HPC Tools 2024, Exercise 9 solutions

Bruno Oliveira Cattelan, Laurent Chôné

Problem 1

OpenMP as a tool tries to allow for parallelization with minimal impact in the code. For example, both serial and parallel versions used in this exercise used the same code. The serial version is achieved by not enabling OpenMP at compile time.

```
1   //*** C++ ***
2   // Simulation proper
3
4   for (n=0;n<maxt;n++) {
5     #pragma omp parallel for simd
6       for (i=0;i<nat;i++) v0[i]=v[i];
7
8     #pragma omp parallel for
9       for (i=0;i<nat;i++)
10        // New potential energy and acceleration
11        accel(nat,i,&ep[i],&a[i],box,x);
12
13    #pragma omp parallel for private(vave) firstprivate(ek, v, x, a, dt, box)
14      for (i=0;i<nat;i++) {
15
16        // Leap frog integration algorithm: update position and velocity
17        v[i]=v[i]+dt*a[i];
18        x[i]=x[i]+dt*v[i];
19
20        // Check periodic boundary conditions
21        if (x[i]<0.0 ) x[i]=x[i]+box;
22        if (x[i]>=box) x[i]=x[i]-box;
23
24        // Calculate kinetic energy (note: mass=1)
25        vave=(v0[i]+v[i])/2.0;
26        ek[i]=1.0/2.0*vave*vave;
27
28      }
29
30
31     // Calculate and print total potential end kinetic energies
32     // and their sum that should be conserved.
33    epsum=eksum=0.0;
34    #pragma omp parallel for reduction(+:epsum)
35     for (i=0;i<nat;i++) epsum+=ep[i];
36    #pragma omp parallel for reduction(+:eksum)
37     for (i=0;i<nat;i++) eksum+=ek[i];
38
39  }
```

From Figure 1 we see that although the speedup is not ideal, the result is actually much faster with 16 threads than the serial counterpart. For this experiment we used as input the following sequence 5000000.001100011000. Meaning 500000 atoms, 0.001 as the time interval, 1000 timesteps and velocities initialized with 1. The last two numbers refer to the frequency of the outputs of the simulation. We leave them here to ensure the code is not optimized away by our $O2$ optimization level. We also see that for larger than 16 threads there is a drop in our parallelization efficiency, which seems to pick back up after. This comes from the fact that the computer used for this had indeed 16 physical processors. If we continued the experiments for even larger number of threads, we would see this behavior repeated in intervals of 16.

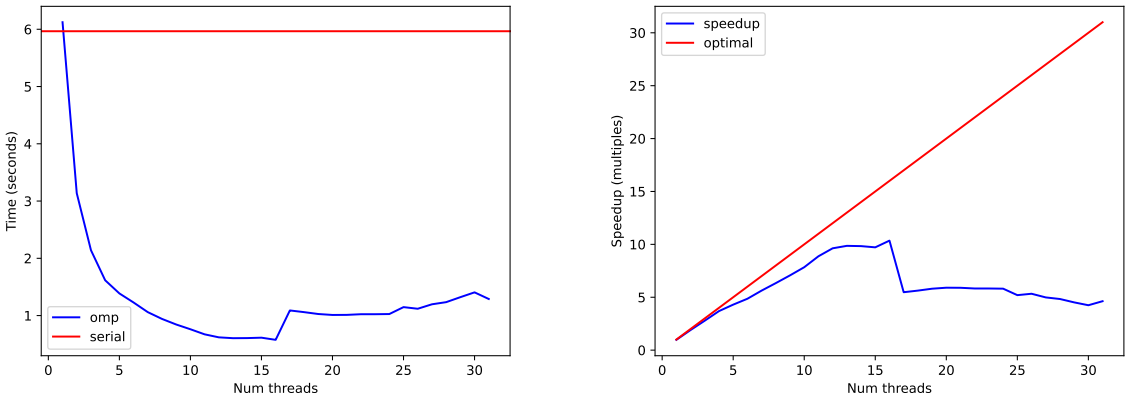

Figure 1: OpenMP vs serial

```
!*** Fortran ***
! Simulation proper
time_loop: do n=1,maxt

  epsum = 0.0
  eksum = 0.0

  v0=v

  !$omp parallel do
  atom_loop1: do i=1,nat
    ! New potential energy and acceleration
    call accel(i,ep(i),a(i))
  end do atom_loop1
  !$omp end parallel do

  !$omp parallel do
  atom_loop2: do i=1,nat
    ! Leap frog integration algorithm: update position and velocity
    v(i)=v(i)+dt*a(i)
    x(i)=x(i)+dt*v(i)
    ! Check periodic boundary conditions
    if (x(i)<0.0 ) x(i)=x(i)+box
    if (x(i)>=box) x(i)=x(i)-box
    ! Calculate kinetic energy (note: mass=1)
    ek(i)=1.0/2.0*((v0(i)+v(i))/2.0)**2
  end do atom_loop2
  !$omp end parallel do

  ! Calculate and print total potential end kinetic energies
  ! and their sum that should be conserved.
  !$omp parallel do reduction(+:epsum)
  epsum_red: do i=1,nat
    epsum=epsum+ep(i)
  end do epsum_red
  !$omp end parallel do
  !$omp parallel do reduction(+:eksum)
  eksum_red: do i=1,nat
    eksum=eksum+ek(i)
  end do eksum_red
  !$omp end parallel do
```

```
42 end do time_loop
```

Finally, it is important to keep in mind Amdahl's law. The Figure 2 shows the theoretical speedup expected for different parallelizable portions of a code. However, super-linear speedups can also be achieved in specific cases!

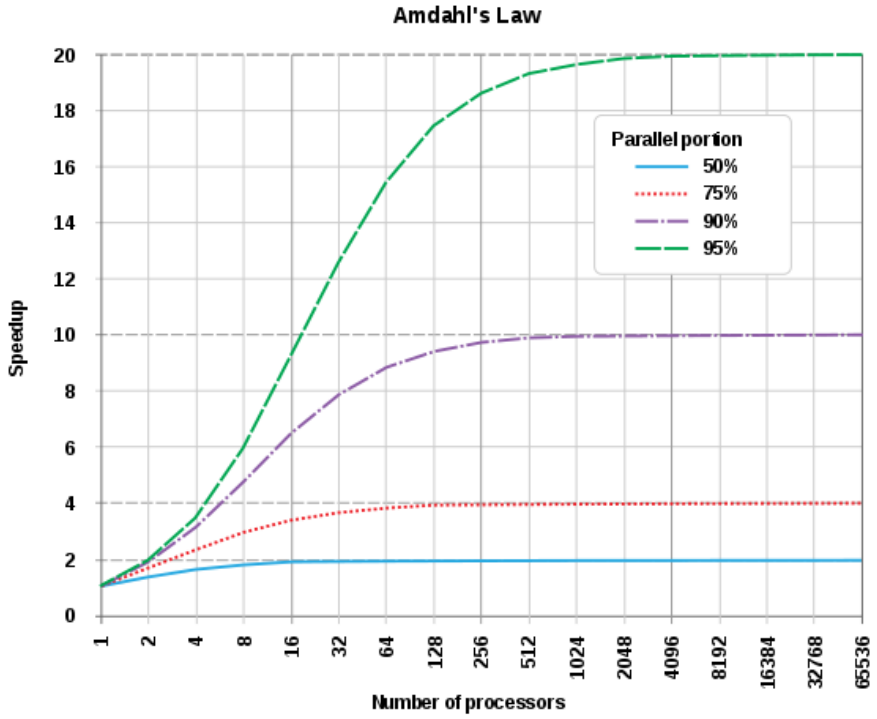

Figure 2: Amdahl's law (Wikipedia)