N-body problem

UROŠ LOTRIČ

Introduction

Physics, chemistry

Problem description

- Force between two bodies
  - $\vec{r}_{ij} = (x_{ij}, y_{ij}, z_{ij}) = (x_i - x_j, y_i - y_j, z_i - z_j)$
  - $r_{ij}^2 = x_{ij}^2 + y_{ij}^2 + z_{ij}^2$
  - $\overrightarrow{F_{ij}} = G\frac{m_i m_j}{r_{ij}^2}\frac{\vec{r}_{ij}}{r_{ij}} = (F_{ijx}, F_{ijy}, F_{ijz}) = (G\frac{m_i m_j}{r_{ij}^2}\frac{x_{ij}}{r_{ij}}, G\frac{m_i m_j}{r_{ij}^2}\frac{y_{ij}}{r_{ij}} G\frac{m_i m_j}{r_{ij}^2}\frac{z_{ij}}{r_{ij}})$
- Total force on body i
  - $F_{ix} = Gm_i\sum_{j\neq i}\frac{m_j}{r_{ij}^2}\frac{x_{ij}}{r_{ij}}$, $F_{iy} = Gm_i\sum_{j\neq i}\frac{m_j}{r_{ij}^2}\frac{y_{ij}}{r_{ij}}$, $F_{iz} = Gm_i\sum_{j\neq i}\frac{m_j}{r_{ij}^2}\frac{z_{ij}}{r_{ij}}$
  - $F_{ix} = Gm_i\sum_{j}\frac{m_j}{R_{ij}^2}\frac{x_{ij}}{R_{ij}}$, $F_{iy} = Gm_i\sum_{j}\frac{m_j}{R_{ij}^2}\frac{y_{ij}}{R_{ij}}$, $F_{iz} = Gm_i\sum_{j}\frac{m_j}{R_{ij}^2}\frac{z_{ij}}{R_{ij}}$
    - Smoothing $R_{ij}^2 = r_{ij}^2 + \varepsilon^2$
    - when r_{ij} = 0, x_{ij} = y_{ij} = z_{ij}= 0
    - Prevents division by zero when $i = j$

Introduction

Problem description

- Body movement
  - $a_{ix} = \frac{F_{ix}}{m_i}$, $a_{iy} = \frac{F_{iy}}{m_i}$, $a_{iz} = \frac{F_{iz}}{m_i}$
  - $a_{ix} = G\sum_j \frac{m_j}{R_{ij}^2}\frac{x_{ij}}{R_{ij}}$, $a_{iy} = G\sum_j \frac{m_j}{R_{ij}^2}\frac{y_{ij}}{R_{ij}}$, $a_{iz} = G\sum_j \frac{m_j}{R_{ij}^2}\frac{z_{ij}}{R_{ij}}$
  - $s_{ix} = s_{ix} + v_{ix}\Delta t + \frac{1}{2}a_{ix}\Delta t^2$, $s_{iy} = s_{iy} + v_{iy}\Delta t + \frac{1}{2}a_{iy}\Delta t^2$, $s_{iz} = s_{iz} + v_{iz}\Delta t + \frac{1}{2}a_{iz}\Delta t^2$
  - $v_{ix} = v_{ix} + a_{ix}\Delta t$, $v_{iy} = v_{iy} + a_{iy}\Delta t$, $v_{iz} = v_{iz} + a_{iz}\Delta t$

Parallelization concepts

Per body

- N bodies, N interactions: $t_s(N) = \chi N^2 = O(N^2)$
- Parallelization: $t_p(N,P) = O\left(\frac{N^2}{P}\right) + \kappa(N,P)$ at P tasks

Per force

- As $F_{ij} = -F_{ji}$, the number of work can be halved
- More complex scheduling scheme

Experiment setup

Technologies

- OpenMP
- OpenCL
- OpenMPI

Timing

- Per iteration
- Start/end data transfer from host

Runs

- 8, 16, 32, …, 65536 bodies
- 5 per configuration

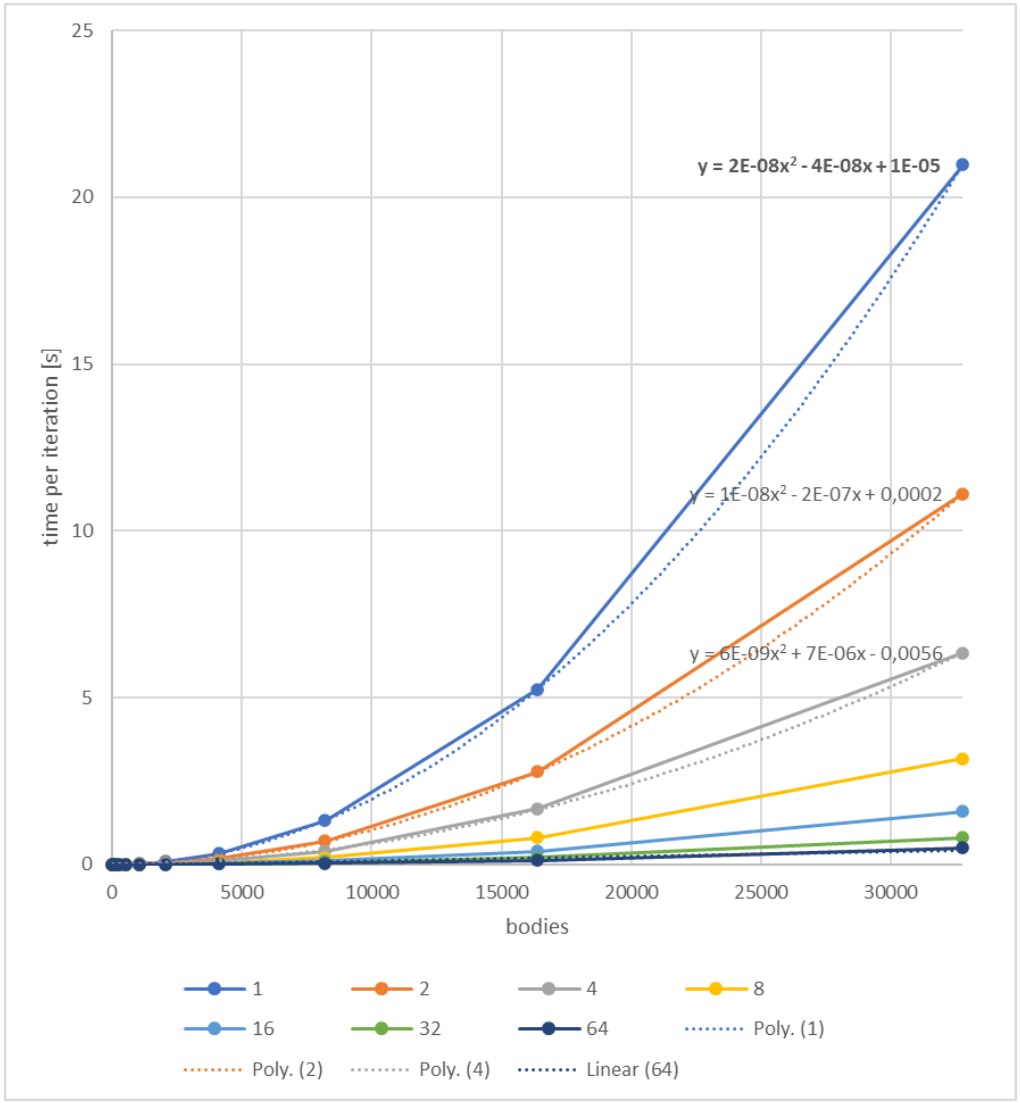
OpenMP

Code overview

- nbody-mp.c

Cores

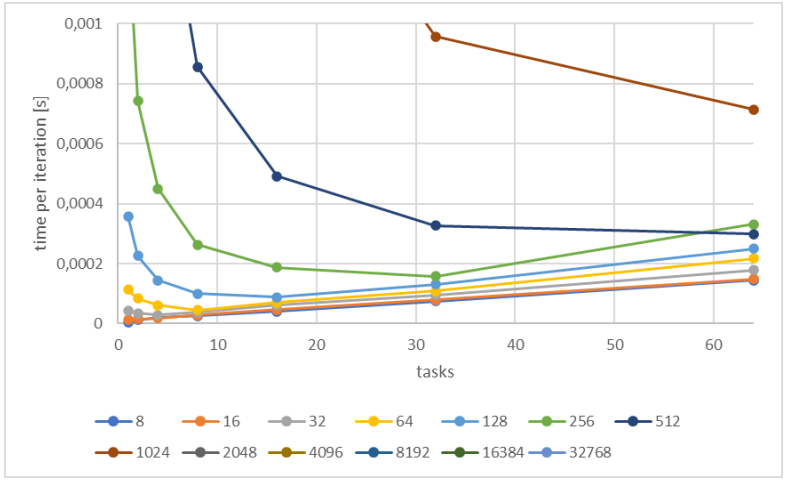
- 1, 2, 4, 8, … 64
- Expected quadratic behaviour
- Time per interaction calculation
  - 1 core: no thread creation and synchronization
  - $\chi_{MP} = 2 \cdot 10^{-8} s$

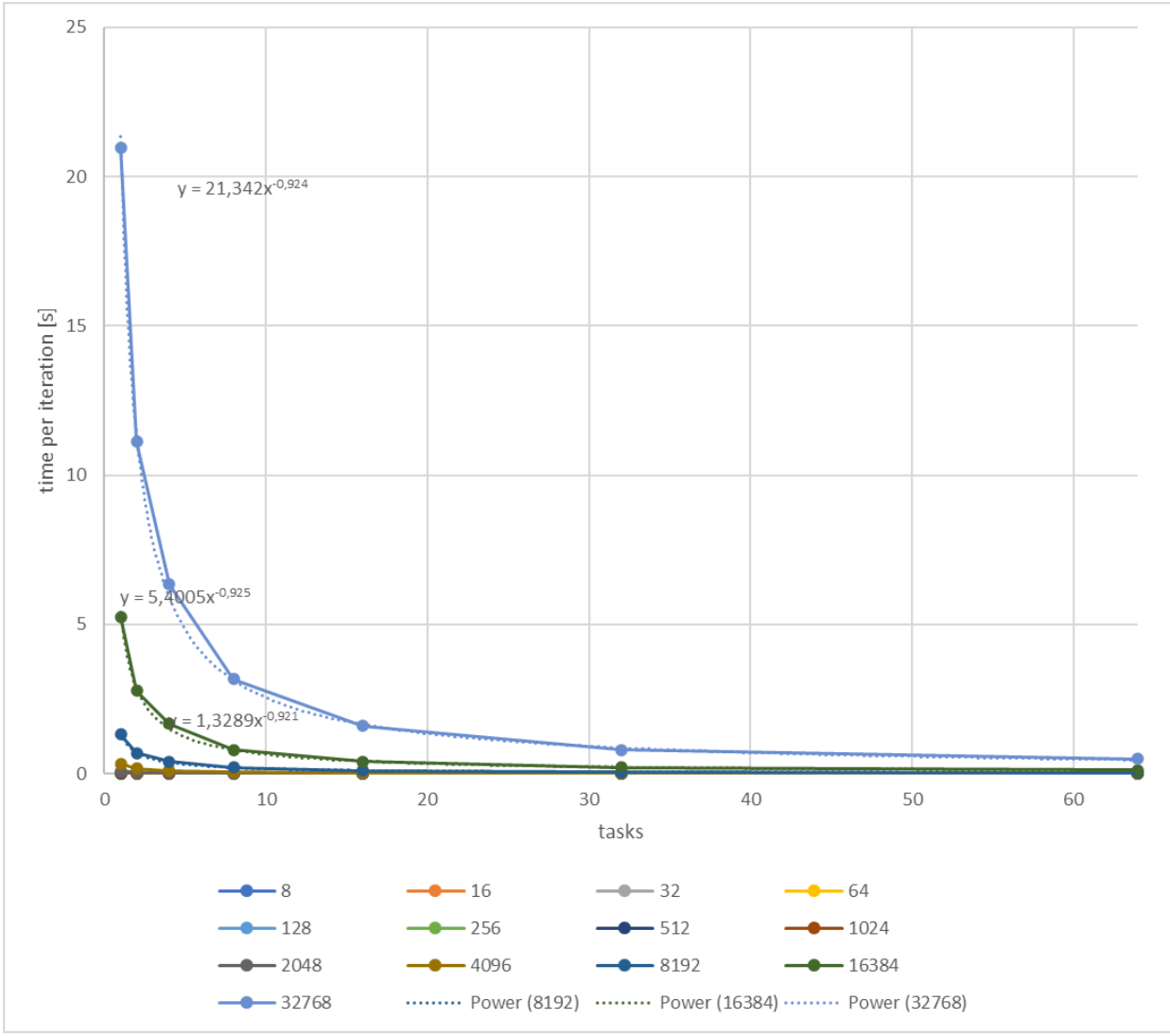


OpenMP

Cores

- Time decreases with power
  - Theory: -1, experiment -0.925
  - Theory does not take into account thread creation and synchronization (barriers)
  - For small problem sizes computing on large number of cores does not pay of

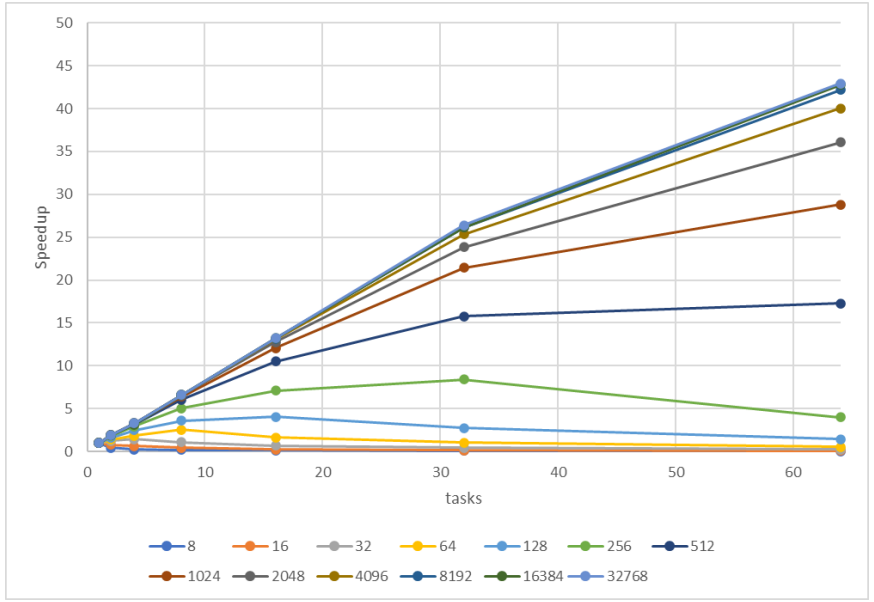


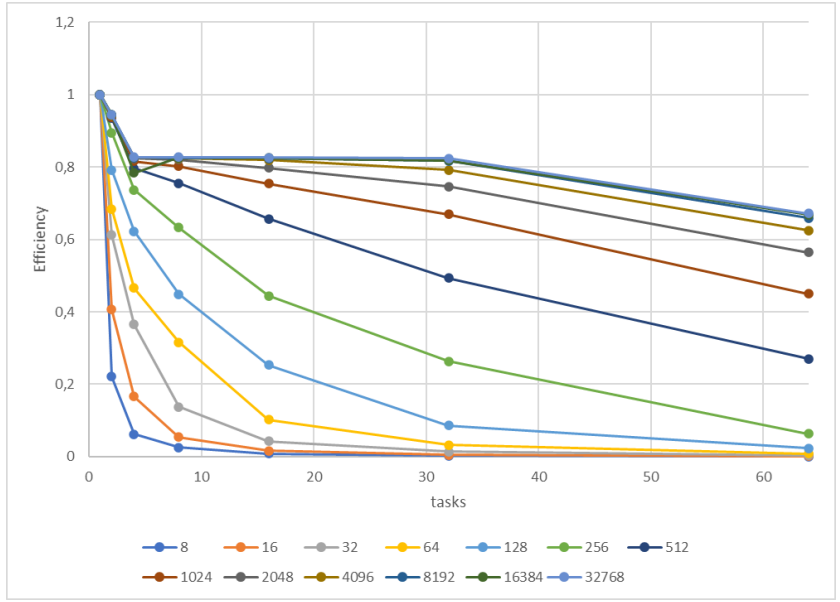


OpenMP

Speedup and efficiency

- $S(N,P) = \frac{t_s(N)}{t_p(N,P)}$, $E(N,P) = \frac{S(N,P)}{P}$
- Speedup increases with number of tasks
- Large problem sizes are computed more efficiently
  - Even 32768 bodies is not enough to compute on 64 cores





OpenCL

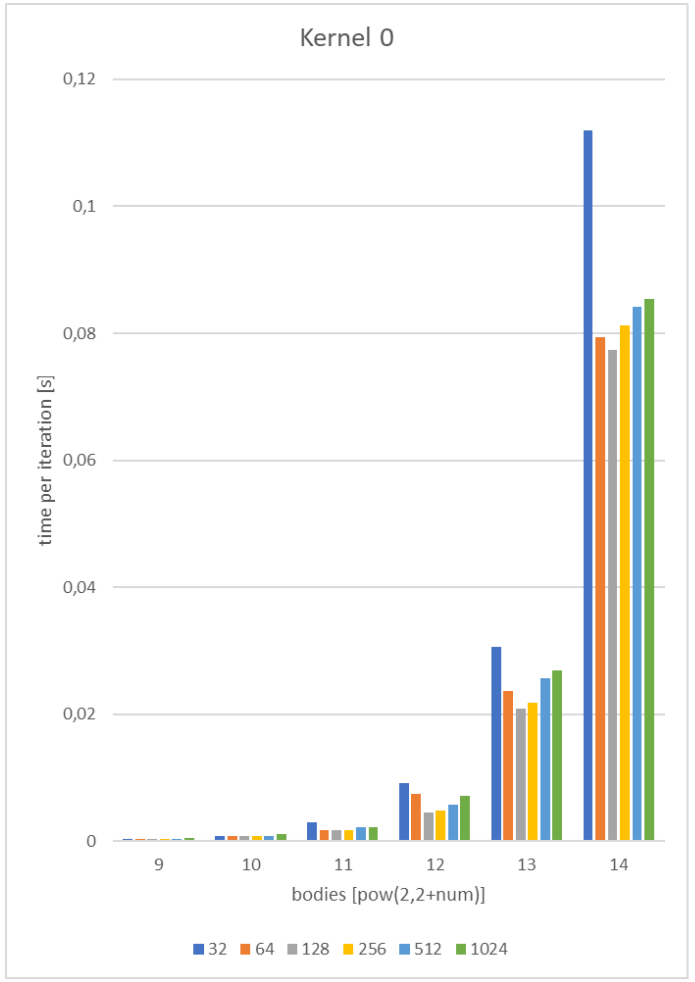
Code overview

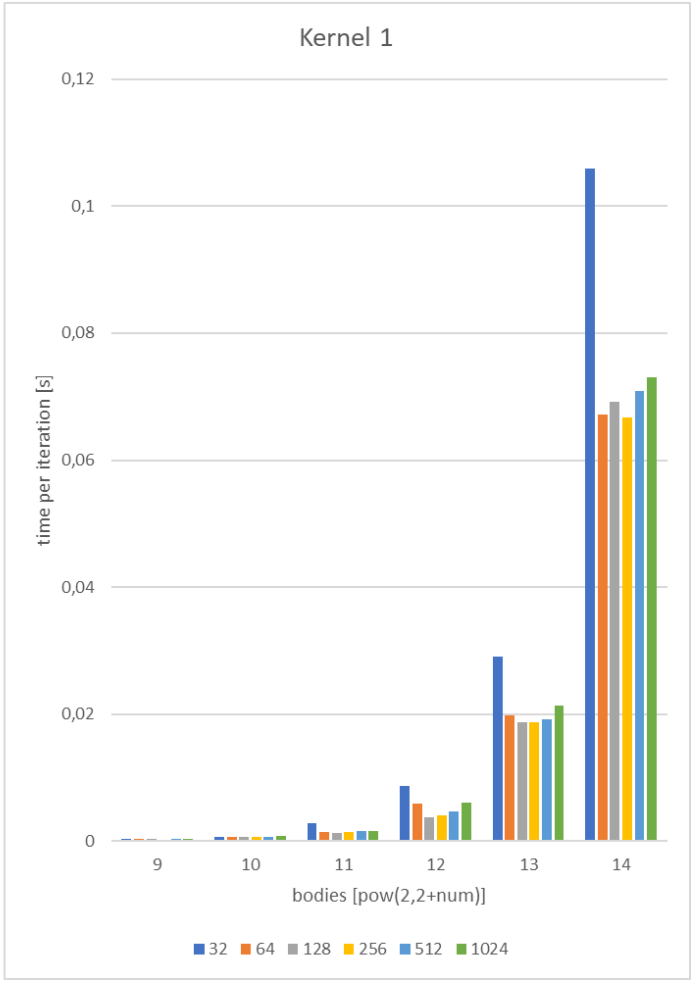
- nbody-cl.c
- Two kernels
  - nbody-cl-0.cl
    - Works directly with global memory
  - nbody-cl-1.cl
    - Copies body locations to local memory before computation

OpenCL

Work-group size

- Tested sizes
  - 32, 64, 128, 256, 512, 1024
- Comment
  - Low values: a lot of barriers
  - High values:
    not enough thread resources
  - K40m:
    - 2880 PE → 192 PE per CE

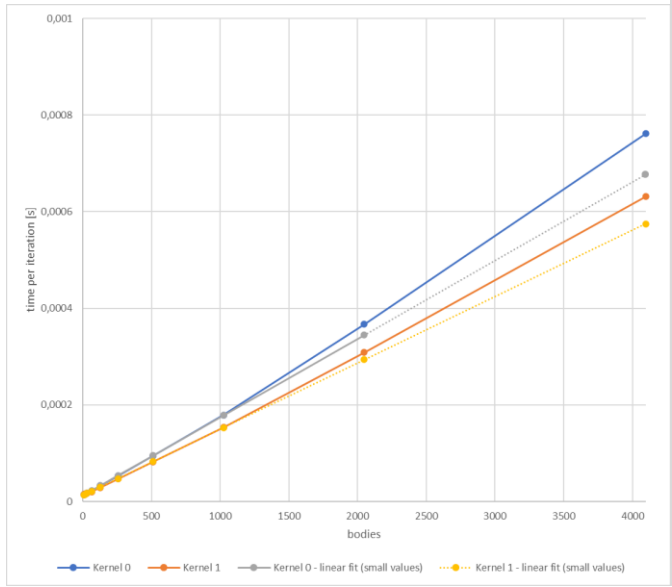


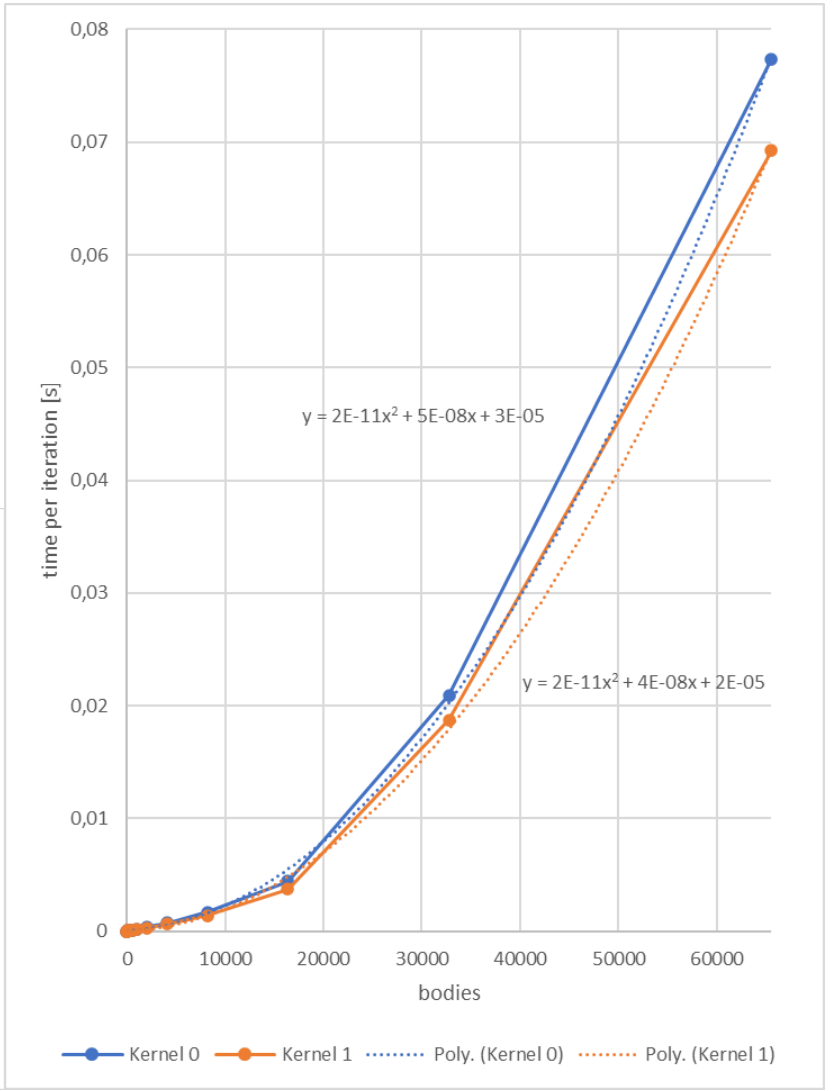


OpenCL

Time per iteration

- Expected quadratic behaviour
- First coefficient is χ
  - Kernel 0: $\chi = 1.72 \cdot 10^{-11} s$, kernel 1: $\chi = 1.56 \cdot 10^{-11} s$
  - Kernel 1 is better
  - Computation on GPU is approx. 1000 times faster compared to CPU
- For small problem sizes linear behaviour
  - GPU is not fully utilized
  - For full utilization needs at least $15 \cdot 128 = 1920$ workers

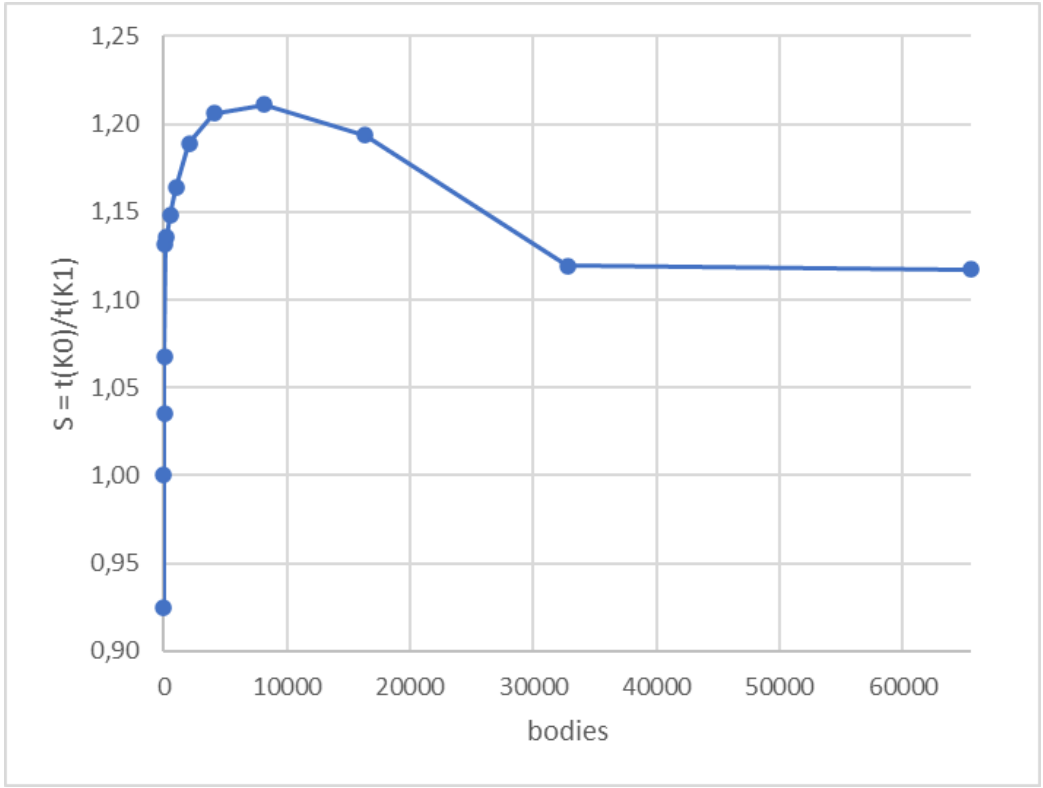




OpenCL

Time per iteration

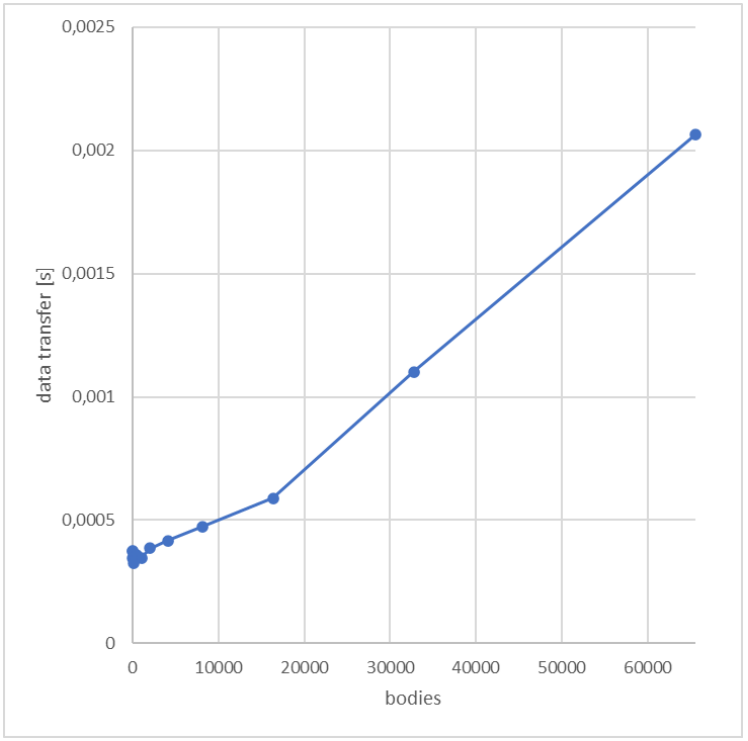
- Speedup: $S(N) = \frac{t_{Kernel\,0}(N)}{t_{Kernel\,1}(N)}$
- Analysis
  - Kernel 0 randomly reads data from global memory
  - Kernel 1 has two barriers to support coalesced reads
  - For very small problems coalesced reading is irrelevant
  - For very large problems speedup stabilizes at approx. 10%
    - GPU caching helps Kernel 0 for small problem sizes, less than 1024 particles
    - A lot of workers, better latency hiding for large problem sizes

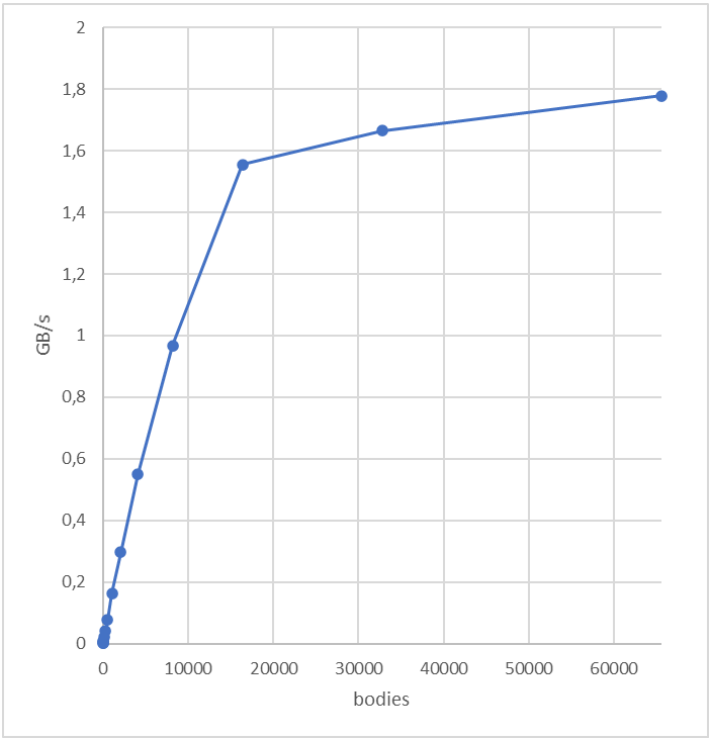


OpenCL

Data transfer

- To GPU + From GPU
  - 2x7xNxW: (to GPU + from GPU) x [mass + 3 positions + 3 velocities) x bodies x bytes per value
- Bandwidth approaches 2GB/s
  - PCI-E 3.0 supports transfers up to 10 GB/s
  - Problem size is to small to reach maximal bandwidth





OpenMPI

Code review

- nbody-mpi.c
- Group communication

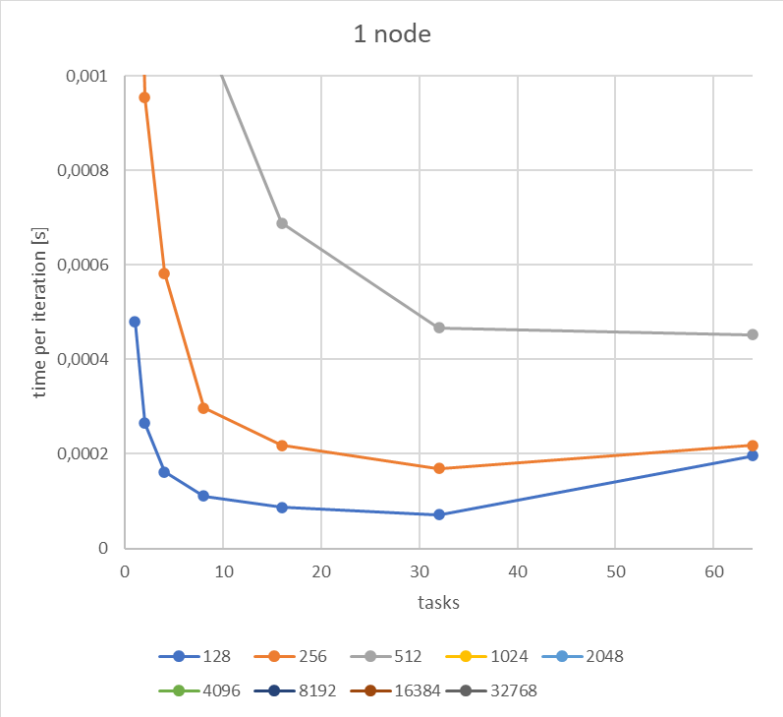
Experiment

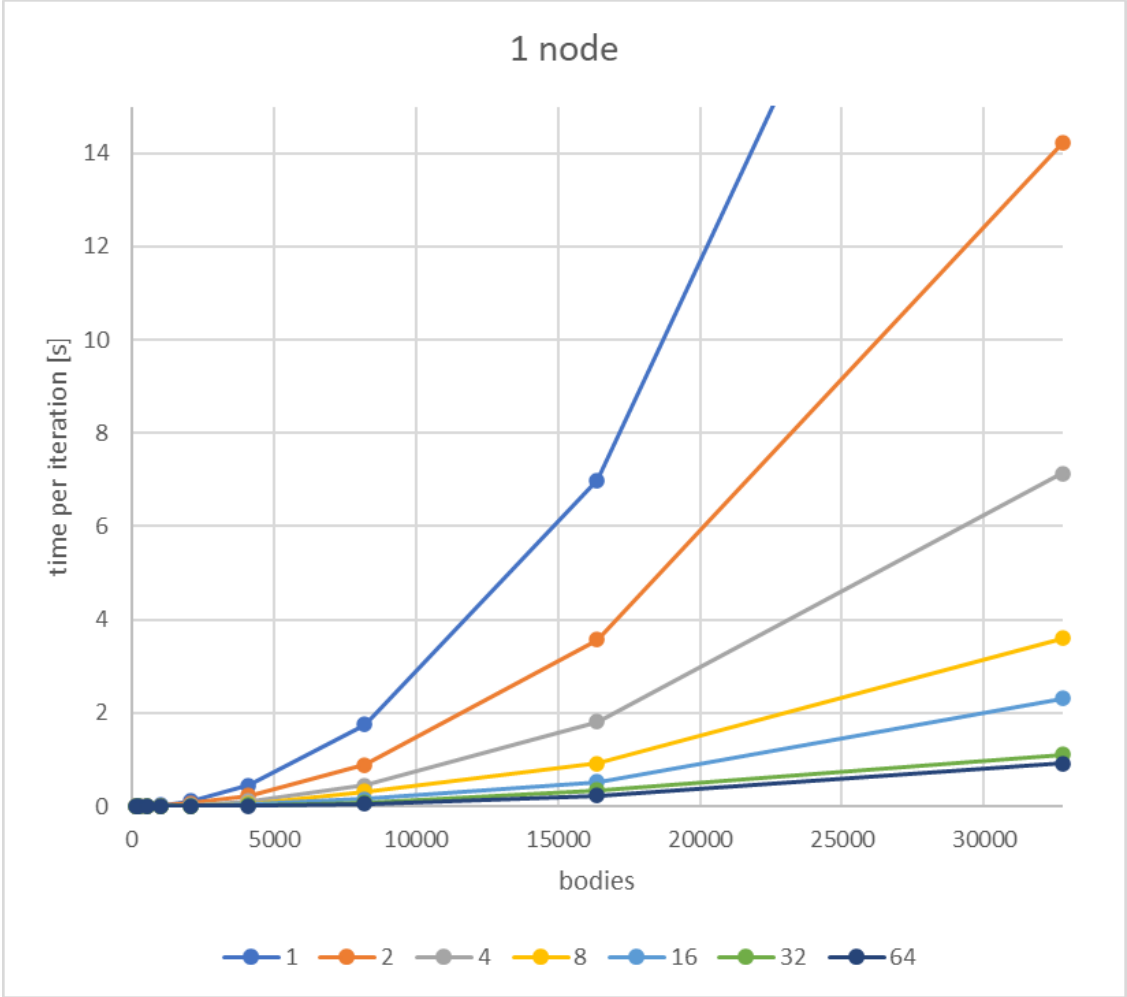
- 1, 2, 4, 8, 16, 32, 64, 128 cores
- 1 or 2 nodes
- Work is equally distributed among nodes and sockets

OpenMPI

Time per iteration, 1 node

- Computation + Allgather
  - At least 512 bodies for 64 cores

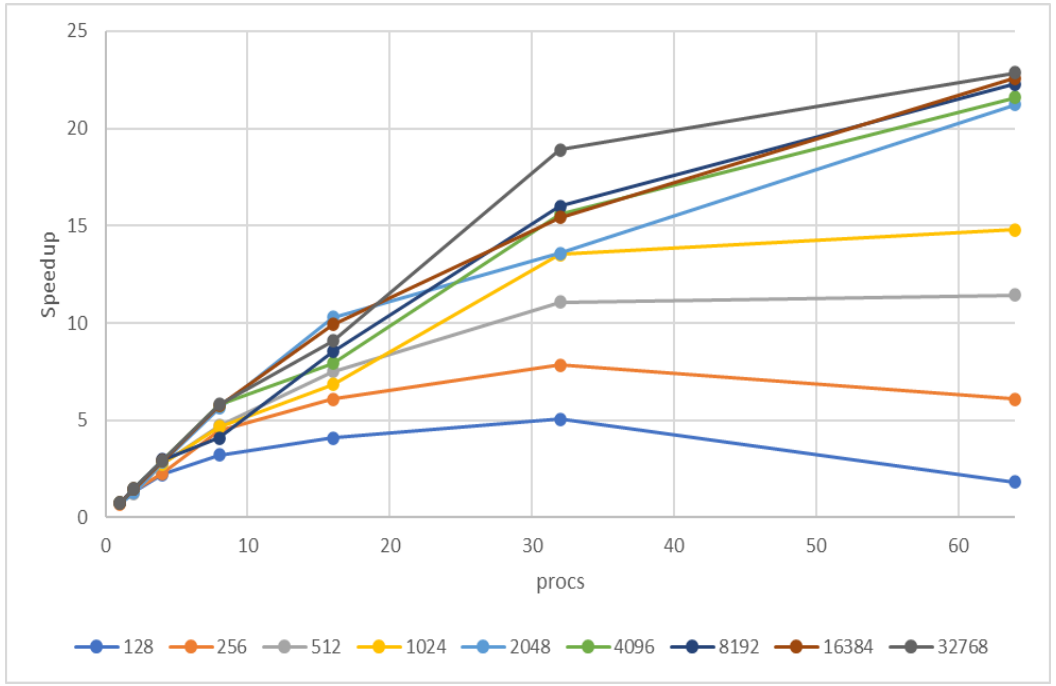


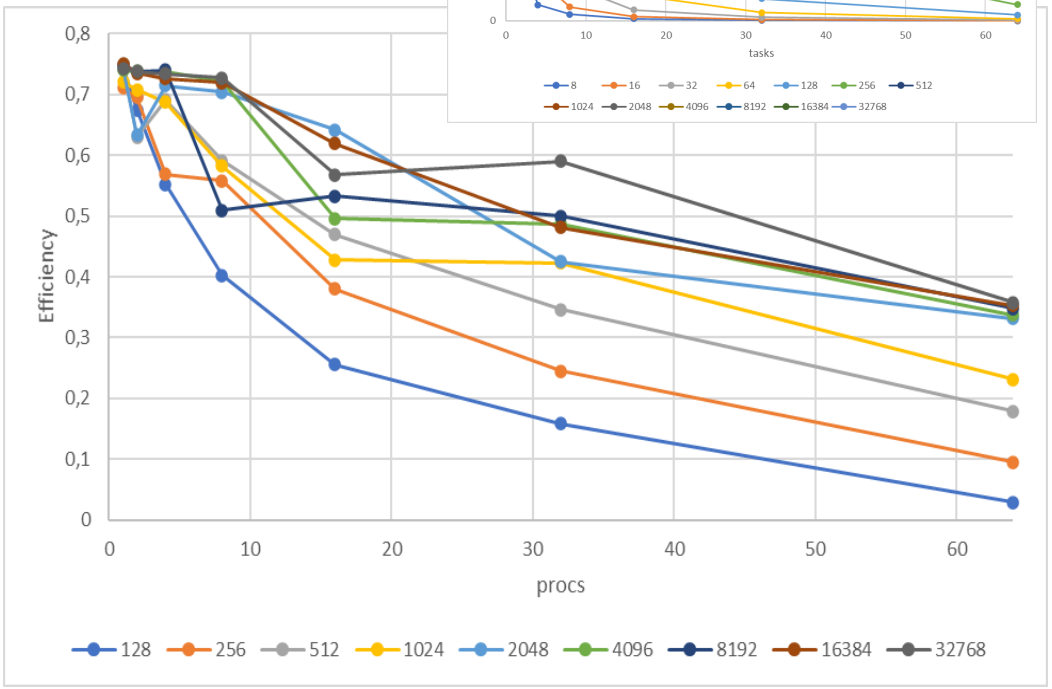


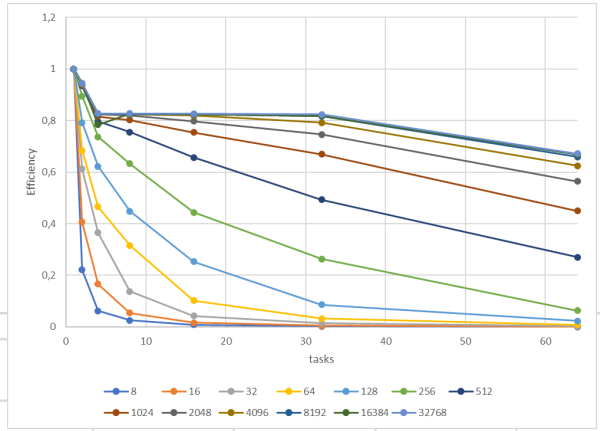
OpenMPI

Time per iteration, 1 node

- Speedup and efficiency
- Smaller efficiency (75% of OpenMP efficiency)
  - OpenMPI needs bigger problems for better efficiency
  - Makes sense to combine OpenMP and OpenMPI





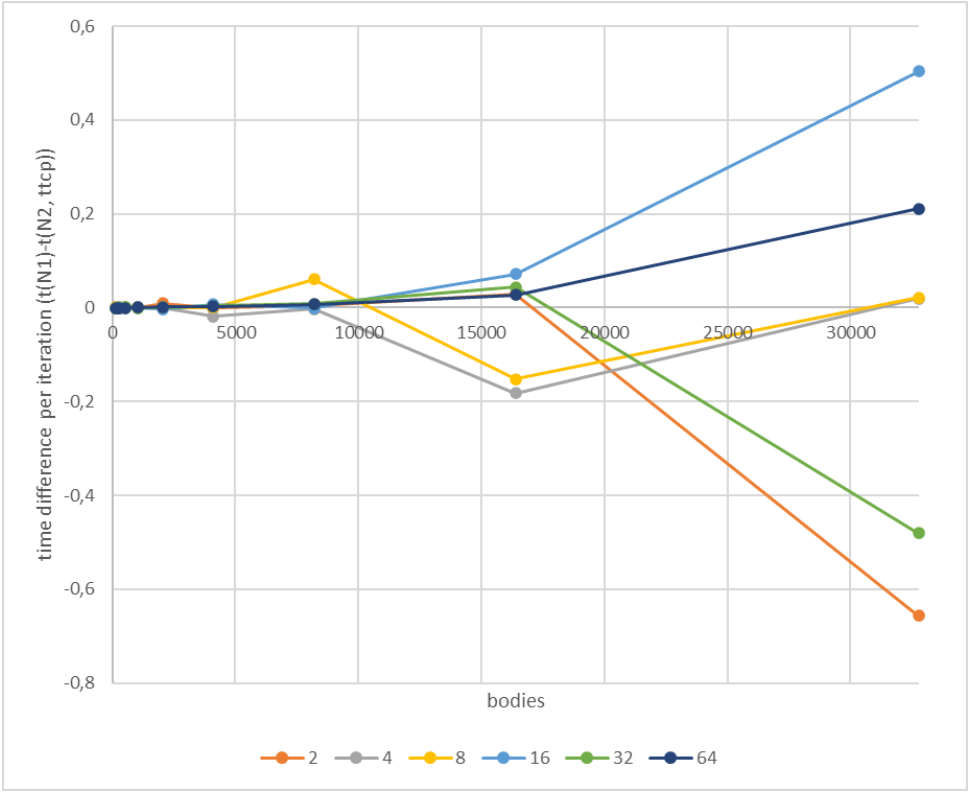


OpenMPI

Time per iteration, 2 nodes

- Time difference per iteration: $t_{1\ node} - t_{2\ nodes}$
- Slightly better on 2 nodes
  - Only communication between two nodes
  - Quality of protocol for data exchange on the same node
  - With 2 nodes less problems with cache and more processor resources per task (2 cores per FP)

bodies	1	2	4	8	16	32	64	128
128		-0,0000534	0,000028	2,88E-05	-0,00027	-0,00035	-0,00042	
256		-4,66E-05	9,64E-05	3,18E-05	-0,00014	-0,0003	-0,00131	
512		0,000558	3,08E-05	-6,8E-06	-8,9E-05	-0,00019	-0,00045	
1024		-3,86E-05	9,34E-05	0,000568	0,000741	-0,00039	-0,0002	
2048		0,0090488	0,000676	0,000314	-0,00274	0,000367	0,000976	
4096		-0,0007266	-0,01835	0,000502	0,007516	0,003891	0,00309	
8192		0,0051994	-0,00328	0,060123	-0,00023	0,007614	0,006723	
16384		0,0290196	-0,18183	-0,1526	0,071758	0,04376	0,027287	
32768		-0,6567244	0,019261	0,021433	0,504166	-0,48074	0,211331	
						Green: better time on 2 nodes		



OpenMPI

Theoretical estimation of communication costs

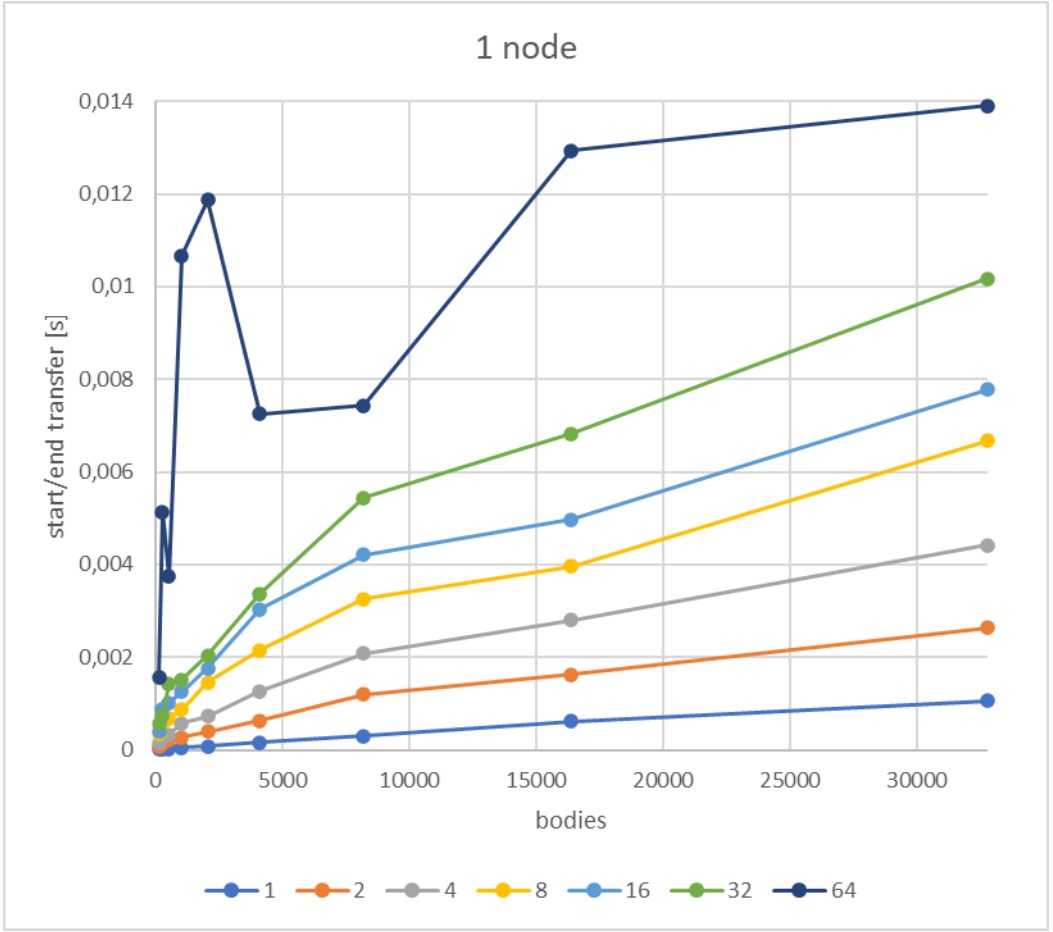
- Bcast: $\log_2 P \left(\lambda + \frac{N}{\beta}\right)$

- Scatter/gather/allgather: $\sum_{i=1}^{\log_2 P} (\lambda + \frac{N}{2^i \beta}) = \lambda \log_2 P + \frac{N}{\beta} \cdot \frac{P-1}{P}$

OpenMPI

Data transfer, 1 node

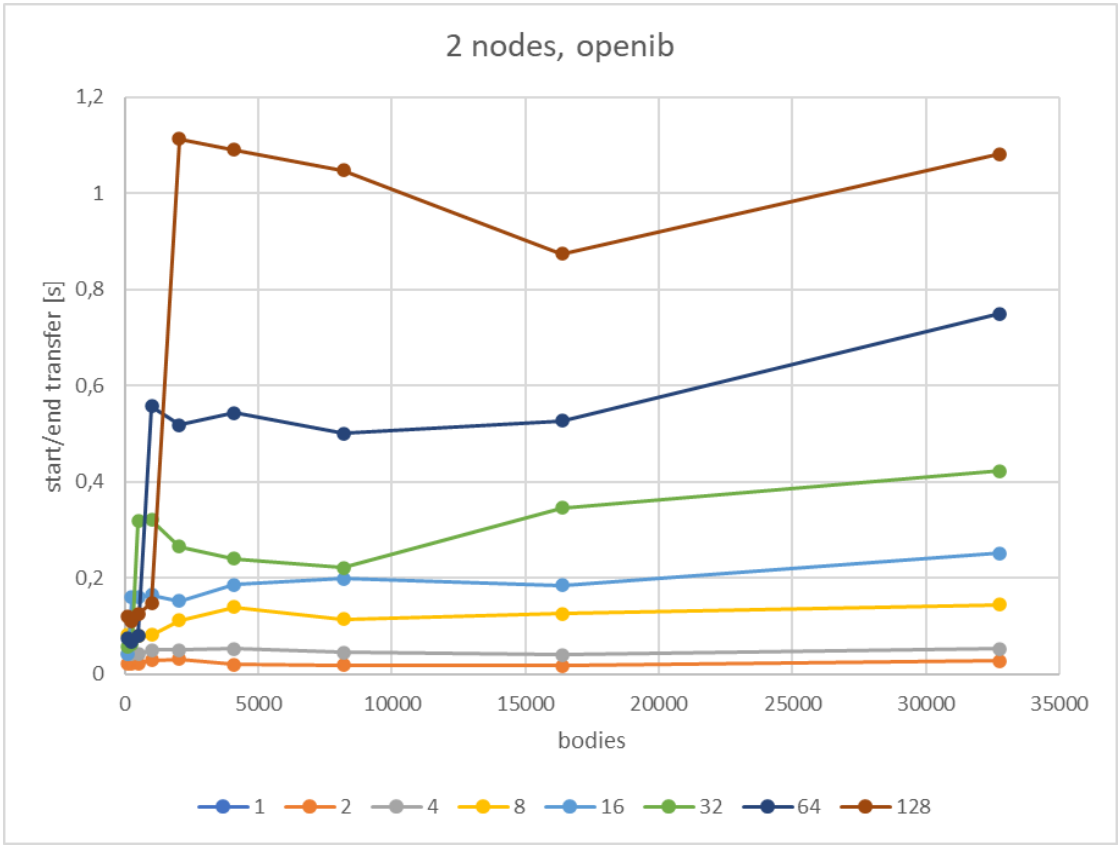
- 1 x bcast + 1 x scatter + 2 x gather
- Linear dependency
  - Latency $\lambda = 6.7 \cdot 10^{-5} s$
  - Bandwidth $\beta = 450\ MB/s$



OpenMPI

Data transfer, 2 nodes

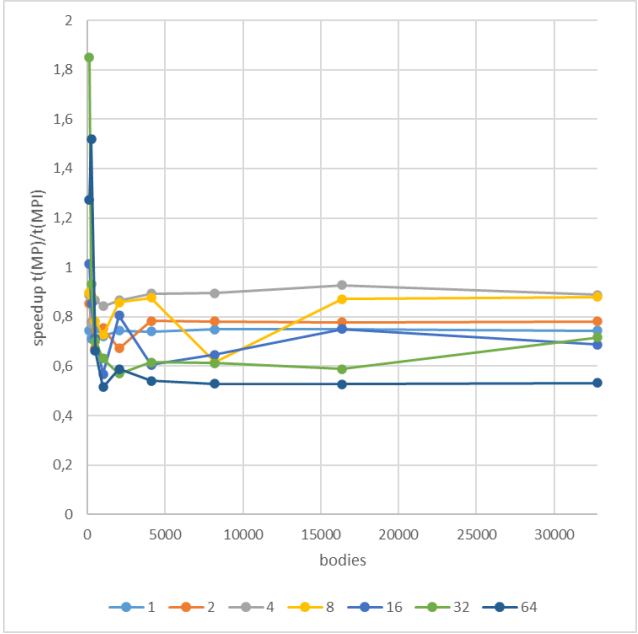
- Estimation of latency and bandwidth
  - Estimates are based on 64/128 tasks and 16384/32768 bodies
  - OpenIB: $\lambda \approx 1.8 \cdot 10^{-2}, \beta \approx 11MB/s$
  - Latency prevails, not enough data to transfer
  - For better estimates
    - Should test with much larger data sizes and do more runs
    - Test communication separately from computation

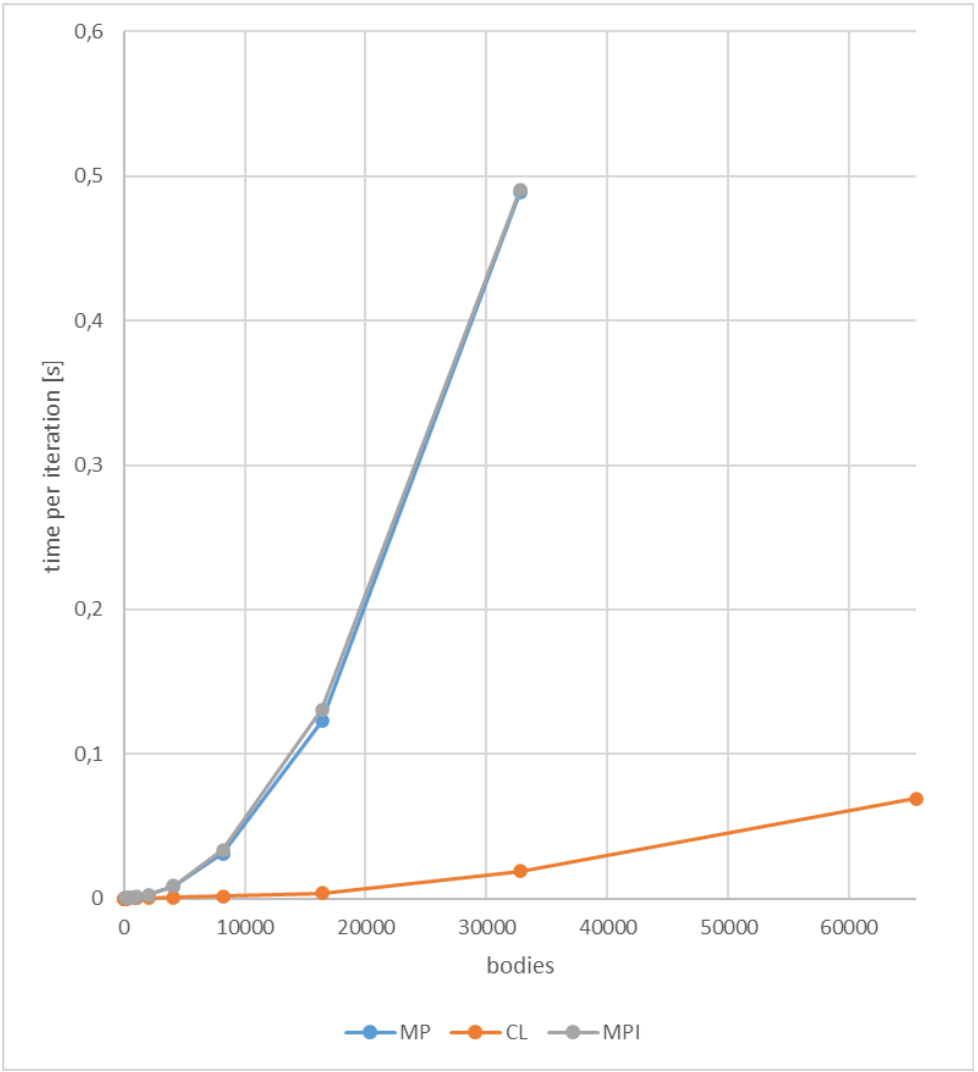


Overall comparison

Time per iteration

- Best results from each technology
- Problem fits OpenCL
- OpenMP and OpenMPI have similar results
  - OpenMPI speedups are approx. 75 % of OpenMP speedups
  - Hybrid solution
    - OpenMPI among nodes
    - OpenMP on a node

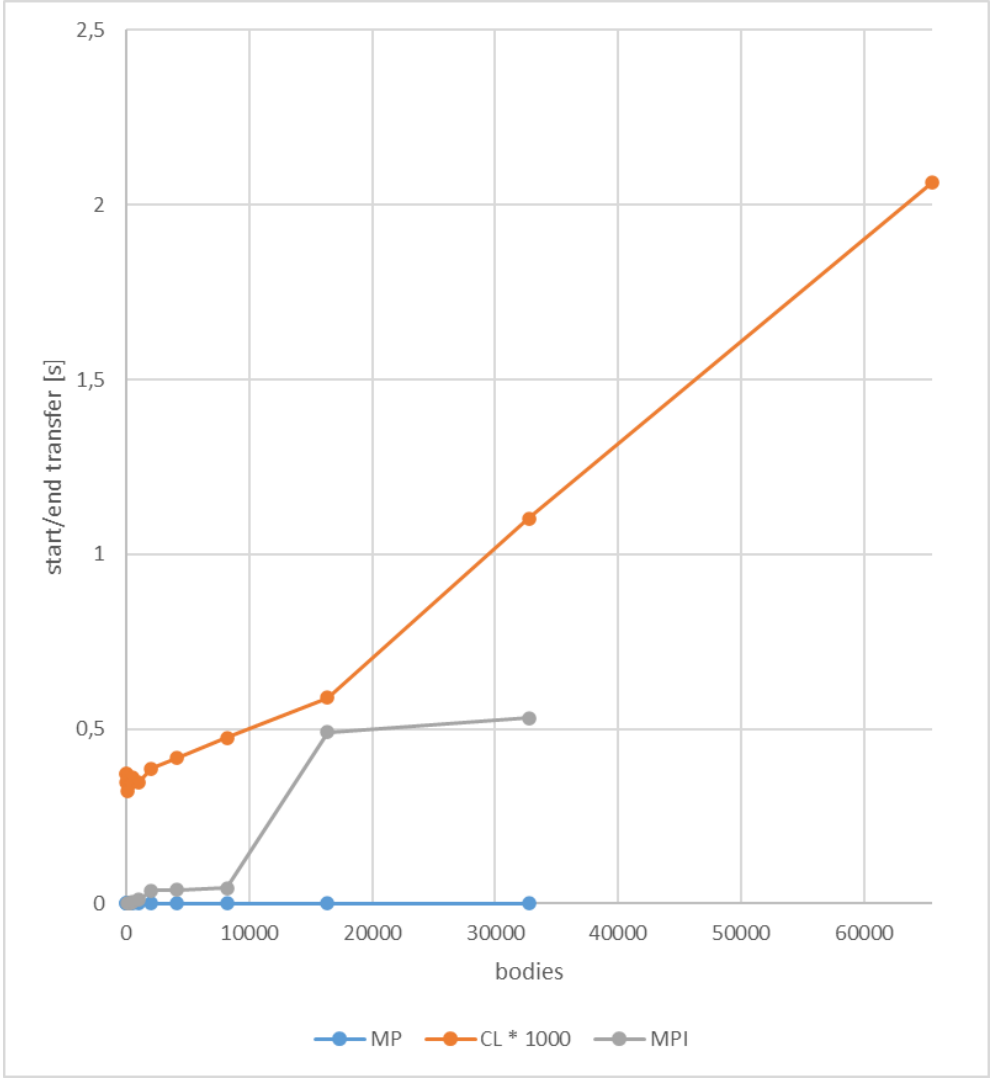




Overall comparison

Data transfer

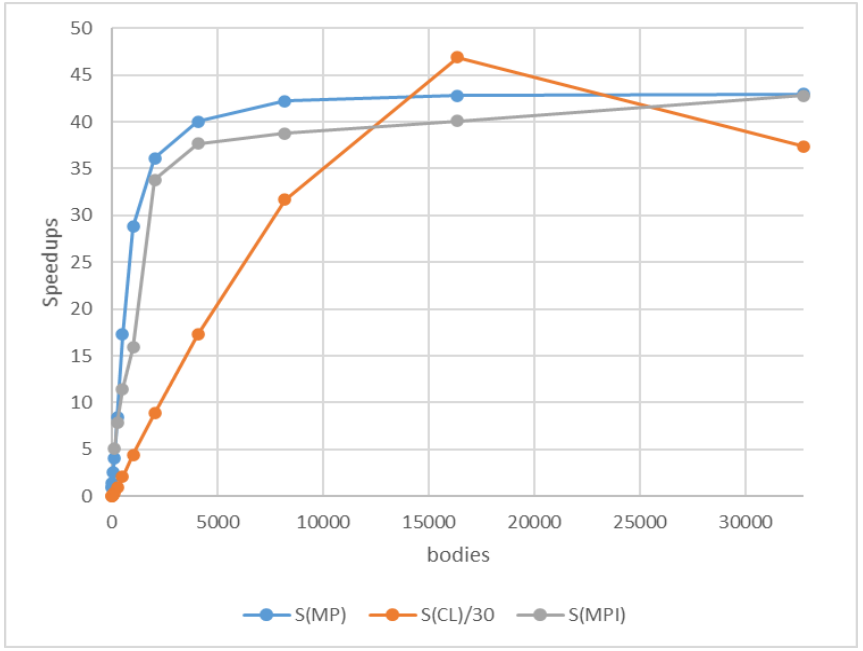
- OpenCL is 1000 x faster
- OpenMP does have no explicit data transfers
- OpenMPI transfers data among processes even on the same node

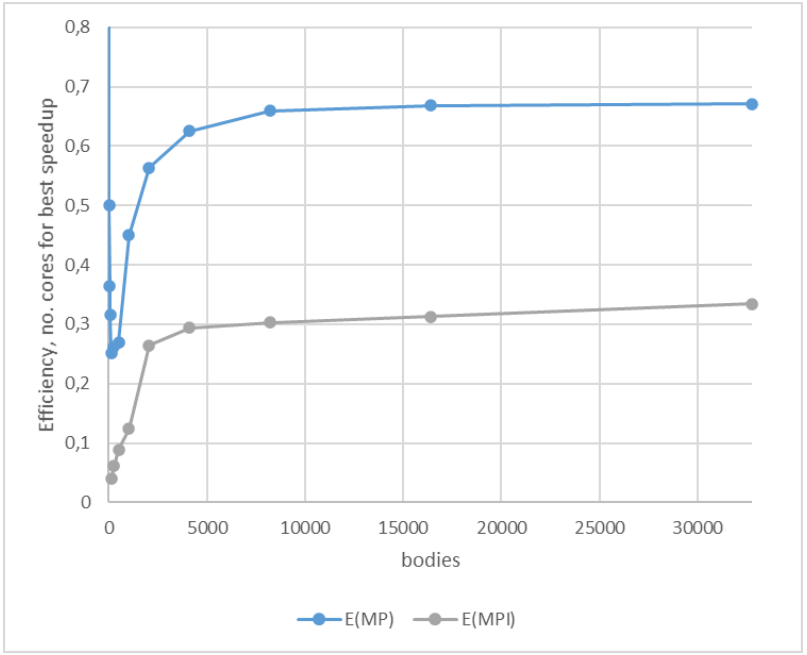


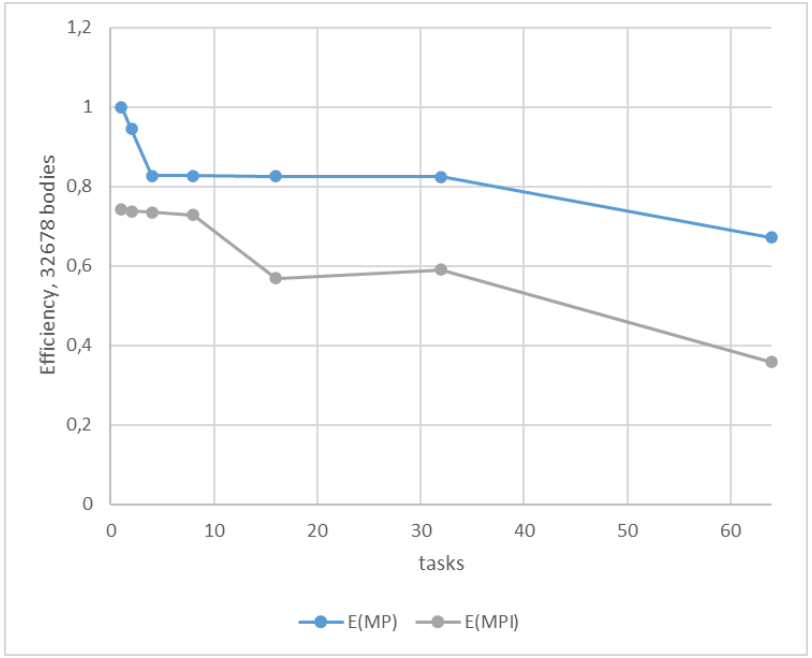
Overall comparison

Speedup and efficiency

- Relative to single core runs
- OpenCL is 30 x faster than OpenMP and OpenMPI
- Efficiency drops with number of tasks
- OpenMP is more efficient than OpenMPI







Overall comparison

Scalability (iso-efficiency)

- $t_s(N) = \chi N^2$
- $t_p(N,P) = \chi \frac{N^2}{P} + \lambda \log_2 P + \frac{N}{\beta} \cdot \frac{P-1}{P}$
- $t_s(N) \geq C \cdot t_{overhead}(N,P) = P \cdot \kappa(N,P)$
- $\chi N^2 \geq C \cdot P \cdot \frac{N}{\beta} \quad \rightarrow \quad N \geq C'P$
  - To maintain scalability, the problem size must linearly increase with the number of parallel tasks
- $M(N) = m \cdot N$
- $\frac{M(N)}{P} \geq \frac{mC'P}{P} = C''$
  - The memory requirements per task are constant, the problem is well scalable.

Possible improvements

To get more reliable estimates of computing and communication times

- Increase number of iterations
- Increase number of runs

Not enough data for reliable estimation of communication

- Separate data transfer measurements on larger problem sizes

Add more partial time measurements to the code

- Easier to estimate theoretical model

Hybrid solutions

OpenCL + OpenMP

- Based on results, there will not be much gain with OpenMP

OpenCL on two GPUs

- Each node has two GPUs: could profit a lot

OpenCL + OpenMPI

- MPI to distributed work among nodes
  - One way to do it is to use as many cores as there are GPUs on each node
- OpenCL to do computation on all GPUs on a node