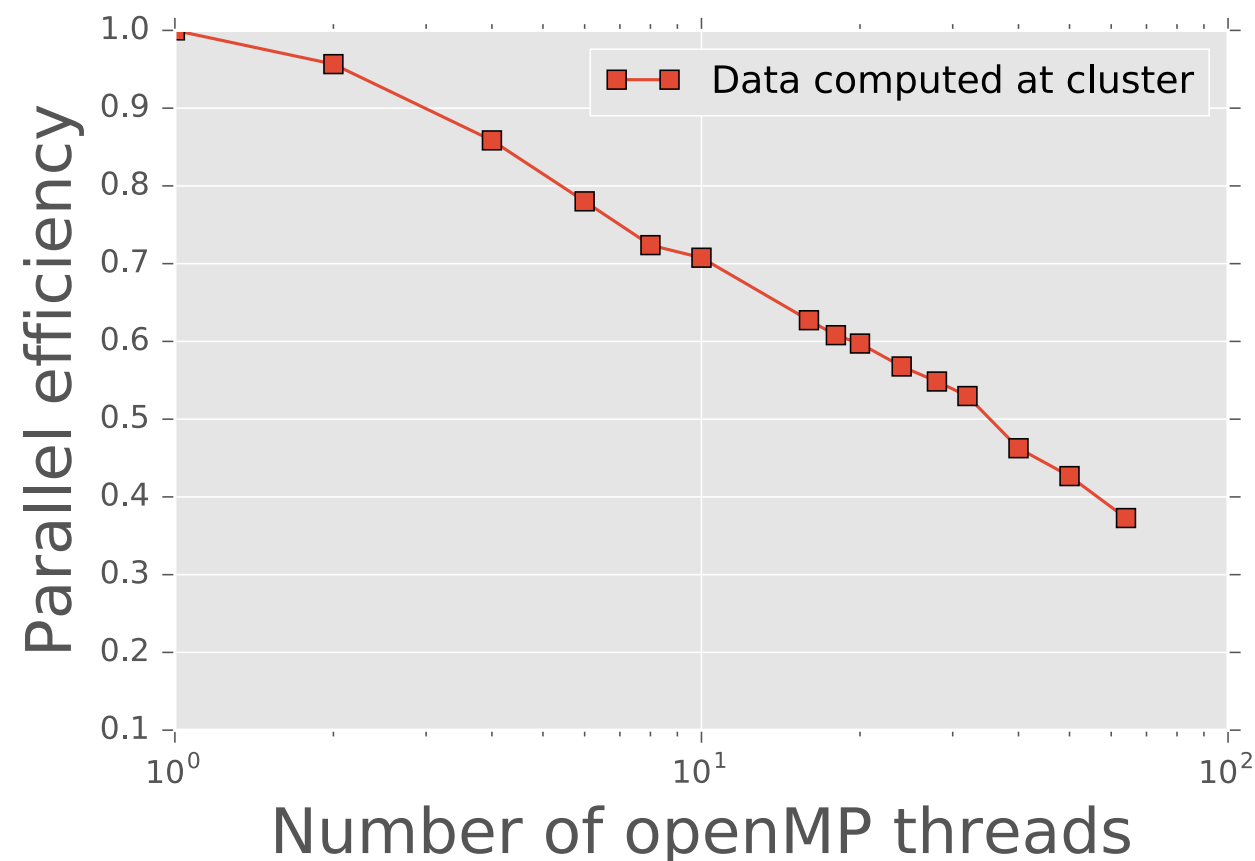


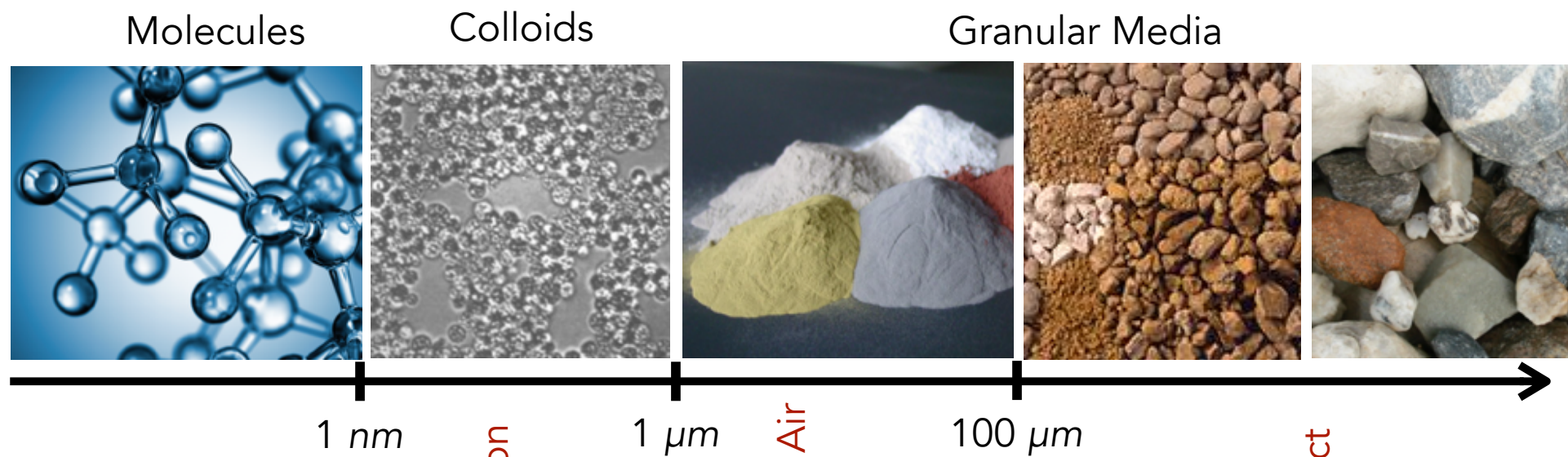
From scratch to optimal number of threads: DEM and openMP

William Oquendo, woquendo@gmail.com



THE PROBLEM

Why did I use my code?

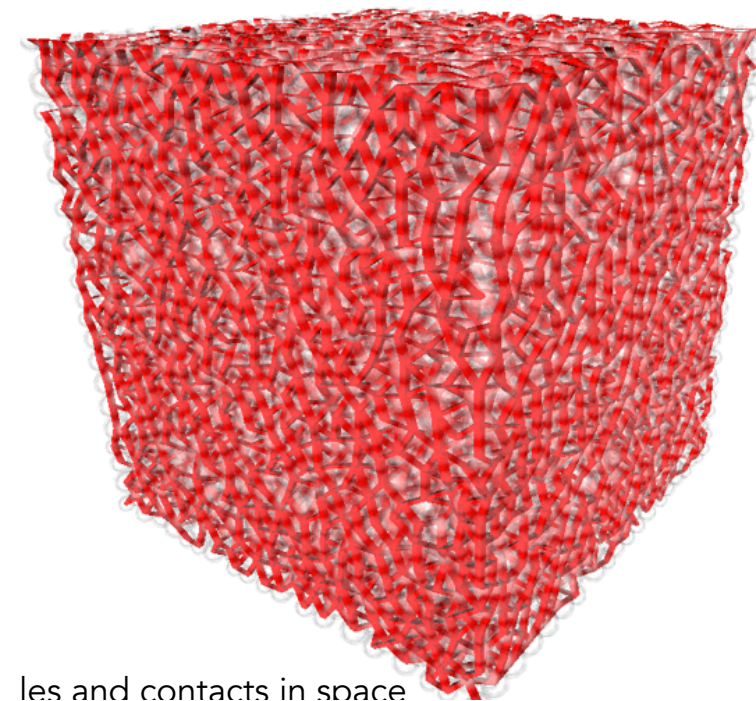
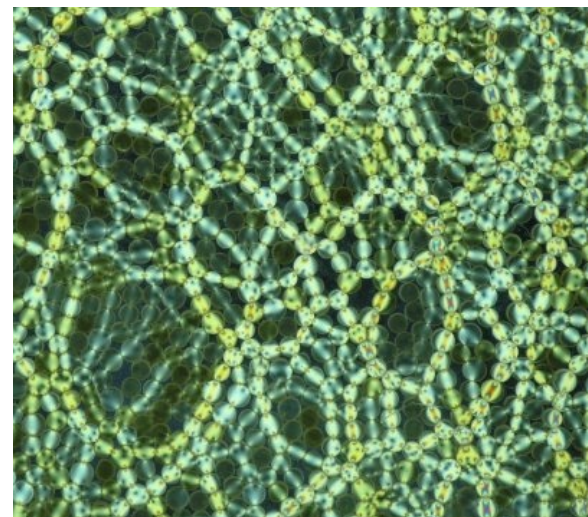
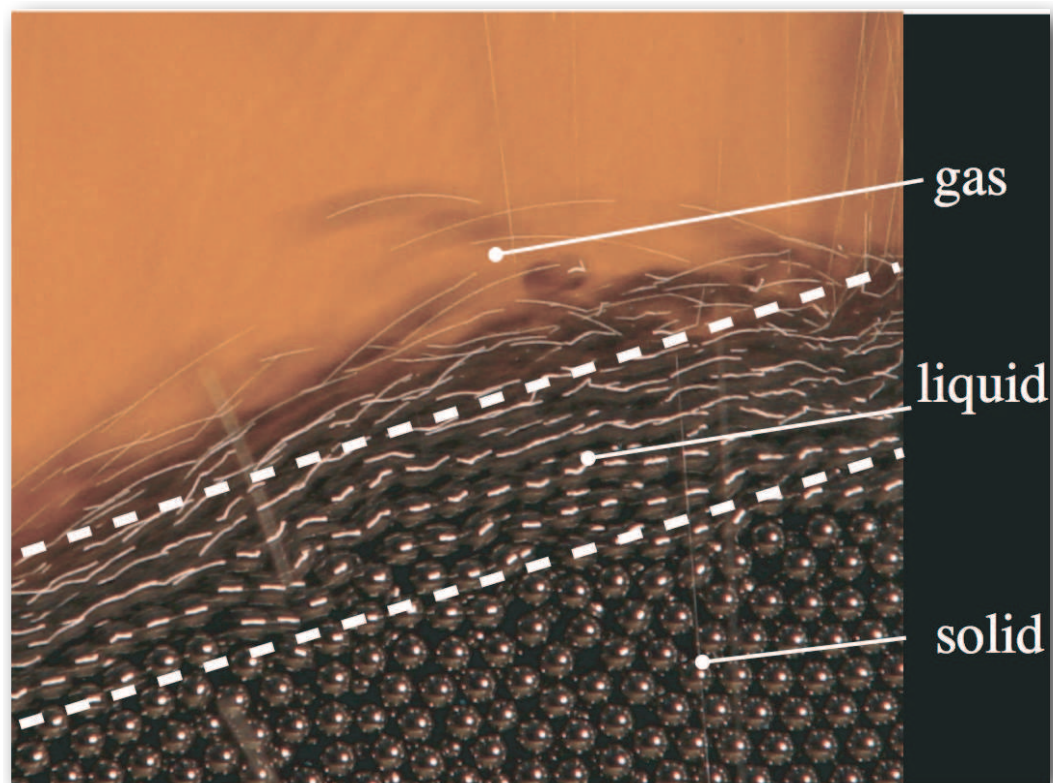


**Frictional
Characteristic particle size**

Thermal agitation
VdW Forces

Interactions with Air
VdW Forces
Humidity

Frictional contact



THE SERIAL VERSION

My code is complex, let's implement a “simpler” version

From scratch ... Maybe not a good idea!
(about 400 loc)

```
2 #include "random_helper.h"  
3 #include "config.h"  
4 #include "types_dem.h"  
5 #include "prepro.h"  
6 #include "helper_dem.h"  
7 #include "time_evolution.h"  
8 #include "forces.h"
```


From scratch ... Maybe not a good idea!
(about 400 loc)

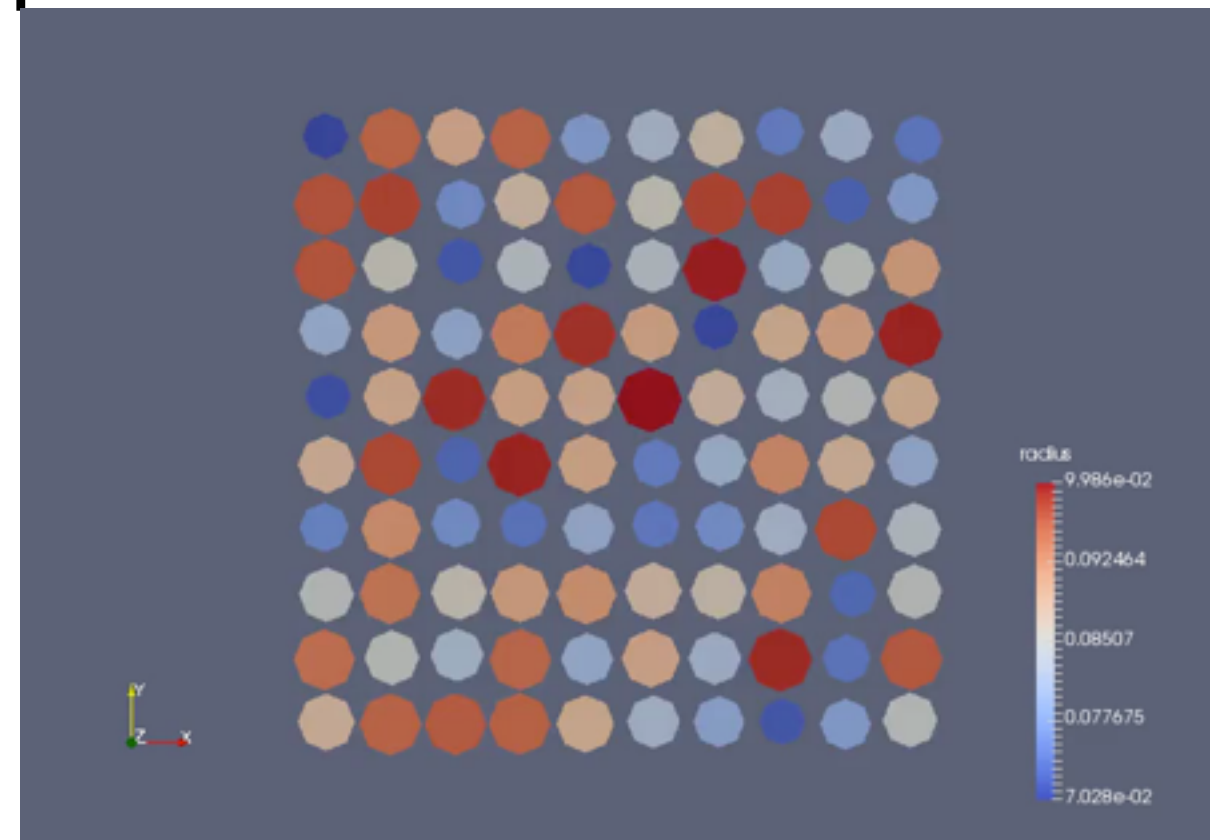
```
2 #include "random_helper.h"  
3 #include "config.h"  
4 #include "types_dem.h"  
5 #include "prepro.h"  
6 #include "helper_dem.h"  
7 #include "time_evolution.h"  
8 #include "forces.h"
```

- **Preprocessing : Put particles on a grid, disorder, random vets, assign radii, set fixed quantities**
- **Fix Vx and Fy on top wall (simulates simple shear)**
- **Use leap frog for time evolution**
- **Periodic boundary conditions (horizontal)**
- **Postprocessing: Visualization with python + paraview**

From scratch ... Maybe not a good idea! (about 400 loc)

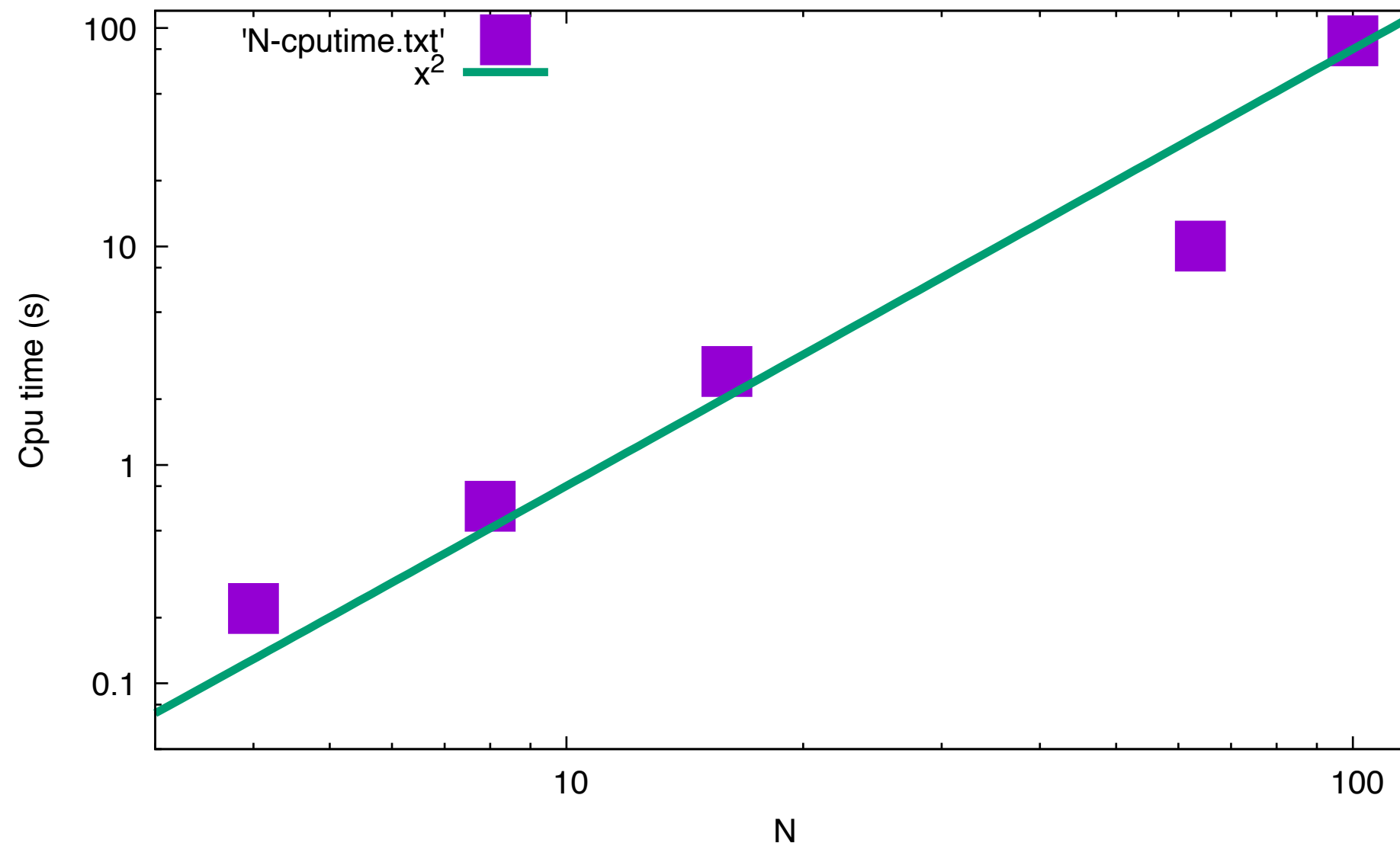
```
2 #include "random_helper.h"  
3 #include "config.h"  
4 #include "types_dem.h"  
5 #include "prepro.h"  
6 #include "helper_dem.h"  
7 #include "time_evolution.h"  
8 #include "forces.h"
```

- **Preprocessing : Put particles on a grid, disorder, random vets, assign radii, set fixed quantities**
- **Fix Vx and Fy on top wall (simulates simple shear)**
- **Use leap frog for time evolution**
- **Periodic boundary conditions (horizontal)**
- **Postprocessing: Visualization with python + paraview**



Simplest algorithm -> Quadratic growth!

Profiling/Debugging with valgrind



```

.      .      .      .      .      .      .      .      .      .      .      .      .      .      .      .
.      .      .      .      .      .      .      .      .      .      .      .      .      .      .      .
.      .      .      .      .      .      .      .      .      .      .      .      .      .      .      .
16,000,000  1,600,000  4,800,000  1  0  0  1  .      .      .      .      .      .      .      .      .      .
704,000,000 25,600,000 24,000,000 1  0  0  1  .      .      .      .      .      .      .      .      .      .
384,000,000 384,000,000 .      .      .      .      .      .      .      .      .      .      .      .      .
2,688,000,000 960,000,000 576,000,000 1  0  0  1  .      .      .      .      .      .      .      .      .
1,728,020,068 384,004,052 1,148 60 99 0 2 7 .      .      .      .      .      .      .      .      .
576,000,000 384,000,000 .      .      .      .      .      .      .      .      .      .      .      .      .
1,536,000,000 0 0 1 0 0 1 .      .      .      .      .      .      .      .      .      .
576,000,000 384,000,000 0 1 0 0 1 .      .      .      .      .      .      .      .      .      .
.      .      .      .      .      .      .      .      .      .      .      .      .      .      .      .
384,000,000 192,000,000 .      .      .      .      .      .      .      .      .      .      .      .      .
.      .      .      .      .      .      .      .      .      .      .      .      .      .      .      .
30,383,242 30,383,242 .      .      .      .      .      .      .      .      .      .      .      .      .
.      .      .      .      .      .      .      .      .      .      .      .      .      .      .      .

// compute all forces for grains
const int ngrains = grains.size();
for (int idx = 0; idx < ngrains; ++idx) {
    for (int jdx = idx+1; jdx < ngrains; ++jdx) {
        double dx = grains[idx].R[0] - grains[jdx].R[0];
        dx -= conf.LX*std::lrint(dx/conf.LX); // periodic, see Alen-Tildesley page 30
        const double dy = grains[idx].R[1] - grains[jdx].R[1];
        const double rij = std::sqrt(dx*dx + dy*dy);
        const double delta = grains[idx].rad + grains[jdx].rad - rij;
        // add force to both grains
        if (delta > 0.0) {
            // dissipative normal force
            const double dVx = grains[idx].V[0] - grains[jdx].V[0];

```


PARALLEL (OMP) OPTIMAL NUMBER OF THREADS

First parallel approach : openmp

What is the ideal number of threads?

```
22 #pragma omp parallel for private(idx)
23     for (idx = 0; idx < ngrains; ++idx) {
24         for (int jdx = idx+1; jdx < ngrains; ++jdx) {
```

First parallel approach : openmp

What is the ideal number of threads?

```
22 #pragma omp parallel for private(idx)
23     for (idx = 0; idx < ngrains; ++idx) {
24         for (int jdx = idx+1; jdx < ngrains; ++jdx) {
```

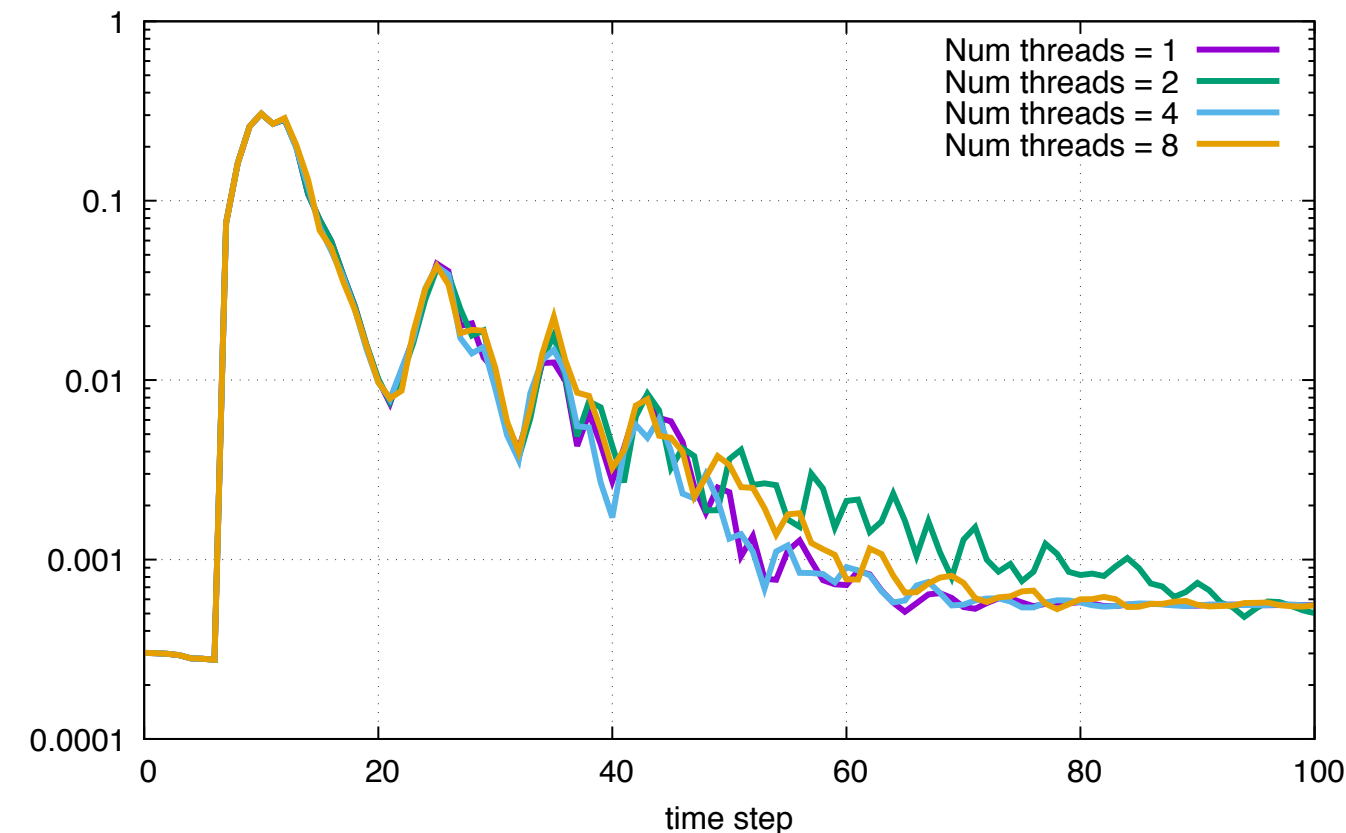
```
21     double dx, dy, rij, urij, delta, dVx, dVy, tmp, Fnx, Fny;
22     int ii, jj;
23 #pragma omp parallel for shared(grains, conf) private(ii, jj, dx, dy, rij, urij, delta, dVx, dVy, tmp, Fnx, Fny)
24     for (ii = 0; ii < ngrains; ++ii) {
25         for (jj = ii+1; jj < ngrains; ++jj) {
26             dx = grains[ii].R[0] - grains[jj].R[0];
27             //if (dx <= conf.LX/2) dx += conf.LX;
```

First parallel approach : openmp

What is the ideal number of threads?

```
22 #pragma omp parallel for private(idx)
23     for (idx = 0; idx < ngrains; ++idx) {
24         for (int jdx = idx+1; jdx < ngrains; ++jdx) {
```

```
21     double dx, dy, rij, urij, delta, dVx, dVy, tmp, Fnx, Fny;
22     int ii, jj;
23 #pragma omp parallel for shared(grains, conf) private(ii, jj, dx, dy, rij, urij, delta, dVx, dVy, tmp, Fnx, Fny)
24     for (ii = 0; ii < ngrains; ++ii) {
25         for (jj = ii+1; jj < ngrains; ++jj) {
26             dx = grains[ii].R[0] - grains[jj].R[0];
27             //if (dx <= conf.LX/2) dx += conf.LX;
```



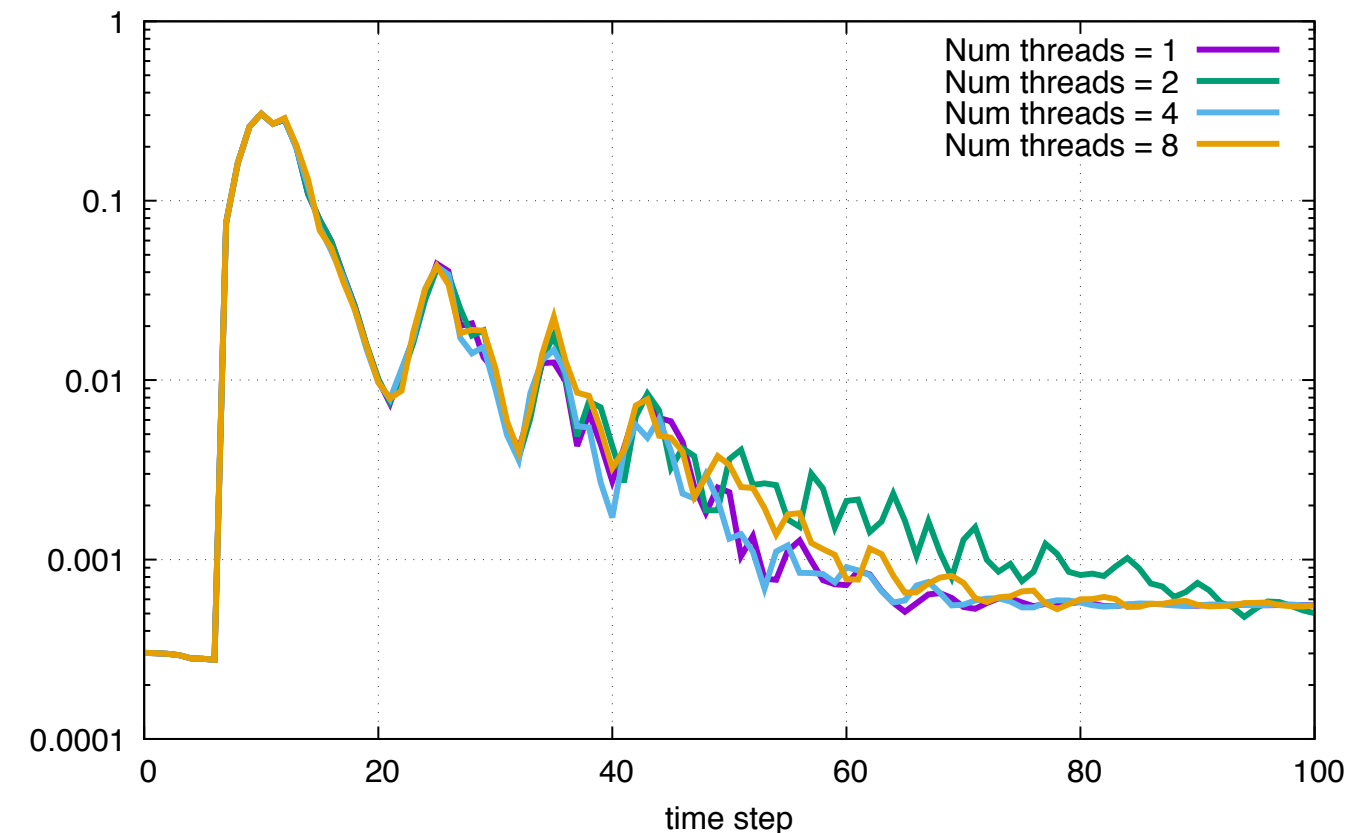
First parallel approach : openmp

What is the ideal number of threads?

```
22 #pragma omp parallel for private(idx)
23   for (idx = 0; idx < ngrains; ++idx) {
24     for (int jdx = idx+1; jdx < ngrains; ++jdx) {
```

```
21   double dx, dy, rij, urij, delta, dVx, dVy, tmp, Fnx, Fny;
22   int ii, jj;
23 #pragma omp parallel for shared(grains, conf) private(ii, jj, dx, dy, rij, urij, delta, dVx, dVy, tmp, Fnx, Fny)
24   for (ii = 0; ii < ngrains; ++ii) {
25     for (jj = ii+1; jj < ngrains; ++jj) {
26       dx = grains[ii].R[0] - grains[jj].R[0];
27       //if (dx <= conf.LX/2) dx += conf.LX;
```

**But time appears to be
slower for more
threads :(**



First parallel approach : openmp

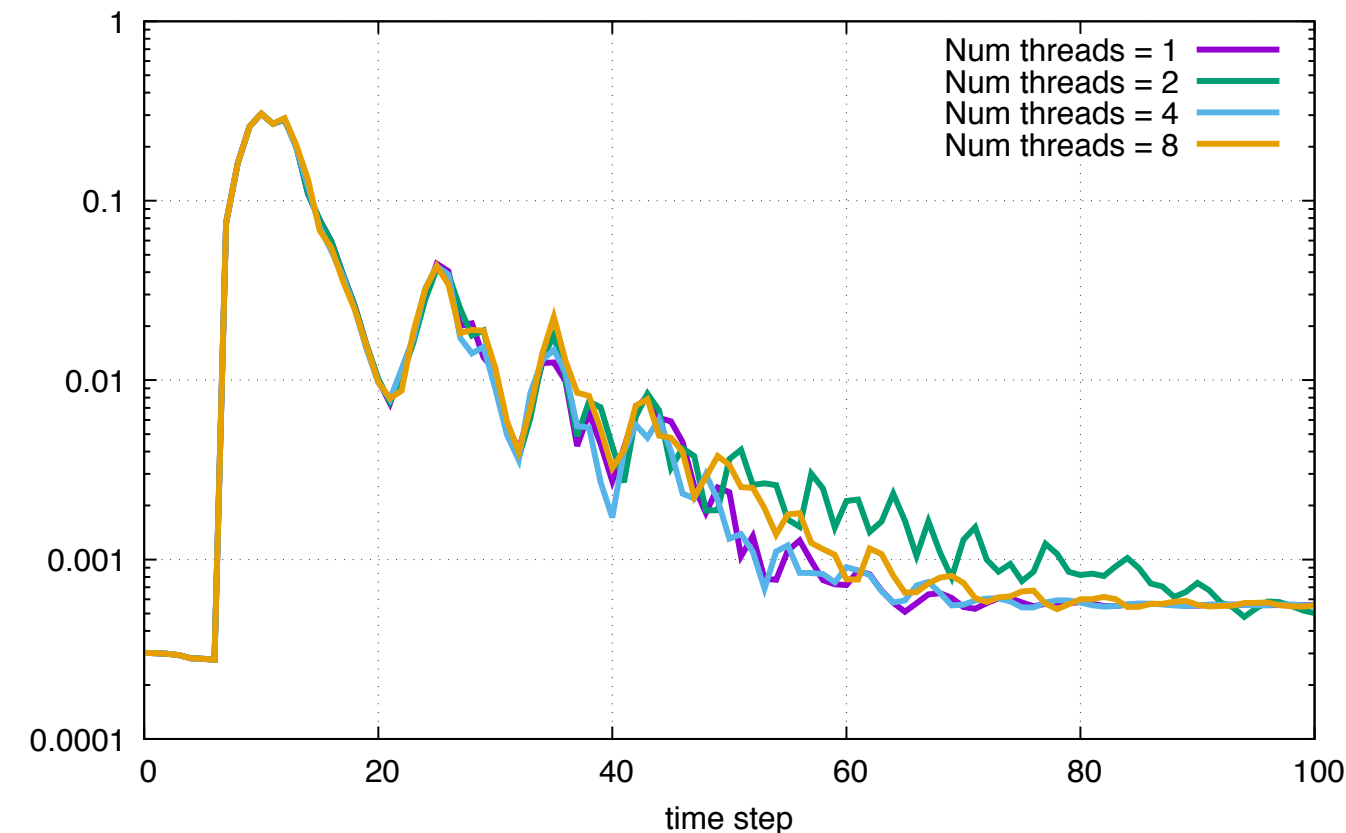
What is the ideal number of threads?

```
22 #pragma omp parallel for private(idx)
23   for (idx = 0; idx < ngrains; ++idx) {
24     for (int jdx = idx+1; jdx < ngrains; ++jdx) {
```

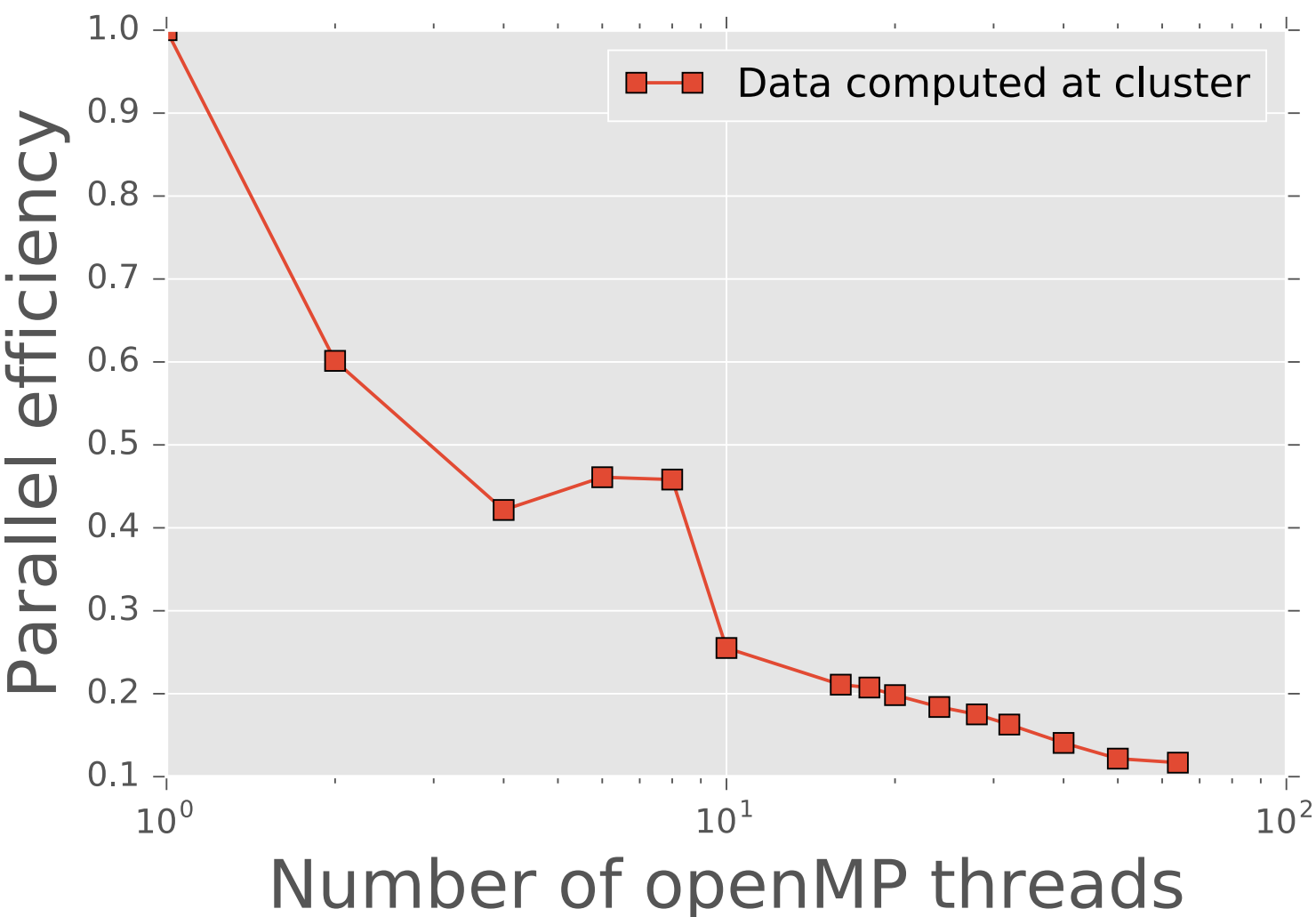
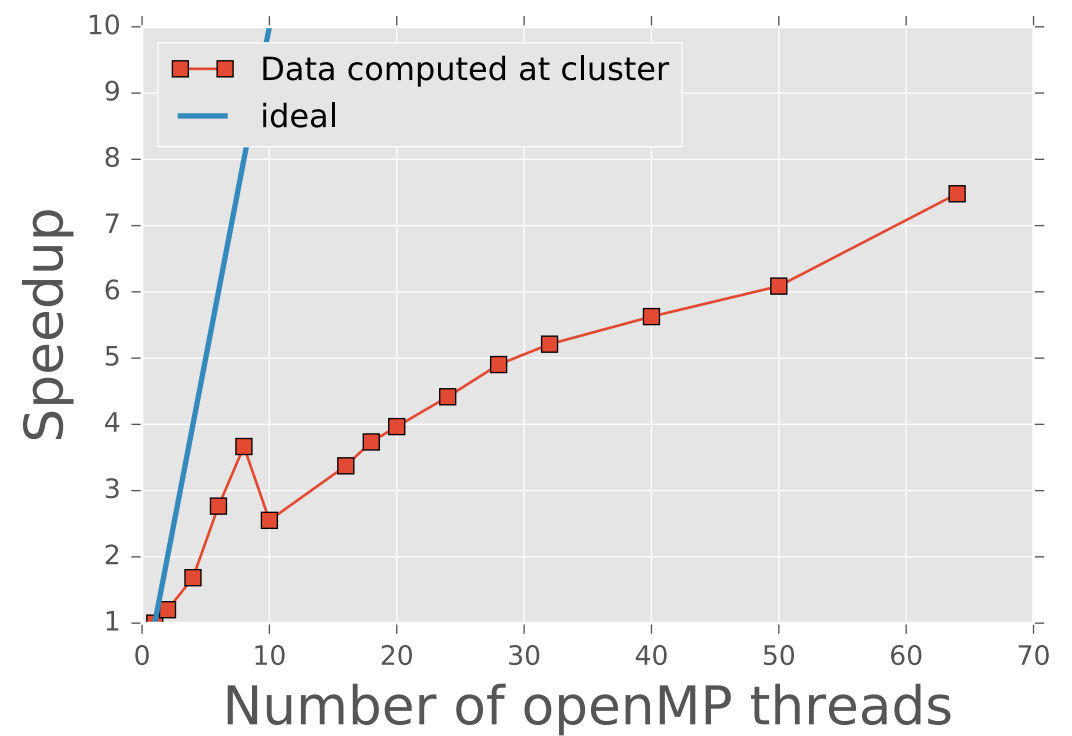
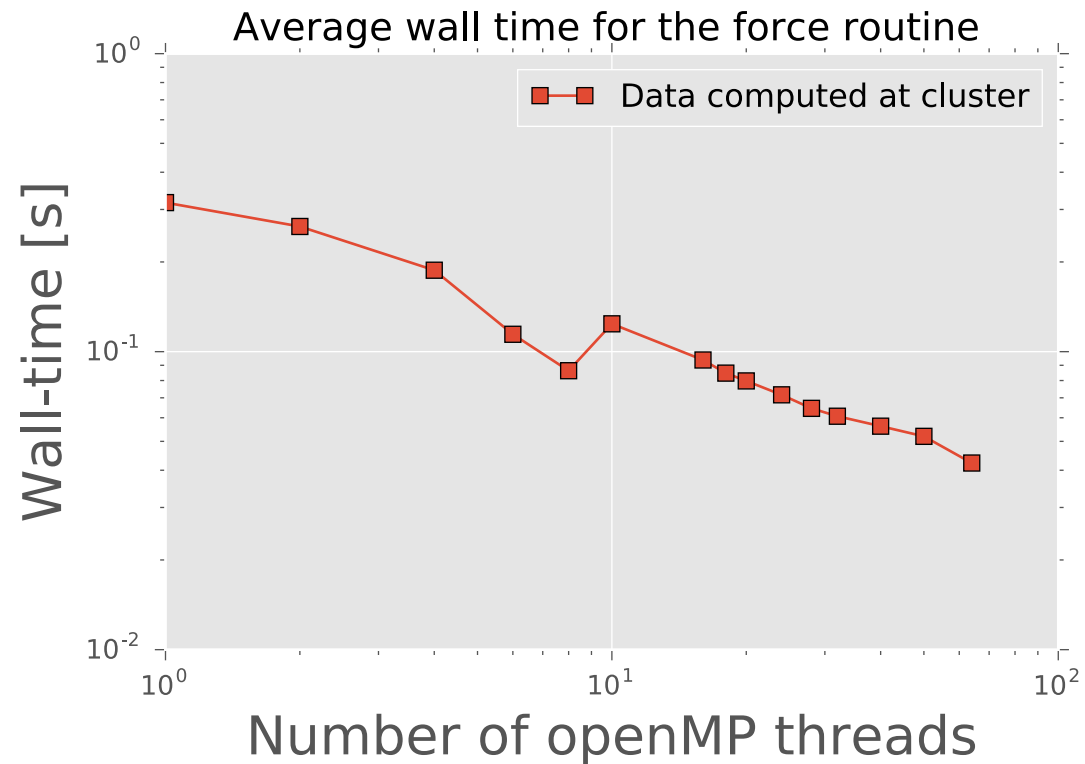
```
21   double dx, dy, rij, urij, delta, dVx, dVy, tmp, Fnx, Fny;
22   int ii, jj;
23 #pragma omp parallel for shared(grains, conf) private(ii, jj, dx, dy, rij, urij, delta, dVx, dVy, tmp, Fnx, Fny)
24   for (ii = 0; ii < ngrains; ++ii) {
25     for (jj = ii+1; jj < ngrains; ++jj) {
26       dx = grains[ii].R[0] - grains[jj].R[0];
27       //if (dx <= conf.LX/2) dx += conf.LX;
```

But time appears to be slower for more threads :(

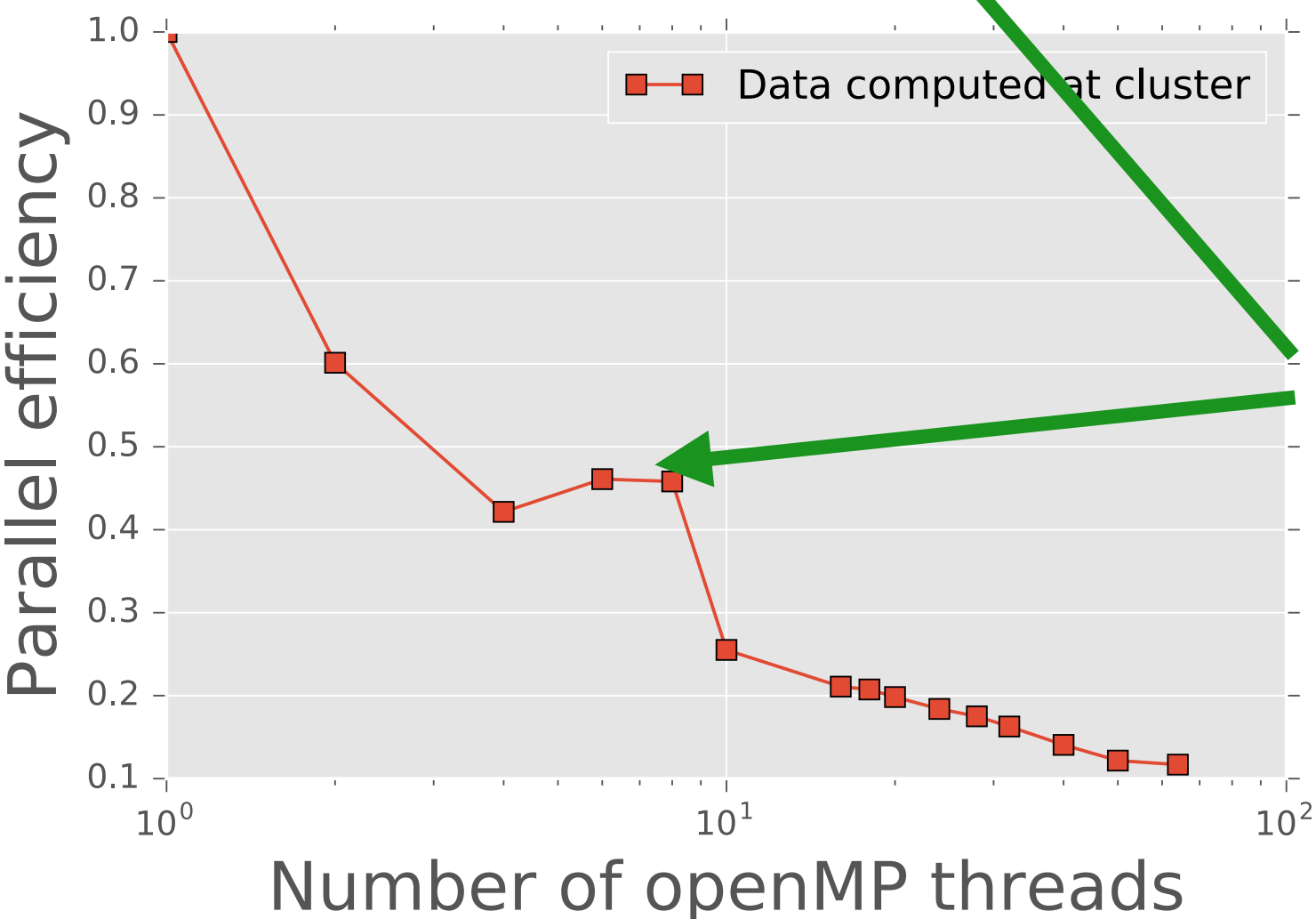
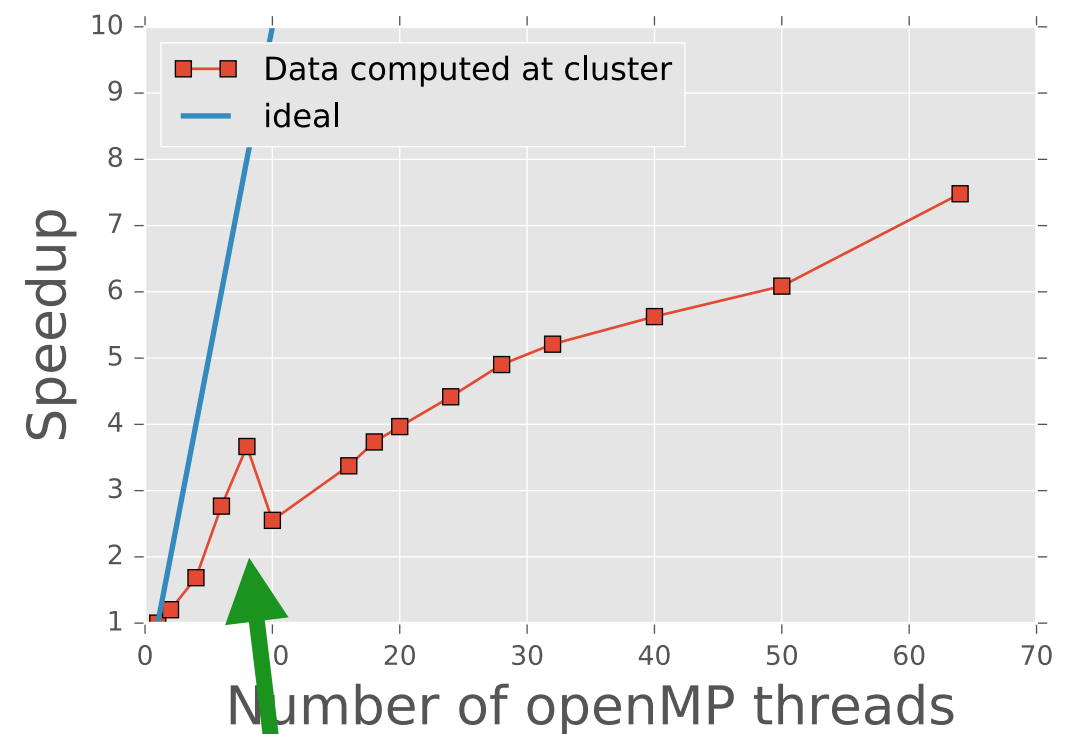
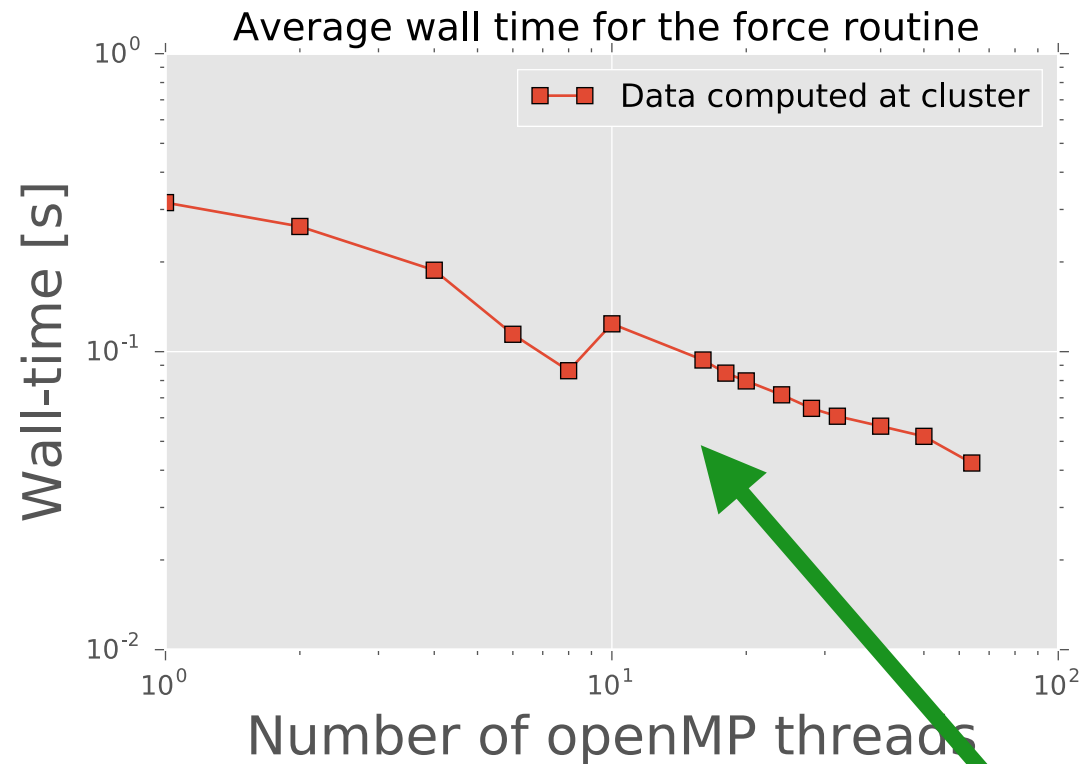
Be ambitious! Increase the system size (decreases the ratio of communication or thread creation/destruction to computation)!!!



Several runs

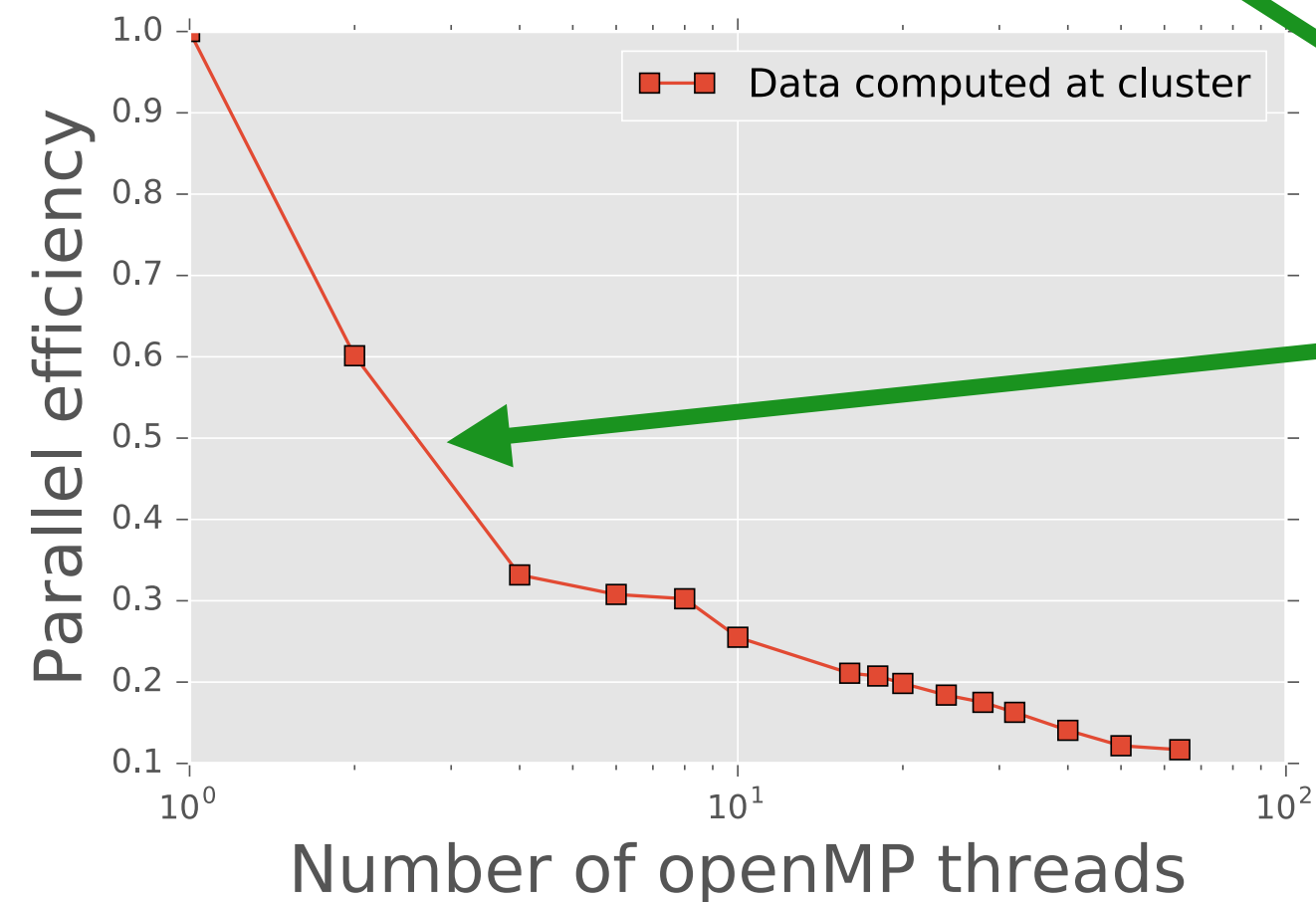
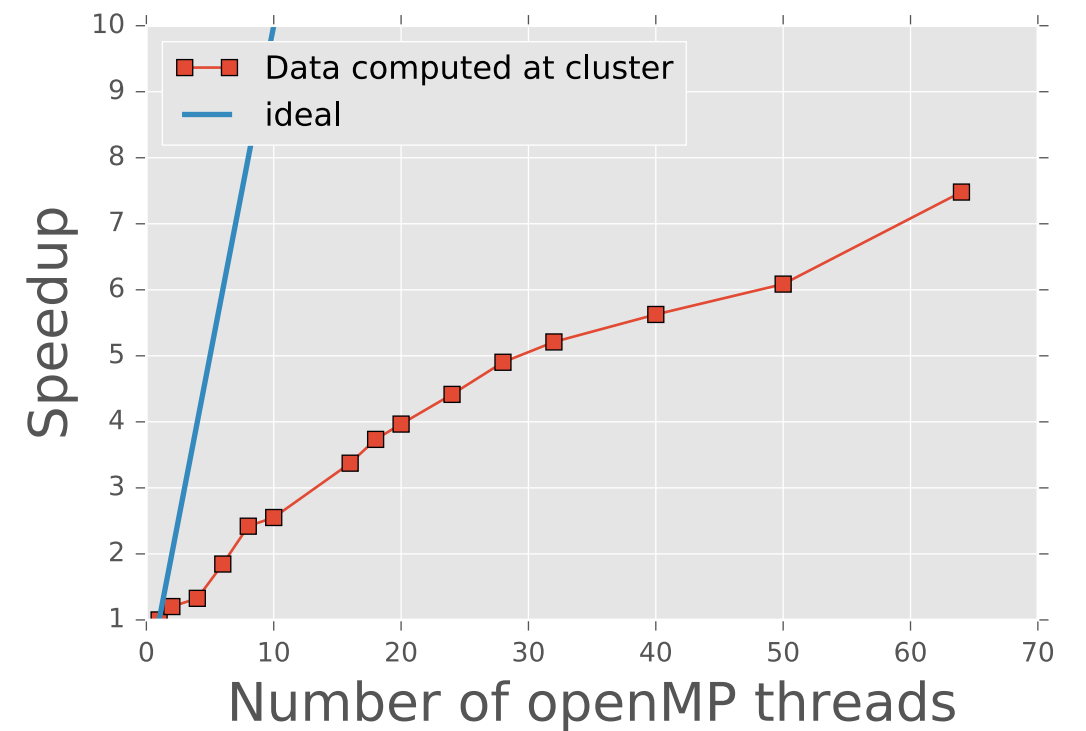
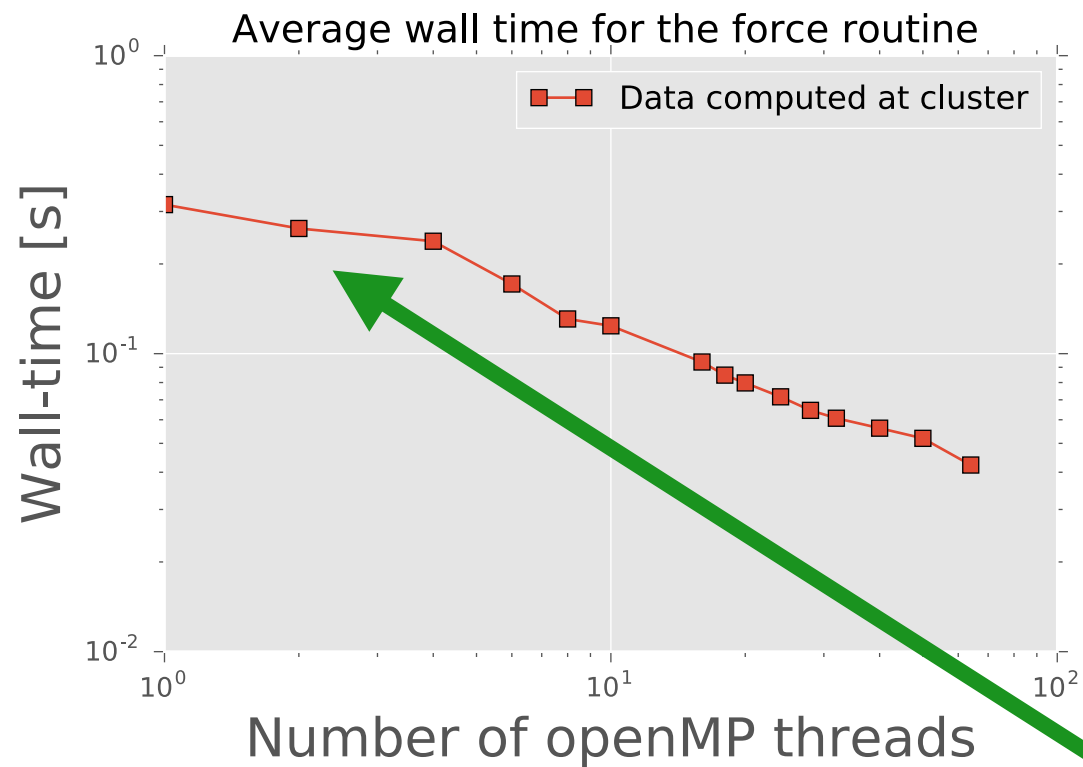


Several runs



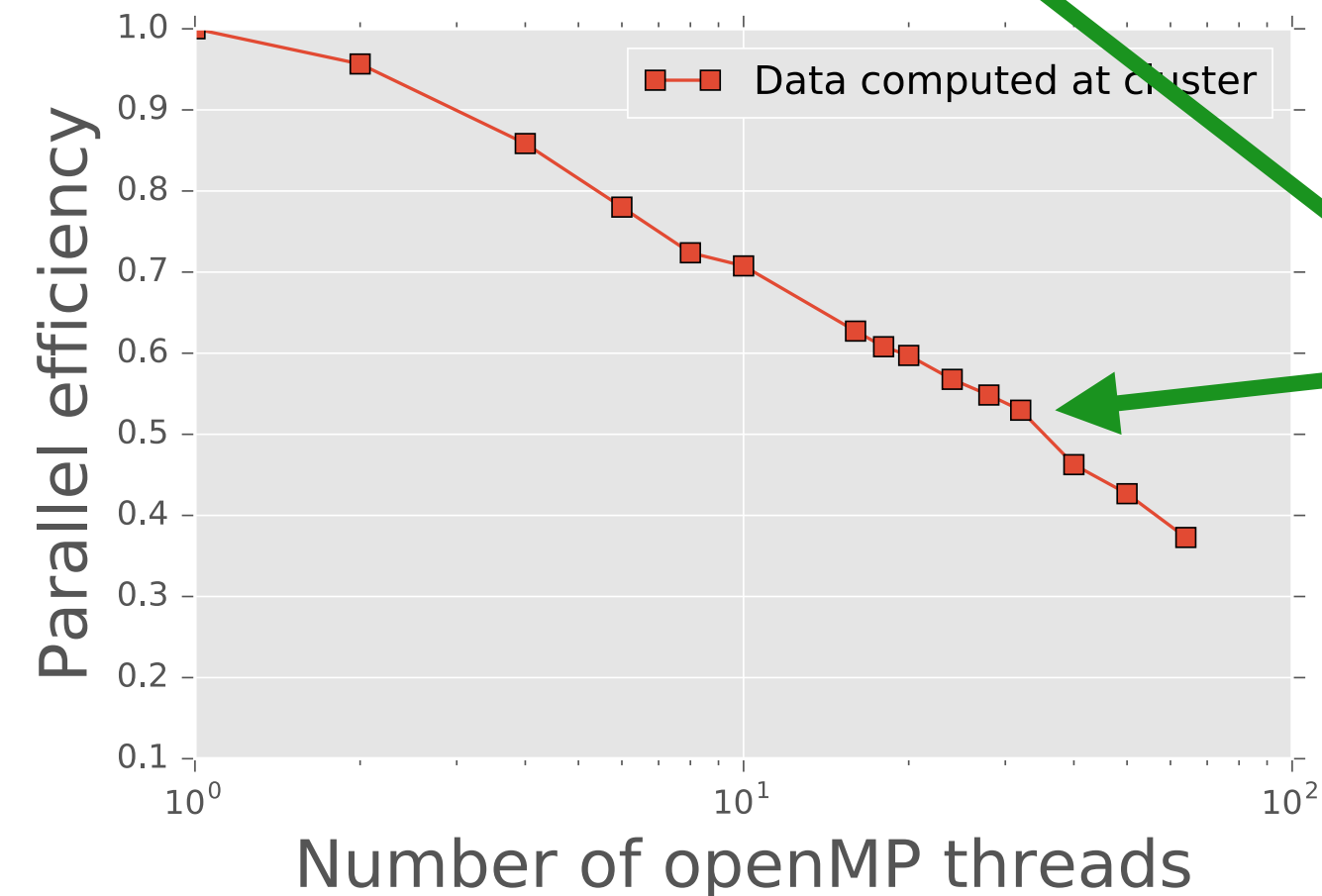
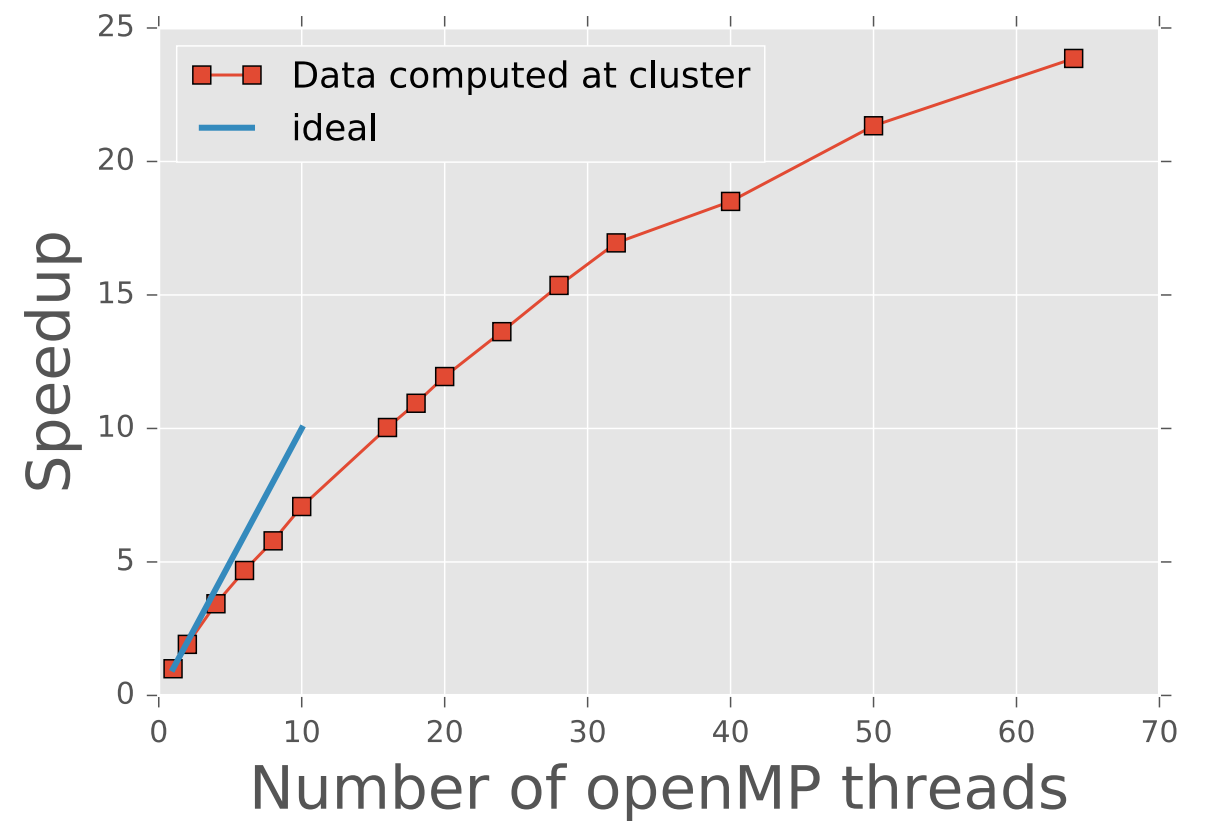
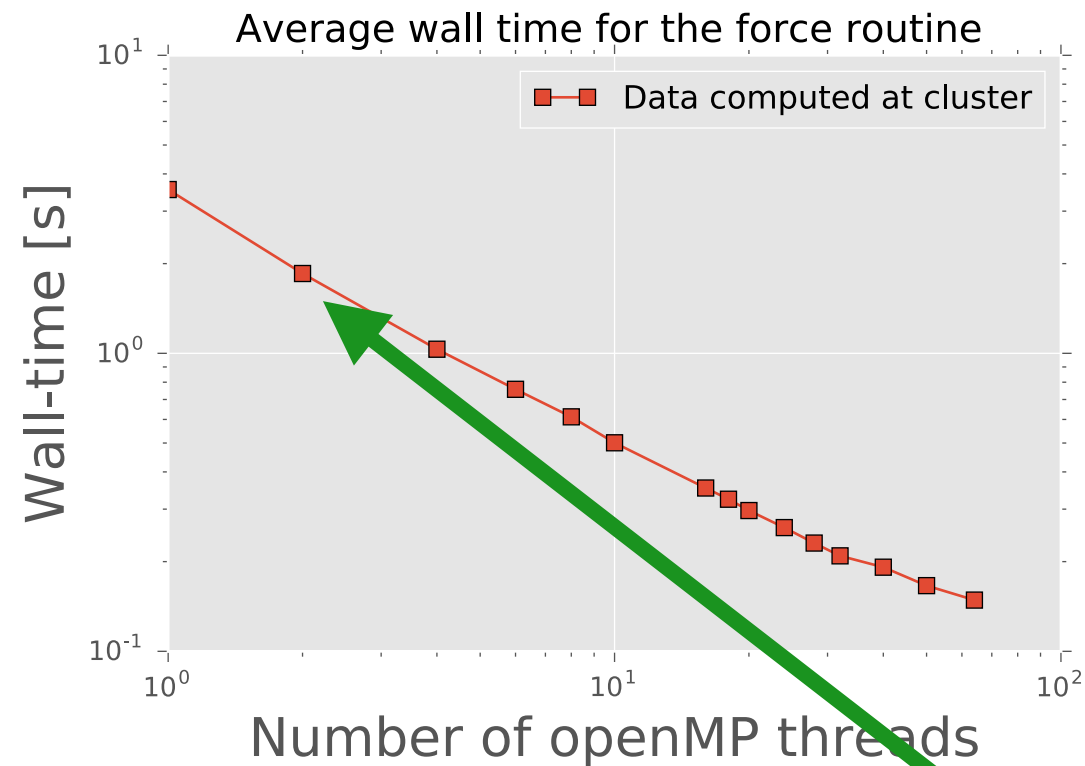
There are some outliers, I need to repeat the simulations

Several runs



The optimal number of cores seems to be 2!!! (wall time equal to 0.25s)

Competing threads because of 3th Newton law



The optimal number of cores seems to be 32!!! Maybe because the actual wall time increased by a factor of 10. (2 seconds)

Conclusions

- Always think if your system size is really testing the parallel approach.
- Be explicit about what is shared and private inside an omp parallel for .
- Always test and debug with the right tools.
- Compute the parallel efficiency to really get an idea of the ideal number of cores