
Introduction to
OpenMP exercises

Getting setup on ccb

First start an “interactive batch job” i.e. get a single node for your exclusive use

```
qsub -q np -I -l EC_nodes=1
```

the first -I is a uppercase I, the second one is a lowercase L
With this, once you have a session, you can work interactively:
edit, compiler, run aprun, etc ..

To get OpenMP exercise files

```
scp -r ecgate:/home/ectrain/trx/George/OpenMP_Course $PERM
```

Class Exercise 1
Experiment with SCHEDULE clause

```
subroutine work(k,a,b,c)
real*8 a(k),b(k),c(k)
!$OMP PARALLEL DO SCHEDULE(STATIC) PRIVATE(I)
do i=1,k
  c(i)=a(i)*exp(b(i))
enddo
!$OMP END PARALLEL DO
return
end
```

Use file omptest7.F90
Also look in file Notes

Try …
(STATIC)
(DYNAMIC)
(DYNAMIC,100)
(DYNAMIC,1000)
(GUIDED)
Which is the best for this loop?

Class Exercise 2

```
subroutine work(k,A,UNIQUE)
integer*8 k,A(k),unique
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Calculate how many integers in array A are             !
! unique and return via integer arg UNIQUE.              !
! Integers in A are between 1 and k.                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
return
end
```

Use file unique.F90

What do I mean by ‘unique’ (different?)

12	1
20	2
16	3
6	4
10	5
1	6
6	7
7	8
14	9
8	10
3	11
17	12
12	13
2	14
6	15
13	16
2	17
20	18
20	19
14	20

unique= 13

Class Exercise 2

```
subroutine work(k,a,unique)
integer*8 k,a(k),b(k),unique
b(:)=0
!$OMP PARALLEL DO PRIVATE(I)
do i=1,k
 b(a(i))=1
enddo
!$OMP END PARALLEL DO
unique=sum(b)
return
end
```

Class Exercise 3 (FiniteDiff)

This exercise involves parallelising a simple code, which contains a number of areas of functionality which you are likely to come across when you parallelise many real codes:

- Reading and writing of data
- Performing finite difference calculation
- "Global" sum of data

Instructions are in file README

FiniteDiff (model_driver)

```
SUBROUTINE Model_Driver (npoints)
IMPLICIT NONE
! Arguments
INTEGER, INTENT(IN) :: npoints
! Local variables
REAL :: DataArray(npoints)        ! Data to process
REAL :: Sum,Min,Max               ! Stats for DataArray
INTEGER :: Nstep                  ! Timestep
CALL Read_Data (npoints,DataArray)
!DO Nstep=1,100     ! Start with this case
DO Nstep=1,100000   ! Then move on to this
  CALL Finite_Difference(npoints,DataArray)
  CALL Sum_Data (npoints,DataArray,Sum,Min,Max)
ENDDO
WRITE(6,*) 'npoints=',npoints,' Sum=',Sum,`,&
  &Min=',Min,' Max=',Max
CALL Write_Data (npoints,DataArray)
RETURN
END
```

FiniteDiff (Finite_Difference solution)

```
SUBROUTINE Finite_Difference(npoints,DataArray)
! Performs a simple 1D finite difference with
! periodic boundary conditions
IMPLICIT NONE
! Arguments
INTEGER, INTENT(IN)   :: npoints
REAL,    INTENT(INOUT) :: DataArray(npoints)
! Local vars
REAL    :: OldData(0:npoints+1)   ! Copy of DataArray before update
INTEGER :: i                      ! Loop counter
!$OMP PARALLEL DO PRIVATE(i)
DO i=1,npoints
  OldData(i)=DataArray(i)
ENDDO
!$OMP END PARALLEL DO
! Set points 0 and npoints+1 (note we overdimensioned OldData for this)
! to the "wrap-around" points from the opposite end for
! periodic boundary conditions
OldData(0)=OldData(npoints)
OldData(npoints+1)=OldData(1)
!$OMP PARALLEL DO PRIVATE(i)
DO i=1,npoints
  DataArray(i)=(OldData(i-1)+OldData(i+1))/2
ENDDO
!$OMP END PARALLEL DO
RETURN
END
```

FiniteDiff (Sum_data solution)

```
SUBROUTINE Sum_Data(npoints,DataArray,Sum,XMin,XMax)
IMPLICIT NONE
! Arguments
INTEGER, INTENT(IN)  :: npoints
REAL,    INTENT(IN)  :: DataArray(npoints)
REAL,    INTENT(OUT) :: Sum,XMin,XMax
! Local vars
INTEGER :: i
Sum=0
XMin=999999.0
XMax=-999999.0
!$OMP PARALLEL DO PRIVATE(i) &
!$OMP&REDUCTION(+:Sum) REDUCTION(MIN:XMin) REDUCTION(MAX:XMax)
DO i=1,npoints
  Sum=Sum+DataArray(i)
  XMin=MIN(XMin,DataArray(i))
  XMax=MAX(XMax,DataArray(i))
ENDDO
!$OMP END PARALLEL DO
RETURN
END
```

Class Exercise 4

The program md.F90 implements a simple molecular dynamics simulation in continuous real space. The velocity Verlet algorithm is used to implement the time stepping. The force and energy computations can be performed in parallel, as can the time integration. No knowledge of the application or science involved are required – the above was just to scare you ☺

4.1 Use OpenMP directives to parallelise the time integration loop

4.2 Use OpenMP directives to parallelise the computation of forces and energies loop nest

Use file md.F90

Class Exercise 4.1 (md.F90)

```
! The time integration is fully parallel
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! SAFE TO ADD OPENMP DIRECTIVES FOR THIS LOOP !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
do i = 1,np
  do j = 1,nd
    pos(j,i) = pos(j,i) + vel(j,i)*dt + 0.5*dt*dt*a(j,i)
    vel(j,i) = vel(j,i) + 0.5*dt*(f(j,i)*rmass + a(j,i))
    a(j,i) = f(j,i)*rmass
  enddo
enddo
```

Class Exercise 4.2 (md.F90)

```
! The computation of forces and energies is fully parallel.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! SAFE TO ADD OPENMP DIRECTIVES FOR THIS LOOP !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
do i=1,np
  ! compute potential energy and forces
  f(1:nd,i) = 0.0
  do j=1,np
       if (i .ne. j) then
         call dist(nd,box,pos(1,i),pos(1,j),rij,d)
         pot = pot + 0.5*v(d)
         do k=1,nd
           f(k,i) = f(k,i) - rij(k)*dv(d)/d
         enddo
       endif
  enddo
  ! compute kinetic energy
  kin = kin + dotr8(nd,vel(1,i),vel(1,i))
enddo
```

4.1 solution

```
!$omp parallel do private(i,j)
do i = 1,np
  do j = 1,nd
    pos(j,i) = pos(j,i) + vel(j,i)*dt + 0.5*dt*dt*a(j,i)
    vel(j,i) = vel(j,i) + 0.5*dt*(f(j,i)*rmass + a(j,i))
    a(j,i) = f(j,i)*rmass
  enddo
enddo
!$omp end parallel do
```

4.2 solution

```
!$omp parallel do private(i,j,k,rij,d)&
!$omp& reduction(+ : pot, kin)
do i=1,np
   ! compute potential energy and forces
   f(1:nd,i) = 0.0
   do j=1,np
        if (i .ne. j) then
          call dist(nd,box,pos(1,i),pos(1,j),rij,d)
          pot = pot + 0.5*v(d)
          do k=1,nd
            f(k,i) = f(k,i) - rij(k)*dv(d)/d
          enddo
        endif
   enddo
   ! compute kinetic energy
   kin = kin + dotr8(nd,vel(1,i),vel(1,i))
enddo
!$omp end parallel do
```