FARABI UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE



Lecture 9.
Introduction to OpenMP (Part 2)

Dr. Bazargul Matkerim, Professor-Researcher

Outline

1. **Performance issues and data dependences**

   loop order, overhead, dependences

2. **Sections directive**

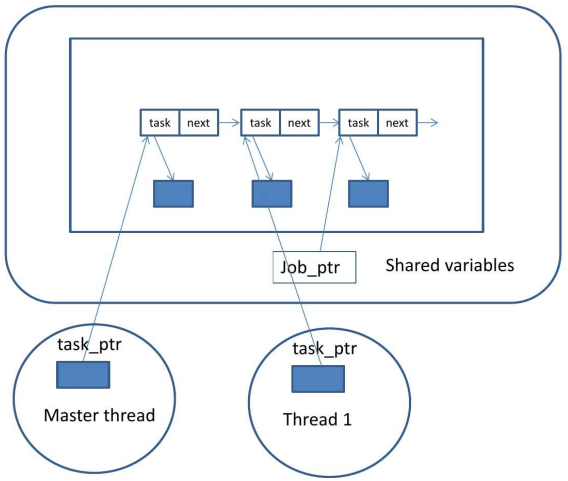
   different tasks for different threads

3. **Synchronization**

   critical, atomic, barrier, master, single, ordered, locks

4. **Data storage attributes and flush**

   shared, private, threadprivate, flush



Source: Lecture 14 material, slide 75

Performance Issues

C/C++ stores matrices in **row-major** order

Loop interchange may increase **cache locality**

Parallelize the **outer-most** loop

- inner loop: a new parallel region in every outer iteration
- this causes parallelization overhead

Avoid parallel overhead at low iteration counts

Good

```
#pragma omp parallel for
for (i = 0; i < N; i++)
  for (j = 0; j < M; j++)
    A[i][j] = B[i][j] + C[i][j];
```

Overhead

```
for (i = 0; i < N; i++) {
  #pragma omp parallel for
  for (j = 0; j < M; j++)
    A[i][j] = B[i][j] + C[i][j];
}
```

Source: Lecture 14 material, slides 52–53

Data Dependences

Two statements use the same memory location and one **writes**

Loop-carried: between different iterations
- such a loop cannot be parallelized directly

Types:
- flow: write in S1, read in S2
- anti: read in S1, write in S2
- output: write in S1 and S2

```
/* loop-carried flow dependence */
for (i = 1; i < N; i++)
    a[i] = a[i-1] + b[i];
```

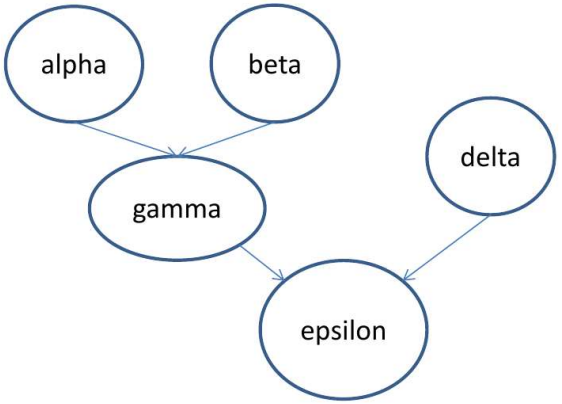
a[i] needs a[i-1] from the previous iteration: the result is wrong in parallel

```
/* anti-dependence removed with a copy */
for (i = 0; i < N-1; i++) a2[i] = a[i+1];
#pragma omp parallel for
for (i = 0; i < N-1; i++)
    a[i] = a2[i] + 1;
```

Source: Lecture 14 material, slides 58–59

Sections Directive

```
#pragma omp parallel sections
{
  #pragma omp section
    v = alpha();
  #pragma omp section
    w = beta();
  #pragma omp section
    y = delta();
}
x = gamma(v, w);
printf("%f\n", epsilon(x, y));
```



Source: Lecture 14 material “Introduction to OpenMP (Part 2)”, slide 67

Each section is executed by **one thread**

Implicit **barrier** at the end (unless nowait)

alpha, beta and delta may be executed in parallel

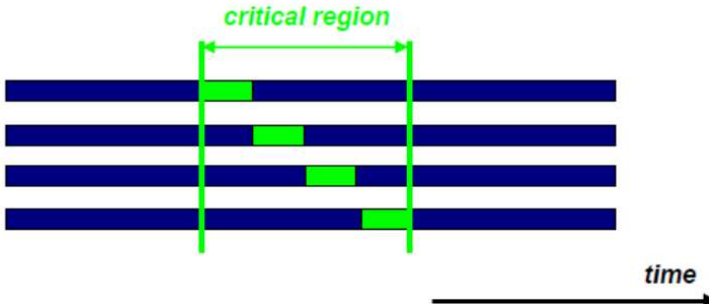
Synchronization: critical

Unsynchronized access to shared data: **race condition**

High level synchronization:
critical, atomic, barrier, ordered

Low level: flush, locks

critical: only one thread at a time



Source: Lecture 14 material “Introduction to OpenMP (Part 2)”, slide 73

```
#pragma omp parallel
{
  double B = big_job(id);
  #pragma omp critical
    consume(B, res);
}
```

Threads wait here: only one thread at a time calls consume()

Synchronization: atomic and barrier

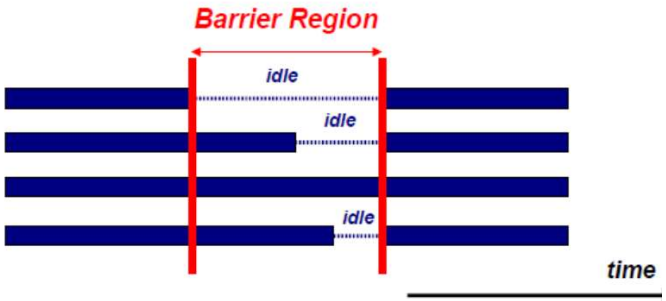
atomic: one memory update, lightweight critical

barrier: each thread waits until all threads arrive

Barriers are expensive: use them only when needed

```
#pragma omp parallel for
for (i = 0; i < N; i++) {
  #pragma omp atomic
    ic = ic + 1;
}
```

```
a[i] = b[i] + c[i];   /* loop 1 */
#pragma omp barrier
d[i] = a[i] + b[i];   /* loop 2 */
```



Source: Lecture 14 material “Introduction to OpenMP (Part 2)”, slide 83

master, single and ordered

master: only the master thread, no implied barrier

single: only one thread, implicit barrier at the end
barrier can be removed with nowait

ordered: executed in sequential loop order

```
#pragma omp parallel
{
  do_many_things();
  #pragma omp single
    { exchange_information(); }
  do_many_other_things();
}
```

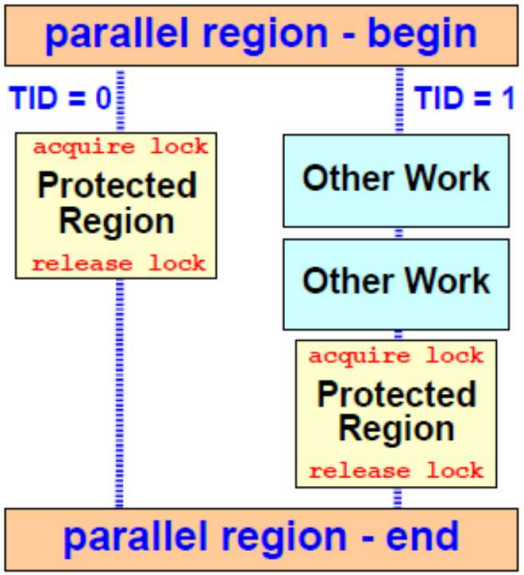
```
#pragma omp parallel for ordered
for (i = 0; i < N; i++) {
  tmp = compute(i);
  #pragma omp ordered
    res += consum(tmp);
}
```

Source: Lecture 14 material, slides 86–90

Lock Routines

1. Define: omp_lock_t lck;
2. Initialize: omp_init_lock(&lck)
3. Set: omp_set_lock or omp_test_lock
4. Unset: omp_unset_lock
5. Remove: omp_destroy_lock

Only one thread holds the lock: **protected region**



Source: Lecture 14 material “Introduction to OpenMP (Part 2)”, slide 93

Data Storage Attributes

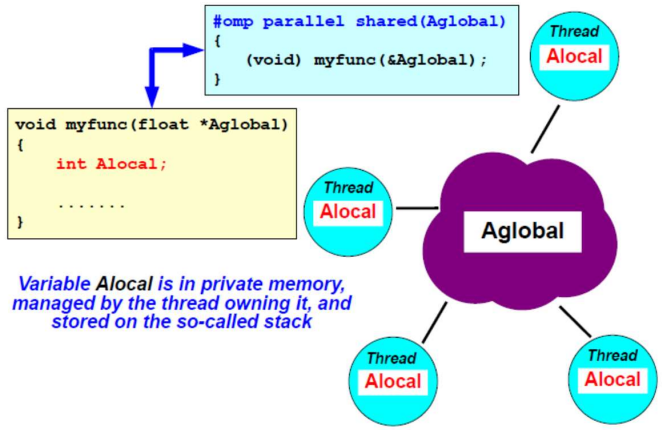
shared: one location for all threads
global and static variables by default

private: a copy for each thread, on its stack

firstprivate / lastprivate

threadprivate: global variable private to each thread

default(none): declare every variable



Source: Lecture 14 material “Introduction to OpenMP (Part 2)”, slide 106

Flush Directive

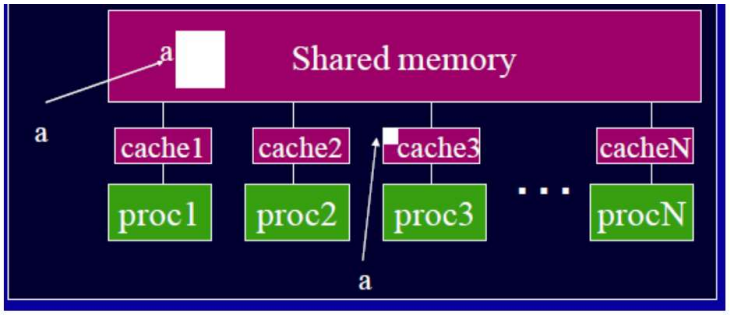
Processors have local memory: registers and cache

An update is **not immediately visible** to other threads

#pragma omp flush (list)

makes the thread's view consistent with memory

Implicit flush at **barriers** and region ends



Source: Lecture 14 material "Introduction to OpenMP (Part 2)", slide 115