COMP464 - Homework 2 – Report

Bruno G M Correa – bcorrea@luc.edu (brunogmc@gmail.com)

There are two main goals that we may be pursuing when we try to parallelize code. In one of them we try to reduce the amount of time taken to run a program. This is called strong scaling. In the other one we try to maintain constant the amount of time taken to run a program by proportionally increasing the number of processors (or threads). This is called weak scaling.

Considering the information gathered during many executions of the N-body problem, I think we can consider that it has a really good strong parallel scalability. 98% of efficiency could be achieved when running the program with 10000 bodies and scaling up the number of threads from 1 to 8. As we can see in the table, the efficiency drops when we try to use more than 8 threads.

We can notice from the charts that when we have a small number of bodies, the process of parallelizing the code does not bring us a better efficiency. Instead, the run-time gets worse.

In the case of weak scaling efficiency, the results are not good if we double the number of bodies when doubling the number of threads. This happens because the serial run-time for the N-body algorithm used in this benchmark scales as n2 (where n is the number of bodies). This means that doubling the number of bodies without increasing the number of threads should lead to a four times slower run-time. So, to achieve a constant run-time when doubling the number of threads we should increase the number of bodies by a factor of sqrt(# of threads). The difference between the efficiency of these two cases can be viewed in the charts. As expected, the weak efficiency is much better when we use the square root factor to define the number of bodies to be used.

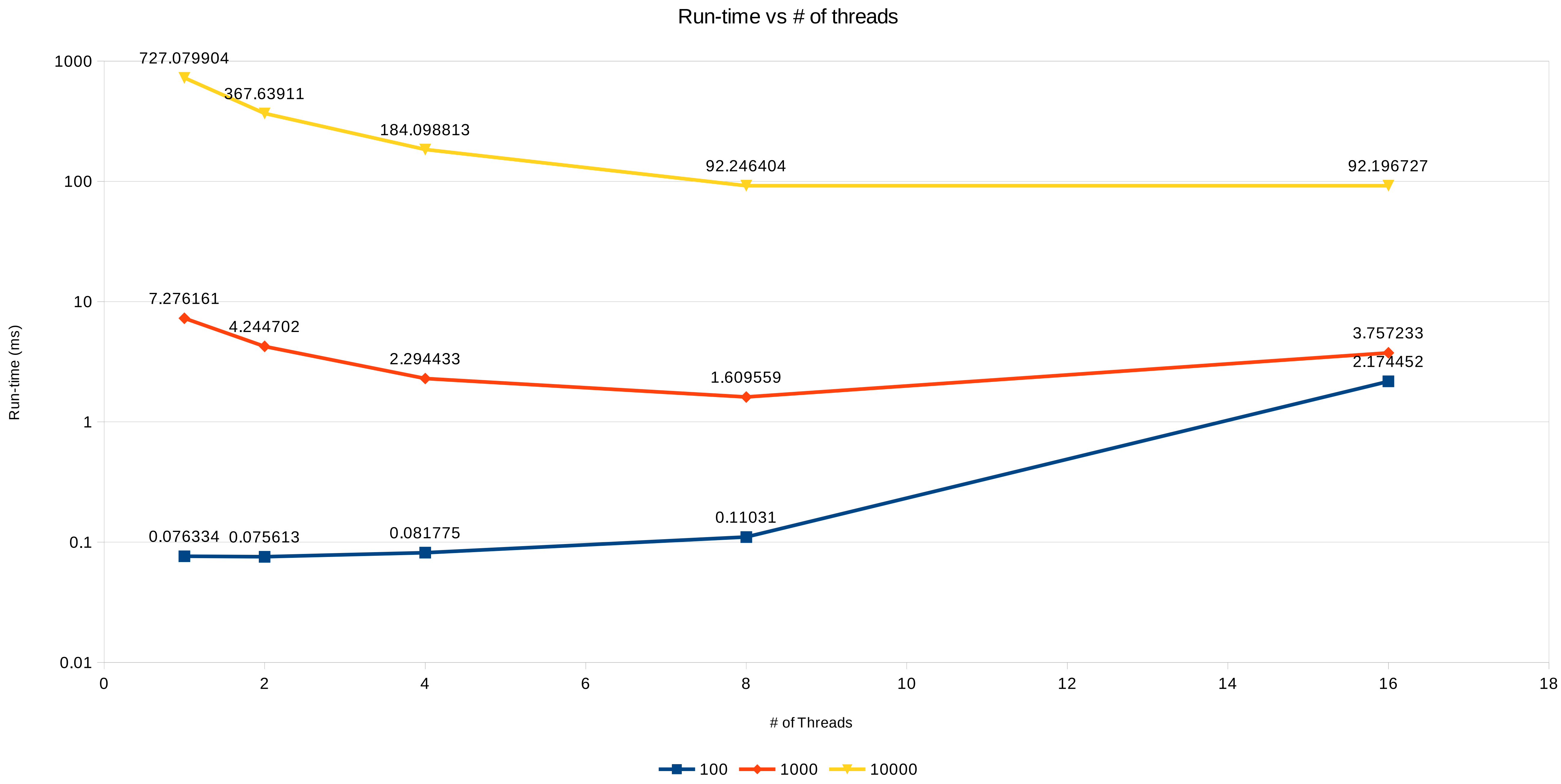
Summarizing, we could say that this algorithm scales well but it is dependent upon the number of threads and bodies. There is a strange behavior that drops the efficiency when we use more than 8 threads in the Stampede machine and I was not able to discover what lead us to this behavior, though.

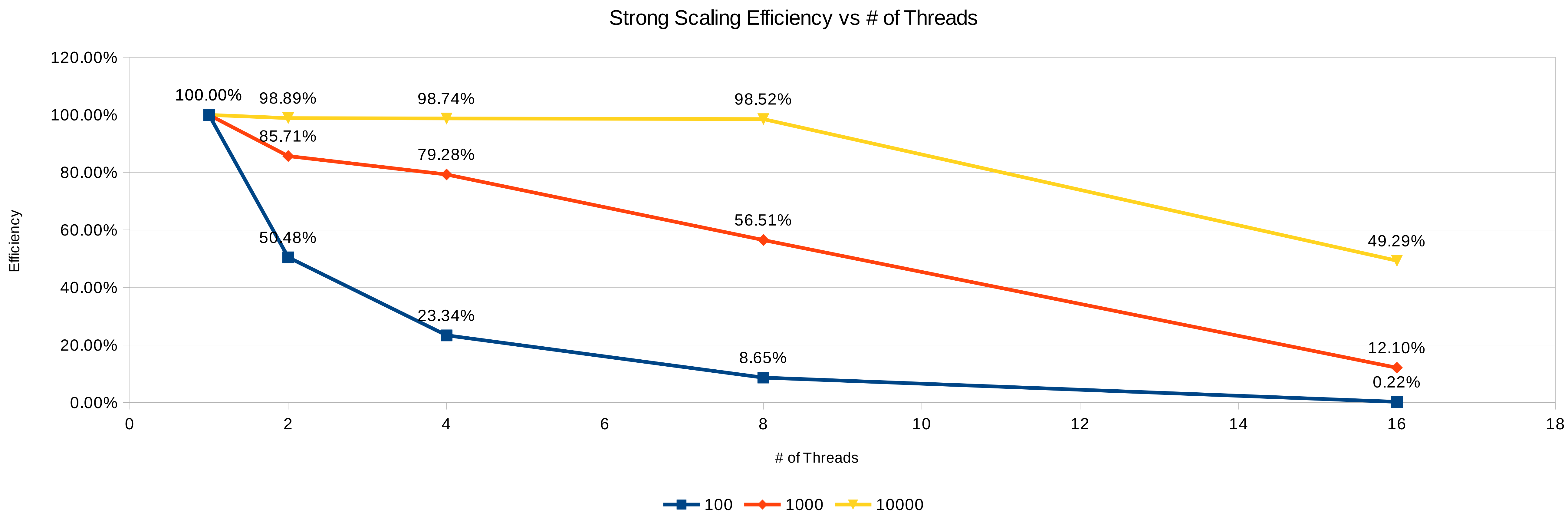
Sources:
Class resources
https://www.cs.rit.edu/~ark/bcbd/ch09.pdf pg. 9-2
https://www.cs.rit.edu/~ark/bcbd/ch10.pdf pg. 10-2
https://www.sharcnet.ca/help/index.php/Measuring_Parallel_Scaling_Performance

Strong Scaling Benchmark					
Type	Bodies	Threads	avg(time)/step (ms)	Speedup	Strong Scaling Efficiency
Serial	100	1	0.076334	1	100.00%
strong/Parallel	100	2	0.075613	1.009535	50.48%
strong/Parallel	100	4	0.081775	0.933464	23.34%
strong/Parallel	100	8	0.11031	0.691995	8.65%
strong/Parallel	100	16	2.174452	0.035105	0.22%
Serial	1000	1	7.276161	1	100.00%
strong/Parallel	1000	2	4.244702	1.714175	85.71%
strong/Parallel	1000	4	2.294433	3.171224	79.28%
strong/Parallel	1000	8	1.609559	4.520593	56.51%
strong/Parallel	1000	16	3.757233	1.936574	12.10%
Serial	10000	1	727.079904	1	100.00%
strong/Parallel	10000	2	367.63911	1.9777	98.89%
strong/Parallel	10000	4	184.098813	3.9494	98.74%
strong/Parallel	10000	8	92.246404	7.881932	98.52%
strong/Parallel	10000	16	92.196727	7.886179	49.29%

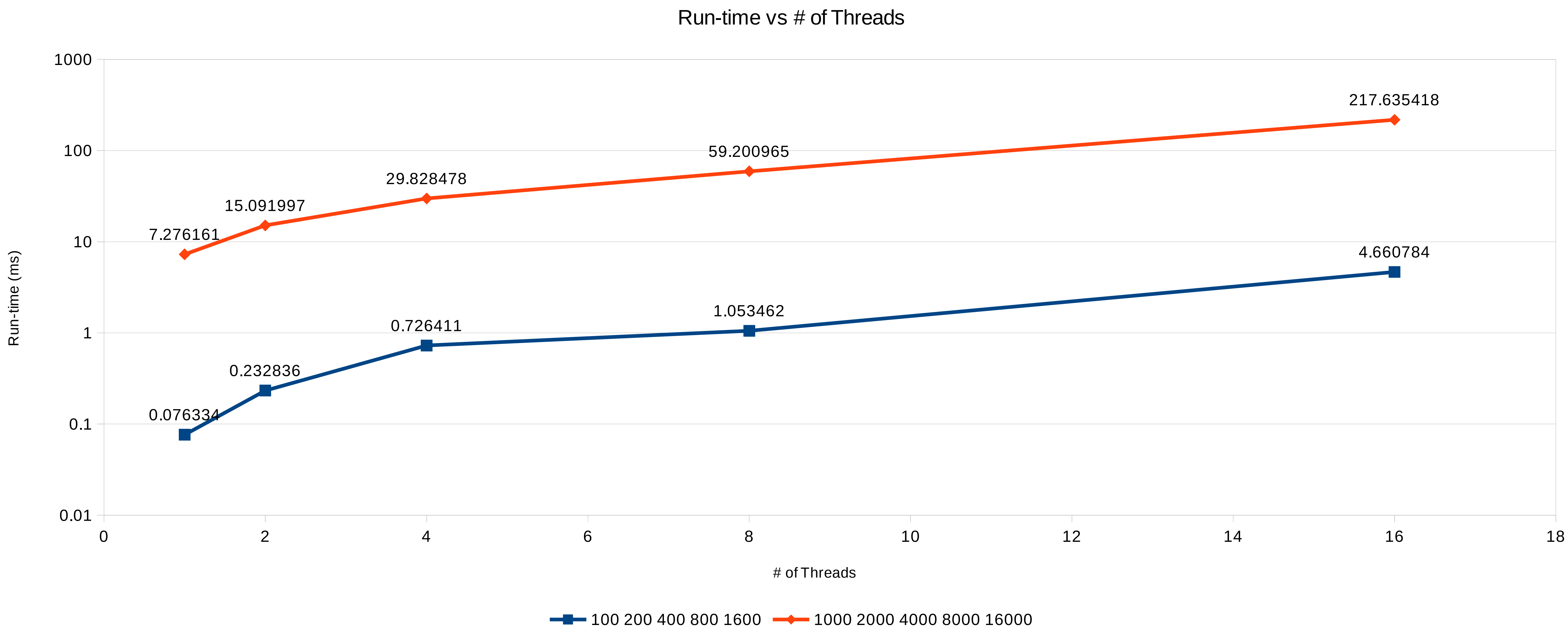
Weak Scaling Benchmark (# of bodies increases by 2)				
Type	Bodies	Threads	avg(time)/step (ms)	Weak Scaling Efficiency
Serial	100	1	0.076334	100.00%
weak/Parallel	200	2	0.232836	32.78%
weak/Parallel	400	4	0.726411	10.51%
weak/Parallel	800	8	1.053462	7.25%
weak/Parallel	1600	16	4.660784	1.64%
Serial	1000	1	7.276161	100.00%
weak/Parallel	2000	2	15.091997	48.21%
weak/Parallel	4000	4	29.828478	24.39%
weak/Parallel	8000	8	59.200965	12.29%
weak/Parallel	16000	16	217.635418	3.34%

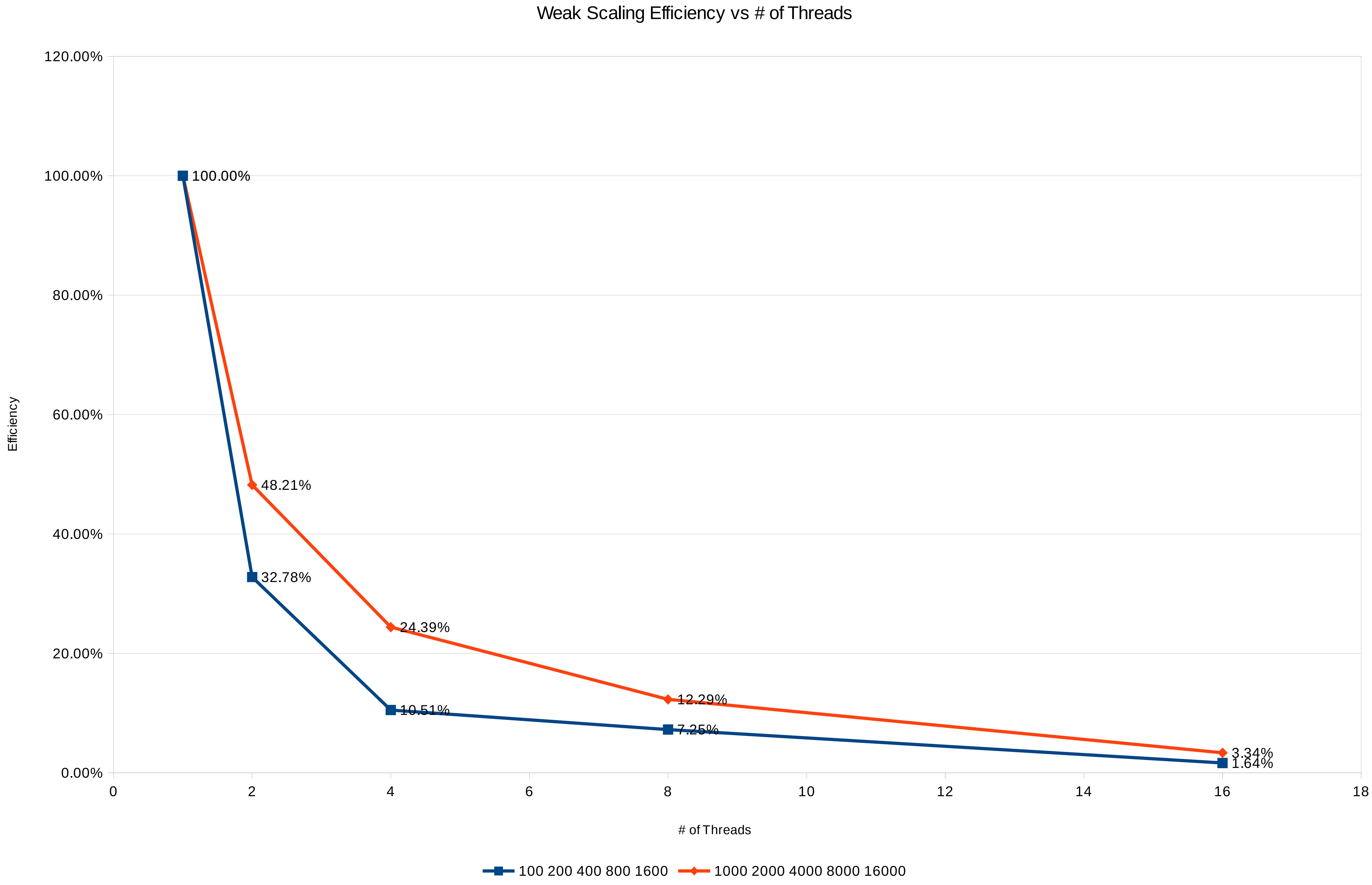
Weak Scaling Benchmark (# of bodies increases by sqrt(# of threads))				
Type	Bodies	Threads	avg(time)/step (ms)	Weak Scaling Efficiency
Serial	100	1	0.076334	100.00%
weak/Parallel	141	2	0.105126	72.61%
weak/Parallel	200	4	0.208851	36.55%
weak/Parallel	282	8	0.463093	16.48%
weak/Parallel	400	16	3.175723	2.40%
Serial	1000	1	7.276161	100.00%
weak/Parallel	1414	2	7.78771	93.43%
weak/Parallel	2000	4	7.756429	93.81%
weak/Parallel	2828	8	7.653985	95.06%
weak/Parallel	4000	16	17.242676	42.20%


Run-time vs # of threads
1000
100
10
1
0.1
0.01
Run-time (ms)
727.079904
367.63911
184.098813
92.246404
92.196727
7.276161
4.244702
2.294433
1.609559
3.757233
2.174452
0.076334
0.075613
0.081775
0.11031
0
2
4
6
8
10
12
14
16
18
# of Threads
100
1000
10000

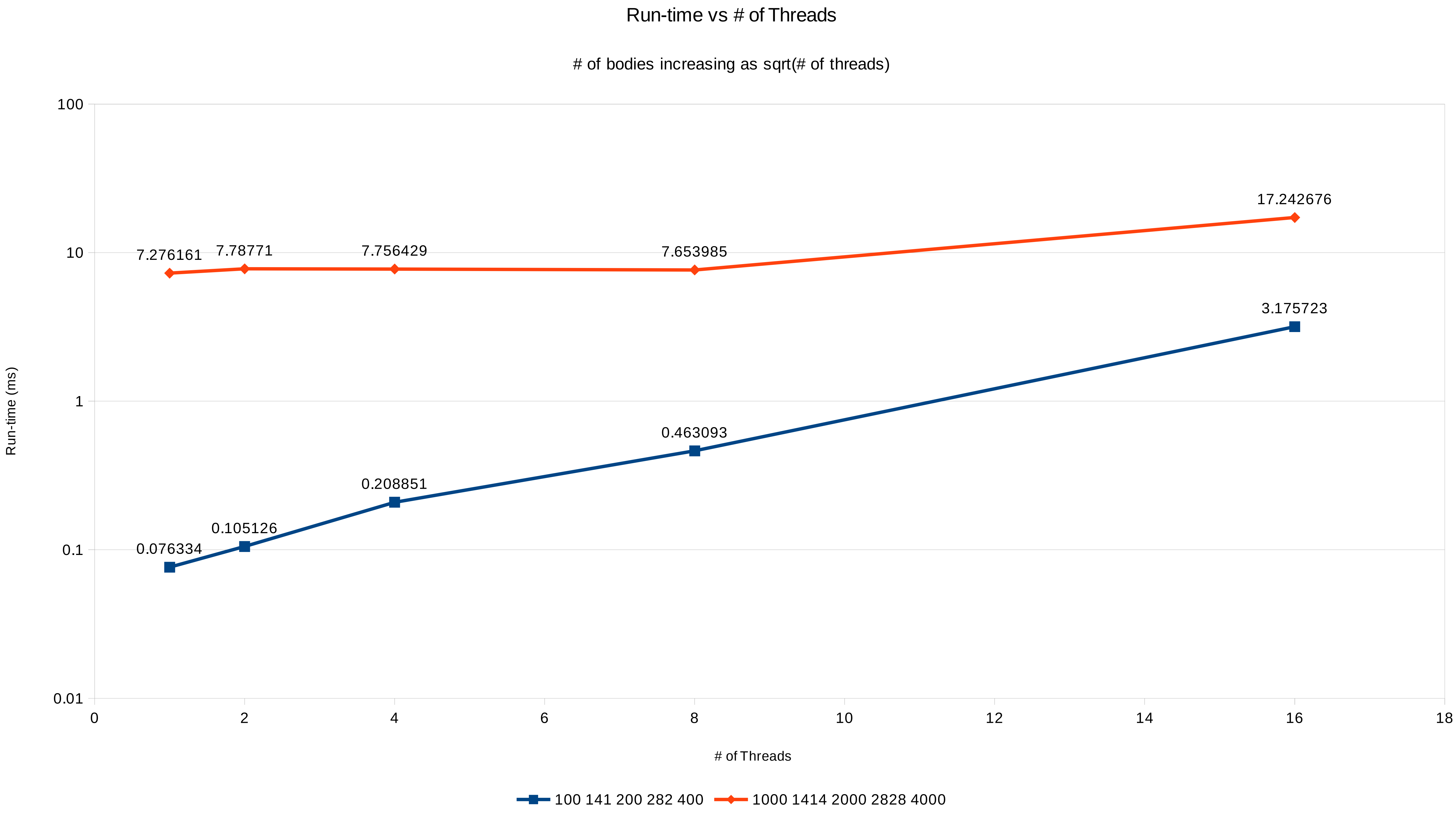

Strong Scaling Efficiency vs # of Threads
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Efficiency
0
2
4
6
8
10
12
14
16
18
# of Threads
100.00%
98.89%
98.74%
98.52%
49.29%
85.71%
79.28%
56.51%
12.10%
50.48%
23.34%
8.65%
0.22%
100
1000
10000

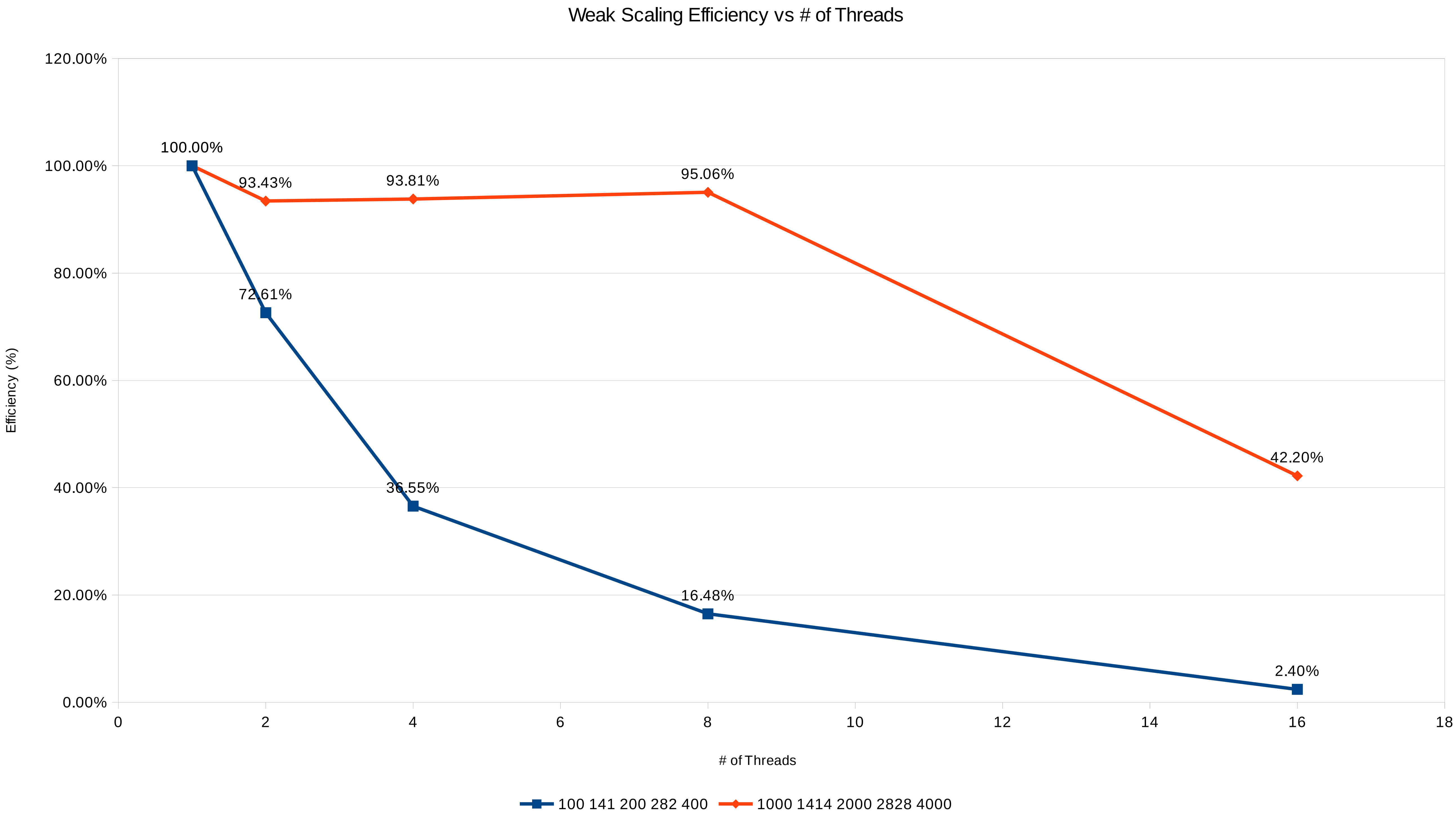
Weak Scaling (increasing 2 times)




Weak Scaling Efficiency vs # of Threads
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Efficiency
0
2
4
6
8
10
12
14
16
18
# of Threads
100.00%
48.21%
32.78%
24.39%
10.51%
12.29%
7.25%
3.34%
1.64%
100 200 400 800 1600
1000 2000 4000 8000 16000

Weak Scale (increasing sqrt(# of threads))




Weak Scaling Efficiency vs # of Threads
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Efficiency (%)
0
2
4
6
8
10
12
14
16
18
# of Threads
100.00%
93.43%
93.81%
95.06%
42.20%
72.61%
36.55%
16.48%
2.40%
100 141 200 282 400
1000 1414 2000 2828 4000