2016 IEEE 23rd International Conference on High Performance Computing

Load Balancing for Molecular Dynamics Simulations on Heterogeneous Architectures

Steffen Seckler, Nikola Tchipev, Hans-Joachim Bungartz, Philipp Neumann
Department of Informatics
Technical University of Munich
Garching, Germany
steffen.seckler@tum.de

***Abstract*—Upcoming exascale compute systems are expected to be built from heterogeneous hardware architectures. According to this trend, there exist various methods to handle clusters composed of CPUs, GPUs or other accelerators. Most of these assume that each node has the same structure—for example a dual socket system with an accelerator (GPU or Xeon Phi). The workload is then distributed homogeneously among the nodes. However, not all clusters fulfill this requirement. They might contain different partitions with and without accelerators. Furthermore, depending on the underlying problem to be solved, accelerator cards may perform better in native mode compared to offloading. Besides, various aspects such as cooling may influence the performance of individual nodes. It therefore cannot always be assumed, that the structure and performance of each node and hence the performance of every MPI rank is the same.**

In this contribution, we apply a k-d tree decomposition method to balance load on heterogeneous compute clusters. The algorithm is incorporated into the molecular dynamics simulation program ls1 mardyn. We present performance results for simulations executed on hybrid AMD Bulldozer–Intel Sandy Bridge, Intel Westmere–Intel Sandy Bridge and Intel Xeon–Intel Xeon Phi-architectures. The only prerequisite for the proposed algorithm is a cost estimation for different decompositions. It is hence expected to be applicable to a variety of n-body scenarios, for which a domain decomposition is possible.

***Keywords*-molecular dynamics; Intel Xeon Phi; AMD Bulldozer; heterogeneous; ls1 mardyn; k-d trees**

I. Introduction

Many of the current and upcoming fastest supercomputers utilize accelerators. Six out of the current top 10[1] supercomputers still provide a homogeneous architecture. A closer look, however, reveals that the two most powerful supercomputers are already heterogeneous systems. The *Tianhe-2* utilizes Intel Xeon Phi accelerator cards, while the supercomputer *Titan* uses Nvidia Tesla accelerators. Due to power limitations, trends foresee various future supercomputing platforms to be heterogeneous. The *Summit* (Nvidia Volta) and the *Sierra* (Nvidia Volta) supercomputers are only two examples of upcoming heterogeneous supercomputers and are expected to have a peak performance in the range of 150-300 PetaFLOPS[2]. While heterogeneous computers allow for an acceptable level of energy efficiency, they come at the price of more enhanced programming and software efforts. It therefore becomes important to be able to leverage both host and accelerator to optimize for time-to-solution or energy consumption, respectively.

Many current heterogeneous clusters do, however, not have such an easy structure. Instead they might contain different versions of hardware from different generations, or contain a partition with and one without accelerators. One such system is the SuperMUC Cluster[3], that contains a partition with Sandy Bridge CPUs, one with Haswell CPUs and another with Ivy Bridge CPUs and Xeon Phi accelerators. The latter partition is often called SuperMIC. Only very few current algorithms are able to handle such heterogeneities.

One of the application fields, that requires considerable compute power is given by molecular dynamics. Application scenarios range from simulations of proteins in life sciences [1] to process engineering [2]. The simulation code *ls1 mardyn* [2] that we focus on, has been designed for the latter purpose. *ls1 mardyn* is able to simulate fluids consisting of one or multiple different types of molecules. Some of these fluids tend to form inhomogeneities, e.g. through nucleation. To handle these a k-d tree-based domain decomposition had been introduced, see e.g. [3]. Moreover, various efforts have been made to optimize *ls1 mardyn* at node level for both recent (Intel) host and accelerator architectures [3], [4]. In this paper, we modify the k-d tree-based domain decomposition and leverage its cost function to efficiently exploit heterogeneous hardware systems. We present the load balancing implementation for large molecular systems as found in process engineering for homogeneous simulated scenarios on heterogeneous systems.

We start with a brief review on molecular dynamics theory in Sec. II. We discuss related work on load balancing in this context and provide details on our implementation in Sec. III. Performance evaluations for various heterogeneous

[1]http://www.top500.org/lists/2015/11/

[2]http://energy.gov/articles/department-energy-awards-425-million-next-generation-supercomputing-technologies

[3]https://www.lrz.de/services/compute/supermuc/

hardware combinations are discussed in Sec. IV. We close the discussion with a short summary and give an outlook to future work in Sec. V.

II. Short-Range Molecular Dynamics

We consider freely moving, rigid molecules, each consisting of one or several *interaction sites* which interact via potentials. Plugged into Newton's equations of (translational and rotational) motion, the resulting inter- and intramolecular forces yield molecular dynamics (MD). We solve these equations with the (rotational) Leapfrog method. Various intermolecular interactions, such as the Lennard-Jones potential [5]

$$U(\|\mathbf{x}_i-\mathbf{x}_j\|) := 4\epsilon\left(\left(\frac{\sigma}{\|\mathbf{x}_i-\mathbf{x}_j\|}\right)^{12}-\left(\frac{\sigma}{\|\mathbf{x}_i-\mathbf{x}_j\|}\right)^{6}\right),$$

with parameters ϵ and σ, depend on the distance of the molecular interaction sites $\|\mathbf{x}_i-\mathbf{x}_j\|$ and decay very rapidly for $\|\mathbf{x}_i-\mathbf{x}_j\| \to \infty$. One can therefore restrict considerations to *short-range* interactions and disregard all intermolecular interactions for distances $\|\mathbf{x}_i-\mathbf{x}_j\| \geq r_c$, where r_c denotes the cut-off radius.

To achieve linear complexity in the runtime of the evaluation of molecule-molecule interactions, we make use of the linked-cell method: molecules are sorted into cells of equal size $\geq r_c$, and a molecule thus only interacts with another molecule if both are located in the same or in neighboring linked cells. Due to the locality of short-range molecular interactions, parallelization can be achieved via a regular domain decomposition. If $n_x \times n_y \times n_z = n_{\text{proc}}$ processes are available, the domain is divided accordingly into $n_x \times n_y \times n_z$ block-shaped subdomains of same size. Each subdomain is surrounded by a *ghost layer* of linked cells. Updating the molecule data in the ghost layer once per time step, each process can locally evaluate the short-range forces. However, due to the randomized motion of molecules, the computational load per cell may change considerably. See amongst others [1] for further details on MD, the linked-cell method and parallelization.

III. Load Balancing in Molecular Dynamics

A. Related Work

Several works have addressed load balancing in the context of molecular dynamics, with early contributions dating back to the 90s [6]. In particular, various community codes have been considered in this regard. The scalable community software GROMACS [7], [8] supports execution on GPU and CPU as well as heterogeneous variants. Parallelization and load balancing is achieved via a combination of the eighth-shell domain decomposition method with triclinic cell adjustment and potential staggering. In case of extreme staggering, additional subdomains may enter the area of potential molecular interactions. The communication between subdomains thus becomes more extensive and tends away from the original nearest-neighbor consideration. This issue and potential overheads can be limited by the user via providing a minimum allowed subdomain size [8]. The software NAMD is used for biomolecular simulations. It provides static load balancing via a recursive bisection algorithm and Kalé et al. [9] have presented a modified greedy strategy for dynamic load balancing of biomolecular simulations. Investigations with regard to topology aware load balancing were conducted for NAMD and resulted in performance gains of up to 10% [10]. Results for hybrid CPU-GPU simulations using the package LAMMPS are reported in [11]. Besides static load balancing, "up to a 20% reduction in loop time was achieved with dynamic load balancing of forces" [11]. A hierarchical parallelization scheme for heterogeneous CPU-GPU systems is presented in [12]. Dynamic-scheduling-based multi-threading is employed to balance CPU and GPU workloads. Various approaches for load balancing of molecular dynamics based on irregular domain decompositions are discussed in [13], [14], [15], in particular targeting highly inhomogeneous and rapidly changing particle loads per process. Successful runs using Voronoi tessellations have been reported on up to 65 thousand MPI tasks [14]; however, issues have been encountered when the tessellation was trapped in local minima of the load balance cost function. A bulk synchronous parallel architecture model is developed and tested for various domain decomposition strategies to statically balance load in molecular dynamics simulations [16]. To dynamically balance work load among processes, performance data are measured and—together with an orthogonal recursive bisection decomposition—are used to determine the new size of the local computational subdomains.

B. Load Balancing in *ls1 mardyn*

Previous Work: In their original purpose k-d trees are used for associative searching [17] by generating a partitioning of a k-dimensional domain, cf. Fig. 1: first, the domain is split along a $k-1$ dimensional hyperplane to generate two subdomains. This splitting is further applied recursively. Hereby the selection of the splitting hyperplane plays an important role and depends on the application domain.

Niethammer et al. provided an implementation of k-d tree-based domain decomposition for inhomogeneous particle load balancing and presented scalability results on up to 1000 MPI ranks [18] using the simulation software *ls1 mardyn*.

The initial implementation of the k-d tree-based load balancing in *ls1 mardyn* is able to balance inhomogeneous particle systems that may arise due to clustering or molecular droplet formation. First, the domain is split into multiple subdomains of equal load. These subdomains are then distributed among the processes. Running the simulation

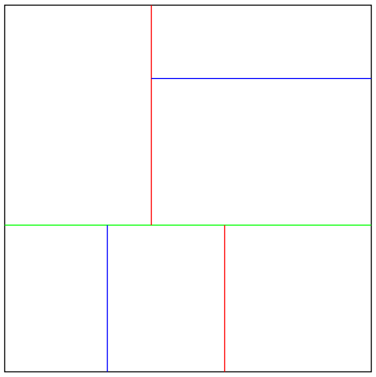


Figure 1. Space-partitioning using a 2-d tree. The different colors represent the different levels of the splitting hyperplanes.

on equally powerful compute cores and using one process per core yields a balance of potentially inhomogeneous particle systems over all processes. However, heterogeneous compute systems have not been taken into account. In the following, we detail the existing algorithm and extend it to heterogeneous compute systems.

Initial k-d Tree-Based Load Balancing Scheme: We employ the k-d splitting of the domain on the linked cell level, i.e. each linked cell contained in the initial domain is assigned to one of the partitions. This allows for a simplified implementation of the load balancing algorithm, since the computational kernel of the molecular dynamics simulation remains untouched. To carry out the partitioning, an estimation of the cost of possible splittings has to be performed. We do this by assigning a load to each linked cell. In each cell, all n_{cell} sites within the cell interact with each other, thus resulting in a $O\left(n_{\text{cell}}^2\right)$ dependence of the load on the particle number. Additionally, each particle site of the cell has to interact with every site of the neighboring cells. This yields a total cost C_{cell} of

$$C_{\text{cell}} = n_{\text{cell}}^2 + \frac{1}{2} \sum_{\text{neighbors}} n_{\text{cell}} n_{\text{neighbor}}. \qquad (1)$$

In contrast to intra-cell interactions, more molecules lie outside of the cutoff radius when two neighboring cells are considered. We therefore modeled the cost of this calculation with the factor $1/2$. Each subdomain is always split along its longest edge. This is a communication avoiding scheme in the sense that subdomains with a high volume to surface ratio are maintained. To find the optimal k-d decomposition all combinations of possible splitting hyperplanes have to be checked. This is however unfeasible for a large amount of linked cells. It is therefore important to find good strategies to determine good splitting hyperplanes. For large subdomains and many processes, a good splitting can be easily found by splitting the domain in half and assigning an appropriate amount of processes to each subdomain. This assignment should be made such that the load ratio of the two partitions equals the ratio of the number of their processes. For small subdomains it is important to determine a close to optimal splitting. Therefore all possible partitionings are tested and the best is chosen. Once only one process is assigned to a subdomain, the subdomain is no longer split. The evaluation is done in a parallel and recursive way and is depicted in algorithm 1. Hereby each process only calculates the cost of its cells and parents of them. In the algorithm the function `get1DSubdivisions`

Algorithm 1: Parallel algorithm to evaluate the best partitioning.

```
1  Function decompose (fatherNode)
       input : fatherNode: a KDNode consisting of the
               subdomain and the assigned processes
               (starting process and numprocs)
       output: complete partitioning of fatherNode
2      if fatherNode.numprocs == 1 then
3          fatherNode.calculateDeviation ()
4          return fatherNode
5      subdividedNodes ← get1DSubdivisions
        (fatherNode)
6      foreach node in subdividedNodes do
7          if myrank in node.child1.processes then
8              node.child1 ←decompose (node.child1)
9          else
10             node.child2 ←decompose (node.child2)
11         partialAllReduceDeviations ()
12         node.deviation ←node.child1.deviation
            +node.child2.deviation
13     return node of subdividedNodes with minimal
        deviation
```

returns one or multiple possible splittings of the subdomain, that is associated to the input parameter. The input parameter (fatherNode) resembles a node of the k-d tree and is associated to a domain region and a certain range of processes. The return type of the function is a collection of subdividedNodes. One subdividedNode is hereby a clone of the initial (fatherNode), but now owning two children. Each of the children again owns a domain region and a range of processes. If the domain and the number of processes is large enough, a single subdividedNode is returned. Only if the domain is small, a cost comparison is feasible and the function returns multiple splittings. Each node provides the function `calculateDeviation`. It returns the deviation of its load from the ideal load. In the initial implementation we chose a splitting such that the load is equally distributed among the processes. The "ideal" load for a process is defined as $C_{\text{ideal},i} = P_{\text{i}} \frac{C_{\text{total}}}{P_{\text{total}}}$, where C is the load and P the performance of one or multiple processes. For homogeneous architectures, the performance of each process P_i is assumed to be the same. The ideal load is thus equal for each process.

Adaptions for Heterogeneous Systems: On heterogeneous systems, the processes run at different levels of performance. We therefore adapted the original implementation [18], [19]. The changes are hightlighted in this paragraph. We extended the algorithm by splitting the domain according to the performance of the processes. First the processes assigned to a subdomain are separated into two groups, that provide roughly the same performance. Then the domain is split such that the loads of the newly created subdomains are distributed according to the ratio of the performance of the process groups. This results in two partitions with almost equal load. The process groups are then assigned to the two subdomains. Those subdomains are then split further, until each subdomain is assigned to exactly one process. For scenarios with a homogeneous particle distribution, it is sufficient to split the domain, such that the ratio of the size of the subdomains and the performance ratios of the process groups match. Only one rebalancing step at the start of the simulation is necessary in this case. Still, the performance at which each process calculates the site-site interactions has to be known. These performance values can be provided statically within the program. Since the values typically depend on the underlying scenario and hardware, this however requires a user to carry out prior simulations of the considered scenario (or very similar ones) on the current hardware and derive respective performance estimates. We follow a different approach by employing a dynamic evaluation of the process performance at runtime. The performance is measured by counting the achieved floating-point operations (FLOPs) per second during the linked cell traversal in software. An alternative approach would be to rely on hardware counters. These, however, tended to be rather inaccurate on Intel's Ivybridge and Sandy Bridge processors and have been deactivated for the Haswell generation. Instead of measuring the performance, one could assume, that a simple time measurement for each process is sufficient. This however is not always the case. To understand this, let us assume, that we have a slow process and a fast process. When assigning the same amount of load to both processes, the slow process will take longer. After the first rebalancing step the load is distributed according to the performance ratio of the two processes. The time both needed to perform the force calculation is therefore almost equal. Using just the ratio of times, one would then assume, that both processes provide the same performance. However, this is obviously not the case. A proper ratio of the work accomplished per unit time can be extracted only if the workload of the respective process is known. This again results in a performance measurement and can only be evaluated, if the previously allocated work is known. In the following we make use of *performance measurements*. We thus do not need to take previously allocated work or a history of performance values into consideration. Instead we measure the performance directly.

Implementation in ls1 mardyn*:* In every rebalancing step of the k-d decomposition, the entire tree is generated anew. One rebalancing step can be split into six steps, including necessary particle transfers of leaving and halo particles:

1) Exchange leaving molecules.
2) Delete halo molecules.
3) Collective call to accumulate the number of particles for each cell.
4) Construct the new tree as described previously.
5) Communication of the particles, that are now owned by different processes. This is done using point-to-point operations.
6) Exchange halo molecules.

In step 3 an all-to-all reduction is necessary to accumulate the number of particles for each cell. Although the size of this collective call scales with the amount of linked cells, it does not pose a problem, since only one value is needed per cell. Even for a million linked cells, only a few MB of data are transferred. Additionally this step is required, only if rebalancing has to be performed. If no rebalancing should occur for the specific time step, a simple exchange of both leaving and halo molecules is sufficient. No collective calls are used.

Further details can be found in [18], [19].

IV. Results

A. Test Environment

We ran our simulations on two different compute clusters to evaluate our load balancing approach. The MAC-Cluster[4] is a general purpose cluster with various different partitions, of which we use the following:

BDZ	A partition with 19 nodes each consisting of a quad socket system using AMD Bulldozer Opteron 6274, 256 GB RAM and QDR infiniband. Each socket has 16 cores with a frequency of 2.2 GHz. The higher boost frequency can only be reached for at most 8 cores. The Bulldozer processors are able to utilize the AVX instruction set and additionally FMA4.
SNB	A partition with 28 nodes. Each node contains two Intel Sandy Bridge-EP E5-2670 as a dual socket system and 128 GB RAM. Each socket provides 8 cores with hyperthreading enabled. The clock frequency is fixed at 2.6 GHz. The interconnect is of QDR infiniband type. AVX is supported by the Sandy Bridge architecture.
WSM	One quad socket node using four Intel Westmere-EX Xeon E7-4830 (8 cores, hyperthreading, 2.13 GHz) and 512 GB RAM. As interconnect,

[4]http://www.mac.tum.de/wiki/index.php/MAC_Cluster

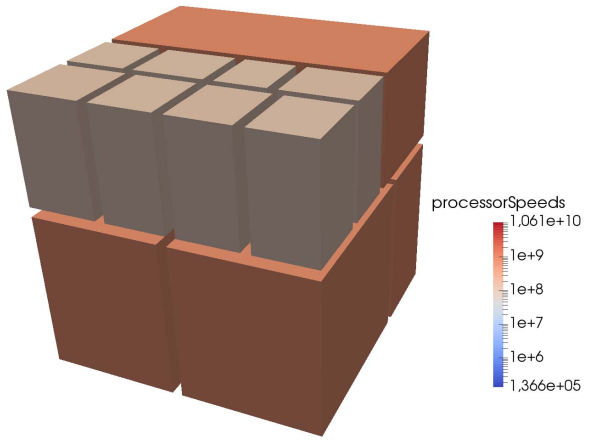


Figure 2. k-d decomposition of the domain on the SuperMIC cluster after 10 time steps and one rebalancing step. Four processes have been used on the host, as well as on the two accelerator cards.

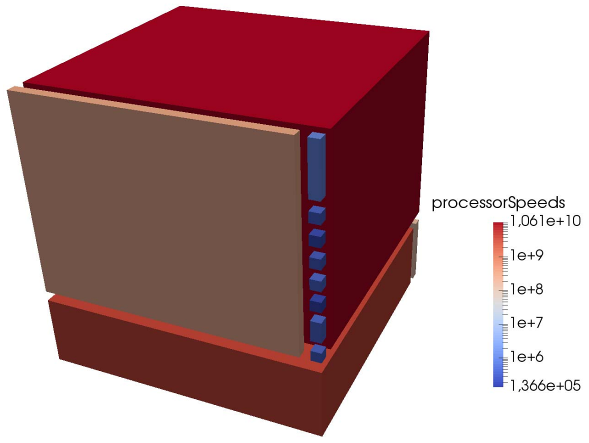


Figure 3. k-d decomposition of the domain on the SuperMIC cluster after 119 time steps and 6 rebalancing steps. Four processes have been used on the host, as well as on the two accelerator cards.

FDR infiniband is used, which, however, only provides performance comparable to that of QDR infiniband, since it is connected via PCIe 2.0. The Westmere architecture is only able to utilize SSE instructions. AVX instructions are not supported.

For our test scenarios all Turbo Boost (Intel) / Turbo Core (AMD) properties of the individual partitions of the MAC-Cluster have been disabled. Concerning single-process performance, we expect the Sandy Bridge architecture to be the fastest. Bulldozer and Westmere are slower, with slight differences in performance between them due to AVX and SSE instructions and molecular dynamics specifics.

The second cluster SuperMIC[5] consists of 32 nodes each with a dual socket system of Intel Ivy Bridge-EP E5-2650 v2 (2.6 GHz). Additionally, each node of the SuperMIC cluster is equipped with two Xeon Phi 5110P coprocessor cards. These coprocessors each consists of 60 cores clocked at 1.1 GHz with 4 hyperthreads each. The host provides 64 GB of memory, the coprocessors include 8 GB each. The compute nodes are connected via Infiniband FDR14.

Both clusters are hosted by the Leibniz Supercomputing Centre in Munich.

Except for the BDZ nodes, all executables have been compiled with the Intel C++ Compiler (ICC, version 16.0.0). For the BDZ nodes the GNU Compiler Collection (GCC) version 4.9.3 has been used. The used MPI version is Intel MPI version 5.0 (MAC-Cluster), resp. version 5.1 (SuperMIC). The code has been compiled with appropriate vector support for each node.

B. Test Scenarios

In this paper we restrict considerations to scenarios with homogeneous particle distributions. This implies, that the workload for each cell is approximately constant. The following scenarios are considered:

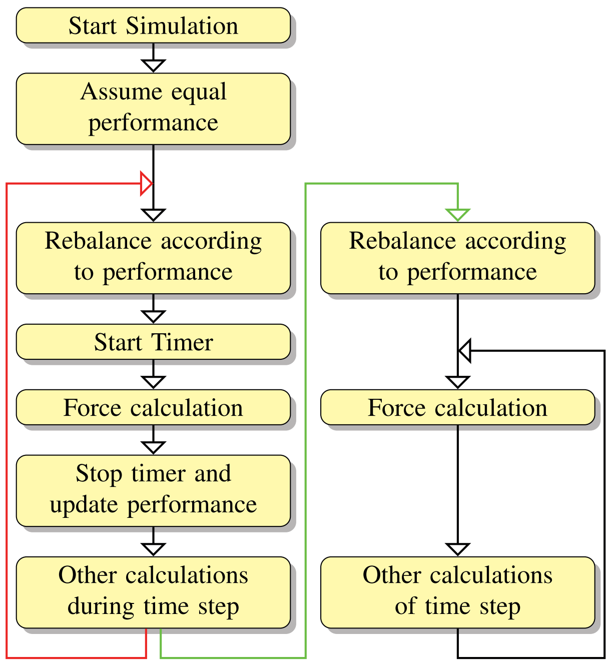
E512k This scenario contains 512 000 ethane (C_2H_6) molecules. Ethane is modeled as a pair of two Lennard-Jones sites with a fixed distance between the two pseudo atoms. The molecule density is chosen such that roughly 37 molecules (74 sites) exist within each cell. The whole scenario consists of $25 \times 25 \times 25$ linked cells.

LJBig The second scenario is used for large simulations on many nodes. It contains around 344 million single-centered Lennard Jones molecules. It consists of $200 \times 200 \times 200$ linked cells. Each cell contains approximately 43 molecules.

C. Performance Measurement

As described in section III-B, we measure the performance by calculating the FLOPs per second during the force calculation and the inherent cell traversal. This can be done statically or dynamically. In the dynamic case, the performance is given by its average during two rebalancing steps. The rebalancing is then accomplished according to the generated performance ratios. To point out limitations of this performance-based recursive rebalancing, consider Figs. 2 and 3. The first rebalancing step yields an appropriate partitioning of the domain, cf. Fig. 2. Repeatedly rebalancing and remeasuring the performance, however, may influence the partitioning drastically, cf. Fig. 3, yielding highly imbalanced simulations with regard to simulation time—but balanced subdomains with regard to performance. This is due to the FLOP rate being dependent on the subdomain size, reaching its maximum only for particle numbers bigger than some threshold $n_{process}$. Assume two processes running on identical platforms. Each process holds a different

[5]https://www.lrz.de/services/compute/supermuc/supermic/



Start Simulation
Assume equal performance
Rebalance according to performance
Start Timer
Force calculation
Stop timer and update performance
Other calculations during time step
Rebalance according to performance
Force calculation
Other calculations of time step

Figure 4. Schematic overview of the performance measurements. The red path indicates the junction that is taken for the dynamic measurement, while the green path indicates the static method. Note that for the dynamic measurement not in every time step a rebalancing is necessary. Instead rebalancing can be done after a certain interval of steps.

number of particles n_{p1} and n_{p2} with $n_{p1} < n_{p2} < n_{process}$. Since process 2 is running at a higher FLOP rate, it will obtain linked cells and therefore particles from process 1. With less particles, the performance of process 1 drops. In each rebalancing step more and more particles are migrated from process 1 to process 2. Holding enough particles per process, that is in the case n_{p1},$n_{p2} > n_{process}$, both processes are expected to show similar performance and a valid balancing of the load is achieved. The static case circumvents this problem by measuring the performance only once. This measurement is conducted at the beginning of the simulation. We call this the static way. The two different techniques are displayed in Fig. 4.

D. Reference Run

To get an overview of the desired efficiency, the performance on two Sandy Bridge nodes has been measured. The measurement has been performed for both the standard domain decomposition method (cf. Fig. 5) and the k-d decomposition algorithm. For the former the domain is split into a grid of n_{proc} subdomains, with n_{proc} being the number of processes. For 13, 17, 19, 23, 26, 29, 31 processes, a simulation is not possible: in these cases, the domain of $25 \times 25 \times 25$ linked cells is not big enough as we need at least 2 cells per process in each dimension. Hence, at most 11 processes can be used per dimension. In both cases,

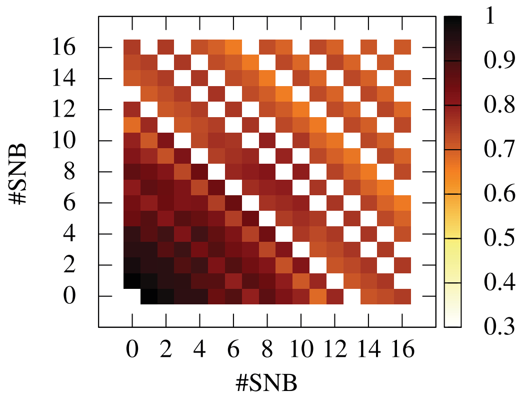


Figure 5. Parallel efficiency of the grid-like domain decomposition for the E512k dataset using two different Sandy Bridge nodes. The x-axis indicates the number of processes used on the first node, the y-axis the number of processes on the second node. White gaps indicate that a measurement could not be performed. The lowest efficiency is measured at 65%.

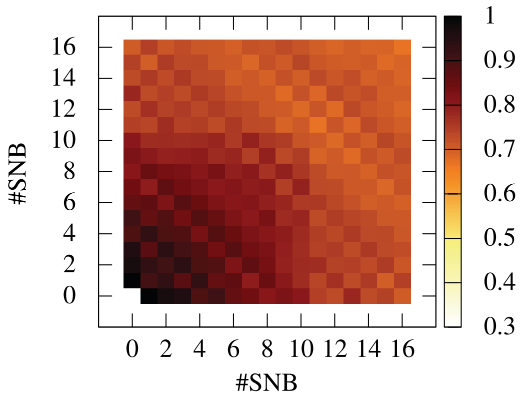


Figure 6. Parallel efficiency of the ***homogeneous k-d decomposition*** (*hom-kDD*) for the E512k dataset using two different Sandy Bridge nodes. The domain is split into subdomains with equal load. The lowest efficiency is measured at around 65 %.

we assume all processes to execute at equal performance. We refer to the k-d decomposition method, that assumes equal FLOP-rates on all processes as ***homogeneous k-d decomposition*** (hom-kDD). For the k-d decomposition the domain does not have to be split into this grid anymore. Instead a tree is constructed. In contrast to the standard grid-based domain decomposition, arbitrary numbers of processes are supported by this method, cf. Fig. 6.

In a second experiment, we compared the performance of the k-d decomposition from above and the ***performance-balanced k-d decomposition*** for heterogeneous systems (pb-kDD) on the SNB-SNB system. Figure 7 shows the arising performance ratio. The heterogeneous version includes performance measurements of the single processes and distributes the load accordingly. Similar performance is obtained in this scenario as expected, since all ranks

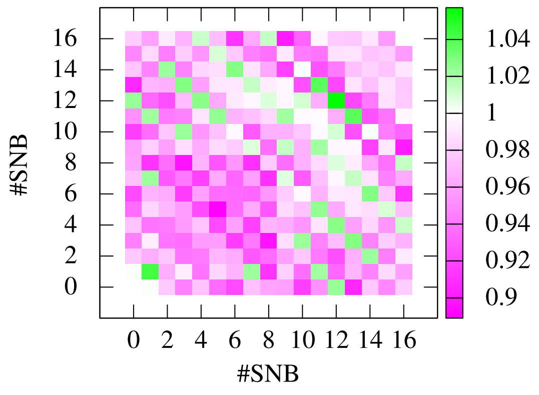


Figure 7. Performance ratio of the ***performance-balanced k-d decomposition*** (pb-kDD) for the E512k dataset and hom-kDD. Since two Sandy Bridge nodes are used and they consist of the same hardware, no big performance improvements could be measured. The maximum decrease in performance could be measured at 10%.

run on equal hardware. Fluctuations in the performance measurements, however, yielded a maximum decrease in the overall performance by 10%. Even small deviations can result in a different tree structure, which results in a different time-to-solution.

E. Load Balancing on Heterogeneous Architectures

The homogeneous k-d decomposition has difficulties to cope with heterogeneous hardware. Assume there exist n_{fast} processes on the faster hardware and n_{slow} processes on the slower hardware. If both types of processes have to handle the same load, the faster processes will finish before the slower ones. Due to communication, however, the faster processes will not be able to do anything while waiting for the slower processes to catch up. In the end both types of processes will have done the same work after the same time. The performance of the processes is therefore limited by the slower processes. The overall performance $P_{\text{total,hom}}$ can therefore be assumed as

$$P_{\text{total,hom}} = P_{\text{slowest}} n_{\text{total}}, \tag{2}$$

where P_{slowest} is the minimum of the individual performances.

This is shown in Fig. 8. Here major performance drops can be measured when adding a single Westmere process to a large amount of processes on a Sandy Bridge node. E.g., the addition of a process on a Westmere node to 14 Sandy Bridge processes results in an increase of time-to-solution of around 20%, while the parallel efficiency drops by roughly 15%.

The introduction of pb-kDD indeed increases the overall performance. Here the overall performance $P_{\text{total,inhom}}$ is bounded by the sum over all processes.

$$P_{\text{total,inhom}} = \sum P_{\text{type}} n_{\text{type}} \tag{3}$$

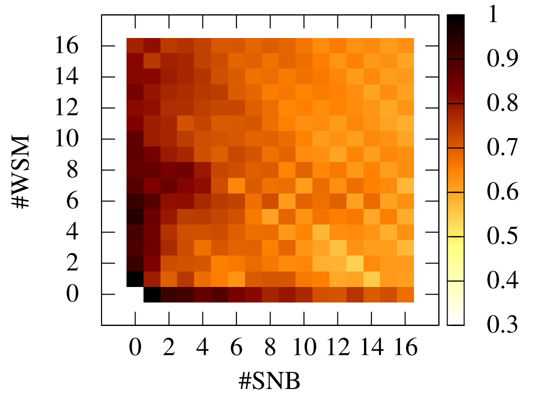


Figure 8. WSM-SNB: Parallel efficiency of hom-kDD for the E512k dataset. The domain is split into subdomains with equal load. The lowest efficiency (55%) is measured, when many ranks are started on Sandy Bridge cores, but only few on Westmere cores.

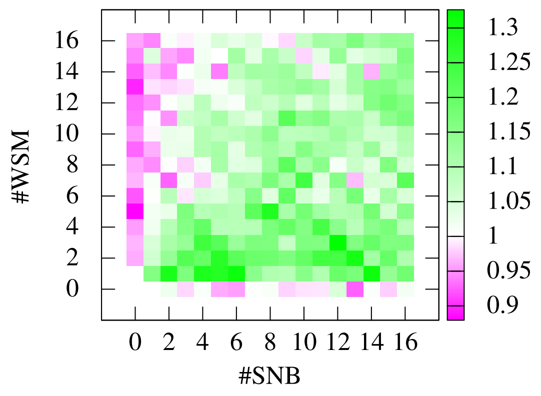


Figure 9. WSM-SNB: Performance ratio of pb-kDD and hom-kDD for the E512k dataset. For most configurations a performance improvement could be measured. The most significant performance decreases could be measured for scenarios, where all ranks belong to the same node.

The performance ratio is thus

$$\frac{P_{\text{total,inhom}}}{P_{\text{total,hom}}} = \frac{\sum P_{\text{type}} n_{\text{type}}}{P_{\text{slowest}} n_{\text{total}}}. \tag{4}$$

An upper bound of this ratio is $P_{\text{fastest}}/P_{\text{slowest}}$ and is obtained for $\lim_{n_{\text{fastest}}} \to \infty, n_{\text{slowest}} > 0$.

Figure 9 shows the performance ratio for the E512k dataset. The maximal obtained performance improvement is above 30%. This is slightly below the maximal expected value of 1.5, that has been calculated by the ratio of the execution times of sequential simulations on the specific hardware (768 s for WSM, 507 s for SNB). The maximum value is a theoretical value, that has been calculated in the limit of a big number of ranks, it is therefore not expected to be reached. Additionally that number does not include any estimation on communication cost. Especially, inter-node communication has not been accounted for. However,

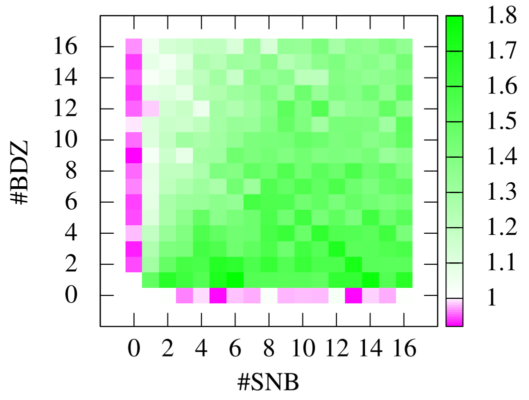


Figure 10. SNB-BDZ: Performance ratio of pb-kDD and hom-kDD for the E512k dataset. A maximum improvement of up to 80% is achieved.

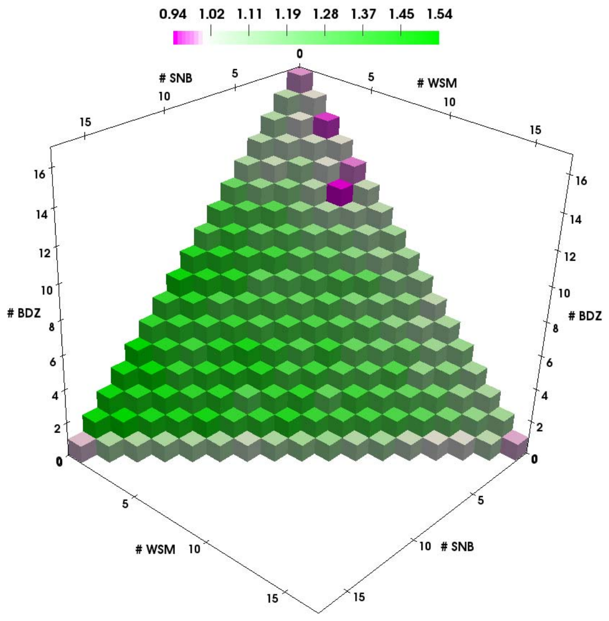


Figure 11. WSM-SNB-BDZ: Performance ratio of pb-kDD and hom-kDD for the E512k dataset. The total number of processes is 16. Speedups of more than 1.5 could be measured.

the measurement of 1.3 already suggests that our method runs at reasonable performance on the specified hardware.

Figure 10 shows the benefits of the heterogeneous version of the k-d tree-based decomposition over the homogeneous version for the combination of one Sandy Bridge node with one Bulldozer node. On the Bulldozer node a sequential execution time of 976 s could be measured. The Sandy Bridge processes are thus performing 1.9 times faster than the Bulldozer processes. This is a larger factor than for the SNB-WSM case. Possible reasons for that include the not

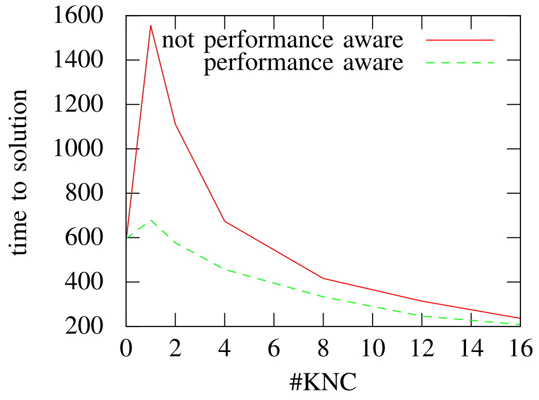


Figure 12. Time to solution on one node of the SuperMIC cluster. Shown is the performance development, when using only one rank on the host and #KNC ranks on one of the accelerator cards.

as heavily optimized GCC compiler in comparison to the Intel compiler, that has been used for the Intel architecture. Since the performance difference is greater compared to the SNB-WSM measurements we expect and confirmed bigger potential speedups. The heterogeneous algorithm is able to increase the performance by up to 80%, thus reducing the needed total time by 45%. The speedup is close to the theoretically calculated maximal value of 90% and resembles a significant gain in efficiency.

Besides of configurations with two different nodes, more nodes can be utilized. In the used MAC-Cluster only three different types of nodes exist. In Figure 11, the performance ratio between the heterogeneous and the homogeneous version of the load balancing algorithm is shown for a combination of one node from each node type of the MAC-Cluster. Hereby the total number of processes is kept constant at 16 which corresponds to the bottom-line of non-hyperthreaded single-node Sandy Bridge usage. For almost all configurations a significant improvement could be measured. Only if only one type of node has been used (right, top and left corner) or if very few Sandy Bridge cores (1 or 2) have been utilized, performance decreases could be measured. These however were of lower single-digit percentage.

F. Host-Accelerator Load Balancing for Xeon-Xeon Phi

As an extreme case for performance differences between two ranks within the simulation the Xeon Phi accelerator cards have been used. The measurements have been performed on the SuperMIC cluster. A process on the host is roughly 5 times faster than a process on the accelerator. Figure 12 shows the performance development, when adding ranks on the accelerator to a single rank on the host. Using the homogeneous version of the domain decomposition a significant performance drop can be seen, when adding a single rank on the accelerator. With the addition of more ranks the time to solution decreases. The heterogeneous version outperforms the homogeneous one significantly. However even when using performance-aware load balancing, adding a single process on the host leads to a performance decrease. Explanations can be found in the far lower speed of those ranks or imperfect load balancing. The time that the processes spend in communication is increased by a factor of more than five in this case. The graph shows that adding only few components with a relatively slow speed can decrease the performance of the homogeneous k-d decomposition. Using the heterogeneous algorithm, the minimum requirements for the added components are relaxed.

G. Large-Scale Run

To confirm, that the load balancing scheme also works for big scenarios and a comparably large amount of ranks, we simulated the LJBig scenario, using 15 Sandy Bridge nodes, 8 Bulldozer nodes and one Westmere node. On each of the Westmere and Bulldozer nodes, 64 processes have been used. On each Sandy Bridge node, 32 processes have been started. The whole simulation thus included 1056 ranks. We measured a speedup of 1.3 when comparing the time-to-solution of the heterogeneous version to the homogeneous version. The maximum theoretical speedup of 1.4 can be calculated through the performance ratios of ranks on the different architectures (1.5:1 (SNB:WSM), 1.9:1 (SNB:BDZ)) and by applying Eq. (4). The observed speedup of 1.3 thus agrees very well with the expected performance gains.

V. Summary & Outlook

Summary: We presented a k-d tree-based load balancing implementation for short-range molecular dynamics. Using it, we achieve good efficiency on heterogeneous systems. We showed this on systems contain Sandy Bridge, Westmere and Bulldozer nodes or a combination of Xeon and Xeon Phi processors. Compared to an implementation that ignores the heterogeneity, it is possible to achieve significant speedups by measuring the performance, detecting performance differences between processes, and allocating the workload dynamically. These speedups are bounded by the performance ratio of the fastest and the slowest process. Our implementation behaves well within the borders of the theoretical limits of our newly introduced simple performance model. The presented implementation needs only a single rebalancing step, as long as homogeneous particle distributions are simulated. The time consumption of the algorithm does therefore not play any significant role. The algorithm has been validated for simulations containing up to ca. 350 million particles, 8 million linked-cells and 1000 processes.

While there exist various plans for heterogeneous supercomputers, there also exist some for homogeneous ones, e.g. the *Aurora* supercomputer. It will use the third generation of Intel Xeon Phi "Knights Hill" as host processors, without any additional Xeon processors. It will provide similar performance as the *Summit* or *Sierra* systems[6]. In either case upon an extension of an existing cluster, there will exist clusters composed of different partitions. Our shown algorithm is able to leverage the full potential of almost any heterogeneous form a cluster can take.

Outlook: In this paper we focused on load balancing for homogeneous particle systems computed on heterogeneous systems. We expect that our load balancing scheme will also work when using heterogeneous particle distributions. We will investigate this in the near future. It is sufficient to rebalance only once for homogeneous particle distributions. In contrast, scenarios which involve heterogeneous particle distributions require dynamic balancing of load, with a rebalancing triggered every 10–100 time steps [6], [7], [10], [14], [11], [12]. For heterogeneous particle systems, we therefore need multiple rebalancing steps. The easiest way to do this is by measuring the performance of the individual processes in the beginning. The rebalancing steps are then carried out according to the calculated cost function and the initially generated performance measures. The performance measurements at the startup of the simulation are susceptible to fluctuations, especially, if only small scenarios are used. Hence it might be useful to remeasure the performance during the simulation. This relaxes the influence of dynamic effects, such as cooling, Turbo Boost/Core or even other processes running on the same node, that hinder the performance. For these dynamic measurements some problems occur (see Sec. IV-C) and need to be circumvented. This can be achieved by modifying either the performance measures, to e.g. include the time spent waiting for the first communication after the force calculation is completed, or by measuring the performance on fixed reference cells. Additionally the partitioning of the process groups can be easily extended, such that any tree-like structure of a cluster can be resembled and speedups achieved. Since a k-d tree structure has been available in *ls1 mardyn*, we used it. Still, comparison with other load balancing approaches or other existing implementations, such as Zoltan [20], would be interesting and shall be investigated in future. Other aspects of the simulation that require improvement are the extension of the cost function to include different types of molecules and sites, as well as time spent for communication. The cost function further does not take the different kinds of neighbor cells into account: more interactions have to be calculated in neighboring cells, that share a face, than for cells, that share an edge or a corner. However, one should note, that the evaluation of that cost function should still be kept simple and efficient.

Additionally, we plan to include communication hiding

[6]http://energy.gov/articles/us-department-energy-awards-200-million-next-generation-supercomputer-argonne-national

and other optimizations, such as neutral territory methods and a hybrid MPI-OpenMP parallelization using our recent OpenMP approach [4], in our simulation code and adapt the load balancing scheme accordingly. Since the k-d approach is applicable for multiple other simulation scenarios it is of interest to verify its usability for other simulation codes.

Acknowledgment

This work is partially supported by the Intel Parallel Computing Center ExScaMIC, and the Federal Ministry of Education and Research, Germany, in the scope of the project "SkaSim - Skalierbare HPC-Software für molekulare Simulationen in der chemischen Industrie", no. 01IH13005F. We thank Leibniz Supercomputing Centre, project "ls1 mardyn IPCC", and the Munich Centre of Advanced Computing, for providing computational resources. P. Neumann acknowledges funding by the German Research Foundation in terms of the priority program *Software for Exascale Computing*.

References

[1] M. Griebel, S. Knapek, and G. Zumbusch, *Numerical Simulation in Molecular Dynamics – Numerics, Algorithms, Parallelization, Applications*, Springer, 2007.

[2] M. Horsch, C. Niethammer, J. Vrabec, H. Hasse, *Computational molecular engineering as an emerging technology in process engineering*, it-Information Technology – Methods and Applications of Informatics and Information Technology vol. 55(3), p.97101, Jun 2013

[3] A. Heinecke, W. Eckhardt, M. Horsch, H.-J. Bungartz, *Supercomputing for Molecular Dynamics Simulations – Handling Multi-Trillion Particles in Nanofluidics*, Springer 2015.

[4] N. Tchipev, A. Wafai, C.W. Glass, W. Eckhardt, A. Heinecke, H.-J. Bungartz, and P. Neumann, *Optimized Force Calculation of Molecular Dynamics Simulations for the Intel Xeon Phi*, in Euro-Par 2015: Parallel Processing Workshops, ser. Lecture Notes in Computer Science, Springer, pp. 774785, 2015, vol 9523.

[5] G. Mie, *Zur kinetischen Theorie der einatomigen Körper*, Annalen der Physik, vol. 316, no. 8, pp. 657–697, 1903.

[6] G.A. Kohring, *Dynamic load balancing for parallelized particle simulations on MIMD computers*, Parallel Comput., vol. 21, pp. 683–693, 1995.

[7] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, *GROMACS4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation*, J. Chem. Theory Comput., vol. 4, pp. 435–447, 2008.

[8] M.J. Abraham, T. Murtola, R. Schulz, S. Páll, J.C. Smith, B. Hess, and E. Lindahl, *GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers*, SoftwareX, vol. 1–2, pp. 19–25, 2015.

[9] L.V. Kalé, M. Bhandarkar, and R. Brunner, *Load Balancing in Parallel Molecular Dynamics*, in *Solving Irregularly Structured Problems in Parallel*, ser. Lecture Notes in Computer Science, Springer, pp. 251–261, 2005, vol 1457.

[10] A. Bhatelé, L.V. Kalé, and S. Kumar, *Dynamic Topology Aware Load Balancing Algorithms for Molecular Dynamics Applications*, in *ICS '09 Proc. of the 23rd International Conference on Supercomputing*, ACM, pp. 110–116, 2009.

[11] W.M. Brown, P. Wang, S.J. Plimpton, and A.N. Tharrington, *Implementing molecular dynamics on hybrid high performance computers – short range forces*, Comput. Phys. Commun., vol. 182, no. 4, pp. 898–911, 2011.

[12] Q. Wu, C. Yang, T. Tang, and L. Xiao, *Exploiting hierarchy parallelism for molecular dynamcis on a petascale heterogeneous system*, J. Parallel Distrib. Comput., vol. 73, pp. 1592–1604, 2013.

[13] Y. Deng, R.F. Peierls, and C. Rivera, *An Adaptive Load Balancing Method for Parallel Molecular Dynamics Simulations*, J. Comput. Phys., vol. 161, pp. 250–263, 2000.

[14] J.L. Fattebert, D.F. Richards, and J.N. Glosli, *Dynamic load balancing algorithm for Molecular Dynamics based on Voronoi cells domain decompositions*, Comput. Phys. Commun., vol. 183, no. 12, pp. 2608–2615, 2012.

[15] C. Begau and G. Sutmann, *Adaptive Dynamic Load-Balancing with Irregular Domain Decomposition for Particle Simulations*, Comput. Phys. Commun., vol. 190, pp. 51–61, 2015.

[16] L. Nyland, J. Prins, R.H. Yun, J. Hermans, H.-C. Kum, and L. Wang, *Modeling Dynamic Load Balancing in Molecular Dynamics to Achieve Scalable Parallel Execution*, in *Solving Irregularly Structured Problems in Parallel*, ser. Lecture Notes in Computer Science, Springer, pp. 356–365, 2005, vol. 1457.

[17] J.L. Bentley, *Multidimensional Binary Search Trees Used for Associative Searching*, Commun. ACM, 1975.

[18] C. Niethammer, S. Becker, M. Bernreuther, M. Buchholz, W. Eckhardt, A. Heinecke, S. Werth, H.-J. Bungartz, C.W. Glass, H. Hasse, J. Vrabec, and M. Horsch, *ls1 mardyn: The Massively Parallel Molecular Dynamics Code for Large Systems*, J. Chem. Theory Comput., vol. 10, no. 10, pp. 4455–4464, 2014.

[19] M. Buchholz, *Framework zur Parallelisierung von Molekulardynamiksimulationen in verfahrenstechnischen Anwendungen*, Verlag Dr. Hut, München, August 2010.

[20] E.G. Boman, Ü.V. Çatalyürek, C. Chevalier and K.D. Devine, *The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering and coloring*, Scientific Programming, vol. 20, no. 2, pp. 129-150, 2012.