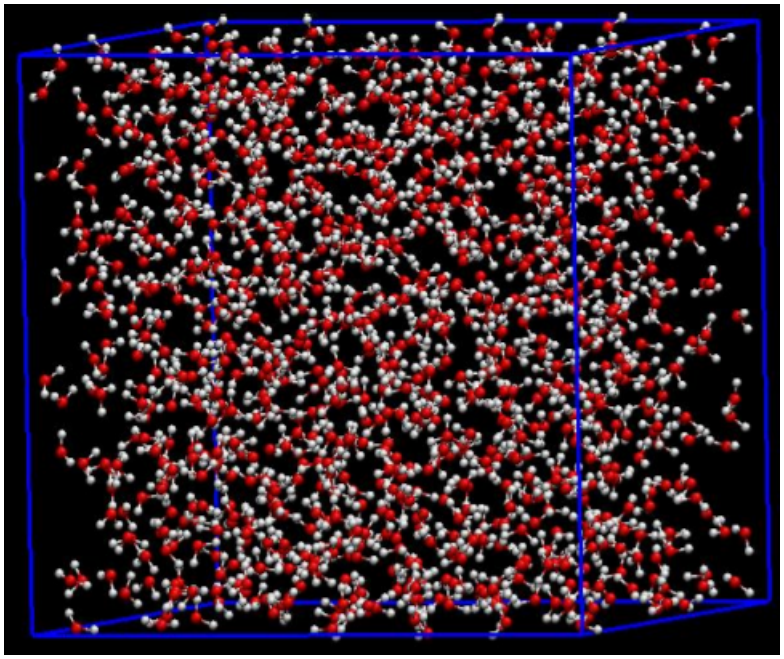
Calculation of Energy of configuration for a Box of Water Molecules using OpenMP

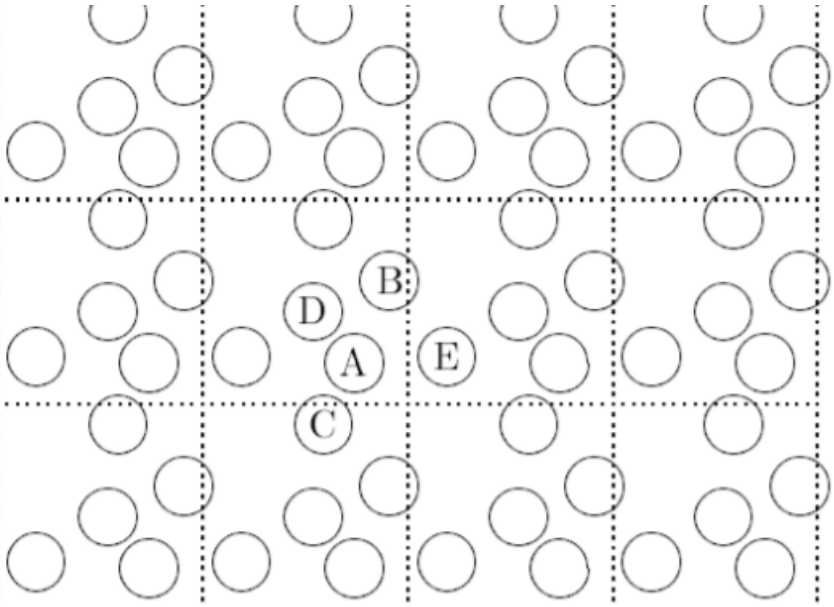
By - Hrithik Rai Saxena



Problem Statement - To parallelize the serial code using **OpenMP**(Open Multi-Processing) for calculating the energy of configuration for a box of water molecules and comparing its efficiency against different settings for number of processing threads and computing units.

Given - Position of each atom , Size of the Box, **.gro** input file(trajectory generated by a software called **Gromax**, used for molecular dynamic simulation) for three settings namely 1k.gro(1000 molecules), 10k.gro(10,000 molecules) and 100k.gro(100,000 molecules), Serial code for calculating the Energy Configuration.

Note - Periodic Boundry Conditions are used wherein the boxes will be replicated, each surronded by its own replica such that the configuration is not disturbed by molecules surpassing the boundries of the box and ensuring box integrity using their replicas.



Periodic Boundry Conditions

Program components to be parallelized:

Energy of Configuration Calculation

```
double energy12(int i1,int i2){
// ============================
  int m,n,xyz;
  double shift[3],dr[3],mn[3],r6,distsq,dist,ene=0;
  const double sig=0.3166,eps=0.65,eps0=8.85e-12,e=1.602e-19,Na=6.022e23,q[3]={-0.8476,0.4238,0.4238};
  double elst,sig6;
  elst=e*e/(4*3.141593*eps0*1e-9)*Na/1e3,sig6=pow(sig,6);

  // periodic boundary conditions
  for(xyz=0;xyz<=2;xyz++){
   dr[xyz]=r[i1][0][xyz]-r[i2][0][xyz];shift[xyz]=-L*floor(dr[xyz]/L+.5); //round dr[xyz]/L to nearest integer
   dr[xyz]=dr[xyz]+shift[xyz];
  }
  distsq=sqr(dr[0])+sqr(dr[1])+sqr(dr[2]);
  if(distsq<rcutsq){ // calculate energy if within cutoff
    r6=sig6/pow(distsq,3);
    ene=4*eps*r6*(r6-1.); // LJ energy
    for(m=0;m<=2;m++){
      for(n=0;n<=2;n++){
        for(xyz=0;xyz<=2;xyz++) mn[xyz]=r[i1][m][xyz]-r[i2][n][xyz]+shift[xyz];
        dist=sqrt(sqr(mn[0])+sqr(mn[1])+sqr(mn[2]));
        ene=ene+elst*q[m]*q[n]/dist;
    } }
  }
  return ene;
}
```

Function to Calculate Energy of Configuration

```
cputime2 = clock();    // assign initial CPU time (IN CPU CLOCKS)
gettimeofday(&start, NULL); // returns structure with time in s and us (microseconds)

for(i=0;i<nmol-1;i++){ // calculate energy as sum over all pairs
  for(j=i+1;j<nmol;j++) energy=energy+energy12(i,j);
}

cputime3= clock()-cputime2;      // calculate  cpu clock time as difference of times after-before
gettimeofday(&end, NULL);
```

Code in main function to calculate the Energy wherein the outer loop needs to be parallelized

Essential variables for energy calculation:

nmol – number of molecules, determined from natoms

r[maxnum][3][3] – coordinates of atoms

L – size of cubic box with configuration

Other parameters of calculation are declared as constants or derived from them

Parralelizing the outer Loop using OpenMP :

Here we parallelize the Outer Loop using OpenMP for 12 threads. Also after setting the npairs variable as private, each thread displays the number of pairs it calculated. Also in the end all the partial energies calculated by the threads were summed using reduction clause.

Also after storing the npairs in an array, we find the minimum and maximum values to compute the load imbalance.

```
long npairs_array[12];
// OpenMP Parralelization
#pragma omp parallel private(i,j) num_threads(12) reduction(+:energy)
{
  long npairs = 0;
  #pragma omp for schedule(dynamic,100)
  for(i=0;i<nmol-1;i++){ // calculate energy as sum over all pairs
    for(j=i+1;j<nmol;j++){
      energy=energy+energy12(i,j);
      npairs++;
    }
  }
  int me = omp_get_thread_num();
  npairs_array[me] = npairs;
  printf(" I am thread number %d and the number of pairs calculated are : %li\n",omp_get_thread_num(),npairs);

}


// Calculating the load imbalance:
int minpairs = min(npairs_array,12);
int maxpairs = max(npairs_array,12);
printf("Minimum pairs are %d\n",minpairs);
printf("Maximum pairs are %d\n",maxpairs);
double load_imb = 2.*(maxpairs-minpairs)/(0.+maxpairs+minpairs);
```

Observation Table :

Comparing the Values with different Scheduling Techniques and with Serial and MPI Version (12 threads, 1 Machine):

scheduling method	time [s]	Load Imbalance
static	6.032	0.0862
static, 1	6.094	0.0985
static, 100	6.013	0.0908
static, 10000	6.085	0.106
dynamic, 100	5.977	0.0842
guided, 100	5.971	0.0947
auto	5.993	0.1066
Serial	61.329	NA
MPI	6.706	0.000991

Observations :

1**. OpenMP comes with a ease of programming** as compared to MPI Parallelization.

2. But **the operations happens inside a black box** with not much control over the parallelization. Also upon every run of the program we get different values for time and load imbalance.

3. Compared to the MPI Implementation, The wall time was low for the 'auto' scheduler setting in OpenMP but **since the MPI parallel code was hardcoded, we had more control over the algorithm to minimize the Load Imbalance**.

4. **MPI provides us with much better control over the parallel environment** that's why no setting of OpenMP was able to beat the load imbalance from the MPI parallelization.

5. **The best setting with time in focus** -> guided scheduler with chunk size 100 (This may vary for every execution because the number of npairs were different for different runs.)

6. **The best setting with Load Imbalance in focus** -> dynamic with chunk size 100.(This may vary too for every run.)

7. Unlike MPI where we were to obtain similar results for a particular setting, in **OpenMP the results varied**. Like, for the dynamic scheduler where the runtime figures out which thread is free and can take up the next chunk, we dont have control over this part.

8. The **overhead contribution** is different for different schedulers and chunk sizes.

9. **Too large chunk size** may not justify the load balancing (thread with more work may get stuck while others wait.) and **too small chunk size** will contribute to larger overhead values for scheduling like dynamic.