

The background is a dark blue gradient with a subtle pattern of white dots. On the left side, there are several concentric circles and a large circular scale with degree markings from 140 to 260. Some circles have arrows indicating a clockwise direction.

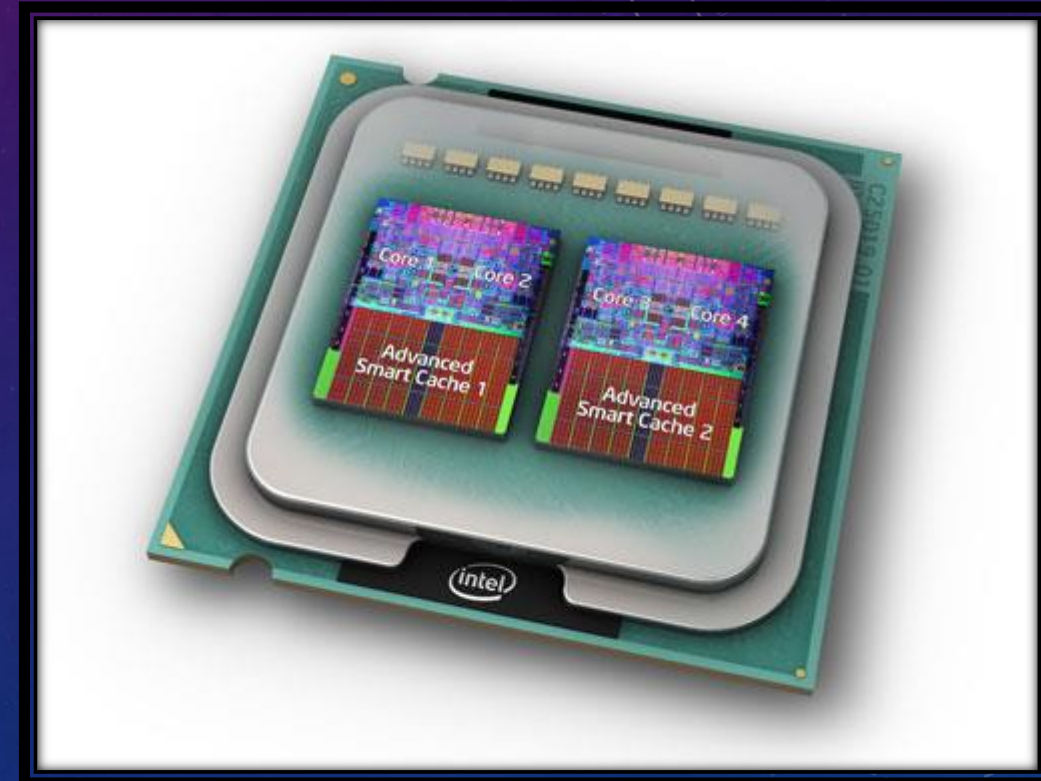
OPENMP

KISHAN KARUNARATNE

6/21/2013

WHY PARALLELIZE?

- CPU clocks not going up; becoming multiprocessor/multicore
- Need to harness power of CPUs
 - Traditional serial code = boring & slow



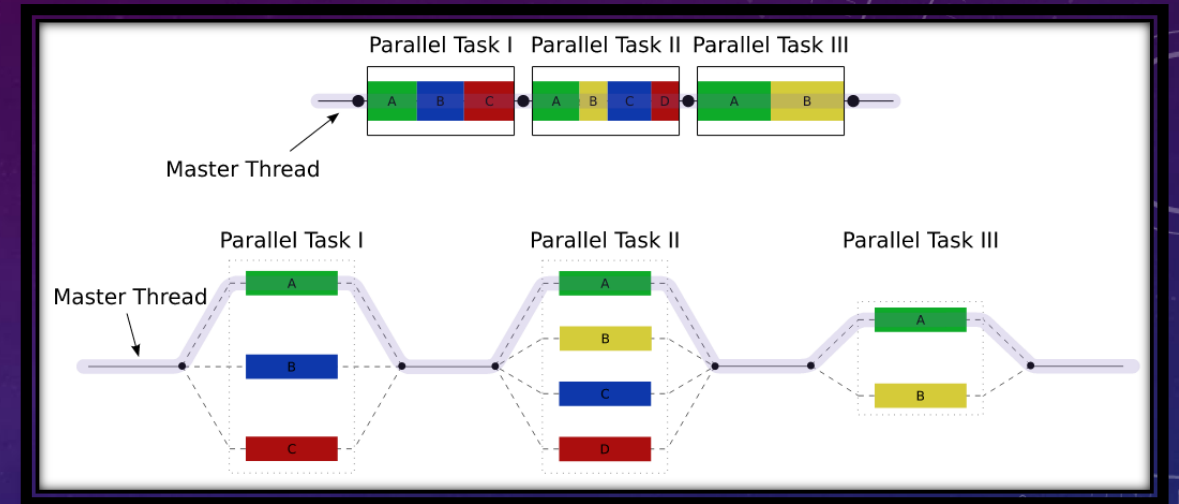
WHAT IS OPENMP?



- API for parallel programming in C and C++
 - Supported by GCC “-fopenmp”
- Supports most processor architectures, OS
- Simple, flexible interface for developing parallel apps

HOW DOES IT WORK?

- Implicit fork/join model
- Regions of code where instructions shared
 - Loops
 - Nested loops
- Use “#pragma omp parallel” directive
 - Other directives on Reference Card



OpenMP API 3.1 C/C++

Page 1

OpenMP 3.1 API C/C++ Syntax Quick Reference Card

OpenMP Application Program interface (API) is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer.

[n.n.n] refers to sections in the OpenMP API Specification available at www.openmp.org.

OpenMP supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures, including Unix platforms and Windows NT platforms.

A separate OpenMP reference card for Fortran is also available.

Directives

An OpenMP executable directive applies to the succeeding structured block or an OpenMP Construct. A structured block is a single statement or a compound statement with a single entry at the top and a single exit at the bottom.

Parallel [2.4]

The **parallel** construct forms a team of threads and starts parallel execution.

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

clause:

if(*scalar-expression*)
num_threads(*integer-expression*)
default(shared | none)
private(list)
firstprivate(list)
shared(list)
copyin(list)
reduction(operator: list)

Loop [2.5.1]

The **loop** construct specifies that the iterations of loops will be distributed among and executed by the encountering team of threads.

#pragma omp for [*clause* [, *clause*] ...]

for-loops

clause:

private(list)
firstprivate(list)

Single [2.5.3]

The **single** construct specifies that the associated structured block is executed by only one of the threads in the team (not necessarily the master thread), in the context of its implicit task.

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

clause:

private(list)
firstprivate(list)
copyprivate(list)
nowait

Parallel Loop [2.6.1]

The **parallel loop** construct is a shortcut for specifying a **parallel** construct containing one or more associated loops and no other statements.

#pragma omp parallel for [*clause* [, *clause*] ...]
for-loop

clause:

Any accepted by the **parallel** or **for** directives, except the **nowait** clause, with identical meanings and restrictions.

Simple Parallel Loop Example

The following example demonstrates how to parallelize a simple loop using the **parallel loop** construct.

Taskyield [2.7.2]

The **taskyield** construct specifies that the current task can be suspended in favor of execution of a different task.

#pragma omp taskyield

Master [2.8.1]

The **master** construct specifies a structured block that is executed by the master thread of the team. There is no implicit barrier either on entry to, or exit from, the **master** construct.

#pragma omp master
structured-block

Critical [2.8.2]

The **critical** construct restricts execution of the associated structured block to a single thread at a time.

#pragma omp critical [*name*]
structured-block

Barrier [2.8.3]

The **barrier** construct specifies an explicit barrier at the point at which the construct appears.

#pragma omp barrier

Taskwait [2.8.4]

The **taskwait** construct specifies a wait on the completion of child tasks of the current task.

#pragma omp taskwait

Atomic [2.8.5]

The **atomic** construct ensures that a specific storage