Vectorization of Multi-Center, Highly-Parallel Rigid-Body Molecular Dynamics Simulations

Wolfgang Eckhardt, Alexander Heinecke, Wolfgang Hölzl and Hans-Joachim Bungartz
Technische Universität München
{eckhardw-heinecke-hoelzlw-bungartz}@in.tum.de

1. INTRODUCTION

The software package ls1 mardyn has been developed in a multi-disciplinary cooperation of chemical engineers and computer scientists. Targeting applications such as nano-scale flow simulations or the investigation of nucleation scenarios [4], the simulation of the fluids of interest usually requires large numbers of comparably simple, small molecules. Therefore, ls1 mardyn implements true rigid-body motion and solves the equations of motion by the Rotational Leapfrog Algorithm, in contrast to most other well-known codes that treat all parts of molecules separately and apply constraint motion algorithms such as RATTLE or SHAKE in the time integration [3]. This scheme not only facilitates parallelisation, but also allows for more efficient interaction computation. Targeting large molecular systems ($10^5 < N < 10^9$), an efficient implementation for current x86 processor architectures is important, due to their large market share in HPC. While much work has been spent on the efficient implementation on vector machines already decades ago, current implementations for SIMD extensions such as SSE and AVX simply vectorize across the spatial coordinates [5–7], thereby limiting possible speed-ups by design. In this poster, we extend our preceding work on a highly efficient vectorization using AVX instructions for single center molecules [2] and now focus on the extensions for multi-center particles.

2. IMPLEMENTATION

We first sketch the basic implementation for single-centered molecules. The most compute intensive part is the computation of Lennard-Jones-12-6 (LJ-12-6) interactions between particles, where particles interact, if the distance between their centers of mass is less than a cut-off radius r_c. Our implementation is based on a modified version of the Linked-Cells algorithm.

The calculation is performed on particle pairs, therefore we broadcast-load the required data of one particle in the first register, the second register is filled by data from four other particles, see Fig. 1. Dealing with four particle pairs at once, we can theoretically reduce the number of instructions by a factor of four. Since the force calculation may be required for all, some or none of the pairs in the vector register, we need to apply some pre- and post-processing performed by regular logical operations in order to mask out unnecessary force calculations.

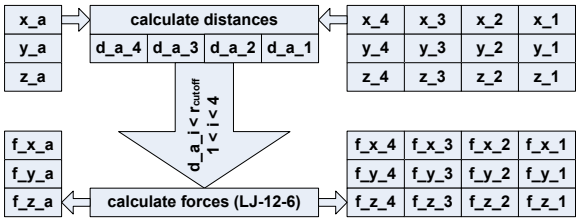

Figure 1: The vectorization of the LJ-12-6 force calculation is optimized by duplicating one particle and streaming four other particles.

The data have to be prepared for a vectorized processing, which can be implemented by a sliding window, moving through the domain: after a cell has been searched for interacting particles for the first time in a time step, its data will be required for several successive force calculations with particles in neighboring cells. While the cells in the window are accessed several times, they naturally move in and out of the window in FIFO order, see Fig. 2.

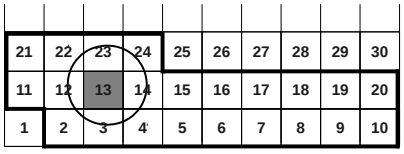

Figure 2: Particles in cells in the window will be accessed several times, cells 2 through 23 are covered by the window in FIFO order.

Particle data outside the sliding window are stored in form of C++ objects in AoS manner. Per cell, particle objects are stored in dynamic arrays. When the sliding window is shifted further and covers a new cell, the positions and velocities of the particles in that cell are converted to SoA representation. The force calculation is now performed on the particles as described above. When a cell has been considered for the last time during an iteration, its particles are converted back to AoS layout.

A straightforward way to handle particles with multiple centers is to treat every site independently. In our computational model, this exposes overheads on the distance calculation between two particles since the distance is only needed on the particle level to evaluate the cut-off condition. In contrast, we calculate the particle-particle distances once and create a sophisticated look-up table as shown in Fig. 3. Note that this is not a full look-up table: within the current cell pair we calculate for a particle i in the first cell the distances with all particles j in the second cell. If at least one distance is smaller than r_c we start an expansion phase that duplicates the decision if a force calculation is required or not according to the number of centers of all particles in the second cell. Since we broadcast particles in the first cell (see Fig. 1) this pumping step is not required for i. If no interaction between i and all js happens, we continue with particle $i+1$ in first cell without setting up the look-up table.

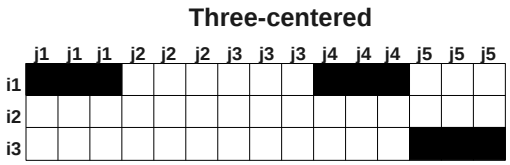

Figure 3: Particle distances are calculated on a particle i-j basis and on the fly extended to a particles i - sites j row, if required.

For the MPI parallelisation, we employ a spatial domain decomposition scheme. For n processes, the domain is divided in n equally-sized sub-domains, which are assigned to one process each. Every sub-domain is surrounded by a layer of ghost cells, residing on neighboring processes, so the particles at the process boundaries have to be exchanged at the beginning of each time step. Thus, main communication takes place only locally with neighboring processes. At the end of a time step, a global MPI_Allreduce()-operation has to be performed to compute global statistical values such as temperature and potential energy.

Plugging everything together, we obtain the performance shown in Fig. 4 on SuperMUC[1] for a 10^7 acetone molecules scenario. Each molecule comprises four LJ centers and our vectorizations gains a speed-up of slightly larger than 3 of the scalar baseline using AVX with its four element wide double precision vector-registers.

3. ON-GOING RESEARCH/OUTLOOK

Our final target applications will be characterized by strong load imbalance (formation of small nuclei, i.e. nanoscopic liquid droplets). Therefore, an efficient load balancing scheme based on KD-Trees has been implemented in ls1 mardyn, cf. [1]. There, the domain is recursively bisected and the resulting sub-domains are assigned a number of processes so that the overall load imbalance is minimized. Currently we are optimizing that implementation avoiding this to become a bottleneck. Based on the present implementation, we plan to introduce static load balancing, allowing more aggressive optimizations. Since the load distribution within the domain will not change dramatically, this is expected to further improve scalability for heterogeneous scenarios.

[1] `http://www.lrz.de/services/compute/supermuc/`

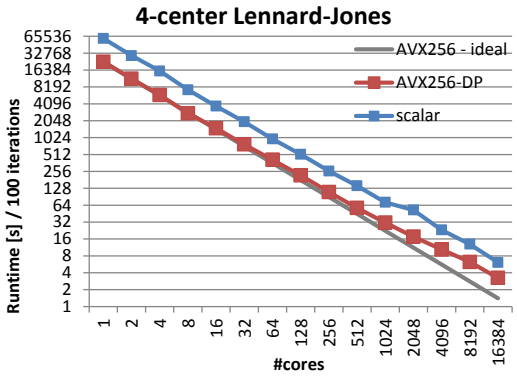

Figure 4: Performance of a 10.7 million acetone molecules (quad-center) simulation on SuperMUC. Using AVX (double precision) a speed-up of slightly larger than 3 is achieved by our vectorization.

From a hardware perspective, we are working on an Intel Xeon Phi port employing gather/scatter instructions. Early performance results are included in our poster draft.

4. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant Number 1137097 and by the University of Tennessee through the Beacon Project. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the University of Tennessee. This work has been partly funded by the German federal ministry of education and research (BMBF) in the project SkaSim within the research programme IKT 2020 under Grant Number 01IH13005F.

5. REFERENCES

[1] M. Buchholz et al. Software design for a highly parallel molecular dynamics simulation framework in chemical engineering. *Journal of Computational Science*, 2(2):124–129, May 2011.

[2] W. Eckhardt, A. Heinecke, et al. 591 TFLOPS Multi-Trillion Particles Simulation on SuperMUC. In *Int. Supercomp. Conf. (ISC) Proceedings 2013*, volume 7905 of *LNCS*, pages 1–12, Heidelberg, Germany, June 2013. Springer.

[3] B. Hess et al. Gromacs 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *Journal of Chemical Theory and Computation*, 4(3):435–447, March 2008.

[4] M. Horsch et al. Homogeneous nucleation in supersaturated vapors of methane, ethane, and carbon dioxide predicted by brute force molecular dynamics. *The Journal of Chemical Physics*, 128(16):164510, 2008.

[5] E. Lindahl et al. Gromacs 3.0: a package for molecular simulation and trajectory analysis. *Journal of Molecular Modeling*, 7:306–317, 2001.

[6] S. Olivier et al. Porting the GROMACS Molecular Dynamics Code to the Cell Processor. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, march 2007.

[7] L. Peng et al. Exploiting hierarchical parallelisms for molecular dynamics simulation on multicore clusters. *The Journal of Supercomputing*, 57:20–33, 2011.