
Introduction to
OpenMP exercises

Getting setup on cca

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 ..
```

Each exercise is in its own subdirectory and has a Makefile

```
# To get the OpenMP exercise files
scp -r cca:/home/ectrain/trx/sami/27jan2016.tgz $PERM
```

Class Exercise 1 Experiment with SCHEDULE clause

[directory: schedule/]

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

Try

```
export OMP_SCHEDULE=
 STATIC
 DYNAMIC
 DYNAMIC,100
 DYNAMIC,1000
 GUIDED
 GUIDED,1000
```

Which is the best for this loop?

Class Exercise 2 (Maximum distance between points)

[directory: maxdistance/]

This exercise involves parallelization of maximum distance calculating between set of points in 3D. Use both OMP DO and OMP TASK constructs. Which one gives better performance ? Vary also the OMP_SCHEDULE.
On which core-id each thread runs ? Small C-code is used as a helper routine here.

```c
#include <sched.h>

/*
 * Find the core id the thread belongs to
 */

int coreid_ ()
{
  /* int sched_getcpu(void); */
  return sched_getcpu();
}

int coreid() { return coreid_(); }
```

Class Exercise 3 (Poisson2D)
[directory: p2d/]

This exercise involves parallelising 2D Poisson solver using Jacobi iteration. Despite being very inefficient algorithm, it shows some common parallel patterns found when parallelizing stencil type of codes.

Make sure you get the same answers when you increase the OMP_NUM_THREADS !

How does your Mlups/s (Millions of Lattice updates per seconds) go up when you increase number of threads?

For visual version use code in directory p2dviz/

Class Exercise 4 ("the bonus ball")

[directory: md/]

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

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

```
!$omp parallel do default(none) &
!$omp& private(i,j) firstprivate(np,nd,dt,rmass) &
!$omp& shared(pos,vel,a,f)
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
```

4.2 solution

```
!$omp parallel do default(none) &
!$omp& private(i,j,k,rij,d) firstprivate(np,nd) &
!$omp& shared(f,pos,rij,vel,box) &
!$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
```