Workerbee.c

• Starts MPI

• Splits the processors into two groups/communicators 0-(N-2) and (N-1)

• Processor (N-1) waits for “ready” from other processors, then sends work

• Rest of processors loop
  • send requests for work
  • do work
  • send results

The Full Source for this program is in the slides but we will skip over most of it and look at just the worker and manager subroutines
module mympi
  include "mpif.h"
end module

! global variables
module globals
  integer numnodes, myid, mpi_err
  integer, parameter :: mpi_root = 0
end module

! end of global variables
subroutine init_it()
    use globals
    use mympi
    call MPI_INIT( ierr )
    call MPI_COMM_RANK( MPI_COMM_WORLD, myid, mpi_err )
    call MPI_COMM_SIZE( MPI_COMM_WORLD, numnodes, mpi_err )
end subroutine

program themain
    use mympi
    use globals
    implicit none
    integer WORKER_WORLD,worker_size
    integer group
    integer num_used,new_id
    integer mannum
    call init_it()
    write(*,*)"hello from ",myid,"of",numnodes
manum=numnodes-1
if(myid == manum) then
  group=0
else
  group=1
endif

call mpi_comm_split(mpi_comm_world, group, myid, WORKER_WORLD, mpi_err)
call mpi_comm_rank( WORKER_WORLD, new_id, mpi_err )
call mpi_comm_size( WORKER_WORLD, worker_size, mpi_err )
write(*,*)myid,new_id,worker_size

<table>
<thead>
<tr>
<th>MPI_COMM_World</th>
<th>WORKER_WORLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Create a communicator that contains everyone but the last process

<table>
<thead>
<tr>
<th>MPI_COMM_World</th>
<th>WORKER_WORLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
if(worker_size .eq. 1) then
  ! if not part of the new group do management.
  num_used=numnodes-1
  call manager(num_used)
  write(*,*), "manager finished"
  call MPI_BARRIER(MPI_COMM_WORLD,mpi_err)
  call MPI_Finalize(mpi_err)
else
  Manager is not part of "WORKER_WORLD" so she manages
Workers work
Note: we are passing in the
   ID for the manager, num_used

! part of the new group do work.
   call worker(WORKER_WORLD, mannum)
   write(*,*)"worker finished"
   call MPI_BARRIER(MPI_COMM_WORLD, mpi_err)
   call MPI_Finalize(mpi_err)
endif
end program
subroutine worker( THE_COMM_WORLD, managerid)
  use mympi
  use globals
  real x
  integer status(MPI_STATUS_SIZE)
  x=0.0
  do while(x > -1.0)
    ! send message says I am ready for data
    call MPI_SEND(x,1,MPI_REAL,managerid,1234,MPI_COMM_WORLD,mpi_err)
    ! get a message from the manager
    call MPI_RECV(x,1,MPI_REAL,managerid,2345,MPI_COMM_WORLD,status,mpi_err)
    ! process data
    x=x*2.0
    call sleep(myid+1)
  enddo
end subroutine

Here the message to do a task is just a real number and the task is to multiply it by 2. The message could be a text string containing a command and the task could be to run that command as a new process.
subroutine manager(num_used)
  use mympi
  use globals
  integer igot,isent,gotfrom,sendto,i
  integer, parameter::TODO=25
  real inputs(0:TODO-1)
  real x
  integer status(MPI_STATUS_SIZE)
  logical flag
  igot=0
  isent=0
  do i=0,TODO-1
    inputs(i)=i+1
  enddo
  do while(igot < TODO)
    ! wait for a request for work
    call MPI_IPROBE(MPI_ANY_SOURCE,MPI_ANY_TAG,MPI_COMM_WORLD,flag,status,mpi_err)
    if(flag)then
      ! where is it coming from
      gotfrom=status(MPI_SOURCE)
      sendto=gotfrom
      call MPI_Recv(x,1,MPI_REAL,gotfrom,1234,MPI_COMM_WORLD,status,mpi_err)
      write(*,*)"worker ",gotfrom," sent ",x
      if(x > 0.0)igot=igot+1
      if(isent < TODO)then
        ! send real data
        x=inputs(isent)
        call MPI_Send(x,1, MPI_REAL,sendto,2345,MPI_COMM_WORLD,mpi_err)
        isent=isent+1
      endif
    endif
  enddo
  ! tell everyone to quit
  do i=0,num_used-1
    x=-1000.0
    call MPI_Send(x,1, MPI_REAL,i,2345,MPI_COMM_WORLD,mpi_err)
  end do
end subroutine

• Manager waits for ready message
• Sends work
• Tells everyone to quit when work is finished
OpenMP

- Each core is running a different thread
- Threads are numbered 0 to (N-1)
- Threads share memory
- Messages between threads are passed via the shared memory
- Number of threads is limited to cores on a node
- Parallelism is suggested to the compiler via directives
Four Independent Matrix Inversions

```c
#pragma omp parallel sections
{
    #pragma omp section
    {
        system_clock(&t1_start);
        over(m1,n);
        over(m1,n);
        system_clock(&t1_end);
        e1=mcheck(m1,n,1);
        t1_start=t1_start-t0_start;
        t1_end=t1_end-t0_start;
    }
    #pragma omp section
    {
        system_clock(&t2_start);
        over(m2,n);
        over(m2,n);
        system_clock(&t2_end);
        e2=mcheck(m2,n,2);
        t2_start=t2_start-t0_start;
        t2_end=t2_end-t0_start;
    }
    #pragma omp section
    {
        system_clock(&t3_start);
        over(m3,n);
        over(m3,n);
        system_clock(&t3_end);
        e3=mcheck(m3,n,3);
        t3_start=t3_start-t0_start;
        t3_end=t3_end-t0_start;
    }
    #pragma omp section
    {
        system_clock(&t4_start);
        over(m4,n);
        over(m4,n);
        system_clock(&t4_end);
        e4=mcheck(m4,n,4);
        t4_start=t4_start-t0_start;
        t4_end=t4_end-t0_start;
    }
}

• The “over” routines could be any independent operations
• Became parallel by adding 5 directive lines
Four Independent Matrix Inversions

-bash-3.2$ export OMP_NUM_THREADS=2
-bash-3.2$ ./invertc
section 1 start time= 0.00095582 end time= 0.32442 error= 6.00659e-06
section 2 start time= 0.32478 end time= 0.64667 error= 0.000453301
section 3 start time= 0.64702 end time= 0.96885 error= 8.78033e-05
section 4 start time= 0.9692 end time= 1.2911 error= 0.000873184

-bash-3.2$ export OMP_NUM_THREADS=1
-bash-3.2$ ./invertc
section 1 start time= 0.0013621 end time= 0.36006 error= 8.78033e-05
section 2 start time= 0.36042 end time= 0.71921 error= 0.000873184

-bash-3.2$ export OMP_NUM_THREADS=4
-bash-3.2$ ./invertc
section 1 start time= 0.001534 end time= 0.32544 error= 6.00659e-06
section 2 start time= 0.001538 end time= 0.32929 error= 0.000453301
section 3 start time= 0.002799 end time= 0.32627 error= 8.78033e-05
section 4 start time= 0.002799 end time= 0.3263 error= 0.000873184
-bash-3.2$
N Independent Matrix Inversions

- OpenMP distributes do (for) loop iterations across the cores
- Here we call the matrix inversion routine DGESV nrays times
- Each inversion works on different data stored in tarf(:,:,i)
- Each thread does nrays/ncores calls and we get near linear speedup