Introduction

This page gives comparisons of CPU times of YADE on multicore computers, following recent launchpad discussions[1][2]. Most Yade users ask how many cores they should use or whether buying a new computer/workstation/server/... with a lot of cpu cores is reasonable.
Several performance tests were carried out. First Yades built-in performance test “--performance” was used. Furthermore Christian Jakob's setup for Yade<->PFC3D was used (consisting of a regular cubic assembly of particles and a slight angular planar wall underneath). Finally test were carried out using a more dynamic simulation which makes use of RotationEngine.

parameters that were tested

- internal graphics (Matrox G200) vs. external graphics (Nvidia Quadro 4000, 2GB DDR5)
- number of cores
- complexity of simulation
  - number of particles
  - movement

Hardware and Software

- 2xIntel Xeon E5-2687W @3.1GHz each got 8 physical cores using hyper-threading → 32 virtual cores
- 128 GB RAM
- 240 GB SSD on which yade-daily is install
- Ubuntu 12.04 LTS
- yade-daily build

Conclusion (draft)

Because of openMP implementation Yade should benefit from an increasing number of cpu cores. Suprisingly a benefit in terms of shorter simulation times is only achieved for a rather small number of cores (depending on simulation between 4 and 7).

“And in many cases it is enough, because more threads require more time for
"communication" and "synchronisation". You can try to use both openmp-
and mpi-based programs and choose the most suitable for you.” Anton Gladky

Bruno Chareyre (bruno-chareyre) said : #9

Thanks.
You could actually plot everything on the same graph more easily if y-axis was cycles*Nparticles / time.

A conclusion from these results seems to be that parallelism gives a 3x speedup, obtained with 3-4 cores, and there is no
point using more than 4 cores.
This is not really what I concluded from my recent tests, but again: different simulation => different conclusions.

A few things to keep in mind:
- The collider (contact detection) is the main non-parallel task.
- the collider takes a larger part of the total time for larger number of particles, and for more dynamic simulations
- BUT it takes less time if verletDist is increased, at the price of more virtual interactions

In my recent tests, the collider was taking about 1% of the total time (*), then it did not matter if the collider is parallel or not. If the collider takes more than that, then it can explain why you get the best speed with 3-4 cores when I get it with 8 cores.

In "--performance", the collider's cost goes from 1.8% (5k bodies) to 55% (200k bodies). This is partly because, the stats there include the cost of initializing the collider (cost of the first iteration in any simulation). Including this cost is not really correct: since the number of steps is varying as a function of Nparticles, the 1st iteration will take proportionaly more time with more particles but this is only because the total number of iterations is smaller, then the result can't be extrapolated in the form of an average time per step.

In the end, there is a clear answer to your question: no, --performance is not good at testing parallelism and/or hardware.

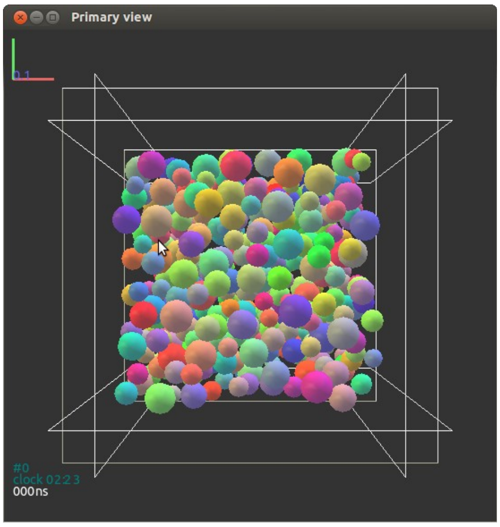
(*) This information is available in the "--performance" output, 2nd line in the table below. If you are currently running tests, it would be good to record such data as it gives a better understanding of how/why speed is affected by the different factors.

```
Name Count Time Rel. time
---------------------------------------------------------------------------------------------------
ForceResetter 12000 369078us 0.39%
InsertionSortCollider 337 1713474us 1.80%
InteractionLoop 12000 74036435us 77.56%
NewtonIntegrator 12000 19331902us 20.25%
TOTAL 95450891us 100.00%
```

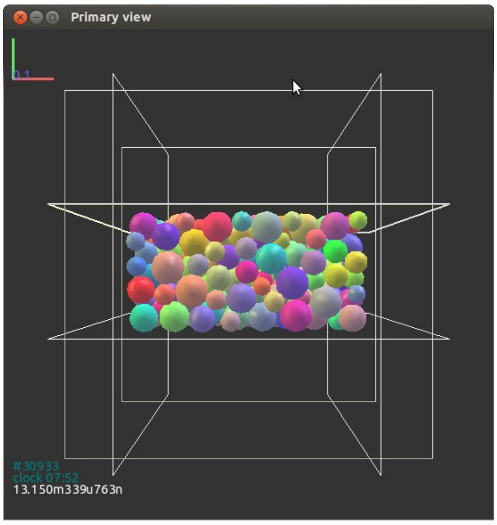
25091

Model setup
<u>Yade built-in performance</u>

Initial state



Steady state



Btw.: "--performance" was not created to test new systems, but to check
regressions after some commits. Nevertheless it can give a first impression of how your system performs. Here is a Bash-Script that may help using it.

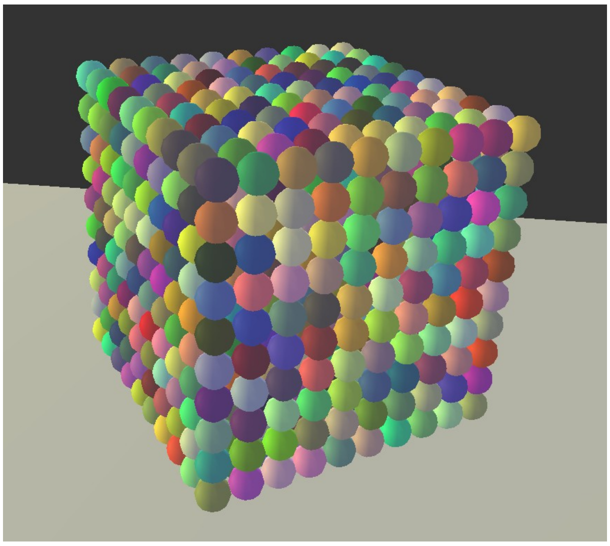
file:///home/dummy/Yade/Simulationen/PerformanceTests/YadePerformanceMonitoring
<code>

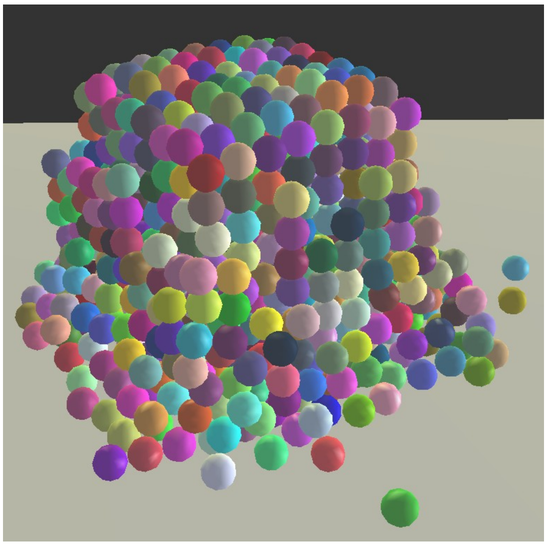
```
#!/bin/bash
#start script via "bash yadeBuiltInPerformanceTest.sh MIN MAX >>
```

```
~/yadeBuiltInPerformance.log" starten
# MIN: number of cores to start with
# MAX: maximum number of cores to test
echo "Beginning Yade Performance Test"

BEGIN=$1
END=$2

for (( I=$BEGIN; $I <= $END; I++ )); do
        DATE=$(date)
        echo $I" core(s); beginning: "$DATE
        yade-daily -j$I --performance >> ~/yadelogBuiltInPerformance_j$I.log
done

DATE=$(date)
echo "Ende: "$DATE
</code>
```

Yade<->PFC3D

The model for determining calculation speed consists of a regular cubic assembly of particles and a slight angular planar wall underneath.





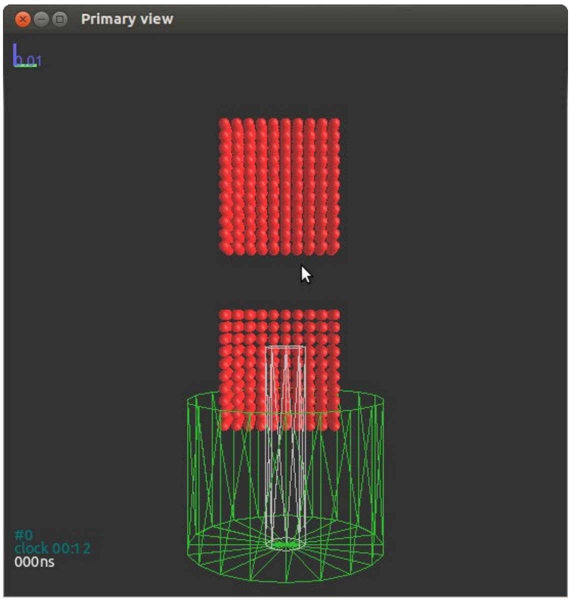
A linear contact model (without viscous damping) with stiffness of <math>10^6</math> for the particles and <math>10^8</math> for the walls was chosen. The friction angle is 26,6&deg; (= friction coefficient 0,5) for the wall and the particles. The density of the particles was set to 1000 [kg/m<math>^3</math>]. Gravity acts in negative z-direction with 9,81 [m/s<math>^2</math>]. A local damping constant of 0,7 was set. The time step was kept constant to <math>10^{-3}</math> [s] during calculations.

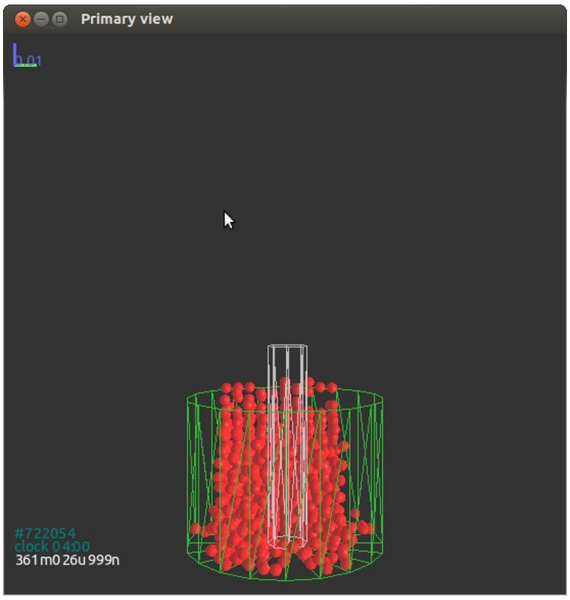
more complex simulations

tub with two packs of spheres falling inside and facet cylinder rotating

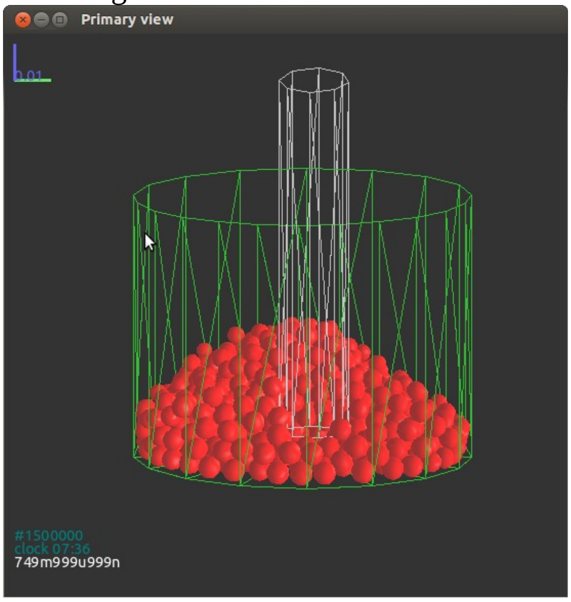
angularVelocityOfCylinder=2*pi/49.2

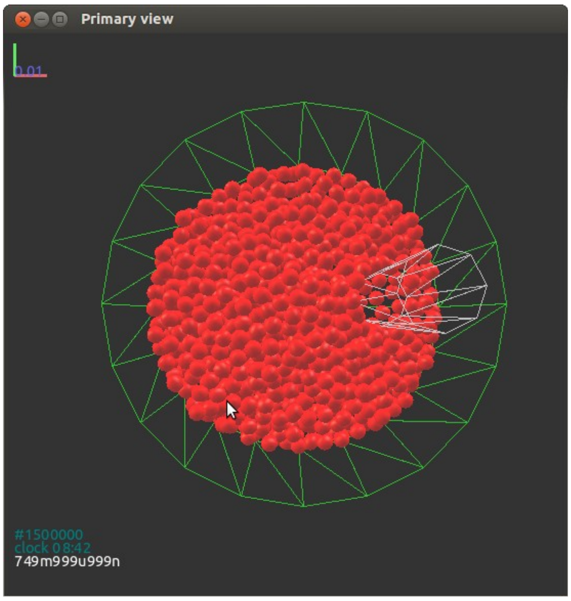
stage 1: falling



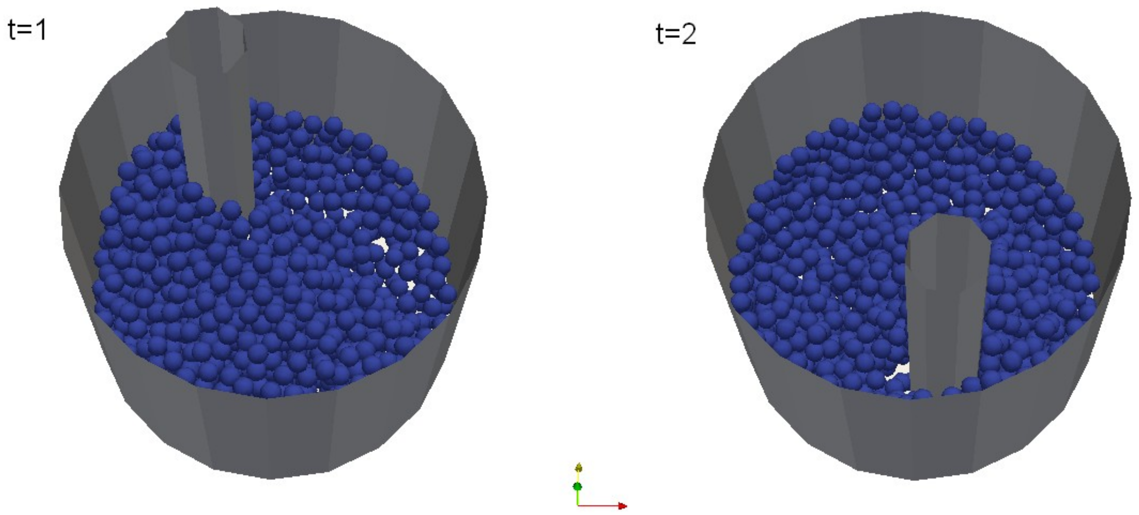


stage 2: stirring

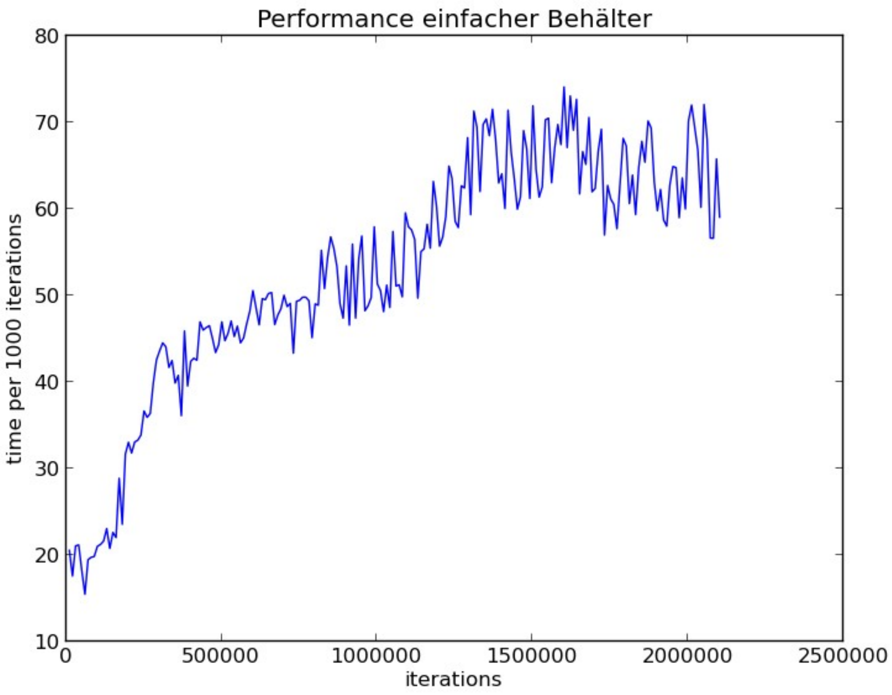




tub filled with 1176 spheres (r=3mm). A facet cylinder is rotating inside with different angular velocities



cylindrical container
height of cylinder = 0.5m
radius of cylinder = 0.275m
number of spheres = 300163 with r=0.003mm



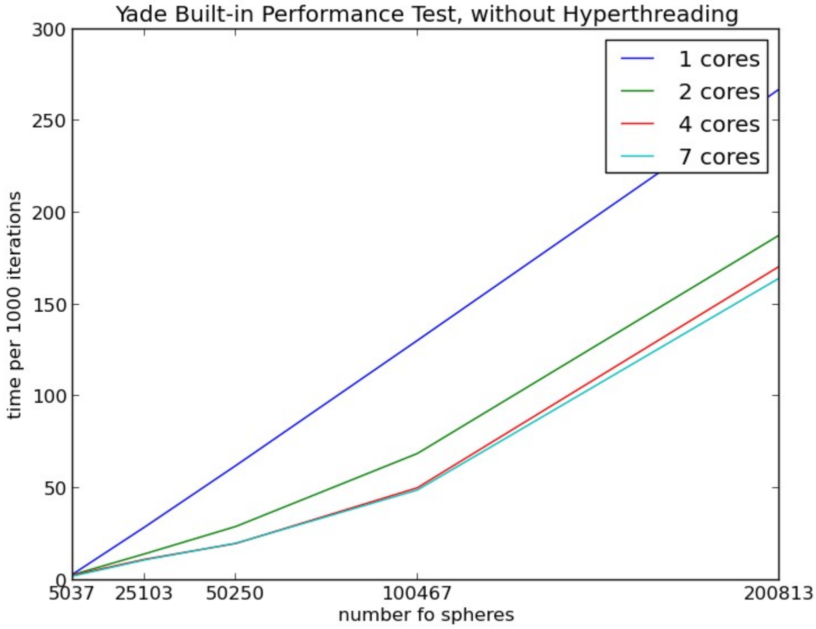
Results

Yade Built-in Performance Test

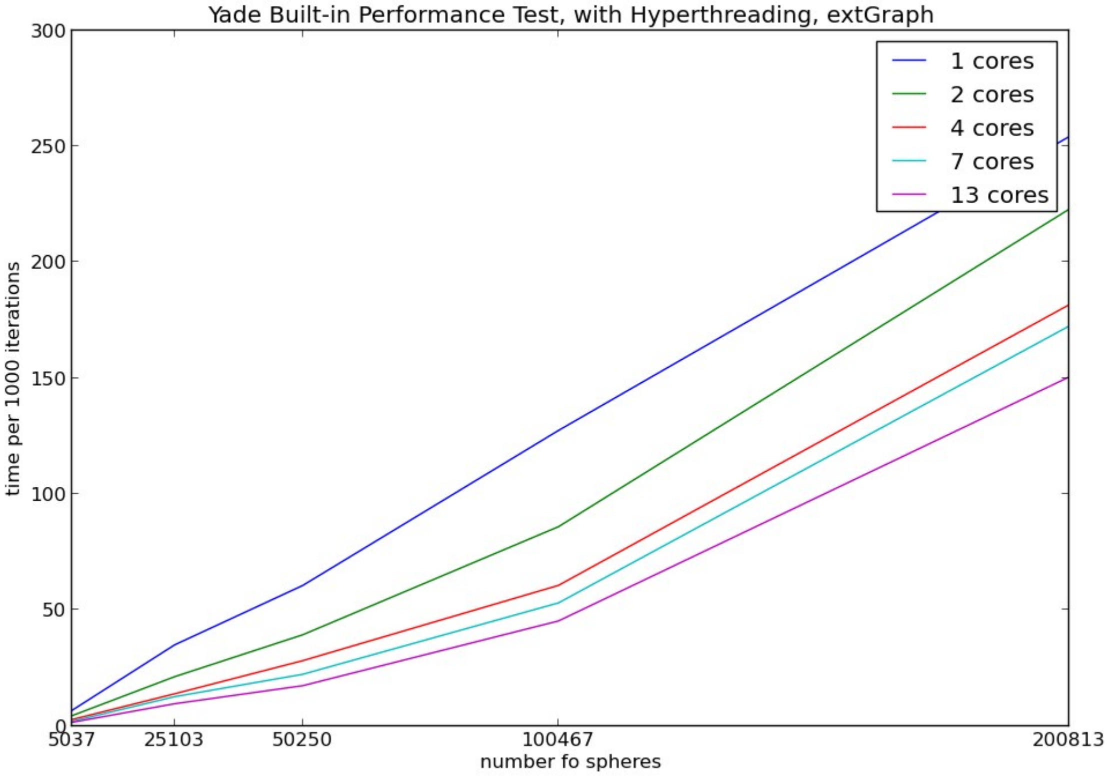
cores\particles	5037	25103	50250	100467	200813
1	3.18	29.07	62.44	130.52	267.62
2	2.93	14.38	29.24	68.96	188.08
3	2.72	11.53	22.58	54.73	172.27
4	2.71	11.40	19.97	50.28	171.06
5	2.46	11.39	20.02	51.11	167.02
6	2.47	**10.64**	19.84	49.17	170.85
7	2.21	11.09	20.11	49.09	**164.64**
8	2.31	11.07	19.61	50.82	169.15
9	2.13	11.92	21.16	**48.78**	170.77
10	2.09	12.37	**18.29**	48.82	171.75
11	**1.94**	11.92	21.58	50.50	172.94
12	2.27	12.16	20.64	51.07	169.67
13	2.22	12.08	20.86	52.84	169.88
14	2.00	12.27	20.55	54.29	176.12

15	2.10	12.86	21.63	53.83	175.78
16	2.72	15.58	24.45	58.83	182.39

Plot für ausgewählte Kerne



with hyperthreading:



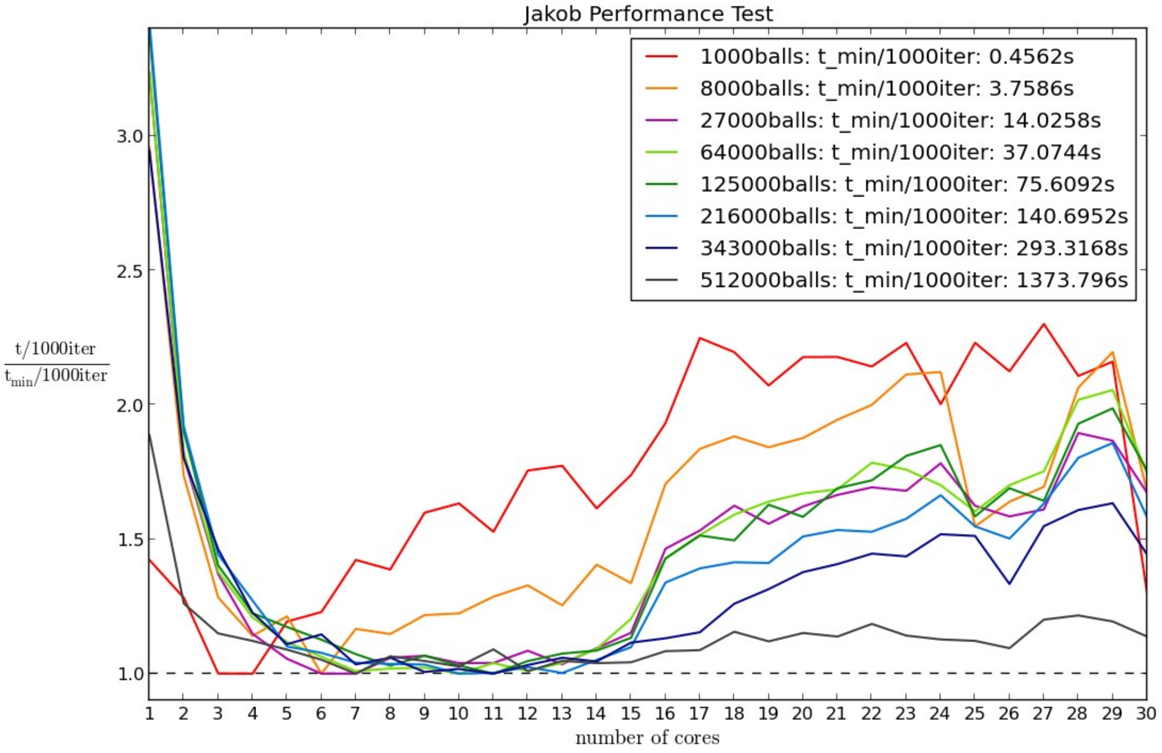
cores\particles	5037	25103	50250	100467	200813

1	6.85	34.99	60.6	127.47	254.31
2	4.52	21.27	39.35	85.96	223.03
3	2.94	16.17	28.13	66.61	180.62
4	2.84	13.89	28.18	60.65	181.81
5	2.26	12.36	24.47	56.48	179.8
6	2.06	11.4	25.02	54.19	168.68
7	2.07	12.66	22.31	53.11	172.6
8	2.12	11.66	23.08	54.71	171.21
9	2.17	12.31	23.57	52.49	169.13
10	2.2	12.04	21.36	54.73	174.62
11	2.31	12.01	22.51	55.34	174.62
12	2.27	12.06	22.42	54.67	170.48
13	1.63	**9.62**	**17.39**	**45.32**	150.63
14	**1.56**	10.62	18.88	48.12	151.11
15	1.68	10.62	19	49	**146.04**
16	2.08	13.54	22.98	47.68	156.61
17	2.39	13.79	23.77	50.01	158.21
18	2.36	14.34	23.89	50.91	152.97
19	2.27	13.89	22.18	50.22	154.7
20	2.38	15.44	23.74	52.58	158.46
21	2.76	14.96	24.9	53.47	154.33

Jakob

The speed test was done with 10x10x10 = 1000 spheres, 20x20x20 = 8000 spheres, ... , 60x60x60 = 216000 spheres. All calculations were performed 20 times with 1000 steps. The average values of the calculation times (in seconds) and the calculation speed (in steps/second) are shown in the following table. Also the relative differences of the calculation times are included (positive, when YADE needed less calc. times).

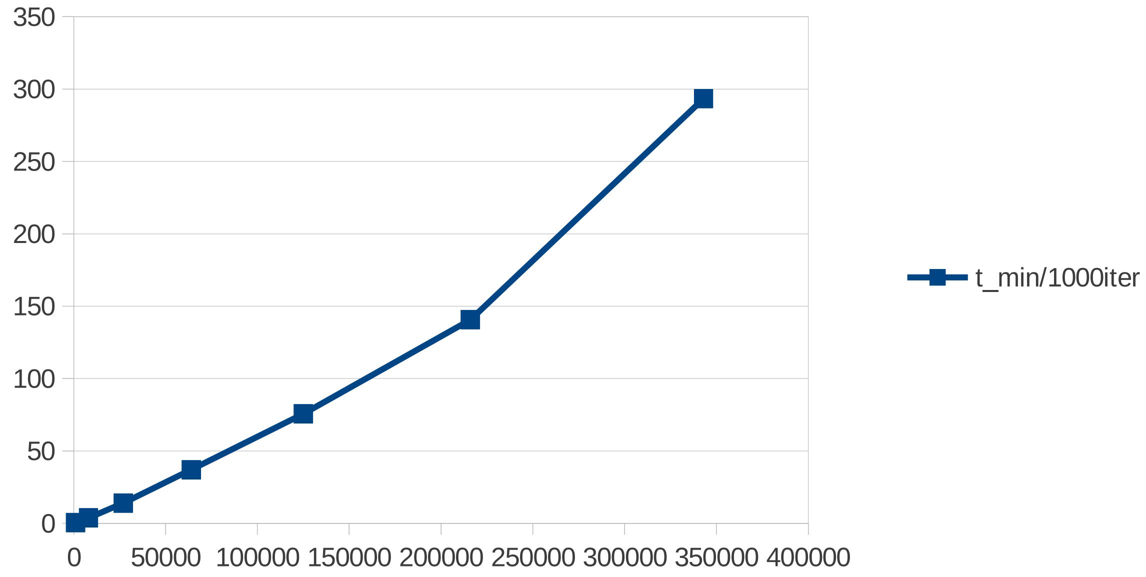
relative times per 1000 iterations:



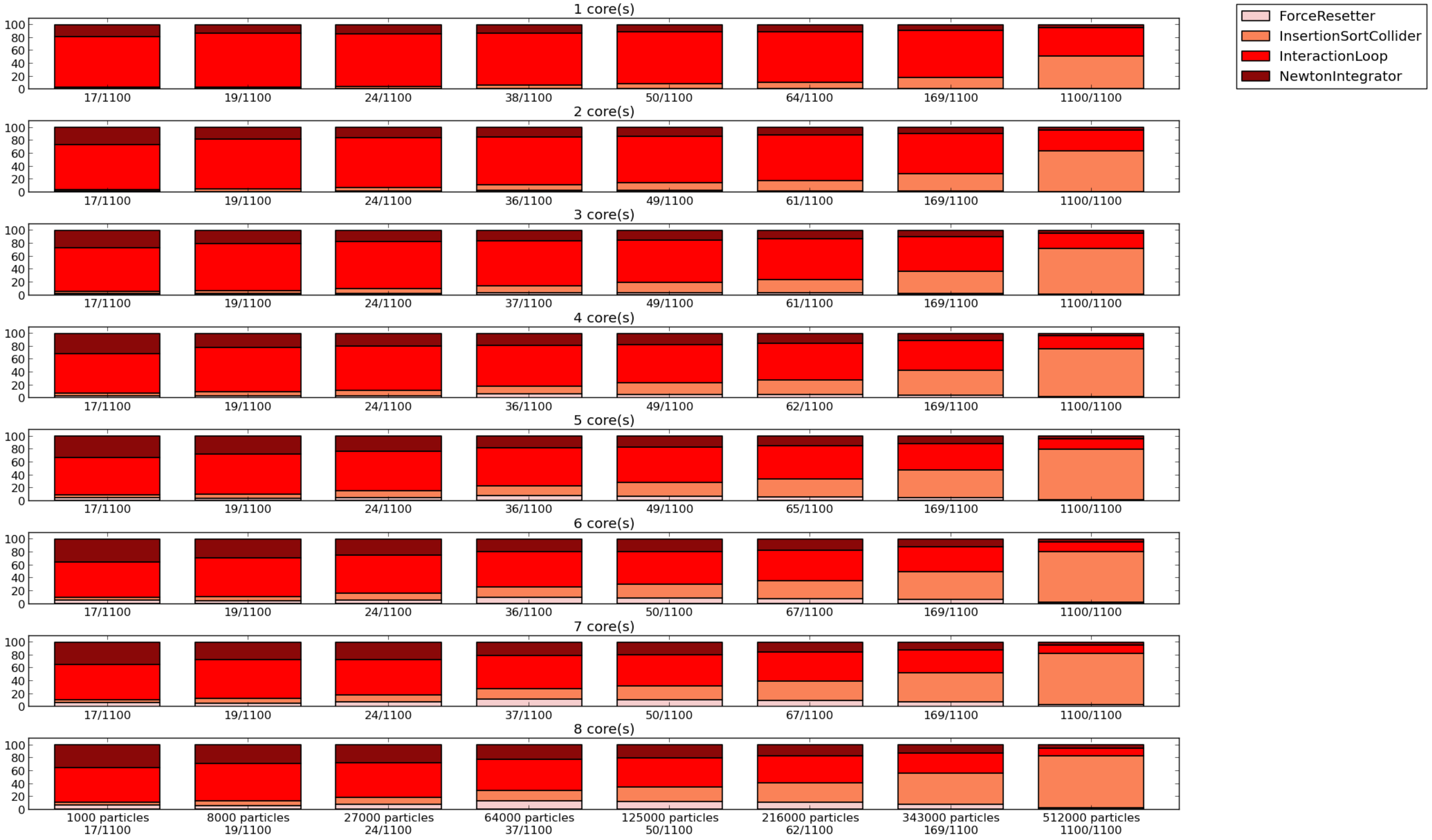
core\number of particles	1000	8000	27000	64000	125000	216000	343000	512000
1	1.42	2.96	3.24	3.24	3.39	3.43	2.95	1.89
2	1.28	1.74	1.81	1.82	1.9	1.92	1.8	1.26
3	**1**	1.28	1.37	1.38	1.4	1.44	1.46	1.15
4	1	1.14	1.15	1.21	1.22	1.27	1.23	1.12
5	1.19	1.21	1.06	1.12	1.17	1.1	1.11	1.09
6	1.23	**1**	**1**	1.06	1.13	1.08	1.15	1.05
7	*1.42*	*1.17*	*1*	*1.01*	*1.07*	*1.04*	*1.03*	***1***
8	1.39	1.15	1.06	1.02	1.03	1.04	1.06	1.06
9	1.6	1.22	1.07	1.02	1.07	1.03	1.01	1.05
10	1.63	1.22	1.04	**1**	1.03	**1**	1.02	1.03
11	1.53	1.29	1.04	1.04	**1**	1	**1**	1.09
12	1.76	1.33	1.09	1.01	1.05	1.02	1.03	1.01
13	1.77	1.25	1.04	1.04	1.07	1	1.06	1.05
14	1.62	1.41	1.09	1.09	1.09	1.05	1.05	1.04
15	1.74	1.34	1.15	1.2	1.13	1.1	1.12	1.04
16	1.93	1.71	1.46	1.43	1.43	1.34	1.13	1.08
17	2.25	1.84	1.53	1.52	1.51	1.39	1.15	1.09

18	2.2	1.88	1.63	1.59	1.5	1.41	1.26	1.16
19	2.07	1.84	1.56	1.64	1.63	1.41	1.31	1.12
20	2.18	1.88	1.62	1.67	1.58	1.51	1.38	1.15
21	2.18	1.94	1.67	1.69	1.69	1.53	1.41	1.14
22	2.14	2	1.69	1.79	1.72	1.53	1.45	1.19
23	2.23	2.11	1.68	1.76	1.81	1.58	1.44	1.14
24	2	2.12	1.78	1.7	1.85	1.66	1.52	1.13
25	2.23	1.55	1.62	1.6	1.59	1.55	1.51	1.12
26	2.12	1.64	1.59	1.7	1.69	1.5	1.33	1.09
27	2.3	1.7	1.61	1.75	1.64	1.63	1.55	1.2
28	2.11	2.06	1.9	2.02	1.93	1.8	1.61	1.22
29	2.16	2.2	1.87	2.06	1.99	1.86	1.63	1.19
30	1.3	1.68	1.67	1.74	1.75	1.58	1.44	1.14

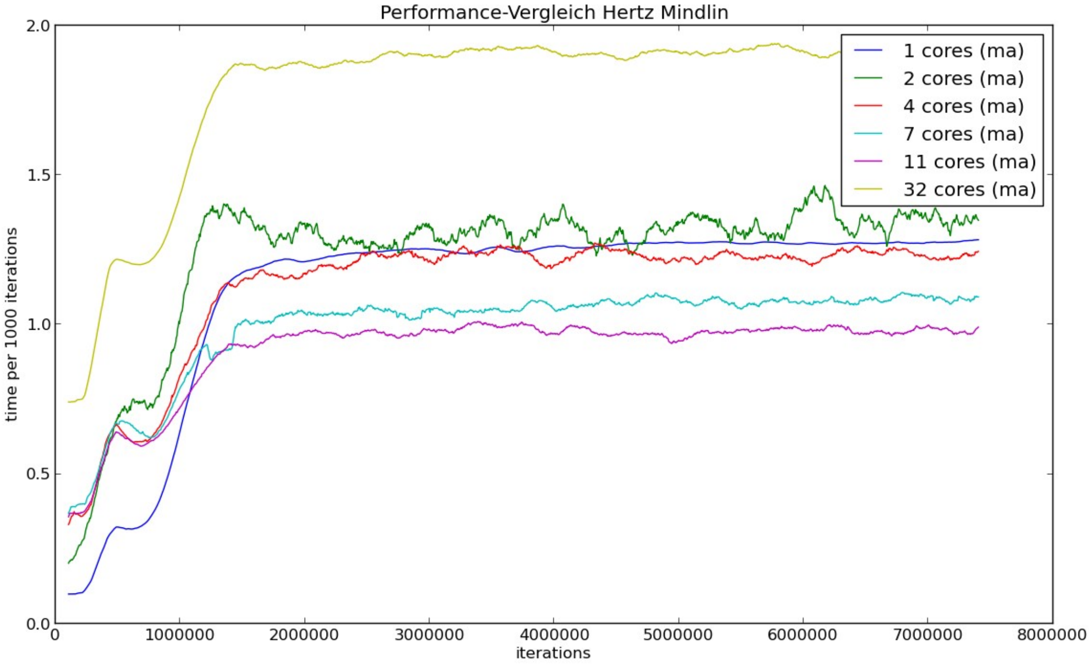
absolute times per 1000 iterations:



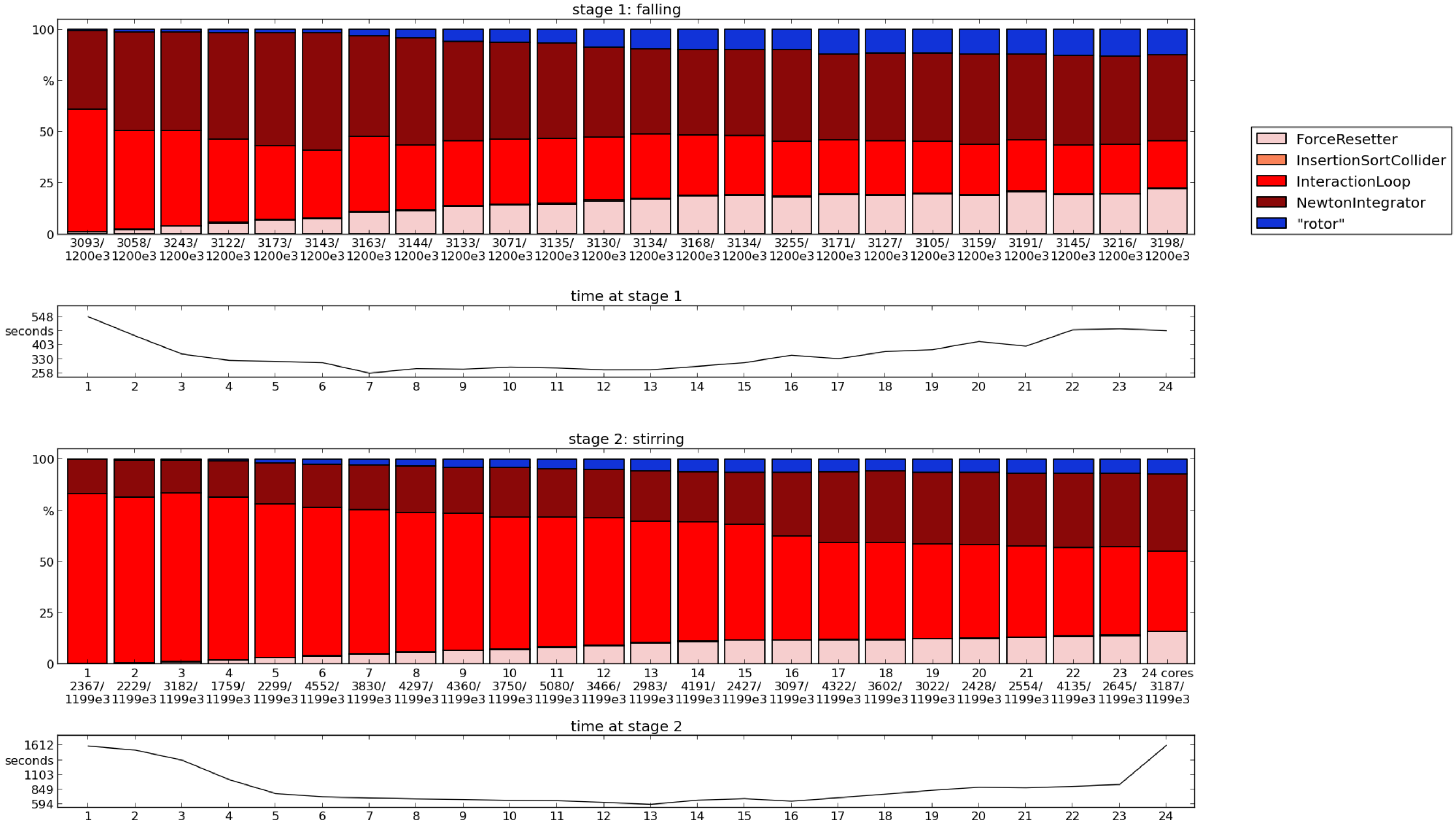
particles	1000	8000	27000	64000	125000	216000	343000	512000
t_min/1000iter [s]	0.46	3.76	14.03	37.07	75.61	140.7	293.32	1373.8
number of cores	3	6	6	10	11	10	11	7


1 core(s)
17/1100
19/1100
24/1100
38/1100
50/1100
64/1100
169/1100
1100/1100
2 core(s)
17/1100
19/1100
24/1100
36/1100
49/1100
61/1100
169/1100
1100/1100
3 core(s)
17/1100
19/1100
24/1100
37/1100
49/1100
61/1100
169/1100
1100/1100
4 core(s)
17/1100
19/1100
24/1100
36/1100
49/1100
62/1100
169/1100
1100/1100
5 core(s)
17/1100
19/1100
24/1100
36/1100
49/1100
65/1100
169/1100
1100/1100
6 core(s)
17/1100
19/1100
24/1100
36/1100
50/1100
67/1100
169/1100
1100/1100
7 core(s)
17/1100
19/1100
24/1100
37/1100
50/1100
67/1100
169/1100
1100/1100
8 core(s)
1000 particles 17/1100
8000 particles 19/1100
27000 particles 24/1100
64000 particles 37/1100
125000 particles 50/1100
216000 particles 62/1100
343000 particles 169/1100
512000 particles 1100/1100
0
20
40
60
80
100
ForceResetter
InsertionSortCollider
InteractionLoop
NewtonIntegrator

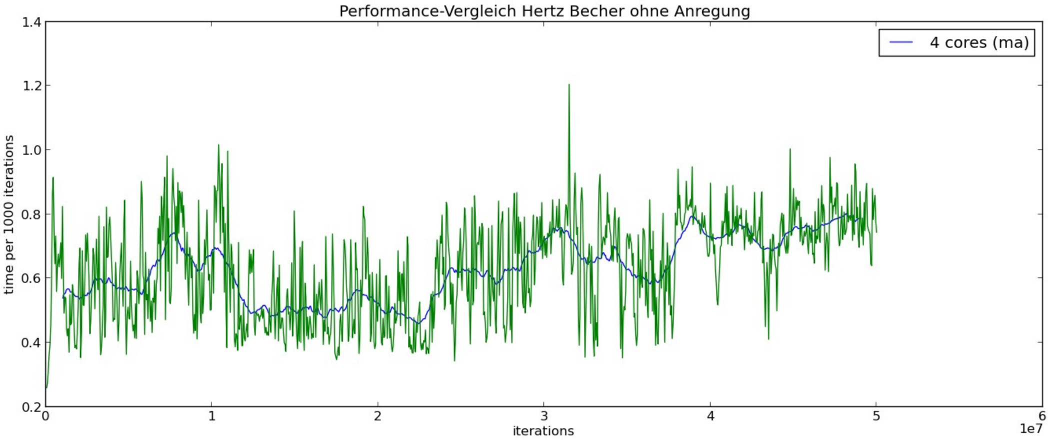
HertzMindlin
tub with two packs of spheres falling inside and facet cylinder rotating



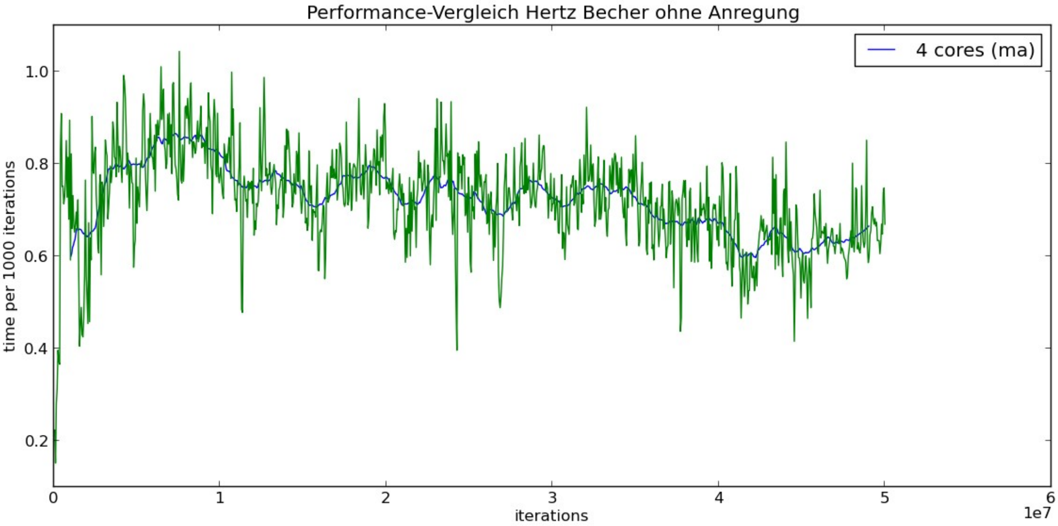
stage 1 and stage 2


stage 1: falling
100
%
50
25
0
3093/ 1200e3
3058/ 1200e3
3243/ 1200e3
3122/ 1200e3
3173/ 1200e3
3143/ 1200e3
3163/ 1200e3
3144/ 1200e3
3133/ 1200e3
3071/ 1200e3
3135/ 1200e3
3130/ 1200e3
3134/ 1200e3
3168/ 1200e3
3134/ 1200e3
3255/ 1200e3
3171/ 1200e3
3127/ 1200e3
3105/ 1200e3
3159/ 1200e3
3191/ 1200e3
3145/ 1200e3
3216/ 1200e3
3198/ 1200e3
ForceResetter
InsertionSortCollider
InteractionLoop
NewtonIntegrator
"rotor"
time at stage 1
548
seconds
403
330
258
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
stage 2: stirring
100
%
50
25
0
1 2367/ 1199e3
2 2229/ 1199e3
3 3182/ 1199e3
4 1759/ 1199e3
5 2299/ 1199e3
6 4552/ 1199e3
7 3830/ 1199e3
8 4297/ 1199e3
9 4360/ 1199e3
10 3750/ 1199e3
11 5080/ 1199e3
12 3466/ 1199e3
13 2983/ 1199e3
14 4191/ 1199e3
15 2427/ 1199e3
16 3097/ 1199e3
17 4322/ 1199e3
18 3602/ 1199e3
19 3022/ 1199e3
20 2428/ 1199e3
21 2554/ 1199e3
22 4135/ 1199e3
23 2645/ 1199e3
24 cores 3187/ 1199e3
time at stage 2
1612
seconds
1103
849
594
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

tub
A) low angularVelocity 2*pi/49.2=0.127; appr. 10e7 iterations per revolution



B) high angularVelocity 10*2*pi/49.2=1.27; appr. 1e7 iterations per revolution



Useful scripts

- extractResults from Logfiles
  - YadePerformance
    - timing
    - score
  - Jakob
  - Timing