Lecture 21 – Shared-Memory Parallel Computing

- **Now let's start to discuss how we can use shared-memory systems to perform parallel computing**
- **Remember that in shared-memory systems, all of the processors see all of the data**

Motivation

- **Popularity of shared memory systems is increasing:**
  - Distributed Shared-Memory (DSM) computers (SGI Altix, SUN Ultra 10000,HP Exemplar) have replaced vector systems in many supercomputing centers
- **Compiler directed parallelism is attracting attention:**
  - Single version of sequential and parallel code
  - Industrial standard is now available: OpenMP

Motivation

- **New generation of parallel machines with multiple CPUs per node**
  - IBM SP3 (older technology):
    - each node is a 16-way shared memory machine
    - there is a single bus to the high performance network per node
  - IBM BlueGene/L (later technology):
    - Each node is a 2-way shared memory machine
    - 1024 nodes per rack, 16-128 I/O nodes per rack, 3D Torus interconnect 350 Mb/s bandwidth, 1.5 μs latency
  - Quad-, Octo-, and now higher Core Chip-Sets (eg. Mic, latest technology)
- **Mixed programming model:**
  - OpenMP in each node
  - MPI across nodes

Programming Models

- **Shared memory options:**
  - Automatic parallelization (some compilers)
  - Pthreads (POSIX threads)
  - Compiler directives: OpenMP
- **Message passing options:**
  - MPI: message passing interface
  - PVM: parallel virtual machine
  - HPF: high performance Fortran

OpenMP

- **OpenMP is an API for writing multithreaded applications:**
  - A set of compiler directives and library routines for parallel application programmers
  - Makes it easy to create multithreaded programs in Fortran, C and C++

OpenMP Supporters

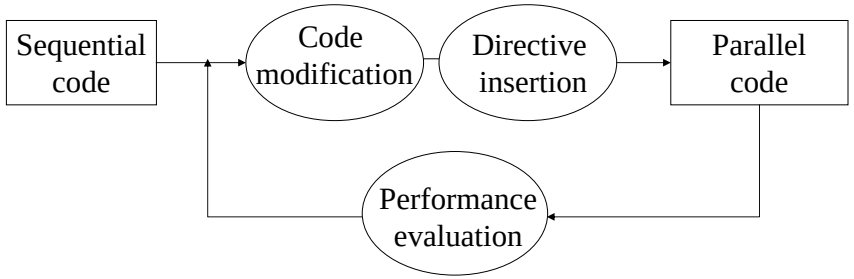
- **Hardware vendors:**
  - Compaq/HP, IBM, Intel, SGI, SUN,…
- **Software vendors**
  - KAI (now Intel), PGI, Absoft, PSR,…

http://openmp.org/wp

OpenMP

- **Fine grained parallelism (at loop level)**
- **Coarse grained parallelism**
- **Compiler directives, library and environment variables *extend* base language**
- **NOT automatic parallelization**
- **Since the constructs are directives, an OpenMP program can be compiled by compilers that do not support OpenMP**

Incremental Parallelization



Work can be done incrementally!!!!!!!

Serial Code

```
      program compute_pi
      integer n, i
      double precision w, x, sum, pi, f, a
! function to integrate
      f(a) = 4.d0 / (1.d0 + a*a)
      print *, 'Enter number of intervals: '
      read *,n
! calculate the interval size
      w = 1.0d0/n
      sum = 0.0d0
      do i = 1, n
            x = w * (i - 0.5d0)
            sum = sum + f(x)
      end do
      pi = w * sum
      print *, 'computed pi = ', pi
      stop
      end
```


MPI Code

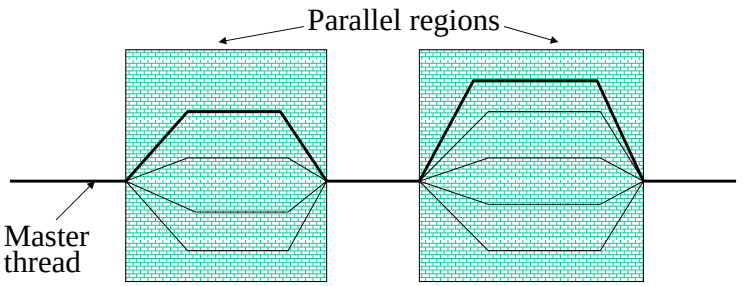
```
      program compute_pi
      include 'mpif.h'
      double precision mypi, pi, w, sum, x, f, a
      integer n, myid, numprocs, i, rc
! function to integrate
      f(a) = 4.d0 / (1.d0 + a*a)
      call MPI_INIT( ierr )
      call MPI_COMM_RANK(MPI_COMM_WORLD,
     &                   myid, ierr )
      call MPI_COMM_SIZE(MPI_COMM_WORLD,
     &                   numprocs, ierr )
      if ( myid .eq. 0 ) then
         print *, 'Enter number of intervals: '
         read *, n
      endif
      call MPI_BCAST(n,1,MPI_INTEGER,0,
     &               MPI_COMM_WORLD,ierr)
! calculate the interval size
      w = 1.0d0/n
      sum = 0.0d0
      do i = myid+1, n, numprocs
         x = w * (i - 0.5d0)
         sum = sum + f(x)
      enddo
      mypi = w * sum
! collect all the partial sums
      call MPI_REDUCE(mypi,pi,1,
     &                MPI_DOUBLE_PRECISION,
     &                MPI_SUM,0,
     &                MPI_COMM_WORLD,ierr)
! node 0 prints the answer
      if (myid .eq. 0) then
         print *, 'computed pi = ', pi
      endif
      call MPI_FINALIZE(rc)
      stop
      end
```


OpenMP Code

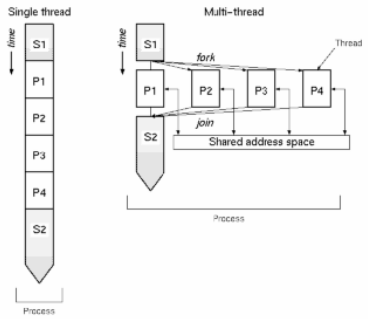
```
      program compute_pi
      integer n, i
      double precision w, x, sum, pi, f, a
! function to integrate
      f(a) = 4.d0 / (1.d0 + a*a)
      print *, 'Enter number of intervals: '
      read *,n
! calculate the interval size
      w = 1.0d0/n
      sum = 0.0d0
!$OMP PARALLEL DO PRIVATE(x), SHARED(w,n)
!$OMP& REDUCTION(+: sum)
      do i = 1, n
         x = w * (i - 0.5d0)
         sum = sum + f(x)
      end do
!$OMP END  PARALLEL DO
      pi = w * sum
      print *, 'computed pi = ', pi
      stop
      end
```


Execution Model

- **Master thread spawns a team of threads as needed**
  - Concept of threads also used in GPU parallel computing
- **Parallelism is added incrementally**



Execution Model



- **OpenMP programs start as single thread**
- **In parallel regions, additional threads are created**
- **Master thread is part of the team**
- **Outside parallel regions, additional threads go away (or sleep)**

Directives or pragmas

- **Fortran (fixed format)**

  !$omp …

  c$omp …

  *$omp …

  If the character in column 6 is different from a space or a 0 it is a continuation line
- **Fortran (free form)**
  - A line that begins with !$omp is an OpenMP directive
  - A directive that needs to be continued on the next line:

    !$OMP ……. &
- **C:**

  #pragma omp

Conditional Compilation

- **The selective disabling of OpenMP construct applies only to directives.**
- **Application may contain statements that are specific to OpenMP but not intended for OpenMP**

Fortran :

- A line beginning with a *sentinel (!$,c$,*$* in fixed format, !$ in free format) is ignored if not in OpenMP

  !$ iam=omp_get_num_thread()

C:

- Preprocessor macro _OPENMP

  #pragma omp

Programming Model

- **OpenMP is a shared memory model**
  - Threads communicate by sharing variables
- **Unintended sharing of data can lead to race conditions**
  - Race conditions: the program's outcome changes as the threads are scheduled differently

Example in C

Sequential code

```
void main()
{
  double a[100000];
  for (int i=0,i<100000,i++){
     do_something(a[i]);
                    }
 }
```

Parallel code

```
void main()
{
  double a[100000];
#pragma omp parallel for
  for (int i=0,i<100000,i++){
     do_something(a[i]);
                    }
 }
```


Example in Fortran

Sequential code

```
real*8 a(100000)
do i = 0,100000
    do_something(a)
enddo
```

Parallel code

```
real*8 a(100000)
!#omp parallel do
do i = 0,100000
     do_something(a)
 enddo
```


OpenMP Constructs

- **OpenMP's constructs (directives) fall into 5 categories:**
  - Parallel regions
  - Worksharing
  - Synchronization
  - Data environment
  - Runtime functions/environment variables
- **We will begin to discuss these constructs in the next lecture**