Topics in Computing for Astronomy:

Parallel Programming with OpenMP (I)

Rich Townsend, University of Wisconsin-Madison
Preliminaries

- The Mad SDK (contains gfortran, an OpenMP-capable Fortran 90/95/2003/2008 compiler)
  

- This talk & accompanying code examples:
  
What is OpenMP?

- An Application Programming Interface (API) for writing multithreaded programs
- Consists of a set of compiler directives, library routines and environment variables
- Supports Fortran, C and C++
- Developed by vendors (AMD, IBM, Intel, ...)
- Mainly targeted at Symmetric Multiprocessing systems
What is a Symmetric Multiprocessing (SMP)?

- A multiprocessor system with
  - centralized, shared memory
  - a single operating system
  - two or more homogeneous processors

- Modern desktops/laptops are SMP systems:
  - Multiple cores in each physical processor
  - Multiple physical processors
What is Multithreading?

- The ability of a program to have multiple, independent *threads* executing different sequences of instructions
- The threads can access the resources (e.g., memory) of the parent process
- The threads run concurrently on a multiprocessor systems
OpenMP Core Syntax

- Constructs use compiler directives specified using special Fortran comments:
  
<table>
<thead>
<tr>
<th>Fortran 77</th>
<th>Fortran 90+</th>
</tr>
</thead>
<tbody>
<tr>
<td>C$omp parallel do</td>
<td>!$omp parallel</td>
</tr>
</tbody>
</table>

- Most constructs apply to a “structured block” with one point of entry and one point of exit
- Additional library functions and subroutines accessed through the omp_lib module
Example: Hello World

program hello_world
    use omp_lib
    implicit none

    !$omp parallel num_threads(4)
    print *, 'Hello from thread', omp_get_thread_num()
    !$omp end parallel

end program hello_world

% gfortran -fopenmp -o hello_world hello_world.f90
% ./hello_world
Hello from thread 0
Hello from thread 2
Hello from thread 3
Hello from thread 1
Example: Hello World

```fortran
program hello_world
  use omp_lib
  implicit none

  !$omp parallel num_threads(4)
  print *, 'Hello from thread', omp_get_thread_num()
  !$omp end parallel

end program hello_world
```

% gfortran -fopenmp -o hello_world hello_world.f90
% ./hello_world
Hello from thread 0
Hello from thread 2
Hello from thread 3
Hello from thread 1
Example: Hello World

```fortran
program hello_world
  use omp_lib
  implicit none

  !$omp parallel num_threads(4)
  print *, 'Hello from thread', omp_get_thread_num()
  !$omp end parallel

end program hello_world
```

% gfortran -fopenmp -o hello_world hello_world.f90
% ./hello_world

Hello from thread 0
Hello from thread 2
Hello from thread 3
Hello from thread 1
Example: Hello World

```fortran
program hello_world
    use omp_lib
    implicit none
    !$omp parallel num_threads(4)
    print *, 'Hello from thread', omp_get_thread_num()
    !$omp end parallel
end program hello_world
```

% gfortran -fopenmp -o hello_world hello_world.f90
% ./hello_world
Hello from thread           0
Hello from thread           2
Hello from thread           3
Hello from thread           1
Example: Hello World

program hello_world
    use omp_lib
    implicit none

    !$omp parallel num_threads(4)
    print *, 'Hello from thread', omp_get_thread_num()
    !$omp end parallel

end program hello_world

% gfortran -fopenmp -o hello_world hello_world.f90
% ./hello_world
Hello from thread 0
Hello from thread 2
Hello from thread 3
Hello from thread 1
Example: Hello World

```fortran
program hello_world

use omp_lib
implicit none

!$omp parallel num_threads(4)
print *, 'Hello from thread', omp_get_thread_num()
!$omp end parallel

end program hello_world
```

% gfortran -fopenmp -o hello_world hello_world.f90
% ./hello_world
Hello from thread 0
Hello from thread 2
Hello from thread 3
Hello from thread 1

Not sequential — OpenMP does not guarantee execution order!
The Fork-Join Thread Model

- At the start of a program, there is a single thread — the *master thread*
- When entering a parallel region, the master thread ‘forks’ into a team of threads
- When exiting a parallel region, the team joins back into a single master thread
Sharing Variables

program hello_world_shared
use omp_lib
implicit none
integer :: id
!
$omp parallel num_threads(4)
id = omp_get_thread_num()
print *, 'Hello from thread', id
$omp end parallel
!
end program hello_world_shared

This overwrites the same memory location!
Sharing Variables

```fortran
program hello_world_shared

  use omp_lib
  implicit none

  integer :: id

  !$omp parallel num_threads(4)
  id = omp_get_thread_num()
  print *, 'Hello from thread', id
  !$omp end parallel

end program hello_world_shared
```

The `id` variable is by default shared amongst threads.

This overwrites the same memory location!
Not Sharing Variables

```fortran
program hello_world_private
  use omp_lib
  implicit none

  integer :: id

  !$omp parallel num_threads(4) private(id)
  id = omp_get_thread_num()
  print *, 'Hello from thread', id
  !$omp end parallel

end program hello_world_private
```
Not Sharing Variables

program hello_world_private

use omp_lib
implicit none

integer :: id

!$omp parallel num_threads(4) private(id)
id = omp_get_thread_num()
print *, 'Hello from thread', id
!$omp end parallel

end program hello_world_private

hello_world_private.f90

The *id* variable is private; each thread has its own copy
program count_threads

   use omp_lib
   implicit none

   integer :: n

   n = 0

   !$omp parallel num_threads(4)
   n = n + 1
   !$omp end parallel

   print *, 'There were', n, 'threads'

end program count_threads
program count_threads

    use omp_lib
    implicit none

    integer :: n

    n = 0

    !$omp parallel num_threads(4)
    n = n + 1
    !$omp end parallel

    print *, 'There were', n, ' threads'

end program count_threads
program count_threads_atomic

use omp_lib
implicit none

integer :: n

n = 0

!$omp parallel num_threads(4)
!$omp atomic
n = n + 1
!$omp end parallel

print *, 'There were', n, 'threads'

end program count_threads_atomic
program count_threads_atomic
use omp_lib
implicit none

integer :: n

n = 0

!$omp parallel num_threads(4)
!$omp atomic
n = n + 1
!$omp end parallel

print *, 'There were', n, 'threads'
end program count_threads_atomic
Synchronization Directives

- `!$omp atomic` is an example of a synchronization directive

Other examples:
- `!$omp critical` — only execute on one thread at a time
- `!$omp master` — only execute on master thread
- `!$omp barrier` — wait for all threads
program square_nums

use omp_lib
implicit none

integer :: i, j(4)

 !$omp parallel num_threads(4)
i = omp_get_thread_num() + 1
j(i) = i**2
!$omp end parallel

print *, 'Square numbers:', j

end program square_nums
program square_nums

use omp_lib
implicit none

integer :: i, j(4)

!$omp parallel num_threads(4)
i = omp_get_thread_num() + 1
j(i) = i**2
!$omp end parallel

print *, 'Square numbers:', j

end program square_nums
program square_nums_do

  use omp_lib
  implicit none

  integer :: i, j(13)

  !$omp parallel num_threads(4)
  !$omp do
  do i = 1, SIZE(j)
    j(i) = i**2
  enddo
  !$omp end parallel

  print *, 'Square numbers:', j

end program square_nums_do
Parallelizing DO loops

program square_nums_do

  use omp_lib
  implicit none

  integer :: i, j(13)

  !$omp parallel num_threads(4)
  !$omp do
  do i = 1, SIZE(j)
    j(i) = i**2
  enddo
  !$omp end parallel

  print *, 'Square numbers:', j

end program square_nums_do
Parallelizing DO loops

```fortran
program square_nums_do
  use omp_lib
  implicit none

  integer :: i, j(13)

  !$omp parallel num_threads(4)
  !$omp do
  do i = 1, SIZE(j)
    j(i) = i**2
  enddo
  !$omp end parallel

  print *, 'Square numbers:', j
end program square_nums_do
```

- The program begins by importing the OpenMP library.
- It declares an integer array `j` of size 13.
- The `do` directive is used to parallelize the loop, distributing iterations amongst 4 threads.
- The `print` statement outputs the squared numbers.
program square_nums_pardo

use omp_lib
implicit none

integer :: i, j(13)

!$omp parallel do num_threads(4)
do i = 1, SIZE(j)
  j(i) = i**2
enddo
!$omp end parallel do

print *, 'Square numbers:', j

end program square_nums_pardo
program square_nums_pardo

use omp_lib
implicit none

integer :: i, j(13)

!$omp parallel do num_threads(4)
do i = 1, SIZE(j)
  j(i) = i**2
endo
!$omp end parallel do

print *, 'Square numbers:', j

end program square_nums_pardo
Work-Sharing Constructs

- `!$omp do` is an example of a work-sharing construct.
- A work-sharing construct divides the execution of the enclosed region among the members of the thread team that encounters it.

Other examples:
- `!$omp sections` — divide multiple enclosed blocks amongst threads of team
- `!$omp workshare` — divide single enclosed block amongst threads of team
- `!$omp single` — execute all work on a single thread
program square_nums_workshr

    use omp_lib
    implicit none

    integer :: i, j(13)

    j = [(i,i=1,SIZE(j))]

    !$omp parallel workshare num_threads(4)
    j = j**2
    !$omp end parallel workshare

    print *,'Square numbers:', j

end program square_nums_workshr
The workshare Construct

program square_nums_workshr

    use omp_lib
    implicit none

    integer :: i, j(13)

    j = [(i,i=1,SIZE(j))]

    !$omp parallel workshare num_threads(4)
    j = j**2
    !$omp end parallel workshare

    print *, 'Square numbers:', j

end program square_nums_workshr
The workshare Construct

program square_nums_workshr

    use omp_lib
    implicit none

    integer :: i, j(13)

    j = [(i,i=1,SIZE(j))]

    !$omp parallel workshare num_threads(4)
    j = j**2
    !$omp end parallel workshare

    print *,'Square numbers: ', j

end program square_nums_workshr
Restrictions on workshare Constructs

- Only the following operations permitted:
  - array assignments
  - scalar assignments
  - FORALL statements
  - FORALL constructs
  - WHERE statements
  - WHERE constructs
  - atomic constructs
  - critical constructs
  - parallel constructs
The sections Construct

program hello_sections

use omp_lib
implicit none

!$omp parallel sections num_threads(4)
!$omp section
print *,'Hello from thread', omp_get_thread_num()
!$omp section
print *,'Hola from thread', omp_get_thread_num()
!$omp section
print *,'Aloha from thread', omp_get_thread_num()
!$omp end parallel sections

end program hello_sections
The sections Construct

program hello_sections

use omp_lib
implicit none

 !$omp parallel sections num_threads(4)
 !$omp section
 print *, 'Hello from thread', omp_get_thread_num()
 !$omp section
 print *, 'Hola from thread', omp_get_thread_num()
 !$omp section
 print *, 'Aloha from thread', omp_get_thread_num()
 !$omp end parallel sections

end program hello_sections