Original published: 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)*, pages 617–624. IEEE, 2012. Online available at http://dx.doi.org/10.1109/HPCC.2012.88.

Automatic Tuning of the Fast Multipole Method Based on Integrated Performance Prediction

Holger Dachsel[1] *h.dachsel@fz-juelich.de*
Michael Hofmann[2] *michael.hofmann@cs.tu-chemnitz.de*
Jens Lang[2] *jens.lang@cs.tu-chemnitz.de*
Gudula Rünger[2] *ruenger@cs.tu-chemnitz.de*

[1] Institute for Advanced Simulation, Jülich Supercomputing Centre, Forschungszentrum Jülich, Jülich, Germany
[2] Department of Computer Science, Chemnitz University of Technology, Chemnitz, Germany

***Abstract*—The Fast Multipole Method (FMM) is an efficient, widely used method for the solution of N-body problems. One of the main data structures is a hierarchical tree data structure describing the separation into near-field and far-field particle interactions. This article presents 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 prediction method exploits benchmarking of significant parts of the FMM implementation to adapt the tuning to the specific hardware system being used. Furthermore, a separate analysis phase at runtime is used to predict the computational load caused by the specific particle system to be computed. The tuning method was integrated into an FMM implementation. Performance results show that a reliable determination of the tree depth is achieved, thus leading to minimal execution times of the FMM algorithm.**

I. Introduction

The *Fast Multipole Method* (FMM) [1] is a popular method for solving the *N-body problem*, which calculates interactions between a set of particles, e.g. Coulomb forces between charges or gravitational forces between masses. Solving the N-body problem for systems of thousands to millions of particles is an essential part of simulations in the fields of molecular dynamics, astrophysics, electrostatics and many more. The significance of the Fast Multipole Method in today's research is shown by the existence of many implementations for different platforms, e.g. for massively parallel distributed memory machines [2], for GPUs [3], and for clusters of GPUs [4]. The competitiveness of an FMM implementation in comparison to other solution methods strongly depends on the optimal choice of several FMM specific parameters. These parameter settings are influenced by the computing capabilities of the hardware system being used as well as by the specific particle problem to be solved. This leads to ongoing efforts to further increase the efficiency of the FMM based on both algorithmic and programming levels.

The direct calculation of interactions between n particles has an execution time $\mathcal{O}(n^2)$ since each particle interacts with each other. The FMM achieves an execution time $\mathcal{O}(n)$ by dividing the interactions into *near-field interactions* and *far-field interactions* and calculating them separately. Interactions between particles in close proximity represent the near-field interactions and their contributions are still calculated directly. Interactions between distant particles represent the far-field interactions and are calculated by dividing the particles hierarchically into groups based on the spatial positions of the particles. The contributions of particles within the same group are then approximated using multipole expansions such that the overall far-field interactions can be calculated with a significantly smaller number of interactions between whole groups of particles. The hierarchical subdivision of particles into groups is represented by a tree data structure where the chosen depth of the FMM tree determines the separation into near-field and far-field interactions.

Although the time complexity of the FMM is already very competitive, the actual execution time can vary due to implementation details. Our goal is to provide very fast implementations, especially for large simulations and for particle systems with homogeneous as well as arbitrary inhomogeneous particle distributions. The work is based on the FMM implementation presented in [5], incorporating a two-stage error estimation scheme to achieve a reliable control of the errors induced by the usage of finite multipole expansions. The contribution of this article is to present a method for determining the depth of the FMM tree automatically in such a way that the overall execution time of the near-field and far-field computations achieves a minimum. Determining this optimal tree depth is based on a precise performance prediction method integrated into the FMM implementation. A benchmark phase is used to adapt the parameters of the performance prediction to the actual hardware system being used. At runtime of the FMM algorithm, an additional analysis phase is performed to predict the load of the different computational parts of the FMM depending on the specific particle system to be computed. Experimental results for different particle systems, e.g. from a laser plasma simulation, show the performance improvements achieved when exploiting the automatic tuning method.

The rest of this article is organized as follows: Section II gives an overview of the Fast Multipole Method. Section III describes the determination of the optimal FMM tree depth. Experimental results are presented in Sect. IV. Related work is discussed in Sect. V, and Sect. VI concludes the article.

II. The Fast Multipole Method

The Fast Multipole Method (FMM) uses a hierarchical subdivision of the particle system to achieve a separation of the particle interactions into near-field interactions and far-field interactions. In the following, it is assumed that all particles are located inside a unit cube called *system box* and that the FMM is used to calculate the total energy of the system of interacting particles.

The main data structure of the FMM is an octree whose nodes represent boxes of particles resulting from the hierarchical subdivision of the system box on different levels. The root node of the octree at level 0 represents the overall system box. Boxes at level 1 of the octree represent the eight *child boxes* of the system box that are created by dividing the system box into two halves along each of the three dimensions. This subdivision is repeated recursively with all newly created boxes until an optimal tree depth d of the octree is reached. The boxes of the last level d are called *leaf boxes*. Each leaf box contains only a small number of particles, which is usually less than 10.

After the construction of the octree and the subdivision of particles into boxes, the FMM algorithm is performed with the following five passes:

I. In the first pass, pseudo particles are built: All particles within the same leaf box are treated as being concentrated in one pseudo particle, which is located at the centre of the leaf box. To create a pseudo particle, a multipole expansion is used. The length of the multipole expansion, i.e. the number of multipole moments used, has an influence on the precision and on the computational costs of the FMM. Starting with the multipole expansions of the leaf boxes, new pseudo particles for boxes up to level 2 are created by shifting the multipole expansions of child boxes towards the centres of their parent boxes. This pass is also called the multipole-to-multipole translation.

II. The second pass computes box-box interactions using a well-separated criterion: Two boxes are well separated if and only if they are not adjacent, that means they share neither an edge nor a corner. Box-box interactions are performed for each box with every other box of the same level that is well separated to the box itself, but whose parents are not well separated. Each of these box-box interactions is performed by translating the existing multipole expansions of the source box into a local expansion of the interacting target box. This pass is also called the multipole-to-local translation and involves all boxes up to level 2. The system box at level 0 and their child boxes at level 1 do not have any interacting boxes in this scheme.

III. In the third pass, the contributions from the box-box interactions are distributed to the leaf boxes. This is performed by shifting the local expansions of parent boxes towards the centres of their child boxes and adding the shifted expansions to the existing local expansions of the child boxes. This pass is also called the local-to-local translation. The overall translation starts at the first level which has well-separated boxes, i.e. level 2, and ends at the leaf boxes.

IV. The contributions of the far-field interactions to the total energy of the system are calculated by evaluating the local expansion of each leaf box.

V. The near-field interactions are calculated directly between pairs of particles and added to the total energy of the system. These particle-particle interactions are performed between particles inside each leaf box and between particles of leaf boxes that are not well-separated.

The number of interactions performed in the far-field passes I–IV and in the near-field pass V depend on the tree depth of the FMM. A low tree depth results in leaf boxes with a large number of particles and therefore the FMM algorithm consists mainly of particle-particle interactions in the near-field. With increasing tree depth, the number of boxes in the octree increases and therefore also the number of box-box interactions in the far field increases. Additionally, the size of the leaf boxes decreases and therefore also the number of particle-particle interactions decreases. In order to have a fast simulation code for the FMM, the optimal tree depth of the octree has to be chosen such that the accumulated execution time of the near-field and the far-field computations achieves a minimum. Pass II and pass V represent the most time-consuming parts of the FMM algorithm and are used in the following to determine the optimal tree depth.

Computing a single particle-particle interaction directly in pass V consists only of few instructions for determining the inverse distance of the two interacting particles. A single box-box interaction of pass II represents a complex translation operator of the FMM algorithm. The computational costs of this operator depend on the length of the multipole expansions used, which is governed by the maximum error bound request for the FMM computations. The specific computational costs of both operations depend on the actual code generated by the compiler and the capabilities of the actual processor used. Thus, these computational costs represent platform-specific parameters for determining the optimal tree depth.

The exact number of particle-particle interactions and box-box interactions for a chosen tree depth depends on the total number of particles and on their distribution within the system box. With homogeneous particle distributions, there is approximately the same number of particles in all boxes of the same level. Thus, the number of interactions for each tree depth can be predicted very well in these cases. For inhomogeneous particle distributions, the prediction of the number of interactions is still a challenge. Due to the varying number of particles per box, the number of particle-particle interactions cannot be predicted a priori. Boxes without particles lead to a decreasing number of box-box interactions. Furthermore, since empty boxes do not need to be considered, the whole tree can be unbalanced leading to irregular execution times for traversing the tree during the FMM passes. Thus, the actual numbers of particle-particle interactions and box-box interactions are problem-specific parameters that depend strongly on the individual particle system that is computed.

III. Determining the optimal tree depth

The proposed method for determining the optimal tree depth for the FMM algorithm is based on an integrated prediction of the performance of far-field and near-field computations for different tree depths. This performance prediction consists of two steps: First, several *benchmark runs* of the FMM algorithm are performed to determine platform-specific parameters, such as the computational costs of the FMM operators, for the specific hardware and compiler settings used. Second, before each computational run of the FMM for a specific particle system, a separate *analysis run* is performed to determine problem-specific parameters, such as the number of interactions required for each tree depth. In the following, the tree traversal algorithm of the FMM is sketched and the benchmark and analysis runs required for the integrated performance prediction are described.

A. Tree traversal for far-field and near-field computations

```
1  loop over tree levels l = 2 ... d
2      loop over all boxes b of level l
3          find parent box p of box b
4          if level l = last level d then
5              compute particle-particle interactions in
                 box b and between box b and all other
                 child boxes of parent p
6          loop over neighbour boxes n of parent p
7              loop over child boxes c of neighbour n
8                  if boxes c and b are well separated
                     then
9                      compute box-box interaction
                         between boxes c and b
10                 else if level l = last level d then
11                     compute particle-particle
                         interactions between particles in
                         boxes c and b
```

Algorithm 1: Tree traversal for pass II and pass V of the FMM algorithm using tree depth d.

Algorithm 1 shows the nested loop structure that describes the traversal of the octree to compute box-box interactions and particle-particle interactions. This algorithm represents a general scheme that is used during the benchmark runs and the analysis run as well as for the computational run of the FMM. However, in the computational run of FMM, the computation of box-box interactions and particle-particle interactions is performed in two separate passes as described in Sect. II. This separation leads to a better data locality as computing the box-box interactions in pass II accesses only box data and computing the particle-particle interactions in pass V accesses only particle data.

The algorithm assumes that the tree depth d is greater than 1, because otherwise the whole particle system is computed directly with particle-particles interactions only. The first loop in Alg. 1 (l-loop in line 1) iterates from level 2 to the last level d of the octree. Within the l-loop, the b-loop in line 2 iterates over all boxes b of the current level l. For each box b within the b-loop, the parent box p is determined and then the interactions are computed. If the current level l is the last level d of the octree, then box b is a leaf box and particle-particle interactions are computed for all particles in box b as well as between particles in box b and all other child boxes of parent p (line 5). The n-loop in line 6 iterates over the 26 neighbour boxes n surrounding the parent box p. This is implemented by three nested loops, i.e. one for each direction x, y, and z. If box p is located at the border of the system box, then neighbour boxes outside the system box are skipped such that fewer than 26 neighbour boxes are considered. The inner c-loop iterates over all child boxes of a neighbour box n and checks whether a child box c is well separated from box b or not. If boxes c and b are well separated, then a box-box interaction is performed (line 9). Otherwise, if the current level l is the last level d of the octree, then particle-particle interactions are computed between the particles in the leaf boxes c and b (line 11).

Computing the interactions in lines 5, 9, and 11 usually requires a major part of the execution time such that it is sufficient to consider only these computations for an accurate performance prediction. However, if the requested error bound leads to a small number of multipole moments or if there exist large numbers of empty boxes due to inhomogeneous particle distributions, all statements of Alg. 1 need to be considered. This includes the varying overheads of the nested loop structures as well as the different octree operations for determining parent, neighbour, and child boxes.

B. Benchmark runs

The benchmark runs are used to determine the computational costs for computing single box-box and particle-particle interactions as well as for executing single iterations of the different loops of Alg. 1. The results of these measurements for a specific platform are integrated into the FMM program and are later used as platform-specific cost parameters to determine the optimal tree depth of the FMM during the analysis run. For the benchmark runs, hardware performance counters are accessed through the PAPI library to measure the cost parameters in terms of machine instructions executed by the processor. Furthermore, in Sect. IV results for measuring both machine instructions and clock cycles are presented and their influence on the performance prediction is investigated.

The number of machine instructions required to compute a single box-box interaction depends on the length of the multipole expansions chosen in order to adhere the requested error bound. Since this error bound is not known during the benchmark run, all reasonable expansion lengths from 0 to 50 multipole moments are benchmarked. For each multipole length, the translation operator from pass II is executed 1000 times with random input data and the number of instructions measured with the hardware performance counter is used to calculate the instruction count for a single execution.

The instruction counts for a single particle-particle interaction as well as for single iterations of the loops of Alg. 1 are determined by running the FMM algorithm with an inhomogeneous particle system as test data set. The particles of the test system are concentrated in the centre of the system box, thus leading to large numbers of empty boxes as well as to strong variations in the number of particles per box. A more detailed description of this Xenon particle distribution is given in Sect. IV. Using test data with these properties ensures that all existing loops and branches within the tree traversal algorithm are executed several times such that their instructions can be measured by the hardware performance counters.

Determining the instruction counts of the nested loop iterations is performed with a separate benchmark run for each loop. All these runs execute Alg. 1 with a fixed tree depth of $d = 8$ and with the computations of particle-particle and box-box interactions in lines 5, 9, and 11 disabled. The hardware performance counters are used to count the overall instructions executed between the beginning and the ending of the l-loop in each run. By proceeding from the outermost loop to the innermost loops, the b-loop, n-loop, and the c-loop are considered one after another. For the benchmark run of each loop, all inner loops are disabled and the overall number of loop iterations is determined using a counter variable.

The first benchmark run determines the overall number of iterations and instructions executed by the b-loop, thus leading to the instruction count for a single iteration of the b-loop. The next benchmark run determines the overall number of iterations of the n-loop and the overall number of instructions executed by the b-loop and the n-loop. However, since the overall number of instructions of the b-loop is already known from the previous benchmark run, the results are used to calculate the instruction count for a single iteration of the n-loop. Since the n-loop is implemented as three nested loops, two further benchmark runs are performed for the n-loop followed by a benchmark run for the c-loop. After considering all nested loops, a last benchmark run is used to determine the instruction count for a single particle-particle interaction. This is achieved by running the separate near-field pass, counting the instructions and particle-particle interactions, and using the results to calculate the instruction count for a single particle-particle interaction. The overall result of the benchmark runs are specific values for the cost parameters that describe the instruction counts for single box-box and particle-particle interactions as well as for single iterations of the nested loops.

C. Analysis run

Before each computational run of the FMM algorithm, the given particle system is analyzed to determine the optimal tree depth. Furthermore, this analysis is also used to implement an error-control mechanism that determines the multipole length required to adhere the requested error bound for the given particle system [5]. The analysis run executes Alg. 1 with a maximum tree depth $d = 20$ and with counter variables added to accumulate the numbers of iterations of each loop. The computation of box-box interactions (line 9) is disabled and a counter variable is added to accumulate the numbers of box-box interactions over all levels l of the l-loop. The computation of particle-particle interactions (lines 5 and 11) is disabled and a counter variable is added to count the number of particle-particle interactions for each level l as if it were the last level with the leaf boxes. A single execution of this modified variant of Alg. 1 is then used to predict the performance of the FMM algorithm with different tree depths.

The overall octree data structure required for the analysis run is created incrementally by adding a new level at the beginning of each iteration of the l-loop. After each iteration of the l-loop, the current values of the counter variables and the cost parameters determined during the benchmark runs are used to predict the total number of instructions required for performing the FMM algorithm with the current level l as the depth of the octree. If this total instruction count is smaller then the total instruction count of the previous iteration of the l-loop, then the l-loop is continued. However, if the total instruction count is larger than the previous one, then the performance prediction has discovered the minimum number of instructions, thus leading to a maximum performance of the FMM algorithm. In this case, the analysis run is stopped and the level l associated to the minimum number of total instructions represents the optimal tree depth for the following computational run of the FMM. Since the number of particle-particle interactions always decreases with increasing level l and the number of box-box interactions always increases with increasing level l, there can not exist local minimums for the total number of instructions.

IV. Results

This section presents the experiments that were performed to evaluate the proposed performance prediction method for determining the optimal tree depth of the FMM. This includes a comparison between using machine instructions and clock cycles for the performance prediction. Furthermore, the influence of the nested loops within the prediction is investigated and the precision of the prediction as well as their computational overhead within the FMM implementation itself is determined.

A. *Experimental setting*

For the experiments, three different particle distributions were used. The *homogeneous* distribution consists of 8^6 particles distributed regularly on a lattice in the unit cube. The *concentric spheres* distribution is composed of 8 concentric spheres of different size, each consisting of 8^5 particles distributed homogeneously on the surface of the sphere. The *Xenon* distribution consists of 114 537 particles representing a Coulomb explosion of Xenon, i.e. when matter transits into plasma exited by a focused laser. The Xenon distribution has many particles concentrated in the centre while regions towards the border of the system are very sparsely populated. For example, half of the particles are concentrated in 0.025 % of the system box. This results in a very inhomogeneous distribution with a large number of empty boxes during the FMM algorithm.

The FMM implementation is a highly optimized Fortran code. All following measurements were performed using the GNU Fortran compiler with optimization flag `-O0`. Two different hardware systems were used: an Intel system with a 2.67 GHz Xeon X5650 processor and 12 GB RAM, and an AMD system with a 1.9 GHz Opteron 8347 processor and 16 GB RAM. The PAPI library was used to access the hardware performance counters of the processors using PAPI events `PAPI_TOT_INS` and `PAPI_TOT_CYC`.

B. *Comparison between instruction and cycle prediction*

Figure 1 (top) compares the number of instructions predicted in the analysis run with the measured number of instructions during the computational run of the FMM depending on the tree depth. The results were obtained for computing the inhomogeneous Xenon particle distribution with an requested relative error bound of 10^{-3} using the Intel system. The results show that the presented performance prediction method determines the number of instructions of the far-field computations very well for all tree depths. Predicting the instructions of the near-field computations matches the measured instructions only up to a tree depth of 13. With further increasing tree depth, decreasing numbers of instructions are predicted while the measured instructions of the FMM remain constant. This is caused by the separation of near-field and far-field computations in separate passes in the computational run of the FMM. The prediction of the near-field instructions considers only the particle-particle interactions, but not the additional instructions for the loops required in the separate near-field pass. These differences are only significant when the number of particle-particle interactions becomes very small. However, in these cases the number of far-field instructions are about an order of magnitude larger and the mispredicted near-field instructions have no influence on the prediction of the total number of instructions.

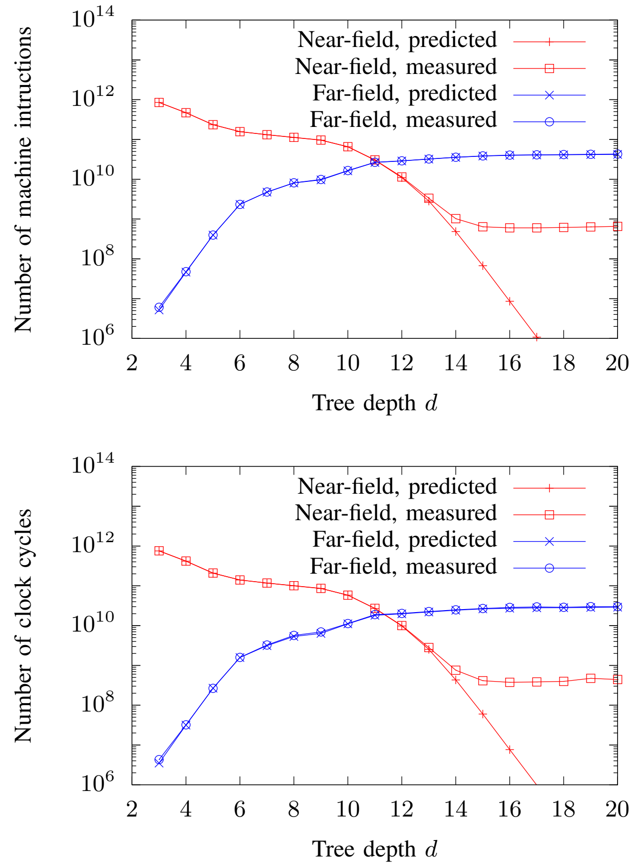

Figure 1. Comparison of the predicted and measured numbers of instructions (top) and cycles (bottom) for the near-field and far-field computations of the FMM with the Xenon particle distribution using an error bound of 10^{-3}.

Figure 1 (bottom) shows results for performing the same experiment as in Fig. 1 (top), but using cycles for the performance prediction instead of instructions. While the numbers of cycles are usually a little bit smaller than the corresponding numbers of instructions, the general behavior of the FMM algorithm and of the performance prediction shows no difference. Comparing both the predicted near-field and far-field instructions to the measured near-field and far-field instructions leads to a maximum deviation of about 4 %. The same comparison using cycles leads to a maximum deviation of about 7 %. The difference is slightly larger, because the measured cycles include effects such as pipeline stalls or cache misses. However, their impact on the FMM computations is small and thus can be neglected for the performance prediction.

To show the necessity of repeating the benchmark runs on every hardware system, the platform-specific cost parameters obtained for the Intel system and for the AMD system are

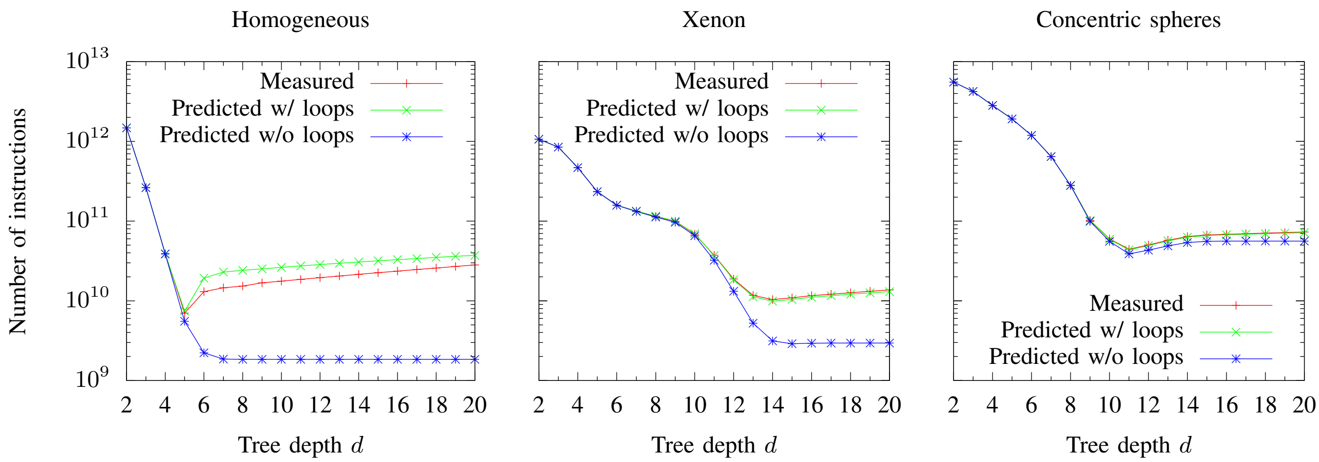

Figure 2. Comparison of the measured and predicted numbers of instructions with and without considering the nested loops with the homogeneous distribution (left), the Xenon distribution (middle), and the concentric spheres distribution (right) using an error bound of 10^{-1}.

		Xeon X5650		Opteron 8347	
cost parameter	counters	instr.	cycles	instr.	cycles
b-loop	4680	2367	1567	2251	2260
n-loop, z direction	14040	14	3	14	5
n-loop, y direction	40080	21	16	21	14
n-loop, x direction	114480	31	21	31	21
c-loop	52184	65	65	65	106
particle-particle	32838560	163	145	163	163
box-box	2436056	2583	1842	2529	1897

Table I

PLATFORM-SPECIFIC COST PARAMETERS MEASURED IN INSTRUCTIONS AND CYCLES FOR THE INTEL SYSTEM AND THE AMD SYSTEM WITH AN REQUESTED RELATIVE ERROR BOUND OF 10^{-3}. COUNTERS OF LOOP ITERATIONS, PARTICLE-PARTICLE INTERACTIONS, AND BOX-BOX INTERACTIONS ARE SHOWN FOR THE HOMOGENEOUS PARTICLE DISTRIBUTION WITH THE OPTIMAL TREE DEPTH $d = 5$.

listed in Table I. For both systems, the computational costs for computing single box-box and particle-particle interactions as well as for executing single iterations of the different loops of Alg. 1 measured in instructions and cycles are shown. Furthermore, the values of the counter variables counting loop iterations and interactions are shown for using the homogeneous particle distribution and the optimal tree depth $d = 5$. The results show that there can be significant differences between using instructions and cycles as well as between the different hardware systems. However, not only the properties of the hardware system, but also compilers and compiler optimizations can have an influence. Thus, it is necessary to perform the benchmark runs once for the current hardware and software environment.

C. *Influence of the nested loops*

Determining the computational costs of the different nested loops of Alg. 1 requires several benchmark runs of the FMM using an appropriate inhomogeneous particle distribution. In comparison to that, the computational costs for box-box and particle-particle interactions can be determined much easier by performing only the corresponding computations of interactions with random input data.

Figure 2 compares the measured number of executed instructions to the predicted number of instructions with and without considering the computational costs of the nested loops depending on the tree depth. Results are shown for computing the homogeneous distribution (left), the Xenon distribution (middle), and the concentric spheres distribution (right) with a requested relative error bound of 10^{-1} using the Intel system. The results show that the computational costs of the nested loops have a significant influence on the total number of instructions. Only if the loops are considered, a precise prediction of the instructions close to the measured number of instructions of the FMM is achieved.

Without considering the loops, the prediction of the instructions for the homogeneous distribution and the Xenon distribution does not lead to minimum. Thus, in these cases the analysis run can determine the optimal tree depth for the FMM only if the loops are considered for the performance prediction. Furthermore, the results also show that the optimal tree depth depends strongly on the particle distribution. While the homogeneous distribution has a low optimal tree depth of $d = 5$, the concentric spheres distribution and the Xenon distribution require to use a tree depth of $d = 11$ and $d = 14$, respectively. This shows the necessity of analyzing the given particle distribution as it is performed within the analysis run of the presented performance prediction method. Deviating from the optimal tree depths can lead to a significantly higher number of instructions executed for the FMM than actually required. For example, using the next best tree depth after the optimal tree depth increases the instructions for the FMM computations of about 5 % with the Xenon distribution, of about 13 % with the concentric spheres distribution, and of about 86 % with the homogeneous distribution.

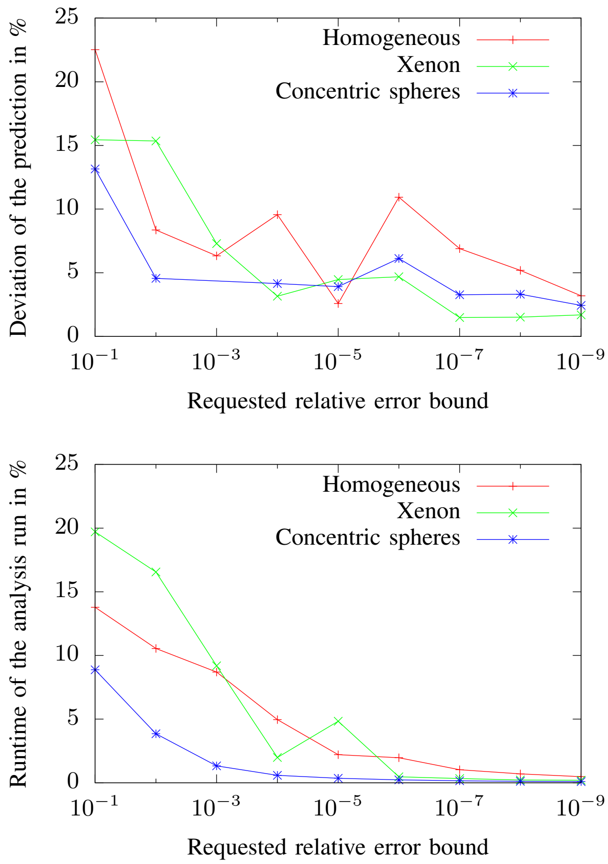

Figure 3. Deviation of the predicted instructions from the measured instructions of the FMM (top) and runtime of the analysis run with respect to the overall runtime of the FMM (bottom).

D. *Precision and overhead of the performance prediction*

The requested error bound for the FMM algorithm has a significant effect on the computational costs of box-box interactions and therefore on the overall runtime. Furthermore, the computational costs of these box-box interactions can be predicted more precisely than the costs of the nested loops. Therefore, the requested error bound has an influence on both the precision and the computational overhead of the performance prediction method.

Figure 3 (top) shows the deviation of the predicted instructions from the measured instructions of the FMM depending on the requested relative error bound. The deviation encloses the instruction counts with the optimal tree depth and the next lower and higher tree depth. The results show that if only the instructions around the optimal tree depth are considered, then the precision of the performance prediction is approximately the same for all three particle distributions. A high error bound of 10^{-1} leads to small runtimes of the FMM and to a deviation between the predicted and measured instructions of up to about 23 %. Lowering the error bound to 10^{-9} leads to increasing runtimes of the FMM and to a more precise performance prediction that deviates only up to 5 % from the measured instructions.

Figure 3 (bottom) shows the runtime of the analysis run used for the performance prediction with respect to the overall runtime of the FMM depending on the requested relative error bound. The results show that with a high error bound of 10^{-1}, the analysis run can require up 20 % of the overall runtime of the FMM. However, lowering the error bound decreases the runtime of the analysis run below 5 % with an error bound of 10^{-4} and even below 0.5 % with an error bound of 10^{-9}. This is caused by the increasing runtimes of the FMM computations required to provide the higher precision while the runtime of the analysis run is independent from the requested error bound and thus remaining constant.

V. Related work

There exists a large variety of implementations of the Fast Multipole Method for various platforms, e.g. for parallel computers with distributed memory [2], [6], for GPUs [3], for clusters of GPUs [4], and for systems of nodes connected via MPI and each having a GPU [7]. However, none of these implementations provide control over the accuracy of the FMM computations. The implementation presented in [5] proposes a two-stage error estimation scheme and is the base of the performance prediction presented in this article for determining the optimal tree depth of the FMM computations.

In [8], a model for estimating the execution time of a parallel implementation of the FMM depending on the number of particles, the number of processors, and the number of boxes at the last level of the FMM tree is presented. In [9], this model is extended by estimations of the computations caused by each tree node in order to improve the load balancing. Furthermore, the communication load and the memory usage is considered in this model. However, these estimations do not consider the real execution times on the hardware platform being used.

Benchmarking several existing implementation variants on the real hardware to determine variants with minimal execution times is known as *automatic tuning*. Existing frameworks are PHiPAC [10] and ATLAS [11] for dense linear algebra, OSKI [12] for sparse linear algebra, FFTW [13] for fast Fourier transform, and SPIRAL [14] for digital signal processing. In the field of particle simulations, automatic tuning is used in [15] for the generation of FFT codes for multipole-to-local translations in spherical harmonics. In [16], an FMM implementation is tuned for multi-core processors. However, the tuning is not performed automatically and concerns mainly implementation and parallelisation aspects, but not algorithmic optimizations.

Counting operations of algorithms is used for the prediction of execution times. In [17], a performance model for a specific hydrodynamics application code is developed using hardware characteristics such as peak performance or memory bandwidth as well as the problem size as input parameters for the execution time prediction. However, the parameters of the specific platform have to be determined manually. Compiler-based approaches are mostly independent from specific applications. In [18], whole applications are tuned by the compiler by extracting the most expensive loops from the source code and applying transformations such as tiling, unrolling or permutation. The approach presented in [19] uses compiled binaries of applications to derive models of their execution times. In [20], the execution times of scientific applications are predicted with an accuracy of about 20 %. This approach determines the costs of each statement in the source code by considering the number of operations and their respective costs for the target platform. In general, compiler-based approaches are widely applicable and have strong opportunities for optimizations. However, these approaches cannot exploit application-specific knowledge or adapt to problem-specific characteristics of the application data. In contrast, the application-specific performance prediction method presented in this article adapts automatically to both platform- and problem-specific parameters.

VI. Conclusions

This article presented a method for determining the optimal tree depth for the FMM algorithm. By balancing the execution times of near-field and far-field computations, the total execution time of the FMM is minimised. The approach uses a performance prediction method integrated into the FMM implementation. Numerical operations as well as the computational costs of complex nested loops of the FMM tree traversal are considered. The results show the high precision and the low additional overhead of the prediction method. Furthermore, it was shown that the optimal tree depth can be determined for homogeneous as well as for arbitrary inhomogeneous particle distributions. Using the optimal tree depth was shown to be of great importance while slightly differing non-optimal tree depths lead to significantly higher execution times for the FMM.

References

[1] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comput. Phys.*, vol. 73, no. 2, pp. 325–348, 1987.

[2] J. Kurzak and B. M. Pettitt, "Massively parallel implementation of a fast multipole method for distributed memory machines," *Journal of Parallel and Distributed Computing*, vol. 65, no. 7, pp. 870–881, 2005.

[3] N. A. Gumerov and R. Duraiswami, "Fast multipole methods on graphics processors," *J. Comput. Phys.*, vol. 227, pp. 8290–8313, 2008.

[4] R. Yokota, T. Narumi, R. Sakamaki, S. Kameoka, S. Obi, and K. Yasuoka, "Fast multipole methods on a cluster of GPUs for the meshless simulation of turbulence," *Computer Physics Communications*, vol. 180, no. 11, pp. 2066–2078, 2009.

[5] H. Dachsel, "An error-controlled fast multipole method," *J. Chem. Phys.*, vol. 132, no. 119901, 2010.

[6] S. Ogata, T. J. Campbell, R. K. Kalia, A. Nakano, P. Vashishta, and S. Vemparala, "Scalable and portable implementation of the fast multipole method on parallel computers," *Computer Physics Communications*, vol. 153, no. 3, pp. 445–461, 2003.

[7] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros, "A massively parallel adaptive fast-multipole method on heterogeneous architectures," in *Proc. of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009, pp. 1–58.

[8] L. Greengard and W. D. Gropp, "A parallel version of the fast multipole method," *Comput. Math. Appl.*, vol. 20, no. 7, pp. 63–71, 1990.

[9] F. A. Cruz, M. G. Knepley, and L. A. Barba, "PetFMM – A dynamically load-balancing parallel fast multipole library," *Internat. J. Numer. Methods Engrg.*, vol. 85, no. 4, pp. 403–428, 2011.

[10] J. Bilmes, K. Asanović, C. whye Chin, and J. Demmel, "Optimizing matrix multiply using PHiPAC: a Portable, High-Performance, ANSI C coding methodology," in *Proc. of Int. Conf. on Supercomputing*, 1997.

[11] R. C. Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimizations of software and the atlas project," *Parallel Computing*, vol. 27, no. 1-2, pp. 3–35, 2001.

[12] R. Vuduc, J. W. Demmel, and K. A. Yelick, "OSKI: A library of automatically tuned sparse matrix kernels," *Journal of Physics: Conference Series*, vol. 16, p. 521, 2005.

[13] M. Frigo and S. Johnson, "The design and implementation of FFTW3," *Proc. of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[14] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo, "SPIRAL: Code generation for DSP transforms," *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, vol. 93, no. 2, pp. 232–275, 2005.

[15] J. Kurzak, D. Mirkovic, B. M. Pettitt, and S. L. Johnsson, "Automatic generation of FFT for translations of multipole expansions in spherical harmonics," *Int. J. High Perform. Comput. Appl.*, vol. 22, no. 2, pp. 219–230, 2008.

[16] A. Chandramowlishwaran, S. Williams, L. Oliker, I. Lashuk, G. Biros, and R. Vuduc, "Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures," in *Int. Symposium on Parallel Distributed Processing (IPDPS), 2010*. IEEE, 2010, pp. 1–12.

[17] D. J. Kerbyson, A. H. J., A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive performance and scalability modeling of a large-scale application," in *Proc. ACM/IEEE Conf. on Supercomputing*. ACM, 2001, pp. 37–37.

[18] A. Tiwari, J. K. Hollingsworth, C. Chen, M. Hall, C. Liao, D. J. Quinlan, and J. Chame, "Auto-tuning full applications: A case study," *International Journal of High Performance Computing Applications*, vol. 25, no. 3, pp. 286–294, 2011.

[19] G. Marin and J. Mellor-Crummey, "Cross-architecture performance predictions for scientific applications using parameterized models," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, pp. 2–13, 2004.

[20] C. Cascaval, L. DeRose, D. Padua, and D. Reed, "Compile-time based performance prediction," in *Languages and Compilers for Parallel Computing*, ser. LNCS, L. Carter and J. Ferrante, Eds. Springer, 2000, vol. 1863, pp. 365–379.