Parallel(?) Tempering

Rômulo Cenci

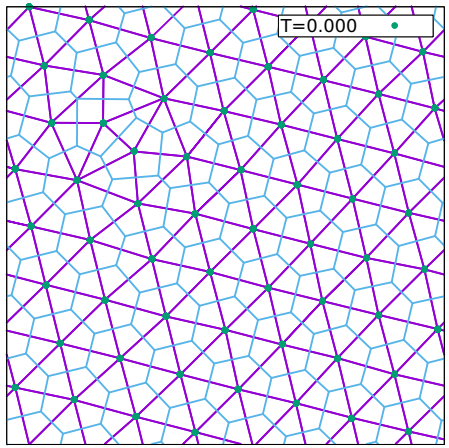
December 13, 2019

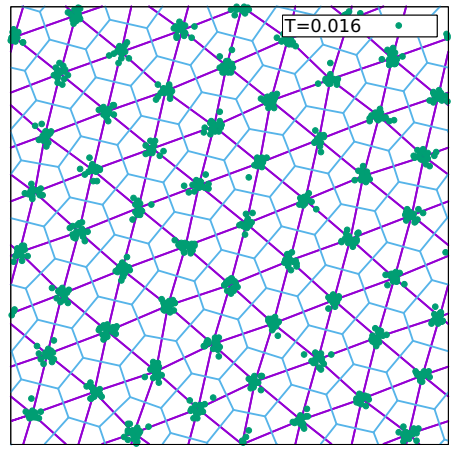
So.. Who am I?

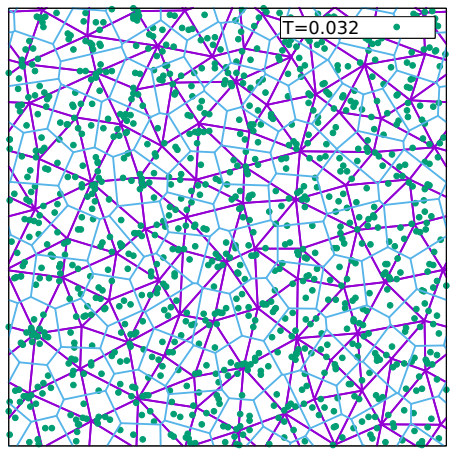
- Universidade Federal de Santa Catarina - Florianópolis, Brazil.
- Advisor: Lucas Nicolao
- Research area: Condensed Matter/Statistical Mechanics

My masters project

- I'm studying a model called GEM-α that could model some colloidal suspensions and ultracold atoms;
- And that forms some special patterns know as cluster cristals.

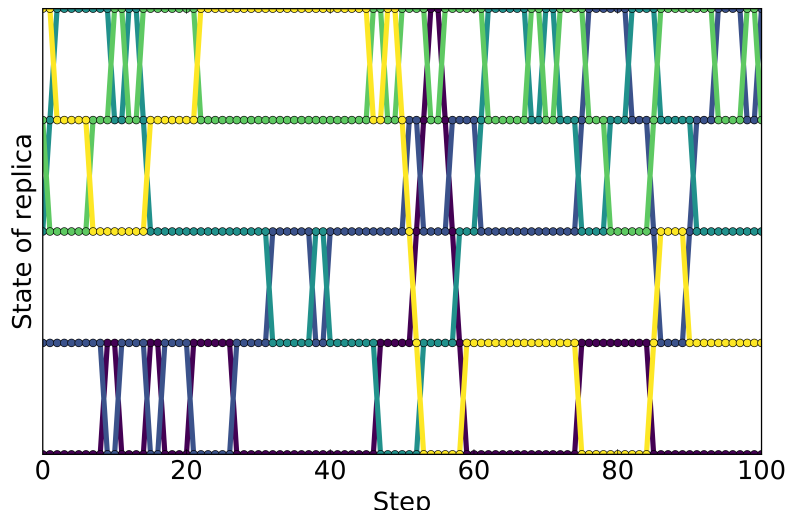






What is my problem?

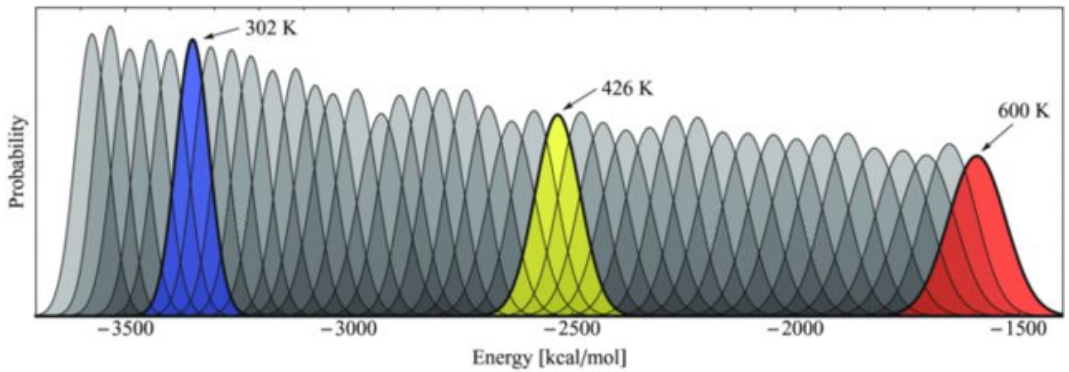
- We generally want to get out of metastable states;
- Equilibrate simulation in various temperatures;
- We don't want to waste time doing bad annealing that may not be in a equilibrium state.



- So, to solve all that problems we can use the **parallel tempering** technique;

Parallel Tempering

- In my simulations, the temperature is a fixed variable and energy could flutuate $\rightarrow$ Canonical ensemble;
- So if we look at the energy histograms, we will see a overlap between differents temperatures;

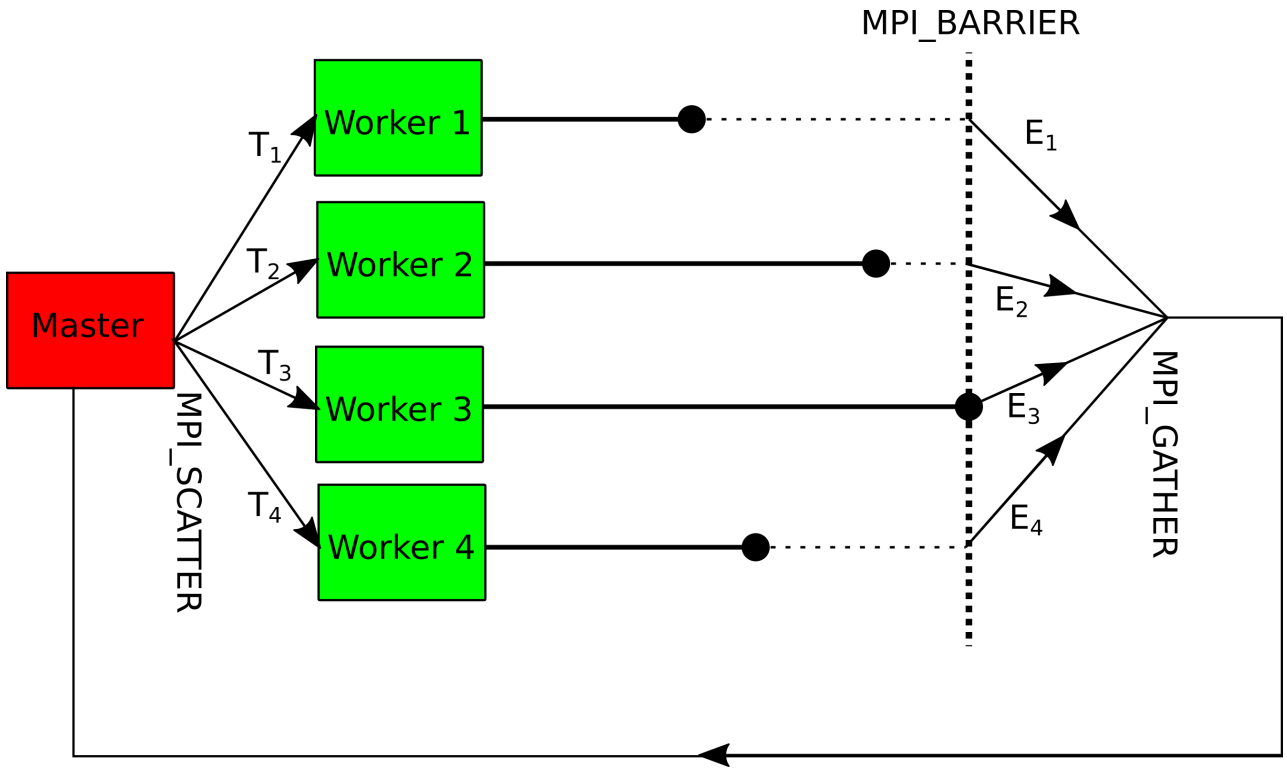


- This overlap could be interpreted as the probability of the configuration at this temperature could be at the other temperature;
- We can calculate the total probability of two configurations have been changed using the Boltzmann weight;

$$p(i,j) = min\left(1, \frac{e^{-\beta_i E_j - \beta_j E_i}}{e^{-\beta_i E_i - \beta_j E_j}}\right) = min\left(1, e^{(E_i - E_j)(\beta_i - \beta_j)}\right)$$

How can we parallelize?

- Just a task farming, so...

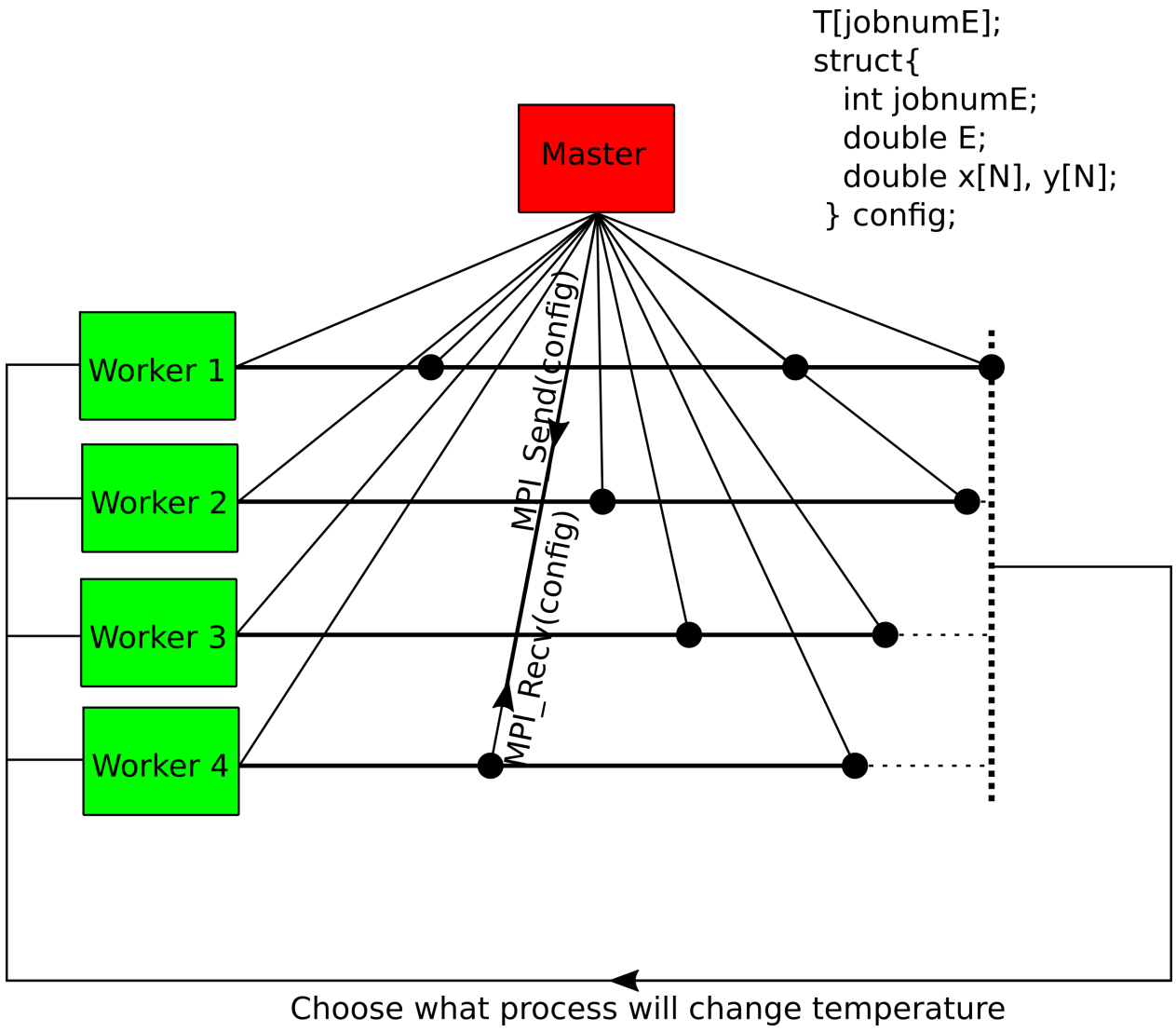


```
for(trials=0; trials<300; trials++){
  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Scatter(allT, 1, MPI_DOUBLE, &T, 1, MPI_DOUBLE, 0, comm);

  // update all the ensembles in the same time, one per thread
  for(k=0; k<Kmax; k++){
   update(x, y, T);
  }

  // wait all the evolutions finish
  MPI_Barrier(MPI_COMM_WORLD);

  // measure the energy of each ensemble and send to master
  E = ener(x, y);

  MPI_Gather(&E, 1, MPI_DOUBLE, allE, 1, MPI_DOUBLE, 0, comm);

  // decides who changes temperature with who
  if(pt==0){
    for(i=0; i<PT-1; i++){
      delta=-(1./allT[confT[i+1]]-1./allT[confT[i]])*(allE[confT[i+1]]-allE[confT[i]]);
      if(delta<0 || FRANDOM<exp(-delta)){
        aux=confT[i];
        confT[i]=confT[i+1];
        confT[i+1]=aux;

        aux=allT[i+1];
        allT[i+1]=allT[i];
        allT[i]=aux;
      }
    }
  }
}
```

Load Balance

- Okay, so now we have a truly **parallel** tempering, running at different cores;
- But to running Ntemps temperatures we need Ntemps cores;
- So, when we have a small cluster, how the ours: 14 quad-core nodes = 48 processing elements, we can just run 48 temperatures;
- How to simulate more temperatures in this enviroment without losing much time with one process and almost anything with the other?

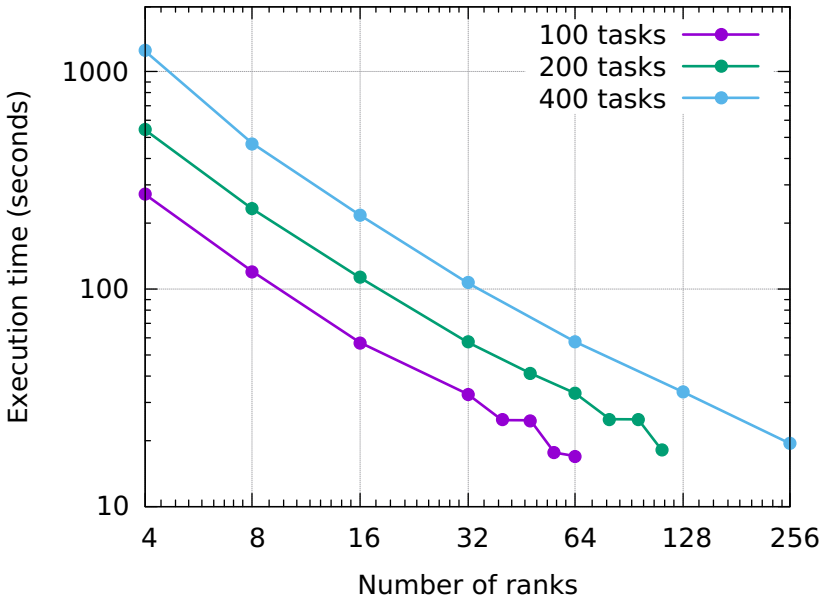
Load Balance

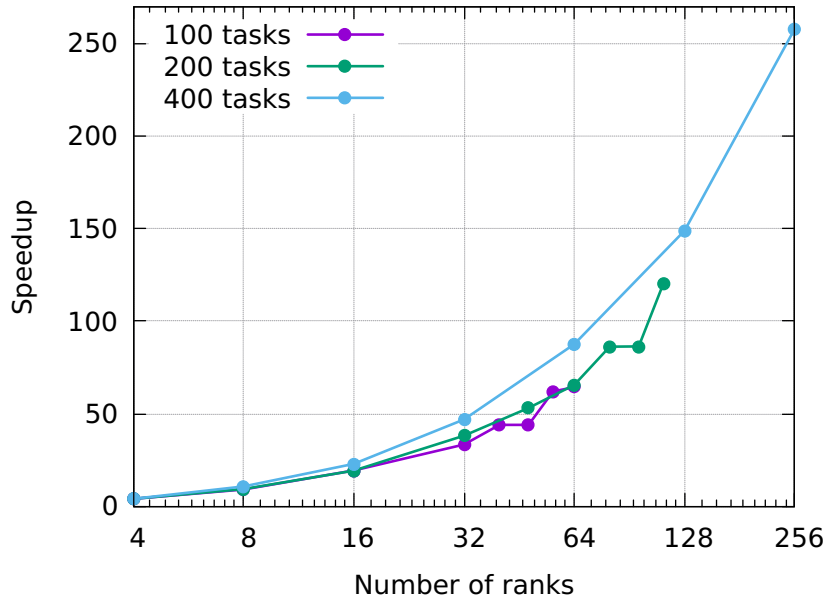


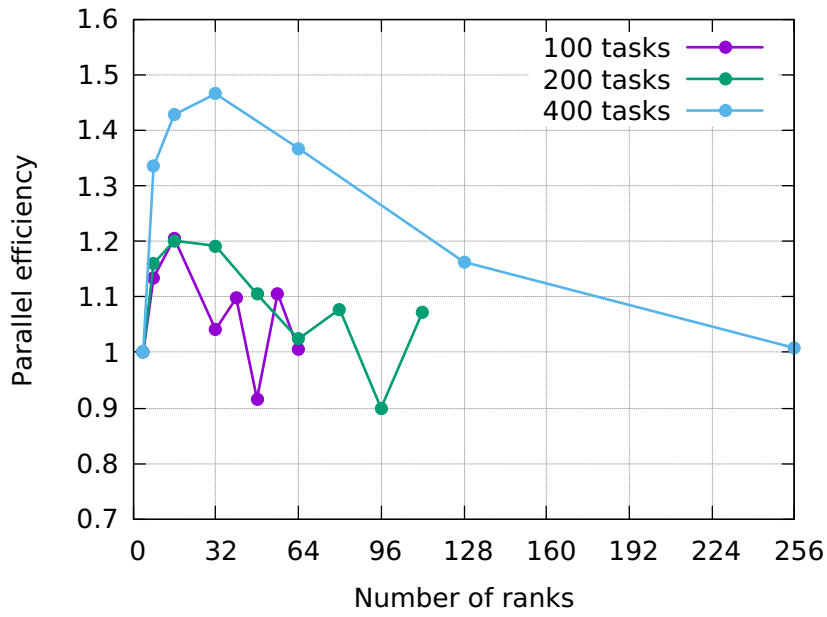
```
// master process
if(!rank){
  // initiates sending one process to each rank
  for(jobnum=0; jobnum<Nranks-1; jobnum++){
    configs[jobnum].jobnumE = jobnum;
    MPI_Send(&configs[jobnum], 1, MPI_Config,\
             jobnum+1, flag, comm);
  }

  // still has jobs to do
  for(jobnum=Nranks-1; jobnum<Ntemps; jobnum++){
    // wait for any rank finish to send another job
    MPI_Recv(&jobnumE, 1, MPI_INT,\
             MPI_ANY_SOURCE, 0, comm, &stat);
    MPI_Recv(&configs[jobnumE], 1, MPI_Config,\
             stat.MPI_SOURCE, 1, comm,\
             MPI_STATUS_IGNORE);

    configs[jobnum].jobnumE = jobnum;
    MPI_Send(&configs[jobnum], 1, MPI_Config,\
             stat.MPI_SOURCE, 0, comm);
  }

  // for the last ones just recv energies and\
     send a -1 to says that its over
  for(dest=1; dest<Nranks; dest++){
    MPI_Recv(&jobnumE, 1, MPI_INT,\
             MPI_ANY_SOURCE, 0, comm, &stat);
    MPI_Recv(&configs[jobnumE], 1, MPI_Config,\
             stat.MPI_SOURCE, 1, comm,\
             MPI_STATUS_IGNORE);

    configs[jobnum].jobnumE = -1;
    MPI_Send(&configs[jobnum], 1, MPI_Config,\
             stat.MPI_SOURCE, 0, comm);
  }
}
```

```
// worker process
else{
  while(1){
    MPI_Recv(&configs[0], 1, MPI_Config,\
             0, 0, comm, MPI_STATUS_IGNORE);
    if(configs[0].jobnumE>=0) {
      // update Nranks ensembles at the same time,\
         one per rank
      for(k=0; k<Kmax; k++){
        update(configs[0].x, configs[0].y,\
               allT[configs[0].jobnumE]);
      }

      // measure the energy and send back to the root
      configs[0].E=ener(configs[0].x, configs[0].y);
      MPI_Send(&configs[0].jobnumE, 1, MPI_INT,\
               0, 0, comm);
      MPI_Send(&configs[0], 1, MPI_Config,\
               0, 1, comm);
    }
    else{
      break;
    }
  }
}
```

Results


100 tasks
200 tasks
400 tasks
Execution time (seconds)
1000
100
10
4
8
16
32
64
128
256
Number of ranks


100 tasks
200 tasks
400 tasks
Speedup
250
200
150
100
50
0
4
8
16
32
64
128
256
Number of ranks


100 tasks
200 tasks
400 tasks
Parallel efficiency
1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0
32
64
96
128
160
192
224
256
Number of ranks