Programming with OpenMP*

Objectives

- Upon completion of this module you will be able to use OpenMP to:
  - implement data parallelism
  - implement task parallelism

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Agenda

- What is OpenMP?
  - Parallel regions
  - Worksharing
  - Data environment
  - Synchronization
- Optional Advanced topics

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

What Is OpenMP?

- Portable, shared-memory threading API
  - Fortran, C, and C++
  - Multi-vendor support for both Linux and Windows
- Standardizes task-level parallelism
- Supports coarse-grained parallelism
- Combines serial and parallel code in single source
- Standardizes ~ 20 years of compiler-directed threading experience

http://www.openmp.org

Current spec is OpenMP 3.0

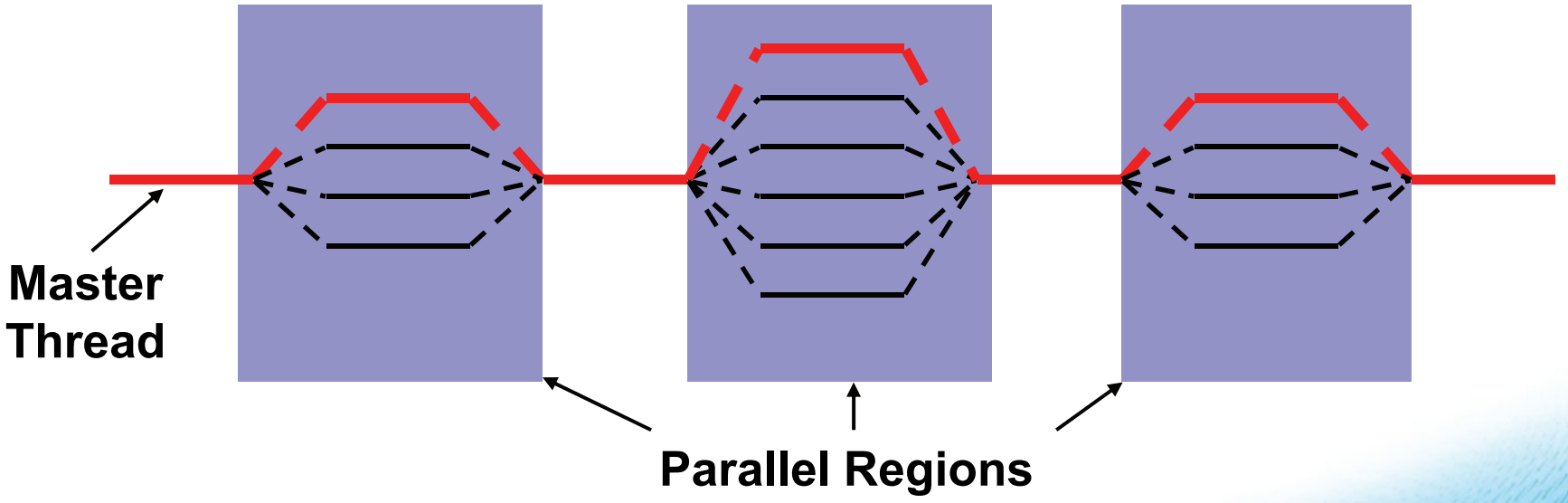
318 Pages

(combined C/C++ and Fortran)

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Programming Model

- **Master thread spawns a team of threads as needed**
- **Parallelism is added incrementally: that is, the sequential program evolves into a parallel program**



Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

A Few Syntax Details to Get Started

- Most of the constructs in OpenMP are compiler directives or pragmas
  - For C and C++, the pragmas take the form:

```
#pragma omp construct [clause [clause]…]
```

  - For Fortran, the directives take one of the forms:

```
C$OMP construct [clause [clause]…]
!$OMP construct [clause [clause]…]
*$OMP construct [clause [clause]…]
```

- Header file or Fortran 90 module

```
#include "omp.h"
use omp_lib
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Agenda

- What is OpenMP?
  - Parallel regions
  - Worksharing
  - Data environment
  - Synchronization
- Optional Advanced topics

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Parallel Region & Structured Blocks (C/C++)

- Most OpenMP constructs apply to structured blocks
  - Structured block: a block with one point of entry at the top and one point of exit at the bottom
  - The only “branches” allowed are STOP statements in Fortran and exit() in C/C++

```
#pragma omp parallel
{
  int id = omp_get_thread_num();

more: res[id] = do_big_job (id);

  if (conv (res[id]) goto more;
}
printf (“All done\n”);
```

A structured block

```
if (go_now()) goto more;
#pragma omp parallel
{
  int id = omp_get_thread_num();
more:  res[id] = do_big_job(id);
  if (conv (res[id]) goto done;
  goto more;
}
done: if (!really_done()) goto more;
```

Not a structured block

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Activity 1: Hello Worlds

- Modify the “Hello, Worlds” serial code to run multithreaded using OpenMP*

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Agenda

- What is OpenMP?
  - Parallel regions
  - Worksharing – Parallel For
  - Data environment
  - Synchronization
- Optional Advanced topics

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Worksharing

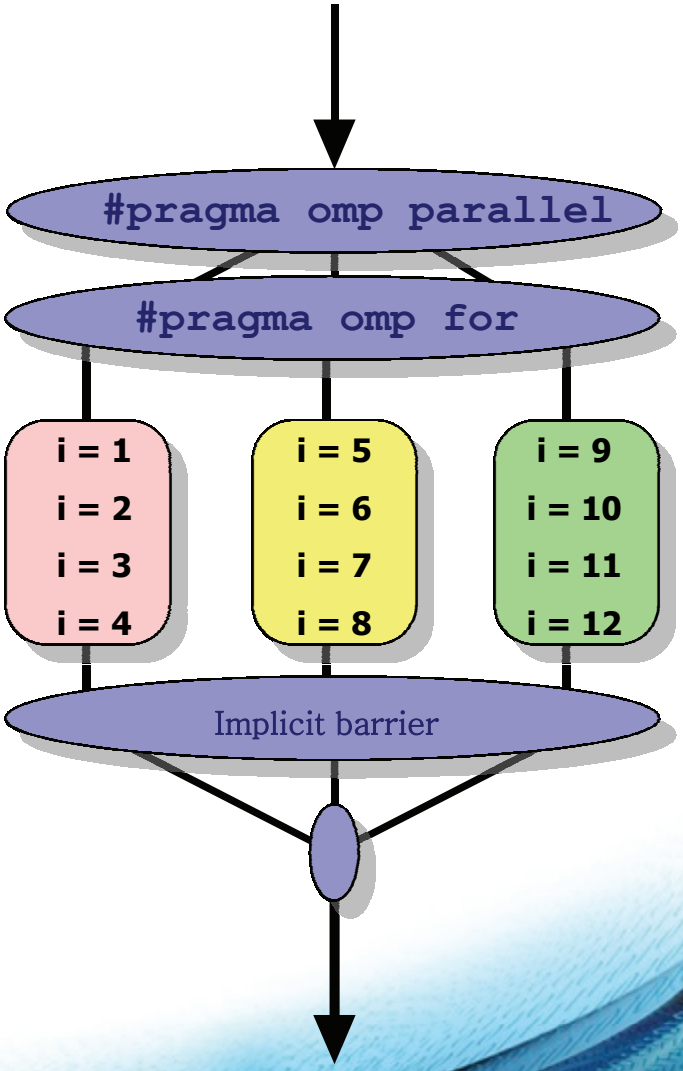
- **Worksharing** is the general term used in OpenMP to describe distribution of work across threads.
- Three examples of worksharing in OpenMP are:
- omp for construct
- om
- omp task construct

Automatically divides work among threads

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

omp **for** construct

```
// assume N=12
#pragma omp parallel
#pragma omp for
   for(i = 1, i < N+1, i++)
      c[i] = a[i] + b[i];
```



- Threads are assigned an independent set of iterations
- Threads must wait at the end of work-sharing construct

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Combining constructs

- These two code segments are equivalent

```
#pragma omp parallel
{
    #pragma omp for
    for (i=0;i< MAX; i++) {
       res[i] = huge();
    }
}
```

```
#pragma omp parallel for
    for (i=0;i< MAX; i++) {
         res[i] = huge();
    }
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

The Private Clause

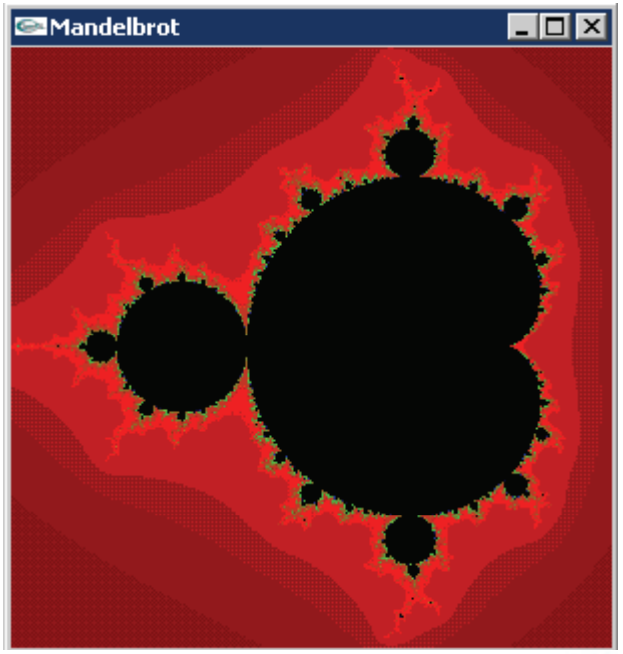
- Reproduces the variable for each task
  - Variables are un-initialized; C++ object is default constructed
  - Any value external to the parallel region is undefined

```
void* work(float* c, int N) {
  float x, y; int i;
 #pragma omp parallel for private(x,y)
     for(i=0; i<N; i++) {
       x = a[i]; y = b[i];
       c[i] = x + y;
     }
}
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Activity 2 – Parallel Mandelbrot

- Objective: create a parallel version of Mandelbrot. Modify the code to add OpenMP worksharing clauses to parallelize the computation of Mandelbrot.
- Follow the next Mandelbrot activity called Mandelbrot in the student lab doc



Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

The schedule clause

- The schedule clause affects how loop iterations are mapped onto threads

```
schedule(static [,chunk])
```

- Blocks of iterations of size “chunk” to threads
- Round robin distribution
- Low overhead, may cause load imbalance

```
schedule(dynamic[,chunk])
```

- Threads grab “chunk” iterations
- When done with iterations, thread requests next set
- Higher threading overhead, can reduce load imbalance

```
schedule(guided[,chunk])
```

- Dynamic schedule starting with large block
- Size of the blocks shrink; no smaller than “chunk”

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Schedule Clause Example

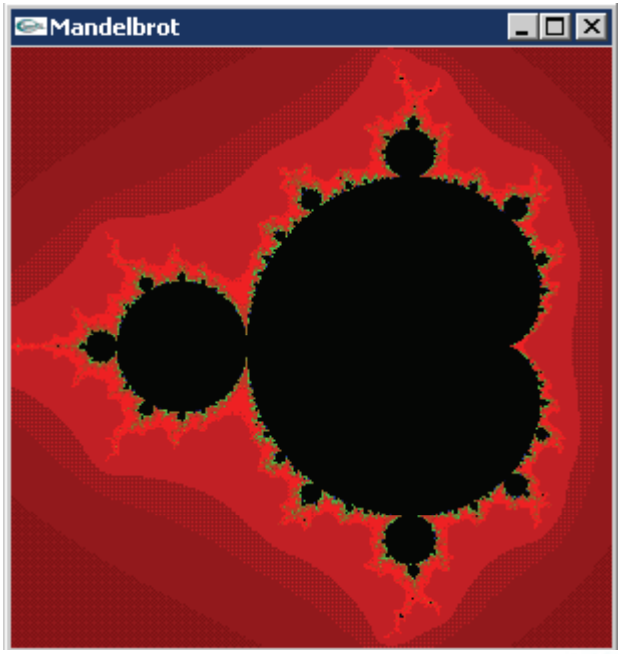
```
#pragma omp parallel for schedule (static, 8)
    for( int i = start; i <= end; i += 2 )
    {
       if ( TestForPrime(i) )  gPrimesFound++;
    }
```

- Iterations are divided into chunks of 8
  - If start = 3, then first chunk is `i`={3,5,7,9,11,13,15,17}

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Activity 2b –Mandelbrot Scheduling

- Objective: create a parallel version of mandelbrot. That uses OpenMP dynamic scheduling
- Follow the next Mandelbrot activity called Mandelbrot Scheduling in the student lab doc



Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

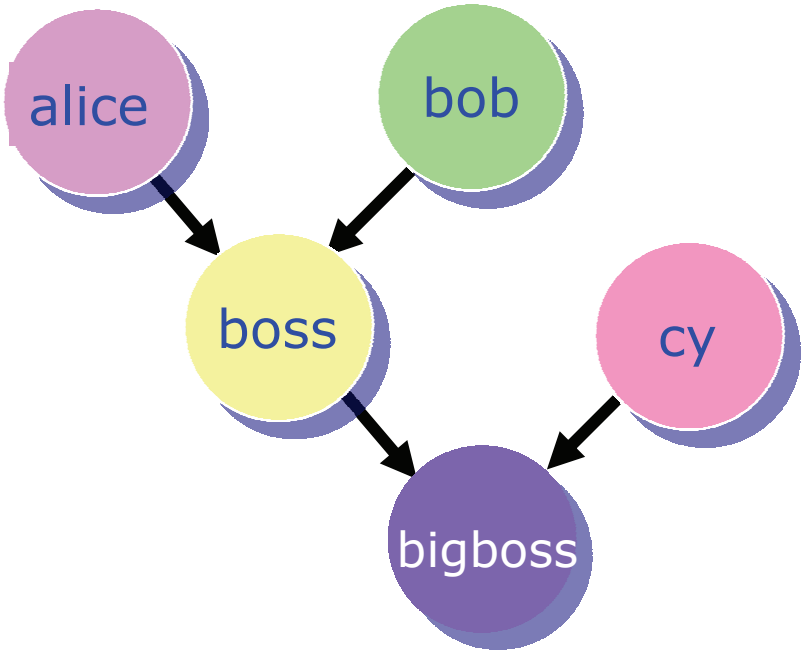
Agenda

- What is OpenMP?
  - Parallel regions
  - Worksharing – Parallel Sections
  - Data environment
  - Synchronization
- Optional Advanced topics

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Task Decomposition

```
a = alice();
b = bob();
s = boss(a, b);
c = cy();
printf ("%6.2f\n",
  bigboss(s,c));
```



alice,bob, and cy can be computed in parallel

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

omp sections

- `#pragma omp sections`
- Must be inside a parallel region
- Precedes a code block containing of *N* blocks of code that may be executed concurrently by *N* threads
- Encompasses each omp section

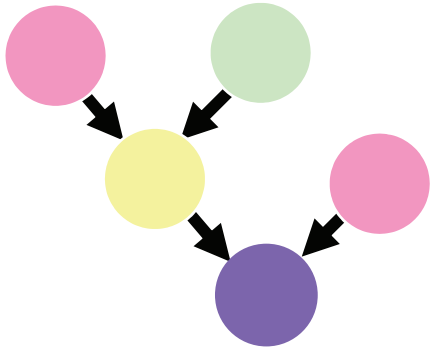
- `#pragma omp section`
- Precedes each block of code within the encompassing block described above
- May be omitted for first parallel section after the parallel sections pragma
- Enclosed program segments are distributed for parallel execution among available threads

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Functional Level Parallelism w sections

```
#pragma omp parallel sections
{
#pragma omp section  /* Optional */
    a = alice();
#pragma omp section
    b = bob();
#pragma omp section
    c = cy();
}

s = boss(a, b);
printf ("%6.2f\n",
          bigboss(s,c));
```

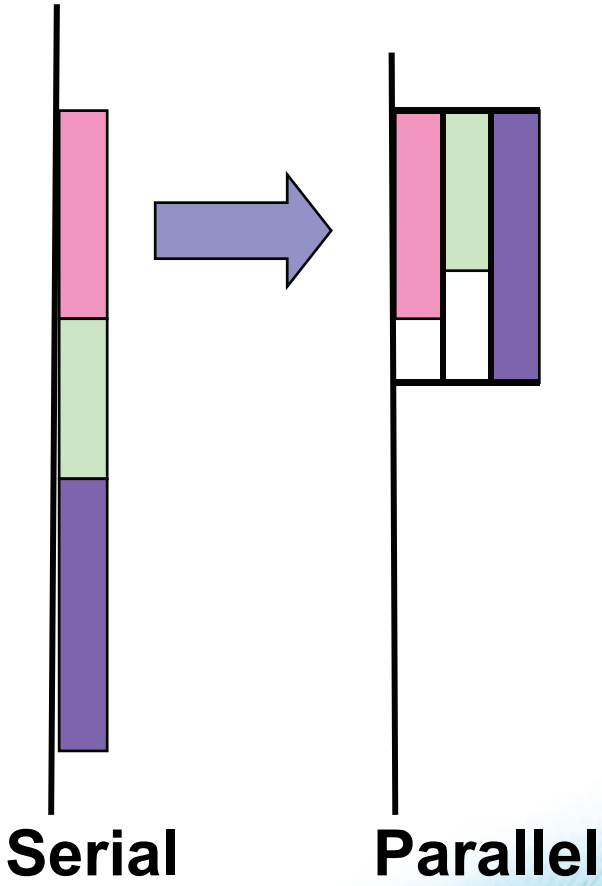


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Advantage of Parallel Sections

- Independent sections of code can execute concurrently – reduce execution time

```
#pragma omp parallel sections
{
    #pragma omp section
    phase1();
    #pragma omp section
    phase2();
    #pragma omp section
    phase3();
}
```



Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Agenda

- What is OpenMP?
  - Parallel regions
  - Worksharing – Tasks
  - Data environment
  - Synchronization
- Optional Advanced topics

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

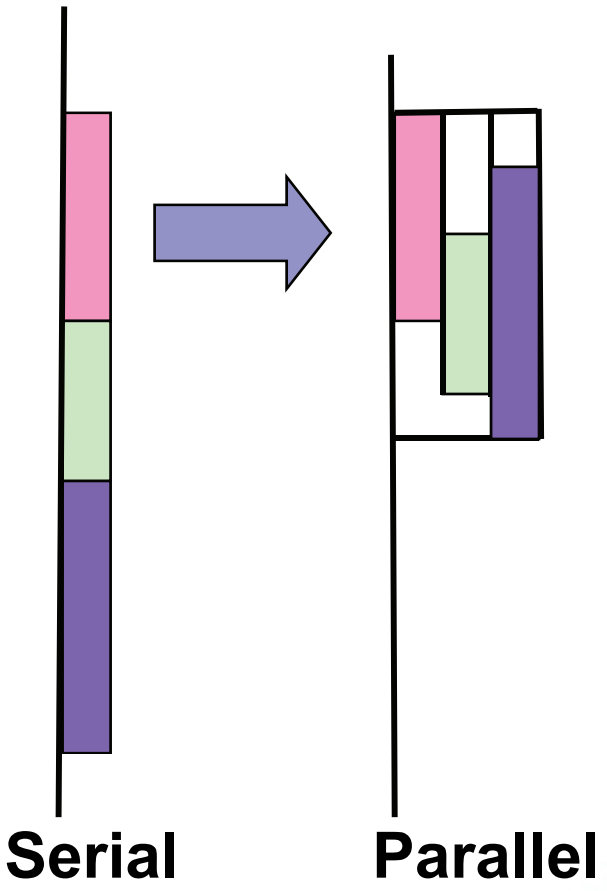
New Addition to OpenMP

- Tasks – Main change for OpenMP 3.0
- Allows parallelization of irregular problems
  - unbounded loops
  - recursive algorithms
  - producer/consumer

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

What are tasks?

- Tasks are independent units of work
- Threads are assigned to perform the work of each task
  - Tasks may be deferred
- Tasks may be executed immediately
- The runtime system decides which of the above
  - Tasks are composed of:
    - **code** to execute
    - **data** environment
    - **internal control variables** (ICV)



Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Simple Task Example

```
#pragma omp parallel
// assume 8 threads
{
 #pragma omp single private(p)
 {
   …
   while (p) {
   #pragma omp task
     {
       processwork(p);
     }
     p = p->next;
   }
 }
}
```

A pool of 8 threads is created here

One thread gets to execute the while loop

The single "while loop" thread creates a task for each instance of processwork()

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Task Construct – Explicit Task View

- A team of threads is created at the omp parallel construct
- A single thread is chosen to execute the while loop – lets call this thread "L"
- Thread L operates the while loop, creates tasks, and fetches next pointers
- Each time L crosses the omp task construct it generates a new task and has a thread assigned to it
- Each task runs in its own thread
- All tasks complete at the barrier at the end of the parallel region's single construct

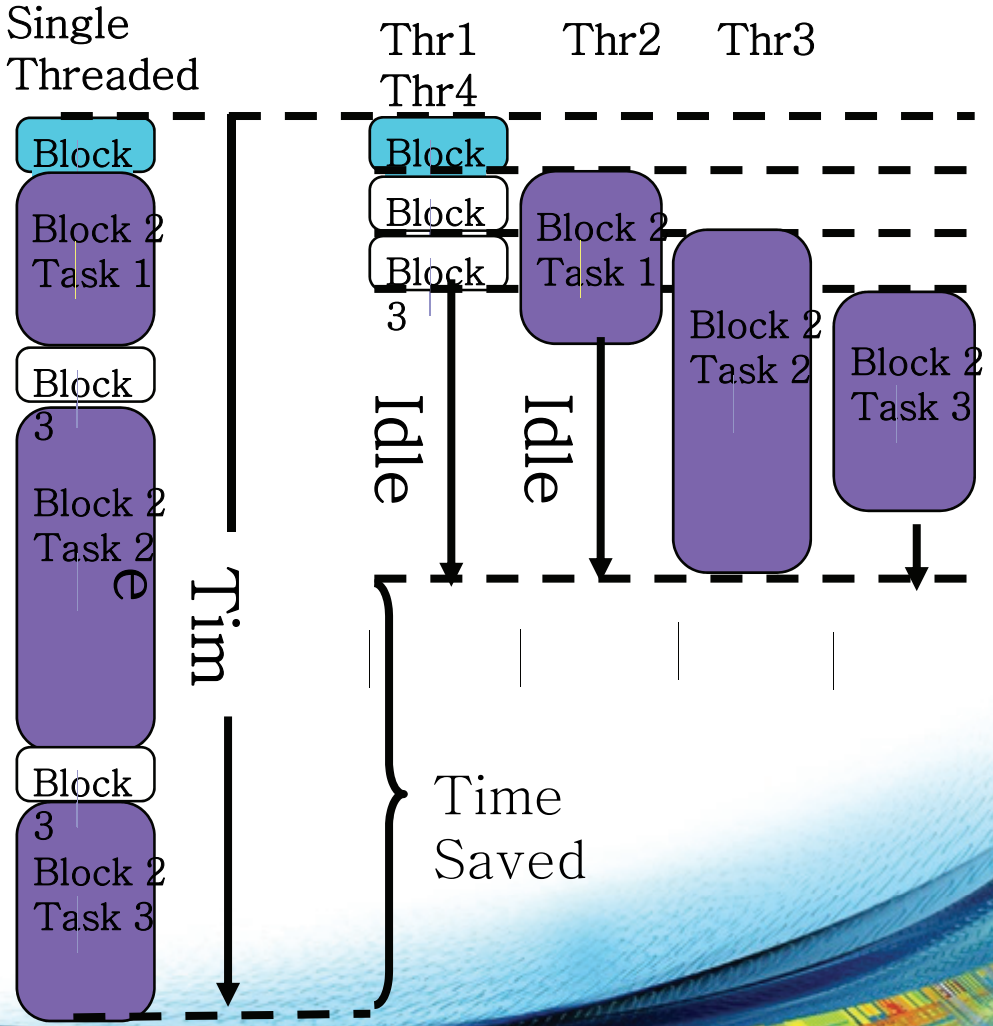
```
#pragma omp parallel
{
  #pragma omp single
  {  // block 1
    node * p = head;
    while (p) {  //block 2
    #pragma omp task private(p)
      process(p);
    p = p->next;  //block 3
    }
  }
}
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Why are tasks useful?

Have potential to parallelize irregular patterns and recursive function

```
#pragma omp parallel
{
   #pragma omp single
   {  // block 1
      node * p = head;
      while (p) {  //block 2
      #pragma omp task
         process(p);
      p = p->next;  //block
      }
   }
}
```



Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Activity 3 – Linked List using Tasks

- **Objective:** Modify the linked list pointer chasing code to implement tasks to parallelize the application
- Follow the Linked List task activity called LinkedListTask in the student lab doc

```
while(p != NULL){
  do_work(p->data);
  p = p->next;
}
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

When are tasks gauranteed to be complete?

- Tasks are gauranteed to be complete:
- At thread or task barriers
  - At the directive: `#pragma omp barrier`
  - At the directive: `#pragma omp taskwait`

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Task Completion Example

```
#pragma omp parallel
{
   #pragma omp task
   foo();
   #pragma omp barrier
   #pragma omp single
   {
      #pragma omp task
      bar();
   }
}
```

Multiple foo tasks created here – one for each thread

All foo tasks guaranteed to be completed here

One bar task created here

bar task guaranteed to be completed here

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Agenda

- What is OpenMP?
  - Parallel regions
  - Worksharing
  - Data environment
  - Synchronization
- Optional Advanced topics

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Data Scoping – What's shared

- OpenMP uses a shared-memory programming model
- **Shared variable** - a *variable* that can be read or written by multiple threads
- Shared clause can be used to make items explicitly shared
  - Global variables are shared by default among tasks
    - File scope variables, namespace scope variables, static variables, Variables with const-qualified type having no mutable member are shared, Static variables which are declared in a scope inside the construct are shared

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Data Scoping – What's private

- But, not everything is shared...
  - Examples of implicitly determined private variables:
    - Stack (local) variables in functions called from parallel regions are PRIVATE
    - Automatic variables within a statement block are PRIVATE
    - Loop iteration variables are private
    - Implicitly declared private variables within tasks will be treated as firstprivate
- **Firstprivate** clause declares one or more list items to be private to a task, and initializes each of them with a value

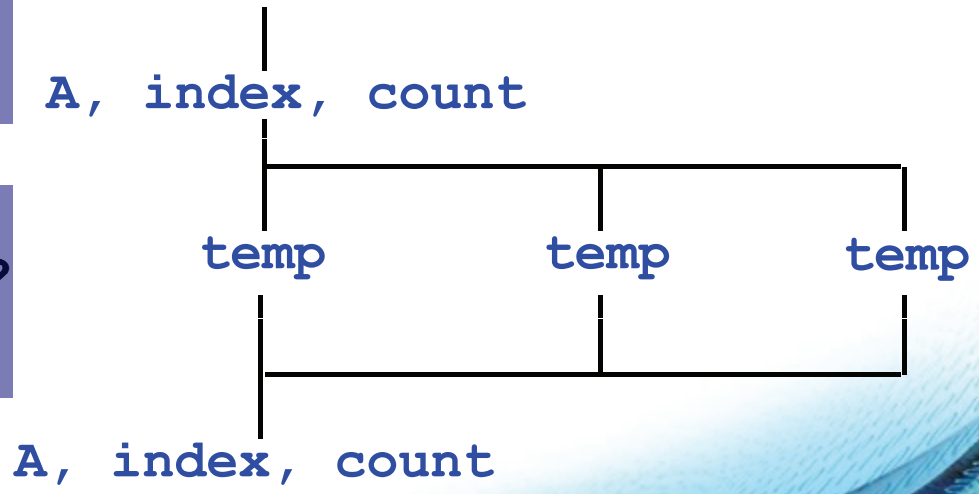
Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

A Data Environment Example

```
float A[10];
main ()
{
  integer index[10];
  #pragma omp parallel
  {
      Work (index);
  }
    printf ("%d\n", index[1]);
}
```

```
extern float A[10];
void Work (int *index)
{
  float temp[10];
  static integer count;
  <...>
}
```

A, ***index***, and ***count*** are shared by all threads, but ***temp*** is local to each thread



Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Data Scoping Issue – fib example

```
int fib ( int n )
{

int x,y;
   if ( n < 2 ) return n;
#pragma omp task
   x = fib(n-1);
#pragma omp task
   y = fib(n-2);
#pragma omp taskwait
   return x+y
}
```

n is private in both tasks

x is a private variable
y is a private variable

What's wrong here?

Can't use private variables outside of tasks

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Data Scoping Example – fib example

```
int fib ( int n )
{

int x,y;
   if ( n < 2 ) return n;
#pragma omp task shared(x)
   x = fib(n-1);
#pragma omp task shared(y)
   y = fib(n-2);
#pragma omp taskwait
   return x+ y;
}
```

n is private in both tasks

x & y are shared
Good solution
we need both values to compute the sum

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Data Scoping Issue – List Traversal

```
List ml; //my_list
Element *e;
#pragma omp parallel
#pragma omp single
{
   for(e=ml->first;e;e=e->next)
#pragma omp task
      process(e);
}
```

What's wrong here?

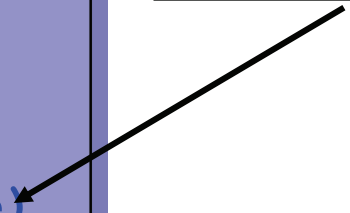
Possible data race !
Shared variable e
updated by multiple tasks

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Data Scoping Example – List Traversal

```
List ml; //my_list
Element *e;
#pragma omp parallel
#pragma omp single
{
   for(e=ml->first;e;e=e->next)
#pragma omp task firstprivate(e)
      process(e);
}
```

Good solution – e is firstprivate



Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Data Scoping Example – List Traversal

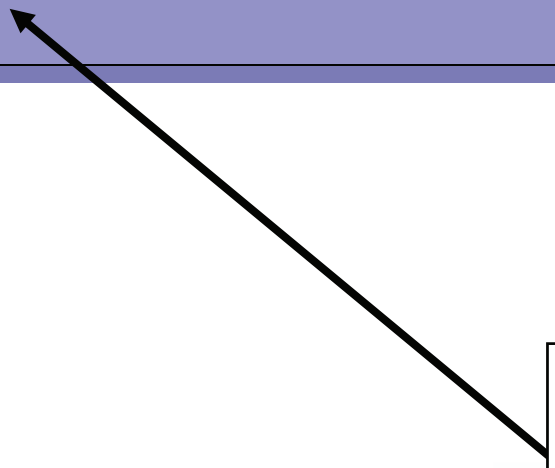
```
List ml; //my_list
Element *e;
#pragma omp parallel
#pragma omp single private(e)
{
   for(e=ml->first;e;e=e->next)
#pragma omp task
      process(e);
}
```

Good solution – e is private

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Data Scoping Example – List Traversal

```
List ml; //my_list
#pragma omp parallel
{
   Element *e;
   for(e=ml->first;e;e=e->next)
#pragma omp task
      process(e);
}
```



Good solution – e is private

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Agenda

- What is OpenMP?
  - Parallel regions
  - Worksharing
  - Data environment
  - Synchronization
- Optional Advanced topics

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Example: Dot Product

```
float dot_prod(float* a, float* b, int N)
{
  float sum = 0.0;
#pragma omp parallel for shared(sum)
   for(int i=0; i<N; i++) {
     sum += a[i] * b[i];
   }
  return sum;
}
```

What is Wrong?

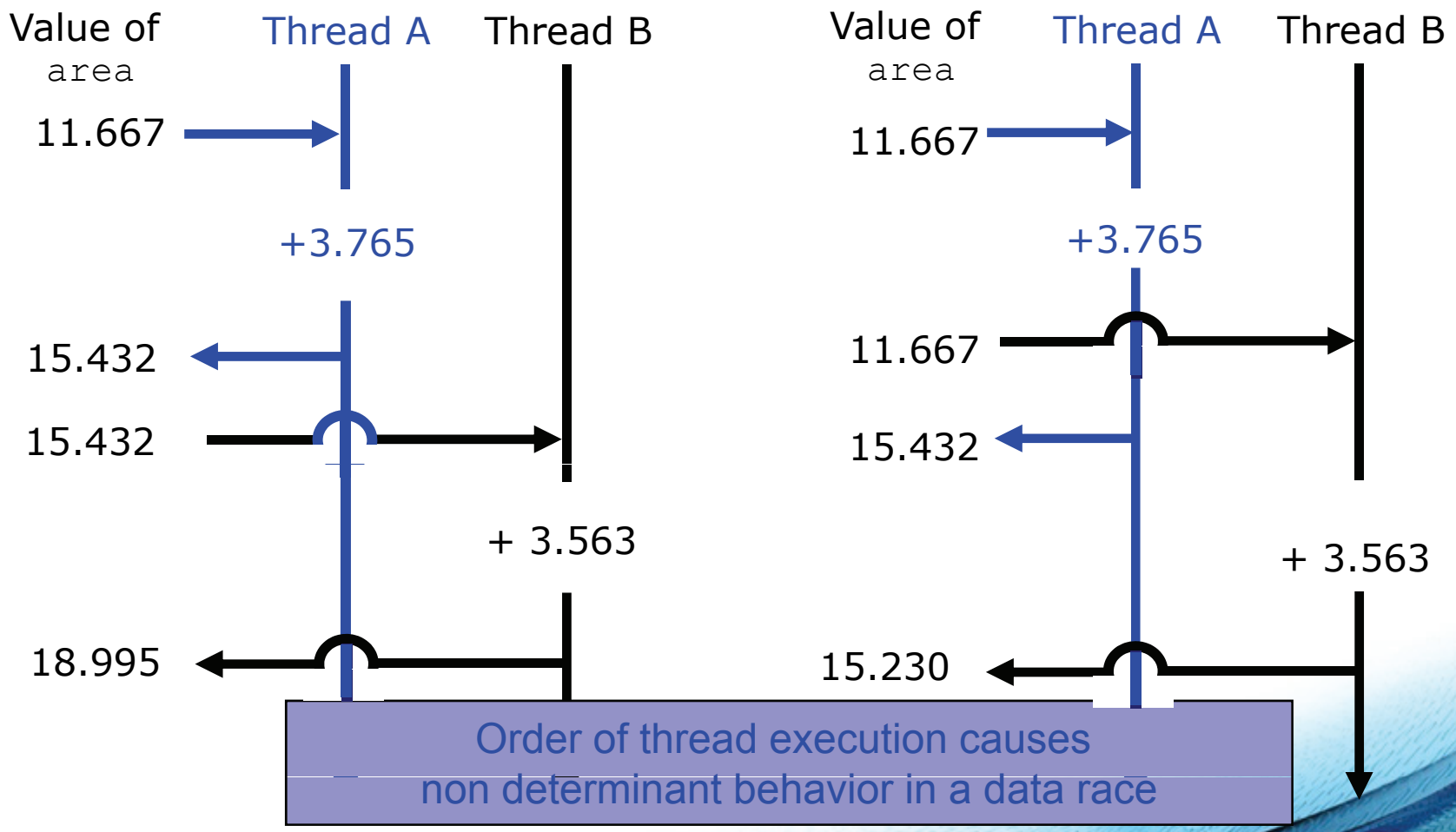
Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Race Condition

- A *race condition* is nondeterministic behavior caused by the times at which two or more threads access a shared variable
- For example, suppose both Thread A and Thread B are executing the statement
- `area += 4.0 / (1.0 + x*x);`

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Two Timings



Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Protect Shared Data

- Must protect access to shared, modifiable data

```
float dot_prod(float* a, float* b, int N)
{
  float sum = 0.0;
#pragma omp parallel for shared(sum)
   for(int i=0; i<N; i++) {
#pragma omp critical
     sum += a[i] * b[i];
   }
  return sum;
}
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

OpenMP* Critical Construct

- `#pragma omp critical [(lock_name)]`
- Defines a critical region on a structured block

Threads wait their turn – only one at a time calls `consum()` thereby protecting RES from race conditions

Naming the critical construct `RES_lock` is optional

```
float RES;
#pragma omp parallel
{ float B;
#pragma omp for
  for(int i=0; i<niters; i++){
    B = big_job(i);
#pragma omp critical (RES_lock)
    consum (B, RES);
  }
}
```

Good Practice – Name all critical sections

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

OpenMP* Reduction Clause

- `reduction (op : list)`
- The variables in "*list*" must be shared in the enclosing parallel region
- Inside parallel or work-sharing construct:
  - A PRIVATE copy of each list variable is created and initialized depending on the "op"
  - These copies are updated locally by threads
  - At end of construct, local copies are combined through "op" into a single value and combined with the value in the original SHARED variable

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

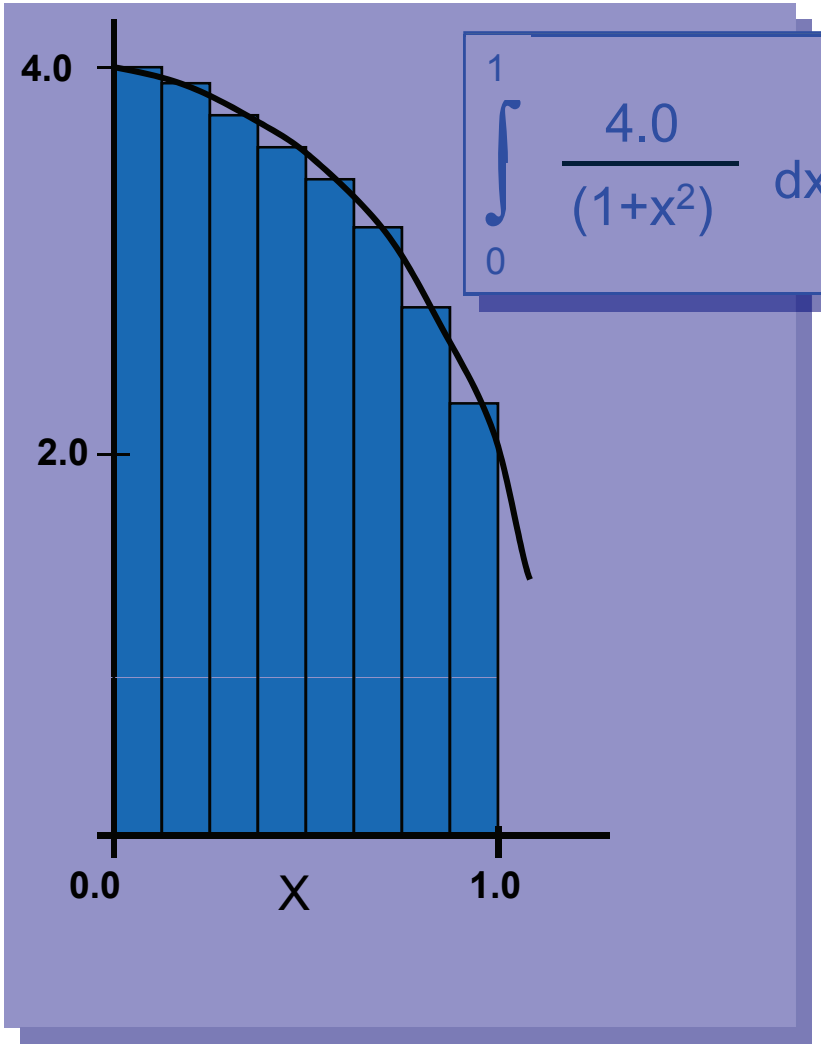
Reduction Example

```
#pragma omp parallel for reduction(+:sum)
   for(i=0; i<N; i++) {
     sum += a[i] * b[i];
   }
```

- Local copy of *sum* for each thread
- All local copies of *sum* added together and stored in “global” variable

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Numerical Integration Example



$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx$$

```
static long num_steps=100000; double
step, pi;

void main()
{  int i;
   double x, sum = 0.0;

   step = 1.0/(double) num_steps;
   for (i=0; i< num_steps; i++){
      x = (i+0.5)*step;
      sum = sum + 4.0/(1.0 + x*x);
   }
   pi = step * sum;
   printf("Pi = %f\n",pi);
}
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

C/C++ Reduction Operations

- A range of associative operands can be used with reduction
- Initial values are the ones that make sense mathematically

Operand	Initial Value
+	0
*	1
-	0
^	0

Operand	Initial Value
&	~0
|	0
&&	1
||	0

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Activity 4 - Computing Pi

```
static long num_steps=100000;
double step, pi;

void main()
{  int i;
   double x, sum = 0.0;

   step = 1.0/(double) num_steps;
   for (i=0; i< num_steps; i++){
      x = (i+0.5)*step;
      sum = sum + 4.0/(1.0 + x*x);
   }
   pi = step * sum;
   printf("Pi = %f\n",pi);
}
```

- Parallelize the numerical integration code using OpenMP
- What variables can be shared?
- What variables need to be private?
- What variables should be set up for reductions?

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Single Construct

- Denotes block of code to be executed by only one thread
  - First thread to arrive is chosen
- Implicit barrier at end

```
#pragma omp parallel
{
   DoManyThings();
#pragma omp single
   {
     ExchangeBoundaries();
   }  // threads wait here for single
   DoManyMoreThings();
}
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Master Construct

- Denotes block of code to be executed only by the master thread
- No implicit barrier at end

```
#pragma omp parallel
{
   DoManyThings();
#pragma omp master
   {                  // if not master skip to next stmt
     ExchangeBoundaries();
   }
   DoManyMoreThings();
}
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Implicit Barriers

- Several OpenMP* constructs have implicit barriers
  - Parallel – necessary barrier – cannot be removed
  - for
  - single
- Unnecessary barriers hurt performance and can be removed with the nowait clause
  - The nowait clause is applicable to:
    - For clause
    - Single clause

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Nowait Clause

```
#pragma omp for nowait
   for(...)
     {...};
```

```
#pragma single nowait
{ [...] }
```

- Use when threads unnecessarily wait between independent computations

```
#pragma omp for schedule(dynamic,1) nowait
 for(int i=0; i<n; i++)
   a[i] = bigFunc1(i);

#pragma omp for schedule(dynamic,1)
 for(int j=0; j<m; j++)
   b[j] = bigFunc2(j);
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Barrier Construct

- Explicit barrier synchronization
- Each thread waits until all threads arrive

```
#pragma omp parallel shared (A, B, C)
{
    DoSomeWork(A,B); // Processed A into B

#pragma omp barrier

     DoSomeWork(B,C); // Processed B into C

}
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Atomic Construct

- Special case of a critical section
- Applies only to simple update of memory location

```
#pragma omp parallel for shared(x, y, index, n)
   for (i = 0; i < n; i++) {
      #pragma omp atomic
        x[index[i]] += work1(i);
      y[i] += work2(i);
   }
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Agenda

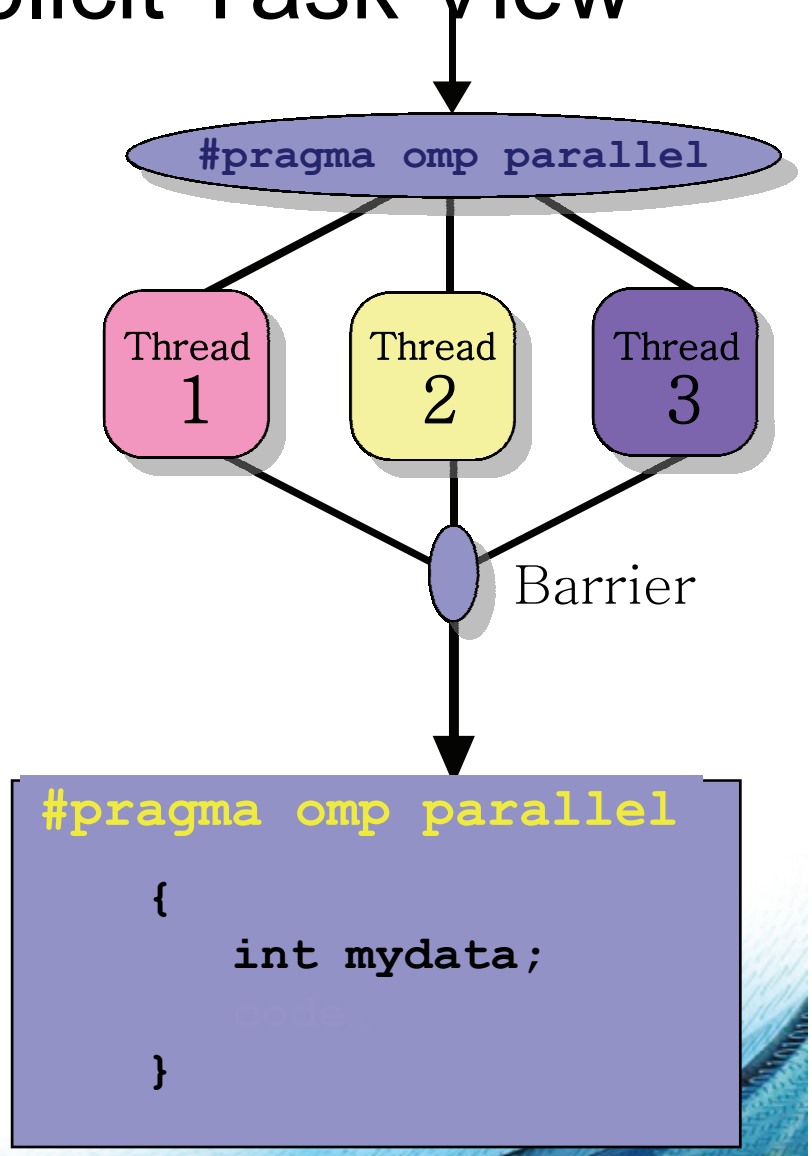
- What is OpenMP?
  - Parallel regions
  - Worksharing
  - Data environment
  - Synchronization
- Optional Advanced topics

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Advanced Concepts

Parallel Construct – Implicit Task View

- Tasks are created in OpenMP even without an explicit task directive.
- Lets look at how tasks are created implicitly for the code snippet below
  - Thread encountering parallel construct packages up a set of *implicit* tasks
  - Team of threads is created.
  - Each thread in team is assigned to one of the tasks (and *tied* to it).
  - Barrier holds original master thread until all implicit tasks are finished.



Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Task Construct

#pragma omp task [*clause*[[,]*clause*] ...]
structured-block

where *clause* **can be one of:**

if (*expression*)
untied
shared (*list*)
private (*list*)
firstprivate (*list*)
default(shared | none)

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Tied & Untied Tasks

- Tied Tasks:
  - A tied task gets a thread assigned to it at its first execution and the same thread services the task for its lifetime
  - A thread executing a tied task, can be suspended, and sent of to execute some other task, but eventually, the same thread will return to resume execution of its original tied task
  - Tasks are tied unless explicitly declared untied

- Untied Tasks:
  - An united task has no long term association with any given thread. Any thread not otherwise occupied is free to execute an untied task. The thread assigned to execute an untied task may only change at a "task scheduling point".
  - An untied task is created by appending “untied” to the task clause
  - Example: #pragma omp task untied

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Task switching

- **task switching** The act of a *thread* switching from the execution of one *task* to another *task.*

- The purpose of task switching is distribute threads among the unassigned tasks in the team to avoid piling up long queues of unassigned tasks

- Task switching, for tied tasks, can only occur at task scheduling points located within the following constructs

  - encountered **task** *constructs*
  - encountered **taskwait** *constructs*
  - encountered **barrier** *directives*
  - implicit **barrier** *regions*
  - at the end of the *tied task region*

- Untied tasks have implementation dependent scheduling points

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Task switching example

- The thread executing the “for loop” , AKA the generating task, generates many tasks in a short time so...
- The SINGLE generating task will have to suspend for a while when “task pool” fills up
  - Task switching is invoked to start draining the “pool”
  - When “pool” is sufficiently drained – then the single task can being generating more tasks again

```
#pragma omp single
{
    for (i=0; i<ONEZILLION; i++)
     #pragma omp task
         process(item[i]);
    }
}
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Optional foil - OpenMP* API

- Get the thread number within a team

```
int omp_get_thread_num(void);
```

- Get the number of threads in a team

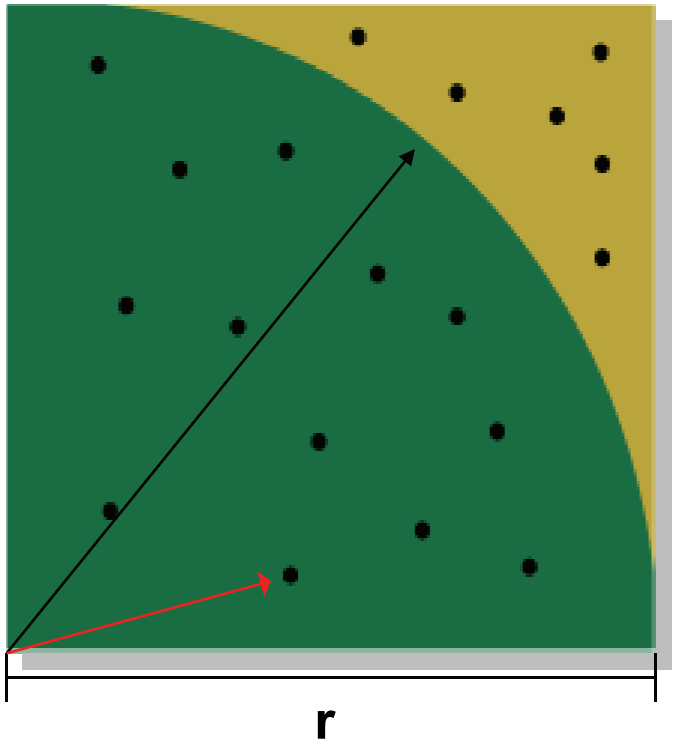
```
int omp_get_num_threads(void);
```

- Usually not needed for OpenMP codes
  - Can lead to code not being serially consistent
  - Does have specific uses (debugging)
  - Must include a header file

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


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Optional foil - Monte Carlo Pi



$$\frac{\#\text{ of darts hitting circle}}{\#\text{ of darts in square}} = \frac{1/4\pi r^2}{r^2}$$

$$\pi = 4\frac{\#\text{ of darts hitting circle}}{\#\text{ of darts in square}}$$

```
loop 1 to MAX
   x.coor=(random#)
   y.coor=(random#)
   dist=sqrt(x^2 + y^2)
   if (dist <= 1)
      hits=hits+1

 pi = 4 * hits/MAX
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Optional foil - Making Monte Carlo's Parallel

```
hits = 0
call SEED48(1)
DO I = 1, max
    x = DRAND48()
    y = DRAND48()
    IF (SQRT(x*x + y*y) .LT. 1)
  THEN
        hits = hits+1
    ENDIF
END DO
pi = REAL(hits)/REAL(max) * 4.0
```

What is the challenge here?

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Optional Activity 5: Computing Pi

- Use the Intel® Math Kernel Library (Intel® MKL) VSL:
  - Intel MKL's VSL (Vector Statistics Libraries)
  - VSL creates an array, rather than a single random number
  - VSL can have multiple seeds (one for each thread)
- Objective:
  - Use basic OpenMP* syntax to make Pi parallel
  - Choose the best code to divide the task up
  - Categorize properly all variables

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Firstprivate Clause

- Variables initialized from shared variable
- C++ objects are copy-constructed

```
incr=0;
#pragma omp parallel for firstprivate(incr)
for (I=0;I<=MAX;I++) {
      if ((I%2)==0) incr++;
      A(I)=incr;
}
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Lastprivate Clause

- Variables update shared variable using value from last iteration
- C++ objects are updated as if by assignment

```
void sq2(int n,
         double *lastterm)
{
  double x; int i;
  #pragma omp parallel
  #pragma omp for lastprivate(x)
  for (i = 0; i < n; i++){
     x = a[i]*a[i] + b[i]*b[i];
     b[i] = sqrt(x);
  }
  lastterm = x;
}
```


Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Threadprivate Clause

- Preserves global scope for per-thread storage
- Legal for name-space-scope and file-scope
- #pragma omp threadprivate(A) from master thread

```
struct Astruct A;
…

#pragma omp parallel copyin(A)
  do_something_to(&A);
…

#pragma omp parallel
  do_something_else_to(&A);
```

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

20+ Library Routines

- Runtime environment routines:
  - Modify/check the number of threads

    `omp_[set|get]_num_threads()`

    `omp_get_thread_num()`

    `omp_get_max_threads()`
  - Are we in a parallel region?

    `omp_in_parallel()`
  - How many processors in the system?

    `omp_get_num_procs()`
  - Explicit locks

    `omp_[set|unset]_lock()`
  - And many more...

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

Library Routines

- To fix the number of threads used in a program
  - Set the number of threads
  - Then save the number returned

```
#include <omp.h>

void main ()
{
  int num_threads;
  omp_set_num_threads (omp_num_procs ());

  #pragma omp parallel
  {
    int id = omp_get_thread_num ();

    #pragma omp single
    num_threads = omp_get_num_threads ();

    do_lots_of_stuff (id);
  }
}
```

Request as many threads as you have processors.

Protect this operation because memory stores are not atomic

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

intel

BACKUP

Copyright © 2014, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.