# Performance evaluation and optimization of the ls1-MarDyn Molecular Dynamics code on the Cray XE6

Christoph Niethammer
*High Performance Computing Center Stuttgart (HLRS),*
*University of Stuttgart,*
*70550 Stuttgart, Germany*
*Email: niethammer@hlrs.de*

***Abstract*—Today Molecular Dynamics (MD) Simulations are a key tool in many research and industry areas: Biochemistry, solid state physics, chemical engineering, to just mention some. While in the past MD was a playground for some very simple problems, the ever-increasing compute power of super computers lets handle more and more complex problems: It allows increasing number of particles and more sophisticated molecular models which were too compute intensive in the past. In this paper we will present performance studies and results obtained with the ls1-MarDyn MD code on the new Hermit System (Cray XE6) at HLRS. The code's scalability up to the full system with 100.000 cores will be discussed as well as a comparison to other platforms. Furthermore we will present in detail code analysis using the Cray software environment. From the obtained results we will discuss further improvements which will be necessary for upcoming systems in the post petascale era.**

***Keywords*-performance, optimization, Cray, XE6, molecular dynamics**

## I. Introduction

Molecular Dynamics (MD) Simulations are a key tool in many research and industry areas such as biochemistry, solid state physics or chemical engineering, to just mention some. While in the past MD was a playground for some very simple problems, the ever-increasing compute power of super computers lets handle more and more complex problems today. This allows not only to simulate larger systems with higher number of particles but also more sophisticated molecular models which were too compute intensive in the past. The basics of MD simulations will be introduced in section II

In this paper we will present performance studies and results obtained with the ls1-MarDyn MD code on the new Hermit System (Cray XE6) at HLRS. ls1-MarDyn is a MD code used to simulate nano fluids. It allows the study of flow phenomena or condensation processes at the nano scale. As the density of fluids is high it is essential to simulate large systems.

ls1-MarDyn is written in C++ and entirely MPI parallel. It is build in a modular fashion which allows the separation of physical calculations and parallel algorithms allowing easy collaboration of experts in both fields. A more detailed description of the program will be given in section III

Porting and optimization of ls1-MarDyn for the Cray XE6 system started with a detailed code analysis. At first we present a comparison of the code on different platforms. This includes a study of the different compiler environments on the Cray system which can have a major impacts on the overall performance as will be shown in section V-A.

Further on we present results of a more in detail performance analysis using the Cray performance tools. Here we identified different bottlenecks. As will be seen communication and I/O are the main problems for limited scalability. This requires some code modifications for the new Cray system. E.g. reducing global communication which can hurt performance dramatically on such large systems. Improvements in the MPI communication will be discussed in section V-B of this paper. A second major point is the I/O part, e.g. used for reading/writing checkpoints. Improvements gained in this part of the program will be presented in section V-C. As we show ls1-MarDyn can scale up to the full system with 100.000 cores applying these modifications today.

At the end we will discuss further improvements which will be necessary for upcoming systems in the post petascale era.

## II. Basics

Starting point for all classical MD simulations is Newton's 2nd law which states, that the motion of each molecule is determined by the simple differential equation

$$m_i \dot{v}_i = m_i \ddot{x}_i = F_i \tag{1}$$

where $m_i$ is the mass of the molecule, $x_i$ and $v_i$ its position and velocity as well as $F_i$ the force acting on it. In case of molecules with an angular momentum a similar equation determines the time evolution of the molecules rotation.

The force $F_i$ is given by the superposition of all forces between molecule $i$ and its interaction partners $j$. The interaction between molecules are determined by the interaction potential $U(r)$ where $r_{ij}$ is the distance of the two

molecules:

$$F_i = \sum_{j \neq i} F_{ij} = -\sum_{j \neq i} \nabla U(r_{ij}) \tag{2}$$

Depending on the studied problem different potentials are used to model the system. The most common potentials in MD simulations of solids, liquids and gases are the Lennard-Jones 12-6 potential, Coulomb interactions and their higher order electrostatic interactions like those of dipoles and quadrupoles as well as the Tersoff potential [1], [2].

The equations of motion for all particles in the system build up a large coupled differential equation system. This system cannot be solved analytically. MD simulations approximate the solution by a time discretization. In each time step the current forces on all molecules are calculated. Then the velocities and positions at the next time step are determined from the forces, positions and velocities at the current time step. The sequence of these states build up a trajectory in the physical phase space. Different physical properties can be gained from this like the ensemble properties temperature, pressure, chemical potential or time dependent properties like the diffusion coefficient or condensation rate.

## III. Program description

Section II introduced different physical potentials. In the case of a potential, which changes significantly only in a short range and stays nearly constant in the far part, some simplifications can be introduced. For these short range potentials the potential can be split up into a near field and a far filed part. While the near field part will be calculated exactly within a cutoff radius $r_c$, the far field part will not be evaluated for every interaction but taken into account by a far field correction.

This splitting allows now to modify the interaction algorithm. Only molecules within the cutoff radius have to be considered. Decomposing the simulation domain into cells of the dimension $r_c$ guarantees that all interaction partners of a molecule are found in the cell of the molecule itself and the direct neighbouring cells. By this the computational costs can be reduced from $\mathcal{O}(N^2)$ required to compute all the interactions between $N$ molecules to $\mathcal{O}(N)$ required to compute all interactions within the cutoff distance.[1]

In ls1-MarDyn molecules are modeled as rigid rotators and are build up from one or more interaction sites. Different site types are implemented for all the interaction potentials: Lennard-Jones, charge, dipole, quadrupole and Tersoff. The interaction of two molecules is calculated from the interaction between all the sites within the interacting molecules.

## IV. The Hermit system

The Hermit system at HLRS used for the studies is a Cray XE6 with 3552 compute nodes. Each compute node is equipped with two AMD Opteron 6276 (Interlagos) processors resulting in a total of 113664 cores. The Interlagos processor comes along with new instructions allowing the acceleration of floating point operations. The first one are the new advanced vector extension (AVX) SIMD operations which allow four double precision floating point operations per cycle and introduce a non-destructive three-operand form $c = a + b$. The second one is the fused multiply-add instruction FMA4 allowing the computation of $d = a + b \times c$ in a single cycle. By this the peak performance of the entire system is about 1 PFLOPS/s. Each node is equipped with at least 32GB RAM.[3]

The nodes are connected in a 3D torus network via the Cray Gemini interconnect[4]. This interconnect allows for very low latencies and high bandwidths. For MPI messages the latencies are around $1,4\mu$s and the bandwidth between two nodes can become as high as 6GB/s.

## V. Program analysis

The over all performance of ls1-MarDyn is influenced by three parts: Computation, communication and I/O. As ls1-MarDyn is written as a portable C++ program without any hand tuned assembler code, the computational performance is heavily influenced by the compiler. We will study the impact of the used compiler environment in part V-A. The main goal of the ls1-MarDyn MD code are large systems. Their calculation require a high level of parallelism so that the communication within the program has a major impact on the program speed. The communication is influenced at two points: At first by the MPI usage and implementation within the program, and second by MPI library parameters which can be used to optimize the MPI environment. This part will be examined in part V-B. The last part which has to be considered is I/O. This is influenced by the usage of I/O calls within the program and the file system. We will study both aspects in part V-C

To get a rough overview of the performance Cray pat can be used. This provides a profile as well as detailed information from hardwaer performance counters. As can be seen in figure 1....

The over all data 1 and data 2 cache hit and miss ratio reported by craypat are $100.0\%$ or $0.0\%$ respectively[1]. So the basic data structures are well designed with respect to the used algorithms.

### A. Compiler environment

As mentioned in sec. IV, Hermit is equipped with AMD Opteron 6276 (Interlagos) processors. As these processors have a new architectural layout, compilers may have a great impact on the system performance. In this part benchmarking results with the different programming environments are presented.

The compiler versions as well as the used compiler options of the different programming environments are listed

[1]For the craypat analysis the gnu programming environment was used.

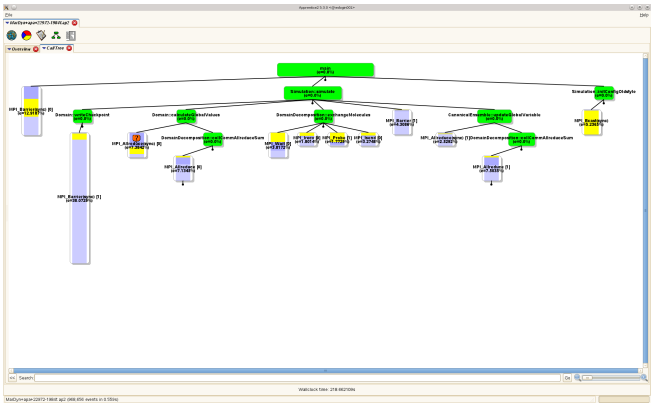


Figure 1. Initial call tree of the ls1-MarDyn program obtained with a Cray pat sampling experiment. The program was run using 8192 processes. The initial and final I/O (including MPI_Barrier calls) take a big part of the time because the program was run only for a view time steps.

in table I. The listed options were selected from a set of different combinations being the once producing the fastest code for the specific compiler.

| Compiler | Version | Options |
|---|---|---|
| GCC | 4.6.2 | -O3 |
| Cray | 8.0.3 | -O3 |
| Intel | 12.1.3 | -O3 |
| PGI | 12.2 | -fast -Mipa=fast,inline -Minline=levels:10 |

Table I
COMPILER VERSIONS IN THE DIFFERENT PROGRAMMING ENVIRONMENTS AND USED OPTIONS ON THE HERMIT SYSTEM.

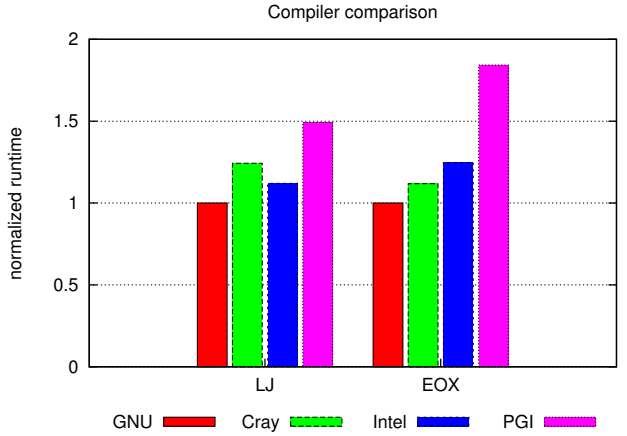



Figure 2. Normalized execution times of a single centered Lennard-Jones (LJ) and an Ethylen Oxide (EOX) system for the different compiler versions.

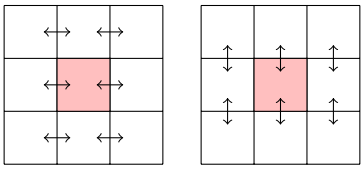
Figure 2 shows the normalized execution times for two physical systems. The GNU compiler produces the fastest code closely followed by the Cray and Intel compiler. With up to 84.4% longer execution times in the EOX scenario, the PGI compiler produces by far the slowest code.

### B. MPI Communication

The parallel programming model used in ls1-MarDyn is MPI. The parallelisation is based on a domain decomposition: Each process gets a spatial domain of the simulated system. While ls1-MarDyn allows arbitrary implementations of domain decompositions, we focus on the default implementation using equal sized cuboidal subdomains.

As the calculation of the potentials and forces requires information about the molecules' spatial neighbours, halo areas are required which cache the information about the surrounding of the subdomain owned by a process. In each time step molecules crossing subdomain boundaries have to be transfered between the processes as well as the molecules in the halo areas.

The communication pattern used to exchange the particle data consists out of 3 communication phases. In each phase the particles are communicated along one of the directions x, y or z as shown in figure 3. This communication scheme reduces the number of send and receive operations and can make usage of a 3D network topology. The drawback coming along with this communication scheme is less opportunity for overlap in the communication as each phase has to be completed before the next one can start. The implementation of this part within ls1-MarDyn makes use of non blocking send and receive operations in combination with overlap of computation and communication making improvements in this part unlikely.



(a) x-direction (b) y-direction

Figure 3. MPI communication pattern used for the molecule exchange between the sub domains in each time step. Representation is limited to the x- and y-direction. The last communication in z direction is done respectively.

So this is not the limiting factor in ls1-MarDyn as can be seen in figure 1: Craypat reports an overall number of 4 places with MPI_Allreduce calls. This is taking up around 20% of the computational part for the given example with 2 million molecules on 8192 cores. Craypat also reports long times for the MPI_Barrier call in the checkpoint writing part which look worse but will not be as critical because checkpointing was enabled for every time step – which will not be the case in a real production environment.

*1) Evaluation and optimization of reduction operations in ls1-MarDyn:* Within ls1-MarDyn global values like energy, temperature or pressure have to be calculated in each time step. This requires reduction operations summing up all the local values of all the processes' subdomains. Looking at figure 1 we see that these collective communication

operations require most of the communication time of ls1-MarDyn. Looking into the source code shows that multiple all-reduction calls are occurring in sequence. As this collective operation limits the scalability of the program these calls should be avoided as much as possible.

Using a derived data type in combination with a custom reduction function allows to combine the sequence of MPI_Allreduce calls for all the variables into a single MPI_Allreduce call. We refer to the implementation using multiple MPI_Allreduce calls as not-agglomerated and to the one using a derived data type as agglomerated.

| PEs | not-agglomerated | agglomerated | improvement |
|---|---|---|---|
| 4096 | 68.57 | 61.83 | 9.83% |
| 8192 | 45.51 | 41.34 | 9.18% |
| 16374 | 41.17 | 36.95 | 11.95% |

Table II
EXECUTION TIMES OF LS1-MARDYN USING ONE MPI REDUCTION FOR EVERY GLOBAL VARIABLE (NOT AGGLOMERATED) AND THE VERSION USING ONE MPI REDUCTION WITH A DERIVED DATA TYPE (AGGLOMERATED).

The obtained results for both versions are presented in table II. As can be seen the agglomerated version reduces the execution time for the given setup by around 10% compared to the not-agglomerated version.

*2) Evaluation and tuning of the MPI environment:* The MPI library used in all programming environments is Cray MPI version 5.4.4 which is based on MPICH2 version 1.3.1. It is optimized for the Cray Gemini interconnect and provides a variety of tuning and steering parameters. Some of them can be derived using the craypat tools. In the following the effect of some of these parameters is evaluated.

As described at the beginning of this section, ls1-MarDyn uses a 3D domain decomposition scheme for the MPI parallelisation. Because the underlying network topology of Hermit is a 3D torus the process topology may be mapped to the network topology[5]. To place the processes on respective compute nodes different strategies are available in the MPI runtime system. The method can be chosen with the `MPICH_RANK_REORDER_METHOD` environment variable. Available methods are round-robin (0), SMP-style (1), folded-rank (2) and custom rank placement (3).

As the rank ordering of processes within the domain is known inside ls1-MarDyn, manual assignment is possible. An optimized rank placement file can be obtained with the grid_order tool. An optimized placement can also be obtained automatically with the Cay performance tools. In table III results for the different placement methods are presented. With a reduction of the execution time by 3% the results show only little effect of the placement onto the overall runtime. This indicates that the overlap of communication with computation can hide latency and bandwidth problems within the code effectively.

| RANK_ORDER | 4096 PEs |
|---|---|
| round-robin | 15.8(7) |
| SMP-style | 15.5(3) |
| folded-rank | 15.9(7) |
| custom (grid-order) | 15.4(6) |
| custom (craypat) | 15.1(2) |

Table III
EXECUTION TIMES OF THE MAIN LOOP WITHIN LS1-MARDYN FOR DIFFERENT RANK ORDER STRATEGIES. NUMBERS IN BRACKETS STATE THE UNCERTAINTY OF THE RESULT WHICH WAS OBTAINED AS AVERAGE OVER MULTIPLE RUNS.

Another possibility to steer the MPI environment is the tuning of the already discussed reduction operations. Of special interest may be here the parameters `MPICH_USE_DMAPP_COLL` which enables optimized DMAPP[2] collective algorithms, `MPICH_COLL_SYNC` which performs a barrier before each MPI collective and `MPICH_REDUCE_NO_SMP` which disables smp-aware algorithms. Results for their usage are given in table IV. As can be seen most of them have no effect. Only `MPICH_COLL_SYNC` results in an increases execution time – as one may expect on a NUMA system like Hermit.

| | 4096 PEs |
|---|---|
| none | 15.5(0) |
| `MPICH_COLL_SYNC` | 16.9(4) |
| `MPICH_USE_DMAPP_COLL` | 15.(58) |
| `MPICH_REDUCE_NO_SMP` | 15.5(2) |

Table IV
INFLUENCE OF DIFFERENT MPICH TUNING PARAMETERS.

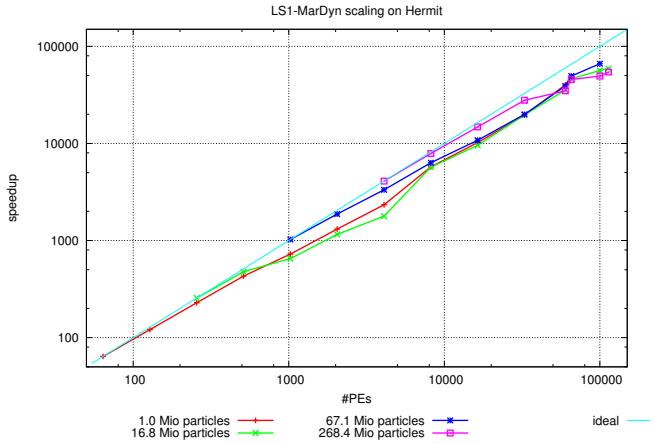



Figure 4. Scaling of ls1-MarDyn on the Hermit system for different numbers of molecules.

[2]DMAPP (Distributed Memory Application) is a new programming API developed which can make use of one-sided communication in Cray Baker networks.

### C. I/O

The studies so far considered only the program parts including calculation and communication. While in normal programs I/O may not be important it can become a real problem at scale. Concurrent access to a single file by 100.000 processes or parallel access to 100.000 files by each process stresses the filesystem badly. In the following the I/O system of ls1-MarDyn will be discussed.

In ls1-MarDyn I/O occurs at different points in the program. At first there is the input file which has to be read in at program start. This input file consists out of two files: A general description file for the simulation parameters and a phase space file holding component information as well as molecule positions and velocities. The latter can be a restart file from a previous run or generated with an external program. The second I/O part in ls1-MarDyn is the output of results during the program run. A generalized interface allows to save data in a regular interval. Data written by the output modules can be either some single values like the current pressure, sets of multiple values e.g. for the radial distribution function or large data sets required e.g. for restart or visualization. The last I/O operation performed during a program run writes out the final result and stores a restart file which can be used later on.

In figure 5 the times required for the different I/O operations performed withing ls1-MarDyn are shown. The times are measured within the program itself using internal timers. While the I/O times would be perfectly constant, they increase considerably with the number of processes.

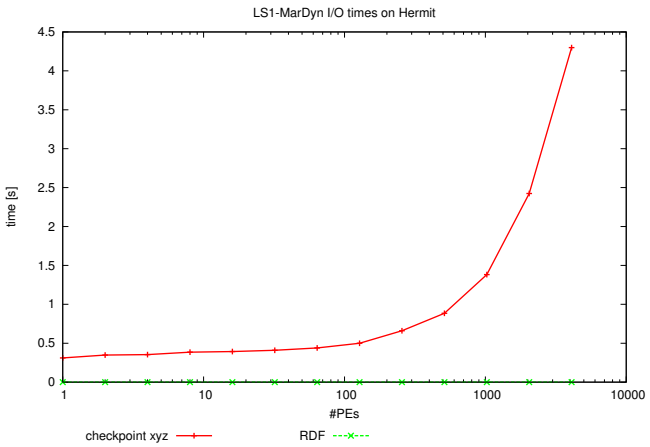



Figure 5. Times for the different I/O parts in ls1-MarDyn. The LJ example was used in combination with intermediate I/O of the radial distribution function output and restart files in every time step. The times for the I/O within the main loop are mean values over all time steps.

To examine the origin of this problem a more detailed analysis was done using the Cray *iobuf* tool. This tool provides detailed information about the I/O operations on a per process level. Information include for example I/O times, the amount of data read/written, the number and size of used I/O buffers or the number of partial writes. Results for a single process are provided in 6.

```
PE 1086: File "lj40000_t300.inp"
              Calls      Seconds      MB/sec
Read              1     2.185619    0.003748
Open              1     0.278424
Close             1     0.731858
Buffer Read       2     3.685361    0.569049
I/O Wait          2     2.184243    0.960128
Buffers used          2 (2 MB)
Prefetches            1
```

Figure 6. Results reported by the *iobuf* tool for the access to a restart file by a single process in a 4096 PE run.

As can be seen in the output of *iobuf*, in the original program each process opened the phase space file and read in all molecules just disregarding the molecules outside the local domain. Parallel access to a single file is very bad. So this was improved by letting only rank 0 read in the phase space file which then broadcasts the molecules to the other processes in chunks. In figure 7 the improved I/O times are presented. They are constant over the number of processes and allow now to (re-)start even large jobs.

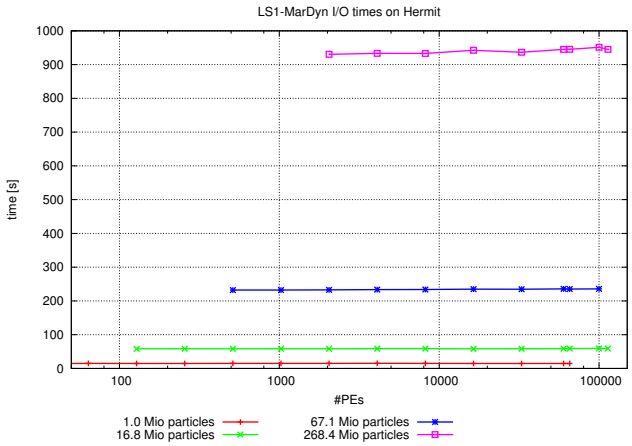



Figure 7. Times for the new phase space file I/O part in ls1-MarDyn. The same examples as in figure 5 were used.

## VI. Conclusion

The GNU programming environment delivered the best sequential program performance for ls1-MarDyn on Hermit. This is a bit surprising but may be due to the fact that ls1-MarDyn is written in C++.

The Optimization of the global communication by usage of derived data types in combination with custom reduction operation in the MPI_Allreduce calls provided roughtly 10% speedup of the execution time for the used examples. The

optimization of the MPI rank placement provided only a smaller benefit due to the already overlapped communication and computation in the particle exchange part of ls1-MarDyn.

The new MPI I/O with broadcasted input provides a clear win to the formerly used approach where each process read in the input file separately. It is now possible to start ls1-MarDyn with any input/restart file at scale. The output plugins were also improved by sending their data to a single process which then writes the output to the file. As a serialization occures here within the program the improvement is not of the same size as for the input but noticeable.

## ACKNOWLEDGMENT

This work was supported by the German Federal Ministry of Education and Research (BMBF) within the program "IKT 2020 - Forschung für Innovationen" in the call "HPC-Software für skalierbare Parallelrechner".

## REFERENCES

[1] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*. Oxford: Clarendon, 1987.

[2] J. Tersoff, "Empirical interatomic potential for carbon, with applications to amorphous carbon," *Phys. Rev. Lett.*, vol. 61, pp. 2879–2882, Dec 1988.

[3] [Online]. Available: http://www.hlrs.de/systems/platforms/cray-xe6-hermit/

[4] R. Alverson, D. Roweth, and L. Kaplan, "The gemini system interconnect," in *Proceedings of the 2010 18th IEEE Symposium on High Performance Interconnects*, ser. HOTI '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 83–87.

[5] A. Bhatelé, L. V. Kalé, and S. Kumar, "Dynamic topology aware load balancing algorithms for molecular dynamics applications," in *Proceedings of the 23rd international conference on Supercomputing*, ser. ICS '09. New York, NY, USA: ACM, 2009, pp. 110–116.