Fundementals of Parallel Programming OpenMP Section

HPC 101 - 2024 Summer

Lin Xi, Li Chenxiao @ ZJUSCT July 8, 2024



1. Introduction

2. OpenMP directives and constructs

3. Shared Data and Data Hazards

4. Miscellaneous



Introduction

_



ZJUSCT · HPC 101 - 2024 Summer

Parallel Programming





Parallel Programming



Multithreaded programming





Shared Memory Parallel Model



NUMA



Uniform memory access

Non-uniform memory

access

In real world



7IUSCT · HPC 101 - 2024 Summer Introduction • 3/38

OpenMP (Open Multi-Processing) is an API that supports multi-platform shared-memory multiprocessing programming in **C**, **C++**, and Fortran. It provides a set of compiler directives, library routines, and environment variables that allow developers to specify parallel regions, tasks, and other parallelism constructs.



OpenMP provides us an easy way to transform serial programs into parallel.



```
#include <stdio h>
#include <omp.h>
int main() {
 printf("Welcome to OpenMP!\n");
 #pragma omp parallel
  ł
    int ID = omp get thread num();
    printf("hello(%d)", ID);
    printf("world(%d)\n", ID);
  }
  printf("Bye!");
  return 0;
}
```

export OMP_NUM_THREADS=4

🔮 Output:

```
• lcx@M602:-/openmp-examples$ ./1_hello_openmp
Welcome to OpenMP program!
hello (2)hello (1)hello (3)world (2)
world (3)
world (1)
hello (0)world (0)
Bye!
elcx@M602:-/openmp-examples$ ./1_hello_openmp
Welcome to OpenMP program!
hello (0)world (2)
hello (2)world (2)
hello (3)world (3)
hello (1)world (1)
Bye!
```

\$ gcc -o hello_omp hello_omp.c -fopenmp # <-- Compiler Option</pre>



```
#include <stdio.h>
#include <omp.h>
int main() {
  printf("Welcome to OpenMP!\n");
  #pragma omp parallel
    int ID = omp get thread num();
    printf("hello(%d)", ID);
    printf("world(%d)\n", ID);
  }
  printf("Bye!");
  return 0:
```

Differences:

· Import OpenMP Header

Preprocessing directive

• Parallel Region



```
#include <stdio.h>
#include <omp.h>
int main() {
  printf("Welcome to OpenMP!\n");
  #pragma omp parallel
    int ID = omp get thread num();
    printf("hello(%d)", ID);
    printf("world(%d)\n", ID);
  }
  printf("Bye!");
  return 0:
```

Differences:

Import OpenMP Header

- $\cdot\,$ Preprocessing directive
 - Will cover commonly used directives
- Parallel Region



```
#include <stdio.h>
#include <omp.h>
int main() {
  printf("Welcome to OpenMP!\n");
  #pragma omp parallel
    int ID = omp get thread num();
    printf("hello(%d)", ID);
    printf("world(%d)\n", ID);
  }
  printf("Bye!");
  return 0:
```

Differences:

Import OpenMP Header

- Preprocessing directive
 - Will cover commonly used directives
- \cdot Parallel Region
 - Relates to the fork-join model



Fork-Join Model





Fork-Join Model

Thread ID: omp_get_thread_num()





OpenMP directives and constructs



ZJUSCT · HPC 101 - 2024 Summer

Work-Distribution Constructs

Overview for This Section

- How is work executed?
 - parallel directive
- How is work distributed to threads?
 - single directive
 - section directive
 - \cdot for directive
 - \cdot for loop schedule
 - Nested for loop
- In next section...
 - · directive: critical, atomic, barrier
 - clause: private, shared, reduction



OpenMP Directives

A legal OpenMP Directive must has the following format (C/C++):

Pragma	Directive	[clause[[,]clause]]
#pragma omp	parallel, atomic, critical,	0 to many

• 🌰 For example:

#pragma omp parallel for collapse(2) private(tmp_v, d, v)

- Case sensitive
- Affects the block (single statement or wrapped by {}) after this directive
- 🤪 Here's an official Cheat Sheet



What is the difference between construct and directive?

🤓 An OpenMP construct is a formation for which the directive is executable.¹

¹https://www.openmp.org/spec-html/5.2/openmpse14.html



Work-distribution constructs



Work-distribution constructs:

- single
- section
- \cdot for



Work-distribution constructs



Work-distribution constructs:

- single
- \cdot section
- \cdot for



Work-distribution constructs



Work-distribution constructs:

- single
- section
- $\cdot \,\, \text{for} \,\,$



parallel Directive

```
#include <stdio.h>
#include <omp.h>
int main() {
  printf("Welcome to OpenMP!\n");
  #pragma omp parallel
  ł
    int ID = omp_get_thread_num();
    printf("hello(%d)", ID);
    printf("world(%d)\n", ID);
  }
  printf("Bye!");
  return 0;
}
```



Combined Constructs and Directives

Example 2: parallel for Directive

```
// Addition of two vectors
for (int i = 0; i < N; i++) {
    c[i] = a[i] + b[i];
}</pre>
```

```
// Addition of two vectors
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    c[i] = a[i] + b[i];
}</pre>
```

```
    lcx@M602:~/openmp-examples$ echo $OMP_NUM_THREADS 4
    lcx@M602:~/openmp-examples$ ./2_vector_addition Serial: 1290.71 us Parallel: 419.164 us Speed Up: 3.07926x
```



Combined Constructs and Directives

Example 2: parallel for Directive

```
// Addition of two vectors
for (int i = 0; i < N; i++) {
    c[i] = a[i] + b[i];
}</pre>
```

```
// Addition of two vectors
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    c[i] = a[i] + b[i];
}</pre>
```

lcx@M602:~/openmp-examples\$ echo \$OMP_NUM_THREADS 4
 lcx@M602:~/openmp-examples\$./2_vector_addition Serial: 1290.71 us Parallel: 419.164 us Speed Up: 3.07926x

큫 Not 4x speed up



Combined Constructs and Directives

Example 2: parallel for Directive

```
// Addition of two vectors
for (int i = 0; i < N; i++) {
    c[i] = a[i] + b[i];
}</pre>
```

```
// Addition of two vectors
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    c[i] = a[i] + b[i];
}</pre>
```

Solution of excess or indirect computation time, memory, bandwidth, or other resources that are required to perform a specific task.



Valid for loops that can be parallelized by OpenMP:

for	index = start	;	index < end index <= end index >= end index > end	;	<pre>index++ ++index indexindex index += incr index -= incr index = index + incr index = incr + index</pre>
					index = incr + index index = index - incr



```
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    c[i] = f(i); // What if f is not 0(1), eg. 0(N)
}</pre>
```



```
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    c[i] = f(i); // What if f is not 0(1), eg. 0(N)
}</pre>
```

Workload is unbalanced!



```
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    c[i] = f(i); // What if f is not 0(1), eg. 0(N)
}</pre>
```

Workload is unbalanced!

• *schedule* clause specifies how iterations of associatedloops are divided into contiguous non-empty subsets, aka. **chunks**.



```
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    c[i] = f(i); // What if f is not 0(1), eg. 0(N)
}</pre>
```

Workload is unbalanced!

- *schedule* clause specifies how iterations of associatedloops are divided into contiguous non-empty subsets, aka. **chunks**.
- Type: Static, Dynamic, Guided, Runtime, Auto



```
#pragma omp parallel for schedule(static)
for (int i = 0; i < N; i++) {
    c[i] = f(i);
}</pre>
```

Is this a good choice?

- 👍 Pros: Less overhead in scheduling
- 👎 Cons: Workload may be unbalanced



```
#pragma omp parallel for schedule(dynamic, 2)
for (int i = 0; i < N; i++) {
    c[i] = f(i);
}</pre>
```

👍 Pros: More flexible scheduling

```
📙 Cons: More overhead in scheduling
```



```
#pragma omp parallel for schedule(guided, 2)
for (int i = 0; i < N; i++) {
    c[i] = f(i);
}</pre>
```

- Compare to *static*: *May* yield a better workload balance under certain circumstances
- Compare to dynamic: Less overhead to dispatch tasks
- In practice: Just try if it works better



Source code at: **3_schedule.c**

0 Static			N-1
thr Ø	thr 1	thr 2	thr 3
0 Static,n	2 thr 3 thr 0	thr 1 thr 2 thr 3	N-1 thr 0 thr 1 thr 2
0 Dynamic			N-1
thr 0 thr 1 thr	2 thr 3 thr 1	thr 0 thr 2 thr 1	thr 3 thr 1 t0
0 Guided	thr 2 thr 3 t	0 t 1 t 2 t 3	N-1
	iteration n	umber	





Static, Dynamic, Guided, Runtime, Auto





Find optimal schedule? 😣 NP-complete!



In some cases, we will encounter nested *for* loops. For example, matrix addition:

```
// Matrix Element-wise Addition
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}</pre>
```

```
#pragma omp parallel for collapse(2)
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}</pre>
```



In some cases, we will encounter nested *for* loops. For example, matrix addition:

```
// Matrix Element-wise Addition
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}</pre>
```

```
#pragma omp parallel for collapse(2)
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}</pre>
```

It's not always a good idea to parallelize nested loops.



In some cases, we will encounter nested *for* loops. For example, matrix addition:

```
// Matrix Element-wise Addition
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}</pre>
```

```
#pragma omp parallel for collapse(2)
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}</pre>
```

It's not always a good idea to parallelize nested loops.

Think about locality and data dependency before you use *collapse* clause.



Shared Data and Data Hazards



ZJUSCT · HPC 101 - 2024 Summer

```
#include <stdio.h>
#include <omp.h>
int main() {
    int a[100];
    int sum = 0:
   // initialize
    for (int i = 0; i < 100; i++) a[i] = i + 1;
    // Sum up from 1 to 100
#pragma omp parallel for
    for (int i = 0; i \le 99; i ++) {
        sum += a[i];
    }
    printf("Sum = %d\n", sum);
}
```



How Data Hazards Happen?



This case is especially important, you will encounter this again and again in future study, eg. Computer Organization and Database System.



- Shared & private data in default
- Explicit scopes definition
 - private
 - shared
 - \cdot firstprivate
 - lastprivate
- Data hazards happen when operating shared data

```
int sum = 0;
// Sum up from 1 to 100
#pragma omp parallel for
for (int i = 0; i <= 99; i++) {
    sum += a[i];
}
```



- Critical Section
- Atomic Operations
- Reduction



- Only one thread can enter critical section at the same time.
- A critical section can contain multiple statements.

```
#pragma omp parallel for
    for (int i = 0; i <= 99; i++) {
    #pragma omp critical
        { sum += a[i]; }
    }
    printf("Sum = %d\n", sum);
```



- Only one thread can enter critical section at the same time.
- A critical section can contain multiple statements.
- 🤯 Isn't this serial again?

```
#pragma omp parallel for
    for (int i = 0; i <= 99; i++) {
    #pragma omp critical
        { sum += a[i]; }
    }
    printf("Sum = %d\n", sum);
```



- Atomic operation cannot be separated.
- Only can be applied to one operation
- Limited set of operators supported

```
#pragma omp parallel for
    for (int i = 0; i <= 99; i++) {
    #pragma omp atomic
        sum += a[i];
    }
    printf("Sum = %d\n", sum);
```



Example: Solution with Atomic Operation

- Atomic operation cannot be separated.
- Only can be applied to one operation
- Limited set of operators supported

C/C++ statement:

if atomic clause is	statement:			
read	v = x;			
write	x = expr;			
update	x++; x; ++x;x; x binop = expr; x = x binop expr; x = expr binop x;			
compare is present	<pre>cond-expr-stmt: x = expr ordop x ? expr : x; x = x ordop expr ? expr : x; x = x = e ? d : x; cond-update-stmt: fl(expr ordop x) { x = expr; } fl(x = ordop x) { x = expr; } fl(x = c) { x = d; }</pre>			
capture is present	<pre>v = expr-stmt {v = x; expr-stmt } {expr-stmt v = x; } (expr-stmt: write-expr-stmt, update-expr-stmt, or cond-expr-stmt.)</pre>			
both compare and capture are present	{v = x; cond-update-stmt } {cond-update-stmt v = x; } if(x = e) {x = d; } else {v = x; } {r = x = e; if(r) {x = d; } {r = x = e; if(r) {x = d; } les {v = x; } }			



- Create temporary private variables for each thread
- Reduce these private variables in the end
- Limited set of operators supported

```
#pragma omp parallel for reduction(+:sum)
    for (int i = 0; i <= 99; i++) {
        sum += a[i];
    }
    printf("Sum = %d\n", sum);</pre>
```



- Critical Region: Based on locking
- Atomic Operation: Based on hardware atomic operations
- **Reduction**: only synchronize in the end

46 41 L1. 47 49 L6: 48 mov rax, QWORD PTR [rbp-40] 49 mov rax, QWORD PTR [rbp-20] 50 mov ecx, DWORD PTR [rbp-20] 51 mov ecx, DWORD PTR [rbp-20] 52 mov ecx, DWORD PTR [rbp-40] 53 51 mov rax, QWORD PTR [rbp-40] 54 52 mov rax, QWORD PTR [rbp-40] 55 Lock add DWORD PTR [rbp-20] 54 53+ mov edx, DWORD PTR [rbp-20] 54+ mov edx, DWORD PTR [rbp-40] 55+ mov edx, DWORD PTR [rbp-40] 56+ mov eax, QWORD PTR [rbp-40] 57+ mov eax, QWORD PTR [rbp-40] 58+ add edx, eax 59+ mov DWORD PTR [rbp-40] 60+ mov DWORD PTR [rbp-40] 60+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 cmp DWORD PTR [rbp-20], ebx 61+ call GOMP_critical_end 56 call GOMP_critical_end 57 cmp DWORD PTR [rbp-20], ebx 63+ cmp DWORD PTR [rbp-20], ebx		4.4	1.71						
47 49 .L6: 48 - mov rax, QWORD PTR [rbp-40] 49 - mov rax, QWORD PTR [rbp-20] 50 - mov ecx, DWORD PTR [rbp-20] 51 - mov ecx, DWORD PTR [rbp-40] 52 - mov ecx, DWORD PTR [rbp-40] 53 51 mov rax, QWORD PTR [rbp-40] 54 52 mov edx, DWORD PTR [rbp-20] 55 - Lock add DWORD PTR [rbp-20] 54 mov edx, DWORD PTR [rbp-40] startds 54+ mov edx, DWORD PTR [rbp-40] startds 55+ mov eax, QWORD PTR [rbp-40] startds 56+ mov eax, DWORD PTR [rbp-40] startds 57+ mov eax, DWORD PTR [rbp-40] mov DWORD PTR [rbp-40] 58+ add edx, eax startds 59+ mov DWORD PTR [rbp-40] mov DWORD PTR [rbp-40] 60+ call GOMP_critical_end startds 61+ call GOMP_critical_end startds 62 add DWORD PTR [rbp-20], 1 startds 57 cmp DWORD PTR [rbp-20], ebx startds <td>46</td> <td>41</td> <td> / ;</td> <td></td> <td></td> <td></td> <td>L</td> <td>,</td> <td></td>	46	41	/ ;				L	,	
48 mov rax, QWORD PTR [rbp-40] 49 mov rax, QWORD PTR [rax+8] 50 mov ecx, DWORD PTR [rbp-20] 51 mov ecx, DWORD PTR [rbp-20] 52 mov ecx, DWORD PTR [rbp-40] 53 mov rax, QWORD PTR [rbp-40] 54 52 55 Lock add 54+ mov edx, DWORD PTR [rbp-20] 54+ mov rax, QWORD PTR [rbp-20] 54+ mov rax, QWORD PTR [rbp-20] 54+ mov edx, DWORD PTR [rbp-40] 54+ mov edx, DWORD PTR [rbp-40] 54+ mov rax, QWORD PTR [rbp-40] 54+ mov rax, QWORD PTR [rbp-40] 57+ mov eax, DWORD PTR [rbp-40] 58+ add edx, eax 59+ mov DWORD PTR [rbp-40] 60+ mov DWORD PTR [rbp-20], 1 56 call GOMP_critical_end 56 call GOMP_critical_end 56 add DWORD PTR [rbp-20], ebx 63+ cmp DWORD PTR [rbp-20], ebx 63+ cmp DWORD PTR [rbp-20], ebx	47	49	.L6:						
49 - mov rax, QWORD PTR [rax+8] 50 - mov ecx, DWORD PTR [rbp-20] 51 - mov secx, DWORD PTR [rbp-20] 52 - mov ecx, DWORD PTR [rax+rcx*4] 50+ - call GOMP_critical_start 53 51 mov rax, QWORD PTR [rbp-40] 54 52 mov rax, QWORD PTR [rax], ecx 55 - - Lock add DWORD PTR [rax], ecx 54+ mov edx, DWORD PTR [rbp-40] sex 54+ mov edx, DWORD PTR [rbp-40] sex 55+ mov eax, QWORD PTR [rbp-40] mov 56+ mov rax, QWORD PTR [rbp-40] mov nov 56+ mov eax, DWORD PTR [rbp-40] mov DWORD PTR [rbp-40] 60+ call GOMP_critical_end 60+ call GOMP_critical_end 61+ 61+ call GOMP_critical_end mov DWORD PTR [rbp-20], 1 1 57 - cmp DWORD PTR [rbp-20], ebx 1 63+ cmp DWORD PTR [rbp-20], ebx	48	-		mov	rax,	QWORD	PTR	[rbp-40]	
50 - mov ecx, DWORD PTR [rbp-20] 51 - movsx rcx, ecx 52 - mov ecx, DWORD PTR [rax+rcx*4] 53 51 mov rax, QWORD PTR [rbp-40] 54 52 - 55 - Lock add DWORD PTR [rax] 55 - - Lock add DWORD PTR [rax] 55 - - Lock add DWORD PTR [rax] 54 mov edx, DWORD PTR [rbp-40] mov acx, QWORD PTR [rbp-40] 54+ mov edx, DWORD PTR [rax+dx*4] mov eax, QWORD PTR [rbp-40] 56+ mov eax, QWORD PTR [rbp-40] mov acx, QWORD PTR [rbp-40] 57+ mov eax, DWORD PTR [rbp-40] mov DWORD PTR [rbp-40] 60+ call GOMP_critical_end 60+ 61+ call GOMP_critical_end 61+ 56 62 add DWORD PTR [rbp-20], 1 57 - cmp DWORD PTR [rbp-20], ebx 63+ cmp DWORD PTR [rbp-20], ebx 63+ cmp DWORD PTR [rbp-20], ebx 58 64 jl . L6	49	-		mov	rax,	QWORD	PTR	[rax+8]	
51 - movSX rCX, eCX 52 - mov ecx, DWORD PTR [rax+rcx*4] 53 51 mov rax, QWORD PTR [rbp-40] 54 52 mov rax, QWORD PTR [rbp-40] 54 52 mov rax, QWORD PTR [rbp-20] 55 - Lock add DWORD PTR [rbp-20] 54+ mov edx, DWORD PTR [rbp-40] 55+ mov edx, DWORD PTR [rbp-40] 56+ mov eax, DWORD PTR [rbp-40] 57+ mov eax, DWORD PTR [rbp-40] 58+ add edx, eax 59+ mov rax, QWORD PTR [rbp-40] 60+ call GOMP_critical_end 61+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 - cmp DWORD PTR [rbp-20], ebx 63+ cmp DWORD PTR [rbp-20], ebx	50	-		mov	ecx,	DWORD	PTR	[rbp-20]	
52 - mov ecx, DWORD PTR [rax+rcx*4] 50+ call GOMP_critical_start 53 51 mov rax, QWORD PTR [rbp-40] 54 52 mov rax, QWORD PTR [rax] 55 - Lock add DWORD PTR [rax] 54 52 mov edx, DWORD PTR [rbp-20] 54+ movsx rdx, edx 55+ mov edx, DWORD PTR [rbp-40] 56+ mov rax, QWORD PTR [rbp-40] 57+ mov eax, DWORD PTR [rbp-40] 58+ add edx, eax 59+ mov rax, QWORD PTR [rbp-40] 60+ mov DWORD PTR [rbp-20], 1 56 call GOMP_critical_end 56 add DWORD PTR [rbp-20], 1 57 cmp DWORD PTR [rbp-20], ebx 63+ cmp DWORD PTR [rbp-20], ebx	51	-		movsx	rcx,	ecx			
50+ call GOMP_critical_start 53 51 mov rax, QWORD PTR [rbp-40] 54 52 mov rax, QWORD PTR [rbp-40] 55 - Lock add DWORD PTR [rax], ecx 55 - Lock add DWORD PTR [rbp-20] 54+ mov edx, DWORD PTR [rbp-20] 54+ 55+ mov edx, DWORD PTR [rbp-40] 56+ mov eax, DWORD PTR [rbp-40] 57+ mov eax, DWORD PTR [rbp-40] 60+ mov DWORD PTR [rbp-40] 60+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 - cmp DWORD PTR [rbp-20], ebx 63+ cmp DWORD PTR [rbp-20], ebx 58 64 jl	52	-		mov	ecx,	DWORD	PTR	[rax+rcx*4]	
53 51 mov rax, QWORD PTR [rbp-40] 54 52 mov rax, QWORD PTR [rax] 55 lock add DWORD PTR [rax], ecx 54 mov edx, DWORD PTR [rbp-20] 54+ mov edx, DWORD PTR [rbp-40] 55+ mov edx, QWORD PTR [rbp-40] 56+ mov edx, DWORD PTR [rbp-40] 57+ mov eax, QWORD PTR [rbp-40] 58+ add edx, eax 59+ mov rax, QWORD PTR [rbp-40] 60+ mov DWORD PTR [rbp-40] 60+ call GOMP_critical_end 56 add DWORD PTR [rbp-20], 1 57 cmp DWORD PTR [rbp-20], ebx 63+ cmp DWORD PTR [rbp-20], ebx 63+ cmp DWORD PTR [rbp-20], ebx		50+		call	GOMP_	criti	cal_s	start	
54 52 mov rax, QWORD PTR [rax] 55 - Lock add DWORD PTR [rax], ecx 53+ mov edx, DWORD PTR [rbp-20] 54+ mov edx, DWORD PTR [rbp-20] 55+ mov edx, DWORD PTR [rbp-40] 56+ mov eax, QWORD PTR [rbp-40] 58+ add edx, eax 59+ mov Tax, QWORD PTR [rbp-40] 60+ call GOMP_critical_end 56 eadd DWORD PTR [rbp-20], 1 57 cmp DWORD PTR [rbp-20], ebx 63+ cmp DWORD PTR [rbp-20], ebx 58 64 jl .L6	53	51		mov	rax,	QWORD	PTR	[rbp-40]	
55 lock add DWORD PTR [rax], ecx 53+ mov edx, DWORD PTR [rbp-20] 54+ movsx rdx, edx 55+ mov edx, DWORD PTR [rax+rdx*4] 56+ mov edx, DWORD PTR [rbp-40] 57+ mov eax, DWORD PTR [rax+8] 58+ add edx, eax 59+ mov rax, QWORD PTR [rbp-40] 60+ mov DWORD PTR [rax+8], edx 61+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 cmp DWORD PTR [rbp-20], edx 63+ cmp DWORD PTR [rbp-20], ebx 58 64 jl .L6	54	52		mov	rax,	QWORD	PTR	[rax]	
53+ mov edx, DWORD PTR [rbp-20] 54+ movsx rdx, edx 55+ mov edx, DWORD PTR [rax+rdx*4] 56+ mov rax, QWORD PTR [rbp-40] 57+ mov eax, DWORD PTR [rbp-40] 58+ add edx, eax 59+ mov rax, QWORD PTR [rbp-40] 60+ mov DWORD PTR [rax+8], edx 61+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 - cmp DWORD PTR [rbp-20], edx 63+ cmp DWORD PTR [rbp-20], edx 58 64 jl .L6	55	-		lock ac	ld	DWO	ORD F	PTR [rax], ec	x
54+ movsx rdx, edx 55+ mov edx, DWORD PTR [rax+rdx*4] 56+ mov rax, QWORD PTR [rbp-40] 57+ mov eax, DWORD PTR [rax+8] 58+ add edx, eax 59+ mov rax, QWORD PTR [rbp-40] 60+ mov DWORD PTR [rbp-40] 61+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 - cmp DWORD PTR [rbp-20], edx 63+ cmp DWORD PTR [rbp-20], edx 63+ cmp DWORD PTR [rbp-20], ebx 58 64 jl .L6		53+		mov	edx,	DWORD	PTR	[rbp-20]	
55+ mov edx, DWORD PTR [rax+rdx*4] 56+ mov rax, QWORD PTR [rbp-40] 57+ mov eax, DWORD PTR [rbp-40] 58+ add edx, eax 59+ mov rax, QWORD PTR [rbp-40] 60+ mov DWORD PTR [rbp-40] 61+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 - cmp DWORD PTR [rbp-20], ebx 56 62 add DWORD PTR [rbp-20], ebx 57 - cmp DWORD PTR [rbp-20], ebx		54+		movsx	rdx,	edx			
56+ mov rax, QWORD PTR [rbp-40] 57+ mov eax, DWORD PTR [rax+8] 58+ add edx, eax 59+ mov rax, QWORD PTR [rbp-40] 60+ mov DWORD PTR [rax+8], edx 61+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 - cmp DWORD PTR [rbp-20], edx 63+ cmp DWORD PTR [rbp-20], ebx 58 64 il .L6		55+		mov	edx,	DWORD	PTR	[rax+rdx*4]	
57+ mov eax, DWORD PTR [rax+8] 58+ add edx, eax 59+ mov rax, QWORD PTR [rbp-40] 60+ mov DWORD PTR [rax+8], edx 61+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 - cmp DWORD PTR [rbp-20], edx 63+ cmp DWORD PTR [rbp-20], edx 58 64 jl .L6		56+		mov	rax,	QWORD	PTR	[rbp-40]	
58+ add edx, eax 59+ mov rax, QWQRD PTR [rbp-40] 60+ mov DWQRD PTR [rax+8], edx 61+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 - cmp DWORD PTR [rbp-20], edx 63+ cmp DWORD PTR [rbp-20], edx 58 64 jl .L6		57+		mov	eax,	DWORD	PTR	[rax+8]	
59+ mov rax, QWORD PTR [rbp-40] 60+ mov DWORD PTR [rbp-40] 61+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 - cmp DWORD PTR [rbp-20], edx 63+ cmp DWORD PTR [rbp-20], ebx 58 64 jl .L6		58+		add	edx,	eax			
60+ mov DWORD PTR [rax+8], edx 61+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 - Cmp DWORD PTR [rbp-20], edx 63+ cmp DWORD PTR [rbp-20], ebx 58 64 jl .L6		59+		mov	rax,	QWORD	PTR	[rbp-40]	
61+ call GOMP_critical_end 56 62 add DWORD PTR [rbp-20], 1 57 - cmp DWORD PTR [rbp-20], edx 63+ cmp DWORD PTR [rbp-20], edx 58 64 jl .L6		60+		mov	DWORD) PTR	[rax+	+8], edx	
56 62 add DWORD PTR [rbp-20], 1 57 - cmp DWORD PTR [rbp-20], edx 63+ cmp DWORD PTR [rbp-20], ebx 58 64 jl .L6		61+		call	GOMP_	critic	cal_e	end	
57 cmp DWORD PTR [rbp-20], edx 63+ cmp DWORD PTR [rbp-20], ebx 58 64 jl .L6	56	62		add	DWORD) PTR	[rbp-	20], 1	
63+ cmp DWORD PTR [rbp-20], ebx 58 64 jl .L6	57	-		cmp	DWORD) PTR	[rbp-	20], edx	
58 64 jl .L6		63+		cmp	DWORD) PTR	[rbp-	20], ebx	
	58	64		jl	.L6				



```
//General Matrix Multiplication (GEMM)
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        for (int k = 0; k < N; k++) {
            c[i][j] += a[i][k] * b[k][j];
        }
    }
}</pre>
```



Miscellaneous



ZJUSCT · HPC 101 - 2024 Summer

$\cdot \bigcirc$ Barrier: Wait until all thread reach here

- Implicit barrier in parallel region
- nowait clause
- 🤒 Locking: wait until obtain the lock
 - Often apply to data structures





Nested Parallel Region

- Disabled by default.
- Use *omp_set_nested* to enable.
- Consider refactor the code.

```
int f(int n, int* a, int* b, int* c) {
    #pragma omp parallel for
    for (int i = 0; i < n; i++) {</pre>
        c[i] = a[i] + b[i];
    }
int main(){
    . . .
    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        f(n[i], a[i], b[i], c[i]);
}
```



- Supposed to be private, but actually perform like sharing
- Seems working independently, but actually sharing the same cache line
- Does harm to parallel performance





- Supposed to be private, but actually perform like sharing
- Seems working independently, but actually sharing the same cache line
- Does harm to parallel performance



Be cache-friendly and avoid false sharing: Make use of space locality



- 1. Where to parallelize: Profiling
- 2. Whether to parallelize: Analyze data dependency
- 3. How to parallelize: Analysis and Skills
 - Sub-task Distribution
 - Scheduling Strategy
 - \cdot Cache and Locality
 - Hardware Environment
 - · Somtimes: transform recursion to iteration
- 4. Get Down to Work: Testing



- 1. Ensure correctness while parallelizing
- 2. Be aware of overhead
- 3. Check more details in official documents
 - For example, OpenMP on GPU.



- An introduction to parallel programming Peter S. Pacheco, Matthew Malensek
- SC17 Loop Schedule for OMP



ΤΗΑΝΚ ΥΟυ

LET'S HAVE A BREAK!



ZJUSCT · HPC 101 - 2024 Summer | Miscellaneous · 38/38