BÀI THỰC HÀNH SỐ 1

HP: HPC

NỘI DUNG:

- Cài đặt và thiết lập môi trường VS C/C++.NET với OpenMP
- Thực hiện các thuật toán song song HPC đơn giản
- Thực hiện các bài toán với thuật toán song song trong môi trường chia sẻ bộ nhớ (OpenMP)

YÊU CẦU: Viết báo cáo nộp cuối buổi với nội dung

- Liệt kê các công việc đã thục hiện (gạch đầu dòng)
- Kết quả chạy các chương trình ví dụ và giải thích

PHẦN 1: CÀI ĐẶT THIẾT LẬP MÔI TRƯỜNG VS C/C++.NET VỚI OPENMP

Bước 1: Cài đặt VS (bản Community) – Nếu máy chưa cài

Bước 2: Tạo dự án VC++. NET và cấu hình OpenMP

1. Tạo dự án VC++.NET
2. Mở hộp thoại trang thuộc tính của dự án (Kích chuột phải vào tên dự án/chọn Properties)
3. Chọn *Configuration Properties > C/C++ > Language*
4. Sửa lại thộc tính *OpenMP Support*
5. Test với chương trình

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[]) {
  int th_id, nthreads;
  #pragma omp parallel private(th_id)
  {
    th_id = omp_get_thread_num();
```

```
    printf("Hello World from thread %d\n", th_id);
    #pragma omp barrier
    if ( th_id == 0 ) {
      nthreads = omp_get_num_threads();
      printf("There are %d threads\n",nthreads);
    }
  }
  return EXIT_SUCCESS;
}
```

Kết quả chạy chương trình:

Hello World from thread 2

Hello World from thread 0

Hello World from thread 3

Hello World from thread 1

There are 4 threads

PHẦN 2: LÀM CHỦ THƯ VIỆN OPENMP API

Thư viện Runtime:

Các hàm thư viện Run-time OpenMP của VC++ chứa các thư viện sau:

OpenMP run-time library	Characteristics
VCOMP.LIB	Multithreaded, dynamic link (import library for VCOMP140.DLL).
VCOMPD.LIB	Multithreaded, dynamic link (import library for VCOMP140D.DLL) (debug)

Các cấu trúc của của thư viện chuẩn OpenMP của VC++:

Các cấu trúc Construct	Mô tả
Directives	Các lệnh chỉ dẫn của OpenMP API.
Clauses	Các mệnh đề được sử dụng trong OpenMP API.
Functions	Các hàm sử dụng trong OpenMP API.
Environment Variables	Các biến môi trường của OpenMP API.

1. Các lệnh chỉ dẫn

a) Các lệnh chỉ dẫn chia sẻ công việc song song

(Chỉ dẫn) Directive	Mô tả (Description)
parallel	Định nghĩa vùng song song mà mã sẽ được thi hành bởi nhiều luồng song song.
for	Thực hiện vòng lặp for bên trong một vùng song song được chia cho các luồng.
sections	Xác định các phần mã sẽ được chia cho tất cả các luồng.
single	Chỉ định phần mã được thực thi trên một luồng duy nhất, không nhất thiết phải là luồng chính.

b) Các lệnh chỉ dẫn cho luồng chính và thực hiện đồng bộ hóa

Directive	Description
master	Chỉ ra chỉ có luồng chính được thực thi một phần của chương trình .

Directive	Description
critical	Mã chỉ được thực thi một luồng tại mỗi thời điểm.
barrier	Đồng bộ hóa tất cả accs luồng trong một nhóm luồng (team). Tất cả các luồng sẽ tạm dừng tại barrier (thanh chắn) cho đến khi tất cả các luồng thực thi barrier.
atomic	Chỉ định rằng một vị trí bộ nhớ sẽ được cập nhật nguyên tử.
flush	Chỉ định rằng tất cả các luồng có cùng một khung nhìn bộ nhớ cho tất cả các đối tượng được chia sẻ.
ordered	Chỉ định rằng mã trong vòng lặp for song song hóa sẽ được thực thi giống như một vòng lặp tuần tự.

c) Chỉ dẫn đối với môi trường dữ liệu

Directive	Description
threadprivate	Chỉ ra một biến chỉ là biến riêng của một luồng.

1.1) Automic:

Cú pháp:

#pragma omp atomic

Expression

Ví dụ:

```
// omp_atomic.cpp
// compile with: /openmp
#include <stdio.h>
#include <omp.h>
#define MAX 10
int main() {
```

```
  int count = 0;
  #pragma omp parallel num_threads(MAX)
  {
    #pragma omp atomic
    count++;
  }
  printf_s("Number of threads: %d\n", count);
}
```

1.2) barrier

Cú pháp:

```
#pragma omp barrier
```

Ví dụ:

```
/* ERROR - The barrier directive cannot be the immediate
*         substatement of an if statement
*/
if (x!=0)
  #pragma omp barrier
...

/* OK - The barrier directive is enclosed in a
*      compound statement.
*/
if (x!=0) {
  #pragma omp barrier
}
```

1.3) Critical

Cú pháp:

```
#pragma omp critical [(name)]
{
  code_block
}
```

Ví dụ:

```
// omp_critical.cpp
// compile with: /openmp
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
int main()
{
   int i;
   int max;
   int a[SIZE];
   for (i = 0; i < SIZE; i++)
   {
      a[i] = rand();
      printf_s("%d\n", a[i]);
   }
   max = a[0];
   #pragma omp parallel for num_threads(4)
      for (i = 1; i < SIZE; i++)
      {
         if (a[i] > max)
         {
            #pragma omp critical
            {
               // compare a[i] and max again because max
               // could have been changed by another thread after
               // the comparison outside the critical section
               if (a[i] > max)
                  max = a[i];
            }
         }
      }
```

1.4) flush

Cú pháp:

#pragma omp flush [(var)]

Ví dụ:

```
// omp_flush.cpp
// compile with: /openmp
#include <stdio.h>
#include <omp.h>
void read(int *data) {
  printf_s("read data\n");
  *data = 1;
}
void process(int *data) {
  printf_s("process data\n");
  (*data)++;
}
int main() {
  int data;
  int flag;
  flag = 0;
  #pragma omp parallel sections num_threads(2)
  {
    #pragma omp section
    {
      printf_s("Thread %d: ", omp_get_thread_num( ));
      read(&data);
      #pragma omp flush(data)
      flag = 1;
      #pragma omp flush(flag)
      // Do more work.
    }
    #pragma omp section
    {
      while (!flag) {
        #pragma omp flush(flag)
      }
      #pragma omp flush(data)
      printf_s("Thread %d: ", omp_get_thread_num( ));
```

```
        process(&data);
        printf_s("data = %d\n", data);
      }
    }
}
   printf_s("max = %d\n", max);
}
```

1.5) for

Cú pháp:

```
#pragma omp [parallel] for [clauses]
  for_statement
```

Các mệnh đề:

- private
- firstprivate
- lastprivate
- reduction
- ordered
- schedule
- nowait

Ví dụ:

```
// omp_for.cpp
// compile with: /openmp
#include <stdio.h>
#include <math.h>
#include <omp.h>
#define NUM_THREADS 4
#define NUM_START 1
#define NUM_END 10
int main() {
  int i, nRet = 0, nSum = 0, nStart = NUM_START, nEnd = NUM_END;
  int nThreads = 0, nTmp = nStart + nEnd;
  unsigned uTmp = (unsigned((abs(nStart - nEnd) + 1)) *
                   unsigned(abs(nTmp))) / 2;
  int nSumCalc = uTmp;
  if (nTmp < 0)
    nSumCalc = -nSumCalc;
```

```
  omp_set_num_threads(NUM_THREADS);
  #pragma omp parallel default(none) private(i) shared(nSum, nThreads, nStart,
nEnd)
  {
    #pragma omp master
    nThreads = omp_get_num_threads();
    #pragma omp for
    for (i=nStart; i<=nEnd; ++i) {
        #pragma omp atomic
        nSum += i;
    }
  }
  if  (nThreads == NUM_THREADS) {
    printf_s("%d OpenMP threads were used.\n", NUM_THREADS);
    nRet = 0;
  }
  else {
    printf_s("Expected %d OpenMP threads, but %d were used.\n",
          NUM_THREADS, nThreads);
    nRet = 1;
  }

  if (nSum != nSumCalc) {
    printf_s("The sum of %d through %d should be %d, "
          "but %d was reported!\n",
          NUM_START, NUM_END, nSumCalc, nSum);
    nRet = 1;
  }
  else
    printf_s("The sum of %d through %d is %d\n",
          NUM_START, NUM_END, nSum);
}
```

1.6) Master

Cú pháp:

```
#pragma omp master
{
  code_block
}
```

Ví dụ:

```
// compile with: /openmp
#include <omp.h>
#include <stdio.h>
int main( )
{
   int a[5], i;
   #pragma omp parallel
   {
      // Perform some computation.
      #pragma omp for
      for (i = 0; i < 5; i++)
         a[i] = i * i;

      // Print intermediate results.
      #pragma omp master
         for (i = 0; i < 5; i++)
            printf_s("a[%d] = %d\n", i, a[i]);
      // Wait.
      #pragma omp barrier
      // Continue with the computation.
      #pragma omp for
      for (i = 0; i < 5; i++)
         a[i] += i;
   }
}
```

1.7) Ordered

Cú pháp:

#pragma omp ordered

structured-block

Ví dụ:

```
// omp_ordered.cpp
// compile with: /openmp
#include <stdio.h>
#include <omp.h>
static float a[1000], b[1000], c[1000];
void test(int first, int last)
{
   #pragma omp for schedule(static) ordered
   for (int i = first; i <= last; ++i) {
      // Do something here.
      if (i % 2)
      {
         #pragma omp ordered
         printf_s("test() iteration %d\n", i);
      }
   }
}
void test2(int iter)
{
   #pragma omp ordered
   printf_s("test2() iteration %d\n", iter);
}
int main( )
{
   int i;
   #pragma omp parallel
   {
      test(1, 8);
      #pragma omp for ordered
      for (i = 0 ; i < 5 ; i++)
         test2(i);
   }
```

}

1.8) Parallel

Cú pháp:

```
#pragma omp parallel [clauses]
{
  code_block
}
```

[clauses]

- if
- private
- firstprivate
- default
- shared
- copyin
- reduction
- num_threads

Ví dụ:

```
// omp_parallel.cpp
// compile with: /openmp
#include <stdio.h>
#include <omp.h>
int main() {
  #pragma omp parallel num_threads(4)
  {
    int i = omp_get_thread_num();
    printf_s("Hello from thread %d\n", i);
  }
}
```

1.9) sections

Cú pháp:

```
#pragma omp [parallel] sections [clauses]
{
  #pragma omp section
  {
    code_block
  }
}
```

[clauses]

- private
- firstprivate
- lastprivate
- reduction
- nowait

Ví dụ:

```
// omp_sections.cpp
// compile with: /openmp
#include <stdio.h>
#include <omp.h>
int main() {
   #pragma omp parallel sections num_threads(4)
   {
      printf_s("Hello from thread %d\n", omp_get_thread_num());
      #pragma omp section
      printf_s("Hello from thread %d\n", omp_get_thread_num());
   }
}
```

1.10) Single

Cú pháp:

```
#pragma omp single [clauses]
{
  code_block
}
```

[clauses]

- private
- firstprivate
- copyprivate
- nowait

Ví dụ:

```
// omp_single.cpp
// compile with: /openmp
#include <stdio.h>
#include <omp.h>
int main() {
  #pragma omp parallel num_threads(2)
  {
    #pragma omp single
    // Only a single thread can read the input.
    printf_s("read input\n");
    // Multiple threads in the team compute the results.
    printf_s("compute results\n");
    #pragma omp single
    // Only a single thread can write the output.
    printf_s("write output\n");
   }
}
```

1.11) Threadprivate

Cú pháp:

#pragma omp threadprivate(var)

Ví dụ:

```
struct MyType
{
   ~MyType();
};
MyType threaded_var;
#pragma omp threadprivate(threaded_var)
int main()
{
   #pragma omp parallel
   {}
}
```

2. Các mệnh đề (Clause)

Cú pháp các mệnh đề:

copyin(var)

copyprivate(var)

default(shared | none)

if(expression)

lastprivate(var)

nowait

num_threads(num)

ordered

private(var)

schedule(type[,size])

shared(var)

Vi dụ 1: Mệnhđề CopyPrivate;

```
// omp_copyprivate.cpp
// compile with: /openmp
```

```
#include <stdio.h>
#include <omp.h>
float x, y, fGlobal = 1.0;
#pragma omp threadprivate(x, y)
float get_float() {
  fGlobal += 0.001;
  return fGlobal;
}
void use_float(float f, int t) {
  printf_s("Value = %f, thread = %d\n", f, t);
}
void CopyPrivate(float a, float b) {
  #pragma omp single copyprivate(a, b, x, y)
  {
    a = get_float();
    b = get_float();
    x = get_float();
    y = get_float();
   }
  use_float(a, omp_get_thread_num());
  use_float(b, omp_get_thread_num());
  use_float(x, omp_get_thread_num());
  use_float(y, omp_get_thread_num());
}
int main() {
```

```
    float a = 9.99, b = 123.456;
    printf_s("call CopyPrivate from a single thread\n");
    CopyPrivate(9.99, 123.456);
    printf_s("call CopyPrivate from a parallel region\n");
    #pragma omp parallel
    {
      CopyPrivate(a, b);
    }
  }
```

Ví dụ 2: Mệnh đề if:

```
  // omp_if.cpp
  // compile with: /openmp
  #include <stdio.h>
  #include <omp.h>
  void test(int val)
  {
    #pragma omp parallel if (val)
    if (omp_in_parallel())
    {
      #pragma omp single
      printf_s("val = %d, parallelized with %d threads\n",
            val, omp_get_num_threads());
    }
    else
    {
```

```
            printf_s("val = %d, serialized\n", val);
        }
    }
    int main( )
    {
        omp_set_num_threads(2);
        test(0);
        test(2);
    }
```

Ví dụ 3: Mệnh đề nowait

```
    // omp_nowait.cpp
    // compile with: /openmp /c
    #include <stdio.h>
    #define SIZE 5
    void test(int *a, int *b, int *c, int size)
    {
        int i;
        #pragma omp parallel
        {
            #pragma omp for nowait
            for (i = 0; i < size; i++)
                b[i] = a[i] * a[i];

            #pragma omp for nowait
            for (i = 0; i < size; i++)
```

```
            c[i] = a[i]/2;
        }
    }
    int main( )
    {
        int a[SIZE], b[SIZE], c[SIZE];
        int i;
        for (i=0; i<SIZE; i++)
            a[i] = i;
        test(a,b,c, SIZE);
        for (i=0; i<SIZE; i++)
            printf_s("%d, %d, %d\n", a[i], b[i], c[i]);
    }
```

Ví dụ 4: Mệnh đề private

```
    // openmp_private.c
    // compile with: /openmp
    #include <windows.h>
    #include <assert.h>
    #include <stdio.h>
    #include <omp.h>
    #define NUM_THREADS 4
    #define SLEEP_THREAD 1
    #define NUM_LOOPS 2
    enum Types {
      ThreadPrivate,
```

```
   Private,
   FirstPrivate,
   LastPrivate,
   Shared,
   MAX_TYPES
};
int nSave[NUM_THREADS][MAX_TYPES][NUM_LOOPS] = {{0}};
int nThreadPrivate;
#pragma omp threadprivate(nThreadPrivate)
#pragma warning(disable:4700)
int main() {
   int nPrivate = NUM_THREADS;
   int nFirstPrivate = NUM_THREADS;
   int nLastPrivate = NUM_THREADS;
   int nShared = NUM_THREADS;
   int nRet = 0;
   int i;
   int j;
   int nLoop = 0;
   nThreadPrivate = NUM_THREADS;
   printf_s("These are the variables before entry "
         "into the parallel region.\n");
   printf_s("nThreadPrivate = %d\n", nThreadPrivate);
   printf_s("      nPrivate = %d\n", nPrivate);
   printf_s(" nFirstPrivate = %d\n", nFirstPrivate);
```

```
  printf_s("  nLastPrivate = %d\n", nLastPrivate);
  printf_s("       nShared = %d\n\n", nShared);
  omp_set_num_threads(NUM_THREADS);
  #pragma omp parallel copyin(nThreadPrivate) private(nPrivate) shared(nShared) firstprivate(nFirstPrivate)
  {
    #pragma omp for schedule(static) lastprivate(nLastPrivate)
    for (i = 0 ; i < NUM_THREADS ; ++i) {
      for (j = 0 ; j < NUM_LOOPS ; ++j) {
        int nThread = omp_get_thread_num();
        assert(nThread < NUM_THREADS);
        if (nThread == SLEEP_THREAD)
          Sleep(100);
        nSave[nThread][ThreadPrivate][j] = nThreadPrivate;
        nSave[nThread][Private][j] = nPrivate;
        nSave[nThread][Shared][j] = nShared;
        nSave[nThread][FirstPrivate][j] = nFirstPrivate;
        nSave[nThread][LastPrivate][j] = nLastPrivate;
        nThreadPrivate = nThread;
        nPrivate = nThread;
        nShared = nThread;
        nLastPrivate = nThread;
        --nFirstPrivate;
      }
    }
  }
```

```
for (i = 0 ; i < NUM_LOOPS ; ++i) {
  for (j = 0 ; j < NUM_THREADS ; ++j) {
    printf_s("These are the variables at entry of "
          "loop %d of thread %d.\n", i + 1, j);
    printf_s("nThreadPrivate = %d\n",
          nSave[j][ThreadPrivate][i]);
    printf_s("     nPrivate = %d\n",
          nSave[j][Private][i]);
    printf_s(" nFirstPrivate = %d\n",
          nSave[j][FirstPrivate][i]);
    printf_s("  nLastPrivate = %d\n",
          nSave[j][LastPrivate][i]);
    printf_s("      nShared = %d\n\n",
          nSave[j][Shared][i]);
  }
}
printf_s("These are the variables after exit from "
      "the parallel region.\n");
printf_s("nThreadPrivate = %d (The last value in the "
      "main thread)\n", nThreadPrivate);
printf_s("     nPrivate = %d (The value prior to "
      "entering parallel region)\n", nPrivate);
printf_s(" nFirstPrivate = %d (The value prior to "
      "entering parallel region)\n", nFirstPrivate);
```

```
        printf_s("  nLastPrivate = %d (The value from the "
                "last iteration of the loop)\n", nLastPrivate);
        printf_s("      nShared = %d (The value assigned, "
                "from the delayed thread, %d)\n\n",
                nShared, SLEEP_THREAD);
    }
```

Ví dụ 5: Mệnh đề reduction

```
// omp_reduction.cpp
// compile with: /openmp
#include <stdio.h>
#include <omp.h>
#define NUM_THREADS 4
#define SUM_START   1
#define SUM_END     10
#define FUNC_RETS   {1, 1, 1, 1, 1}
int bRets[5] = FUNC_RETS;
int nSumCalc = ((SUM_START + SUM_END) * (SUM_END - SUM_START + 1))
/ 2;
int func1( ) {return bRets[0];}
int func2( ) {return bRets[1];}
int func3( ) {return bRets[2];}
int func4( ) {return bRets[3];}
int func5( ) {return bRets[4];}
int main( )
{
   int nRet = 0,
```

```
    nCount = 0,
    nSum = 0,
    i,
    bSucceed = 1;
omp_set_num_threads(NUM_THREADS);
#pragma omp parallel reduction(+ : nCount)
{
    nCount += 1;
    #pragma omp for reduction(+ : nSum)
    for (i = SUM_START ; i <= SUM_END ; ++i)
        nSum += i;
    #pragma omp sections reduction(&& : bSucceed)
    {
        #pragma omp section
        {
            bSucceed = bSucceed && func1( );
        }
        #pragma omp section
        {
            bSucceed = bSucceed && func2( );
        }
        #pragma omp section
        {
            bSucceed = bSucceed && func3( );
        }
```

```
        #pragma omp section
        {
            bSucceed = bSucceed && func4( );
        }
        #pragma omp section
        {
            bSucceed = bSucceed && func5( );
        }
    }
}
printf_s("The parallel section was executed %d times "
        "in parallel.\n", nCount);
printf_s("The sum of the consecutive integers from "
        "%d to %d, is %d\n", 1, 10, nSum);
if (bSucceed)
    printf_s("All of the functions, func1 through "
            "func5 succeeded!\n");
else
    printf_s("One or more of the functions, func1 "
            "through func5 failed!\n");
if (nCount != NUM_THREADS)
{
    printf_s("ERROR: For %d threads, %d were counted!\n",
            NUM_THREADS, nCount);
```

```
        nRet |= 0x1;
    }
    if (nSum != nSumCalc)
    {
        printf_s("ERROR: The sum of %d through %d should be %d, "
                "but %d was reported!\n",
                SUM_START, SUM_END, nSumCalc, nSum);
        nRet |= 0x10;
    }
    if (bSucceed != (bRets[0] && bRets[1] &&
                    bRets[2] && bRets[3] && bRets[4]))
    {
        printf_s("ERROR: The sum of %d through %d should be %d, "
                 "but %d was reported!\n",
                 SUM_START, SUM_END, nSumCalc, nSum);
        nRet |= 0x100;
    }
}
```

Ví dụ 6: Mệnh đề schedule

```
// omp_schedule.cpp
// compile with: /openmp
#include <windows.h>
#include <stdio.h>
#include <omp.h>
```

```
#define NUM_THREADS 4
#define STATIC_CHUNK 5
#define DYNAMIC_CHUNK 5
#define NUM_LOOPS 20
#define SLEEP_EVERY_N 3
int main( )
{
   int nStatic1[NUM_LOOPS],
      nStaticN[NUM_LOOPS];
   int nDynamic1[NUM_LOOPS],
      nDynamicN[NUM_LOOPS];
   int nGuided[NUM_LOOPS];
   omp_set_num_threads(NUM_THREADS);
   #pragma omp parallel
   {
      #pragma omp for schedule(static, 1)
      for (int i = 0 ; i < NUM_LOOPS ; ++i)
      {
         if ((i % SLEEP_EVERY_N) == 0)
            Sleep(0);
         nStatic1[i] = omp_get_thread_num( );
      }
      #pragma omp for schedule(static, STATIC_CHUNK)
      for (int i = 0 ; i < NUM_LOOPS ; ++i)
      {
```

```
    if ((i % SLEEP_EVERY_N) == 0)
        Sleep(0);
    nStaticN[i] = omp_get_thread_num( );
}
#pragma omp for schedule(dynamic, 1)
for (int i = 0 ; i < NUM_LOOPS ; ++i)
{
    if ((i % SLEEP_EVERY_N) == 0)
        Sleep(0);
    nDynamic1[i] = omp_get_thread_num( );
}
#pragma omp for schedule(dynamic, DYNAMIC_CHUNK)
for (int i = 0 ; i < NUM_LOOPS ; ++i)
{
    if ((i % SLEEP_EVERY_N) == 0)
        Sleep(0);
    nDynamicN[i] = omp_get_thread_num( );
}

#pragma omp for schedule(guided)
for (int i = 0 ; i < NUM_LOOPS ; ++i)
{
    if ((i % SLEEP_EVERY_N) == 0)
        Sleep(0);
    nGuided[i] = omp_get_thread_num( );
```

```
    }
  }
  printf_s("-----------------------------------------------\n");
  printf_s("| static | static | dynamic | dynamic | guided |\n");
  printf_s("|   1   |   %d   |   1   |   %d   |       |\n",
        STATIC_CHUNK, DYNAMIC_CHUNK);
  printf_s("-----------------------------------------------\n");
  for (int i=0; i<NUM_LOOPS; ++i)
  {
    printf_s("|   %d   |   %d   |   %d   |   %d   |"
          "   %d   |\n",
          nStatic1[i], nStaticN[i],
          nDynamic1[i], nDynamicN[i], nGuided[i]);
  }
  printf_s("-----------------------------------------------\n");
}
```

Tham khảo các ví dụ khác theo link:

https://curc.readthedocs.io/en/latest/programming/OpenMP-C.html

3. Các biến môi trường

Các biến môi trường được đọc khi khởi động chương trình và các sửa đổi đối với giá trị của chúng bị bỏ qua trong thời gian chạy.

Environment variable	Description
OMP_SCHEDULE	Sửa đổi hành vi của mệnh đề schedule khi lịch trình (thời gian chạy) được chỉ định trong lệnh for hoặc parallel của lệnh chỉ dẫn.

Environment variable	Description
OMP_NUM_THREADS	Đặt số luồng tối đa trong vùng song song, trừ khi bị ghi đè bởi omp_set_num_threads hoặc num_threads.
OMP_DYNAMIC	Chỉ định xem thời gian chạy OpenMP có thể điều chỉnh số luồng trong một vùng song song hay không.
OMP_NESTED	Chỉ định xem chế độ song song lồng nhau có được bật hay không, trừ khi chế độ song song lồng nhau được bật hoặc tắt bằng omp_set_nested.

4. Các hàm OpenMP API

Bao gồm các hàm và các kiểu dữ liệu.

Sử dụng link sau để hiểu công dụng của hàm và ví dụ:

https://github.com/MicrosoftDocs/cpp-docs/blob/main/docs/parallel/openmp/reference/openmp-functions.md#omp-set-num-threads

a) Các hàm đối với môi trược thực thi

Function	Description
omp_set_num_threads	Sets the number of threads in upcoming parallel regions, unless overridden by a num_threads clause.
omp_get_num_threads	Returns the number of threads in the parallel region.
omp_get_max_threads	Returns an integer that is equal to or greater than the number of threads that would be available if a parallel region without num_threads were defined at that point in the code.
omp_get_thread_num	Returns the thread number of the thread executing within its thread team.

Function	Description
omp_get_num_procs	Returns the number of processors that are available when the function is called.
omp_in_parallel	Returns nonzero if called from within a parallel region.
omp_set_dynamic	Indicates that the number of threads available in upcoming parallel regions can be adjusted by the run time.
omp_get_dynamic	Returns a value that indicates if the number of threads available in upcoming parallel regions can be adjusted by the run time.
omp_set_nested	Enables nested parallelism.
omp_get_nested	Returns a value that indicates if nested parallelism is enabled.

b) ***Các hàm đối với đồng hồ xung nhịp***

Function	Description
omp_init_lock	Initializes a simple lock.
omp_init_nest_lock	Initializes a lock.
omp_destroy_lock	Uninitializes a lock.
omp_destroy_nest_lock	Uninitializes a nestable lock.
omp_set_lock	Blocks thread execution until a lock is available.

Function	Description
omp_set_nest_lock	Blocks thread execution until a lock is available.
omp_unset_lock	Releases a lock.
omp_unset_nest_lock	Releases a nestable lock.
omp_test_lock	Attempts to set a lock but doesn't block thread execution.
omp_test_nest_lock	Attempts to set a nestable lock but doesn't block thread execution.

Data type	Description
omp_lock_t	A type that holds the status of a lock, whether the lock is available or if a thread owns a lock.
omp_nest_lock_t	A type that holds one of the following pieces of information about a lock: whether the lock is available, and the identity of the thread that owns the lock and a nesting count.

c) Các hàm đối với thường trình thời gian

Function	Description
omp_get_wtime	Returns a value in seconds of the time elapsed from some point.
omp_get_wtick	Returns the number of seconds between processor clock ticks.

PHẦN 3. LẬP TRÌNH SONG SONG VỚI MỘT SỐ THUẬT TOÁN ĐƠN GIẢN

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

int main(){

   double itime, ftime, exec_time;
   itime = omp_get_wtime();

   // Required code for which execution time has to be found

   ftime = omp_get_wtime();
   exec_time = ftime - itime;
   printf("\n\nTime taken is is %f", exec_gap);

}
```

```
// cpp_compiler_options_openmp.cpp
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
volatile DWORD dwStart;
volatile int global = 0;
double test2(int num_steps) {
  int i;
  global++;
  double x, pi, sum = 0.0, step;
  step = 1.0 / (double) num_steps;
  #pragma omp parallel for reduction(+:sum) private(x)
  for (i = 1; i <= num_steps; i++) {
    x = (i - 0.5) * step;
    sum = sum + 4.0 / (1.0 + x*x);
  }
  pi = step * sum;
  return pi;
}
int main(int argc, char* argv[]) {
  double   d;
  int n = 1000000;
  if (argc > 1)
    n = atoi(argv[1]);
```

```
   dwStart = GetTickCount();
   d = test2(n);
   printf_s("For %d steps, pi = %.15f, %d milliseconds\n", n, d, GetTickCount() - dwStart);
   dwStart = GetTickCount();
   d = test2(n);
   printf_s("For %d steps, pi = %.15f, %d milliseconds\n", n, d, GetTickCount() - dwStart);
}
```