A “Hands-on” Introduction to OpenMP*

Tim Mattson

Intel Corp.

timothy.g.mattson@intel.com

* The name “OpenMP” is the property of the OpenMP Architecture Review Board.

Introduction

- OpenMP is one of the most common parallel programming models in use today.
- It is relatively easy to use which makes a great language to start with when learning to write parallel software.
- Assumptions:
  - We assume you know C. OpenMP supports Fortran and C++, but we will restrict ourselves to C.
  - We assume you are new to parallel programming.
  - We assume you have access to a compiler that supports OpenMP (more on that later).

Acknowledgements

- This course is based on a long series of tutorials presented at Supercomputing conferences.   The following people helped prepare this content:
  - J. Mark Bull (the University of Edinburgh)
  - Rudi Eigenmann (Purdue University)
  - Barbara Chapman (University of Houston)
  - Larry Meadows, Sanjiv Shah, and Clay Breshears (Intel Corp).

- Some slides are based on a course I teach with Kurt Keutzer of UC Berkeley. The course is called “CS194: Architecting parallel applications with design patterns”. These slides are marked with the UC Berkeley ParLab logo:

Preliminaries:

- Our plan ... Active learning!
  - We will mix short lectures with short exercises.
- Download exercises and reference materials.
- Please follow these simple rules
  - Do the exercises we assign and then change things around and experiment.
    - Embrace active learning!
  - Don’t cheat: Do Not look at the solutions before you complete an exercise … even if you get really frustrated.

Outline

- ➡ Unit 1: Getting started with OpenMP
  - Mod1: Introduction to parallel programming
  - Mod 2: The boring bits: Using an OpenMP compiler (hello world)
  - Disc 1: Hello world and how threads work
- Unit 2: The core features of OpenMP
  - Mod 3: Creating Threads (the Pi program)
  - Disc 2: The simple Pi program and why it sucks
  - Mod 4: Synchronization (Pi program revisited)
  - Disc 3: Synchronization overhead and eliminating false sharing
  - Mod 5: Parallel Loops (making the Pi program simple)
  - Disc 4: Pi program wrap-up
- Unit 3: Working with OpenMP
  - Mod 6: Synchronize single masters and stuff
  - Mod 7: Data environment
  - Disc 5: Debugging OpenMP programs
  - Mod 8: Skills practice … linked lists and OpenMP
  - Disc 6: Different ways to traverse linked lists
- Unit 4: a few advanced OpenMP topics
  - Mod 8: Tasks (linked lists the easy way)
  - Disc 7: Understanding Tasks
  - Mod 8: The scary stuff … Memory model, atomics, and flush (pairwise synch).
  - Disc 8: The pitfalls of pairwise synchronization
  - Mod 9: Threadprivate Data and how to support libraries (Pi again)
  - Disc 9: Random number generators
- Unit 5: Recapitulation

Outline

- Unit 1: Getting started with OpenMP
  - ➡ Mod1: Introduction to parallel programming
  - Mod 2: The boring bits: Using an OpenMP compiler (hello world)
  - Disc 1: Hello world and how threads work
- Unit 2: The core features of OpenMP
  - Mod 3: Creating Threads (the Pi program)
  - Disc 2: The simple Pi program and why it sucks
  - Mod 4: Synchronization (Pi program revisited)
  - Disc 3: Synchronization overhead and eliminating false sharing
  - Mod 5: Parallel Loops (making the Pi program simple)
  - Disc 4: Pi program wrap-up
- Unit 3: Working with OpenMP
  - Mod 6: Synchronize single masters and stuff
  - Mod 7: Data environment
  - Disc 5: Debugging OpenMP programs
  - Mod 8: Skills practice … linked lists and OpenMP
  - Disc 6: Different ways to traverse linked lists
- Unit 4: a few advanced OpenMP topics
  - Mod 8: Tasks (linked lists the easy way)
  - Disc 7: Understanding Tasks
  - Mod 8: The scary stuff … Memory model, atomics, and flush (pairwise synch).
  - Disc 8: The pitfalls of pairwise synchronization
  - Mod 9: Threadprivate Data and how to support libraries (Pi again)
  - Disc 9: Random number generators
- Unit 5: Recapitulation

Concurrency vs. Parallelism

- Two important definitions:
  - Concurrency: A condition of a system in which multiple tasks are *logically* active at one time.
  - Parallelism: A condition of a system in which multiple tasks are actually active at one time.

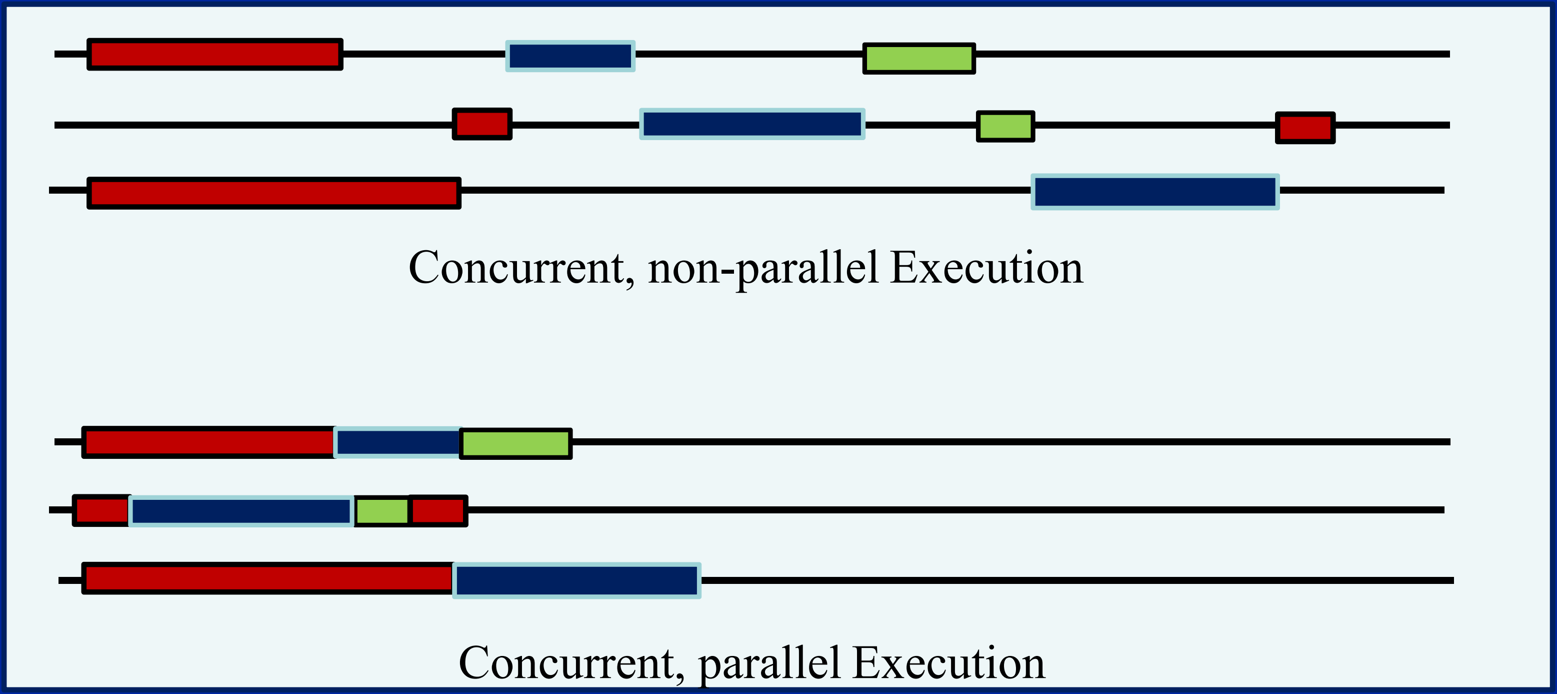

Concurrent, non-parallel Execution
Concurrent, parallel Execution

Figure from "An Introduction to Concurrency in Programming Languages" by J. Sottile, Timothy G. Mattson, and Craig E Rasmussen, 2010

Concurrency vs. Parallelism

- Two important definitions:
  - Concurrency: A condition of a system in which multiple tasks are *logically* active at one time.
  - Parallelism: A condition of a system in which multiple tasks are actually active at one time.

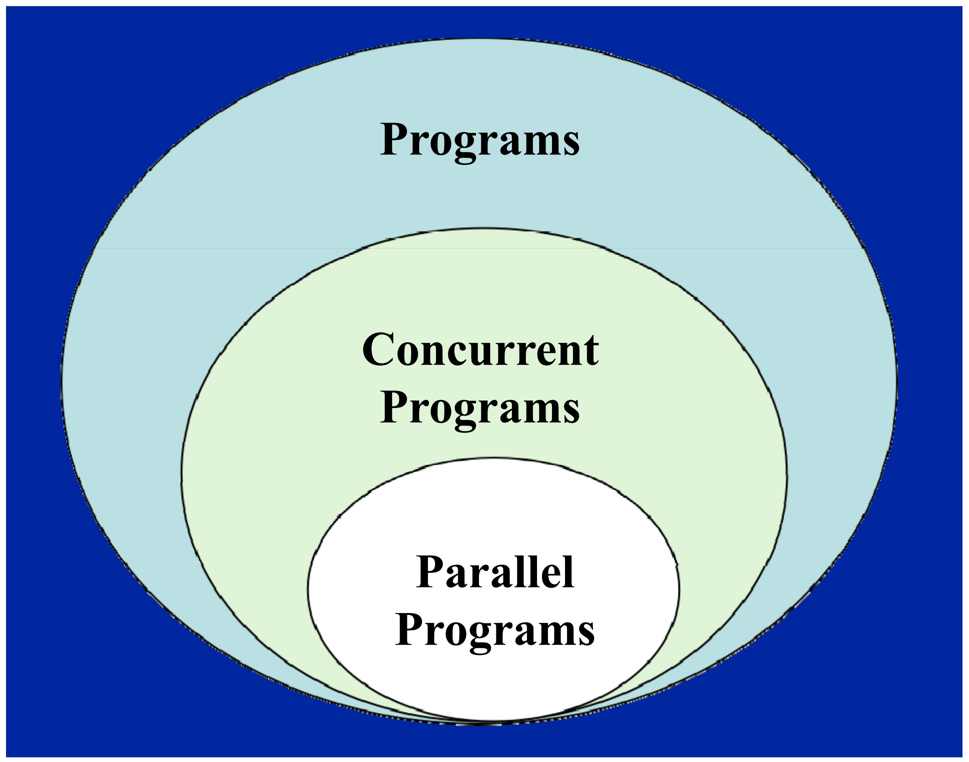
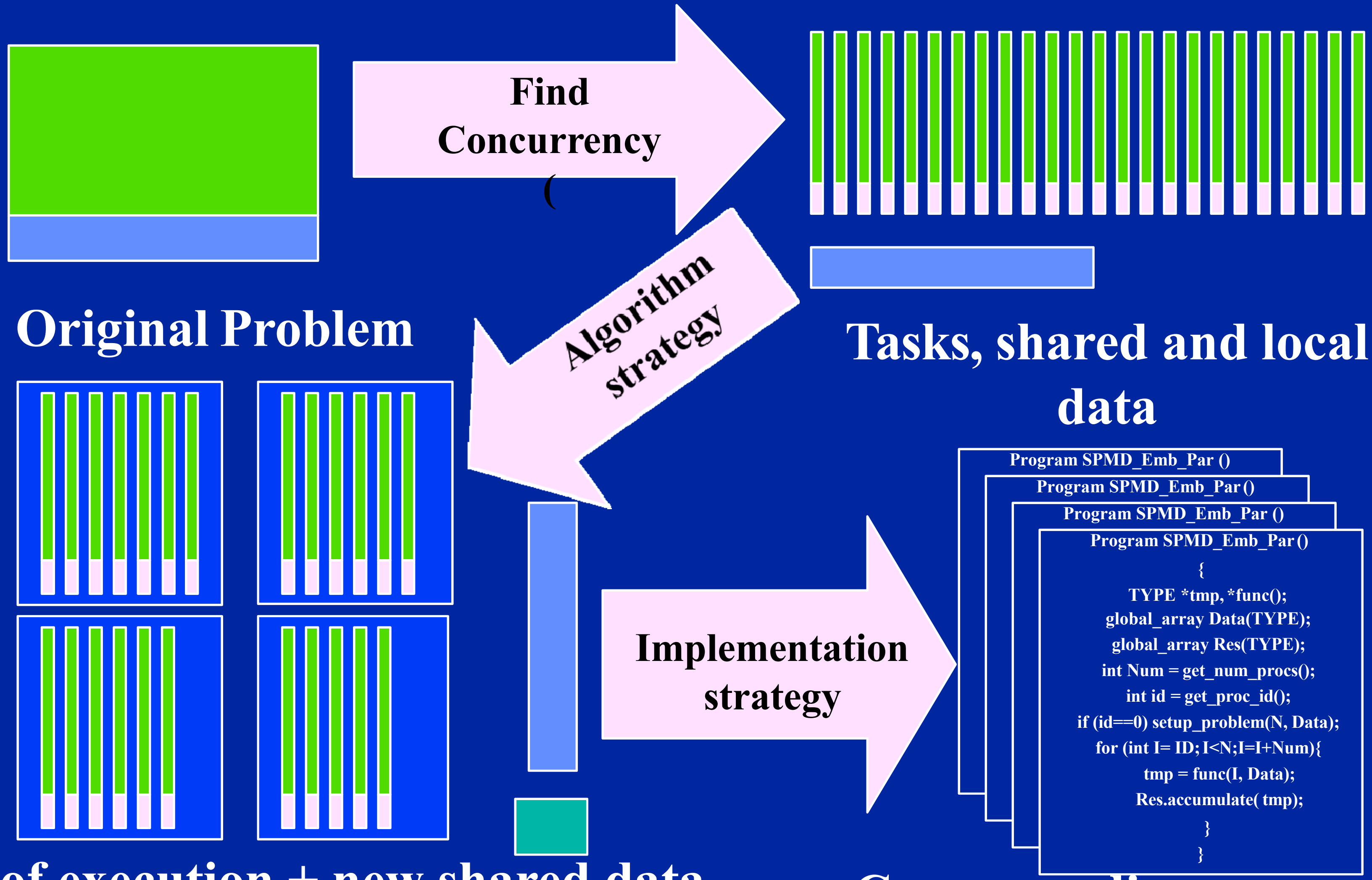


Figure from “An Introduction to Concurrency in Programming Languages” by J. Sottile, Timothy G. Mattson, and Craig E Rasmussen, 2010

Concurrent vs. Parallel applications

- We distinguish between two classes of applications that exploit the concurrency in a problem:
  - Concurrent application: An application for which computations **logically** execute simultaneously due to the semantics of the application.
  - Parallel application: An application for which the computations **actually** execute simultaneously in order to complete a problem in less time.

The Parallel programming process:



Find Concurrency

(

Original Problem

Tasks, shared and local data

Algorithm strategy

Implementation strategy

```
Program SPMD_Emb_Par ()
Program SPMD_Emb_Par ()
Program SPMD_Emb_Par ()
Program SPMD_Emb_Par ()
{
TYPE *tmp, *func();
global_array Data(TYPE);
global_array Res(TYPE);
int Num = get_num_procs();
int id = get_proc_id();
if (id==0) setup_problem(N, Data);
for (int I= ID; I<N;I=I+Num){
tmp = func(I, Data);
Res.accumulate( tmp);
}
}
```

Units of execution + new shared data for extracted dependencies

Corresponding source code

OpenMP* Overview:

```
C$OMP FLUSH
#pragma omp critical
C$OMP THREADPRIVATE(/ABC/)
CALL OMP_SET_NUM_THREADS(10)
C$ON
C$ON
C$(
ED
C:
#p
```

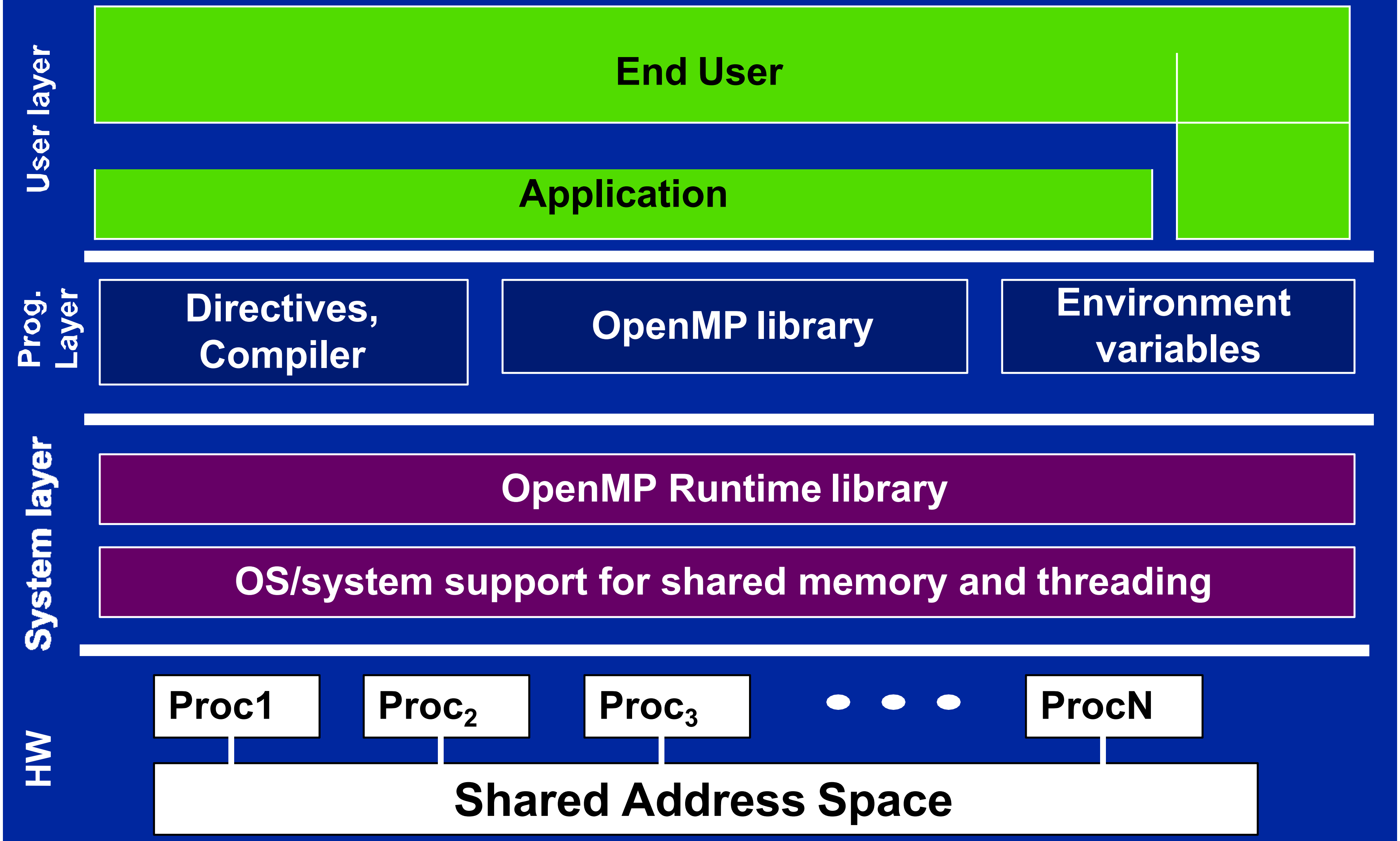
OpenMP: An API for Writing Multithreaded Applications

- A set of compiler directives and library routines for parallel application programmers
- Greatly simplifies writing multi-threaded (MT) programs in Fortran, C and C++
- Standardizes last 20 years of SMP practice

```
C$OMP PARALLEL COPYIN(/blk/)
C$OMP DO lastprivate(XX)
Nthrds = OMP_GET_NUM_PROCS()
omp_set_lock(lck)
```

* The name “OpenMP” is the property of the OpenMP Architecture Review Board.

OpenMP Basic Defs: Solution Stack



User layer

End User

Application

Prog. Layer

Directives, Compiler

OpenMP library

Environment variables

System layer

OpenMP Runtime library

OS/system support for shared memory and threading

HW

Proc1

Proc_2

Proc_3

• • •

ProcN

Shared Address Space

OpenMP core syntax

- Most of the constructs in OpenMP are compiler directives.

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

  - Example

    #pragma omp parallel num_threads(4)

- Function prototypes and types in the file:

  #include <omp.h>

- Most OpenMP* constructs apply to a "structured block".
  - Structured block: a block of one or more statements with one point of entry at the top and one point of exit at the bottom.
  - It's OK to have an exit() within the structured block.

Outline

- Unit 1: Getting started with OpenMP
  - Mod1: Introduction to parallel programming
  - ➡ Mod 2: The boring bits: Using an OpenMP compiler (hello world)
  - Disc 1: Hello world and how threads work
- Unit 2: The core features of OpenMP
  - Mod 3: Creating Threads (the Pi program)
  - Disc 2: The simple Pi program and why it sucks
  - Mod 4: Synchronization (Pi program revisited)
  - Disc 3: Synchronization overhead and eliminating false sharing
  - Mod 5: Parallel Loops (making the Pi program simple)
  - Disc 4: Pi program wrap-up
- Unit 3: Working with OpenMP
  - Mod 6: Synchronize single masters and stuff
  - Mod 7: Data environment
  - Disc 5: Debugging OpenMP programs
  - Mod 8: Skills practice … linked lists and OpenMP
  - Disc 6: Different ways to traverse linked lists
- Unit 4: a few advanced OpenMP topics
  - Mod 8: Tasks (linked lists the easy way)
  - Disc 7: Understanding Tasks
  - Mod 8: The scary stuff … Memory model, atomics, and flush (pairwise synch).
  - Disc 8: The pitfalls of pairwise synchronization
  - Mod 9: Threadprivate Data and how to support libraries (Pi again)
  - Disc 9: Random number generators
- Unit 5: Recapitulation

Compiler notes:

- Linux and OS X with gcc:
  - gcc -fopenmp foo.c
  - export OMP_NUM_THREADS=4
  - ./a.out

for the Bash shell

Exercise 1, Part A: Hello world

Verify that your environment works

- Write a program that prints “hello world”.

```
int main()
{




     int ID = 0;

     printf(“ hello(%d) ”, ID);;
     printf(“ world(%d) \n”, ID);;

}
```

Exercise 1, Part B: Hello world

Verify that your OpenMP environment works

- Write a multithreaded program that prints “hello world”.

```
  #include <omp.h>
int main()
{
   #pragma omp parallel
   {
     int ID = 0;
     printf(“ hello(%d) ”, ID);;
     printf(“ world(%d) \n”, ID);;
   }
}
```