Module 10: Open Multi-Processing

Lecture 20: The “omp sections” Directive

The Lecture Contains:

Previous Next

Module 10: Open Multi-Processing

Lecture 20: The “omp sections” Directive

The “omp sections” Directive

Clauses Supported: “sections” Directive

- Private(list)
- Lastprivate(list)
- Firstprivate(list)
- Reduction(operator:list)
- Nowait

Questions

What happens if the number of threads and the number of sections are different? More threads than sections?

Example 1

A Parallel Section Example

```
#pragma omp parallel shared(n,a,b,c,d) private(i)
{
#pragma omp sections nowait
{
#pragma omp section
for (i = 0; i < n-1; i++)
b[i] = (a[i] + a[i+1])/2;
#pragma omp section
for (i = 0; i < n; i++)
d[i] = 1.0/c[i];
} /*– End of sections –*/
} /*– End of parallel region –*/
```

- By default, there is a barrier at the end of the “omp sections”. Use of “nowait” clause turns off the barrier

Module 10: Open Multi-Processing

Lecture 20: The “omp sections” Directive

The “single” Directive

- The single directive specifies that the enclosed section is to be executed by only one thread in team

Format

#pragma omp single [clause[[,] clause] ...]
{
< code - block >
}

Clauses Supported: Single Directive

- Private(list)
- Firstprivate(list)
- Nowait

- There is no implied barrier on entry or exit

Work-Sharing Constructs: WORKSHARE Directive

- FORTRAN only
- This directive divides the execution of the enclosed structured block into separate units of work, each of which is executed
  only once.

Format

The structured block must consist of only the following:

Array assignments	WHERE statements
Scalar assignment	WHERE constructs
FORALL statement	Atomic constructs
FORALL constructs	Critical constructs
	Parallel constructs

Previous Next

Module 10: Open Multi-Processing

Lecture 20: The “omp sections” Directive

Combined Parallel Work-sharing Constructs

- OpenMP provides combined Parallel Worksharing directive that are merely shortcuts:
  - PARALLEL DO/parallel for
  - Parallel SECTIONS
- These directives behave same as an individual PARALLEL directive being immediately followed by a separate
  work-sharing directive.
- Most of the rules and clauses that apply to both directives are in effect

```
#pragma omp parallel        #pragma omp parallel for
#pragma omp for             for (....)
for (...)
#pragma omp parallel        #pragma omp parallel sections
#pragma omp sections        {.... }
{...}
```

Synchronization

- Consider a simple example where two threads on two different processors are both trying to increment a variable x at the same time (assume x is initially 0):

```
THREAD 1                    THREAD 2
increment(x){               increment(x){
x = x+1;                    x = x+1;
}                           }
THREAD 1                    THREAD 2
10 LOAD A, (x address)      10 LOAD A, (x address)
20 ADD A, 1                 20 ADD A, 1
30 STORE A, (x address)     30 STORE A, (x address)
```

- What are the possible outputs?

Previous Next

Module 10: Open Multi-Processing

Lecture 20: The “omp sections” Directive

Synchronization

- Consider a simple example where two threads on two different processors are both trying to increment a variable x at the same time (assume x is initially 0):

THREAD 1	THREAD 2
increment(x){	increment(x){
x = x+1;	x = x+1;
}	}
THREAD 1	THREAD 2
10 LOAD A, (x address)	10 LOAD A, (x address)
20 ADD A, 1	20 ADD A, 1
30 STORE A, (x address)	30 STORE A, (x address)

- What are the possible outputs? After exuction of both threads resultant value of x may be 1 for some execution sequence
- To avoid situations like this the incrementation of x must be synchronized between the two threads

Synchronization is used to impose order constraints and to protect access to shared data

High Level Synchronization

- Critical
- Atomic
- Barrier
- Ordered

Low Level Synchronization

- Flush
- Locks

Previous Next

Synchronization Construct: “critical” Directive

Format

#pragma omp critical [name]

- “critical” directive ensures mutual exclusion: Only one thread at a time can enter a critical region
- The optional name enables multiple different CRITICAL regions to exist:
  - Different CRITICAL regions with the same name are treated as the same region.
  - All CRITICAL sections which are unnamed, are treated as the same section.
  - If sum is a shared variable, this loop can not run in parallel

```
for (i = 0; i < N; i++){
.....
sum += a[i];
.....
}
```

- If sum is a shared variable, this loop can not run in parallel

```
for (i = 0; i < N; i++){
.....
sum += a[i];
.....
}
```

- Use of critical section to parallelize the loop

Example

```
for (i = 0; i < N; i++){
.....
#pragma omp critical
sum += a[i];
.....
}
```

Synchronization Constructs: “atomic” Clause

Format: atomic

#pragma omp atomic
statement expression

- The atomic directive specifies that a specific memory location must be updated atomically, rather than letting multiple threads attempt to write to it

Barrier

Why barriers?

- Suppose we run each of these two loops in parallel

```
for( i = 0; i < N; i++)
    a[i] = b[i] + c[i];
for(i = 0; i < N; i++)
    d[i] = a[i] + b[i];
```

- This may give us wrong result
- WHY ?

Barrier

- We should have updated all a[]’s before using them

```
for( i = 0; i < N; i++)
    a[i] = b[i] + c[i];
```

- All threads should wait here for other threads to complete so we need a barrier here

```
for(i = 0; i < N; i++)
    d[i] = a[i] + b[i];
```

- All threads wait at the barrier point and only continue when all threads have reached the barrier point
- If there is the guarantee that the mapping of iterations onto threads is identical for both loops, there will not be a data race in this case

Previous Next

Synchronization Constructs: “barrier” Directive

Format: barrier

```
#pragma omp barrier
```

- A thread will wait at barrier until all other threads have reached that barrier. All threads then resume executing in parallel the code that follows the barrier

Synchronization Constructs: “ordered” directive

- Used within a DO/for loop with and ordered clause
- The ordered directive specifies that iterations of the enclosed loop will be executed in the same order as if they were executed on a serial processor.

Format

```
#pragma omp ordered
structured block
```

Example

```
#pragma omp parallel for ordered
for(I = 0 ; I < N ; I++){
tmp = NEAT STUFF(I);
#pragma omp ordered
res += consume(tmp);
}
```

Synchronization Constructs: “flush” Directive

Format

```
# pragma omp flush (list)
```

- Defines a sequence point at which a thread is guaranteed to see a consistent view of memory with respect to the “flush set”
- The flush set is :
  - List of variables
  - In absence of list all thread visible variable are in list
- Flush forces data to be updated in memory so other threads see the most recent value

Module 10: Open Multi-Processing

Lecture 20: The “omp sections” Directive

Data Environment:

Default storage attributes:

- Shared Memory programming model:
  - Most variables are shared by default
- Global variables are shared among threads
  - Fortran: COMMON blocks, SAVE variables, MODULE variables
  - C: File scope variables, static
  - Both: Dynamically allocated memory (ALLOCATE, malloc, new)
- But not everything is shared...
  - Stack variables in subprograms(Fortran) or functions(C) called from parallel regions are private
  - Automatic variables within a statement block are private

Example: Data Sharing

```
double A[10];                  extern double A [ 10 ];
int main() {                   void work(int index) {
int index [ 10 ] ;             double temp[10];
#pragma omp parallel           static int count;
work(index);                   ...
printf(“%d”, index[0]);        }
}
```

- Which variables are “shared” and “private” ?
- Which variables are “shared” and “private” ?
- A index and count are shared by all threads
- Variable temp is local to each thread

Changing Storage Attributes

It is possible to change storage attribute of data for constructs using OpenMP clauses

Previous Next

Data Scope Attribute Clauses

The Data Scope Attribute Clauses are used to explicitly define how variables should be scoped.They include :

- Private
- Firstprivate
- Lastprivate
- Shared
- Default
- Reduction
- Copyin

“private” Clause

Purpose: The private clause declares variables in its list to be private to each thread
Format: private (list)

- A new object of the same type is declared for each thread in the team
- Private(var) creates a new local copy of var for each thread

Example: “private” Clause

```
void wrong() {
int tmp = 0;
#pragma omp parallel for private(tmp)
for (int j = 0; j < 1000; ++j)
tmp += j;
printf(“%d”, tmp);
}
```

- What is wrong with the code ?
- Value of tmp was not initialized at line 5
- What value of tmp it will be printed in second last line ?
- In OpenMP 2.5 the value of the shared variable is undefined after the region So tmp is unspecified for OpenMP 2.5 while 0 in OpenMP 3.0

Previous Next

Data Scope Attribute Clauses

“firstprivate” Clause

Purpose: The firstprivate clause combines the behavior of the private clause with automatic initialization of the variables in the list
Format: firstprivate (list)

- Special case of private clause
- Initializes each private copy with the corresponding value from the master thread prior to entry into the parallel or work-sharing construct

The “firstprivate” Clause

Example: “firstprivate” Clause

```
void useless() {
int tmp = 0;
#pragma omp parallel for firstprivate(tmp)
for (int j = 0; j ¡ 1000; ++j)
tmp += j;
printf(“%d”, tmp);
}
```

- Each thread gets its own code with initial value 0.Is there something still wrong with the code ?
- What value of tmp it will be printed in second last line ?
- Tmp is unspecified for OpenMP 2.5 while 0 in OpenMP 3.0

“lastprivate” Clause

Purpose: The lastprivate clause combines the behavior of the private clause with a copy from the last loop iteration or section to the original variable object.
Format: lastprivate (list)

- The value copied back into the original variable object is obtained from the last (sequentially) iteration or section of the enclosing construct.

Previous Next