ASSIGNMENT 5 SOLUTIONS

HPC1 Fall 2012

Due Date: ***Tuesday, November 20***

(please submit your report electronically by email to the instructor, in one PDF file named *hw5-yourUBITname.pdf*)

Problem 1: Start with your code from Assignment 4 that used Jacobi iteration to solve Laplace's equation on the unit square.

a. Now write a parallel solver using MPI. Note there is some helpful discussion of how to break up the grid (domain decomposition) in an old MPICH tutorial at

`http://www-unix.mcs.anl.gov/mpi/tutorial/mpiexmpl/contents.html`

(I would recommend that you use a similar simple 1D decomposition)

b. Examine (and plot) the performance of your message-passing solver in terms of parallel speedup and efficiency. You should target at least 64 MPI tasks to see representative performance (128 would be better, and use a large enough mesh to justify a large process count)

c. [Optional - extra credit] Now combine both approaches by utilizing **OpenMP** directives in addition to your **MPI** solver. Obtain maximum performance and compare the parallel speedup with that of part **b**.

Solution:

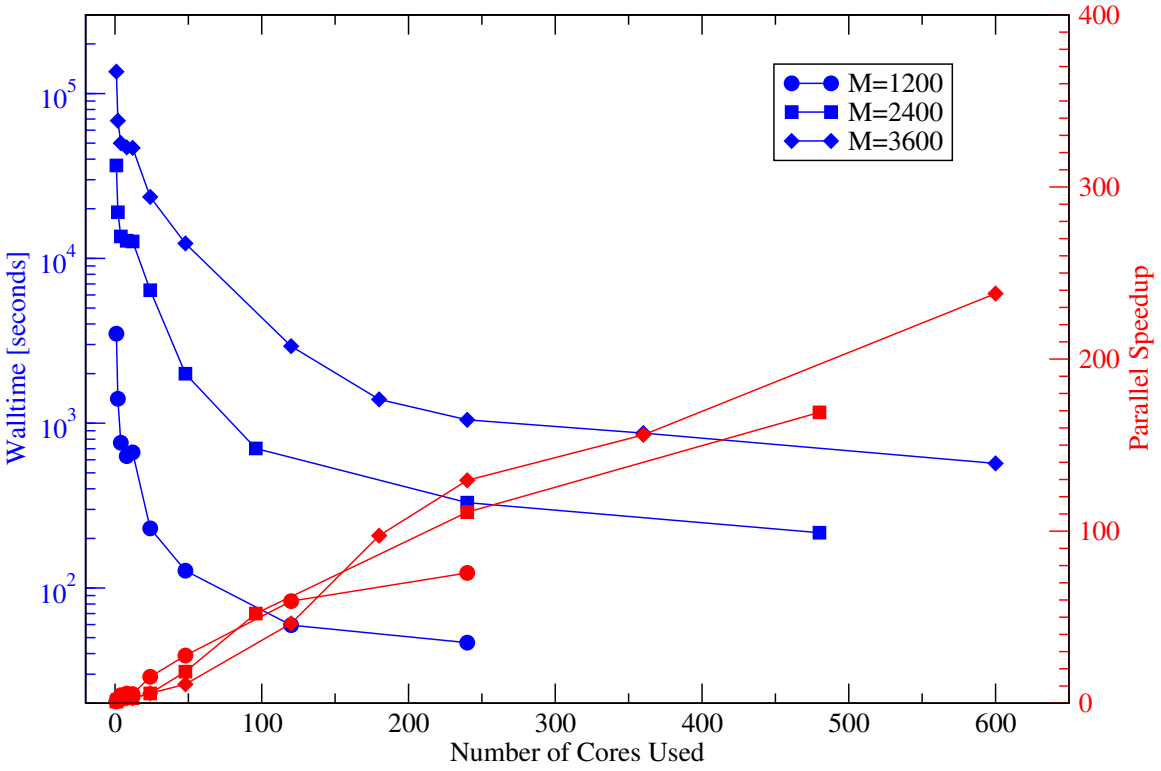

Figure 1: Parallel performance of the slab decomposed Jacobi solver on various grid sizes, M, with an L2-norm convergence criterion of 10^{-4}. All runs on the 12-core nodes using ppn=12.

The results of running the resulting MPI code are shown in Figure 1 for various grid sizes (all were run on the 12-core nodes, filling them using ppn=12). Note the "shoulder" in the plots for each grid size as we increase the size of the grid, where the performance suddenly picks up again after falling off? That has the same source that we saw in the last assignment - contention for main memory from too many cores. As we increase the process count the amount stored on each process is decreasing thanks to the MPI domain decomposition, the way that the speedup picks up again occurs when the domains start to fit into the L3 cache on the processors that we are using. This is shown in Figure 2, where you can more easily pick out the drop in parallel efficiency and then the increase as more cores are utilized. The super-linear spike in efficiency for M=1200 is also due to

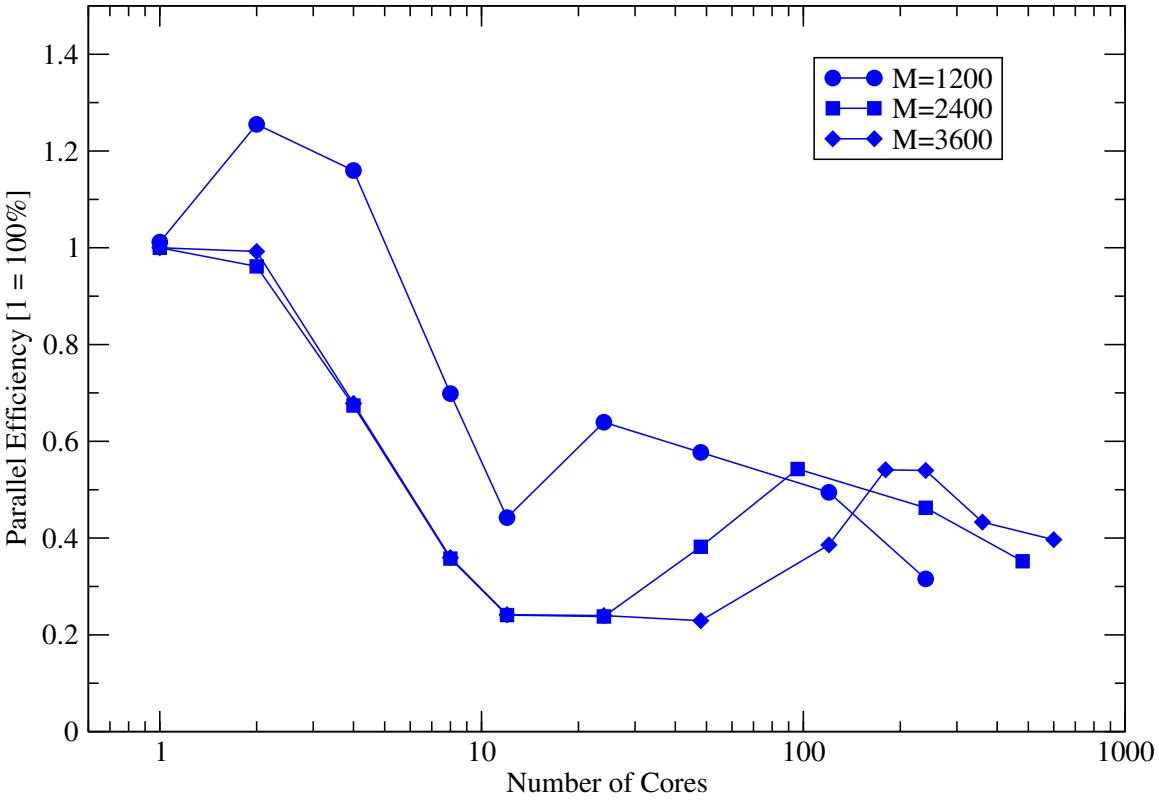

Figure 2: Parallel efficiency of the slab decomposed Jacobi solver on various grid sizes, M, with an L2-norm convergence criterion of 10^{-4}. All runs on the 12-core nodes using ppn=12.

the caching effect, as soon as the domain is split across two 12MB L3 caches the matrices will fit into the cache, and main memory need not be used to store them. That trend eventually suffers as well, though, as the cache itself suffers contention from more than two processes simultaneously trying to hit the data.

It is fairly easy to add the OpenMP directives used in the previous homework to the MPI version, but it is much less easy to get the resulting hybrid code to run efficiently. When using Intel MPI with hybrid MPI-OpenMP the environment variables `I_MPI_PIN_DOMAIN` (core placement/affinity of the MPI tasks) and `KMP_AFFINITY` (core affinity for the OpenMP threads) are essential. I found about a 10% overall improvement in the largest grid size when using 6 MPI tasks per node plus 2 OpenMP threads (on the 12-core nodes), just slightly outperforming the 2 MPI tasks plus 6 OpenMP threads runs (by a few percentage points). The real advantage of the hybrid mode of calculation is for the very largest runs, when the MPI collective (`MPI_Allreduce`) becomes very expensive.

```
MODULE paraMOD
  implicit none
  integer,parameter :: dp=selected_real_kind(2*PRECISION(1.0))
  real(kind=dp),parameter :: PI=3.1415926535897932

CONTAINS
  subroutine initBC(sol,m,ncols,myid,nprocs)
    !
    ! sol = current solution
    ! m = # rows in column-wise decomposition
    ! ncols = number of columns in column-wise decomposition
    ! myid = rank in communicator
    ! nprocs = # procs in communicator
    !
    ! BC: 0<= x,y <=1
    !   y=0: sol = sin(\pi*x)
    !   y=1: sol = exp(-\pi)*sin(\pi*x)
    !
    implicit none
    integer :: m,ncols,myid,nprocs
    real(kind=dp) :: sol(0:m+1,0:ncols+1)

    integer :: i
    real(kind=dp) :: x

    sol=0.0_dp
    do i=0,m+1
       x=dble(i)/dble(m+1)
       if (myid.eq.0) sol(i,0)=sin(PI*x)
       if (myid.eq.nprocs-1) sol(i,ncols+1)=sin(PI*x)*exp(-PI)
    end do
    return
  end subroutine initBC

  subroutine xchange(sol,m,ncols,myid,left,right)
    USE MPI
    implicit none

    integer :: m,ncols,myid,left,right
    real(kind=dp) :: sol(0:m+1,0:ncols+1)

    integer :: ierr,status(MPI_STATUS_SIZE)
    !
    ! use sendrecv combination call as a shortcut to exchange boundary data
    !
    ! ncols column goes right to column 0
    ! column 1 entry goes left to column (ncols+1)
    !
    call MPI_SENDRECV(sol(1,ncols),  m,MPI_DOUBLE_PRECISION,right,0, &
         &            sol(1,0),      m,MPI_DOUBLE_PRECISION,left,  0,MPI_COMM_WORLD,status,ierr)
    call MPI_SENDRECV(sol(1,1),      m,MPI_DOUBLE_PRECISION,left,  1, &
         &            sol(1,ncols+1),m,MPI_DOUBLE_PRECISION,right,1,MPI_COMM_WORLD,status,ierr)

    return
  end subroutine xchange

end MODULE paraMOD

program laplace_mpi
  USE paraMOD
  USE MPI
!$ USE omp_lib
  implicit none

  integer,parameter :: MAXITER=15000000,outiter=10000
  integer :: i,j,k,m,iter,ctick1,ctick2,ctickrate,ctickmax,nthreads
  real(kind=dp) :: delta,exacterr,tolerance
  real(kind=dp),allocatable :: sol(:,:),partsol(:,:),fullsol(:,:),rbuff(:,:),exactsol(:,:)

  integer :: ierr,myid,nprocs,ncols,left,right,status(MPI_STATUS_SIZE)
  real(kind=dp) :: t0,t1,fops
  real(kind=dp) :: all_delta  ! global delta

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)

  if(myid.eq.0) then
     print*,'Enter grid extent: (0:m+1)'
     read(*,*) m
     print*,'Using grid of size: ',m,' 0,m+1 used for BCs.'
     print*,'Enter tolerance for L2-norm:'
     read(*,*) tolerance
!$OMP PARALLEL
!$    nthreads=omp_get_num_threads()
!$    print*,'MPI task: ',myid,' using ',nthreads,' OpenMP threads/MPI task.'
!$OMP END PARALLEL
  endif
  call MPI_BCAST(m,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
  call MPI_BCAST(tolerance,1,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)

  ncols = m/nprocs
  if (myid.eq.0) then
     print*, 'Decomposing grid ',m,' into slabs of ',ncols
  endif
  ALLOCATE(sol(0:m+1,0:ncols+1),partsol(m,ncols))
  call initBC(sol,m,ncols,myid,nprocs) ! Apply Dirichlet boundary conditions
  !
  ! process id to which we need to send/recv updated boundary info
  !  (MPI_PROC_NULL means no shared boundary)
  !
```

```
102     if (myid.eq.0) then
103        left = MPI_PROC_NULL
104        right = myid+1
105     else if (myid.eq.nprocs-1) then
106        left = myid-1
107        right = MPI_PROC_NULL
108     else
109        left = myid-1
110        right = myid+1
111     endif
112     if(nprocs.eq.1) right = MPI_PROC_NULL
113     !
114
115     t0 = MPI_Wtime()
116     iter=0
117     all_delta=1.d6
118     do while (all_delta > tolerance)
119        iter = iter+1
120        ! update interior points based on "+" points
121        delta=0.0_dp
122  !$OMP PARALLEL PRIVATE(i,j) REDUCTION(+:delta)
123  !$OMP DO
124        do j=1,ncols
125           do i=1,m
126              partsol(i,j) = 0.25_dp*( sol(i+1,j)+sol(i-1,j)+sol(i,j+1)+sol(i,j-1) )
127           end do
128        end do
129  !$OMP END DO
130  !$OMP DO
131        do j=1,ncols
132           do i=1,m
133              sol(i,j) = partsol(i,j)
134              delta = delta + (partsol(i,j)-sol(i,j))*(partsol(i,j)-sol(i,j))
135           end do
136        end do
137  !$OMP END DO
138  !$OMP END PARALLEL
139        call MPI_ALLREDUCE(delta,all_delta,1,MPI_DOUBLE_PRECISION,MPI_SUM,MPI_COMM_WORLD,ierr)
140        all_delta=sqrt(all_delta)
141        call xchange(sol,m,ncols,myid,left,right) ! exchange boundary point data
142        if(MOD(iter,outiter).eq.0.and.myid.eq.0) write(*,'("␣iter␣=␣",i8,"␣delta␣=␣",e14.6)') iter,all_delta
143     end do
144     t1=MPI_Wtime()
145     fops = 8.0*m*m
146     fops = fops*iter
147     if (myid.eq.0) then
148        !CALL SYSTEM_CLOCK(ctick2,ctickrate,ctickmax)
149        write(*,'("Total␣Runtime[s],␣(under)estimated␣Mflop/s:␣",2f12.2)') t1-t0,fops/(1.e6*(t1-t0))
150     endif
151     !
152     ! output the full solution - have everyone send their columns back to 0
153     !
154     ALLOCATE(rbuff(0:m+1,0:ncols+1))
155     if(myid.ne.0) then
156        CALL MPI_SEND(sol,(m+2)*(ncols+2),MPI_DOUBLE_PRECISION,0,0,MPI_COMM_WORLD,ierr)
157     else
158        ALLOCATE(fullsol(0:m+1,0:m+1+MOD(m,nprocs)))
159        fullsol(0:m+1,0:ncols+1)=sol
160        do i=1,nprocs-1
161           CALL MPI_RECV(rbuff,(m+2)*(ncols+2),MPI_DOUBLE_PRECISION,i,0,MPI_COMM_WORLD,status,ierr)
162           do j=0,m+1
163              do k=0,ncols+1
164                 fullsol(j,k+i*ncols)=rbuff(j,k)
165              end do
166           end do
167        end do
168     end if
169     if (myid.eq.0) then ! output file for plotting
170        ALLOCATE(exactsol(0:m+1,0:m+1+MOD(m,nprocs)))
171        exacterr = 0.0_dp
172        do i=0,m+1
173           do j=0,m+1
174              exactsol(i,j) = SIN(PI*i/(m+1.0_dp))*EXP(-PI*j/(m+1.0_dp))
175              exacterr = MAX( exacterr, ABS(exactsol(i,j)-fullsol(i,j)) )
176  !             write(*,'("i,j,exact,approx,diff =",2i4,3e14.6)') i,j,exactsol(i,j),fullsol(i,j), &
177  !    &                 ABS(exactsol(i,j)-fullsol(i,j))
178           end do
179        end do
180        open(unit=10,file="laplace_mpi.dat",status='unknown',form='unformatted')
181        print*,'␣Max␣value␣in␣sol:␣',MAXVAL(fullsol)
182        print*,'␣Min␣value␣in␣sol:␣',MINVAL(fullsol)
183        print*,'␣Maximum␣deviation␣from␣exact␣solution:␣',exacterr
184        write(10) ((sngl(fullsol(i,j)),i=0,m+1),j=0,m+1)
185        close(10)
186        DEALLOCATE(fullsol,exactsol)
187        print*,'␣Number␣of␣non-bc␣grid␣points,␣iterations:',m,iter
188     endif
189     DEALLOCATE(sol,partsol,rbuff)
190     call MPI_FINALIZE(ierr)
191  end program laplace_mpi
```