Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

SHAPE Project Scienomics Development of Chameleon Monte Carlo code for HPC: Toward Realistic Modelling of Complex Composite Systems

Orestis Alexiadis[a], Xenophon Krokidis[a], Isabelle Dupays[b], Sylvie Therond[b], Thibaut Véry[b]*

[a]Scienomics

[b]Institut du développement et des ressources en informatique scientifique (IDRIS), CNRS, Université Paris-Saclay, F-91403, Orsay, France

Abstract

Scienomics is developing Chameleon (Chain Altering Monte-Carlo) to perform Monte-Carlo (MC) simulations of several types of chemical systems such as polymers, nanomaterials or composite materials. Realistic modelling of materials requires that the systems are large enough to get reliable results for the properties computed. Chameleon uses an all-atoms model meaning that the number of interactions needed to compute the physical properties, hence the computational resources, increases fast as the number of atoms in the system increases. In this project the work of Scienomics and IDRIS engineers allowed Chameleon to treat larger systems faster thanks to a dramatic improvement of serial and parallel performances.

Introduction

The development and optimisation of new materials for a wide range of applications requires a lot of manpower and financial means. It is therefore important to try to predict how the properties of the materials will change when the composition of an existing compound is changed or when *de novo* materials are designed.

With the increase in computing power, it is now possible to perform realistic simulations of the properties of new classes of materials. Simulations are an important scientific tool and can greatly help in the designing process.

With this collaboration, IDRIS provided Scienomics with an access to supercomputers and computing hours as well as expertise in parallel code execution.

Access to a supercomputing centre provided Scienomics with the means to improve and/or validate three different aspects of the Chameleon code:

1- Improve Chameleon performance for single processor simulations
2- Improve Chameleon parallelization:
    a. Increase the part of the code parallelized
    b. Improve the parallelization scaling for large number of computing cores
    c. Test hybrid implementation (not performed during the project)
3- Leverage the large supercomputer capabilities to run long simulations and a number of simulations statistically meaningful.

* Corresponding author. *E-Mail address*: thibaut.very@idris.fr

Scienomics

Scienomics, established in 2004, is a French company based in Paris with an international presence (Greece, Germany, England, US) and customers worldwide. Scienomics specializes in the development of materials modelling and simulation Software. Its flagship is MAPS (Materials Processes and Simulation Platform), a graphical user interface to build, simulate and analyse several complex materials, thus providing a unique combination of modelling and simulation technology through recent advances in theoretical and computational methods. One of the most powerful tools available in MAPS is Chameleon (Chain Altering Monte-Carlo), an application which is a generalized Connectivity Altering Monte-Carlo molecular modelling code designed predominantly for simulating large molecular weight polymer melts and complex molecules using atom-based force fields. Chameleon is a simulation software which combines the widely used Markov Chain Monte-Carlo (MCMC) technique and recent advances in connectivity altering algorithms. It is capable of relaxing increasingly complex and large modern materials, characteristics that cannot be addressed through standard techniques. The code integrates all these key features in a compact, universal Monte-Carlo tool with many additional new features in order to deal with polymer types and many different force fields (united-atom and all-atom). Chameleon also supports coarse-grained representation of molecules to be used in multiscale modelling techniques.

Presentation of IDRIS computing resources

The Institute for Development and Resources in Intensive Scientific Computing (IDRIS) is a unit of the French Centre for Scientific Research (CNRS) and its major centre for high performance calculation. The computations for this project were carried out on two machines hosted by IDRIS. The technical characteristics of these machines are the following:

- Ada: IBM x3750-M4 compute nodes
    - 4 Intel Sandy Bridge E5-4650 8-core sockets at 2.7 GHz (32 cores/node)
    - 4GiB or 8GiB per core of memory
    - InfiniBand FDR10 Mellanox network
    - GPFS parallel file system
- Ouessant: IBM OpenPower prototype (3 Firestone and 12 Minsky nodes)
    - 2 IBM's POWER8+ 10-core sockets (20 physical cores/node)
    - Up to 8 simultaneous multithreading (SMT) per core
    - 128 GiB of memory
    - 4 Nvidia P100 GPUs (16 GiB of memory)
    - Mellanox EDR IB CAPI interconnexion network

Chameleon

Chameleon is a simulation tool based on the principles of the Markov Chain Monte-Carlo technique. As with any simulation method that aims to explore efficiently those regions of phase space (position-momentum space) that are most likely to physically occur, the Monte-Carlo (MC) method samples phase space by generating new configurations that satisfy some energetic criteria which ensure the states visited are indeed physically realizable states. New configurations are generated using a set of Monte-Carlo moves (in some cases MC moves are unphysical or even fictitious) which aim to equilibrate a set of desired characteristics of the system.

The moves can vary from simple to sophisticated and complex depending on the specific needs of the study. The majority of moves are connected with an ensemble of statistical mechanics. For example, the volume fluctuation move, which is solely responsible for equilibrating the density of the system is implemented in the isothermal-isobaric ensemble (NPT) and the Connectivity Altering End-Bridging move in the semi-grand canonical ensemble (μ*NVT).

Connectivity Altering moves are significant components of Chameleon since they are the most efficient and drastic ones and the only available moves capable of equilibrating long-chain polymer systems. The use of certain moves depends on the size, architecture and chemical constitution of the system.

Chameleon is implemented through a very flexible and comprehensive graphical user interface (GUI) integrated in MAPS. The GUI provides the user with all the necessary information for interaction with the program (statistical ensemble, initial simulation conditions, MC moves to use, etc.). At the MC algorithm level, information from the GUI serves as an input for performing the simulation.

Every Monte-Carlo iteration involves a random selection of a move from the set of MC moves that the user has requested. By performing the selected move a new configuration $\mathbf{R_n}$ of the system is generated representing a possible new state of the system. The energy of the $\mathbf{R_n}$ configuration is estimated based on the assigned force-field that has been selected by the user. To be accepted, the new configuration must fulfill one of the two conditions:

- the energy of the $\mathbf{R_n}$ configuration is smaller than the energy of the previous state configuration $\mathbf{R_{n-1}}$ (Markov Chain process)
- the energy of the new configuration is larger than the one of the configuration. A random number is generated and is lower compared to some metric.

Then the $\mathbf{R_n}$ configuration is set as the new state of the system;

Otherwise, the system returns to its previous state ($\mathbf{R_{n-1}}$ configuration) and a new MC move is attempted. This process continues until the energy, or any other property of interest in the system, is equilibrated (values fluctuating around a plateau value).

Description of code versions

The part of the Chameleon code which is parallelized is the **forcefield** class; that is the functions that are responsible for calculating the non-bonded energy among atom pairs inside the system. These functions are divided into three categories:

1. Calculation of non-bonded interactions of a single atom with the rest of the system (F_{single})
2. Calculation of non-bonded interactions of a list of atoms with the rest of the system (F_{latoms})
3. Calculation of all non-bonded interactions inside the system (F_{total})

For every atom in the system, the effective interacting neighbours are calculated, stored and updated during the simulation through the use of Verlet lists. With Verlet lists the usual $O(n^2)$ calculation for the estimation of the total energy of the system (using the F_{total} function) is reduced to an $O(n \log(n))$ algorithm.

Since most of the Monte-Carlo moves induce changes in a limited number of atoms in the system (for the simplest case of a linear chain, no more than 9 atoms can change simultaneously during an MC move), the F_{single} and F_{latoms} functions the most used in the calculations. The F_{total} function is only used in the case of a volume fluctuation MC move (this move uniformly scales all atom or molecule positions after a volume change in the simulation box) which in a normal MC simulation would not exceed 2% all the moves. However, the F_{total} function is the most efficient to parallelize since its calculation involves a **double for loop** in contrast to the F_{single} and F_{latoms} functions whose estimation is simplified to a **single for loop**. Based on these considerations, the following Chameleon versions were provided and tested:

- V1.0
  - OpenMP parallelization, only implemented in the F_{total} function. A very small portion of the code is actually parallelized, since in the test cases provided the Volume Fluctuation move is used with 1.5% of the total percentage of MC moves.
- V1.1
  - OpenMP parallelization was extended to energy calculation functions of a single atom (F_{single}) and a list of atoms (F_{latoms}).
- V2.0
  - Added dynamic scheduling (**schedule (dynamic)**) in the "**for reduction"** loops of the OpenMP region in both F_{total} and F_{latoms} functions.

    - Removed OpenMP parallelization from F_{single}.
- V2.1
    - Replaced the naive lookup algorithm of a particle with a QMap container.

Modifying the lookup algorithm

Most of the time is spent in the computation of non-bonded interactions especially the van der Waals type, for which the intensity decreases quickly with the distance between the particles. To reduce the number of interactions to compute, it is possible to define Verlet lists which divide the system into smaller cells inside which the interactions are computed. The algorithm used to find particles was the naive $O(n^2)$ implementation. By using a hash table it was possible to reduce the time required by using an $O(n.\log(n))$ algorithm. This implementation explains the dramatic reduction of job duration with Chameleon V2.1.

Porting the code on HPC computers

One of the goals of this project was to port Chameleon on HPC architectures. The use of processors with more cores and memory allows the code to run larger systems. Since the code is written with OpenMP support, the use of a complete node is possible. We compared the performances of two compilers:

On Ada, the code was built with:

- GNU g++ 4.4.7
- Intel icpc 2017.1

On Ouessant, the code was built with:

- PGI pgc++ (16.10, 17.01 and 17.05)
- LLVM xlc++_r

Up to version 2.0 Chameleon depended on Qt for the management of arrays. A bug with the PGI compiler (versions prior to 17.05) prevented the code from compiling correctly. This bug was reported to the manufacturer and corrected in version 17.05.

Results

Test Cases

We tested the performance of Chameleon using three test cases having an increasing number of particles. The goal of varying the size of the system was to see if the behaviour of Chameleon is better when the size of the problem is larger. We have also specified the number of Monte-Carlo steps that were taken during the simulation. In all cases, we used oligomers of polyvinyl chloride.

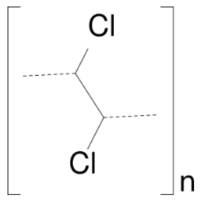


Figure 1. Polyvinyl Chloride

10xPVC100: small

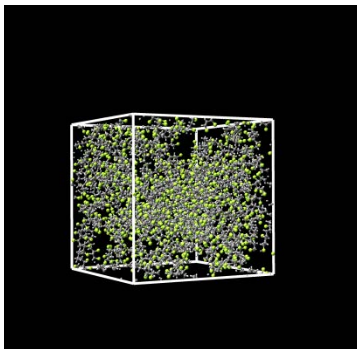


Figure 2. Simulation box of 10xPVC100.

- 10 chains of 100 PVC monomers
- 6,020 particles
- 10,000 Monte-Carlo steps.

6xPVC500: medium

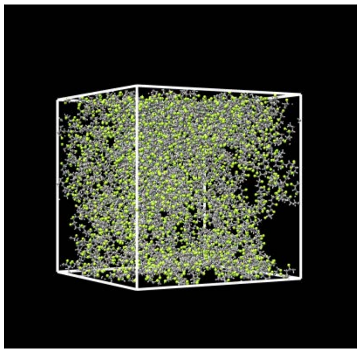


Figure 3. Simulation box of 6xPVC500.

- 6 chains of 500 PVC monomers
- 18,012 particles
- 1,000 Monte-Carlo steps

64xPVC200: large

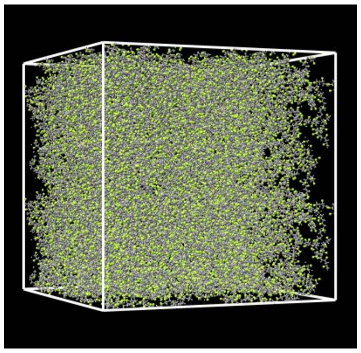


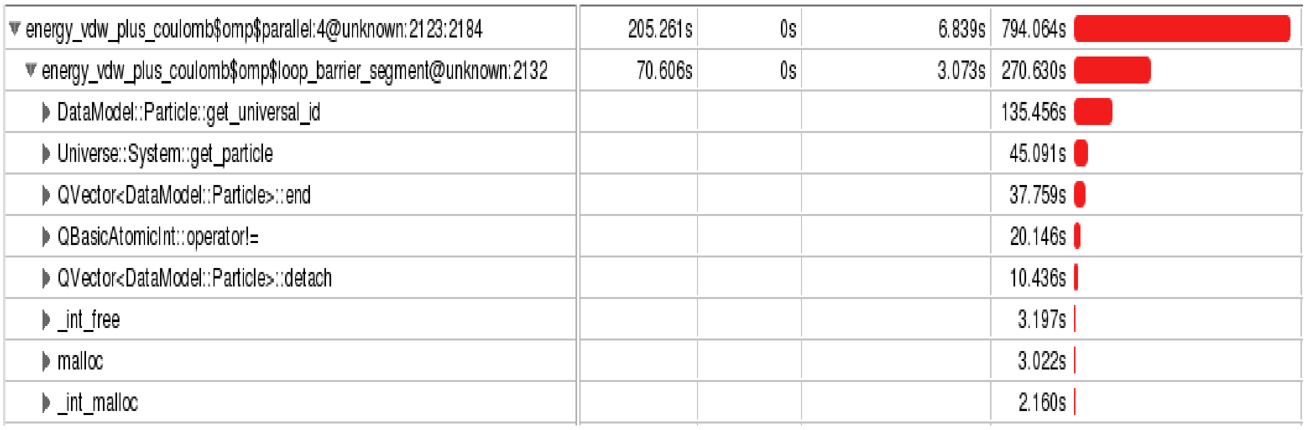
Figure 4. Simulation box of 64xPVC200.

- 64 chains of PVC monomers
- 76,928 particles
- 1000 Monte-Carlo steps

Performance analysis

To assess the performance of Chameleon we used Intel's Vtune and Advisor tools. With these we were able to identify several hotspots in the original code.

Figure 5 shows the output of Vtune after processing Chameleon on 4 threads. We can see that, as expected, the main part of the code is spent in the calculation of non-bonded interactions (energy_vdw_plus_coulomb). In this function, Chameleon is looking for particle IDs (get_particle and get_universal_id). The data structure holding this information is a Qt array and the lookup algorithm is the naive implementation that scales as O(n^2). By using a different data structure (standard map), it was possible to reduce the computing time of this part of the code.



▼ energy_vdw_plus_coulomb$omp$parallel:4@unknown:2123:2184	205.261s	0s	6.839s	794.064s
▼ energy_vdw_plus_coulomb$omp$loop_barrier_segment@unknown:2132	70.606s	0s	3.073s	270.630s
▶ DataModel::Particle::get_universal_id				135.456s
▶ Universe::System::get_particle				45.091s
▶ QVector<DataModel::Particle>::end				37.759s
▶ QBasicAtomicInt::operator!=				20.146s
▶ QVector<DataModel::Particle>::detach				10.436s
▶ _int_free				3.197s
▶ malloc				3.022s
▶ _int_malloc				2.160s

Figure 5 Output of VTune for the most CPU demanding part of V1.1

Ada

Compilers

On Ada it is possible to use GNU or Intel's compiler to build the code. Figure 6 shows the performances for both tools for the small test case (10xPVC100). As one can see, the code has a similar behaviour with both compilers. Nonetheless, the performance is better with Intel's C++ compiler and we decided to continue the tests with only icpc.

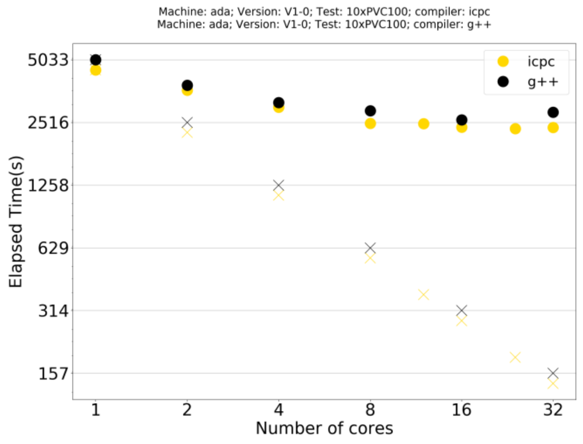


Figure 6. Comparison of GNU and Intel's compiler performances on Ada. The test case used is 10xPVC100. Both axes are base-2 log scales. Crosses indicate perfect scaling

Test case size

Figure 7 shows the scaling behaviour of the different versions of Chameleon used during the project. Versions 1.0 and 1.1 show a similar behaviour above 4 cores whereas V1.0 shows better performance below that threshold.

Version 2.0 introduced some changes in the OpenMP regions and the scalability was better. One has to note that the scalability degrades after 8 cores. It might be due to the NUMA nature of the machine.

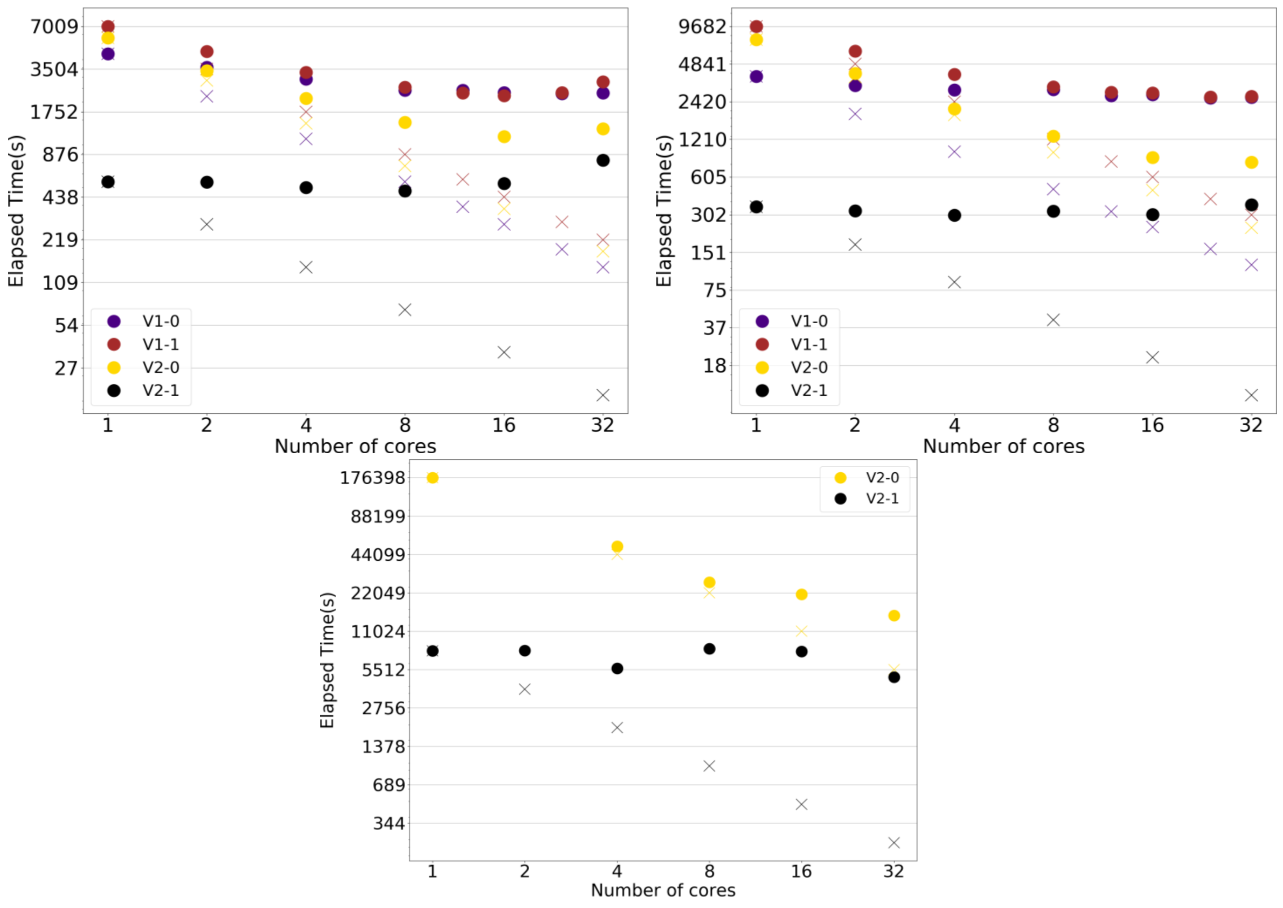


Figure 7. Comparison of the different versions of Chameleon. The small (upper left), medium (upper right) and large (bottom) tests cases were run on Ada. Crosses indicate the perfect scaling time. Both axes are base-2 log scales.

Ouessant

Compilers

On Ouessant we tested IBM XL and PGI compilers. As one can see on Figure 8 both compilers show similar general behaviour. We note that the performance is slightly better with the xlC compiler.

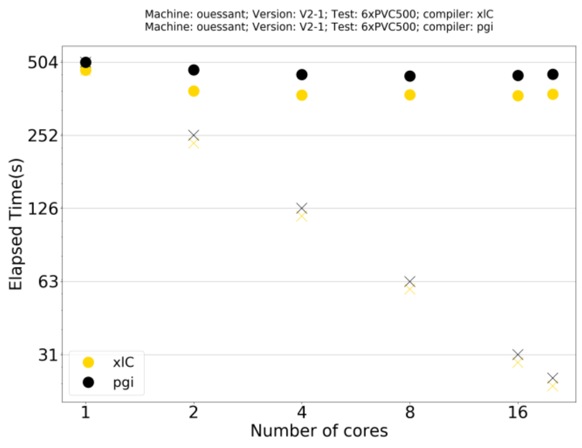


Figure 8. Comparison of xlC and PGI compilers for the version 2.1 of the code. The medium case is used. Both axes are base-2 log scales.

Results

Figure 9 show the scaling of the two latest versions of the code. As expected, Chameleon behaves the same way on Ouessant as on Ada. Version 2.1 outperforms version 2.0 but the parallel efficiency is lost.

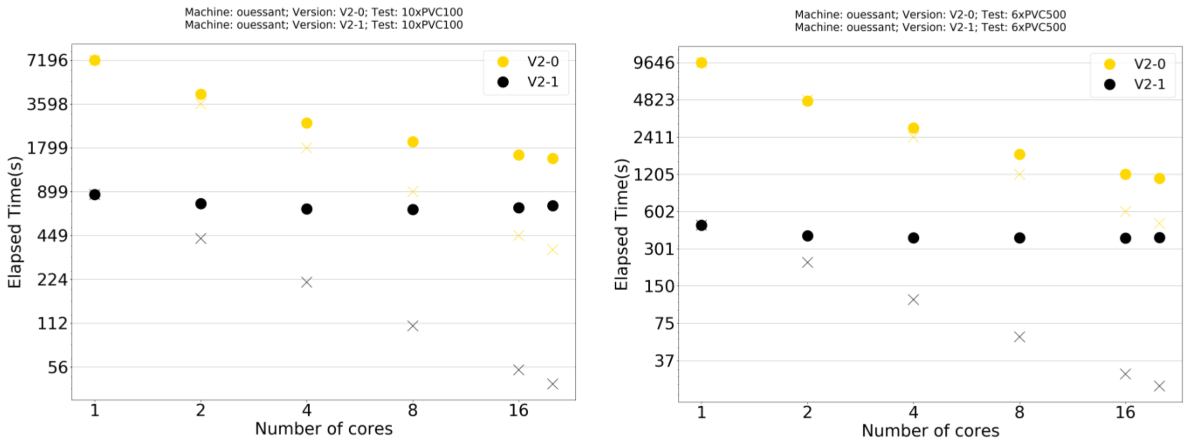


Figure 9 Comparison of the different versions of Chameleon using the xlC compiler. The small (left), medium (right) tests cases were run on Ouessant. Crosses indicate the perfect scaling time. Both axes are logscale in base 2.

Benefits for the SME

By accessing the computational resources provided within the project, Scienomics and IDRIS engineers were able to assess the performance of Chameleon in very extreme system cases. Porting Chameleon on HPC architectures and using an increased number of processors with more cores and memory, allowed us to execute Chameleon using very large realistic systems and at the same time assess the full spectrum of non-bonded interactions (Van der Waals and Coulomb energies). This gave us the necessary feedback to identify the optimum number of cores above which the scalability of the OpenMP parallelization in Chameleon degrades. In addition by applying powerful code profiling and analysis tools we were able to assess the performance of Chameleon algorithms for both sequential and parallel execution and to identify major bottlenecks and hot spots of the code. We managed to achieve more than 10x speed-up for the sequential execution of the code with some fluctuations depending on the system size (Figure 7 and Figure 9) even though some further efforts are needed to achieve improve parallel scalability.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EU’s Horizon 2020 Research and Innovation programme (2014-2020) under grant agreement 730913.