Open MP Programming

Topics covered

1. Introduction to OpenMP
2. Programming Model
3. Contents of OpenMP
4. SMP Programming Errors

Introduction

OpenMP (OMP) is an API for writing portable Shared Memory Parallelism (SMP) application software based on multithreading

- A set of compiler directives and library routines for parallel application programmers
- Makes it easy to create multi-threaded (MT) programs in Fortran, C and C++
- Standardizes last 15 years of SMP practice
- OMP is not a library or a language

Programming Model

- Implements Fork-Join Parallelism
  - Initially takes a sequential program and put parallel directives into the program
  - Master Thread creates new threads as needed
  - When threads complete the statements in parallel section, all the threads terminate synchronously, leaving only one thread
- Parallelism is added incrementally i.e. the sequential program evolves into a parallel program
Lecture 7: March 4

How is OpenMP typically used?

OMP is used to parallelize loops
  • Find your most time consuming loops.
  • Split them up between threads

Sequential Program

void main()
{
    double Res[1000];
    for(int i=0;i<1000;i++) {
        do_huge_comp(Res[i]);
    }
}

Split-up the loop between multiple threads

Parallel Program

void main()
{
    double Res[1000];
    #pragma omp parallel for
    for(int i=0;i<1000;i++) {
        do_huge_comp(Res[i]);
    }
}

How do threads interact?
• OpenMP is a shared memory model.
  ➢ Threads communicate by sharing variables.

• Unintended sharing of data can lead to race conditions
  ➢ Race Condition: when the program’s outcome changes as the threads are scheduled differently.

• To control race conditions
  ➢ Use synchronization to protect data conflicts.

• Synchronization is expensive
  ➢ Change how data is stored to minimize the need for synchronization.

Syntax Details

• 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]…]

• Since the constructs are directives, an OpenMP program can be compiled by compilers that don’t support OpenMP.

Most OpenMP constructs apply to structured blocks i.e. a block of code with one point of entry at the top and one point of exit at the bottom.

OpenMP Contents

OpenMP’s constructs fall into 5 categories:
• Parallel Regions
• Worksharing
• Data Environment
• Synchronization
• Runtime functions/environment variables

OpenMP: Parallel Regions

• We can create threads in OMP with “omp parallel” pragma.
• The following example creates a 4 thread parallel region
  ➢ Each thread calls pooh(ID) for ID = 0 to 3 and redundantly executes the code within the structured block

double A[1000];
omp_set_num_threads(4);
#pragma omp parallel
{
  int ID = omp_thread_num();
  pooh(ID, A);
}
printf(“all done\n”);

• Notice that a single copy of A is shared between all threads.
• Threads wait for all threads to finish before proceeding to printf statement (i.e. a barrier)

**OMP: more details**
• Dynamic mode (the default mode) – number of threads used in a parallel region can vary from one parallel region to another.
• Static mode – number of threads is fixed and controlled by programmer.
• OpenMP allows nesting of parallel regions

OpenMP: Work-Sharing Constructs

• Work-sharing constructs make the work easier by dividing a task among the threads so that each thread executes its allocated part of the code
• Data Parallelism – each thread works on subsets of same data
• Task Parallelism – each processor executes a different thread on the same or different data
• Various constructs used are for, section, schedule, parallel and work sharing construct combined

Scope of OpenMP constructs
OpenMP constructs can span multiple source files

Static or lexical extent of parallel region
poo.f

C$OMP PARALLEL
  call whoami
C$OMP END PARALLEL

• Split threads from poo.f. Threads are mutually exclusive

Dynamic extent of parallel region includes static extent
Bar.f

  subroutine whoami
  external omp_get_thread_num
  integer iam, omp_get_thread_num
  iam = omp_get_thread_num()
OpenMP: Data Environment

Default storage attributes

- **Shared Memory** programming model - most variables are shared by default
- **Global variables** are SHARED among threads
  - Fortran: COMMON blocks, SAVE variables, MODULE variables
  - C: File scope variables, static
- But not everything is shared - Stack variables called from parallel region and Automatic variables are Private
  - Shared variables are passed on each time they are read or write
  - Private variables have a private stack of their own.
  - Threads do not make a local copy of the variable otherwise the change would not be reflected everywhere.

OpenMP: Synchronization

OpenMP has the following constructs to support synchronization:

- Critical Section - Only one thread at a time can enter a critical section
- Atomic - is a special case of a critical section and applies only to the update of memory location.
- Barrier – implicit synchronization at the end of construct or parallel region. Each thread waits until all threads arrive
- Flush – also implicitly applied at the end of block. It’s a sequence point where a thread tries to create a consistent view of memory
- Master - a structured block that is only executed by the master thread. The other threads just skip it (no implied barriers or flushes).
- Single - a block of code that is executed by only one thread.
- Ordered

SMP Programming Errors

- Errors coming from unanticipated shared resource conflicts

2 major SMP errors
• Race Conditions
  ➢ The outcome of a program depends on the detailed timing of the threads in the team.
  ➢ Race conditions occur when shared variables are simultaneously read or write.

• Deadlock
  ➢ Threads lock up waiting on a locked resource that will never become free

Reference Material: www.openmp.org