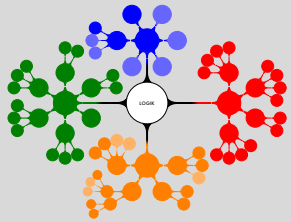
Lecture 11

Programming Shared Memory III

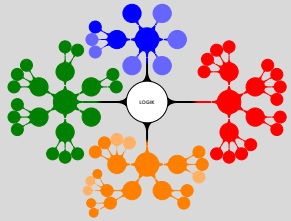
OpenMP (Open Multi-Processing); Parallelization Application Example - Pi Computation

IKC-MH.57 *Introduction to High Performance and Parallel Computing* at December 29, 2023

Dr. Cem Özdoğan
Engineering Sciences Department
İzmir Kâtip Çelebi University



Contents



Computing π I

Computing PI using Sequential Code

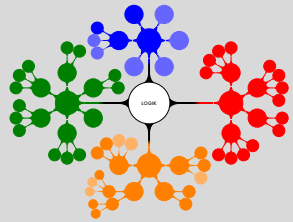
An OpenMP version of a threaded program to compute PI number using random numbers.

Computing PI using OpenMP directives - Random Numbers.

Follow the steps below for executing OpenMP code;

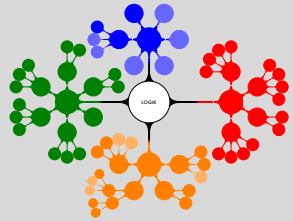
```
gcc -o code21 code21.c -fopenmp
./code21
```

```
1  /* ************************************************
2   An OpenMP version of a threaded program to compute PI.
3   ************************************************ */
4  #pragma omp parallel default(none) private(rand_no_x, rand_no_y,
       num_threads, sample_points_per_thread) shared(npoints)
       reduction(+: sum) num_threads(8)
5      {
6          num_threads = omp_get_num_threads();
7          sample_points_per_thread  = npoints / num_threads;
8
9          sum = 0;
10         for (int i = 0; i < sample_points_per_thread; i++) {
11             rand_no_x = (double)rand() / (double)RAND_MAX;
12             rand_no_y = (double)rand() / (double)RAND_MAX;
13             if (((rand_no_x - 0.5) * (rand_no_x - 0.5) + (
       rand_no_y - 0.5) * (rand_no_y - 0.5)) < 0.25) {
14                 sum = sum + 1;
15             }
16         }
17     }
```

Computing π II

- Note that this program is much easier to write in terms of specifying creation and termination of threads compared to the corresponding POSIX threaded program.
- The *omp_get_num_threads()* function returns the number of threads in the parallel region
- The *omp_get_thread_num()* function returns the integer id of each thread (recall that the master thread has an id 0).
- The *parallel directive* specifies that all variables except *npoints*, the total number of random points in two dimensions across all threads, are local.
- Furthermore, the directive specifies that there are eight threads, and the value of sum after all threads complete execution is the sum of local values at each thread.
- A for loop generates the required number of random points (in two dimensions) and determines how many of them are within the prescribed circle of unit diameter.

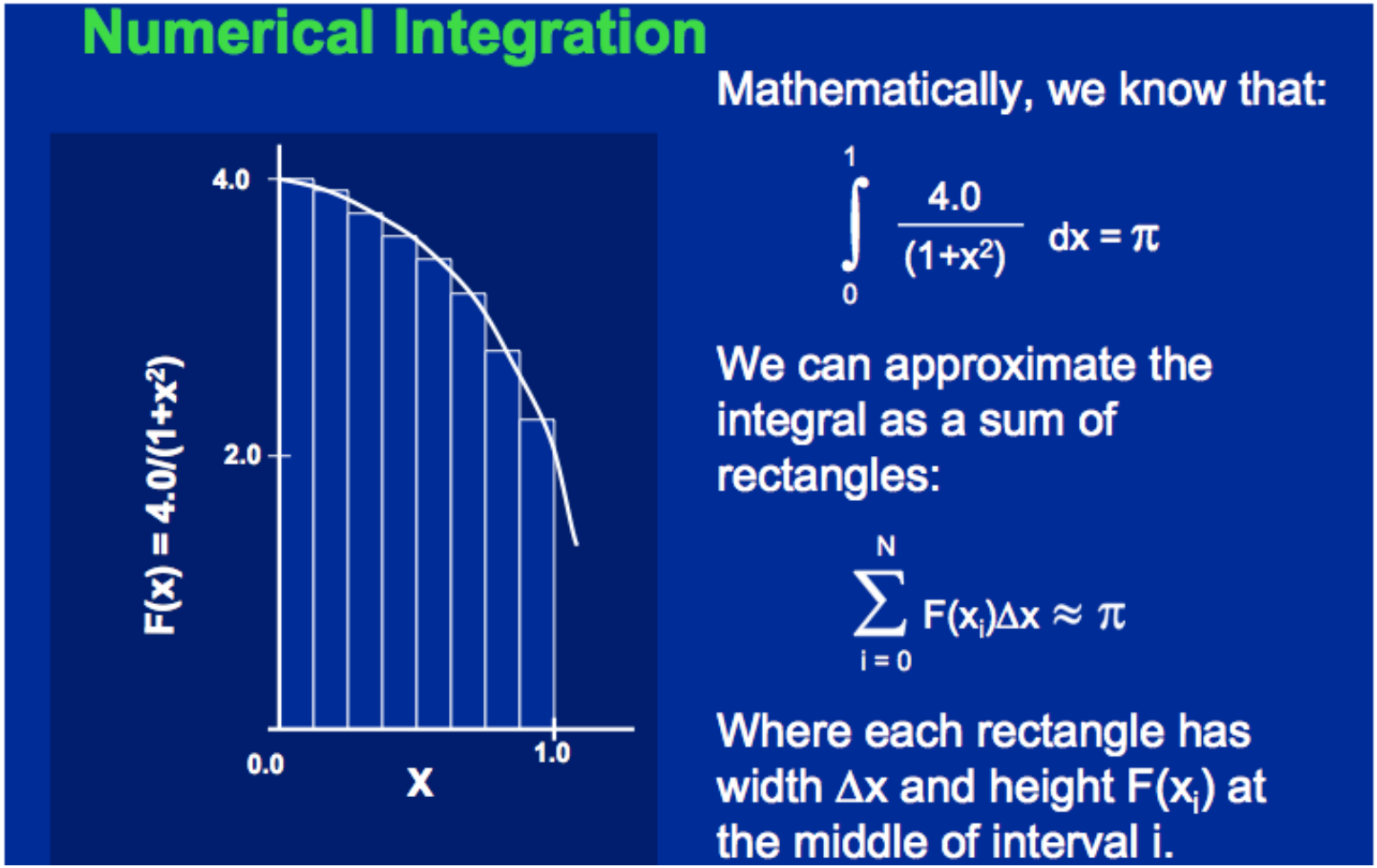


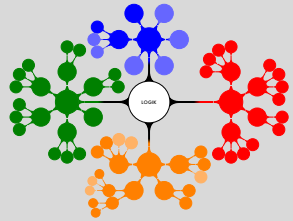
Computing π III

An OpenMP version of a threaded program t o compute PI number by numerical integration without reduction clause.

Computing PI using OpenMP directives - Numerical Integration. Follow the steps below;

```
export OMP_NUM_THREADS=4
gcc -o code22 code22.c -fopenmp
./code22
```

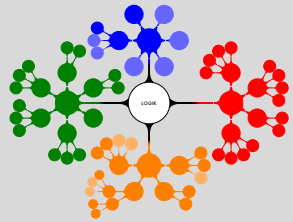




Computing π IV - Sequential Calculation

```c
#include <stdio.h>
#include <math.h>
int main(int argc, char* argv[])
{
    int done = 0, n, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, h, sum, x;
    while (!done)
    {
        printf("Enter the number of intervals: (0 quits) ");
        scanf("%d",&n);

        if (n == 0) break; /* Quit when "0" entered */
        /* Integral limits are from 0 to 1 */
        h   = (1.0-0.0)/(double)n; /* Step length */
        sum = 0.0;             /* Initialize sum variable */
        for (i = 1; i <= n; i += 1) /* loop over interval for integration */
        {
            x = h * ((double)i - 0.5); /* Middle point at step */
            sum += 4.0 / (1.0 + x*x);  /* Sum up at each step */
            //printf("i=%d x=%f sum=%f \n",i,x,sum); /* print intermediate steps */
        }
        mypi = h * sum; /* Obtain resulting pi number */
        printf("pi is approximately %.16f, Error is %.16f\n",mypi, fabs(mypi - PI25DT));
    }
}
```

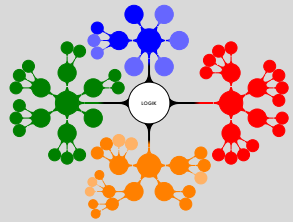
Computing π IV - OpenMP Parallel Calculation

```c
#include <stdio.h>
#include <math.h>
#include <omp.h>
#define NUM_THREADS 4

int main(int argc, char* argv[])
{
    int done = 0, n, i;
    double PI25DT = 3.14159265358979323846264 3;
    double mypi, h, sum[NUM_THREADS], x;

    while (!done)
    {
        printf("Enter the number of intervals: (0 quits) ");
        scanf("%d",&n);

        if (n == 0) break; /* Quit when "0" entered*/
        /* Integral limits are from 0 to 1 */
        h   = (1.0-0.0)/(double)n; /* Step length*/
        #pragma omp parallel private ( i, x )
        {
            int id = omp_get_thread_num();
            sum[id]=0.0;            /* Initialize sum variable */
            for (i = id+1; i <= n; i += NUM_THREADS) /* loop over interval for integration*/
            {
                x = h * ((double)i - 0.5); /* Middle point at step */
                sum[id] += 4.0 / (1.0 + x*x);  /* Sum up at each step */
            }
        }
        for(i=1; i<NUM_THREADS; i++)
            sum[0] += sum[i];
        mypi = h * sum[0]; /* Obtain resulting pi number */
        printf("pi is approximately %.16f, Error is %.16f\n",mypi, fabs(mypi - PI25DT));
    }
}
```

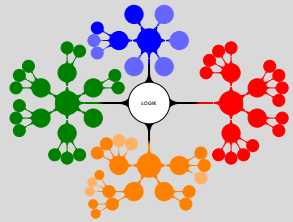
Computing π V - OpenMP Parallel Calculation

Access to Sequential Code & OpenMP Parallel Code

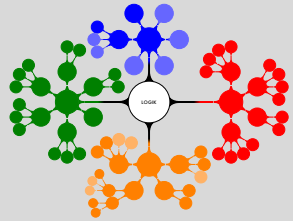
```
gcc -o sequential_pi sequential_pi.c
./sequential_pi
Enter the number of intervals: (0 quits) 100
pi is approximately 3.1416009869231254, Error is 0.0000083333333323
Enter the number of intervals: (0 quits) 1000
pi is approximately 3.1415927369231227, Error is 0.0000000833333296
Enter the number of intervals: (0 quits) 10000
pi is approximately 3.1415926544231341, Error is 0.0000000008333410
Enter the number of intervals: (0 quits) 0
```

```
gcc -o parallel_OpenMP_pi parallel_OpenMP_pi.c -fopenmp
./parallel_OpenMP_pi
Enter the number of intervals: (0 quits) 100
pi is approximately 3.1416009869231249, Error is 0.0000083333333318
Enter the number of intervals: (0 quits) 1000
pi is approximately 3.1415927369231267, Error is 0.0000000833333336
Enter the number of intervals: (0 quits) 10000
pi is approximately 3.1415926544231239, Error is 0.0000000008333307
Enter the number of intervals: (0 quits) 0
```

Table: Sequential and OpenMP outputs for computing π number.



Computing π VI - OpenMP Parallel Calculation: Reduction



An OpenMP version of a threaded program to compute PI number by numerical integration with reduction clause. **Computing PI** using OpenMP directives - Reduction. Follow the steps below for executing OpenMP code;

```
export OMP_NUM_THREADS=8
gcc -o code23 code23.c -fopenmp
./code23
```

Computing π VII - OpenMP Parallel Calculation: Reduction

```c
#include <stdio.h>
#include <math.h>
#include <omp.h>
int main(int argc, char* argv[])
{
    int done = 0, n, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, h, sum, x;
    while (!done)
    {
        printf("Enter the number of intervals: (0 quits) ");
        scanf("%d",&n);
        if (n == 0) break; /* Quit when "0" entered*/
        /* Integral limits are from 0 to 1 */
        h   = (1.0-0.0)/(double)n; /* Step length*/
        sum = 0.0;              /* Initialize sum variable */
        #pragma omp parallel for private(x) reduction (+:sum)
        for (i = 1; i <= n; i += 1) /* loop over interval for integration*/
            {
                x = h * ((double)i - 0.5);/*Middle point at step*/
                sum += 4.0 / (1.0 + x*x);/*Sum up at each step*/
//printf("i=%d x=%f sum=%f \n",i,x,sum); /*intermediate steps*/
            }
        mypi = h * sum; /* Obtain resulting pi number */
        printf("pi is approximately %.16f, Error is %.16f\n",mypi,
          fabs(mypi - PI25DT));
    }
}
```