Report of OpenMP

Speaker : Guo-Zhen Wu

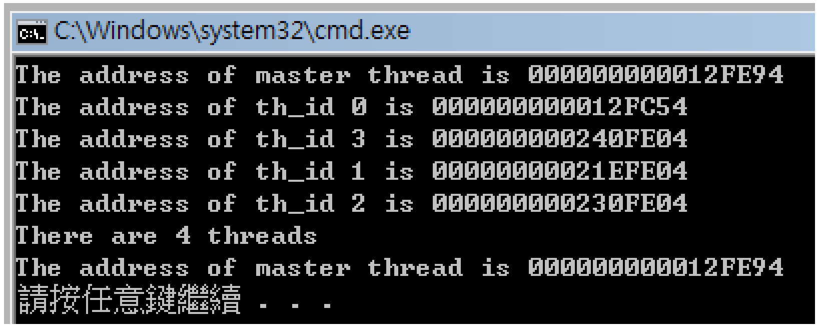
Date : 2008/12/28

Question 5: What happens if number of threads is larger than number of cores of host machine?

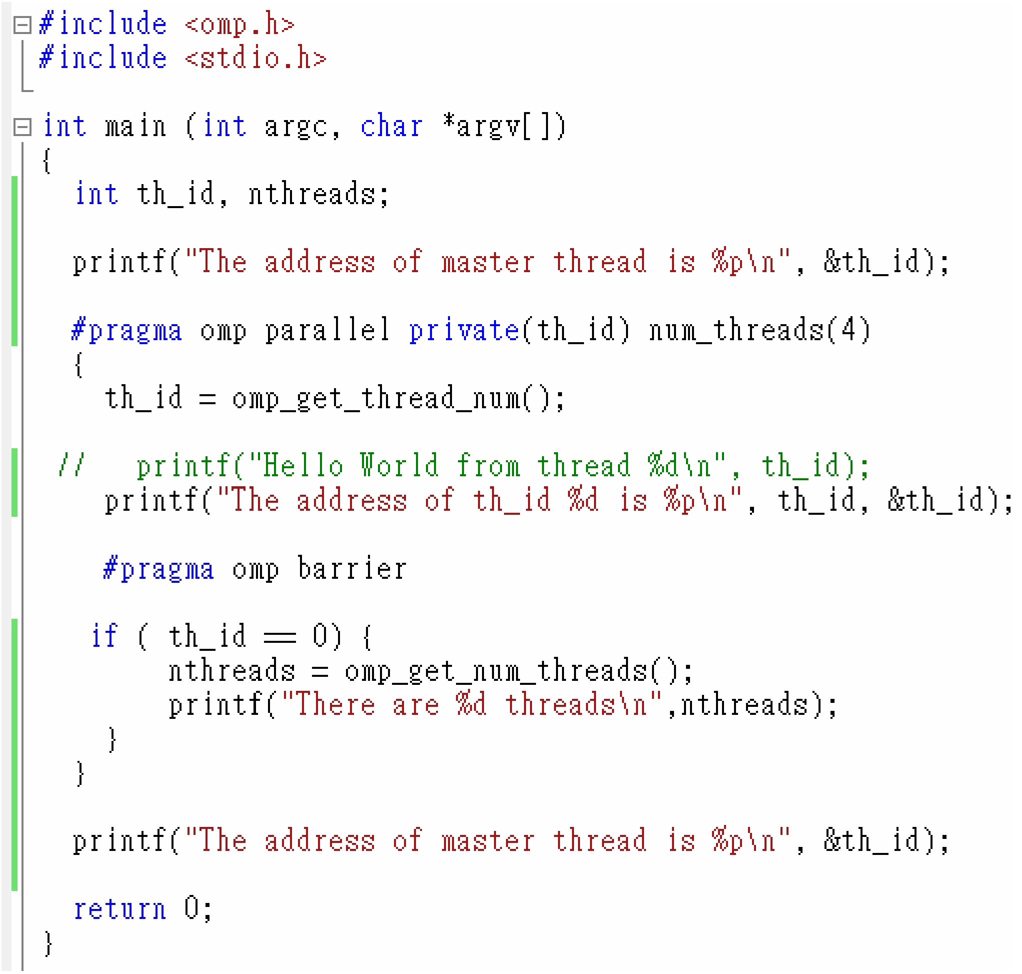
Exercise 1: modify code of hello.cto show “every thread has its own private variable ***th_id***”, that is, shows th_idhas 5 copies.

Ans : we can use printf to tell each thread print its own th_id

printf(“th_id : %p”,&th_id) ;

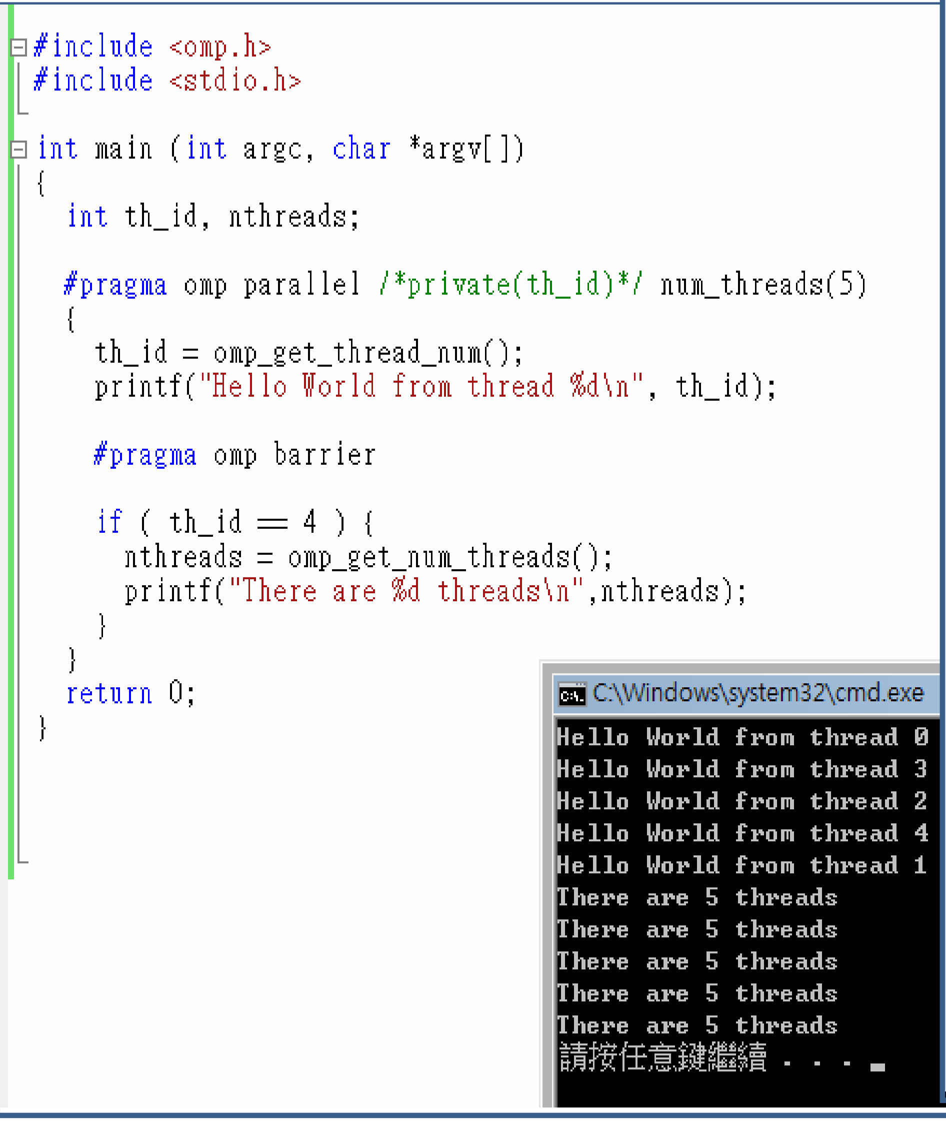


```
The address of master thread is 000000000012FE94
The address of th_id 0 is 000000000012FC54
The address of th_id 3 is 000000000240FE04
The address of th_id 1 is 000000000021EFE04
The address of th_id 2 is 000000000230FE04
There are 4 threads
The address of master thread is 000000000012FE94
請按任意鍵繼續 . . .
```



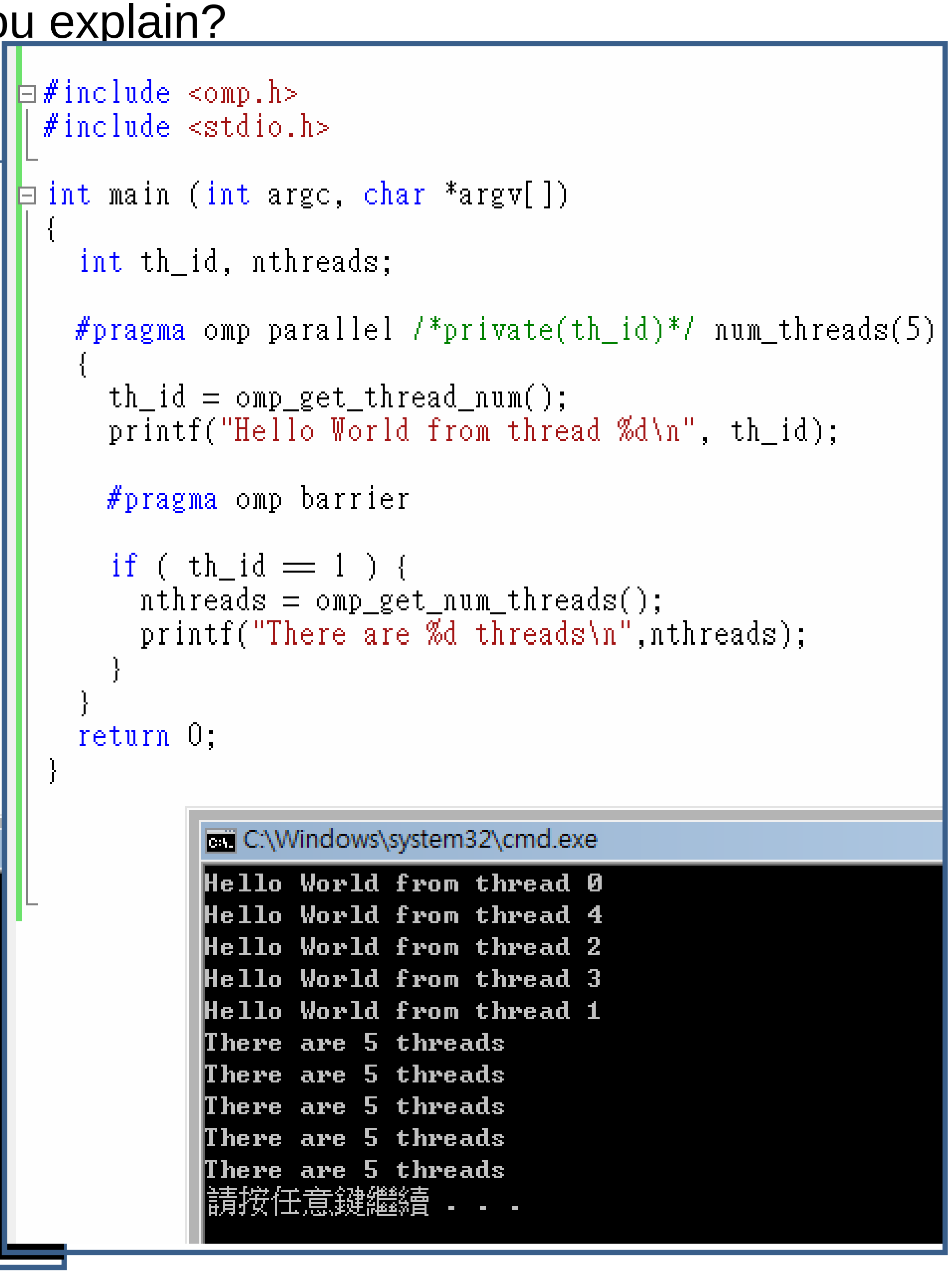
```
#include <omp.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
  int th_id, nthreads;

  printf("The address of master thread is %p\n", &th_id);

  #pragma omp parallel private(th_id) num_threads(4)
  {
    th_id = omp_get_thread_num();

 //   printf("Hello World from thread %d\n", th_id);
    printf("The address of th_id %d is %p\n", th_id, &th_id);

    #pragma omp barrier

   if ( th_id == 0) {
        nthreads = omp_get_num_threads();
        printf("There are %d threads\n",nthreads);
    }
  }

  printf("The address of master thread is %p\n", &th_id);

  return 0;
}
```

Exercise 2: modify code of hello.c, remove clause “private (th_id)”in #pragmadirective, what happens? Can you explain?



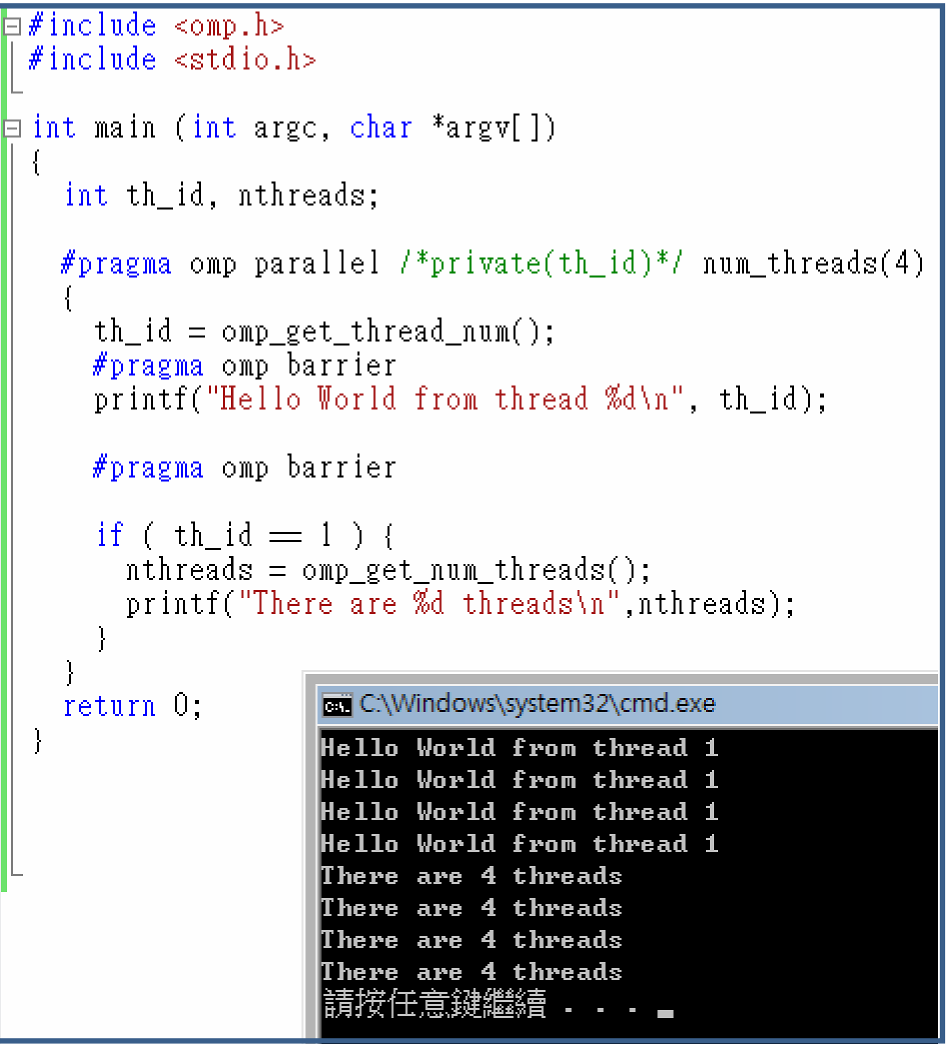
```
#include <omp.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
  int th_id, nthreads;

  #pragma omp parallel /*private(th_id)*/ num_threads(5)
  {
    th_id = omp_get_thread_num();
    printf("Hello World from thread %d\n", th_id);

    #pragma omp barrier

    if ( th_id == 4 ) {
      nthreads = omp_get_num_threads();
      printf("There are %d threads\n",nthreads);
    }
  }
  return 0;
}
```

```
Hello World from thread 0
Hello World from thread 3
Hello World from thread 2
Hello World from thread 4
Hello World from thread 1
There are 5 threads
There are 5 threads
There are 5 threads
There are 5 threads
There are 5 threads
請按任意鍵繼續 . . .
```



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

int main (int argc, char *argv[])
{
  int th_id, nthreads;

  #pragma omp parallel /*private(th_id)*/ num_threads(5)
  {
    th_id = omp_get_thread_num();
    printf("Hello World from thread %d\n", th_id);

    #pragma omp barrier

    if ( th_id == 1 ) {
      nthreads = omp_get_num_threads();
      printf("There are %d threads\n",nthreads);
    }
  }
  return 0;
}
```

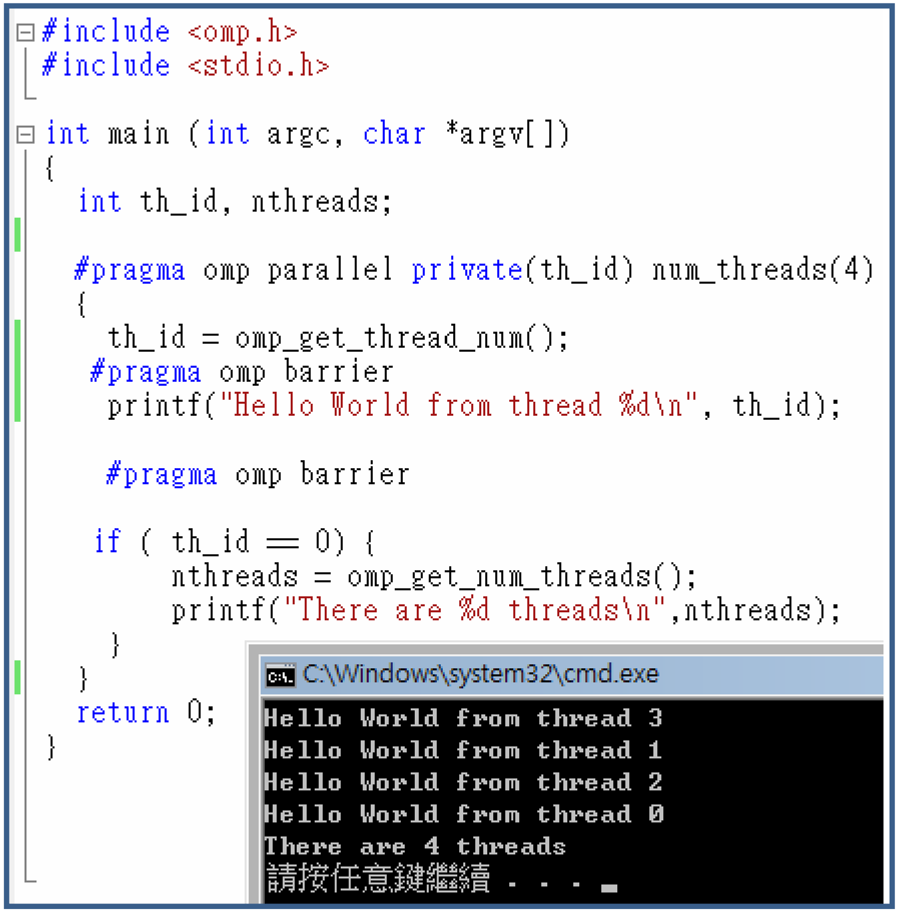
```
Hello World from thread 0
Hello World from thread 4
Hello World from thread 2
Hello World from thread 3
Hello World from thread 1
There are 5 threads
There are 5 threads
There are 5 threads
There are 5 threads
There are 5 threads
請按任意鍵繼續 . . .
```



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

int main (int argc, char *argv[])
{
  int th_id, nthreads;

  #pragma omp parallel /*private(th_id)*/ num_threads(4)
  {
    th_id = omp_get_thread_num();
    #pragma omp barrier
    printf("Hello World from thread %d\n", th_id);

    #pragma omp barrier

    if ( th_id == 1 ) {
      nthreads = omp_get_num_threads();
      printf("There are %d threads\n",nthreads);
    }
  }
  return 0;
}
```

C:\Windows\system32\cmd.exe

```
Hello World from thread 1
Hello World from thread 1
Hello World from thread 1
Hello World from thread 1
There are 4 threads
There are 4 threads
There are 4 threads
There are 4 threads
請按任意鍵繼續 . . .
```

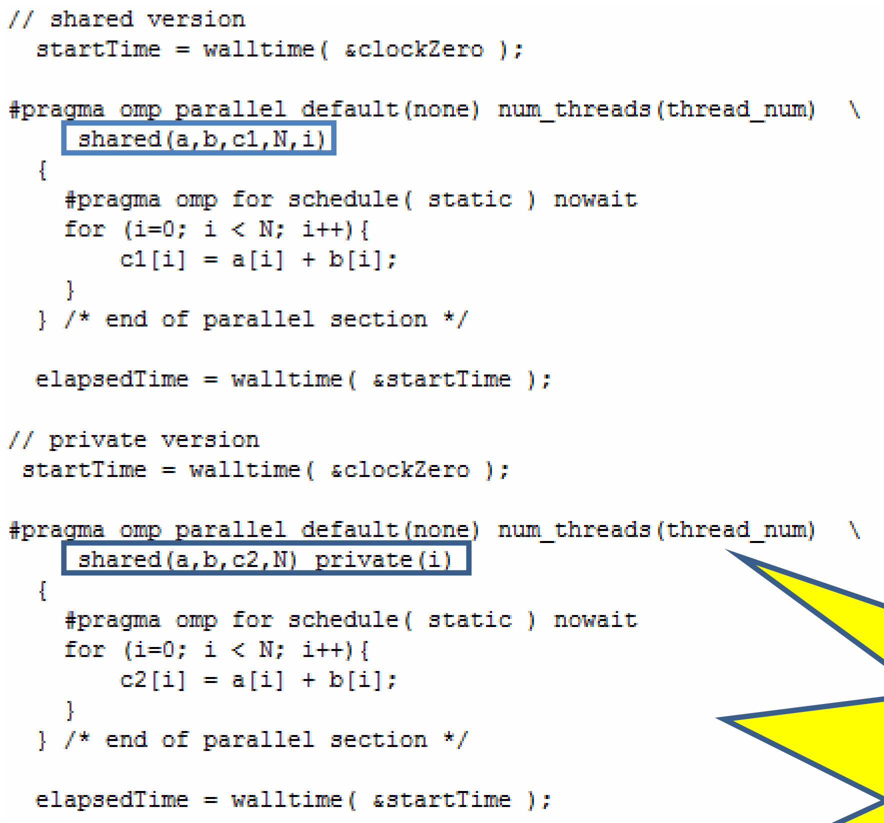


```
#include <omp.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
  int th_id, nthreads;

  #pragma omp parallel private(th_id) num_threads(4)
  {
    th_id = omp_get_thread_num();
   #pragma omp barrier
    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 0;
}
```

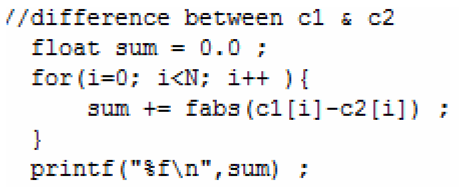
C:\Windows\system32\cmd.exe

```
Hello World from thread 3
Hello World from thread 1
Hello World from thread 2
Hello World from thread 0
There are 4 threads
請按任意鍵繼續 . . .
```

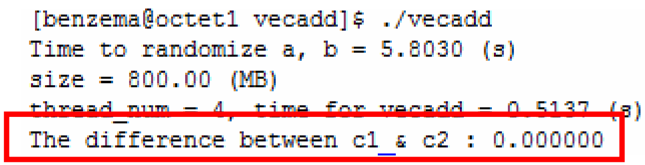
Question 6: Why index ***i*** must be private variable and ***a,b,c,N*** can be shared variable? What happens if we change ***i*** to shared variable? What happens if we change ***a,b,c,N*** to private variable?



```
// shared version
  startTime = walltime( &clockZero );

#pragma omp parallel default(none) num_threads(thread_num)  \
     shared(a,b,c1,N,i)
  {
    #pragma omp for schedule( static ) nowait
    for (i=0; i < N; i++){
        c1[i] = a[i] + b[i];
    }
  } /* end of parallel section */

  elapsedTime = walltime( &startTime );

// private version
 startTime = walltime( &clockZero );

#pragma omp parallel default(none) num_threads(thread_num)  \
     shared(a,b,c2,N) private(i)
  {
    #pragma omp for schedule( static ) nowait
    for (i=0; i < N; i++){
        c2[i] = a[i] + b[i];
    }
  } /* end of parallel section */

  elapsedTime = walltime( &startTime );
```

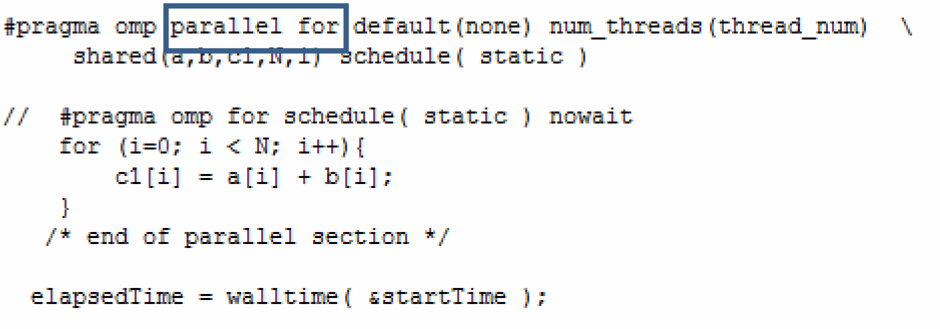


```
  //difference between c1 & c2
    float sum = 0.0 ;
    for(i=0; i<N; i++ ){
        sum += fabs(c1[i]-c2[i]) ;
    }
    printf("%f\n",sum) ;
```

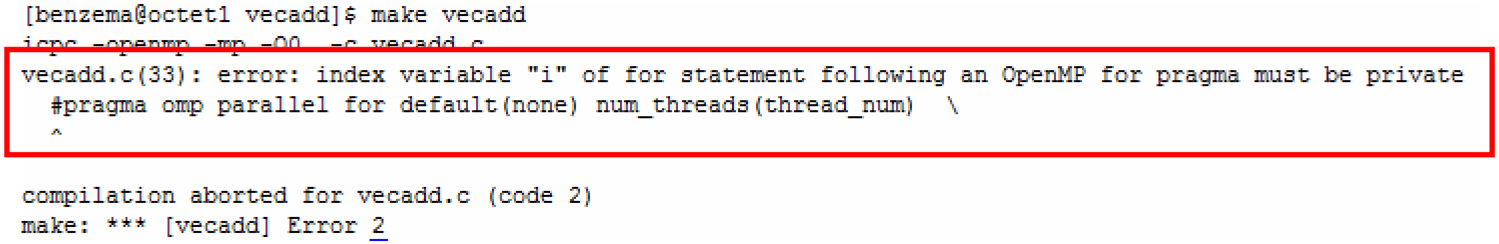


```
[benzema@octet1 vecadd]$ ./vecadd
Time to randomize a, b = 5.8030 (s)
size = 800.00 (MB)
thread_num = 4, time for vecadd = 0.5137 (s)
The difference between c1 & c2 : 0.000000
```

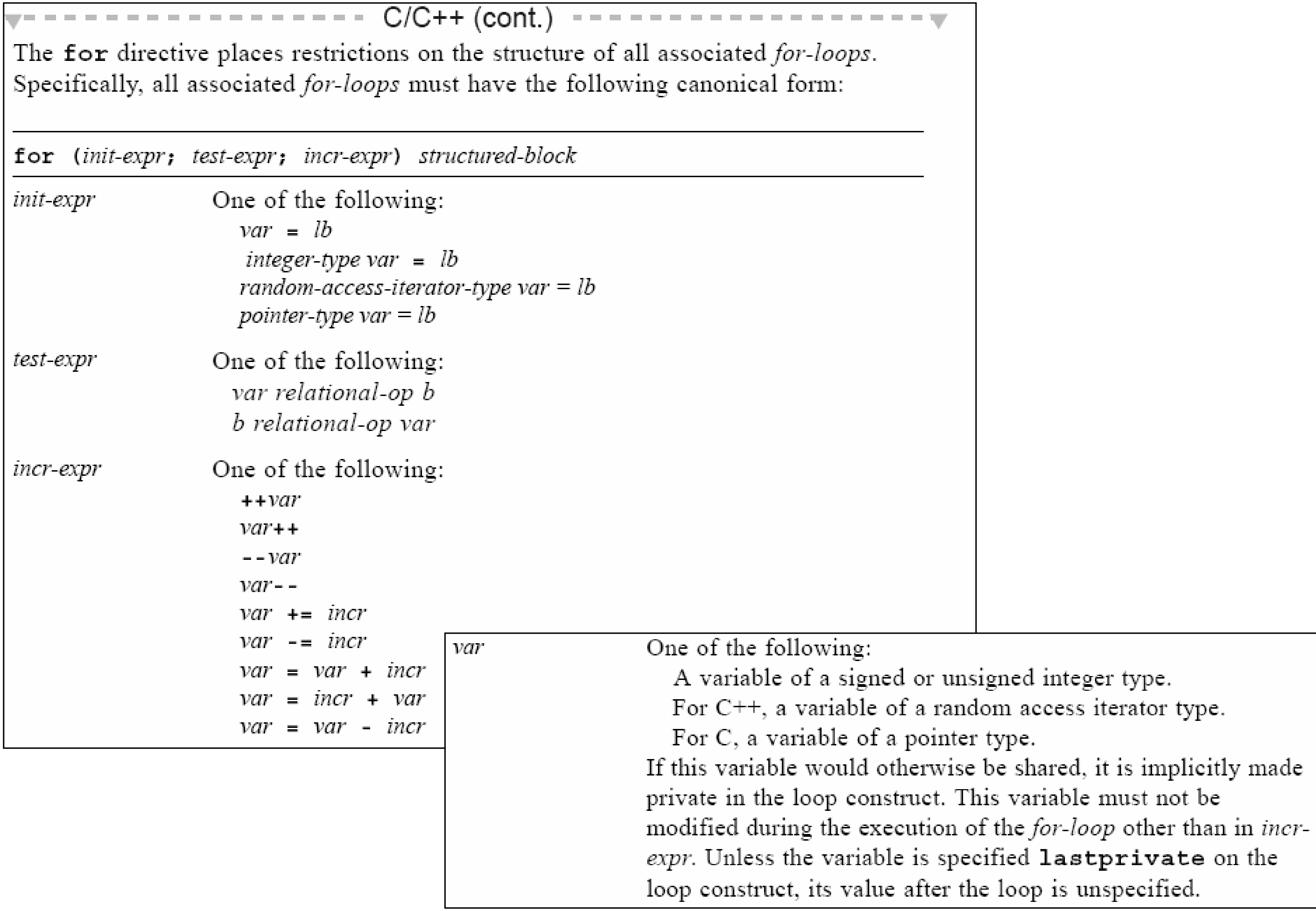
The result are the same !!!
What’s happened ?



```
#pragma omp parallel for default(none) num_threads(thread_num)   \
     shared(a,b,c1,N,i) schedule( static )

//  #pragma omp for schedule( static ) nowait
    for (i=0; i < N; i++){
        c1[i] = a[i] + b[i];
    }
   /* end of parallel section */

  elapsedTime = walltime( &startTime );
```



```
[benzema@octet1 vecadd]$ make vecadd
icpc -openmp -mp -O0  -c vecadd.c
vecadd.c(33): error: index variable "i" of for statement following an OpenMP for pragma must be private
  #pragma omp parallel for default(none) num_threads(thread_num)   \
  ^

compilation aborted for vecadd.c (code 2)
make: *** [vecadd] Error 2
```



C/C++ (cont.)

The **for** directive places restrictions on the structure of all associated *for-loops*. Specifically, all associated *for-loops* must have the following canonical form:

```
for (init-expr; test-expr; incr-expr) structured-block
```

init-expr One of the following:

var = *lb*
integer-type var = *lb*
random-access-iterator-type var = *lb*
pointer-type var = *lb*

test-expr One of the following:

var relational-op b
b relational-op var

incr-expr One of the following:

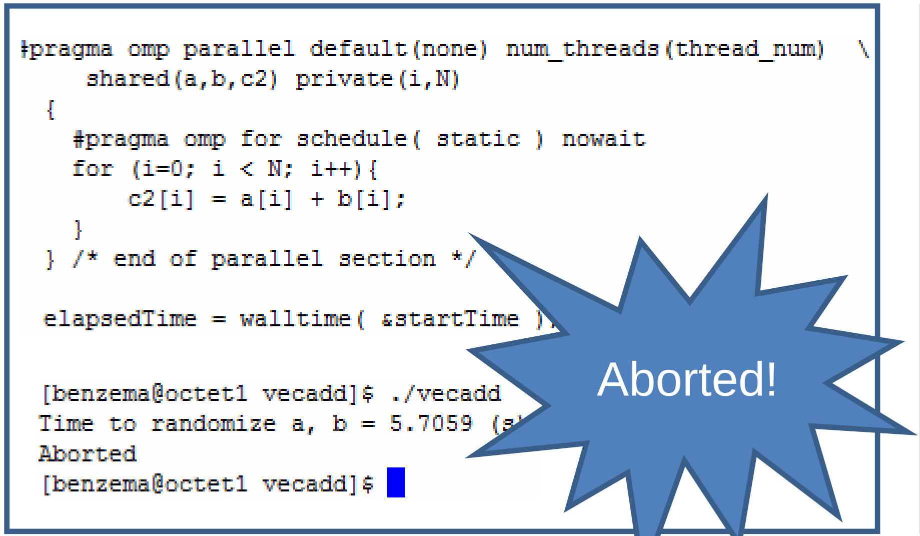
++*var*
var++
--*var*
var--
var += *incr*
var -= *incr*
var = *var* + *incr*
var = *incr* + *var*
var = *var* - *incr*

var One of the following:

A variable of a signed or unsigned integer type.
For C++, a variable of a random access iterator type.
For C, a variable of a pointer type.

If this variable would otherwise be shared, it is implicitly made private in the loop construct. This variable must not be modified during the execution of the *for-loop* other than in *incr-expr*. Unless the variable is specified **lastprivate** on the loop construct, its value after the loop is unspecified.

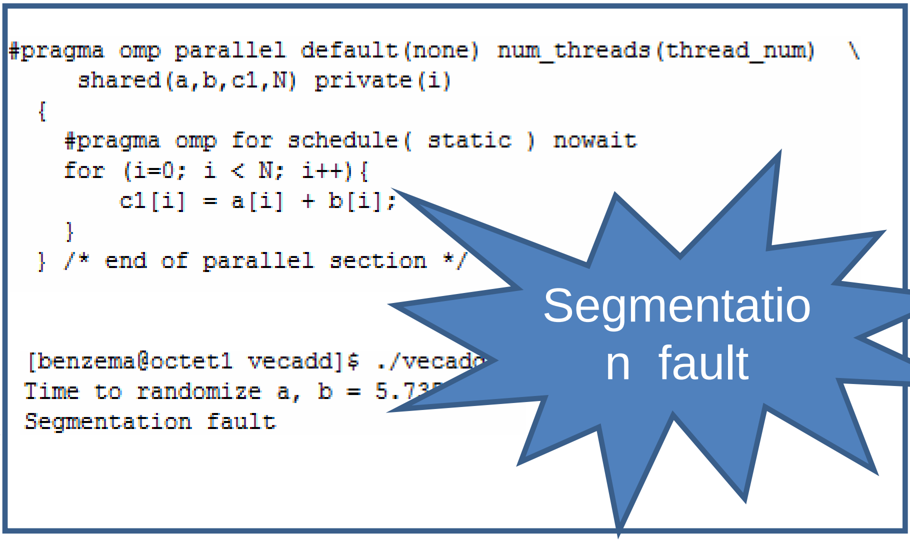
Change N from shared to private



```
#pragma omp parallel default(none) num_threads(thread_num)  \
     shared(a,b,c2) private(i,N)
  {
    #pragma omp for schedule( static ) nowait
    for (i=0; i < N; i++){
        c2[i] = a[i] + b[i];
    }
  } /* end of parallel section */

  elapsedTime = walltime( &startTime )

  [benzema@octet1 vecadd]$ ./vecadd
  Time to randomize a, b = 5.7059 (s
  Aborted
  [benzema@octet1 vecadd]$
```

Aborted!

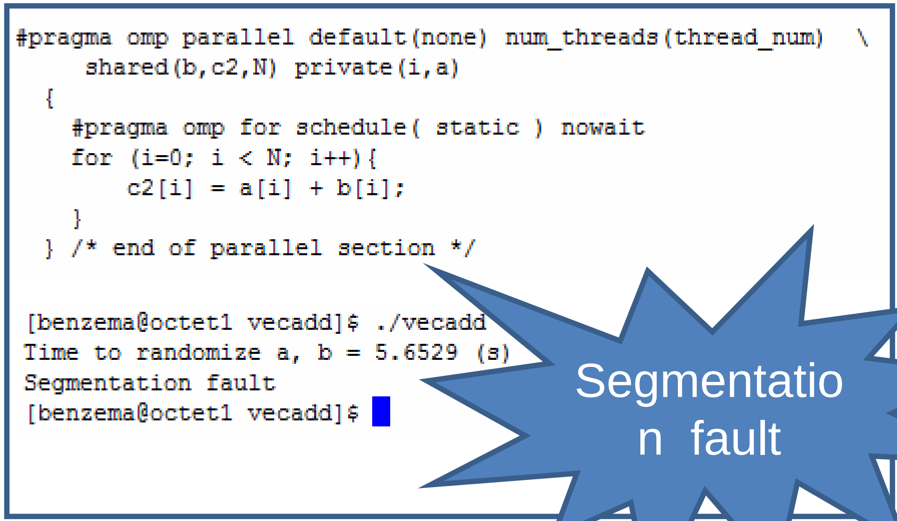
Change b from shared to private



```
#pragma omp parallel default(none) num_threads(thread_num)  \
     shared(a,b,c1,N) private(i)
  {
    #pragma omp for schedule( static ) nowait
    for (i=0; i < N; i++){
        c1[i] = a[i] + b[i];
    }
  } /* end of parallel section */


 [benzema@octet1 vecadd]$ ./vecad
 Time to randomize a, b = 5.73
 Segmentation fault
```

Segmentatio n fault

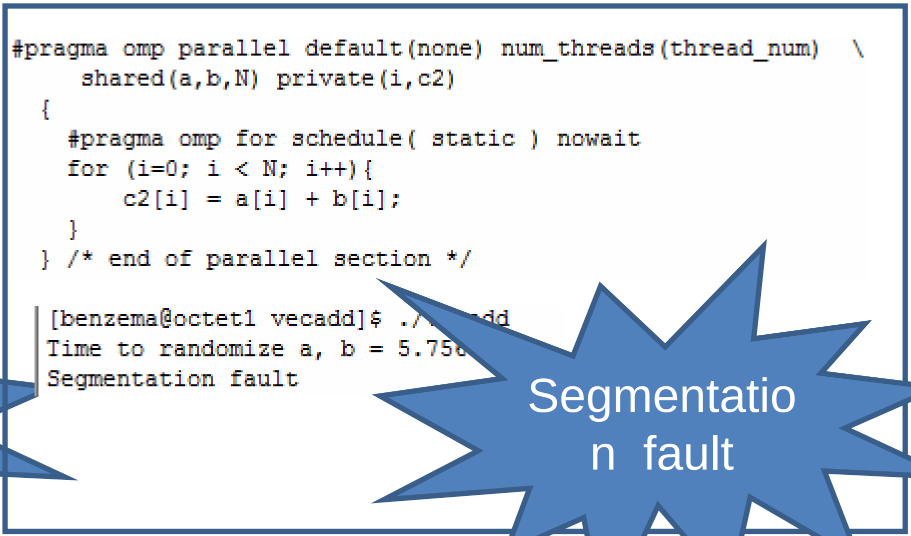
Change a from shared to private



```
#pragma omp parallel default(none) num_threads(thread_num)  \
     shared(b,c2,N) private(i,a)
  {
    #pragma omp for schedule( static ) nowait
    for (i=0; i < N; i++){
        c2[i] = a[i] + b[i];
    }
  } /* end of parallel section */

 [benzema@octet1 vecadd]$ ./vecadd
 Time to randomize a, b = 5.6529 (s)
 Segmentation fault
 [benzema@octet1 vecadd]$
```

Segmentatio n fault

Change c from shared to private



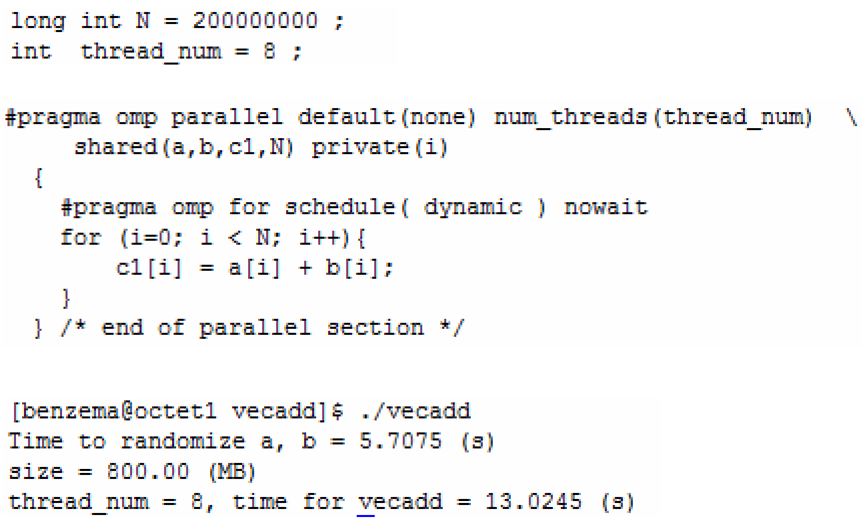
```
#pragma omp parallel default(none) num_threads(thread_num)  \
     shared(a,b,N) private(i,c2)
  {
    #pragma omp for schedule( static ) nowait
    for (i=0; i < N; i++){
        c2[i] = a[i] + b[i];
    }
  } /* end of parallel section */

 [benzema@octet1 vecadd]$ ./    dd
 Time to randomize a, b = 5.75
 Segmentation fault
```

Segmentatio n fault

Number of thread	Cost time (s)
1	1.5362
2	0.8610
4	0.5586
8	0.4852
16	0.6037
32	0.7258
64	0.8244

Question 7: the limitation of performance improvement is 3, why? Can you use different configuration of schedule clause to improve this number?

Dynamic



```
long int N = 200000000 ;
int  thread_num = 8 ;

#pragma omp parallel default(none) num_threads(thread_num)   \
     shared(a,b,c1,N) private(i)
  {
    #pragma omp for schedule( dynamic ) nowait
    for (i=0; i < N; i++){
        c1[i] = a[i] + b[i];
    }
  } /* end of parallel section */
```

```
[benzema@octet1 vecadd]$ ./vecadd
Time to randomize a, b = 5.7075 (s)
size = 800.00 (MB)
thread_num = 8, time for vecadd = 13.0245 (s)
```

$$\frac{T(Single)}{T(8-core)} = \frac{1.5362}{0.4852} = 3.166$$

dynamic takes 13.024 (s) ! Whereas, static needs only 0.4852(s) .

Number of thread = 8

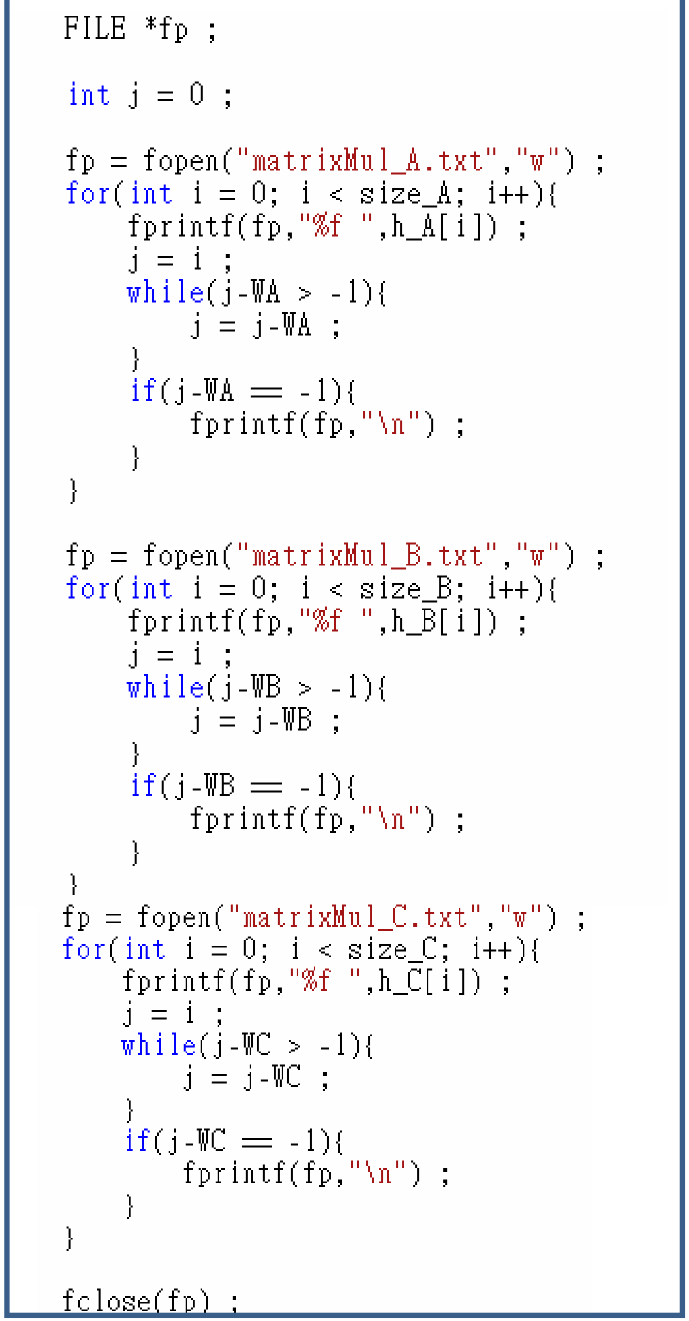
Number of chunk	Cost time (s)
2	2.1173
8	1.8999
32	1.5697
128	1.4190
512	0.6780
2048	0.5562
8196	0.4944
32784	0.4891
131136	0.4937
524544	0.4881

Question 8: we have three for-loop, one is for “***i***”, one is for “***j***” and last one is for “***k***”, which one is parallelized by OpenMP directive?

Question 9: explain why variable ***i, j, k, sum, a, b*** are declared as *private*? Can we move some of them to *shared* clause?

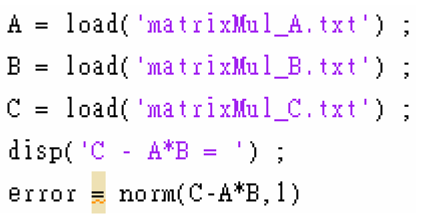
Private->Shared	i	j	k	sum	a	b
Result compare with original	The same	Error is larger than 10^13	System error!	Error is larger than 10^7	Error is larger than 10^5	Error is larger than 10^5
Can we move it to shared clause	Yes	No	No	No	No	No

Exercise 3: verify subroutine ***matrixMul_parallel***



```
FILE *fp ;

int j = 0 ;

fp = fopen("matrixMul_A.txt","w") ;
for( int i = 0; i < size_A; i++){
    fprintf(fp,"%f ",h_A[i]) ;
    j = i ;
    while(j-WA > -1){
        j = j-WA ;
    }
    if(j-WA == -1){
        fprintf(fp,"\n") ;
    }
}

fp = fopen("matrixMul_B.txt","w") ;
for( int i = 0; i < size_B; i++){
    fprintf(fp,"%f ",h_B[i]) ;
    j = i ;
    while(j-WB > -1){
        j = j-WB ;
    }
    if(j-WB == -1){
        fprintf(fp,"\n") ;
    }
}
fp = fopen("matrixMul_C.txt","w") ;
for( int i = 0; i < size_C; i++){
    fprintf(fp,"%f ",h_C[i]) ;
    j = i ;
    while(j-WC > -1){
        j = j-WC ;
    }
    if(j-WC == -1){
        fprintf(fp,"\n") ;
    }
}

fclose(fp) ;
```

Matlab code

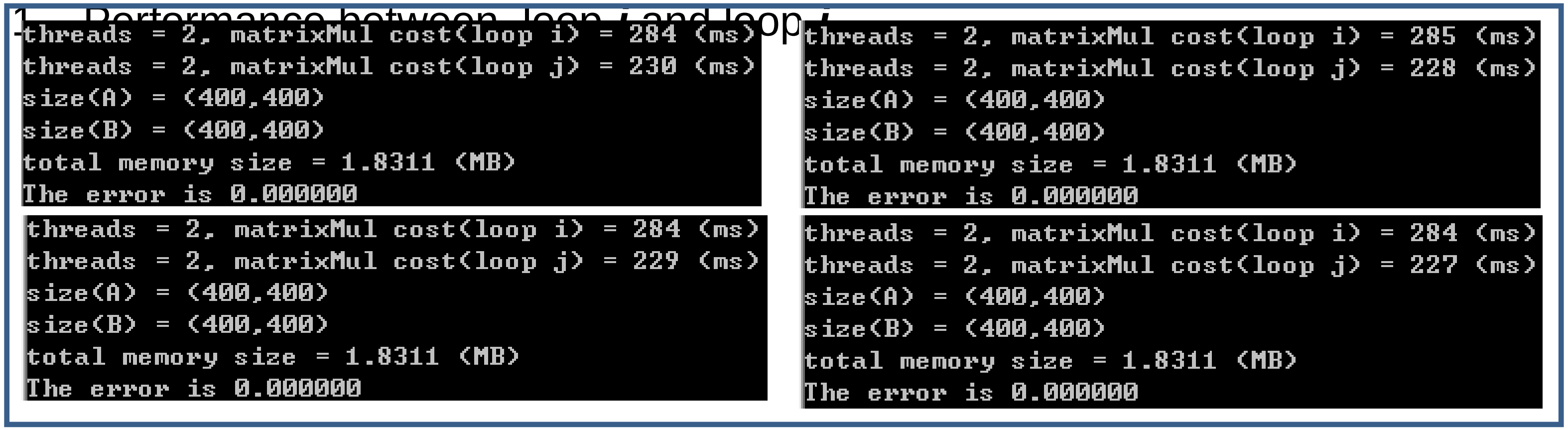


```
A = load('matrixMul_A.txt') ;
B = load('matrixMul_B.txt') ;
C = load('matrixMul_C.txt') ;
disp('C - A*B = ') ;
error = norm(C-A*B,1)
```

BLOCK_SIZE	WA=HA=WB	Error
1	25xBLOCK_SIZE	3.4261e-005
2	25xBLOCK_SIZE	1.1290e-004
4	25xBLOCK_SIZE	5.0797e-004
8	25xBLOCK_SIZE	0.0012
16	25xBLOCK_SIZE	0.0037
32	25xBLOCK_SIZE	0.0115
64	25xBLOCK_SIZE	0.0316

Exercise 4: verify following subroutine *matrix_parallel*, which parallelizes loop-***j*** , not loop-***i.***

1. Performance between loop *i* and loop *j*



```
threads = 2, matrixMul cost(loop i) = 284 (ms)
threads = 2, matrixMul cost(loop j) = 230 (ms)
size(A) = (400,400)
size(B) = (400,400)
total memory size = 1.8311 (MB)
The error is 0.000000
```

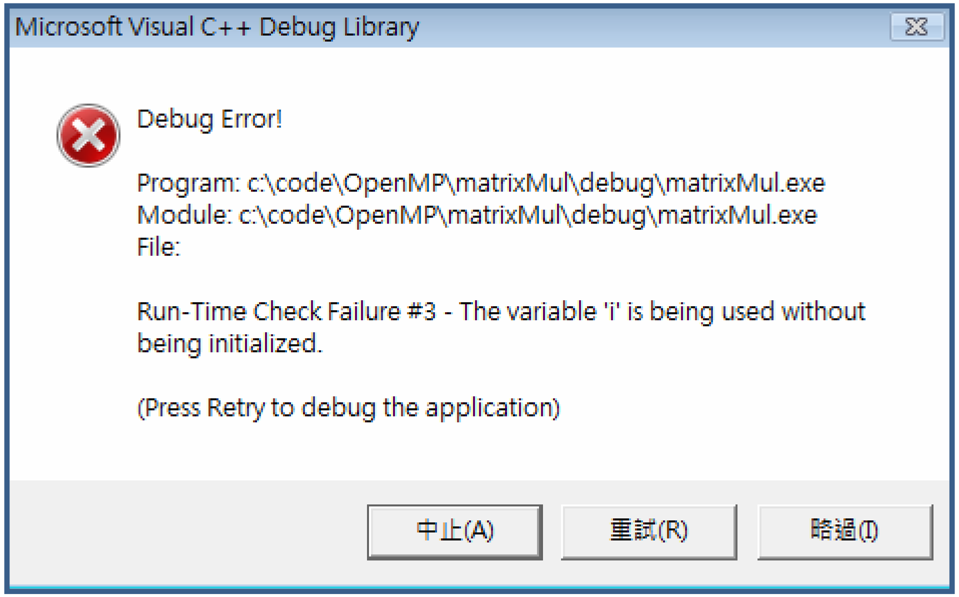
```
threads = 2, matrixMul cost(loop i) = 285 (ms)
threads = 2, matrixMul cost(loop j) = 228 (ms)
size(A) = (400,400)
size(B) = (400,400)
total memory size = 1.8311 (MB)
The error is 0.000000
```

```
threads = 2, matrixMul cost(loop i) = 284 (ms)
threads = 2, matrixMul cost(loop j) = 229 (ms)
size(A) = (400,400)
size(B) = (400,400)
total memory size = 1.8311 (MB)
The error is 0.000000
```

```
threads = 2, matrixMul cost(loop i) = 284 (ms)
threads = 2, matrixMul cost(loop j) = 227 (ms)
size(A) = (400,400)
size(B) = (400,400)
total memory size = 1.8311 (MB)
The error is 0.000000
```

Conclusion : loop j is a little faster than loop j, and the result of computation is the same.

2. why do we declare index ***i*** as shared variable? What happens if we declare index ***i*** as private variable?



Microsoft Visual C++ Debug Library

Debug Error!

Program: c:\code\OpenMP\matrixMul\debug\matrixMul.exe
Module: c:\code\OpenMP\matrixMul\debug\matrixMul.exe
File:

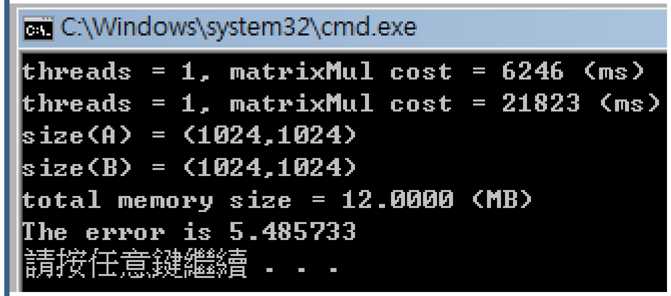
Run-Time Check Failure #3 - The variable 'i' is being used without being initialized.

(Press Retry to debug the application)

Exercise 5: verify subroutine ***matrixMul_block_seq*** with non-block version, you can use high precision package.

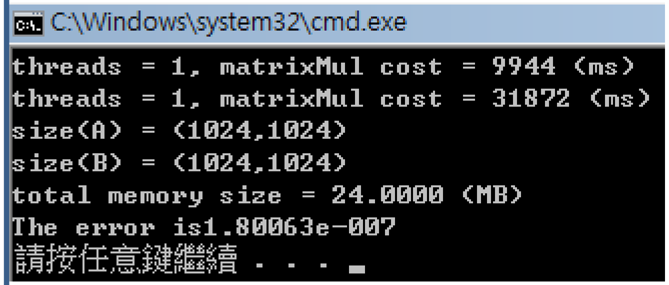
Take threads = 1, i.e. sequentially

Float



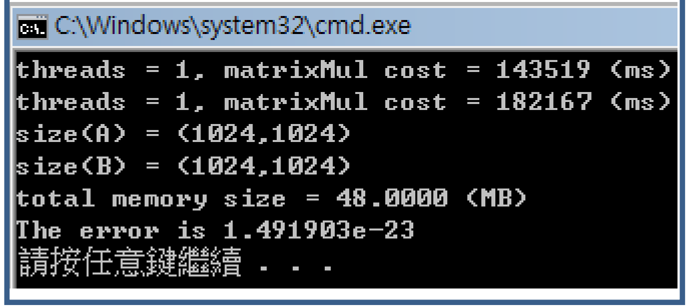
```
threads = 1, matrixMul cost = 6246 (ms)
threads = 1, matrixMul cost = 21823 (ms)
size(A) = (1024,1024)
size(B) = (1024,1024)
total memory size = 12.0000 (MB)
The error is 5.485733
請按任意鍵繼續 . . .
```

Double



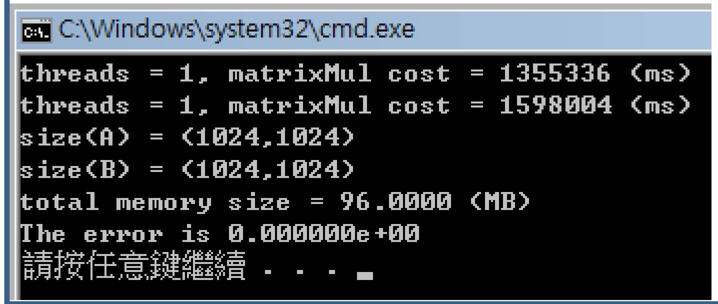
```
threads = 1, matrixMul cost = 9944 (ms)
threads = 1, matrixMul cost = 31872 (ms)
size(A) = (1024,1024)
size(B) = (1024,1024)
total memory size = 24.0000 (MB)
The error is1.80063e-007
請按任意鍵繼續 . . .
```

Double-double



```
threads = 1, matrixMul cost = 143519 (ms)
threads = 1, matrixMul cost = 182167 (ms)
size(A) = (1024,1024)
size(B) = (1024,1024)
total memory size = 48.0000 (MB)
The error is 1.491903e-23
請按任意鍵繼續 . . .
```

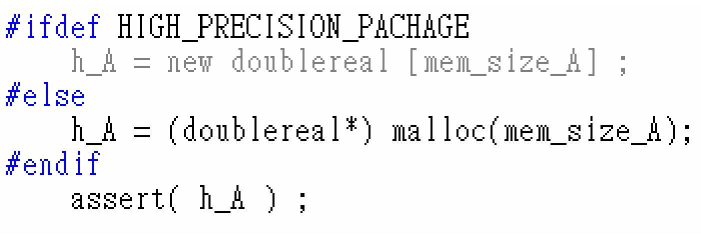
Quad-double



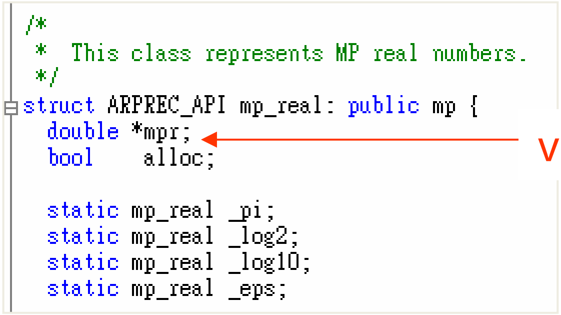
```
threads = 1, matrixMul cost = 1355336 (ms)
threads = 1, matrixMul cost = 1598004 (ms)
size(A) = (1024,1024)
size(B) = (1024,1024)
total memory size = 96.0000 (MB)
The error is 0.000000e+00
請按任意鍵繼續 . . .
```

Recall : How to modify code such that it can work with arbitrary precision ?

1



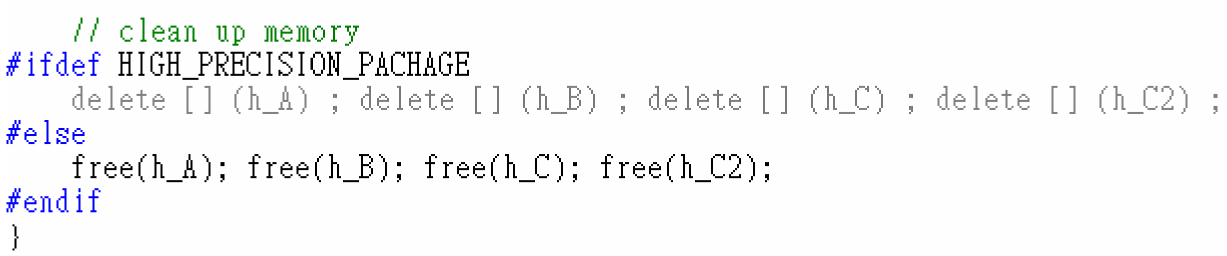
```
#ifdef HIGH_PRECISION_PACHAGE
    h_A = new doublereal [mem_size_A] ;
#else
    h_A = (doublereal*) malloc(mem_size_A);
#endif
    assert( h_A ) ;
```



```
/*
 *  This class represents MP real numbers.
 */
struct ARPREC_API mp_real: public mp {
  double *mpr;
  bool    alloc;

  static mp_real _pi;
  static mp_real _log2;
  static mp_real _log10;
  static mp_real _eps;
```

variable-length

2



```
    // clean up memory
#ifdef HIGH_PRECISION_PACHAGE
    delete [] (h_A) ; delete [] (h_B) ; delete [] (h_C) ; delete [] (h_C2) ;
#else
    free(h_A); free(h_B); free(h_C); free(h_C2);
#endif
}
```

Exercise 6: if we use “double”, how to choose value of BLOCK_SIZE, show your experimental result.

If we want to keep size of As and Bs are 1MB, since one double is 8 byte, that is twice of float (4 byte) .

$$8 \times n^2 = 4 \times 512 \times 512$$

$$n^2 = 256 \times 512$$

$$n \approx 362$$

Block version, BLOCK_SIZE = 362 (double)

Work Station : 140.114.34.1

N	Total size	Thread 1	Thread 2	Thread 4	Thread 8
2	12 MB	1,268 ms	645 ms	319 ms	194 ms
4	48 MB	10,031ms	4,876ms	2,481 ms	1,304 ms
8	192 MB	79,744ms	39,116ms	19,609ms	10,278ms

$$N_d^2 \times n^2 = N_f^2 \times 512 \times 512$$

$$N_d = N_f \times \sqrt{2}$$

$$Cost - time \propto N_d^3$$

Block version, BLOCK_SIZE = 362 (double)

Dimension	Thread 1	Thread 2	Thread 4	Thread 8
1024x1024	3,586 ms	1,824 ms	902 ms	548 ms
2048x2048	28,372ms	13,791ms	7,017 ms	3,688ms
4096x4096	225,550ms	110,640ms	55,463ms	29,071ms

Block version, BLOCK_SIZE = 512 (float)

Dimension	Thread 1	Thread 2	Thread 4	Thread 8
1024x1024	3,454 ms	1,881ms	882ms	4,63ms
2048x2048	28,990ms	14,302ms	6,991ms	3,540ms
4096x4096	224,142 ms	111,344ms	55,845ms	28,198ms

Conclusion : it cost almost the same time no matter you choose float or double under the same dimension of matrix.

Exercise 7: Can you modify subroutine ***matrixMul_block_parallel*** to improve its performance?

Exercise 8: compare parallel computation between CPU and GPU in your host machine