
Master Course – High-Performance Computing

Parallel Programming with OpenMP
Part II

Olaf Schenk
Institute of Computational Science
USI Lugano, Switzerland
October 6, 2020

Outline — Data environment and combined constructs

- Introduction into OpenMP
- Programming and Execution Model
 - Parallel regions: team of threads
 - Syntax
 - Data environment (part 1)
 - Environment variables
 - Runtime library routines
 - Exercise 1: Parallel region / library calls / privat & shared variables
- Worksharing directives
 - Which thread executes which statement or operation?
 - Synchronization constructs, e.g., critical section
 - Exercise 2: Pi
- **Data environment and combined constructs**
 - **Nesting and Binding**
 - **Private and shared variables, Reduction clause**
 - **Combined parallel worksharing directives**
 - **Exercise 3: Pi with reduction clause and combined constructs**
 - **Exercise 4: Heat**
- Summary of OpenMP API
- OpenMP Pitfalls & Optimization Problems

Scope of variables (1)

```
#pragma omp parallel for \  
default(none) private(i) \  
shared (a,b,c)  
for (i=0; i<N; i++)  
    a[i]=b[i]+c[i];
```

- Local variables of a method/function called in a parallel region are put onto the stack. They are **private**.
- Static variables of a function called in a parallel region are **shared**.
- The shared memory programming model:
By default all variables are **shared**.
- Global variables (variables with a static or extern attribute) are **shared**.
- An exception are loop iteration variables, which automatically are **private**.
- The default can be changed by:
default (shared|none).

Scope of variables (2)



- One can selectively change storage attributes constructs using the following clauses
 - **SHARED**
 - **PRIVATE**
 - **FIRSTPRIVATE**
- The default status can be modified with:
 - **DEFAULT (SHARED | NONE)**
- The value of a private variable inside a parallel loop can be transmitted to a global value outside the loop with:
 - **LASTPRIVATE**

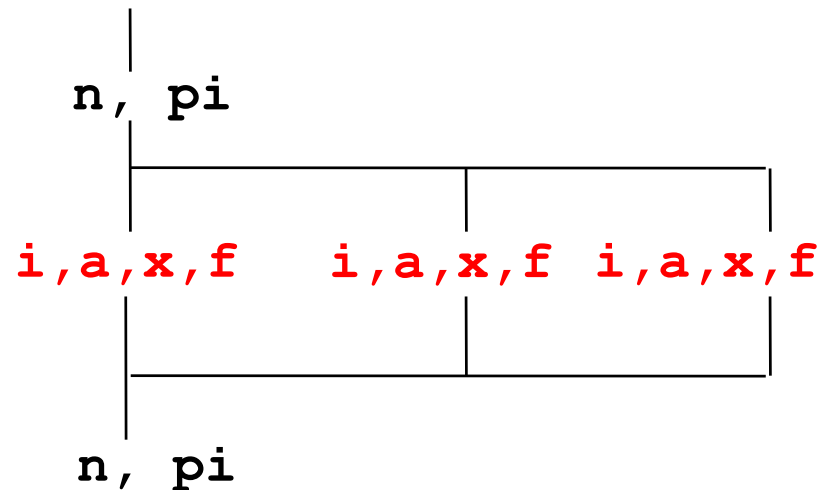
Scope of variables (3) - defaults

```
int n;  
void calc_pi(double *);  
main()  
{  
    double pi;  
    ...  
    #pragma omp parallel  
    {  
        for ( . . . )  
        {  
            call calc_pi( &pi )  
        }  
    } //end of parallel region  
    ...  
} // end of program main
```

n and pi are shared by all threads.
a, i, x, f are local to each thread

```
extern int n;  
void calc_pi(double *pi)  
{  
    int i;  
    static double sum, h;  
    double a, x, f;  
    . . .  
    *pi =  
    ...  
}
```

 = shared
 = private

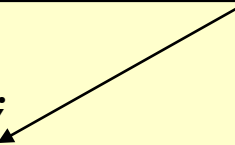


Scope of variables (4) - private

- **private(var)** creates a local copy of var for each thread.
 - The value is uninitialized
 - Private copy is *not* storage associated with the original

```
#include <stdio.h>
#include <omp.h>
int main(void)
{
    int i=42;
    printf("before PR: i=%d \n", i);
    #pragma omp parallel private(i)
    {
        printf("( %d):i=\n", omp_get_thread_num(), i);
        i+= omp_get_thread_num();
        printf("( %d):  i=%d\n", omp_get_thread_num(), i);
    }
    printf("after PR: i=%d \n", i);
}
```

An uninitialized copy is
allocated for each thread



Output:

```
before PR: i=42
(0): i=0
(1): i=3445
(0):  i=1
(1):  i=3446
after PR: i=42
```

Scope of variables (4) – firstprivate, lastprivate

```
int main(void)
{
    int i=42;
    printf("before PR: i=%d \n", i);
    #pragma omp parallel firstprivate(i)
    {
        printf("( %d): \n", omp_get_thread_num(), i );
        i+= omp_get_thread_num();
        printf("( %d): i %d\n", omp_get_thread_num(), i)
    }
    printf("after PR: i=%d \n", i);
}
```

The private copy is initialized with the original value before the parallel region

Output:

before PR: i=42

(0): i:42

(1): i:42

(0): i:42

(1): i:43

after PR: i=42

```
#pragma omp parallel firstprivate(i) lastprivate (i)
{
    printf("( %d): \n", omp_get_thread_num(), i );
    i+= omp_get_thread_num();
    printf("( %d): i %d\n", omp_get_thread_num(), i );
}
```

i gets the value of the last (sequential) iteration

after PR: i=43

Scope of variables (5) - A data environment test

- Here's an example of `private` and `firstprivate`

```
variables A=1, B=1, and C=1  
#pragma omp parallel private(B) shared(A) \  
firstprivate(C)
```

- Inside the parallel region ...
 - “A” is shared by all threads; equals 1
 - “B” and “C” are local to each thread.
 - B’s initial value is undefined
 - C’s initial value equals 1
- Outside this parallel region ...
 - The values of “B” and “C” are undefined.

Critical regions (6) – Reduction clause

```
for (i=0; i<100; i++)  
    s = s + a[i];
```

```
#pragma omp parallel for \  
private(i) reduction(+:s)  
for (i=0; i<100; i++)  
    s += a[i];
```

The reduction clause is tailored for this frequently occurring case

reduction({op}:list)

with

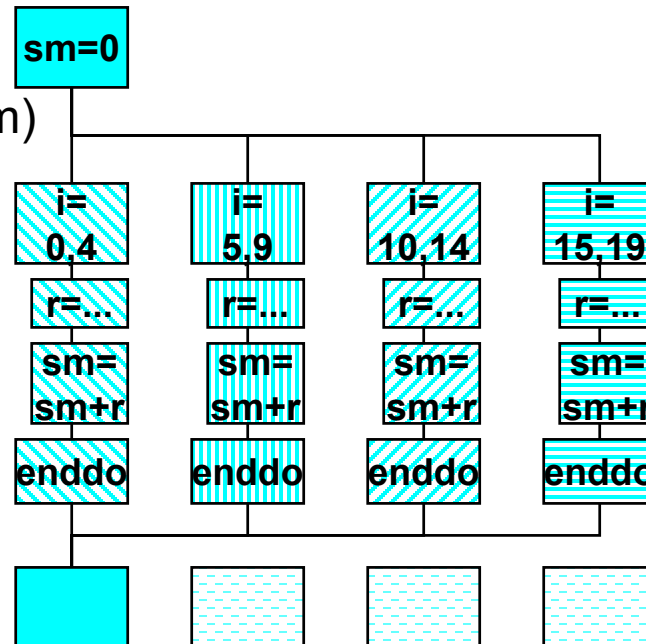
op = { + | * | - | || | && }

list is a comma separated list of variables.

OpenMP reduction — an example (C/C++)

C / C++:

```
    sm = 0;  
#pragma omp parallel for reduction(+:sm)  
    for( i=0; i<20; i++)  
    {   double r;  
        r = work(i);  
        sm = sm + r ;  
    } /*end for*/  
/*omp end parallel for*/
```



Outline — Exercise 3: pi with reduction

- Introduction into OpenMP
- Programming and Execution Model
 - Parallel regions: team of threads
 - Syntax
 - Data environment (part 1)
 - Environment variables
 - Runtime library routines
 - Exercise 1: Parallel region / library calls / privat & shared variables
- Worksharing directives
 - Which thread executes which statement or operation?
 - Synchronization constructs, e.g., critical section
 - Exercise 2: Pi
- Data environment and combined constructs
 - Nesting and Binding
 - Private and shared variables, Reduction clause
 - Combined parallel worksharing directives
 - **Exercise 3: Pi with reduction clause and combined constructs**
 - **Exercise 4: Heat (Assignment 1)**

In-class exercise 3: pi Program (1)

- Goal: usage of
 - worksharing constructs: **do/for**
 - **critical directive**
 - **reduction clause**
 - combined parallel worksharing constructs:
parallel do/parallel for
 - Modify **pi2.c** -> **pi3.c**
- remove **critical** directive in **pi3.c**, add **reduction** clause and compile
- set environment variable **OMP_NUM_THREADS** to **2** and run
 - value of pi?
- run again
 - value of pi?
- set environment variable **OMP_NUM_THREADS** to **4** and run
 - value of pi? execution time?
- run again
 - value of pi? execution time?

In-class exercise 3: pi Program (2)

- change parallel region + **do/for** to the combined parallel worksharing construct **parallel for** and compile
- set environment variable **OMP_NUM_THREADS** to **2** and run
 - value of pi?
- run again
 - value of pi?
- set environment variable **OMP_NUM_THREADS** to **4** and run
 - value of pi?
- run again
 - value of pi?
- At the end, compile again **without** OpenMP
 - Does your code still compute a **correct** value of **pi**?

OpenMP Bugs – Example 1

```
25  a[0] = 0;
26  #pragma omp parallel for
27  for (i=1; i<N; i++)
28  {
29      a[i] = 2.0*i*(i-1);
30      b[i] = a[i] - a[i-1];
31  } // end of omp parallel for
```

- What's wrong?
- How to solve?

OpenMP Bugs– Analysis 1

ID	Short Info	Type	Con-text	Description	1st Access	2. Access
1	Write ->Read	Error	omp for	Memory read of a[] at “test.c:30” conflicts with a prior memory write of a[] at test.c: 29 (flow dependencies)	“test.c”: 29	“test.c”: 30

```
25    a[0] = 0;
26    #pragma omp parallel for
27    for (i=1; i<N; i++)
28    {
29        a[i] = 2.0*i*(i-1);
30        b[i] = a[i] - a[i-1];
31    } // end of omp parallel for
```

Solution:

- Two separate loops → will cause bad cache reuse! **or**
- Re-computing of $a[i-1]$ i.e. $b[i] = a[i] - 2.0*(i-1)*(i-2);$

OpenMP Bugs – Example 2

```
43  a[0] = 0;
44  #pragma omp parallel
45  {
46      #pragma omp for nowait
47      for (i=1; i<N; i++)
48      {
49          a[i] = 3.0*i*(i+1)
50      } // end of omp for nowait
51      #pragma omp for
52      for (i=1; i<N; i++)
53      {
54          b[i] = a[i] - a[i-1];
55      } // end of omp for
56  } // end of omp parallel
```

- What's wrong?
- How to solve?

OpenMP Bugs – Analysis 2

ID	Short Info	Type	Con-text	Description	1st Access	2. Access
2	Write ->Read Data-race	Error	omp parallel region	Memory read of a[] at “test.c:54 conflicts with a prior memory write of a[] at test.c: 49 (flow dependencies)	“test.c”: 49	“test.c”: 54
3	Read ->Write	Error	omp parallel region	Memory write of a[] at “test.c:49 conflicts with a prior memory read of a[] at test.c: 54 (anti dependencies)	“test.c”: 54	“test.c”: 49

```
43 a[0] = 0;
44 #pragma omp parallel
45 {
46     #pragma omp for nowait
47     for (i=1; i<N; i++)
48     {
49         a[i] = 3.0*i*(i+1)
50     } // end of omp for nowait
```

```
51 #pragma omp for
1   for (i=1; i<N; i++
2   {
54       b[i] = a[i] - a[i-1];
1   } // end of omp for
56} // end of omp parallel
```

Solution: Remove nowait

OpenMP Bugs – Example 3

```
68  #pragma omp parallel for
69    for (i=1; i<N; i++)
70    {
71        x = sqrt(b[i]) - 1;
72        a[i] = x*x + 2*x + 1
73    } // end of omp parallel for
```

- What's wrong?
- How to solve?

OpenMP Bugs – Analysis 3

ID	Short Info	Type	Con-text	Description	1st Access	2. Access
4	Write ->Write Datarace	Error	omp for	Memory write of x at “test.c: 71 ” conflicts with a prior memory write of x at test.c: 71 (output dependencies)	“test.c”: 71	“test.c”: 72
5	Read ->Write	Error	omp for	Memory write of x at “test.c: 71 ” conflicts with a prior memory read of x at test.c: 72 (anti dependencies)	“test.c”: 72	“test.c”: 71

```
68  #pragma omp parallel for
69  for (i=1; i<N; i++)
70  {
71      x = sqrt(b[i]) - 1;
72      a[i] = x*x + 2*x + 1
73  } // end of omp parallel for
```

Solution: Add `privat(x)`

In the printed version, the solutions can be found in the appendix

OpenMP Bugs – Example 4

```
84     f = 2;
85     # pragma omp parallel for private (f,x)
86     for (i=1; i<N; i++)
87     {
88         x = f * b[i];
89         a[i] = x - 7;
90     } // end of omp parallel for
91     a[0] = x;
```

- What's wrong?
- How to solve?

OpenMP Bugs – Analysis 4

ID	Short Info	Type	Con-text	Description	1st Access	2. Access
6	undefined in access	Caution	omp parallel region	OpenMP – the access at “test.c”:88 is undefined, the expected value was defined at “test.c”:84 in serial execut.	“test.c”: 84	“test.c”: 88
7	undefined in access	Warning	“test.c”: 17	OpenMP – undefined in the serial code (original program) at “test.c”:91 with “test.c”: 88	“test.c”: 88	“test.c”: 91

```
84      f = 2;
85      # pragma omp parallel for private (f,x)
86      for (i=1; i<N; i++)
87      {
88          x = f * b[i];
89          a[i] = x - 7;
90      } // end of omp parallel for
91      a[0] = x;
```

Solution: Use firstprivate(f) lastprivate(x) instead of private(f,x)

OpenMP Bugs – Example 5

```
101     sum = 0;
102  # pragma omp parallel
103     for (i=1; i<N; i++)
104     {
105         sum = sum + b[i];
107     } // end of omp parallel for
```

- What's wrong?
- How to solve?

OpenMP Bugs – Analysis 5

ID	Short Info	Type	Con-text	Description	1st Access	2. Access
8	Read ->Write	Error	omp for	Memory write of sum at “test.c:105 conflicts with a prior memory read of sum at test.c: 105 (anti dependencies)	“test.c”: 105	“test.c”: 105

```
101    sum = 0;
102    # pragma omp parallel for
103    for (i=1; i<N; i++)
104    {
105        sum = sum + b[i];
107    } // end of omp parallel for
```

Solution: Add reduction(+:sum)

In the printed version, the solutions can be found in the appendix

Appendix

Example 1:

- Two separate loops → will cause bad cache reuse! **or**
- Re-computing of $a[i-1]$ i.e. $b[i] = a[i] - 2.0*(i-1)*(i-2);$

Example 2:

- Remove nowait

Example 3:

- Add `privat(x)`

Example 4:

- Use `firstprivate(f)` `lastprivate(x)` instead of `private(f,x)`

Example 5:

- Add `reduction(+:sum)`