Lab-SDA-PC05

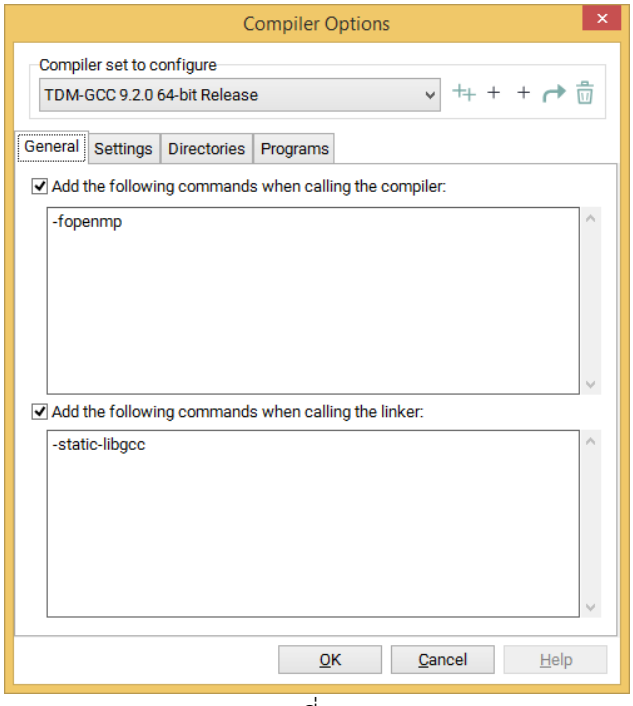
Introduction to Multi-core programming using OpenMP

1. พิจารณาโปรแกรมภาษาซีในรูปที่ 5.1 นำโค้ดดังกล่าวมาเขียนลงใน Dev C++ จากนั้นให้ตั้งค่าโปรเจ็กต์ใน Dev C++ ให้เป็นชนิด 64-bit Release แล้วเลือกเมนู Tools>Compiler Options เพิ่มข้อความ -fopenmp ลงในแท็บ General ของไดอะล็อกบอกซ์ Compiler Options ดังรูปที่ 5.2

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char *argv[])
{
    #pragma omp parallel
    {
       printf("Hello, from thread %d.\n", omp_get_thread_num());
       printf("Goodbye, from th %d\n", omp_get_thread_num());
    }
   return 0;
 }
/* Tools > Compiler Options > Check the option "Add the following commands
when compiler is called" > in the text area put "-fopenmp */
```

รูปที่ 5.1



รูปที่ 5.2

2. ลองคอมไพล์โปรแกรมในรูปที่ 5.3 และทดลองรันโปรแกรม สังเกตจำนวนเธรดที่รันขึ้นมาในโปรแกรม และลำดับการทำงานของเธรดแต่ละตัว และบันทึกผลการทดลอง

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <math.h>
#include <time.h>
#define n 20
//#define n 50000000

//---------------------------------------------------------
double calculate_elapsed_time(clock_t start, clock_t stop)
{
	return (double)(stop - start) / CLOCKS_PER_SEC;
}

//---------------------------------------------------------
int main(int argc, char *argv[])
{
	int i, tid, num_t;
	double *d, *s1, *s2, time_serial, time_parallel;
	d = (double*) malloc(sizeof(double)*n);
	s1 = (double*) malloc(sizeof(double)*n);
	s2 = (double*) malloc(sizeof(double)*n);
	clock_t start_time, stop_time;

    //-------------------------------------------------
    //initialize values in S1 and S2
    //-------------------------------------------------
	for (i=0; i<n; i++)
	{
		s1[i] = i;
		s2[i] = i*i;
	}

    //---------------------------------------------------------
	start_time = clock();
	#pragma omp parallel for
	for (i=0; i<n; i++)
	{
		d[i] = sin(s1[i])*s2[i] + cos(s2[i])* s1[i];
		num_t = omp_get_num_threads();
		tid = omp_get_thread_num();
		if (n<100)
			printf("i=%d in thread %d of %d\n", i, tid, num_t);
	}
	stop_time = clock();
	time_parallel = calculate_elapsed_time(start_time, stop_time);
	printf("Elapsed time (using openMP): %.2f seconds\n", time_parallel);

	//---------------------------------------------------------
	start_time = clock();
	for (i=0; i<n; i++)
		d[i] = sin(s1[i])*s2[i] + cos(s2[i])* s1[i];
	stop_time = clock();
	time_serial = calculate_elapsed_time(start_time, stop_time);
	printf("Elapsed time (not using openMP): %.2f seconds\n", time_serial);

	//---------------------------------------------------------
	printf("Parallel version is %.2f times faster than serial\n", time_serial/time_parallel);

	free(s2);
	free(s1);
	free(d);
	return 0;
 }
```

รูปที่ 5.3

3. จากโปรแกรมในรูปที่ 5.3 ให้จับเวลาการทำงานของโปรแกรมส่วนที่มีการทำงานแบบขนานซึ่งใช้ #pragma omp parallel for เปรียบเทียบกับส่วนของโปรแกรมที่ทำงานแบบธรรมดา สังเกตค่าเวลาของโปรแกรมทั้งสอง และจำนวนเธรดที่ใช้ในการรัน และทำการบันทึกค่าสถานะจำนวนคอร์ของซีพียูของเครื่องที่นักศึกษาใช้ในการรันลงในผลการทดลอง (คำแนะนำ สามารถตรวจสอบได้โดยใช้โปรแกรม CPU-Z)

4. ให้เปลี่ยนค่า n ของโปรแกรมในรูปที่ 5.3 จาก 20 เป็น 50000000 จากนั้นทำการคอมไพล์และรันโปรแกรม และหาว่าส่วนของโปรแกรมที่มีการทำงานแบบขนานสามารถทำงานได้เร็วกว่าส่วนที่ทำงานแบบธรรมดากี่เท่า

5. พิจารณาโปรแกรมในนรูปที่ 5.4 จงหาว่าเพราะเหตุใดผลลัพธ์ของการคำนวณแบบ Serial และแบบ Parallel จึงไม่เท่ากัน จงวิเคราะห์และอธิบายเหตุผลประกอบ

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <math.h>
#include <time.h>
#define n 50000000

//-------------------------------------------------------------
double calculate_elapsed_time(clock_t start, clock_t stop)
{
      return (double)(stop - start) / CLOCKS_PER_SEC;
}

//-------------------------------------------------------------
int main(int argc, char *argv[])
{
      int i, tid, num_t;
      double *s1, *s2;
      double r_serial=0.00, r_parallel=0.00;
      s1 = (double*) malloc(sizeof(double)*n);
      s2 = (double*) malloc(sizeof(double)*n);

      clock_t start_time, stop_time;
      double time_serial, time_parallel;

   //-------------------------------------------------
   //initialize values in S1 and S2
   //-------------------------------------------------
      for (i=0; i<n; i++)
   {
      s1[i] = i;
      s2[i] = i*i;
      }

   //---Parallel-----------------------------------------------------
      start_time = clock();
      #pragma omp parallel for
      for (i=0; i<n; i++)
   {
       r_parallel += sin(s1[i])*s2[i] + cos(s2[i])* s1[i];
       if (n<100)
       {
                   num_t = omp_get_num_threads();
                   tid = omp_get_thread_num();
                   printf("i=%d in thread %d of %d\n", i, tid, num_t);
          }
   }
```

```
    stop_time = clock();
    time_parallel = calculate_elapsed_time(start_time, stop_time);
       printf("Elapsed time (using openMP): %.2f seconds\n", time_parallel);

       //---Serial-------------------------------------------------
       start_time = clock();
       for (i=0; i<n; i++)
    {
       r_serial += sin(s1[i])*s2[i] + cos(s2[i])* s1[i];
    }
    stop_time = clock();
    time_serial = calculate_elapsed_time(start_time, stop_time);
       printf("Elapsed time (not using openMP): %.2f seconds\n", time_serial);

       //----------------------------------------------------
       printf("result serial = %.6f\n", r_serial);
       printf("result parallel = %.6f\n", r_parallel);
       printf("Parallel version is %.2f times faster than serial\n",
time_serial/time_parallel);

       free(s2);
       free(s1);
       return 0;
 }
```

รูปที่ 5.4

6. พิจารณาโปรแกรมในรูปที่ 5.5 จงเปรียบเทียบเวลาในการทำงานในโปรแกรมส่วนที่ทำงานแบบ Parallel และส่วนที่ทำงานแบบ Serial และหาว่าผลลัพธ์การทำงานของโปรแกรมแบบ Serial และแบบ Parallel เท่ากันหรือไม่ (ให้สังเกตการทำงานของฟังก์ชัน compare_two_double_values)

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <math.h>
#include <time.h>
#define n 50000000
#define epsilon 0.000000001

//------------------------------------------------------------
double calculate_elapsed_time(clock_t start, clock_t stop)
{
       return (double)(stop - start) / CLOCKS_PER_SEC;
}

//-------------------------------------------------------------------
//---this function returns 1 if two values are considered equal
int compare_two_double_values(double a, double b)
{
       double eps;
       if (a>b)
              eps = epsilon*a;
       else
              eps = epsilon*b;
       if (fabs(a-b) < eps)
              return 1;
       else
              return 0;
}

//-----------------------------------------------------------
int main(int argc, char *argv[])
{
```

```
      int i, tid, num_t;
      double *s1, *s2;
      double r_serial=0.00, r_parallel=0.00;
      s1 = (double*) malloc(sizeof(double)*n);
      s2 = (double*) malloc(sizeof(double)*n);

      clock_t start_time, stop_time;
      double time_serial, time_parallel;

   //-------------------------------------------------
   //initialize values in S1 and S2
   //-------------------------------------------------
      for (i=0; i<n; i++)
   {
      s1[i] = i;
      s2[i] = i*i;
      }

   //---Parallel-------------------------------------------------------
      start_time = clock();
      #pragma omp parallel for
      for (i=0; i<n; i++)
   {
       #pragma omp critical
       r_parallel += sin(s1[i])*s2[i] + cos(s2[i])* s1[i];
       if (n<100)
       {
                   num_t = omp_get_num_threads();
                   tid = omp_get_thread_num();
                   printf("i=%d in thread %d of %d\n", i, tid, num_t);
          }
   }
   stop_time = clock();
   time_parallel = calculate_elapsed_time(start_time, stop_time);
      printf("Elapsed time (using openMP): %.2f seconds\n", time_parallel);

      //---Serial---------------------------------------------------
      start_time = clock();
      for (i=0; i<n; i++)
   {
       r_serial += sin(s1[i])*s2[i] + cos(s2[i])* s1[i];
   }
   stop_time = clock();
   time_serial = calculate_elapsed_time(start_time, stop_time);
      printf("Elapsed time (not using openMP): %.2f seconds\n", time_serial);


      //------------------------------------------------------
      printf("result serial = %.6f\n", r_serial);
      printf("result parallel = %.6f\n", r_parallel);
      if (compare_two_double_values(r_serial, r_parallel))
            printf("result_parallel = result_serial\n");
      else
            printf("result_parallel != result_serial\n");
      printf("Parallel version is %.2f times faster than serial\n",
time_serial/time_parallel);

      free(s2);
      free(s1);
      return 0;
 }

 //Tools > Compiler Options > Check the option "Add the following commands when compiler
is called" > in the text area put "-fopenmp
```

รูปที่ 5.5

7. ให้เปรียบเทียบค่าเวลาการทำงานของโปรแกรมในรูปที่ 5.5 จงหาว่าโปรแกรมส่วน Parallel ทำงานได้เร็วหรือช้ากว่าส่วน Serial กี่เท่า

8. พิจารณาการทำงานของโปรแกรมในรูปที่ 5.6 สังเกตผลลัพธ์ของการทำงานของส่วนของโปรแกรมที่มีการทำงานแบบ Parallel และส่วนที่มีการทำงานแบบ Serial จงหาว่าผลลัพธ์การทำงานในส่วนที่เป็น Parallel และ Serial เท่ากันหรือไม่เมื่อเปรียบเทียบค่าสองค่าด้วยฟังก์ชัน compare_two_double_values

9. ให้เปรียบเทียบค่าเวลาการทำงานของโปรแกรมในรูปที่ 5.6 จงหาว่าโปรแกรมส่วน Parallel ทำงานได้เร็วหรือช้ากว่าส่วน Serial กี่เท่า

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <math.h>
#include <time.h>
#define n 50000000
#define epsilon 0.000000001

//------------------------------------------------------------
double calculate_elapsed_time(clock_t start, clock_t stop)
{
	return (double)(stop - start) / CLOCKS_PER_SEC;
}

//-------------------------------------------------------------------
//---this function returns 1 if two values are considered equal
int compare_two_double_values(double a, double b)
{
	double eps;
	if (a>b)
		eps = epsilon*a;
	else
		eps = epsilon*b;
	if (fabs(a-b) < eps)
		return 1;
	else
		return 0;
}

//------------------------------------------------------------
int main(int argc, char *argv[])
{
	int i, tid, num_t;
	double *s1, *s2;
	double r_serial=0.00, r_parallel=0.00;
	s1 = (double*) malloc(sizeof(double)*n);
	s2 = (double*) malloc(sizeof(double)*n);

	clock_t start_time, stop_time;
	double time_serial, time_parallel;

    //-------------------------------------------------
    //initialize values in S1 and S2
    //-------------------------------------------------
	for (i=0; i<n; i++)
    {
	   s1[i] = i;
	   s2[i] = i*i;
```

```
      }

   //---Parallel-------------------------------------------------------
      start_time = clock();
      #pragma omp parallel for reduction (+: r_parallel)
      for (i=0; i<n; i++)
   {
      r_parallel += sin(s1[i])*s2[i] + cos(s2[i])* s1[i];
      if (n<100)
      {
                  num_t = omp_get_num_threads();
                  tid = omp_get_thread_num();
                  printf("i=%d in thread %d of %d\n", i, tid, num_t);
         }
   }
   stop_time = clock();
   time_parallel = calculate_elapsed_time(start_time, stop_time);
      printf("Elapsed time (using openMP): %.2f seconds\n", time_parallel);

      //---Serial---------------------------------------------------------
      start_time = clock();
      for (i=0; i<n; i++)
   {
      r_serial += sin(s1[i])*s2[i] + cos(s2[i])* s1[i];
   }
   stop_time = clock();
   time_serial = calculate_elapsed_time(start_time, stop_time);
      printf("Elapsed time (not using openMP): %.2f seconds\n", time_serial);


      //------------------------------------------------------
      printf("result serial = %.6f\n", r_serial);
      printf("result parallel = %.6f\n", r_parallel);
      if (compare_two_double_values(r_serial, r_parallel))
            printf("result_parallel = result_serial\n");
      else
            printf("result_parallel != result_serial\n");
      printf("Parallel version is %.2f times faster than serial\n",
time_serial/time_parallel);

      free(s2);
      free(s1);
      return 0;
 }
```

รูปที่ 5.6

โปรแกรมสำหรับบวกเมทริกซ์

10. พิจารณาโปรแกรมในรูปที่ 5.7 ซึ่งเป็นโปรแกรมสำหรับบวกเมทริกซ์ ซึ่งมีการทำงานคือ C = A * B โดยกำหนดให้ C, A, B เป็นเมทริกซ์จัตุรัสขนาด row*row อิลิเมนต์ โดยในโปรแกรมกำหนดให้ row เท่ากับ 10 ทำการนำโค้ดดังกล่าวคอมไพล์ด้วย Dev C++ แล้วลองรันโปรแกรมแล้วสังเกตุผลลัพธ์การทำงาน

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <math.h>
#include <time.h>
#define row  10//12000
#define column      row
#define epsilon 0.000000001
typedef double array[row];
```

```
//-------------------------------------------------------------
double calculate_elapsed_time(clock_t start, clock_t stop)
{
	return (double)(stop - start) / CLOCKS_PER_SEC;
}

//-------------------------------------------------------------------
//---this function returns 1 if two values are considered equal
int compare_two_double_values(double a, double b)
{
	double eps;
	if (a>b)
		eps = epsilon*a;
	else
		eps = epsilon*b;
	if (fabs(a-b) < eps)
		return 1;
	else
		return 0;
}

void add_matrix_serial(double A[][column], double B[][column], double C[][column])
{
	for (int r=0; r<row; r++)
		for(int c=0; c<column; c++)
		{
			C[r][c] = A[r][c] + B[r][c];
		}
}

void add_matrix_parallel(double A[][column], double B[][column], double C[][column])
{
	#pragma omp parallel for
	for (int r=0; r<row; r++)
	{
		for(int c=0; c<column; c++)
		{
			C[r][c] = A[r][c] + B[r][c];
		}
	}
}



void print_matrix(double A[][column])
{
	int r, c;
	if (column <=12)
	{
		for (r=0; r<row; r++)
		{
			for(c=0; c<column; c++)
				printf("%.2f ", A[r][c]);
			printf("\n");
		}
	}
	else
		printf("Matrix is too large to be displayed on the screen\n");
}

//-------------------------------------------------------------
int main(int argc, char *argv[])
{
	int r,c, tid, num_t;

	array *A, *B, *C1, *C2;
	double *memA, *memB, *memC1, *memC2;
```

```
      memA = (double*) malloc(sizeof(double)*row*column);
      memB = (double*) malloc(sizeof(double)*row*column);
      memC2 = (double*) malloc(sizeof(double)*row*column);
      memC1 = (double*) malloc(sizeof(double)*row*column);
      A = (array *) memA;
      B = (array *) memB;
      C1 = (array *) memC1;
      C2 = (array *) memC2;
      clock_t start_time, stop_time;
      double time_serial, time_parallel;

   //-------------------------------------------------
   //initialize values of matrices A, B, and C
   //-------------------------------------------------
   for (r=0; r<row; r++)
            for (c=0; c<column; c++)
      {
            A[r][c] = (c+1.0) * (r+1.0);
            B[r][c] = (c+3.0) * (r+1.0);
            C1[r][c] = 0.0;
            C2[r][c] = 0.0;
            }

      printf("Matrix A\n");
      print_matrix(A);
      printf("\nMatrix B\n");
      print_matrix(B);

      //---Serial-------------------------------------------------
      start_time = clock();
      add_matrix_serial(A, B, C2);
   stop_time = clock();
   time_serial = calculate_elapsed_time(start_time, stop_time);
      printf("Elapsed time (Serial): %.2f seconds\n", time_serial);



  //---Parallel-----------------------------------------------------
      start_time = clock();
      add_matrix_parallel(A, B, C1);
   stop_time = clock();
   time_parallel = calculate_elapsed_time(start_time, stop_time);
      printf("Elapsed time (Parallel using openMP): %.2f seconds\n", time_parallel);


      printf("\nMatrix C1\n");
      print_matrix(C1);
      printf("\nMatrix C2\n");
      print_matrix(C2);

      printf("Parallel version is %.2f times faster than serial\n",
time_serial/time_parallel);

      free(memC1);
      free(memC2);
      free(memB);
      free(memA);
      return 0;
 }
```

รูปที่ 5.7

11. ทดลองเปลี่ยนค่า row จากค่า 10 เป็น 12000 แล้วลองทดสอบดูว่าโปรแกรมแบบขนานทำงานเร็วกว่าโปรแกรมแบบปกติเป็นจำนวนกี่เท่า

Checkpoint 1

จากโปรแกรมในรูปที่ 5.7 จงเขียนฟังก์ชัน compare_two_matrices เพื่อตรวจสอบว่าเมทริกซ์ C1 และ C2 เท่ากันทุกอิลิเมนต์หรือไม่ กำหนดให้ prototype ของฟังก์ชันดังกล่าวเป็นตามรูป 5.8

```
int compare_two_matrices(double C1[][column], double C2[][column]);
```

รูปที่ 5.8

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} \begin{bmatrix} g & h & i \\ j & k & l \end{bmatrix} \Rightarrow \begin{bmatrix} ag+bj & ah+bk & ai+bl \\ cg+dj & ch+dk & ci+dl \\ eg+fj & eh+fk & ei+fl \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \Rightarrow \begin{bmatrix} 27 & 30 & 33 \\ 61 & 68 & 75 \\ 95 & 106 & 117 \end{bmatrix}$$

รูปที่ 5.9 ตัวอย่างการคูณเมทริกซ์

Checkpoint 2

จงเขียนโปรแกรมเพื่อคูณเมทริกซ์ขนาด 2048x2048 อิลิเมนต์ โดยกำหนดให้มีฟังก์ชันในการคูณเมทริกซ์สองแบบ คือ แบบ parallel และแบบ serial แสดงดังรูปที่ 5.10 (คำแนะนำ ให้นำโปรแกรมในรูปที่ 5.7 มาดัดแปลง) พร้อมทั้งเปรียบเทียบความเร็วในการทำงานของฟังก์ชันทั้งสอง

```
void multiply_matrix_parallel(double A[][column], double B[][column], double
C[][column])
{
}

void multiply_matrix_serial(double A[][column], double B[][column], double C[][column])
{
}
```

รูปที่ 5.10

คำถามท้ายการทดลอง

1. โปรแกรมในรูปที่ 5.1 การรันแต่ละครั้งมีลำดับการทำงานของเธรดเหมือนกันหรือไม่ อย่างไร จงอธิบายเหตุผลว่าทำไมจึงได้ผลการทดลองเช่นนั้น

2. เพราะเหตุใดโปรแกรมในรูปที่ 5.5 จึงต้องเปรียบเทียบค่าของผลลัพธ์การทำงานแบบ Serial และแบบ Parallel โดยใช้ฟังก์ชัน compare_two_double_values ทำไมไม่เอาค่าทั้งสองมาเปรียบเทียบกันโดยใช้เครื่องหมาย == โดยตรงในภาษาซี