0.324423
durasi = 1.064818
integral = 0.216773
integral = 0.076120
The result =1.000000
durasi = 1.065400
durasi = 0.892424
The result = 1.000000 and time = 892429
```

NP=8

```
(base) abdulrahman@192 code  % mpirun -np 8  -oversubscribe ./integral_mpi
integral = 0.195090
durasi = 0.368664
integral = 0.187593
durasi = 0.367042
durasi = 0.367654
integral = 0.172887
integral = 0.151537
durasi = 0.534875
integral = 0.124363
durasi = 0.535142
integral = 0.092410
integral = 0.056906
durasi = 0.535287
durasi = 0.534920
integral = 0.019215
The result =1.000000
durasi = 0.373134
```

NP=16

```
(base) abdulrahman@192 code  % mpirun -np 16  -oversubscribe integral_mpi
durasi = 0.182551
durasi = 0.181300
durasi = 0.181633
durasi = 0.180871
durasi = 0.181116
integral = 0.098017
integral = 0.097073
integral = 0.095194
integral = 0.092399
durasi = 0.180971
durasi = 0.181868
integral = 0.088713
integral = 0.084173
integral = 0.078823
integral = 0.072713
durasi = 0.264848
durasi = 0.266466
durasi = 0.265805
durasi = 0.265469
durasi = 0.265038
durasi = 0.264435
integral = 0.065904
integral = 0.058459
integral = 0.050452
integral = 0.041958
integral = 0.033061
durasi = 0.264282
integral = 0.023845
integral = 0.014399
integral = 0.004815
The result =1.000000
durasi = 0.183646
The result = 1.000000 and time = 183650
durasi = 0.265860
```

Dan seterusnya …