What is OpenMP?

OpenMP is an API for parallel programming

First developed by the OpenMP Architecture Review Board (1997), now a standard

Designed for shared-memory multiprocessors

Set of compiler directives, library functions, and environment variables, but not a language

Can be used with C, C++, or Fortran

Based on fork/join model of threads

Copyright © 2009, 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.

Fork/Join Programming Model

When program begins execution, only master thread active

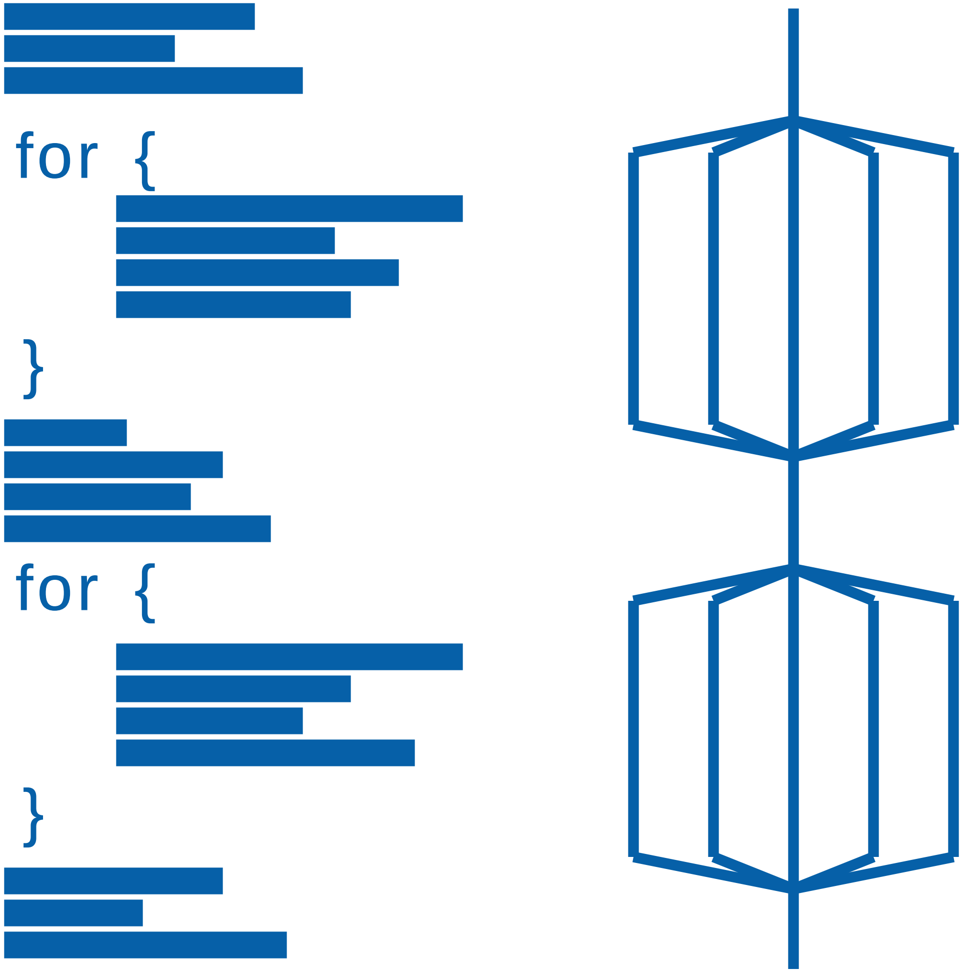
Master thread executes sequential portions of program

For parallel portions of program, master thread forks (creates or awakens) additional threads

At join (end of parallel section of code), extra threads are suspended or die

Copyright © 2009, 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.

Relating Fork/Join to Code



Sequential code

Parallel code

Sequential code

Parallel code

Sequential code

Copyright © 2009, 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.

Incremental Parallelization

Sequential program a special case of threaded program

Programmers can add parallelism incrementally

Profile program execution

Repeat

- Choose best opportunity for parallelization
- Transform sequential code into parallel code

Until further improvements not worth the effort

Copyright © 2009, 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.

Syntax of Compiler Directives

A C/C++ compiler directive is called a *pragma*

Pragmas are handled by the preprocessor

All OpenMP pragmas have the syntax:

`#pragma omp` <*rest of pragma*>

Pragmas appear immediately before relevant construct

Copyright © 2009, 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.

Pragma: parallel for

The compiler directive

```
#pragma omp parallel for
```

tells the compiler that the for loop which immediately follows can be executed in parallel

The number of loop iterations must be computable at run time before loop executes

Loop must not contain a break, return, or exit

Loop must not contain a goto to a label outside loop

Copyright © 2009, 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

```
int first, *marked, prime, size;
...
#pragma omp parallel for
for (i = first; i < size; i += prime)
   marked[i] = 1;
```

Threads are assigned an independent set of iterations

Threads must wait at the end of construct

Copyright © 2009, 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.

Pragma: parallel

Sometimes the code that should be executed in parallel goes beyond a single for loop

The `parallel` pragma is used when a block of code should be executed in parallel

```
#pragma omp parallel
{
  DoSomeWork(res, M);
  DoSomeOtherWork(res, M);
}
```

Copyright © 2009, 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.

Pragma: for

The for pragma can be used inside a block of code already marked with the parallel pragma

Loop iterations should be divided among the active threads

There is a *barrier synchronization* at the end of the for loop

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

Copyright © 2009, 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.

Which Loop to Make Parallel?

```
main () {
int i, j, k;
float **a, **b;
...
for (k = 0; k < N; k++)            Loop-carried dependences
   for (i = 0; i < N; i++)         Can execute in parallel
      for (j = 0; j < N; j++)      Can execute in parallel
         a[i][j] = MIN(a[i][j], a[i][k] + a[k][j]);
```

Copyright © 2009, 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.

Minimizing Threading Overhead

There is a fork/join for every instance of

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

Since fork/join is a source of overhead, we want to maximize the amount of work done for each fork/join

Hence we choose to make the middle loop parallel

Copyright © 2009, 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.

Almost Right, but Not Quite

```
main () {
int i, j, k;
float **a, **b;
...
for (k = 0; k < N; k++)
  #pragma omp parallel for
  for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
      a[i][j] = MIN(a[i][j], a[i][k] + a[k][j]);
```

Problem: j is a shared variable

Copyright © 2009, 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.

Problem Solved with private Clause

```
main () {
int i, j, k;
float **a, **b;
...
for (k = 0; k < N; k++)
   #pragma omp parallel for private (j)
   for (i = 0; i < N; i++)
      for (j = 0; j < N; j++)
         a[i][j] = MIN(a[i][j], a[i][k] + a[k][j]);
```

Tells compiler to make listed variables private

Copyright © 2009, 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 thread

- 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 © 2009, 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 private(sum)
   for(int i = 0; i < N; i++) {
     sum += a[i] * b[i];
   }
  return sum;
}
```

Why won’t the use of the **private** clause work in this example?

Copyright © 2009, 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.

Reductions

Given associative binary operator ⊕ the expression

$$a_1 \oplus a_2 \oplus a_3 \oplus \ldots \oplus a_n$$

is called a *reduction*

Copyright © 2009, 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

Reductions are so common that OpenMP provides a reduction clause for the `parallel for` pragma

```
reduction (op : list)
```

A PRIVATE copy of each list variable is created and initialized depending on the “op”

- The identity value “op” (e.g., 0 for addition)

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 © 2009, 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];
   }
```

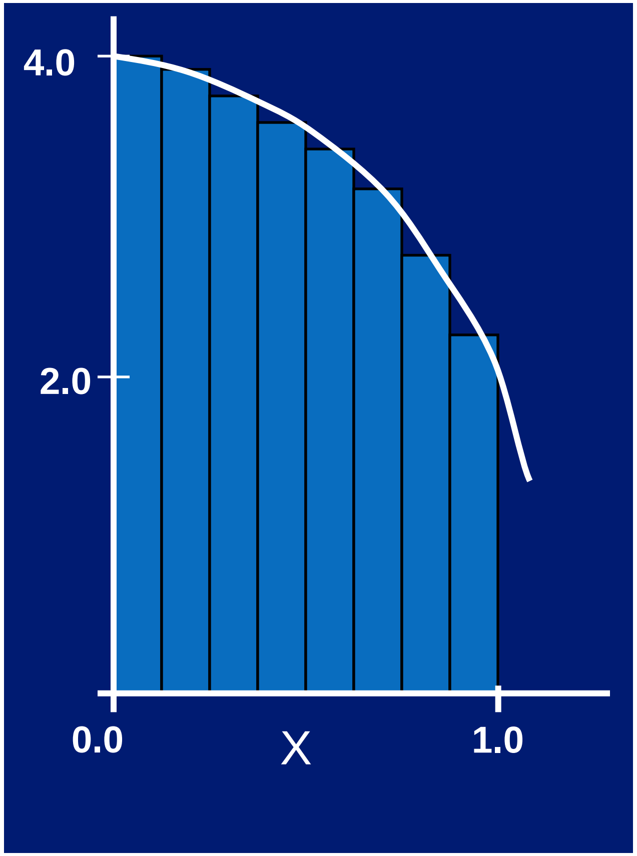
Local copy of **sum** for each thread

All local copies of sum added together and stored in shared copy

Copyright © 2009, 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

$$\int_{0}^{1} \frac{4.0}{(1+x^2)}\, dx = \pi$$



```
static long num_rects=100000;
double width, pi;

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

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

Copyright © 2009, 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: What’s Shared?

```
static long num_rects=100000;
double width, pi;

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

   width = 1.0/(double) num_rects;
   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);
}
```

What variables can be shared?

`width, num_rects`

Copyright © 2009, 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: What’s Private?

```
static long num_rects=100000;
double width, pi;

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

   width = 1.0/(double) num_rects;
   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);
}
```

What variables need to be private?

`x, i`

Copyright © 2009, 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: Any Reductions?

```
static long num_rects=100000;
double width, pi;

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

   width = 1.0/(double) num_rects;
   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);
}
```

What variables should be set up for reduction?

sum

Copyright © 2009, 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.

Solution to Computing Pi

```
static long num_rects=100000;
double width, pi;

void main()
{  int i;
   double x, sum = 0.0;
#pragma omp parallel for private(x) reduction(+:sum)
   width = 1.0/(double) num_rects;
   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 © 2009, 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.

References

OpenMP API Specification, www.openmp.org.

Rohit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, Ramesh Menon, *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers (2001).

Barbara Chapman, Gabriele Jost, Ruud van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, MIT Press (2008).

Barbara Chapman, "OpenMP: A Roadmap for Evolving the Standard (PowerPoint slides)," http://www.hpcs.cs.tsukuba.ac.jp/events/wompei2003/slides/barbara.pdf

Michael J. Quinn, *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill (2004).

Copyright © 2009, 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®