OpenMP Overview

For Parallel Computing Overview Tutorial
OpenMP: Conceptual Overview

- Serial “process” and process address space
- OpenMP parallel region & threads
- Worksharing constructs
- Shared and Private variables
- OpenMP compiler directives
- OpenMP functions
- OpenMP build & runtime: tools & environments
- Documentation & resources
Serial Process

Program starts
Memory Address space
Variables
code
Single thread
Sequential operations
Process ends

PROGRAM serial_example_1

Serial / Sequential Code

stop
ebd

© 2009 Regents of the University of Minnesota. All rights reserved.
Parallel with OpenMP Threads

Process starts
Single “master” thread

Fork
Concurrent execution
Independent threads
Shared variables
Private variables

Join
Back to single thread
Process ends

PROGRAM threaded_example_1

!$OMP PARALLEL

Serial/sequential

OpenMP Parallel Region

!$OMP END PARALLEL

Serial/sequential

stop

ebd
Serial Application

Program starts
Memory Address space
Variables
  code
Thread of execution
  Sequential operations
Process ends

```
PROGRAM serial_example_2
real a(1000), b(1000), c(100)

4000 bytes for a, b, and c

c(1) = a(1) * b(1)
c(2) = a(2) * b(2)
...
c(1000) = a(1000) * b(1000)

do i = 1, 1000
  c(i) = a(i) * b(i)
enddo

stop

```

© 2009 Regents of the University of Minnesota. All rights reserved.
Parallel But Redundant

Process starts
Allocate shared arrays

Concurrent threads
Parallel execution
REDUNDANT Work

Process ends

PROGRAM threaded_example_bad
real a(1000), b(1000), c(100)

!$OMP PARALLEL shared(a,b,c)

do i = 1, 1000
  c(i) = a(i) * b(i)
endo

!$OMP END PARALLEL
stop

© 2009 Regents of the University of Minnesota. All rights reserved.
PROGRAM threaded_example_good
real a(1000), b(1000), c(100)
!
$OMP PARALLEL shared(a,b,c)
!
$OMP DO
do i = 1, 1000
   c(i) = a(i) * b(i)
endo
!
$OMP END DO
!
$OMP END PARALLEL
!
stop
!

© 2009 Regents of the University of Minnesota. All rights reserved.
**Work Sharing: SECTIONS**

Inside Parallel region

Threads do different sections of code.

Max threads used: = number of sections

Fork 3 threads

A(1) = ... A(2) = ... ...

B(1) = ... B(2) = ... ...

C(1) = ... C(2) = ... ...

Join thread back to 1

!$OMP PARALLEL shared(A,B,C)

!$OMP SECTIONS

!$OMP SECTION

A(:) = ...

!$OMP SECTION

B(:) = ...

!$OMP SECTION

C(:) = ...

!$OMP END SECTION

!$OMP END PARALLEL
Work Sharing: SINGLE

Inside Parallel region

Only one thread calls BBB().

Usually 1st thread to reach section of code will do it.
Supercomputing Institute
for Advanced Computational Research

Work Sharing: MASTER

Inside Parallel region
Only MASTER thread calls BBB().
MASTER thread:
  Initial thread of app
  Thread 0
  Full context of app

Fork 3 threads
  call AAA()  call AAA()  call AAA()
  call BBB()  
  call CCC()  call CCC()  call CCC()

Join thread back to 1

 !$OMP PARALLEL
  call AAA()

 !$OMP MASTER
  Call BBB()

 !$OMP END MASTER
  call CCC()

 !$OMP END PARALLEL

© 2009 Regents of the University of Minnesota. All rights reserved.
inside Parallel region

Threads stop at barrier and wait for others.

Fork 3 threads

$omp parallel

call AAA()
call AAA()
call AAA()

$omp barrier

call CCC()
call CCC()
call CCC()

$omp end parallel

Join thread back to 1

© 2009 Regents of the University of Minnesota. All rights reserved.
# OpenMP Scope of Variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SHARED</strong></td>
<td>Variable is same on each thread&lt;br&gt;Same memory location used $\Rightarrow$ no new memory allocated&lt;br&gt;Value set by one thread is seen by all threads</td>
</tr>
<tr>
<td><strong>PRIVATE</strong></td>
<td>Variable is different on each thread&lt;br&gt;Different memory locations used $\Rightarrow$ memory allocated for each thread&lt;br&gt;Changes to variable values invisible between threads</td>
</tr>
<tr>
<td><strong>THREADPRIVATE</strong></td>
<td>Private variables with values that persist from between parallel regions&lt;br&gt;New memory allocated only when variable 1st appears in a parallel region</td>
</tr>
</tbody>
</table>

**Default Variable Scopes:**

- Declared **outside** parallel region: **shared**
- Declared **inside** parallel region: **private**
## OMP PARALLEL DIRECTIVE

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>C / C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>$OMP PARALLEL [clauses] CODE BLOCK $OMP END PARALLEL</td>
<td>#pragma omp parallel [clause ...]</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>structured block</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

### Clauses:

- **shared (list)**: Variable in list are the same (shared) between threads
- **private (list)**: Variable in list are different from thread to thread
- **firstprivate (list)**: Private with initial value copied in
- **default(shared | none)**: C/C++
- **default(PRIVATE | FIRSTPRIVATE | SHARED | NONE)**: FORTRAN
OMP PARALLEL DIRECTIVE

Clauses (continued):

num_threads (integer-expression)  Specifies number of threads in region

reduction (operator: list)  list=operator(list) returned to MASTER
if (scalar_expression)  Run region non-parallel if FALSE

Private variables & common blocks that persist between parallel regions:

threadprivate (list)  Private variable with global scope in code
copyin (list)  Copies values from MASTER thread to threadprivate
### OMP DO Directive

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>C / C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>!$OMP DO [clauses]</td>
<td>#pragma omp do [clause …]</td>
</tr>
<tr>
<td>DO_LOOP</td>
<td>for_loop { … }</td>
</tr>
<tr>
<td>!$OMP END PARALLEL [nowait]</td>
<td></td>
</tr>
</tbody>
</table>

**Clauses:**
- `schedule (type [,chunk])`  
- `ordered`  
- `lastprivate (list)`  
- `collapse (n)`  
- `nowait`  

- `static | dynamic`, `size of index blocks`  
- `force sequential ordering of indices`  
- `copy values in list from those of last loop index`  
- `collapse nested loops (n deep) into single loop`  
- `allow threads to continue before all complete loop`
## OMP SECTION DIRECTIVES

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>C / C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>!$OMP SECTIONS [clauses]</td>
<td>#pragma omp sections</td>
</tr>
<tr>
<td>!$OMP SECTION CODE_BLOCK</td>
<td>{</td>
</tr>
<tr>
<td>!$OMP SECTION CODE_BLOCK</td>
<td>#pragma omp section</td>
</tr>
<tr>
<td>...</td>
<td>init_field(field);</td>
</tr>
<tr>
<td>!$OMP SECTION CODE_BLOCK</td>
<td>check_grid(grid);</td>
</tr>
<tr>
<td>!$OMP END SECTION</td>
<td>}</td>
</tr>
</tbody>
</table>

**Relevant Clauses:**
- private (list)
- firstprivate (list)
- lastprivate (list)
- reduction (operator: list)
real*4 a(0:301), b(0:301)
a(:) = 0.0

!$OMP PARALLEL
SHARED(a, b)

!$OMP DO
do i = 1, 300
a(i) = float(i)
enddo

!$OMP END DO

!$OMP END PARALLEL
## OpenMP Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMP_GET_NUM_THREADS()</td>
<td>Returns an Integer: number of threads in current section</td>
</tr>
<tr>
<td>OMP_GET_THREAD_NUM()</td>
<td>Returns an Integer: thread number ([0,\ldots,N-1])</td>
</tr>
<tr>
<td>OMP_SET_THREAD_NUM(N)</td>
<td>Set the default thread number for the next section to be (N)</td>
</tr>
<tr>
<td>OMP_WTIME()</td>
<td>Returns Double Precision wall clock time</td>
</tr>
</tbody>
</table>

There are many more functions

Most get or set properties of parallel regions or loops.
OpenMP Thread Control with Functions

**Take full control**

- Maximum flexibility
- Minimum overhead
- Minimal synchronization
- Complete transparency

integer `OMP_GET_NUM_THREADS`

```c
!$OMP PARALLEL private(k,j1,j2)
```

```c
k = OMP_GET_THREAD_NUM()
```

```c
j1 = 1 + 100 * j
```

```c
j2 = 100 * (j+1)
```

```c
!$OMP END PARALLEL
```

- `OMP_GET_THREAD_NUM()`
  - Get thread number

- `OMP_GET_NUM_THREADS()`
  - Get number of threads

- `OMP_SET_NUM_THREADS(n)`
  - Set default number of threads

```
fork 3 threads
```

```
j0 = 0
j1 = 1
j2 = 100
```

```
k = 0
j1 = 1
j2 = 100
```

```
k = 1
j1 = 101
j2 = 200
```

```
k = 2
j1 = 201
j2 = 300
```

```
join thread back to 1
```

© 2009 Regents of the University of Minnesota. All rights reserved.
A Full Small Example

Making all threads say “Hello World”, in C.

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

int main(void) {
    printf("\n");
    #pragma omp parallel
    {
        const int thread_num = omp_get_thread_num();
        const int threads_total = omp_get_num_threads();
        printf("Hello World from thread = %d out of %d total threads.\n", \n               thread_num, threads_total);
    }
    printf("\n");
    return 0;
}
```
# OpenMP: Environment, Build, & Run

<table>
<thead>
<tr>
<th>Modules</th>
<th>Intel</th>
<th>GNU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>module load intel</td>
<td>module load gcc/7.2.0</td>
</tr>
</tbody>
</table>

## Build

<table>
<thead>
<tr>
<th></th>
<th>Intel</th>
<th>GNU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ifort -qopenmp code.f -o app</td>
<td>gfortran -fopenmp code.f -o app</td>
</tr>
<tr>
<td></td>
<td>icc -qopenmp code.c -o app</td>
<td>gcc -fopenmp code.c -o app</td>
</tr>
</tbody>
</table>

## Shell Variables

<table>
<thead>
<tr>
<th>Variables</th>
<th>Intel</th>
<th>GNU</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMP_NUM_THREADS=24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KMP_AFFINITY=granularity=core,scatter</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Run

<table>
<thead>
<tr>
<th></th>
<th>Intel</th>
<th>GNU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>./app</td>
<td>./app</td>
</tr>
<tr>
<td></td>
<td>numactl --interleave=0,1 ./app</td>
<td></td>
</tr>
</tbody>
</table>

- Run on MSI’s HPC systems
- This is a sample. See documentation on next slide
OpenMP Documentation & Resources

Tutorial:  https://computing.llnl.gov/tutorials/openMP/

Summary:  http://www-inst.eecs.berkeley.edu/~cs61c/resources/OpenMP3.0-SummarySpec.pdf


GNU:  https://gcc.gnu.org/onlinedocs/libgomp/index.html#Top

Specifications (OpenMP Versions 2.5 - 5.0):  
https://www.openmp.org/specifications/