OpenMP Overview 1

Getting Up To Speed On OpenMP 4.0

Ruud van der Pas
ORACLE
Distinguished Engineer
Architecture and Performance
SPARC Microelectronics, Santa Clara, CA, USA

HPCS 2015
Amsterdam, The Netherlands
July 20, 2015

OpenMP Overview 2

Agenda

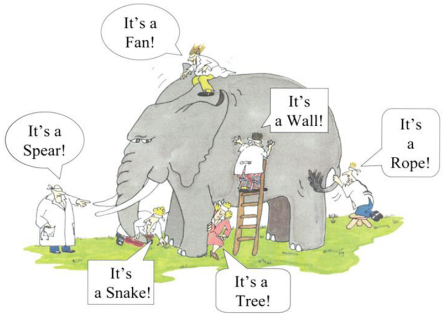
- The One Hour Elevator Ride
- Using OpenMP
  - Directives
  - Environment Variables
  - Run time Functions
- Additional Functionality
- ~~Performance Tuning Case Studies~~

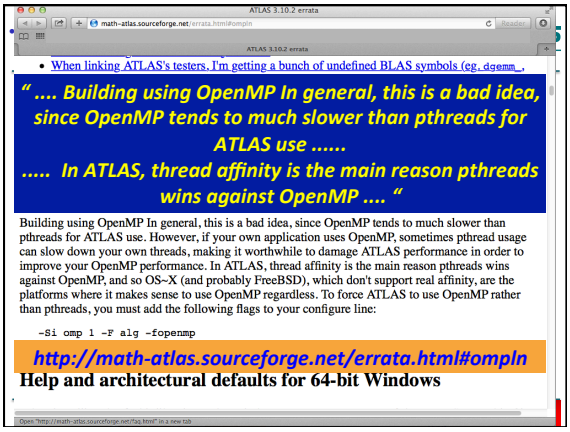
OpenMP Overview 3

The One Hour Elevator Ride

OpenMP Overview 4

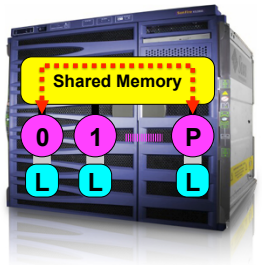
What Is OpenMP ?





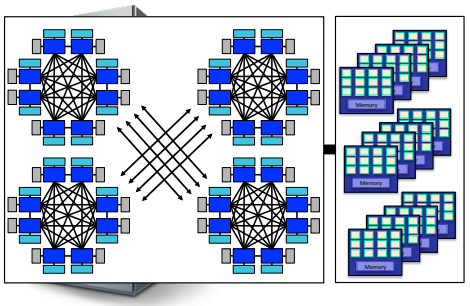
OpenMP Overview 6

1997 - 2013



OpenMP Overview 7

> July 2013



V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 8

OpenMP™

http://www.openmp.org
http://www.iwomp.org
http://www.openmpcon.org
http://www.compunity.org

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 10

Shameless Plug

OpenMP

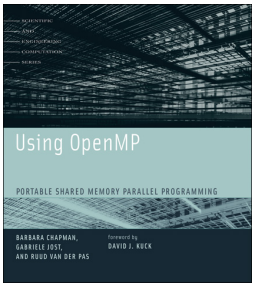
"Using OpenMP"
Portable Shared Memory Parallel Programming

Chapman, Jost, van der Pas

MIT Press, 2008

ISBN-10: 0-262-53302-2
ISBN-13: 978-0-262-53302-7

List price: ~35 $US



V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

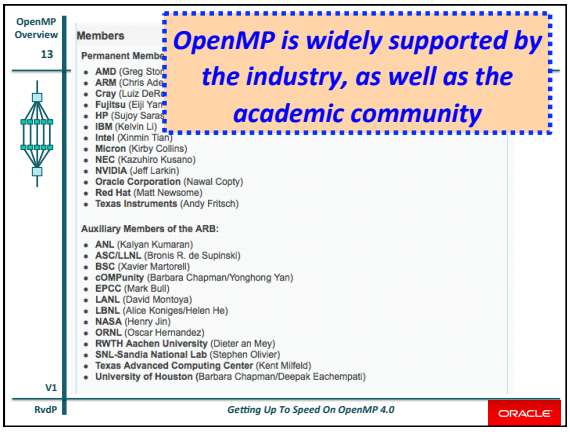
OpenMP Overview 12

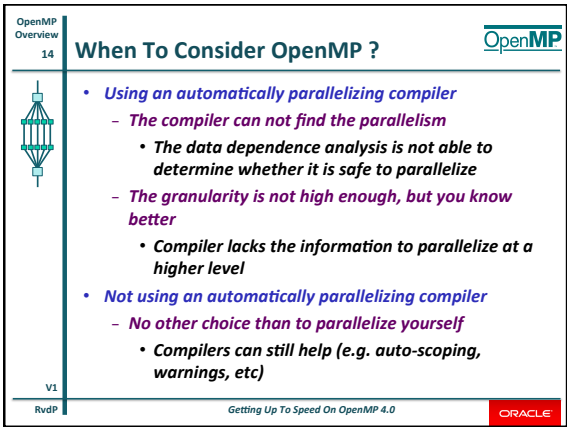
What Is OpenMP ?

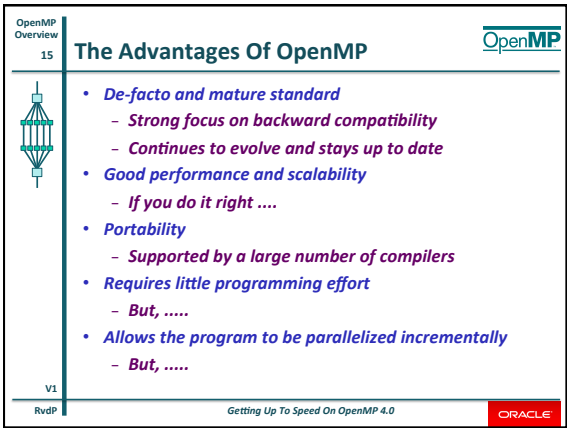
OpenMP

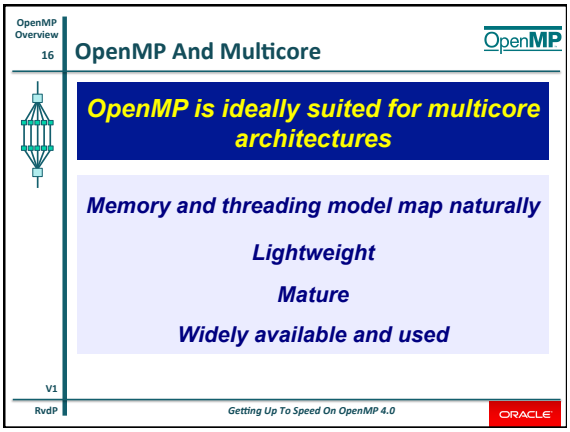
- *The de-facto standard Application Programming Interface (API) to write shared memory parallel applications in C, C++, and Fortran*
- *Consists of Compiler Directives, Runtime routines and Environment variables*
- *Specification maintained by the OpenMP Architecture Review Board (http://www.openmp.org)*
- *Current version: 4.0 (released July 2013)*
  - *The draft 4.1 specs are expected later this year*

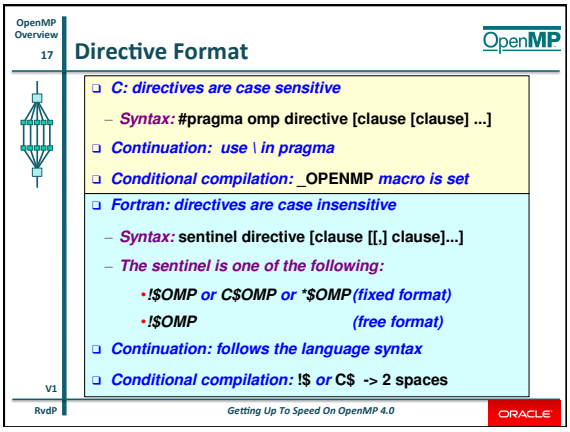
V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

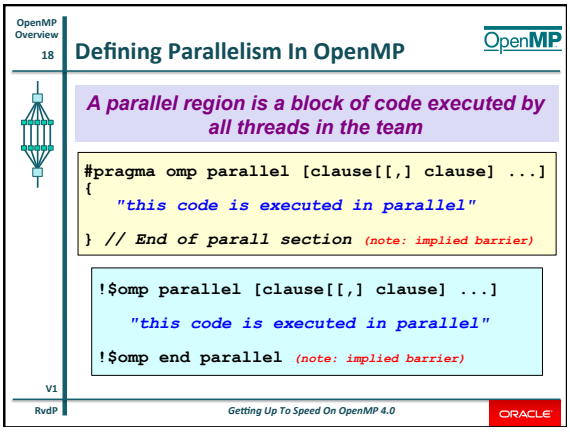

OpenMP Overview
13
Members
Permanent Members
AMD (Greg Stoner)
ARM (Chris Adeniyi-Jones)
Cray (Luiz DeRose)
Fujitsu (Eiji Yamanaka)
HP (Sujoy Saraswati)
IBM (Kelvin Li)
Intel (Xinmin Tian)
Micron (Kirby Collins)
NEC (Kazuhiro Kusano)
NVIDIA (Jeff Larkin)
Oracle Corporation (Nawal Copty)
Red Hat (Matt Newsome)
Texas Instruments (Andy Fritsch)
Auxiliary Members of the ARB:
ANL (Kalyan Kumaran)
ASC/LLNL (Bronis R. de Supinski)
BSC (Xavier Martorell)
cOMPunity (Barbara Chapman/Yonghong Yan)
EPCC (Mark Bull)
LANL (David Montoya)
LBNL (Alice Koniges/Helen He)
NASA (Henry Jin)
ORNL (Oscar Hernandez)
RWTH Aachen University (Dieter an Mey)
SNL-Sandia National Lab (Stephen Olivier)
Texas Advanced Computing Center (Kent Milfeld)
University of Houston (Barbara Chapman/Deepak Eachempati)
OpenMP is widely supported by the industry, as well as the academic community
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
14
When To Consider OpenMP ?
OpenMP
Using an automatically parallelizing compiler
The compiler can not find the parallelism
The data dependence analysis is not able to determine whether it is safe to parallelize
The granularity is not high enough, but you know better
Compiler lacks the information to parallelize at a higher level
Not using an automatically parallelizing compiler
No other choice than to parallelize yourself
Compilers can still help (e.g. auto-scoping, warnings, etc)
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
15
The Advantages Of OpenMP
OpenMP
De-facto and mature standard
Strong focus on backward compatibility
Continues to evolve and stays up to date
Good performance and scalability
If you do it right ....
Portability
Supported by a large number of compilers
Requires little programming effort
But, ...
Allows the program to be parallelized incrementally
But, ...
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
16
OpenMP And Multicore
OpenMP
OpenMP is ideally suited for multicore architectures
Memory and threading model map naturally
Lightweight
Mature
Widely available and used
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
17
Directive Format
OpenMP
C: directives are case sensitive
Syntax: #pragma omp directive [clause [clause] ...]
Continuation: use \ in pragma
Conditional compilation: _OPENMP macro is set
Fortran: directives are case insensitive
Syntax: sentinel directive [clause [[,] clause]...]
The sentinel is one of the following:
!$OMP or C$OMP or *$OMP (fixed format)
!$OMP (free format)
Continuation: follows the language syntax
Conditional compilation: !$ or C$ -> 2 spaces
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
18
Defining Parallelism In OpenMP
OpenMP
A parallel region is a block of code executed by all threads in the team
#pragma omp parallel [clause[[,] clause] ...]
{
"this code is executed in parallel"
} // End of parall section (note: implied barrier)
!$omp parallel [clause[[,] clause] ...]
"this code is executed in parallel"
!$omp end parallel (note: implied barrier)
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE

OpenMP Overview 19

Parallel Region - An Example/1

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {


          printf("Hello World\n");


   return(0);
}
```


OpenMP Overview 20

Parallel Region - An Example/2

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {

   #pragma omp parallel
   {
          printf("Hello World\n");

   } // End of parallel region

   return(0);
}
```


OpenMP Overview 21

Parallel Region - An Example/3

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
Hello World
Hello World
$ export OMP_NUM_THREADS=4
$ ./a.out
Hello World
Hello World
Hello World
Hello World
$
```


OpenMP Overview 22

The If Clause

if (scalar expression)

- ✓ *Only execute in parallel if expression evaluates to true*
- ✓ *Otherwise, execute serially*

```
#pragma omp parallel if (n > some_threshold) \
        shared(n,x,y) private(i)
  {
    #pragma omp for
     for (i=0; i<n; i++)
        x[i] += y[i];
  } /*-- End of parallel region --*/
```


OpenMP Overview 23

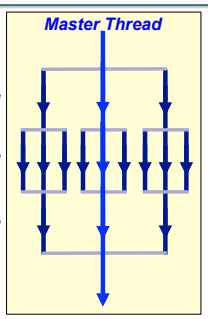
A More Elaborate Example

```
#pragma omp parallel if (n>limit) default(none) \
        shared(n,m,a,b,c,x,y,z) private(f,i,scale)
{
    f = 1.0;                          <- Statement is executed by all threads

#pragma omp for nowait
    for (i=0; i<n; i++)               <- parallel loop (work is distributed)
       z[i] = x[i] + y[i];

#pragma omp for nowait
    for (i=0; i<m; i++)               <- parallel loop (work is distributed)
       a[i] = b[i] + c[i];
          ....

#pragma omp barrier                   <- synchronization
    scale = sum(a,0,m) + sum(z,0,n) + f;   <- Statement is executed by all threads
          ....

} /*-- End of parallel region --*/
```

parallel region

OpenMP Overview 24

Nested Parallelism



3-way parallel — *Outer parallel region*

9-way parallel — *Nested parallel region*

3-way parallel — *Outer parallel region*

Note: Nesting level can be arbitrarily deep

OpenMP Overview 25

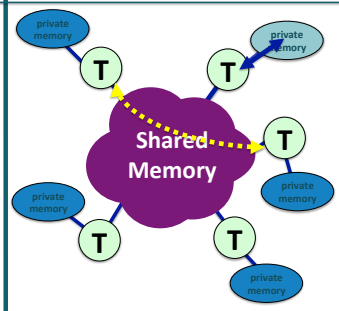
Components Of OpenMP

Directives	Environment variables	Runtime functions
Worksharing	Thread Settings	Thread Management
Tasking	Thread Controls	Work Scheduling
Affinity	Work Scheduling	Tasking
Accelerators	Affinity	Affinity
Cancellation	Accelerators	Accelerators
Synchronization	Cancellation	Cancellation
	Operational	Locking

RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 26

The OpenMP Memory Model



- All threads have access to the same, globally shared memory
- Data in private memory is only accessible by the thread owning this memory
- No other thread sees the change(s) in private memory
- Data transfer is through shared memory and is 100% transparent to the application

V1
RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 27

Data-Sharing Attributes

- In an OpenMP program, data needs to be "labeled"
- Essentially there are two basic types:
  - Shared - There is only one instance of the data
    - Threads can read and write the data simultaneously unless protected through a specific construct
    - All changes made are visible to all threads
      - But not necessarily immediately, unless enforced .....
  - Private - Each thread has a copy of the data
    - No other thread can access this data
    - Changes only visible to the thread owning the data

V1
RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 28

The Private And Shared Clauses

private (list)

- ✓ No storage association with original object
- ✓ All references are to the local object
- ✓ Values are undefined on entry and exit

shared (list)

- ✓ Data is accessible by all threads in the team
- ✓ All threads access the same address space

V1
RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 29

About Storage Association

- Private variables are undefined on entry and exit of the parallel region
- A private variable within a parallel region has no storage association with the same variable outside of the region
- Use the **firstprivate** and **lastprivate** clauses to override this behavior
- We illustrate these concepts with an example

V1
RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 30

The Firstprivate And Lastprivate Clauses

firstprivate (list)

- ✓ All variables in the list are initialized with the value the original object had before entering the parallel construct

lastprivate (list)

- ✓ The thread that executes the sequentially last iteration or section updates the value of the objects in the list

V1
RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 31

Example Firstprivate

```
n = 2; indx = 4;
#pragma omp parallel default(none) private(i,TID)\
        firstprivate(indx) shared(n,a)
{ TID = omp_get_thread_num();
  indx = indx + n*TID;
  for(i=indx; i<indx+n; i++)
    a[i] = TID + 1;
} /*-- End of parallel region --*/
```

TID = 0 TID = 1 TID = 2

index	0	1	2	3	4	5	6	7	8	9
value					1	1	2	2	3	3

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 32

Example Lastprivate

```
#pragma omp parallel for lastprivate(a)
for (int i=0; i<n; i++)
{ .......
  a = i + 1;
  .......
} // End of parallel region

b = 2 * a; // value of b is 2*n
```

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 33

The Default Clause

default (none | shared) C/C++

default (none | shared | private | threadprivate) Fortran

none
- ✓ *No implicit defaults; have to scope all variables explicitly*

shared
- ✓ *All variables are shared*
- ✓ *The default in absence of an explicit "default" clause*

private
- ✓ *All variables are private to the thread*
- ✓ *Includes common block data, unless THREADPRIVATE*

firstprivate
- ✓ *All variables are private to the thread; pre-initialized*

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 34

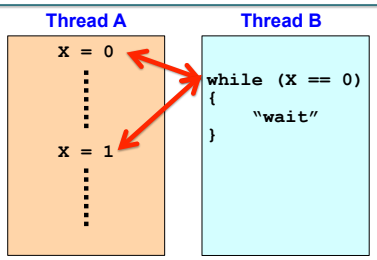
Data Scoping – Gotcha's

- *Need to get this right*
  - *Part of the learning curve*
- *Private data is undefined on entry and exit*
  - *Can use firstprivate and lastprivate to address this*
- *Each thread has its own temporary view on the data*
  - *Applicable to shared data only*
  - *Means different threads may temporarily not see the same value for the same variable ...*
  - *Let me explain*

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 35

The Flush Directive



If shared variable X is kept within a register, the modification may not be made visible to the other thread(s)

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 36

Example Of The Flush Directive

```
void wait_read(int i)
{
   #pragma omp flush
   while ( execution_state[i] != READ_FINISHED )
   {
      system("sleep 1");
      #pragma omp flush
   }
} /*-- End of wait_read --*/
```

This fragment is from the source code for the pipeline example from "Using OpenMP" (overlaps I/O with processing)

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 37

About The Flush Construct

- *Do not use the flush directive with a list*
  - *Could give very subtle interactions with compilers*
  - *If you insist on still doing so, be prepared to face the OpenMP language lawyers in court*
- *The flush is included in many constructs*
  - *A Good Thing - This is your safety net*
  - *Check the specifications for the specific constructs*

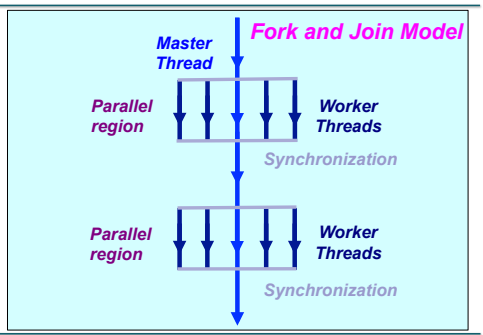
```
#pragma omp flush [(list)]
!$omp flush [(list)]
```

Don't use the list

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 38

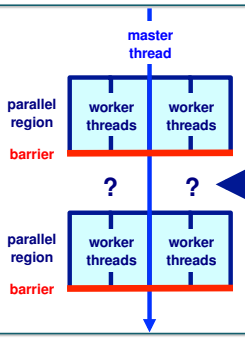
The OpenMP Execution Model



V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 39

Implementing The Fork-Join Model



Use the OMP_WAIT_POLICY environment variable to control the behaviour of idle threads

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 40

The Barrier/1

Suppose we run each of these two loops in parallel over i:

```
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 a wrong answer (one day)

Why ?

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 41

The Barrier/2

*We need to have updated all of a[ ] first, before using a[ ] **

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

wait !

barrier

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

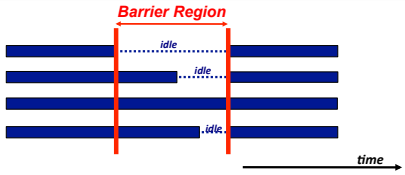
All threads wait at the barrier point and only continue when all threads have reached the barrier point

*) 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

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 42

The Barrier/3



Barrier syntax in OpenMP:

```
#pragma omp barrier
```

```
!$omp barrier
```

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 43

When To Use Barriers ?

- *In case data is updated asynchronously and data integrity is at risk*
- *Examples:*
  - *Between parts in the code that read and write the same section of memory*
  - *After one timestep/iteration in a solver*
- *Unfortunately, barriers tend to be expensive and also may not scale to a large number of processors*
- *Therefore, use them with care*

OpenMP Overview 44

The OpenMP Barrier

- *Several constructs have an implied barrier*
  - *This is another safety net (has implied flush by the way)*
- *In some cases, the implied barrier can be left out through the **nowait** clause*
- *This can help fine tuning the application*
  - *But you'd better know what you're doing*
- *The explicit barrier comes in quite handy then*

```
#pragma omp barrier
```

```
!$omp barrier
```


OpenMP Overview 45

The Nowait Clause

- *To minimize synchronization, some directives support the optional nowait clause*
  - *If present, threads do not synchronize/wait at the end of that particular construct*
- *In C, it is one of the clauses on the pragma*
- *In Fortran, it is appended at the closing part of the construct*

```
#pragma omp for nowait
{
        :
}
```

```
!$omp do
        :
        :
!$omp end do nowait
```


OpenMP Overview 46

OpenMP Directives (An Important Subset)

OpenMP Overview 47

The Worksharing Constructs

```
#pragma omp for
{
   ....
}

!$OMP DO
   ....
!$OMP END DO
```

```
#pragma omp sections
{
      ....
}

!$OMP SECTIONS
      ....
!$OMP END SECTIONS
```

```
#pragma omp single
{
      ....
}

!$OMP SINGLE
      ....
!$OMP END SINGLE
```

- ✓ *The work is distributed over the threads*
- ✓ *Must be enclosed in a parallel region*
- ✓ *Must be encountered by all threads in the team, or none at all*
- ✓ *No implied barrier on entry; implied barrier on exit (unless the nowait clause has been specified)*
- ✓ *A work-sharing construct does not launch any new threads*

OpenMP Overview 48

The Fortran Workshare Construct

Fortran has a fourth worksharing construct:

```
!$OMP WORKSHARE

    <array syntax>

!$OMP END WORKSHARE [NOWAIT]
```

Example:

```
!$OMP WORKSHARE
    A(1:M) = A(1:M) + B(1:M)
!$OMP END WORKSHARE NOWAIT
```

OpenMP Overview 49

The Omp-For/Do Directive

```
#pragma omp for [clauses]
  for (.....)
  {
     <code-block>
  }
```

```
!$omp do [clauses]
  do ...
     <code-block>
  end do
!$omp end do[nowait]
```

The iterations of the loop are distributed over the threads

OpenMP Overview 50

The Omp-For Directive – An Example

```
#pragma omp parallel default(none)\
        shared(n,a,b,c,d) private(i)
  {
    #pragma omp for nowait
      for (i=0; i<n-1; i++)
          b[i] = (a[i] + a[i+1])/2;
    #pragma omp for nowait
      for (i=0; i<n; i++)
          d[i] = 1.0/c[i];
  } /*-- End of parallel region --*/
```

(implied barrier)

OpenMP Overview 51

The Reduction Clause - Example

```
      sum = 0.0
!$omp parallel default(none) &
!$omp shared(n,x) private(i)
!$omp do reduction (+:sum)
      do i = 1, n
         sum = sum + x(i)
      end do
!$omp end do
!$omp end parallel
      print *,sum
```

Variable SUM is a shared variable

- ✓ *Care needs to be taken when updating shared variable SUM*
- ✓ *With the reduction clause, the OpenMP compiler generates code such that a race condition is avoided*

OpenMP Overview 52

The Reduction Clause

reduction (operator: list)	*C/C++*
reduction ([operator | intrinsic]) : list)	*Fortran*

- ✓ *Reduction variable(s) must be shared variables*
- ✓ *A reduction is defined as:*

Check the specs for details

Fortran	*C/C++*
`x = x operator expr`	`x = x operator expr`
`x = expr operator x`	`x = expr operator x`
`x = intrinsic (x, expr_list)`	`x++, ++x, x--, --x`
`x = intrinsic (expr_list, x)`	`x <binop> = expr`
	`"min" and "max" intrinsic`

- ✓ *Note that the value of a reduction variable is undefined from the moment the first thread reaches the clause till the operation has completed*
- ✓ *The reduction can be hidden in a function call*

OpenMP Overview 53

User Defined Reductions (UDR)

- *In addition to the pre-defined reduction operators supported, one can now write a user specific reduction operator*
  - *Combines ease of use of the reduction clause with application specific reductions*
- *Restrictions apply (please read the fine print on this)*

```
#pragma omp declare reduction ....
```

```
!$omp declare reduction ....
```


OpenMP Overview 54

Loop Collapse

- *Allows parallelization of perfectly nested loops without using nested parallelism*
- *The collapse clause on for/do loop indicates how many loops should be collapsed*
- *Compiler forms a single loop and then parallelizes it*

```
!$omp parallel do collapse(2) ...
  do i = il, iu, is
     do j = jl, ju, js
        do k = kl, ku, ks
           .....
        end do
     end do
  end do
!$omp end parallel do
```

OpenMP Overview 55

SIMD Support/1

- *SIMD = Single Instruction Multiple Data*
  - *Also called 'micro-vectorization'*
- *A single instruction performs the same operation on multiple data elements in parallel*

a[0]	a[1]	a[2]	a[3]
+			
b[0]	b[1]	b[2]	b[3]
=			
c[0]	c[1]	c[2]	c[3]

64-bit

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 56

SIMD Support/2

- *The **simd** construct can be applied to:*
  - *A serial loop (only SIMD instructions generated)*
  - *A parallel loop (two level parallelism)*

```
#pragma omp simd [clause(s)]
#pragma omp for simd [clause(s)]
```

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 57

Parallel Sections

```
#pragma omp sections [clauses]
{
  #pragma omp section
     {....}
  #pragma omp section
     {....}
  ....
}
```

Individual section blocks are executed in paralle

```
!$omp sections [clauses]
  !$ omp section
     {....}
  !$ omp section
     {....}
  ....
!$omp end sections [nowait]
```

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 58

The Sections Directive – An Example

```
#pragma omp parallel default(none)\
        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 --*/
```

RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 59

Overlap I/O And Processing/1

Input Thread	Processing Thread(s)	Output Thread
0		
1	0	
2	1	0
3	2	1
4	3	2
5	4	3
	5	4
		5

Time

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 60

Overlap I/O And Processing/2

```
#pragma omp parallel sections
{
  #pragma omp section
  {
    for (int i=0; i<N; i++) {
      (void) read_input(i);
      (void) signal_read(i);
    }
  }
  #pragma omp section
  {
    for (int i=0; i<N; i++) {
      (void) wait_read(i);
      (void) process_data(i);
      (void) signal_processed(i);
    }
  }
  #pragma omp section
  {
    for (int i=0; i<N; i++) {
      (void) wait_processed(i);
      (void) write_output(i);
    }
  }
} /*-- End of parallel sections --*/
```

Input Thread

Processing Thread(s)

Output Thread

OpenMP Overview 61

Single Processor Region/1

OpenMP

Original Code

```
.....
"read A[0..N-1]";
.....
```

Only one thread executes the single region

```
#pragma omp parallel \
        shared (A)
{
    .....
   #pragma omp single nowait
     {"read A[0..N-1]";}
    .....
   #pragma omp barrier
     "use A"
}
```

Parallel Version

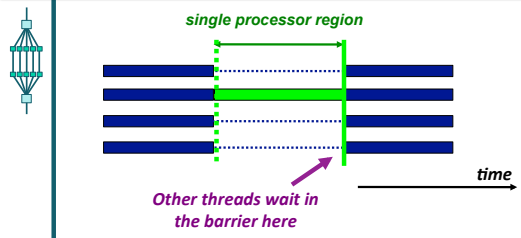
ideally suited for I/O or initializations

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 62

Single Processor Region/2

OpenMP



V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 63

The Single Directive

OpenMP

Only one thread in the team executes the code enclosed

```
#pragma omp single [private][firstprivate] \
                   [copyprivate][nowait]
{
    <code-block>
}
```

```
!$omp single [private][firstprivate]
    <code-block>
!$omp end single [copyprivate][nowait]
```

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 64

Worksharing Shortcuts

OpenMP

`#pragma omp parallel`
`#pragma omp for`
`  for (...)`
	`#pragma omp parallel for`
	`for (....)`

Single PARALLEL loop	

`!$omp parallel`
`!$omp do`
`  ...`
`!$omp end do`
`!$omp end parallel`
	`!$omp parallel do`
	`  ...`
	`!$omp end parallel do`

Single WORKSHARE loop	

`!$omp parallel`
`!$omp workshare`
`  ...`
`!$omp end workshare`
`!$omp end parallel`
	`!$omp parallel workshare`
	`  ...`
	`!$omp end parallel workshare`

`#pragma omp parallel`
`#pragma omp sections`
`{ ...}`
	`#pragma omp parallel`
	`sections`
	`{ ... }`

Single PARALLEL sections	

`!$omp parallel`
`!$omp sections`
`  ...`
`!$omp end sections`
`!$omp end parallel`
	`!$omp parallel sections`
	`  ...`
	`!$omp end parallel sections`

OpenMP Overview 65

The Critical Region/1

OpenMP

If sum is a shared variable, this loop can not be run in parallel by simply using a "#pragma omp for"

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

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

All threads execute the update, but now only one at a time will do so

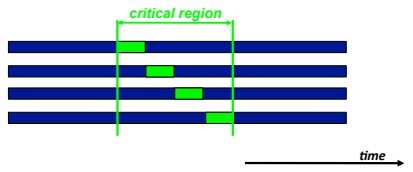
V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 66

The Critical Region/2

OpenMP

- *Very useful to avoid a race condition, or to perform I/O*
  - *But the order of execution is still undefined*
- *Be aware there is a performance cost associated with a critical region*



V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 67

The Critical Construct

```
#pragma omp critical [(name)]
{<code-block>}
```

```
!$omp critical [(name)]
    <code-block>
!$omp end critical [(name)]
```

Very useful to avoid data races

OpenMP Overview 68

The OpenMP Environment Variables

OpenMP Overview 69

Environment Variables/1

OpenMP Environment Variable	Category
OMP_DISPLAY_ENV	Diagnostics
OMP_NUM_THREADS	Thread Management
OMP_THREAD_LIMIT	Thread Management
OMP_DYNAMIC {true| false}	Thread Management
OMP_NESTED {true| false}	Parallelism
OMP_MAX_ACTIVE_LEVELS	Parallelism
OMP_STACKIZE "size [b|k|m|g]"	Operational

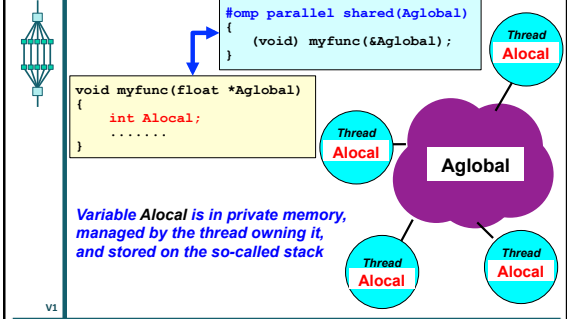
- ✓ *The names have to be in uppercase; the values are case insensitive*
- ✓ *Be careful when relying on defaults (they are compiler dependent)*

OpenMP Overview 70

About The Stack

```
#omp parallel shared(Aglobal)
{
   (void) myfunc(&Aglobal);
}
```

```
void myfunc(float *Aglobal)
{
   int Alocal;
   .......
}
```



Variable Alocal is in private memory, managed by the thread owning it, and stored on the so-called stack

OpenMP Overview 71

Environment Variables/2

OpenMP Environment Variable	Category
OMP_PLACES	Affinity
OMP_PROC_BIND	Affinity
OMP_DEFAULT_DEVICE	Accelerators
OMP_CANCELLATION	Thread Management
OMP_WAIT_POLICY [active | passive]	Thread Management
OMP_SCHEDULE "schedule, [chunk]"	Work Distribution

OpenMP Overview 72

The OpenMP Runtime Functions

OpenMP Overview 73

OpenMP Runtime Functions

- *OpenMP provides a set of runtime functions*
- *They all start with "omp_"*
- *These functions can be used to:*
  - *Query for a specific feature or setting*
    - *For example, the current iteration scheduling policy for loops*
  - *Change the setting*
    - *For example, to change the iteration scheduling policy for loops*
- *A special category consists of the locking functions*

C/C++ : Need to include file <omp.h>
Fortran : Add "use omp_lib" or include file "omp_lib.h"

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 74

Runtime Functions/1

Function Name	Category
omp_get_num_threads/omp_set_num_threads	Thread Management
omp_get_max_threads	Thread Management
omp_get_thread_num	Thread Management
omp_get_dynamic/omp_set_dynamic	Thread Management
omp_in_parallel	Parallelism
omp_get_nested/omp_set_nested	Parallelism

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 75

Runtime Functions/2

Function Name	Category
omp_get_num_procs	Operational
omp_get_wtime	Operational
omp_get_wtick	Operational
omp_init_lock/omp_init_nest_lock	Locking
omp_set_lock/omp_set_nest_lock	Locking
omp_unset_lock/omp_unset_nest_lock	Locking
omp_test_lock/omp_test_nest_lock	Locking
omp_destroy_lock/omp_destroy_nest_lock	Locking

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 76

Runtime Functions/3

Function Name	Category
omp_get_thread_limit	Thread Management
omp_get_schedule/omp_set_schedule	Work Distribution
omp_get_max_active_levels/ omp_set_max_active_levels	Parallelism
omp_get_level	Parallelism
omp_get_active_level	Parallelism
omp_get_ancestor_thread_num	Parallelism
omp_get_team_size	Parallelism

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 77

Runtime Functions/4

Function Name	Category
omp_in_final	Tasking
omp_get_proc_bind	Thread Affinity
omp_get_cancellation	Cancellation
omp_get_num_teams	Accelerators
omp_get_team_num	Accelerators
omp_get_default_device	Accelerators
omp_set_default_device	Accelerators
omp_get_num_devices	Accelerators
omp_is_initial_device	Accelerators

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 78

Using OpenMP

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 79

Using OpenMP

- *The overview given so far can already be used to parallelize applications*
- *It is now time to look at some features in more detail*
- *And introduce some additional features*

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 80

Load Balancing

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 81

The Schedule Clause

schedule (static | dynamic | guided | auto [, chunk])
schedule (runtime)

static [, chunk]

- ✓ *Distribute iterations in blocks of size "chunk" over the threads in a round-robin fashion*
- ✓ *In absence of "chunk", each thread executes approx. N/P chunks for a loop of length N and P threads*
  - *Details are implementation defined*
- ✓ *Under certain conditions, the assignment of iterations to threads is the same across multiple loops in the same parallel region*

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 82

Example Of A Static Schedule

A loop of length 16 using 4 threads

Thread	0	1	2	3
no chunk *	1-4	5-8	9-12	13-16
chunk = 2	1-2	3-4	5-6	7-8
	9-10	11-12	13-14	15-16

*) The precise distribution is implementation defined

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 83

Loop Workload Scheduling Choices

dynamic [, chunk]

- ✓ *Fixed portions of work; size is controlled by the value of chunk*
- ✓ *When a thread finishes, it starts on the next portion of work*

guided [, chunk]

- ✓ *Same dynamic behavior as "dynamic", but size of the portion of work decreases exponentially*

auto

- ✓ *The compiler (or runtime system) decides what is best to use; choice could be implementation dependent*

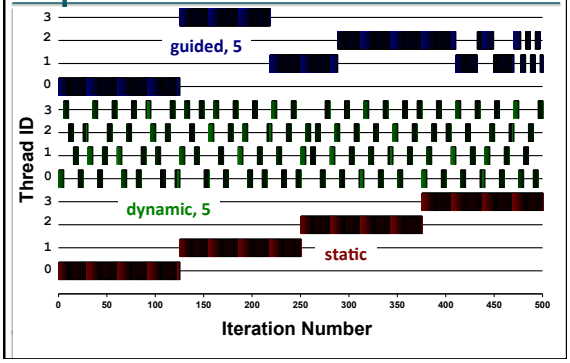
runtime

- ✓ *Iteration scheduling scheme is set at runtime through environment variable OMP_SCHEDULE*

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 84

Experiment – 500 Iterations, 4 Threads



OpenMP Overview 85

The Master Directive

Only the master thread executes this code block

```
#pragma omp master
{<code-block>}
```

```
!$omp master
    <code-block>
!$omp end master
```

There is no implied barrier on entry or exit !

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 86

The Ordered Directive

The enclosed block of code within a parallel loop is executed in the order in which iterations would be executed sequentially:

```
#pragma omp ordered
{<code-block>}
```

```
!$omp ordered
    <code-block>
!$omp end ordered
```

May introduce serialization (could be expensive)

Note: Need to use the ordered clause on the parallel for/do loop

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 87

Atomic Operations

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 88

The Atomic Construct/1

"An atomic operation can be executed without any other thread being able to read or change state that is read or changed during the operation"

```
#pragma omp atomic update
{
      x++;
}
```

Source: http://wiki.osdev.org/Atomic_operation

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 89

The Atomic Construct/2

```
#pragma omp atomic [clause]
```

```
!$omp atomic [clause]
```

Lightweight

Clauses on the atomic:
read, write, update, capture, seq_cst

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 90

Example – The Capture Clause

This captures the original (or final) value, stored in a designated variable

```
#pragma omp atomic capture
{
      capture-statement
      write-statement
}
```

```
#pragma omp atomic capture
{
  old_value = *p;
  (*p)++;
}
// Can use old_value here
```

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 91

Sequentially Consistent Atomics

"...the results of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program." – Leslie Lamport

*Specify through the **seq_cst** clause on the atomic construct*

This implies a flush without a list

OpenMP Overview 92

Orphaning

OpenMP Overview 93

Orphaning

```
        :
#pragma omp parallel
{
        :
      (void) dowork()
        :
} // End of parallel
        :
```

```
void dowork()
{
        :
  #pragma omp for
   for (int i=0;i<n;i++)
   {
        :
   }
        :
}
```

orphaned work-sharing directive

- *The OpenMP specification does not restrict worksharing and synchronization directives (omp for, omp single, critical, barrier, etc.) to be within the lexical extent of a parallel region. These directives can be orphaned*
- *That is, they can appear outside the lexical extent of a parallel region*

OpenMP Overview 94

More On Orphaning

```
(void) dowork(); !- Sequential FOR
#pragma omp parallel
{
  (void) dowork(); !- Parallel FOR
}
```

```
void dowork()
{
#pragma omp for
   for (i=0;....)
   {
      :
   }
}
```

When an orphaned worksharing or synchronization directive is encountered in the sequential part of the program (outside the dynamic extent of any parallel region), it is executed by the master thread only. In effect, the directive will be ignored

OpenMP Overview 95

Example - Parallelizing Bulky Loops

```
for (i=0; i<n; i++) /* Parallel loop */
{
    a = ...
    b = ... a ..
    c[i] = ....
         ......
    for (j=0; j<m; j++)
    {
      <a lot more work in this loop>
    }
         ......
}
```

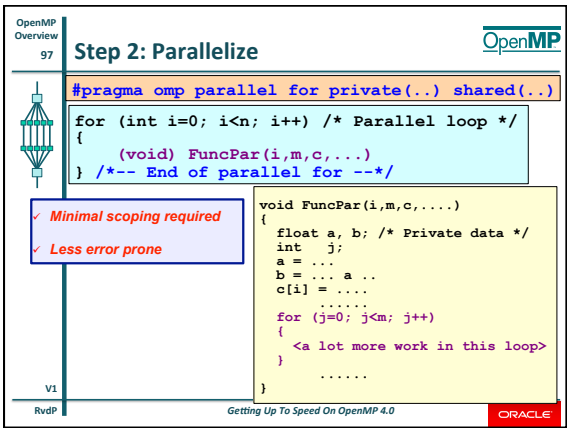

OpenMP Overview 96

Step 1: "Outlining"

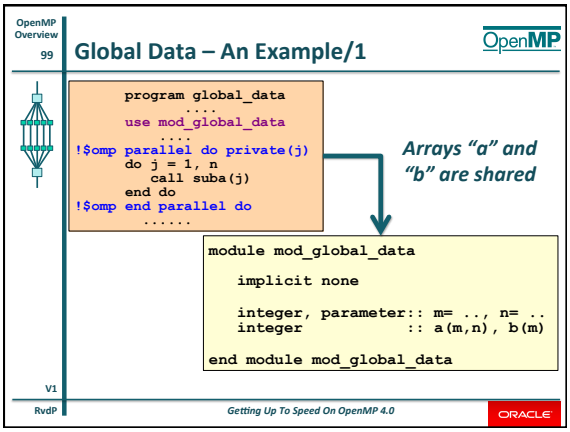
```
for (int i=0; i<n; i++) /* Parallel loop */
{
    (void) FuncPar(i,m,c,...)
}
```

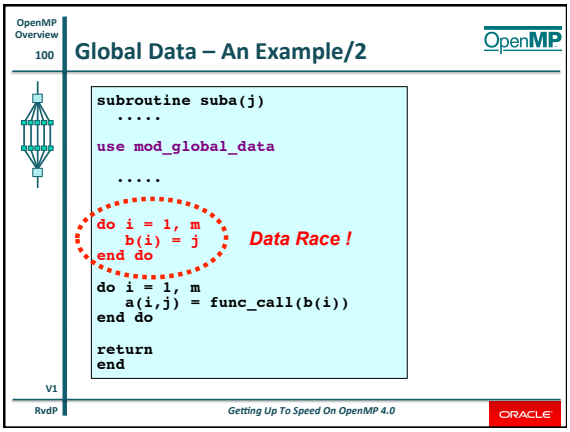
- ✓ *Still a sequential program*
- ✓ *Should behave identically*
- ✓ *Easy to test for correctness*
- ✓ *But, parallel by design*

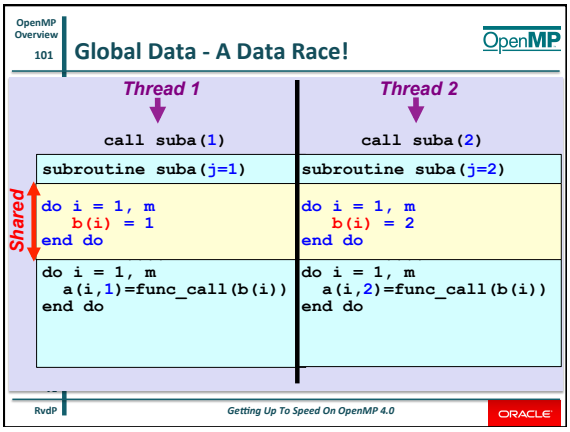
```
void FuncPar(i,m,c,....)
{
  float a, b; /* Private data */
  int   j;
  a = ...
  b = ... a ..
  c[i] = ....
       ......
  for (j=0; j<m; j++)
  {
    <a lot more work in this loop>
  }
       ......
}
```

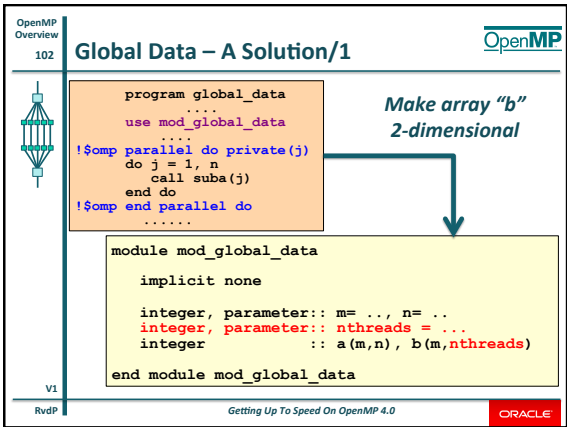

OpenMP Overview
97
Step 2: Parallelize
OpenMP
#pragma omp parallel for private(..) shared(..)
for (int i=0; i<n; i++) /* Parallel loop */
{
(void) FuncPar(i,m,c,...)
} /*-- End of parallel for --*/
Minimal scoping required
Less error prone
void FuncPar(i,m,c,....)
{
float a, b; /* Private data */
int j;
a = ...
b = ... a ..
c[i] = ...
.....
for (j=0; j<m; j++)
{
<a lot more work in this loop>
}
.....
}
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
98
OpenMP
Global Data
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
99
Global Data – An Example/1
OpenMP
program global_data
....
use mod_global_data
....
!$omp parallel do private(j)
do j = 1, n
call suba(j)
end do
!$omp end parallel do
.....
Arrays "a" and "b" are shared
module mod_global_data
implicit none
integer, parameter:: m= .., n= ..
integer :: a(m,n), b(m)
end module mod_global_data
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
100
Global Data – An Example/2
OpenMP
subroutine suba(j)
.....
use mod_global_data
.....
do i = 1, m
b(i) = j
end do
Data Race !
do i = 1, m
a(i,j) = func_call(b(i))
end do
return
end
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
101
Global Data - A Data Race!
OpenMP
Thread 1
Thread 2
call suba(1)
call suba(2)
subroutine suba(j=1)
subroutine suba(j=2)
Shared
do i = 1, m
b(i) = 1
end do
do i = 1, m
b(i) = 2
end do
do i = 1, m
a(i,1)=func_call(b(i))
end do
do i = 1, m
a(i,2)=func_call(b(i))
end do
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
102
Global Data – A Solution/1
OpenMP
program global_data
....
use mod_global_data
....
!$omp parallel do private(j)
do j = 1, n
call suba(j)
end do
!$omp end parallel do
.....
Make array "b" 2-dimensional
module mod_global_data
implicit none
integer, parameter:: m= .., n= ..
integer, parameter:: nthreads = ...
integer :: a(m,n), b(m,nthreads)
end module mod_global_data
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE

OpenMP Overview 103

Global Data – A Solution/2

```
subroutine suba(j)
  .....
use omp_lib
use mod_global_data
  .....
TID = omp_get_thread_num()+1
do i = 1, m
   b(i,TID) = j
end do
do i = 1, m
   a(i,j) = func_call(b(i),TID)
end do
return
end
```

A lot of work and not very portable

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 104

About Global Data

- *Global data is shared and requires special care*
- *A problem may arise in case multiple threads access the same memory section simultaneously:*
  - *Read-only data is no problem*
  - *Updates have to be checked for race conditions*
- *It is your responsibility to deal with this situation*
- *In general one can do the following:*
  - *Split the global data into a part that is accessed in serial parts only and a part that is accessed in parallel*
  - *Manually create thread private copies of the latter*
  - *Use the thread ID to access these private copies*
- *Alternative: Use OpenMP's threadprivate directive !*

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 105

The Threadprivate Directive

```
#pragma omp threadprivate (list)
!$omp threadprivate (/cb/ [,/cb/] ...)
```

- *Thread private copies of the designated global variables and common blocks are created*
- *Several restrictions and rules apply when doing this:*
  - *The number of threads has to remain the same for all the parallel regions (i.e. no dynamic threads)*
    - *Some implementations support changing the number of threads*
  - *Initial data is undefined, unless copyin is used*
  - *.....*
- *Check the specifications when using threadprivate !*

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 106

Global Data – The Preferred Solution

```
      program global_data
          ...
      use mod_global_data
          ...
!$omp parallel do private(j)
      do j = 1, n
         call suba(j)
      end do
!$omp end parallel do
          ......
```

Only add the "threadprivate" directive to the module file; no other changes needed !

This solution also automatically adapts to the number of threads used

```
module mod_global_data

   implicit none

   integer, parameter:: m= .., n= ..
   integer            :: a(m,n), b(m)

   !$omp threadprivate(b)

end module mod_global_data
```

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 107

The Copyin Clause

copyin (list)

- ✓ *Applies to THREADPRIVATE common blocks only*
- ✓ *At the start of the parallel region, data of the master thread is copied to the thread private copies*

Example:

```
      common /cblock/velocity
      common /fields/xfield, yfield, zfield

! create thread private common blocks

!$omp threadprivate (/cblock/, /fields/)

!$omp parallel          &
!$omp default (private) &
!$omp copyin ( /cblock/, zfield )
```

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 108

C++ And Threadprivate

- *As of OpenMP 3.0, it has been clarified where/how threadprivate objects are constructed and destructed*
- *Allow C++ static class members to be threadprivate*

```
class T {
  public:
  static int i;
  #pragma omp threadprivate(i)
  ...
};
```

V1 RvdP Getting Up To Speed On OpenMP 4.0

OpenMP Overview 109

A Locking Example

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 110

OpenMP Locking Routines

- *Locks provide greater flexibility over critical sections and atomic updates:*
  - *Possible to implement asynchronous behavior*
  - *Not block structured*
- *The so-called lock variable, is a special variable:*
  - *C/C++: type omp_lock_t and omp_nest_lock_t for nested locks*
  - *Fortran: type INTEGER and of a KIND large enough to hold an address*
- *Lock variables should be manipulated through the API only*
- *It is illegal, and behavior is undefined, in case a lock variable is used without the appropriate initialization*

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 111

Nested Locking

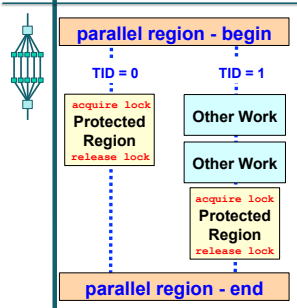
- *Simple locks: may not be locked if already in a locked state*
- *Nestable locks: may be locked multiple times by the same thread before being unlocked*
- *In the remainder, we discuss simple locks only*
- *The interface for functions dealing with nested locks is*

Simple locks	Nestable locks
omp_init_lock	omp_init_nest_lock
omp_destroy_lock	omp_destroy_nest_lock
omp_set_lock	omp_set_nest_lock
omp_unset_lock	omp_unset_nest_lock
omp_test_lock	omp_test_nest_lock

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 112

OpenMP Locking Example



- *The protected region contains the update of a shared variable*
- *One thread acquires the lock and performs the update*
- *Meanwhile, the other thread performs some other work*
- *When the lock is released again, the other thread performs the update*

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 113

Locking Example - The Code

```
      Program Locks
            ....
      Call omp_init_lock (LCK)
!$omp parallel shared(LCK)
       Do While ( omp_test_lock (LCK) .EQV. .FALSE. )
          Call Do_Something_Else()
       End Do
       Call Do_Work()
       Call omp_unset_lock (LCK)
!$omp end parallel
      Call omp_destroy_lock (LCK)
      Stop
      End
```

Initialize lock variable

Check availability of lock (also sets the lock)

Release lock again

Remove lock association

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 114

Example Output Using 2 Threads

```
TID:  1 at 09:07:27 => entered parallel region
TID:  1 at 09:07:27 => done with WAIT loop and has the lock
TID:  1 at 09:07:27 => ready to do the parallel work
TID:  1 at 09:07:27 => this will take about 18 seconds
TID:  0 at 09:07:27 => entered parallel region
TID:  0 at 09:07:27 =>   WAIT for lock - will do something else for  5 seconds
TID:  0 at 09:07:32 =>   WAIT for lock - will do something else for  5 seconds
TID:  0 at 09:07:37 =>   WAIT for lock - will do something else for  5 seconds
TID:  0 at 09:07:42 =>   WAIT for lock - will do something else for  5 seconds
TID:  1 at 09:07:45 => done with my work
TID:  1 at 09:07:45 => done with work loop - released the lock
TID:  1 at 09:07:45 => ready to leave the parallel region
TID:  0 at 09:07:47 => done with WAIT loop and has the lock
TID:  0 at 09:07:47 => ready to do the parallel work
TID:  0 at 09:07:47 => this will take about 18 seconds
TID:  0 at 09:08:05 => done with my work
TID:  0 at 09:08:05 => done with work loop - released the lock
TID:  0 at 09:08:05 => ready to leave the parallel region
Done at 09:08:05 - value of SUM is 1100
```

Used to check the answer

Note: program has been instrumented to get this information

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 115

OpenMP

Additional Functionality

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 117

Additional Functionality

OpenMP

- OpenMP continues to evolve
  - Increasingly flexible to support more algorithms
  - Support for modern parallel architectures
    - cc-NUMA and Accelerators
- In recent years, several key features have been added to significantly enhance the functionality
- The following features are discussed next:
  - Thread Cancellation
  - Accelerator Support
  - Tasking
  - Thread Affinity

Focus for today

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 118

OpenMP

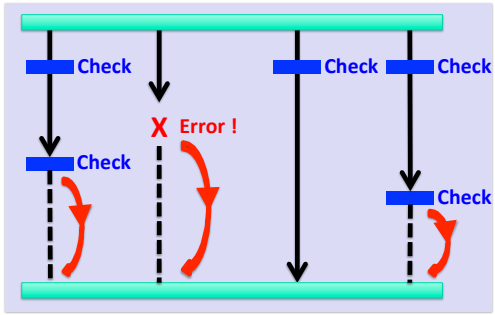
Thread Cancellation

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 119

Thread Cancellation – Basic Idea

OpenMP



V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 120

About Thread Cancellation

OpenMP

- Cancellation – Abort an OpenMP region
  - Implicit or explicit tasks proceed to the end of the canceled region
- Cancellation point – Point where tasks check if cancellation has been requested
  - Take action accordingly (abort or continue)
- Cancellation points are:
  - Implicit barriers, barrier regions, cancel regions, cancellation point regions

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 121

Cancellation Directives

Activate cancellation of the innermost enclosing region (if cancellation has been enabled)

```
#pragma omp cancel <construct> [if-clause]
```

Define an explicit cancellation point

```
#pragma omp cancellation point <construct>
```

Supported constructs:

```
parallel
sections
for
taskgroup
```

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

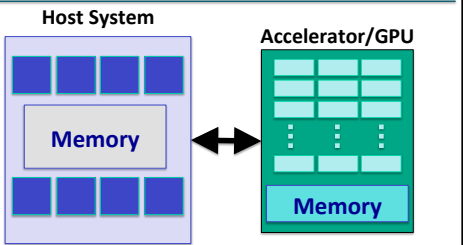
OpenMP Overview 122

Accelerator Support

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 123

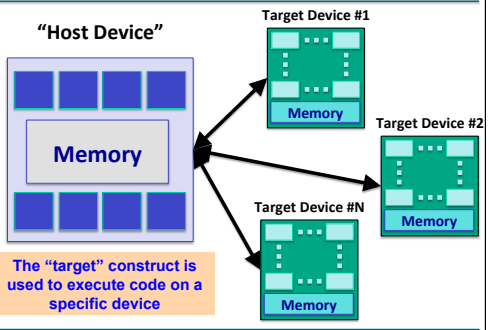
Accelerator Support/1



V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 124

Accelerator Support/2



The "target" construct is used to execute code on a specific device

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 125

Accelerator Support/3

```
#pragma omp target device(2)
{
    "execute this code on target device #2"
}
```

- *User needs to explicitly manage input and output data using the* **map** *clause*
- *This includes "to", "from" and "tofrom" syntax*

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

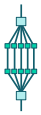
OpenMP Overview 126

Tasking

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 127

Tasking In OpenMP

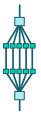


- *Tasking was introduced in OpenMP 3.0*
- *Until then, it required a heroic effort to efficiently implement certain types of parallelism*
  - *Divide and Conquer algorithms*
  - *Recursive algorithms*
  - *Linked lists*
  - *...*
- *The initial tasking functionality in 3.0 was very simple*
  - *The idea was to augment tasking as we collectively gain more insight and experience*
  - *Successive specifications have added functionality*

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 128

What Is An OpenMP Task ?



- *Loosely said, a task is a block of code that can be executed simultaneously with other tasks*
- *It is upon the programmer to identify tasks in the code*
- *From there on, the compiler and run time system handle the creation and execution of the tasks*

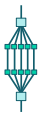
```
#pragma omp task [clauses]
!$omp task [clauses]
```

Defines a task

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 129

Tasking Example



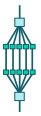
```
#pragma omp parallel
{
    #pragma omp task
      {printf("I am a task\n");}
} // End of parallel region
```

All threads execute this task

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 130

Tasking - Terminology



*A **task** is a specific instance of executable code and its data environment, generated when a thread encounters a task construct or a parallel construct.*

*A **task region** is a region consisting of all code encountered during the execution of a task.*

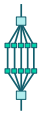
*The **data environment** consists of all the variables associated with the execution of a given task. The data environment for a given task is constructed from the data environment of the generating task at the time the task is generated.*

Note: The data environment is not covered here. Come to our tutorial at SC'15 to learn more about this!

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 131

Tasking - Who Does What ?



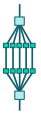
- *When a thread encounters a task construct, a new explicit task is generated*
  - *But not necessarily executed yet*
- *Execution of tasks is handled by the run time system*
  - *Tasks are assigned to the threads in the current team*
  - *This is subject to the thread's availability and thus could be immediate, or deferred until later*
- *Threads are allowed to suspend the current task region at a **task scheduling point** in order to execute a different task*

Note: Tasks can be nested (not for the faint of heart)

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 132

Tasking – And When ?



- *Task completion occurs when the end of the structured block associated with the construct that generated the task is reached*
- *Completion of a subset of all explicit tasks bound to a given parallel region may be specified through the use of **task synchronization constructs***
  - *Completion of all explicit tasks bound to the implicit parallel region is guaranteed by the time the program exits*
- *A task synchronization construct is a **taskwait**, **taskgroup** or a **barrier** construct*

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 133

How To Enforce Task Completion ?

```
#pragma omp taskwait
!$omp flush taskwait
```

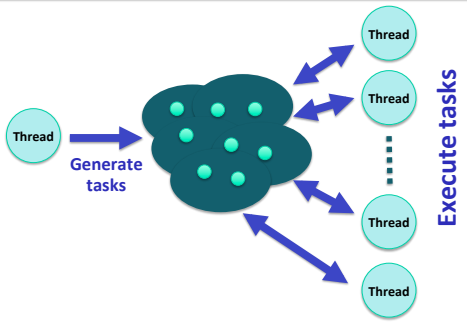
*Wait on completion of all child tasks **

***) This includes direct children only, not descendant tasks. Use the taskgroup construct for the latter**

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 134

A Common Tasking Structure



V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 135

A Simple Plan

Your Task for Today:

*Write a program that prints either "**A race car**" or "**A car race**" and maximize the parallelism*

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 136

Tasking Example/1

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {

          printf("A ");
          printf("race ");
          printf("car ");

   printf("\n");
   return(0);
}
```

```
$ cc -fast hello.c
$ ./a.out
A race car
$
```

What will this program print ?

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 137

Tasking Example/2

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {

   #pragma omp parallel
   {
          printf("A ");
          printf("race ");
          printf("car ");

   } // End of parallel region

   printf("\n");
   return(0);
}
```

What will this program print using 2 threads ?

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 138

Tasking Example/3

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A race car A race car
```

Note that this program could (for example) also print

"A A race race car car" or

"A race A car race car", or

"A race A race car car", or

.....

But I have not observed this (yet)

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 139

Tasking Example/4

```
#include <stdlib.h
#include <stdio.h

int main(int argc

   #pragma omp parallel
   {
     #pragma omp single
      {
          printf("A ");
          printf("race ");
          printf("car ");
      }
   } // End of parallel region

   printf("\n");
   return(0);
}
```

What will this program print using 2 threads ?

OpenMP Overview 140

Tasking Example/5

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A race car
```

But of course now only 1 thread executes .....

OpenMP Overview 141

Tasking Example/6

```
int main(int argc, char *argv[]) {

   #pragma omp parallel
   {
     #pragma omp single
      {
         printf("A ");
         #pragma omp task
          {printf("race ");}
         #pragma omp task
          {printf("car ");}
      }
   } // End of parallel region

   printf("\n");
   return(0);
}
```

What will this program print using 2 threads ?

OpenMP Overview 142

Tasking Example/7

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
A race car
$ ./a.out
A race car
$ ./a.out
A car race
$
```

Tasks can be executed in arbitrary order

OpenMP Overview 143

Another Simple Plan

You did well and quickly, so here is a final task to do

Have the sentence end with "is fun to watch" (hint: use a print statement)

OpenMP Overview 144

Tasking Example/8

```
int main(int argc, char *argv[]) {

   #pragma omp parallel
   {
     #pragma omp single
      {
         printf("A ");
         #pragma omp task
          {printf("race ");}
         #pragma omp task
          {printf("car ");}
        printf("is fun to watch ");
      }
   } // End of parallel region

   printf("\n");
   return(0);
}
```

What will this program print using 2 threads ?

OpenMP Overview 145

Tasking Example/9

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out

A is fun to watch race car
$ ./a.out

A is fun to watch race car
$ ./a.out

A is fun to watch car race
$
```

Tasks are executed at a task execution point

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 146

Tasking Example/10

```
int main(int argc, cha
  #pragma omp parallel
  {
    #pragma omp single
     {
        printf("A ");
        #pragma omp task
          {printf("car ");}
        #pragma omp task
          {printf("race ");}
        #pragma omp taskwait
        printf("is fun to watch ");
     }
   } // End of parallel region

   printf("\n");return(0);
}
```

What will this program print using 2 threads ?

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 147

Tasking Example/11

```
$ cc -xopenmp -fast hello.c
$ export OMP_NUM_THREADS=2
$ ./a.out
$
A car race is fun to watch
$ ./a.out
A car race is fun to watch
$ ./a.out
A race car is fun to watch
$
```

Tasks are executed first now

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 148

More About Tasking

- *Several features have not been covered yet*
- *These are beyond the scope of this tutorial*
  - *but it doesn't mean they're not useful*
- *Please check the specifications for much more information on tasking*
  - *Or attend our tutorial at SC'15!*

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 149

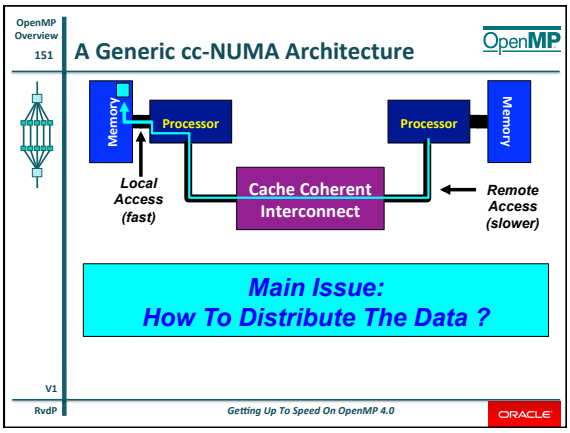
Thread Affinity

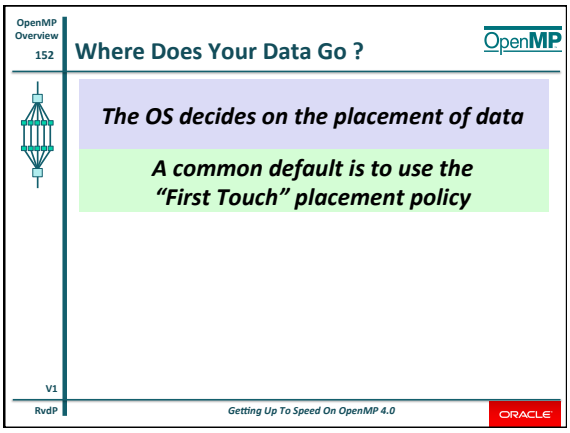
V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

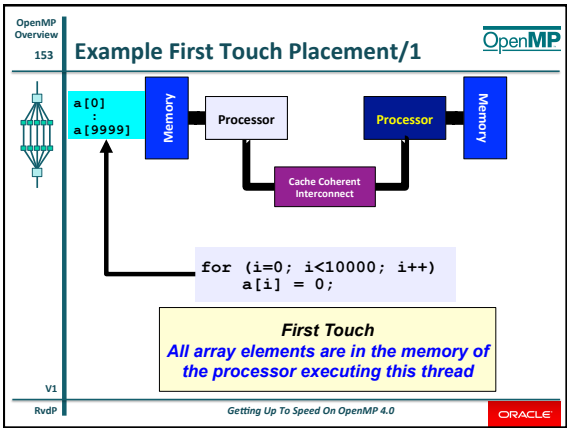
OpenMP Overview 150

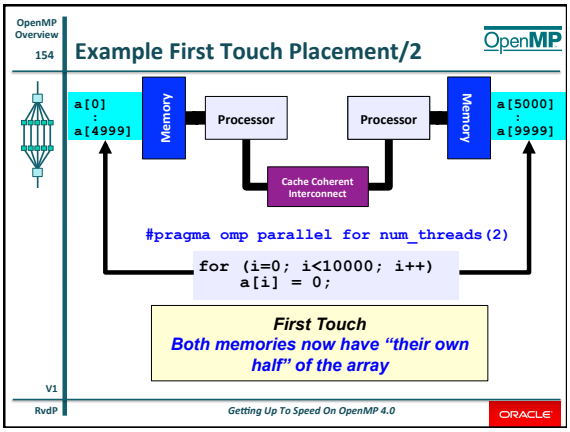
Why Worry ?

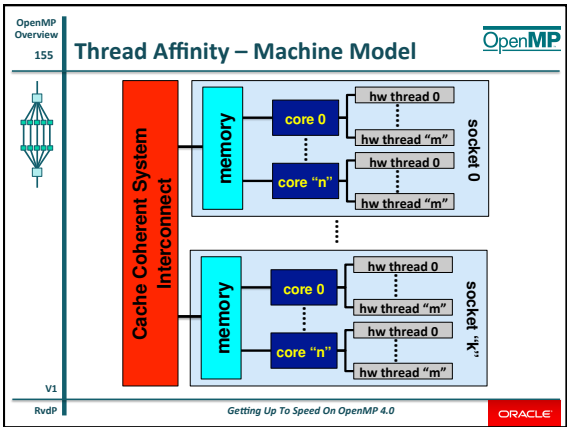
V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

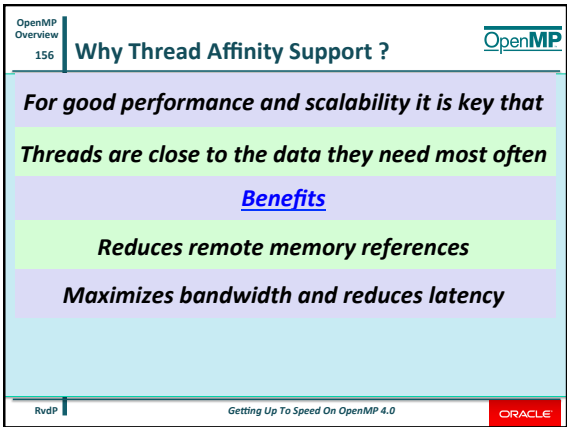

OpenMP Overview
151
A Generic cc-NUMA Architecture
OpenMP
Memory
Processor
Processor
Memory
Local Access (fast)
Cache Coherent Interconnect
Remote Access (slower)
Main Issue:
How To Distribute The Data ?
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
152
Where Does Your Data Go ?
OpenMP
The OS decides on the placement of data
A common default is to use the "First Touch" placement policy
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
153
Example First Touch Placement/1
OpenMP
a[0] : a[9999]
Memory
Processor
Processor
Memory
Cache Coherent Interconnect
for (i=0; i<10000; i++)
a[i] = 0;
First Touch
All array elements are in the memory of the processor executing this thread
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
154
Example First Touch Placement/2
OpenMP
a[0] : a[4999]
Memory
Processor
Processor
Memory
a[5000] : a[9999]
Cache Coherent Interconnect
#pragma omp parallel for num_threads(2)
for (i=0; i<10000; i++)
a[i] = 0;
First Touch
Both memories now have "their own half" of the array
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
155
Thread Affinity – Machine Model
OpenMP
Cache Coherent System Interconnect
memory
core 0
core "n"
hw thread 0
hw thread "m"
hw thread 0
hw thread "m"
socket 0
memory
core 0
core "n"
hw thread 0
hw thread "m"
hw thread 0
hw thread "m"
socket "k"
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
156
Why Thread Affinity Support ?
OpenMP
For good performance and scalability it is key that
Threads are close to the data they need most often
Benefits
Reduces remote memory references
Maximizes bandwidth and reduces latency
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE

OpenMP Overview 157

Basic Affinity Philosophy

OpenMP

Data is wherever it may be

Threads are moved to the data they need most

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 158

Two Key Concepts

OpenMP

The Place List

The Thread Affinity Policy

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 159

The Place List

OpenMP

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 160

Places – Definition And Notation

OpenMP

A place consists of a (set of) numbers

Each number represents a scheduling unit

That is, something a thread can run on

For example, a hardware thread

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 161

Example Multicore System

OpenMP

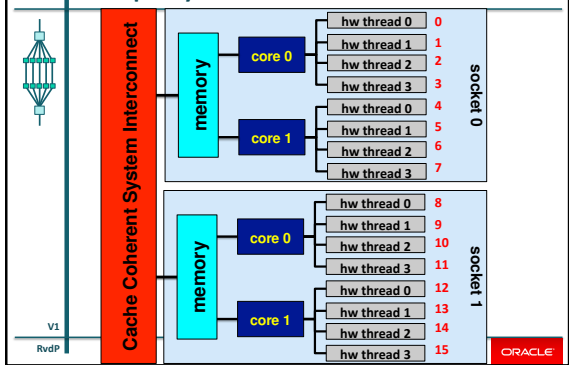
Component	Count	Total
Sockets	2	2
Cores/socket	2	4
Threads/core	4	16

V1 RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 162

Example System Architecture

OpenMP

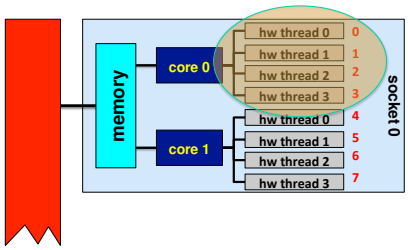


V1 RvdP ORACLE

OpenMP Overview 163

Places - Example

{0,1,2,3} identifies the threads in socket 0, core 0



OpenMP Overview 164

Places - Example

{0,1,2,3} identifies threads in socket 0, core 0

Easy interval notation: {0 : 4 : 1}

start : count: stride

OpenMP Overview 165

Places – The Order Does Not Matter

The order of the numbers within a place does not matter

{0,1,2,3} is the same as {3,2,1,0}

Assumption
No preference regarding memory access time

OpenMP Overview 166

The Place List - Definition

The Place List consists of a comma separated list of places

For example: {0,1,2,3} , {8,9,10,11}

The Order Of The Places In The List DOES Matter

{0,1,2,3} , {8,9,10,11} =/= {8,9,10,11} , {0,1,2,3}

OpenMP Overview 167

The Place List – How To Set It

Environment variable OMP_PLACES is used to define the place list

Example: OMP_PLACES= "{0,1,2,3} , {8,9,10,11}"

The interval notation is very convenient

Example: OMP_PLACES="{0:4:1} , {8:4:1}"

OpenMP Overview 168

The Place List – Abstract Names

Three abstract names are always available:

sockets , cores , threads

Example: OMP_PLACES=cores

Example: OMP_PLACES="cores(4)"

Note: Implementation can add names

OpenMP Overview 169

Example – OMP_PLACES

OpenMP

OMP_PLACES=cores

Equivalent To

OMP_PLACES="{0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15}"

Equivalent To

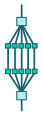
OMP_PLACES="{0:4:1},{4:4:1},{8:4:1},{12:4:1}"

Equivalent To

OMP_PLACES = "{0:4:1}:4:4"

RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 170



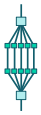
OpenMP

The Affinity Policy

V1

RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 171



Thread Affinity Policies

OpenMP

The Affinity Policy defines which places to use

This is defined in a symbolic way:

master, close, or spread

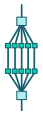
Each parallel region has such an affinity policy

Thread binding to a place is implied

V1

RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 172



Setting The Thread Affinity Policy

OpenMP

Defined through OMP_PROC_BIND

Example: OMP_PROC_BIND="spread,close"

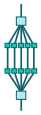
Can also use the "proc_bind" clause

Applies to the current parallel region only

V1

RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 173



Recap Places And Affinity

OpenMP

The Place List defines what is available

(fixed for the duration of the program)

The Affinity Policy defines thread placement

(can be set for each parallel region)

V1

RvdP Getting Up To Speed On OpenMP 4.0 ORACLE

OpenMP Overview 174

Example – The Spread Policy

OpenMP

OMP_PLACES=cores
(this means 4 places in the place list)

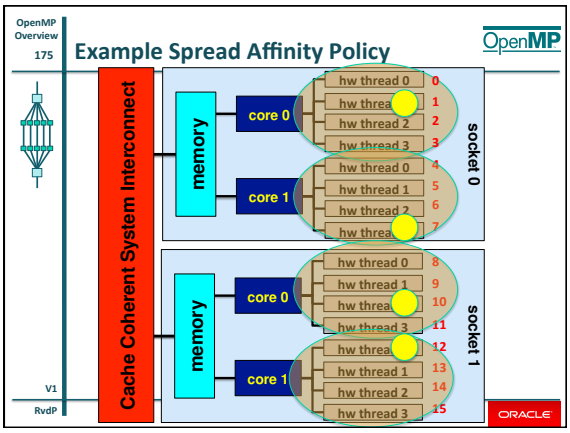
OMP_PROC_BIND=spread

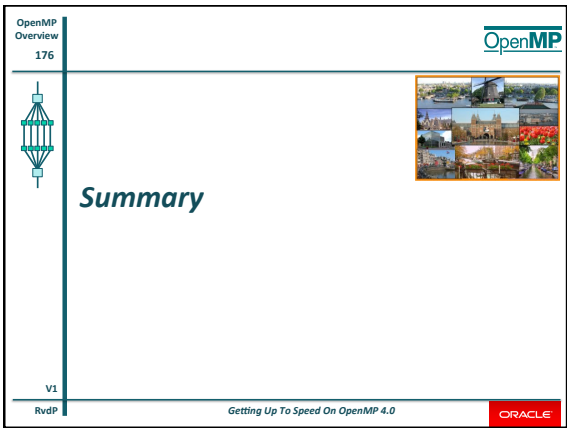
OMP_NUM_THREADS=4

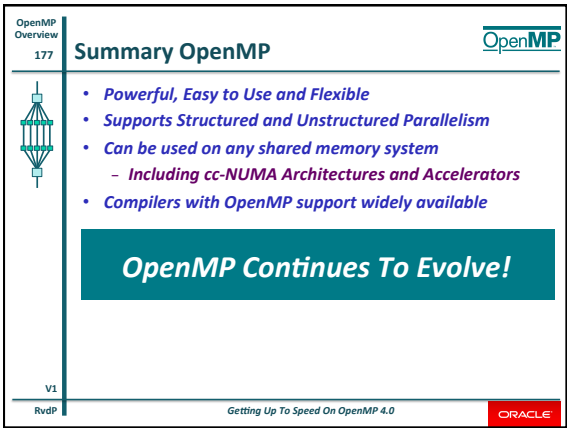
Result: One OpenMP thread per place

V1

RvdP Getting Up To Speed On OpenMP 4.0 ORACLE


OpenMP Overview
175
Example Spread Affinity Policy
OpenMP
Cache Coherent System Interconnect
memory
core 0
core 1
hw thread 0
hw thread 1
hw thread 2
hw thread 3
0
1
2
3
4
5
6
7
socket 0
memory
core 0
core 1
hw thread 0
hw thread 1
hw thread 2
hw thread 3
8
9
10
11
12
13
14
15
socket 1
V1
RvdP
ORACLE


OpenMP Overview
176
OpenMP
Summary
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
177
Summary OpenMP
OpenMP
Powerful, Easy to Use and Flexible
Supports Structured and Unstructured Parallelism
Can be used on any shared memory system
Including cc-NUMA Architectures and Accelerators
Compilers with OpenMP support widely available
OpenMP Continues To Evolve!
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE


OpenMP Overview
178
OpenMP
Thank You And ..... Stay Tuned !
ruud.vanderpas@oracle.com
ORACLE
Bad OpenMP
Does Not Scale
V1
RvdP
Getting Up To Speed On OpenMP 4.0
ORACLE