The Lecture Contains:

Previous Next

Open Multi-Processing

What is Parallelization?

Parallelization

Simultaneous use of more than one processor to complete some work is parallelization of the work.

- This work can be:
  - A collection of program statements
  - An algorithm
  - A part of program
  - The problem you are trying to solve

Parallel Overhead

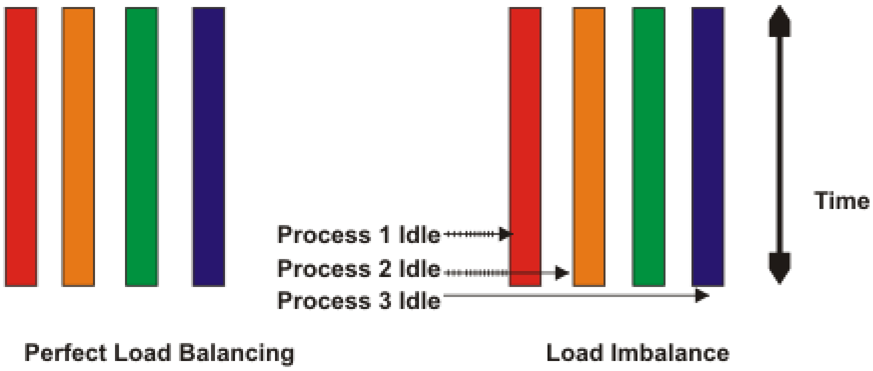
- Overhead is introduced during parallelization due to :
  - Creation of threads(fork())
  - Joining of threads (join())
  - Thread synchronization and communication e.g. critical
  - sections
  - False sharing
  - Overhead is introduced during parallelization due to :
- Creation of threads(fork())
  - Joining of threads (join())
  - Thread synchronization and communication e.g. critical
  - sections
  - False sharing
  - Overhead increases with number of threads
  - Efficient parallelization is minimizing this overheads

Previous Next

Load Balancing

Perfectly Load-Balanced Program

In a perfectly balanced parallel program no set of processors is idle while other set of processors is doing some computation



Amdahl's Law

- Assume that code has serial fraction
- Let T(1) be the execution time in one processor
- T(P) is execution time on P processors

$$T(P) = \gamma\, T(1) + \frac{(1-\gamma)T(1)}{P}$$

- Then speed up on P processors is given by

$$S(P) = \frac{T(1)}{T(P)} = \frac{1}{\gamma + \frac{1-\gamma}{\rho}}$$

About Data

- In a shared memory parallel program variables are either “shared” or “private”

“private” Variables

- Visible to one thread only
- Changes made to these variable are not visible to other threads
- Example : Local variables in a function that is executed in parallel

“shared” Variables

- Visible to all threads
- Changes made to these variable by one thread are visible to other threads
- Example : Global data

Module 10: Open Multi-Processing

Lecture 19: What is Parallelization?

What is Data Race ?

- When two or more different threads in a multithreaded shared memory model access the same memory location the program
  may produce unexpected results
- Data race occurs under following conditions:
  - There are two or more different threads accessing the same memory location concurrently
  - They don't host any locks
  - At least one access is write

A "for" Loop

"for" Loop

```
for(int i = 0 ; i < 8 ; i++)
a[i] = a[i] + b[i];
```

Execution in Parallel With 2 Threads

Thread 1	Thread 2
a[0] = a[0] + b[0]	a[4] = a[4] + b[4]
a[1] = a[1] + b[1]	a[5] = a[5] + b[5]
a[2] = a[2] + b[2]	a[6] = a[6] + b[6]
a[3] = a[3] + b[3]	a[7] = a[7] + b[7]

Overview to OpenMP

What is OpenMP ?

An API that may be used to explicitly direct multi-threaded, shared memory parallelism.

When to Use OpenMP For Parallelism ?

- A loop is not parallelized
  The data dependence analysis is not able to determine whether it is safe to parallelize or not
- The Granularity is not enough
  The compiler lacks information to parallelize at highest possible level

Previous Next

Why OpenMP ?

OpenMP is :

- Portable
  The API is specified for C/C++ and FORTRAN
  Supported in Most major platforms e.g. Unix and Windows
- Standardized
- Lean and Mean
- Easy in use

We should parallelize only when the overhead due to parallelization is less than the speed-up obtained.

Components of OpenMP

OpenMP consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior

Directives

- Parallel Regions
- Work Sharing
- Synchronization
- Data Sharing Attributes
- Orphaning

Environment Variables

- Number of Threads
- Scheduling Type
- Dynamic Thread Adjustment
- Nested Parallelism

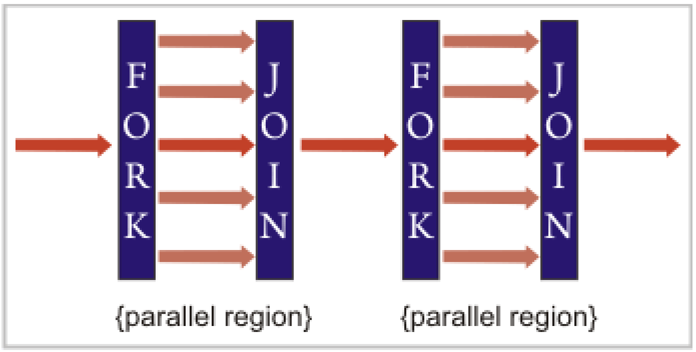
Runtime Environment

- Number of Threads
- Thread ID
- Dynamic Thread Adjustment
- Nested Parallelism
- Timers
- API for Locking

Previous Next

OpenMP Programming Model

- Shared Memory, Thread Based Parallelism
- Explicit Parallelism
- Fork Join Model



- Compiler Directive Based
- Nested Parallelism Support
- Dynamic Threads

OpenMP Directives

Fortran Directive Format

- Format Sentinel directive [clause....]
- Sentinel $OMP or C$OMP or $OMP
- Example $OMP PARALLEL DEFAULT(SHARED) PRIVATE(BETA,PI)
- General Rules
    - Comments can not appear on the same line as a directive
    - Several Fortran OpenMP directives come in pair and have the form shown below
      $OMP directive
      [Structured block of code]
      $OMP end directive

OpenMP Format

C/C++ Directive Format

- Format #pragma omp directive-name [clause....] newline
- Example #pragma omp parallel default(shared) private(beta,pi)
- General Rules
    - Case sensitive
    - Each directive applies to at most one succeeding segment, which must be a structured block

OpenMP Directives

Parallel Region Construct

A parallel region is a block of code executed by multiple threads simultaneously

```
#pragma omp parallel [clause[[,] clause] ...]
{
“this is executed in parallel”
} (implied barrier)
```

Clauses Supported

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

Example 1

A Multi-threaded “Hello World” Program

Example Code

```
#include “omp.h”
void main(){
#pragma omp parallel
{
int id = omp get thread num();
printf(“hello(%d)”,ID);
}
}
```

Previous Next

Example 1

A Multi-threaded "Hello World" Program

Example Code	Sample Output
#include "omp.h"	hello(0) hello(3) hello(1)
void main(){	hello(2)
#pragma omp parallel	hello(1) hello(2) hello(0)
{	hello(3)
int id = omp get thread num();	
printf("hello(%d)",ID);	
}	
}	

"IF" Clause

If an "if" clause is present it must evaluate to .TRUE. (Fortran) or non-zero (C/C++) in order to create a team of threads. Otherwise, the region is executed serially by master thread.

Example 2

A Multi-threaded "Hello World" Program With Clauses

```
#include "omp.h"
void main( ){
int x = 10;
#pragma omp parallel if(x > 10) num threads(4)
{
int id = omp get thread num();
printf("hello(%d)",ID);
}
}
```

- Num threads clause to request certain no of threads
- Omp get thread num() runtime function to return thread ID

Previous Next

OpenMP:Terminology and Behavior

How Does It Work ?

- When a thread reaches a parallel directive, it creates a team of threads and becomes the master of the team.
- The master thread always have ID 0 and it is the part of team
- There is an implied barrier at the end of parallel section.
- Thread adjustment (if enabled) is only done before entering a parallel region
- Parallel regions can be nested depends on implementation.
- An "if" clause can be used to guard the parallel region
- It is illegal to branch in or out of parallel region
- Only a single IF or NUM THREADS clause is permitted

Work Sharing Constructs

- A work sharing construct divides the execution of enclosed code region among the members of team
- They don't launch new threads
- Must be enclosed in a parallel region
- No implied barrier on entry; implied barrier on exit(unless nowait is specified)
- Must be encountered by all threads in team or none at all

#pragma omp for	#pragma omp sections	#pragma omp single
{	{	{
....	....	....
}	}	}
$OMP DO	$OMP SECTIONS	$OMP SINGLE
....	....	....
$OMP END DO	$OMP END SECTIONS	$OMP END SINGLE

The "omp for" Directive

- The iterations of loop are distributed over the members of the team.
- This assumes a parallel region has already been initiated, otherwise it executes in serial on a single processor.

Format

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

- There is and implied barrier at exit unless "nowait" clause is specified

Previous Next

The “omp for” Directive

Clauses Supported

- Schedule(type[,chunk])
- Private(list)
- Lastprivate(list)
- Collapse\
- Ordered
- Firstprivate(list)
- Shared(list)
- Reduction(operator:list)
- Nowait

Example 1

A Parallel For Loop Example

```
#pragma omp parallel
{
#pragma omp for
for(int i = 0; i < N; i++){
do some work( i );
}
}
```

- The variable i is made private to each thread by default you could do it explicitly by private(i) clause.

The “sections” Directive

- It specifies that the enclosed section(s) of codes are to be divided among the threads in the team

```
#pragma omp sections [ clause(s) ]
{
#pragma omp section
< codeblock1 >
#pragma omp section
< codeblock2 >
#pragma omp section
:
}
```

- Independent section directives are nested within a sections directive.Each section is executed once by a thread in the team.

Previous Next