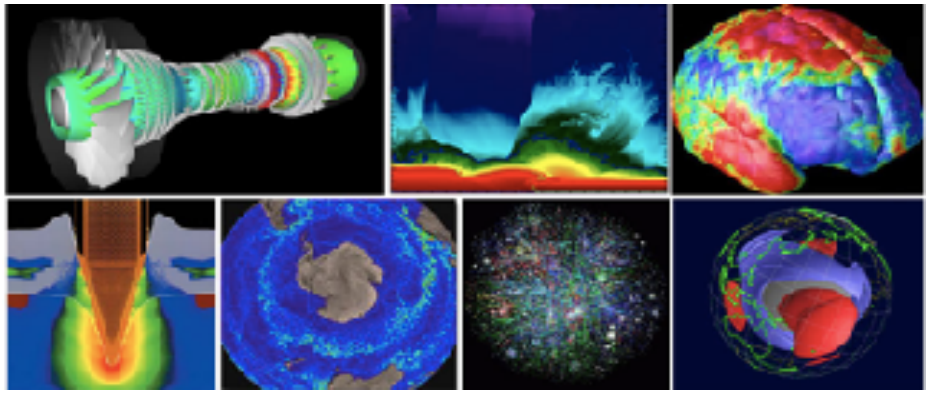
Tính toán song song và phân tán

PGS.TS. Trần Văn Lăng
langtv@vast.vn
langtv@gmail.com
http://fair.conf.vn/~lang



8. Lập trình OpenMP với shared-memory

- OpenMP (Open Multi-Processing) được cung cấp bởi The OpenMP Architecture Review Board (ARB) published its first API specifications,
- 10/1997: OpenMP for Fortran 1.0
- 10/1998. OpenMP for C/C++ standard
- 2000: version 2.0 of the Fortran specification

OpenMP

The OpenMP API specification for parallel programming

Home | Specifications | Community | Resources | News & Events | About | 🔍

Specifications

Home > Specifications

OpenMP 5.2 Specification

- OpenMP API 5.2 Specification – Nov 2021
- Softcover Version on Amazon
- OpenMP API Additional Definitions 2.0 – Nov 2020
- OpenMP API 5.2 Reference Guide (English) – PDF
- OpenMP API 5.2 Reference Guide (Japanese) – PDF
- OpenMP API 5.2 Supplementary Source Code
- OpenMP API 5.2 Stack Overflow

OpenMP 5.1 Specification

- OpenMP API 5.1 Specification – Nov 2020 – HTML Version
- Softcover Version on Amazon
- OpenMP API Additional Definitions 2.0 – Nov 2020
- OpenMP API 5.1 Reference Guide – PDF
- OpenMP API 5.1 Supplementary Source Code
- OpenMP API 5.1 Examples – August 2021
- OpenMP API 5.1 Stack Overflow

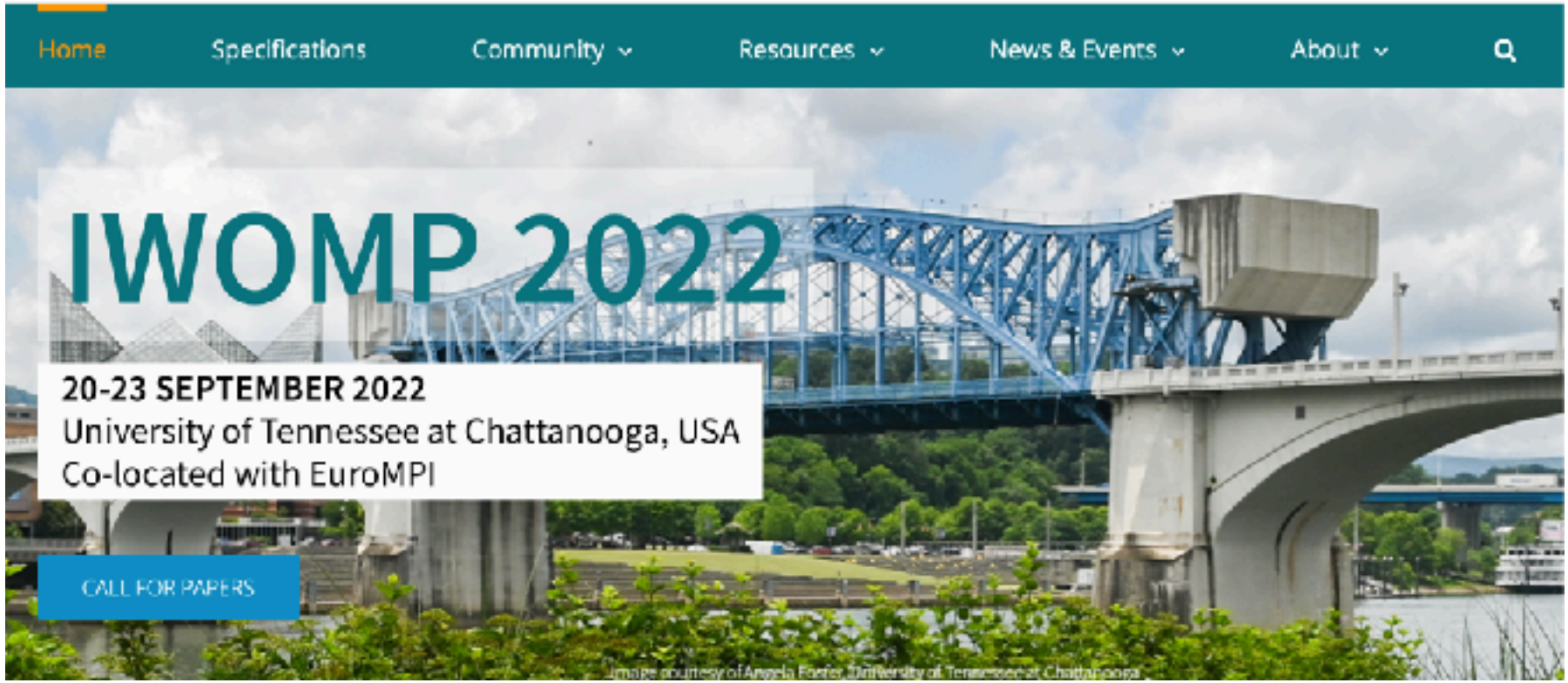
https://www.openmp.org/specifications/

- 2002: version 2.0 of the C/C++ specifications
- 2005: Version 2.5 is a combined C/C++/Fortran specification
- 5/2008: Version 3.0
- 7/2014: Version 4.0
- 11/2018: Version 5.9
- The current version is 5.2 (11/2021)

http://www.openmp.org

OpenMP

The OpenMP API specification for parallel programming



- Để install, có thể thông qua website https://releases.llvm.org để download phiên bản cần thiết.
- Hoặc install thư viện libomp dùng Homebrew ở terminal của hệ điều hành, bằng lệnh:
  - brew install libomp

- Với C/C++, có thể dùng:
  - GCC (GNU Compiler Collection)
  - Clang++
  - Intel C Compiler
  - Microsoft Visual C++
- Chẳng hạn, với macOS,
  - install GCC(vd gcc-11): brew install gcc
  - gcc-11 -o vidu vidu.c -fopenmp
  - **Lưu ý**: với gcc, đã có OpenMP để dùng

- OpenMP là dạng lập trình MultiThread
- Dùng để lập trình (Code Generation) song song trên bộ nhớ chia sẻ và multicore
- Chương trình viết trong OpenMP dùng mô hình SPMD (Single Program Multi Data)

- Các thành phần của OpenMP bao gồm:
  - Các chỉ thị biên dịch (Compiler Directives)
  - Các hàm thư viện (Runtime Library Routines)
  - Các biến môi trường (Environment Variables)

Chỉ thị parallel

- OpenMP được điều khiển thông qua các chỉ thị
- Những chỉ thị của chương trình C/C++ được bắt đầu bởi từ khóa tiền xử lý #pragma.
- Với OpenMP, chỉ thị được bắt đầu bởi #pragma omp

- Chỉ thị #parama omp parallel cho biết các câu lệnh trong khối được thực thi song song trên các thread của máy.
- Ví dụ:

```
int main(){
#pragma omp parallel
    {
        printf( "OpenMP: Hello\n" );
    }
    return 1;
}
```

- Chương trình này xuất ra 4 dòng

"OpenMP: Hello"

nếu máy đó có 4 thread

```
[Lion:openmp lang$
[Lion:openmp lang$ gcc -o first first.c -fopenmp
[Lion:openmp lang$ ./first
 OpenMP: Hello
 OpenMP: Hello
 OpenMP: Hello
 OpenMP: Hello
 Lion:openmp lang$
```

- Lưu ý:
  - Chương trình sử dụng các hàm thư viện, nên phải có `#include <omp.h>`
  - Khi biên dịch, phải chỉ định thư viện với –fopenmp. Chẳng hạn, để dịch tập tin first.c ra tập tin first khi dùng **GCC**:

```
gcc –o first first.c –fopenmp
g++ –o first first.cc –fopenmp
```

  - Hoặc dùng **Clang** phải chỉ định, chẳng hạn

```
clang –o first first.c –fopenmp
```

Chỉ thị parallel for

- Chỉ thị này chia các công việc trong vòng lặp for cho các thread
- Ví dụ:

```
int main() {
    int n;
#pragma omp parallel for
    for( n = 0; n < 12; ++n )
        printf( " %d", n );
    printf( ".\n" );
    return 1;
}
```

```
int main() {
    int n, tid;
#pragma omp parallel for
    for( n = 0; n < 12; ++n ){
        tid = omp_get_thread_num();
        printf( "(%did, %d) ", tid,
n );
    }
    printf( "\n" );
    return 1;
}
```

- Trên máy có 4 thread, với 12 câu lệnh lặp được phân chia cho mỗi thread đảm trách 3 câu lệnh.
- Câu lệnh printf giá trị 0, 1, 2 (của vòng lặp) được thực hiện bởi thread thứ I
- Câu lệnh in giá trị 3, 4, 5 bởi thread thứ II
- Câu lệnh in giá trị 6, 7, 8 bởi thread thứ III
- Câu lệnh in giá trị 9, 10, 11 bởi thread thứ IV

- Kết quả lần lượt 4 thread xuất ra 3 con số của mình cần print (4 x 3 = 12 con số)
- Kết quả của 5 lần chạy

```
[Lion:openmp lang$ ./second
 6 3 0 9 7 4 1 10 8 5 2 11.
[Lion:openmp lang$ ./second
 9 0 3 6 10 1 4 7 11 2 5 8.
[Lion:openmp lang$ ./second
 6 0 3 9 7 1 4 10 8 2 5 11.
[Lion:openmp lang$ ./second
 0 3 6 9 1 4 7 10 2 5 8 11.
[Lion:openmp lang$ ./second
 0 3 6 9 1 4 7 10 2 5 8 11.
```

- Thay 12 bởi 8, trên máy với 4 thread.
- Chương trình như trên cho kết quả (0,1: Thread I; 2,3: Thraed II; 4,5: Thread III, 6,7: Thread IV)

```
[Lion:openmp lang$ ./second
 2 0 4 6 3 1 5 7.
[Lion:openmp lang$ ./second
 4 0 2 6 5 1 3 7.
[Lion:openmp lang$ ./second
 0 4 2 6 1 5 3 7.
```

- ***Lưu ý:***
  - Với chỉ thị này, ta có thể chỉ định số thread cần thực hiện. Chẳng hạn, chỉ muốn dùng 2 thread:
    ```
    #pragma omp parallel for num_threads(2)
    ```
  - Khi đó, với ví dụ trên, các câu lệnh in 0, 1, 2, 3, 4, 5 cho thread thứ I; 6, 7, 8, 9, 10, 11 cho thread thứ II.

- ***Lưu ý:***
  - Việc khai báo sau đây là tương đương nhằm chỉ ra *n* là biến địa phương của từng thread

```
int n;
#pragma omp parallel for private(n)
for( n = 0; n < 12; ++n )
```

Và

```
#pragma omp parallel for
for( int n = 0; n < 12; ++n )
```

Ví dụ

- Khởi tạo giá trị ban đầu cho một mảng dữ liệu có N phần tử, trong đó $N \gg 1$
- Cách giải quyết:
  - Chia thành P mảng con, mỗi mảng có $\frac{N}{P}$ phần tử
  - Từ đó P tiến trình đồng thời khởi tạo các mảng con này

- Chương trình con khởi tạo của mỗi tiến trình:

```
void create( float *x,long int start,long int np ){
    long int i;
    for ( i = 0; i < np; i++ )
        x[start+i] = i*3.14159/np;
}
```

- Chương trình con phân chia cho mỗi tiến trình:

```
void subroutine( float *x,long int n ){
    int p,P;
    long int np,start;
    #pragma omp parallel private(p,P,np,start)
    {
        p = omp_get_thread_num();
        P =  omp_get_num_threads();
        np = n/P;
        start = p*np;
        if (p == P-1)
            np = n - start;
        printf( "Call sudomain() to create array %d elements from index = %d\n", np, start
        create( x,start,np );
    }
}
```

Chương trình Simple3.c++

- Chương trình chính:

```
int main(){
    long N;
    printf( "Number of elements of array: " );
    scanf( "%ld", &N );
    float *A = (float *)calloc( N, sizeof(float) );

    double t0 = omp_get_wtime();
    subroutine( A,N );
    printf( "Elapsed time for Parallel Computation: %f seconds\n", omp_get_wtime()-t0 );

    t0 = omp_get_wtime();
    create( A,0,N );
    printf( "Elapsed time for Sequence Computation: %f seconds\n", omp_get_wtime()-t0 );

    return 1;
}
```

Tính tổng đơn giản

```
1  /************************/
2  /**** SimpleSum.c++ ****/
3  /************************/
4  #include <omp.h>
5  #include <iostream>
6  #include <ctime>
7  #include <iomanip>
8  using namespace std;
9  int main(){
10     long n;
11     cout << "Number of elements: ";
12     cin >> n;
13
14     double *a;
15     a = new double[n];
16     srand( time(0) );
17     for ( int i = 0; i < n; i++ )
18         a[i] = 1959 + rand() % 63;  // a[i] in [1959,2022]
19     double s, ls;
20     cout << setprecision(15);
21
```

```
22 // Sequence
23     double wt = omp_get_wtime();
24     ls = 0.0;
25     for ( int i = 0; i < n; i++ )
26         ls += a[i];
27     cout << "Elapsed time for Sequence computing "
28     << omp_get_wtime() - wt << " seconds\n";
29     cout << "Sum = " << ls << endl;
30
31 // Parallel
32     wt = omp_get_wtime();
33     ls = 0.0;
34     s = 0.0;
35 #pragma omp parallel private( ls )
36     {
37 #pragma omp for
38         for ( int i = 0; i < n; i++ )
39             ls += a[i];
40         s += ls;
41     }
42     cout << "Elapsed time for Parallel computing "
43     << omp_get_wtime() - wt << " seconds\n";
44     cout << "Sum = " << s << endl;
45
46     return 1;
47 }
```

- Với thuật giải trong phần paradigm về chia và chế ngự

Algorithm: Tính tổng chia và chế ngự

Input: A(1..n)
Output: B(1)
1. **For** i=1 **to** n/logn **doPar**
2. B(i) = 0
3. **For** j=1 **to** logn **do**
4. B(i) = B(i)+ A(ik+j-logn)
5. **EndFor**
6. **EndPar**

Algorithm: Nhị phân tính tổng B(1..r)

Input: B(1..r)
Output: B(1)
1. p = r/2
2. While p > 0 do
3. **For** i=1 **to** p **doPar**
4. B(i) = B(2i-1) + B(2i)
5. **EndPar**
6. p = p/2
7. **EndWhile**

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

int main(){

    float *A, *B, *S;
    long int N, i, j;
    int R, K, p;

    printf("No. of elements:");
    scanf( "%ld", &N );
    K = log2(N);
    R = N/K;

    A = (float *)calloc( N, sizeof(float) );
    for( i = 0; i < N; i++ )
        A[i] = i+1;
    B = (float *)calloc( R, sizeof(float) );
    double wt = omp_get_wtime();

#pragma omp parallel for
    for( i = 0; i < R; i++ ){
        B[i] = 0.0;
        for(int j = 0; j < K;j++)
            B[i]+= A[(i+1)*K+j-K];
    }
    p = R/2;
    while( p > 0 ){
#pragma omp for
        for( i = 0; i < p; i++ )
            B[i]= B[2*i]+B[2*i+1];
        p /= 2;
    }
    printf( "Elapsed Time: %lf sec\n",
            omp_get_wtime()-wt );
    printf( "Sum of Sequence: %lf\n", B[0] );
    return 1;
}
```

- Ví dụ: nhân ma trận $c = a \times b$, với $c_{ij} = \sum_{k=1}^{l} a_{ik} b_{kj}$
- Chương trình tuần tự:

```
int mult( double **a, double **b, double**c, int n, int m, int l ){
    double wt;
    wt = omp_get_wtime();
    int i, j, k;
    for ( i = 0; i < n; i++ )
        for ( j = 0; j < m; j++ ){
            c[i][j] = 0.0;
            for ( k = 0; k < l; k++ )
                c[i][j] += a[i][k]*b[k][j];
        }
    printf( "Elapsed time: %lf seconds\n", omp_get_wtime() - wt );
}
```

- Chương trình song song

```
int par_mult( double **a, double **b, double**c, int n, int m, int l ){
      double wt;
      wt = omp_get_wtime();
      int i, j, k;
#pragma omp parallel private( i, j, k )
      {
            if ( omp_get_thread_num() == 0 )
                  printf( "Num of threads %d\n", omp_get_num_threads() );
#pragma omp for
            for ( i = 0; i < n; i++ )
                  for ( j = 0; j < m; j++ ){
                        c[i][j] = 0.0;
                        for ( k = 0; k < l; k++ )
                              c[i][j] += a[i][k]*b[k][j];
            }
      }
      printf( "Elapsed time: %lf seconds\n", omp_get_wtime() - wt );
}
```

- Kết quả tính với C = A x B trên máy gpu.vast.vn với 24 processor

Matrix A		Matrix B		Par Time (s)	Seq Time (s)
800	800	800	800	0,49	3,25
800	800	800	8000	5,39	60,29
8000	800	800	8000	51,56	598,01

Chỉ thị critical

- Xem xét ví dụ tính tổng của một dãy $\{a_n\}$ gồm n số thực.
  - Khi đó, từng thread tính tổng riêng (khai báo là ls).
  - Thread chủ làm nhiệm vụ tính các tổng (khai báo là s) từ các tổng riêng này rồi xuất ra mà hình.
- Khi đó, việc tính *s* = *s* + *ls* được ngăn ngừa để các thread không tính.
  - Việc này được thể hiện qua chỉ thị critical.

- Ví dụ

```
int main(){
    long int n = 10000;
    double ls = 0.0, s = 0.0;
    long int i;
    double *a = (double *)calloc( n, sizeof(double) );
    for ( i = 0; i < n; i++ )
        a[i] = i+1;
#pragma omp parallel private(i,ls)
    {
#pragma omp for
        for ( i = 0; i < n; i++ )
            ls += a[i];
#pragma omp critical
        s += ls;
    }
#pragma omp end parallel
    printf( "Sum of sequence a: %lf\n", s );
    return 1;
}
```

Mệnh đề reduction

- Với cách tính tổng dãy số $\{a_n\}$ như trên, có thể dùng reduction để có kết quả bằng

```
    s = 0.0;
#pragma omp parallel for reduction(+:s)
    for ( i = 0; i < n; i++ )
        s += a[i];
    printf( "Sum of sequence a: %lf\n", s );
```

Mệnh đề private

- Nhằm chỉ ra các biến liệt kê trong danh sách của mệnh đề này là biến địa phương của thread.
- Xem các ví dụ trên

Một số hàm thư viện

- double omp_get_wtime(): trả về thời gian hiện tại, tính ra giây
- int omp_get_thread_num(): để nhận biết thread nào thực hiện
- int omp_num_threads(): cho biết tổng số thread
- int omp_get_num_procs(): trả về số processors hiệu lực vào thời điểm hàm này được gọi.

- int omp_get_max_threads(): trả về số thread tối đa được sử dụng mà không có mệnh đề num_threads trong chỉ thị #pragma.
- int omp_get_num_threads(): trả về số thread được sử dụng trong vùng tác động của lệnh thi hành song song mà nó được gọi.

- Ví dụ: chỉ chương trình chạy trên thread #0 mới xuất ra số lượng thread đang sử dụng dưới một chỉ thị song song

```
if ( omp_get_thread_num() == 0 )
   printf( "Num of threads %d\n", omp_get_num_threads() )
```

Chỉ thị Section

- Khi phân rã theo chức năng, có thể có những đoạn chương trình chạy ở những tiến trình khác nhau.
- Chẳng hạn,
  - Work0: thực thi ở tất cả các tiến trình và song song với các Work1, Work2, Work4, Work5
  - Work1: thực thi trong 1 tiến trình
  - Work2, Work3: thực thi tuần tự trong cùng 1 tiến trình, nhưng song song với Work4
  - Work4: thực thi song song với Work2
  - Work5: thực thi ở tiến trình chủ (tiến trình #0)

```c
int main() {
#pragma omp parallel
    {
        if ( omp_get_thread_num() == 0 )
            printf( "Work0 would be run by %d threads.\n", omp_get_num_threads() );

#pragma omp sections
        {
            printf( "Work1 is run only once on thread#%d\n", omp_get_thread_num() );
#pragma omp section
            {
                printf( "Work2 is run only once on thread#%d\n", omp_get_thread_num() );
                printf( "Work3 is run only once on thread#%d\n", omp_get_thread_num() );
            }
#pragma omp section
                printf( "Work4 is run only once on thread#%d\n", omp_get_thread_num() );
        }
    }
    printf( "Work5 is run only once on thread#%d\n", omp_get_thread_num() );
    return 1;
}
```

- Kết quả

```
[iami.langtv@gpu openmp]$ ./sixth
Work1 is run only once on thread#7
Work2 is run only once on thread#18
Work3 is run only once on thread#18
Work0 would be run by 24 threads.
Work4 is run only once on thread#20
Work5 is run only once on thread#0
[iami.langtv@gpu openmp]$
```

```
[iami.langtv@gpu openmp]$ ./sixth
Work0 would be run by 24 threads.
Work2 is run only once on thread#18
Work3 is run only once on thread#18
Work4 is run only once on thread#17
Work1 is run only once on thread#6
Work5 is run only once on thread#0
[iami.langtv@gpu openmp]$
```

```
[iami.langtv@gpu openmp]$ ./sixth
Work0 would be run by 24 threads.
Work4 is run only once on thread#21
Work1 is run only once on thread#12
Work2 is run only once on thread#20
Work3 is run only once on thread#20
Work5 is run only once on thread#0
[iami.langtv@gpu openmp]$
```

- Ví dụ, có 2 section, thực thi 2 công việc đồng thời

```
    double t0 = omp_get_wtime();

#pragma omp parallel sections
    {
#pragma omp section
    par_mult( a, b, c, RA, CB, CA );
#pragma omp section
    mult( a, b, c, RA, CB, CA );
    }
    cout << "Total elapsed time: " << setw(10) << omp_get_wtime()
        - t0 << " seconds\n";
```

Ví dụ tính tổng song song và tuần tự cùng nhau

```
1  /**********************/
2  /**** SectionSum.c ****/
3  /**********************/
4  #include <omp.h>
5  #include <time.h>
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  int main(){
10     long n;
11     printf( "Number of elements: " );
12     scanf( "%ld", &n );
13
14     double *a = (double *)calloc( n, sizeof(double) );
15     srand( time(0) );
16     for ( int i = 0; i < n; i++ )
17         a[i] = 1959 + rand() % 63;  // a[i] in [1959,2022]
18     double s1, s2, ls;
19
```

```
21 #pragma omp parallel sections
22 {
23     #pragma omp section
24     {
25 // Sequence
26     double w1 = omp_get_wtime();
27     s1 = 0.0;
28     for ( int i = 0; i < n; i++ )
29         s1 += a[i];
30     printf( "Elapsed time for Sequence computing %f seconds\n"
31     , omp_get_wtime() - w1 );
32     printf( "Sum = %lf\n", s1 );
33     }
34
```

```
35      #pragma omp section
36      {
37  // Parallel
38      double w2 = omp_get_wtime();
39      ls = 0.0;
40      s2 = 0.0;
41      #pragma omp parallel private( ls )
42      {
43      #pragma omp for
44          for ( int i = 0; i < n; i++ )
45              ls += a[i];
46      #pragma omp critical
47          s2 += ls;
48      }
49      printf( "Elapsed time for Parallel computing %f seconds\n"
50      , omp_get_wtime() - w2 );
51      printf( "Sum = %lf\n", s2 );
52      }
53  }
54      printf( "Total Elapsed time %f seconds\n", omp_get_wtime() - w0 );
55      free(a);
56      return 1;
57  }
```