Position Papers of the Federated Conference on Computer Science and Information Systems pp. 31–37
DOI: 10.15439/2019F344
ISSN 2300-5963 ACSIS, Vol. 19

Performance and Energy Evaluation of Parallel Particle Simulation Algorithms for Different Input Particle Data

Robert Kiesel, Gudula Rünger
Department of Computer Science
Chemnitz University of Technology
Chemnitz, Germany
Email: {robert.kiesel, ruenger}@informatik.tu-chemnitz.de

***Abstract*—Particle simulations are popular methods for the simulation of applications from a wide range of sciences, including astrophysics, biology or chemistry. Usually, these applications require a large number of simulation steps, each of which computes a change of the entire particle system. Depending on the number of simulation steps and also the size and structure of the specific particle system, the computation time can be quite large and the exploitation of parallel architectures is usually necessary. In this article, we investigate the performance and energy consumption for different particle simulation methods and distinguish different input particle data. The investigations are done for the particle simulation methods from the ScaFaCoS library and use the various input data of homogeneous or inhomogeneous nature. Experiments are performed on multicore systems.**

I. Introduction

PARTICLE SIMULATIONS are popular methods for simulating various scientific problems from areas, such as astrophysics, biology or chemistry. The computation time for particle simulations can be quite large, especially for simulating long-range particle interactions, for example occurring in gravitational or Coulomb interactions, since the direct computation for a simulation with N particles has the complexity $\mathcal{O}(N^2)$. Efficient implementations of the particle simulation often split the computation with respect to a cut-off radius, which means that only the particle interactions of particles lying within the cut-off radius are computed exactly and the interactions of particles with a distance larger than the cut-off radius are computed by an approximation. The computation time can be reduced to $\mathcal{O}(N \log N)$ or $\mathcal{O}(N)$ with efficient methods, such as the Fast Multipole Method (FMM) [1] or the fast Fourier-transform (FFT) [2].

However, the actual execution time of a particle simulation depends on the given size N of the particle system and the structure of the initial distribution of the N particles in the particle system. Thus, for a fast simulation the simulation method has to be chosen carefully to be suitable for the size and characteristics of the particle input set. For very small particle systems, the direct computation could still be the fastest, since there is no splitting overhead with respect to the cut-off radius. Also, a different simulation method might be suitable for particle systems of different size N which have the same characteristics with respect to the particle distribution. Concerning the initial distribution, the particle systems are often distinguished being homogeneous, inhomogeneous or a mix of both with parts of the particle system being homogeneous but also containing some inhomogeneous regions. While in homogeneous particle systems, the particles are equally distributed in the entire particle system, an inhomogeneous particle system can exhibit distributions in which the particles are clustered in certain areas. Since particle simulations are time-step based algorithms which compute a new particle situation in each step, the structure of a particle system can change during the simulation process. Thus, different simulation methods might be suitable at different points in simulation time so that exchanging the method after some time steps could be beneficial. To support the adaptation of the simulation method, it is required to know in which situation which setting might lead to the desired performance improvement.

Naturally, the hardware platform has a large influence on the performance and accelerators, such as GPUs can be exploited when implementation variants for GPUs, e.g., with CUDA, OpenCL or OpenACC, are available. However, using GPUs requires a transfer of data to the device which might cause a big overhead, and thus is not always advantageous. Depending on the size and structure of the input particle data and availability of hardware, e.g., a CPU or a GPU, the usage of a specific method on specific hardware has to be chosen. For the grid-based methods there exists an OpenCL implementation to use GPUs for the near-field, but these methods are, in consequence of the regular grid over the complete particle system, designed for homogeneous systems. In contrast, tree-based methods are not dependent on a homogeneous system.

In this article, we consider different particle simulation methods of the Scalable Fast Coulomb Solvers (ScaFaCoS) library [3] and study their performance and energy consumption for various particle systems. The ScaFaCoS library contains parallel implementations of efficient solver methods for long-range particle interactions. The parallelizations use the Message Passing Interface (MPI). Additional parallel implementations are available for selected modules. An example is the OpenCL implementation of the near-field module for grid-based methods [4], which allows an execution on various hardware platforms, such as GPUs.

The objective of our work is to investigate the performance and energy consumption behavior of selected particle simulation methods with respect to the characteristics of the input particle system. The contribution of this article includes the detailed measurements and investigation of different algorithms for particle simulation for different input sets and hybrid hardware platforms. The work is meant to provide a rich data basis of performance and energy data for future tuning approaches.

The rest of this article is organized as follows: Section II introduces the particle methods used. Section III describes the generation of the particle systems used. Section IV presents the experimental results. Section V summarizes the performance results. Section VI discusses related work. Section VII concludes the article and discusses the tuning potential.

II. Particle simulation methods

The particle simulation method is a general solution method to simulate all kinds of problems which can be represented by a set of so-called particles which react to each other according to problem-specific rules. For several decades, different versions of particle solution methods have been invented and different implementations of these methods have been developed, all of which solve particle problems but may have a different performance. In this article, we concentrate on particle simulation methods being implemented in the ScaFaCoS library.

A. Particle models

Particle models are simulation models in which physical phenomena are described by a discrete representation of interacting particles. A particle has usually problem specific attributes, such as position, mass, momentum or velocity. The motion in the physical system is calculated in a series of simulation time steps each of which computes one interaction event between the particles by recomputing the attribute values. Such particle models have been used to explain properties of solids, liquids or gases represented by a finite input set of particles and corresponding rules for the specific interaction. Usually, the number of particles is constant for one simulation run, and thus there is no need for updates during one specific simulation. Three principal types of particle simulation models have been identified, which are the particle-particle model using action at a distance, the particle-mesh model using an approximation by a mesh and the P^3M model being a combination of both. The specific use of the simulation type depends on the physical model to be simulated and also on the computational cost for a computer simulation.

In this article, we consider molecular dynamics simulations for Coulomb forces which are characterized by long-range interactions. In the simulation of long-range interactions, the number of interactions per simulation time step is not limited to particles in the proximity. Thus, all pairwise interactions between all particles in the system have to be calculated in each time step, which leads to a computationally expensive simulation. This problem can be treated by hierarchical approximation algorithms reducing the quadratic complexity to a linear complexity or by parallel implementations on different parallel devices. In our investigations, we use MPI implementations but also parallel hybrid implementations on CPU/GPU for particle solvers from the ScaFaCoS library.

B. The ScaFaCoS library

The Scalable Fast Coulomb Solvers (ScaFaCoS) library contains parallel implementations for several different particle simulation methods, e.g., a Direct method, Particle-Particle Particle-Mesh (P^3M), Particle-Particle nonequispaced fast Fourier transforms (P^2NFFT), Fast Multipole Method (FMM) or Pretty Efficient Parallel Coulomb Solver (PEPC). The ScaFaCoS library is fully parallelized using MPI and with certain parts using OpenCL.

The direct computation, e.g., a pairwise interaction between all N particles, requires $O(N^2)$ operations. More efficient methods are reducing this complexity by using approximation approaches which split the calculation into a near-field and a far-field part with respect to each particle. The implementation of this splitting into near- and far-field are different for different particle solution methods. While in the near-field part the pairwise interaction between particles is computed, the far-field part might be computed approximately leading to a more efficient computation. A comparison of the solver methods of the ScaFaCoS library is given in [5].

Besides the accuracy of the computations and also some solver specific parameters, the performance of particle simulations depends on the particle distribution of the particle system. If the particles are equally distributed in the particle system, it is called a homogeneous particle system and usually less expensive to simulate for the particle solvers, in contrast to inhomogeneous particle systems, in which particles are irregularly distributed in the particle system. The input data for the solvers in the ScaFaCoS library describe the corresponding particle system by attributes consisting of the charge value and the three-dimensional position of each particle. In this article, the emphasis is on three of the ScaFaCoS simulation methods, which are the direct method, which is a particle to particle calculation, the P^2NFFT, which is a fourier based approach, and the FMM method, which is tree based to reduce the complexity.

C. Fourier based

Fourier-based methods compute the far-field computations in Fourier space, mostly by using the fast Fourier-transforms (FFT). The method P^2NFFT [2] is used as an example method for the Fourier-based approach. The computational demands of the far-field and the near-field parts are influenced by parameters that specify the size of the FFT grid and the cut-off range. Far-field potentials are computed via convolution in Fourier space. The computation of the near-field interactions is calculated by the ScaFaCoS near-field module, which computes pairwise interactions. The P^{2}NFFT and P^{3}M implementation of the ScaFaCoS library are using the same near-field implementation. An OpenCL implementation for this near-field has been developed to use both Multicore-CPUs and GPUs in [4].

D. Tree based

Another approach to split the particle system is possible by using an octree structure. The particles are sorted into spatial boxes respective to their position in the particle system. The boxes are then organized into an octree which is exploited to compute the interactions on different levels. Since this approach does not split the particle system into a regular box system, it works on homogeneous systems as well as on inhomogeneous input particle systems. The FMM [1] is an example for this approach and has been implemented in ScaFaCoS. This FMM version has its own near-field implementation for which for which an OpenCL version for execution on GPU does not exists.

For a specific particle, the near field potential is determined by calculating the potential at the position of the particle caused by each of the particles in the same and neighboring octree boxes. The far-field potential is calculated using approximate values of the potential caused by all particles in a particular octree box. These approximations are calculated for each octree level. The approximations at appropriate octree levels are then used to approximate the far-field potential at a particular particle position. The tree depth determines the separation in the near-field and the far-field potential and, thus, the tree depth is an important parameter for the accuracy as well as for the performance of a simulation run.

III. Generation of particle systems

The particle systems used have particles that are Hammersley distributed [6] to ensure a minimal space between the particles. All tested particle systems are periodical, i.e., if a particle leaves the particle system, a new particle enters the system on the opposite side.

The Hammersley distributed particle systems are generated with the formulas given in this section as described in [7].

If p is a prime number, each nonnegative number k can be displayed as:

$$k = a_0 + a_1 p + a_2 p^2 + \cdots + a_r p^r \tag{1}$$

with $a_i \in \{0, \dots, p-1\}, \quad i = 0, \dots, r, \quad r \in \mathbb{N}$.

A function $\phi(k)$ can be defined as follows:

$$\phi_p(k) = \frac{a_0}{p} + \frac{a_1}{p^2} + \frac{a_2}{p^3} + \cdots + \frac{a_r}{p^{r+1}} \tag{2}$$

The following explains the Hammersley distribution:

We define d as the dimension of the data to be generated and $p_1, p_2, \dots, p_{d-1}$ the prime numbers with $p_1 < p_2 < \cdots < p_{d-1}$. N is the number of particles to be generated. The particle k is defined as follows:

$$\left(\frac{k}{N}, \phi_{p_1}(k), \dots, \phi_{p_{d-1}}(k)\right), k = 0, \dots, N-1 \tag{3}$$

Since the first component of particle k depends on N , the number of particles has to be set before the generation starts.

The particle systems used are distributed in a cube of size $[0,1]^3$ with different Hammersley distributions.

There exists an implementation named HAMMERSLEY[1] which can provide different Hammersley distributions, e.g., Ball, Two Balls, Grid Face and Cube as shown in Figure 1. The four different distributions used in this article are generated as follows:

- **Cube:** The particles are Hammersley distributed in the whole particle cube.
- **Grid Face:** The number of particles is $N = 4 \cdot N_c^3, N_c \in \mathbb{N}$ and j is defined as:

$$\begin{aligned} j &= (N_c \cdot u + v) \cdot N_c + w \\ &(u, v, w \in \{0, \dots, N_c - 1\}) \end{aligned} \tag{4}$$

  The positions of the particles x_{4j+1} to x_{4j+4} are then defined as follows:

$$\begin{aligned} x_{4j+1} &= (u, v, w)^T / (N_c - 0.5) \\ x_{4j+2} &= (u + 0.5, v + 0.5, w)^T / N_c - 0.5 \\ x_{4j+3} &= (u + 0.5, v, w + 0.5)^T / N_c - 0.5 \\ x_{4j+4} &= (u, v + 0.5, w + 0.5)^T / N_c - 0.5 \end{aligned} \tag{5}$$

- **Ball:** Inside of the particle cube, the particles are Hammersley distributed in the following ball:

$$(x - 0.5)^2 + (y - 0.5)^2 + (z - 0.5)^2 <= (0.5)^2 \tag{6}$$

- **Two Balls:** Two Balls of different sizes are created as balls as in Formula 6. The first Ball with $(1 - 1/64) \cdot N$ particles and the second with $1/64 \cdot N$ particles. They get a distance of 20 and are acurately scaled and shifted in the particle cube.

The charge $q_i \in \{-1; 1\}$ of each particle i is generated randomly, such that the following holds:

$$\sum_{i=1}^{N} q_i \in \{-1; 0; 1\} \tag{7}$$

To use the generated particle systems with the ScaFaCoS test program we used, they have to be converted into XML files as input data, containing the position and the actual charge of each particle.

IV. Performance results

The performance of different solvers is tested for various particle systems, i.e., two homogeneous and two inhomogeneous systems. For the Fourier-based solver also the OpenCL variant is tested. The experiments are split by the particle system distributions. To look at the solvers in more detail, the Fourier-based algorithms are executed with different solver specific parameter settings.

[1]HAMMERSLEY. The Hammersley Quasirandom Sequence. people.scs.fsu.edu/~burkardt/cpp_src/hammersley/hammersley.html

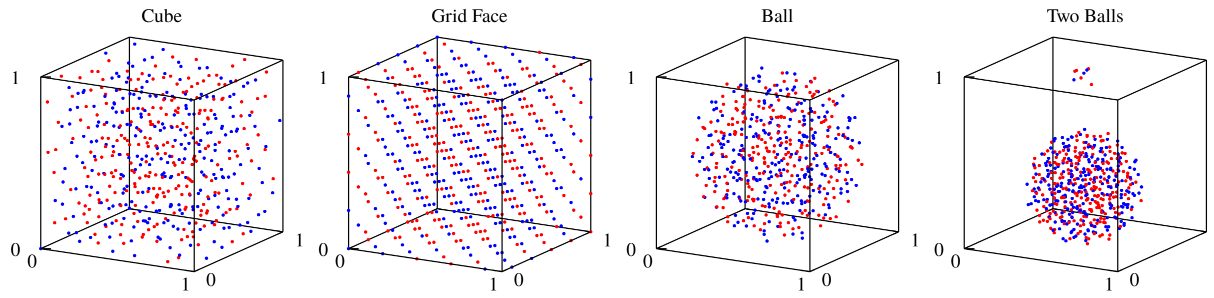


Figure 1. Illustration of two homogeneous particle distributions (left) and two inhomogeneous particle distributions (right) with positive and negative charges.

A. *Experimental setup*

The experiments are performed on a multicore system with four GPUs. The Haswell system consists of two Intel Xeon E5-2683 v3 processors with 14 cores each, which have 2.0 GHz. The system is equipped with four Nvidia GeForce GTX Titan Blacks. The energy consumption is measured using PAPI 5.6.0 and the RAPL interface to read the appropriate MSR registers. The energy measurements do only include the CPU, i.e., it does not include the energy consumption of the DRAM or any other component in the system. The energy consumption of the GPUs is not measured. All measurements are repeated 5 times to obtain the shown average values. For the measurements, the number of MPI processes is set to 56, which is equal to the number of cores on the Haswell system plus Hyperthreading. The frequency is set to 2.0 GHz and Intel Turbo Boost is disabled for the experiments.

B. *Homogeneous systems*

In homogeneous particle systems, the particles are uniformly distributed in the particle system without irregularities. The particles are not grouped into multiple clusters. Figure 1 (left) shows the two homogeneous particle systems Cube and Grid Face. The positive and negative charges are randomly generated.

Figure 2 shows the runtime and energy consumption for the FMM and P^{2}NFFT (MPI and OpenCL variant) solver with the two homogeneous systems with varying number of particles. For small particle systems, i.e., less than 50,000 particles, the P^{2}NFFT solver has a low runtime. If the particle system has more particles, the FMM solver outperforms the P^{2}NFFT. Since the transfer to the GPU takes some time, it is only useful to use the GPU with big particle systems. However for the big systems, the FMM algorithm outperforms the P^{2}NFFT even when GPUs are used. If the CPU would be less powerful, the OpenCL variant of the P^{2}NFFT should outperform the MPI variant of the FMM with bigger particle systems. The energy consumption shows the same behaviour as the runtime for homogeneous systems.

Figure 3 shows parameter tests for the P^{2}NFFT solver (MPI variant) for the two homogeneous particle systems with different system sizes. The grid size is varied from 128 to 512 to determine the best grid size for each system. The more particles a system has, the bigger the best grid size. Since there is only one global minimum and no other local minimum, simple tuning algorithms can be used to find that minimum. This minimum can be different for the runtime and the energy consumption. As the 50,000 particle system shows, the explicit best grid size can differ with the particle system structure size, e.g., 512 for the Grid Face but 448 for the Cube.

C. *Inhomogeneous systems*

In inhomogeneous particle systems, particles are irregularly distributed in the particle system. For example, they are clustered in single or multiple regions, thus, there are also empty regions in the field. Figure 1 (right) shows two inhomogeneous particle systems. The left particle system is a single big ball in the centre of the system, while the right particle systems is a dense ball and a smaller additional ball with less particles in distance. The right system has a bigger empty region in the particle system than the left system.

Figure 4 tests the two inhomogeneous particle systems with varying particle system sizes. As expected, the P^{2}NFFT algorithm has more problems, in terms of runtime and energy consumption, with inhomogeneous systems compared to homogeneous systems. The Ball particle system has a similar behaviour like the homogeneous systems, but with the Two Balls system P^{2}NFFT has a worse runtime and energy consumption compared to FMM, even with few particles. The GPU variant of the P^{2}NFFT has a better runtime for big particle systems than the MPI variant but is still slower than the FMM method using MPI. Like with homogeneous systems, the energy consumption shows the same behaviour. Consequently, the Ball particle system is homogeneous enough for the P^{2}NFFT algorithm, but for more inhomogeneous systems, like the Two Balls system, the FMM algorithm is better.

Figure 5 shows parameter tests for the P^{2}NFFT system for the two inhomogeneous particle systems with different system sizes. The grid size is varied from 128 to 512 to determine the best grid size for the particle system. The more particles a system has, the bigger the best grid size. The Two Balls system with 5,000 particles shows that the runtime and energy consumption can have different settings, i.e., grid size of 384 for the shortest runtime, but 448 for the lowest energy consumption.

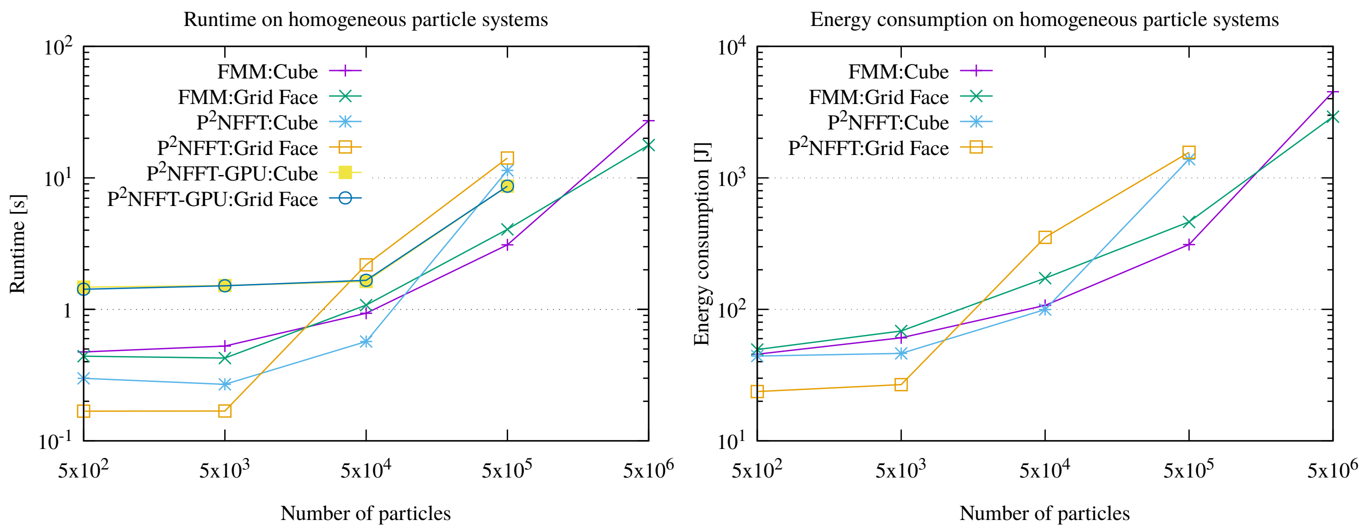


Figure 2. Parallel runtime (left) and energy measurements (right) for homogeneous systems with 56 MPI processes on the Haswell system and the Geforce GTX Titan Black.

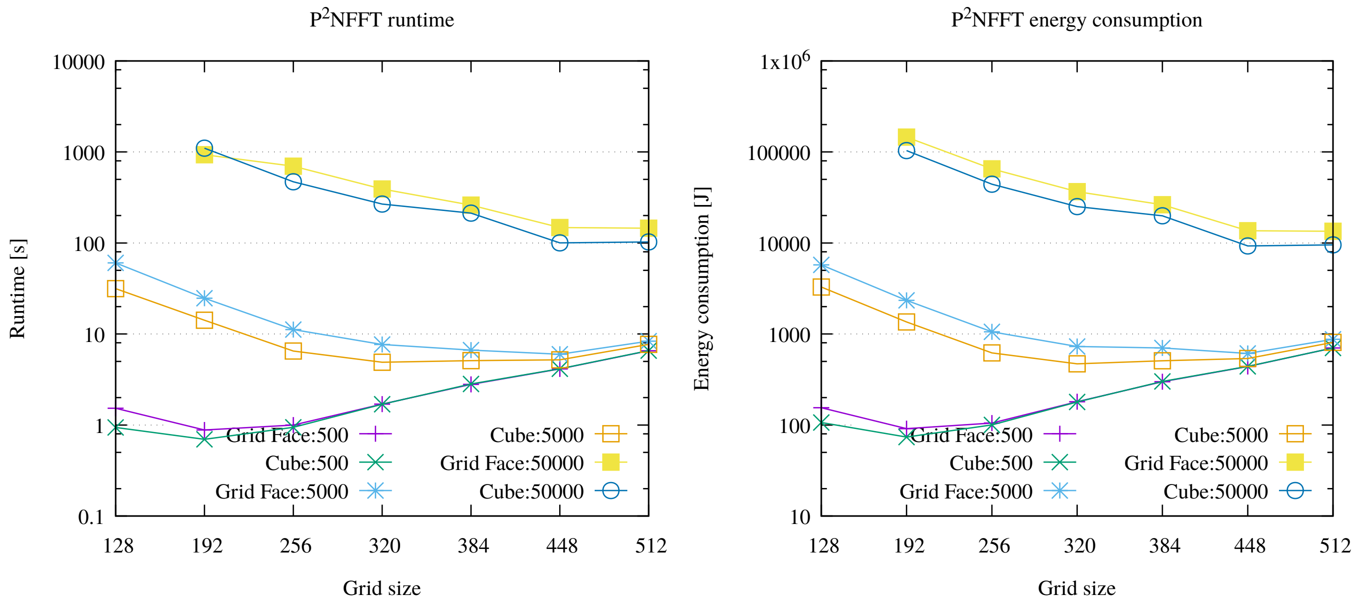


Figure 3. Parallel runtime (left) and energy measurements (right) for homogeneous systems with varying grid size with 56 MPI processes on the Haswell system.

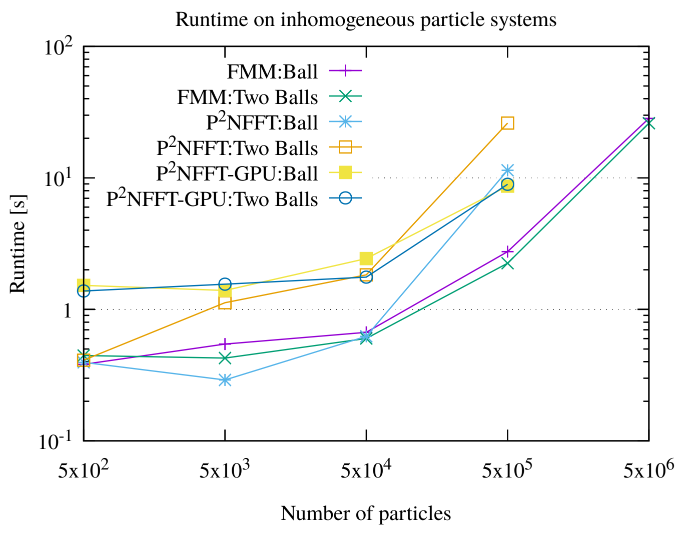
V. Summary of the Performance Results

The measurement results from Section IV have shown that the performance of the different particle simulation implementations strongly depend on the execution platform as well as on the characteristics of the input particle system. Depending on the optimizing goal, the availability of the hardware and the prior knowledge of the particle system distribution and size, some decisions can be made to achieve the best performance or lowest energy consumption. Thus, based on the measurements an appropriate particle simulation method can be selected. The following observation show how it can be decided whether the FMM, the P^{2}NFFT or the P^{2}NFFT with GPU solver should be used for best performance. Selection strategies might help to select the simulation algorithm with the best execution time and/or energy consumption.

Table I
Solver Selection

less than 50,000 particles	
homogeneous distribution	inhomogeneous distribution
P^{2}NFFT solver	FMM solver
more than 50,000 particles	
strong CPU	weak CPU but GPU available
FMM solver	P^{2}NFFT solver on GPU

Table I summarizes the selection of a particle simulation solver for the given HPC system. If the particle system has less than 50,000 particles, the selection of the best performing



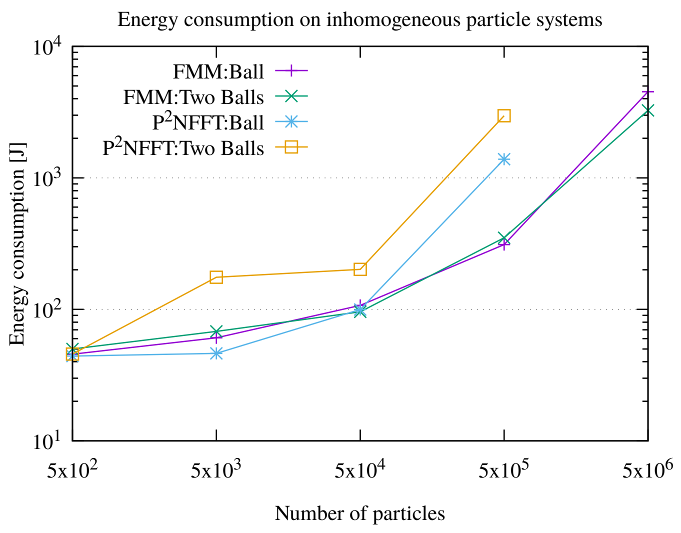
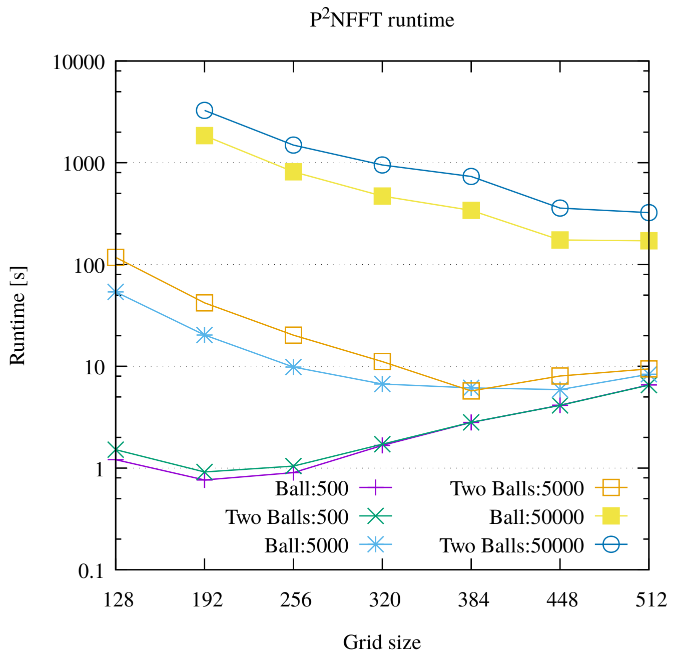


Figure 4. Parallel runtime (left) and energy measurements (right) for inhomogeneous systems with 56 MPI processes on the Haswell system and the Geforce GTX Titan Black.



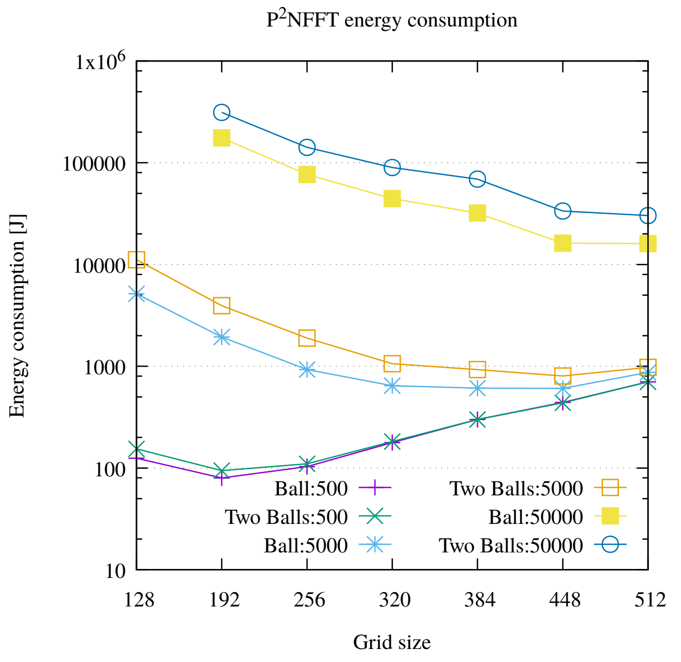


Figure 5. Parallel runtime (left) and energy measurements (right) for inhomogeneous systems with varying grid size with 56 MPI processes on the Haswell system.

solver depends on the distribution of the particle system, e.g. P^{2}NFFT for homogeneous distributions and FMM for inhomogeneous distributions (Two Balls particle system). For particle systems with more than 50,000 particles, the FMM solver is the best performing one if only CPUs are available. For a system with a less performing CPU, e.g., a single core CPU, the P^{2}NFFT OpenCL implementation on GPUs has the best performance for particle systems with more than 50,000 particles.

Usually, the characteristic of the particle input system is known before the execution starts, and measurement and evaluation results such as described above can help to start the most efficient simulation algorithm. However, there might be cases in which it is not *a priori* clear which characteristic the distribution of the input data might have. In these situations, it is possible to execute one time-step with each solver to measure the performance and then the results are compared to select the best one. In cases in which the solver specific parameters, e.g., the grid size, differ too much for the particle system distribution and size, the solvers have to be tested for several time-steps before the best one can be chosen.

In summary, the investigations of this article have shown that the performance results of the different simulation methods can differ for different particle distribution characteristics and different hardware, but that some behavior classes can be detected. This shows that there is a potential for designing tuning strategies based on a larger data basis.

VI. Related Work

Performance analysis and prediction of a particle simulation method was examined in [8]. As test system they used the SB-PRAM, a shared memory machine with up to 2048 processors.

Many implementations of the FMM approach invented in [1] exist and contain specific optimisations for the actual execution run. In [9] a parallel sorting for the particles in the particle systems is presented which improves the locality of interacting particles for computation on a distributed memory architecture. A more application specific optimization has been presented in [10], which introduces a method for automatic tuning of the FMM by selecting the optimal FMM tree depth based on an integrated performance prediction of the FMM computations.

The autotuning potential of particle simulation methods from the ScaFaCoS library are examined in [11] and [12]. In these articles, only one particle distribution is considered. In our article we consider different distributions of the particle systems and additionally examine an OpenCL solver. The OpenCL solver used is introduced and tested in [4].

In [13] and [14] autotuning strategies are introduced for different N-Body simulations on heterogeneous and hybrid CPU/GPU systems. The focus of these articles is on load balancing the GPUs, and thus less on CPU performance and energy consumption. The authors of [7] investigated two different ScaFaCoS solvers on different particle systems. We extended their work with a GPU solver and investigated different particle systems.

VII. Conclusions

The investigations of this article have shown that varying particle system distributions and sizes have a significant impact on the execution time and energy consumption. Using GPUs with the OpenCL implementation is useful when the CPU performance of the system is limited. The results show that some decisions can be made before runtime, but others, e.g., the solver specific parameters, have to be tested during runtime. For each particle simulation execution, the availability of hardware and the size of the particle system are fixed, but the distribution of the particles may change after some time steps, and thus a different particle simulation method could then be the best. Experiments have shown that the solver specific parameters, e.g., the grid size, have different optimal settings for different distributions and sizes. Thus, the performance results must be checked and compared to the values last checked. This can be done by monitoring during runtime. Our observations show that it is necessary to use both tuning approaches to tune runtime or energy consumption, an offline tuning to set the start parameters as well as possible, and an online approach to fine-tune the parameters, e.g., the solver specific parameters, and to respond to particle distribution changes.

Acknowledgment

This work is supported by the German Ministry of Science and Education (BMBF) project Selbstadaption für zeitschrittbasierte Simulationstechniken auf heterogenen HPC-Systemen (SeASiTe), Grant No. 01IH16012B.

References

[1] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. of Computational Physics*, vol. 73, pp. 325–348, 1987.

[2] M. Pippig and D. Potts, "Parallel three-dimensional nonequispaced fast fourier transforms and their application to particle simulation," *SIAM J. on Scientific Computing*, vol. 35, no. 4, pp. C411–C437, 2013. doi: 10.1137/120888478

[3] M. Bolten, F. Fahrenberger, R. Halver, F. Heber, M. Hofmann, I. Kabadshow, O. Lenz, M. Pippig, and G. Sutmann, "ScaFaCoS, C subroutine library," `http://scafacos.github.com/`. [Online]. Available: http://scafacos.github.com

[4] M. Hofmann, R. Kiesel, D. Leichsenring, and G. Rünger, "A hybrid cpu/gpu implementation of computationally intensive particle simulations using opencl," in *2018 17th International Symposium on Parallel and Distributed Computing (ISPDC)*, June 2018. doi: 10.1109/ISPDC2018.2018.00011 pp. 9–16.

[5] A. Arnold, F. Fahrenberger, C. Holm, O. Lenz, M. Bolten, H. Dachsel, R. Halver, I. Kabadshow, F. Gähler, F. Heber, J. Iseringhausen, M. Hofmann, M. Pippig, D. Potts, and G. Sutmann, "Comparison of scalable fast methods for long-range interactions," *Physical Review E*, vol. 88, p. 063308, 2013.

[6] J. M. Hammersley, "Monte carlo methods for solving multivariable problems," *Annals of the New York Academy of Sciences*, vol. 86, no. 3, pp. 844–874, 1960. doi: 10.1111/j.1749-6632.1960.tb42846.x. [Online]. Available: https://nyaspubs.onlinelibrary.wiley.com/doi/abs/10.1111/j.1749-6632.1960.tb42846.x

[7] M. Pippig and D. Potts, "Particle simulation based on nonequispaced fast fourier transforms," 01 2011, pp. 131 – 158.

[8] T. Rauber, G. Rünger, and C. Scholtes, "Execution Behavior Analysis and Performance Prediction for a Shared-Memory Implementation of an Irregular Particle Simulation Method," *Simulation: Practice and Theory*, vol. 6, no. 7, pp. 665–687, 1998. doi: 10.1016/S0928-4869(98)00006-8

[9] H. Dachsel, M. Hofmann, and G. Rünger, "Library Support for Parallel Sorting in Scientific Computations," in *Proceedings of the 13th International Euro-Par Conference*, ser. LNCS, vol. 4641. Springer, August 2007. doi: 10.1007/978-3-540-74466-5_73. ISBN 978-3-540-74465-8 pp. 695–704.

[10] H. Dachsel, M. Hofmann, J. Lang, and G. Rünger, "Automatic Tuning of the Fast Multipole Method Based on Integrated Performance Prediction," in *Proceedings of the 14th IEEE International Conference on High Performance Computing and Communications (HPCC-2012)*. IEEE, Juni 2012. doi: 10.1109/HPCC.2012.88. ISBN 978-1-4673-2164-8 pp. 617–624.

[11] N. Kalinnik, R. Kiesel, T. Rauber, M. Richter, and G. Rünger, "On the Autotuning Potential of Time-stepping methods from Scientific Computing," in *Proceedings of the 2018 Federated Conference on Computer Science and Information Systems (FedCSIS 2018), 11th Workshop on Computer Aspects of Numerical Algorithms (CANA'18)*, vol. 15. ACSIS, September 2018. doi: 10.15439/2018F169. ISSN 2300-596 pp. 329–338.

[12] M. Hofmann, R. Kiesel, and G. Rünger, "Energy and Performance Analysis of Parallel Particle Solvers from the ScaFaCoS Library," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE 2018)*. ACM, April 2018. doi: 10.1145/3184407.3184409. ISBN 978-1-4503-5095-2 pp. 88–95.

[13] R. Yokota and L. Barba, "Hierarchical N-body Simulations with Autotuning for Heterogeneous Systems," *Computing in Science Engineering*, vol. 14, no. 3, pp. 30–39, May 2012. doi: 10.1109/MCSE.2012.1

[14] M. Holm, S. Engblom, A. Goude, and S. Holmgren, "Dynamic Autotuning of Adaptive Fast Multipole Methods on Hybrid Multicore CPU and GPU Systems," *SIAM Journal on Scientific Computing*, vol. 36, no. 4, pp. C376–C399, Jan. 2014. doi: 10.1137/130943595. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/130943595