HAL
open science

Interactive Molecular Dynamics: Scaling up to Large Systems

Matthieu Dreher, Marc Piuzzi, Turki Ahmed, Chavent Matthieu, Marc Baaden, Nicolas Férey, Sébastien Limet, Bruno Raffin, Sophie Robert

To cite this version:

Matthieu Dreher, Marc Piuzzi, Turki Ahmed, Chavent Matthieu, Marc Baaden, et al.. Interactive Molecular Dynamics: Scaling up to Large Systems. International Conference on Computational Science, ICCS 2013, Jun 2013, Barcelone, Spain. hal-00809024

HAL Id: hal-00809024

https://inria.hal.science/hal-00809024v1

Submitted on 8 Apr 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2013) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2013

Interactive Molecular Dynamics: Scaling up to Large Systems

Matthieu Dreher[a], Marc Piuzzi[b], Ahmed Turki[c], Matthieu Chavent[b], Marc Baaden[b], Nicolas Férey[d], Sébastien Limet[c], Bruno Raffin[a,*], Sophie Robert[c]

[a]*INRIA, LIG, France*
[b]*Laboratoire de Biochimie Théorique, CNRS, UPR9080, Univ Paris Diderot, France*
[c]*Université d'Orléans, LIFO, France*
[d]*Université d'Orsay, LIMSI, France*

Abstract

Combining molecular dynamics simulations with user interaction would have various applications in both education and research. By enabling interactivity the scientist will be able to visualize the experiment in real time and drive the simulation to a desired state more easily. However, interacting with systems of interesting size requires significant computing resources due to the complexity of the simulation. In this paper, we propose an approach to combine a classical parallel molecular dynamics simulator, Gromacs, to a 3D virtual reality environment allowing to steer the simulation through external user forces applied with an haptic device to a selection of atoms. We specifically focused on minimizing the intrusion in the simulator code, on efficient parallel data extraction and filtering to transfer only the necessary data to the visualization environment, and on a controlled asynchronism between various components to improve interactivity. We managed to steer molecular systems of 1.7M atoms at about 25 Hz using 384 CPU cores. This framework allowed us to study a concrete scientific problem by testing one hypothesis of the transport of an iron complex from the exterior of the bacteria to the periplasmic space through the FepA membrane protein.

Keywords: Interactive Molecular Dynamics; Computational Steering with Haptic Arm; High Performance Interactive Computing

1. Introduction

The study of molecular assemblies provides important insights into their biological function and potential underlying mechanisms. To date, there are no experimental techniques simultaneously providing both atomic-scale spatial and pico- to microsecond temporal precision.

Molecular dynamics (MD) simulations are commonly used to study the behavior of molecular assemblies. Molecular assemblies span a broad range of sizes, typically from several hundred to hundreds of millions of atoms, depending on their complexity. MD simulation packages such as Gromacs [8] or NAMD [17] are complex codes, often parallelized to enable the simulation of large systems. Result analysis also requires dedicated graphical tools like VMD [9]. Visualization and analysis of the simulation are mainly carried out off-line, once the simulation has ended. In case a problem occurs or the system does not reach the desired state, the simulation must be restarted with a new set of parameters and/or constraints. As these simulations may run for several days, weeks or even

*Bruno Raffin. Bruno.Raffin@inria.fr

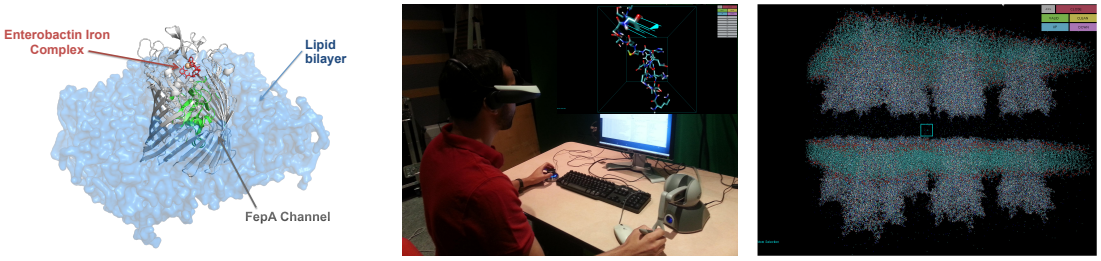


Fig. 1. Schematic representation of the FepA iron transporter protein (gray) along with the lipid bilayer (blue) and the iron-enterobactin complex (red). The complex is known to pass through the channel but a plug (green) blocks the way in this static form (left). Interaction with a small peptide using an haptic device, a space navigator and a stereoscopic viewer (center). Interactive simulation of a large membrane protein system composed of 1.7 million atoms (right). The solvent atoms are filtered out of the image before rendering.

months on hundreds of processors, an erroneous combination of parameters may lead to the waste of numerous compute cycles. Current simulation techniques may fail to capture some biological processes because of the large amount of time needed for important conformational changes to take place. A common approach to accelerate the simulation consists in imposing external constraints to bias the simulation towards phenomena of interest. But, as these constraints are defined at starting-time, one simulation is required per constraint test set, which can be very time-consuming, requiring many trial-and-error cycles.

An alternative approach to solve these issues consists in coupling the simulation code with a set of input devices and a 3D visualization engine to enable interactive analysis and steering of the simulation behavior [20, 21]. These systems, called Interactive Molecular Dynamics (IMD), allow the user to interact directly with the simulation by driving the experiment to a particular structural or thermodynamical state, while validating the whole process by live analysis. This direct dynamics 3D visualization helps comprehending structural changes. However, IMD use remains typically limited to demonstration and educational purposes. One of the possible reasons is the difficulty to scale to molecular systems large enough to match current simulations running in batch mode, while maintaining performance enabling interactivity and spanning biologically relevant time-scales.

In this paper we introduce the Fvnano numerical workbench. Fvnano focuses on pushing the limits of IMD to steer large systems. Fvnano is a data-flow oriented framework to harness a parallel MD simulator and augment it with interaction capabilities. Here we use the Gromacs simulation code [8]. The system is designed to limit intrusion in the simulation while getting close enough to the source of data production to minimize the overhead when extracting relevant data. These data are then forwarded asynchronously and processed up to the 3D visualization environment. An haptic device enables the user to select and apply constraints to a group of atoms.

Experiments with various system sizes and compute nodes show an interactive performance can be achieved for systems up to 1.7 million atoms running on 384 CPU cores. We also used the framework to tackle a first realistic problem: testing alternative hypotheses on the transport of an iron-siderophore complex through the lipid bilayer of a bacterial cell.

After analysing related work (Section 2), we present the framework (Section 3) and some of its algorithmic and system details. Experiments for testing the system scalability (Section 4) and a real use case (Section 5) precede the conclusion.

2. Related Work

For a long time, structural biologists have been using MD-type simulations to study the dynamics of biological complexes. Through simulation libraries, they pushed the system sizes that can be tackled, improving parallelization strategies and scalability. NAMD [17] and Gromacs [8] are two very common packages. Today these packages are also evolving to take advantage of new architectures, such as multi-core CPUs and GPUs.

Steered molecular dynamics (SMD) is a direct precursor to the development of IMDs, derived from regular MDs by applying external forces to an object (atom, residue, protein, etc ...) to probe its mechanical function as

well as to accelerate a process that may be too slow to model otherwise. Such a system was proposed by [11]. External forces are defined a priori, and applied at run-time but without the possibility for the user to change them.

The introduction of the user in the analysis loop has been highlighted in [23]. SCIRun [16] proposes a generic system to construct large applications that can be steerable. Relying on the dataflow paradigm, the programmer can link the simulation to various modules including visualization to obtain a continuous feedback on the simulation and change its parameters when necessary. However, the system requires a global space memory; therefore it can not be deployed on clusters. More recent work has been carried out to mitigate the intrusion in the simulation code and create clear separations between the simulation and the visualization. [10] proposes an online visualization for artery studies. Each process of the parallel simulation also runs a data manager in charge of writing result files. The visualization module requests data from listeners that read back these files and pre-process the data. This approach limits the impact on the simulation but in turn the interactivity is limited due to the high latency introduced by accessing the intermediate files. [15] proposes a tighter coupling. The data are gathered on a master node by the simulation and subsequently forwarded to the visualization node. The system achieves high frequencies and a good interactivity, but incurs a significant overhead compared to the original simulation (50% performance drop).

Visualization of molecular assemblies classically relies on a variety of 3D representations, many of them available in common visualization tools like VMD [9] or Pymol [19]. Rendering large 3D models at interactive frame rates is still challenging though. Issues include performance as well as representations. Grottel et at. [7] propose various levels of optimizations (culling, caching, ...) to interactively visualize time varying molecular datasets containing tens of millions of atoms. The quicksurf algorithm sinks the molecular system in a regular 3D grid, enabling to extract an iso-surface, an approach that is well suited to large assemblies [13].

The interest of haptic rendering to study molecular interactions has already been highlighted during the Grope project [4], many years ago. In particular, the haptic feedback eases molecule manipulation and perception [3]. IMD with haptics has been further developed in [20, 21]. The framework is based on VMD for the visualization and NAMD for the molecular dynamics engine. Interaction is handled through a VRPN server, supporting 6 degrees of freedom input devices and haptic feedback. Initially focused on steering a simulation running on a parallel machine, the authors also recently experimented steering an MD simulated on a GPU. The advantage is to be able to simulate a reasonably complex system on a single GPU.

To further improve the user experience, visualization and steering of a scientific application can take place in an immersive environment. Renambot et al. [18] steered from a CAVE a molecular dynamics application running on the DAS parallel computer. Koutek et al. proposed a system to steer particles using a virtual workbench called MolDRIVE [12]. MD simulations were launched on SGI or Cray supercomputers but, because of communication issues, the best results were achieved using only 8 nodes thus limiting the size of the studied system. Using VRDD [2] in a CAVE, users could experiment protein docking in an immersive way using both visual and auditory feedback.

To our knowledge, the largest published molecular systems simulated interactively reach a few tens of thousands of atoms so far. Our goal is here to significantly push this limit to a few million atoms.

3. The Fvnano Workbench

3.1. Architecture Overview

The FvNano application is composed of four main components: visualization, graphical user interface, device manager and simulation (Fig. 2(a)). The main pipeline steps are:

- The simulation outputs its current state toward the visualization.
- The visualization produces a graphical representation of the on-going simulation, the user being able to control asynchronously his viewpoint.
- The user can select and apply forces to a set of atoms, using a combination of devices.
- The simulation updates its state taking into account user controlled parameters, such as the external forces to be applied on a selection of atoms.

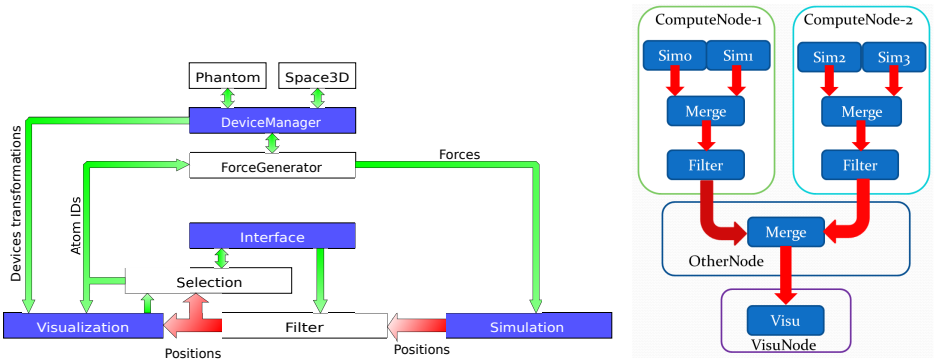


Fig. 2. The FvNano network is composed of four main components (blue boxes) with a few secondary modules (white boxes). The red arrows represent the main bandwidth consuming data-flows issued from the simulation. The green arrows represent light communications between modules.

Secondary components include a *selection* module to select a subset of atoms from the visualization by clicking on atoms or through the user interface. This selection is mainly used to apply forces on the selected atoms. The *forcegenerator* defines the forces to be applied to the selected atoms, these forces being subsequently forwarded to the simulation, the visualization and the force feedback device. This framework targets clusters of hundreds of cores providing the necessary computing resources to simulate large systems interactively.

Performance is a key issue. Considering that the parallel simulation code is already highly optimized, our concern is to efficiently extract the data produced by the simulation to filter and forward them to the visualization. The simulation engine usually supports some mechanisms to save intermediate results in files. One approach consists in reading these files or grabbing the data on the master node that usually centralizes the data for building the file. We experimented these schemes with Gromacs, but the performance was low. The main reason is that while all the data are centralized on the master node, the simulation is suspended. We obtained the best performance relying on a different approach. The data are extracted directly from each Gromacs process. Their aggregation and filtering is performed in parallel and asynchronously, while the simulation runs.

Modularity was another concern. By relying on a data-flow and component oriented approach, we enforce the ability to change components with a limited impact on the remaining elements of the application. In this paper we rely on the Gromacs [8] MD simulation engine, but following a similar approach we could develop a component for other tools like NAMD [17] for instance.

3.2. Programming Environnement and Run-time System

Our implementation relies on the FlowVR framework [1]. FlowVR allows to describe an application as a dataflow graph where nodes (components) are operations on data and edges are communication channels. Here, we describe the main features relevant for our work. Refer to [1, 14] for more details.

A component (also called module) encapsulates an iterative code that can communicate with other components using input and output ports. When each input port has received a message, the module starts a new iteration. The data are processed and possibly new data are sent through the output ports. The strong point of this system is that the developer of a module does not have to know where the data comes from and where he has to send the produced data.

A communication channel is a link from one module output port to another module input port. By default, this channel is FIFO but special components can be installed along a data path to elaborate more complex communication policies (broadcast, gather, scatter, resampling, etc.). Such components will be described later.

This architecture allows a high level of modularity. Each component can be replaced independently without having to change anything in the code of other modules or in the overall application network.

The API to program a module is simple to limit the code refactoring necessary to turn an existing code into a module. Basically, a module is based on three main operations:

- *wait()*: block the execution of a module until each input port is filled with a message.

- *get()*: grab a message from an input port.
- *put()*: put a message on an output port (the actual data transmission is asynchronous)

A FlowVR application can be executed on a cluster by distributing the different modules on different nodes. At runtime, one daemon runs on each machine node involved in the application. Each daemon is responsible for the modules hosted on its machine and for the associated communication channels. Modules never communicate with each other directly. To create and send a message, a module allocates memory into a shared memory space provided by the daemon and writes directly into that space. Through the *put()* instruction, a module posts a message to its daemon. If the destination module is located on the same node, the daemon simply passes a pointer on the message to the destination module, avoiding unnecessary copies. Otherwise, the daemon sends the message through the network to the daemon hosting the target module. The daemon stores the message in its local shared memory segment and forwards a pointer to the destination module.

To implement communication patterns between modules, we use specific FlowVR components called *filters*. These components are hosted directly by the daemons and are dedicated to light operations on messages. FlowVR comes with several *filters*, but the user can also develop custom filters. *MergeFilter* is an example of such a filter. This filter takes several input messages and produces one output message by concatenation of the input messages. We also used filters to bound the frequency of some modules or to extract the newest message waiting in a FIFO communication channel (resampling pattern), the older messages being discarded, for an improved latency.

Embedding a MPI parallel code into a FlowVR network is straight-forward. For each MPI process, a module is created encapsulating this process (done at the initialization). Then, each individual process can communicate using MPI and through its FlowVR ports.

3.3. Gromacs Integration

For the simulation module, we rely on Gromacs[8] to provide a molecular dynamics engine. Gromacs runs on desktop machines using a thread based parallelism or in a distributed context using MPI. A heterogeneous method using both MPI and OpenMP should be available in a future version of Gromacs but it is not yet released. In the remainder of this article, we assume that Gromacs only uses MPI.

3.3.1. Gromacs Internal Organization

Gromacs follows a master/slave organization. The first MPI process (the master) loads the simulation, configures it and propagates all necessary information to the slaves. During the simulation, only the master can access the global state of the simulation. The Algorithm-1 pseudo-code describes a usual Gromacs iteration.

```
while step < nstep do
    dd_partition_system();
    do_force() ;
    write_traj() ;
    update() ;
    step++ ;
end
```

Algorithm 1: Gromacs iteration loop

```
while step < nstep && wait() do
    dd_partition_system() ;
    do_force() ;
    user_forces ← get(forces_port) ;
    if user_forces ∈ home_atoms then
    end
    if traj_enable then
        write_traj()
    end
    position_message ← home_atoms ;
    put(position_message) ;
    update() ;
    step++ ;
end
```

Algorithm 2: Modified Gromacs iteration loop

Each process is responsible only for the atoms contained inside its domain (referred to as the *home atoms*). Several communication steps take place between processes, to dynamically rebalance the work load as the atoms move in the simulation domain ($dd_partition_system()$), or to compute the forces that apply to each atom ($do_force()$).

To write the trajectory and other backup files ($write_traj()$), a global state of the molecular system is gathered on the master node. This operation involves costly communications. To mitigate this cost, a common tradeoff is to write the state of the simulation only every x steps where x is supposed to be small enough to guarantee that the scientist will not miss anything important. Finally, the molecular system is updated and the atom states (positions, energies,etc) are modified ($update()$). Refer to [8] for details.

3.3.2. External Force Integration

The forces are produced by the *forcegenerator* module. A force is represented as a vector describing the intensity and direction of the force and an id indicating the atom the force applies to.

To add external forces to the simulation, we broadcast the array of forces to all the MPI processes through a FlowVR broadcast communication. The memory footprint is H times the size of the forces with H the number of different nodes (often smaller than the number of processes as several processes run per node). No copies are required inside the simulation because Gromacs gets a pointer to the message in shared memory. Then each process checks for each force whether it applies to a home atom. In this case the process adds the forces to the Gromacs force array.

We expect the user to apply forces on a few atoms and not on the full molecule. The size of the force message is thus small, making a full broadcast the best method. We could use a table linking the atom IDs and the hosts to route the atoms directly to the correct host. However, the cost of maintaining such a table does not compensate for the bandwidth gain if the force messages are too small.

3.3.3. Extracting Atom Positions from Gromacs

Each Gromacs process handles a set of home atoms. To extract these positions, Gromacs code was slightly modified to copy these atom positions into a FlowVR message. Gromacs proceeds on the current iteration as soon as the related *put()* call returns (Alg. 2). Then, the local FlowVR daemon asynchronously takes cares of the built message as detailled in the next section. In total, about 50 code lines have been added to Gromacs, mainly to initialize the FlowVR context and process messages. The only Gromacs function that has been modified is the saving of trajectories to files. We preserved the possibility to backup trajectories. However, to achieve the maximum performance, it is preferable to diminish or even disable the related communications and file writes, which can create some lags at run time and degrade the interactivity quality.

3.4. Processing Messages of Atom Positions

On each node, the local FlowVR daemon gets one handle per position message provided by each Gromacs process running on that node. These messages need to be gathered and forwarded to the visualization module. We adopt a three levels merge pattern that makes it possible to parallelize the merging, limit the amount of message copies and the number of messages transiting on the network. On each node the messages are gathered, sent to other nodes which don't host the simulation by bunch of ten, gathered and finally sent to the node running the visualization (See Fig. 2(b)). Received messages are merged and transfered to the visualization. This pattern is defined through a tree pattern where nodes are *Merge* filters and edges are communication channels for position messages. This pattern can flexibly be tuned to adapt to the target architecture (tree arity for instance).

The user may only be interested by some specific atoms. For that purpose we insert selection filters after the first merge operation. These filters can discard atoms based on their atom type or their residue. For large models, a common request from users is to filter the solvent, which represents a large number of atoms and may not be necessary to visualize. By removing these atoms - which commonly represent up to 80 percent of the total number of atoms - early in the process, we can save a lot of resources. By filtering out unused data as soon as possible, we reduce the amount of data transiting on the network and lighten the next operations along the pipeline (data conversion and visualization).

3.5. Rendering

Rendering performance is critical as we target an interactive rendering frame rate of large molecular systems. We rely on the Hyperball algorithm [5], an improved ball-and-stick representation replacing cylinders linking the atom spheres by hyperboloids. Changing the hyperboloid parameters enables to sweep through different

representations and in particular the classical ball-and-stick, licorice and space-filling van der Waals spheres. The implementation is optimized to take advantage of GPU capabilities and support large molecular systems. Refer to [5] for details. We also support VMD [9] for visualization. All interactive experiments presented here rely on the Hyperball algorithm.

3.6. Interaction

Our framework supports several commodity devices for interactions. The *devicemanager* module gets device events through VRPN [22], converts them to a standard representation and forwards them through FlowVR channels.

Our final implementation uses two devices : a SpaceNavigator 3D mouse to control the camera of the visualization and a Phantom haptic arm to select atoms and apply forces. The SpaceNavigator allows to translate and rotate the model. The Phantom allows to select atoms, apply forces and feel a force feedback. To help the user in this task, the visualization provides highlights of the atom that is selectable by the Phantom and a simple representation of the forces currently applied.

To help the user make the intended atom selection, we add a specific filter to control the speed of the simulation component. At more than 80 iterations per second, atoms are moving and vibrating quickly, making atom selection difficult without this filter. By changing the frequency parameter of this filter online through a GUI, the user can slow down the simulation for a comfortable atom selection (See the attached video [1].)

This filter broadcasts a short message to each Gromacs process at a given rate. The *wait()* call introduced in each process locks it as long as the signal is not received, constraining Gromacs to the maximum target frequency.

When the selection mode is activated, the *selection* module identifies the closest atoms to the Phantom and forwards this list to the visualization and *ForceGenerator* modules. Because the user selects atoms based on visual clues, the atoms the visualization highlights as selectable need to match the one the *selection* module computed. Otherwise, the atoms visually selected may be different from the actual selection. To avoid this pitfall, the visualization and *selection* modules are tightly coupled to run at the same speed. The visualization can not render the next frame unless the *selection* sends the selectable atom for the current position of the phantom.

4. Experimental Results

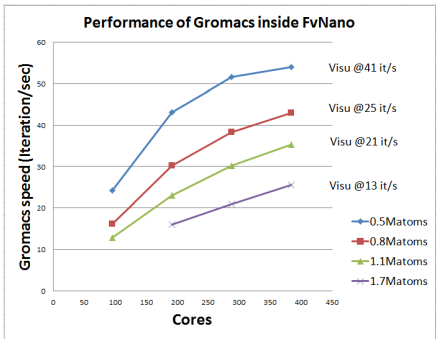
The hardware used for the visualization is a Xeon E5530 quad core CPU at 2.40GHz, a GeForce 680 GTX graphics card. The computational cluster used is composed of 72 nodes, each node running 2 Xeon E5520 quad core CPU at 2.27GHz, 24GB RAM. Communications between the computational cluster nodes go through a QDR Infiniband fabric (40Gbits/s) while we only have a DDR (20 Gbits/s) Infiniband interconnection between the computational cluster and the visualization node.

We launch 6 Gromacs processes per compute node (8 cores available). On each of these nodes runs one *merge* filter to gather the atom positions from the local Gromacs processes and one filter module (see Figure-2(b)). Every 10 nodes running Gromacs processes, an extra node is added to run one *merge* filter gathering the atoms positions from these 10 nodes. The remaining modules and filters, as well as the interaction devices are all executed on the visualization node.

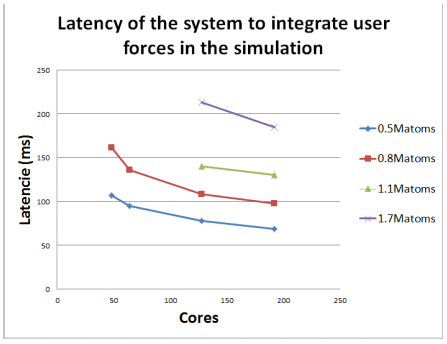
4.1. Interactivity Evaluation on Large Molecular Models

We performed performance tests with models ranging from 500K atoms to 1.7M atoms (Fig. 3(c)). These models were built by replicating a base assembly composed of a Glic membrane protein along with a lipid bilayer and solvent. The 1.7M atoms system is composed of 12 juxtaposed Glic systems. The systems have been set up to use virtual sites and the following simulation parameters : AmberGS forcefield, 5 fs steps, Fast smooth Particle-Mesh Ewald (SPME) electrostatics and 1 nm cut-off. The performance is measured through 3 different metrics: simulation and visualization iterations per second, and the latency time it takes from the emission of forces by the *forcegenerator* module up to the receipt by the *visualization* module of the positions influenced by the emitted forces (Fig. 2(b)). As both modules run on the same machine, the measured latency is not affected by clock synchronization issues.

[1] http://moais.imag.fr/membres/matthieu.dreher/FvNanoICCS1080p.mp4



(c) Simulation and visualization frequencies



(d) Latency

Fig. 3. Simulation (iterations per second), visualization (frames per second) frequencies and latency measures for various molecular assemblies, Gromacs running on an increasing number of cores (from 48 up to 384).

The viewpoint is set to fit the whole molecule within the window (resolution 800x600). The visualization frequency can vary significantly depending on atom visibility, but even for the largest assembly the user can smoothly navigate through the molecular system.

Increasing the number of cores enables to improve performance significantly for all model sizes (not always linearly though). As detailled below this is consistent with the performance obtained when Gromacs runs standalone. Latency benefits not quite as much from additional cores. For small models ($<$ 100K atoms), the latency is mainly dominated by the simulation time. For a system of 70K atoms running at 60 its on 64 cores, 17ms of the 24ms measured latency are due to the simulation time. For large models, even if the simulation times decrease significantly when involving more processors, the volume of data transmitted to the visualization is important and stays unchanged, dominating the latency time. For instance the 1.7M atoms assembly runs on 192 cores at 16 its, where 61ms out of 185ms measured latency are due to the simulation time.

The user experience is generally satisfactory from a latency and frequency point of view for mid size assemblies. Above 1M atoms, some spikes might occur during the user experiments causing discomfort for the user. Another limitation is related to the visual representation. Every individual atom of the scene is represented, cluttering the screen and making the selection of specific atoms difficult. We will experiment with alternative representations like surface-based ones to evaluate if they enable the user to better navigate through large molecular assemblies.

4.2. Cost of Interactivity

To evaluate the cost of interactivity, we compared the performance of running Gromacs standalone or in the Fvnano framework (Table-1). Gromacs performance is affected by the extra computations performed on each node at FlowVR level as well as the extra communications the network needs to perform. The overhead is about 17.74% of Gromacs standalone performance when running on 384 cores.

number of cores	96	192	288	384
Gromacs standalone	13.67	25.02	35.71	42.90
Gromacs & Fvnano w/o com. (% overhead)	13.29 (2.77%)	24.67 (1.32%)	33.80 (5.35%)	41.05 (4.30%)
Gromacs & Fvnano (% overhead)	12.76 (6.62%)	23.00 (8.07%)	30.16 (15.55%)	35.29 (17.74%)

Table 1. Gromacs performance (iterations per second) when running standalone (first row), when coupled with FlowVR extracting atom positions, performing a per node merging but without sending out built messages (second row), when coupled with FlowVR and all Fvnano features enabled but (third row). Notice that for this last case all atom positions are sent on the network (no atom is filtered out). Molecular assembly of 1.1 million atoms simulated on various number of cores, using 6 cores per node for Gromacs processes.

According to the second row, the local treatment of the atom positions cause little arm to the simulation performance (5% in worst case). Compare to Gromacs standalone, 3 extra processes run on each simulation node : the flowvr daemon, an atom filter and a merge filter. Even if we expect these processes to run on separate cores

than Gromacs ones, they can create contention during system calls or memory transfers. Moreover, each Gromacs process needs to handshake with the local daemon during the *wait()* operation, causing some delays on Gromacs execution. The resulting overhead is however very limited. Actually sending the messages on the network has a more import impact on Gromacs (row 3). These communications comes in addition to native Gromacs message exchanges and increase the load on the network cards and switches, causing most likely this performance drop. But a 17% performance drop stays a reasonable price to pay for interactivity.

5. IMD Real Use case

Membrane proteins are essential to the transport of molecules through the lipid bilayer of the cell. Transporters may work in very different ways and, in some cases, the transport mechanism is yet to be discovered. This is the case with the FepA protein, which uptakes an iron atom complexed with a siderophore, a bacterial molecule called enterobactin, and transports it through the lipid bilayer. The structure of the FepA protein has been solved by X-Ray crystallography and shows a barrel shape formed by β strands with a globule folding inside it (Figure-1(a)). This particular folded globule forms a plug which is known to block the passage of the iron complex in this static form. To date, two hypotheses exist for the transport mechanism : the plug (i) undergoes structural rearrangement to open a passage, (ii) is dislodged towards the periplasm (outside the channel). In a recent work [6], two water channels inside the protein have been identified. They may be precursors for transport, which would tend to confirm the first hypothesis. It was observed that these channels are wide enough to let water molecules pass through the protein, thanks to a small structural rearrangement of the plug. However, the iron-enterobactin complex is bigger than those channels and we wanted to determine if the structural rearrangement may be large enough for the entire complex to pass through. As SMDs would be difficult to use in this particular case (the supposed trajectories do not follow a straight line) we decided to use the FvNano IMD framework. The goal was to lead the iron-enterobactin complex through the channel, based on the hypotheses formulated in the earlier study, using the haptic device and the visual feedback.

The simulation parameters are strictly identical to those used for regular simulations. The total system size is about 72K atoms and contains the FepA protein, a small portion of a lipid bilayer, the iron-enterobactin complex and the solvent. This simulation was executed on a computational cluster using 128 cores and a separate node for visualization and interaction. The IMD allowed us to confirm two distinct paths along which the plug only undergoes light structural changes (mainly on flexible loops) when pulling the complex through. To ensure the reproducibility of the results, the simulation was run several times and each time equivalent trajectories were found. These trajectories are compatible with the two water channels that have been previously identified. In this experiment, it is important to track the amount of force applied to the system, as a high level of force would limit the biological meaning. In this case, the mean value of the forces applied to each atom of the iron-enterobactin complex was about 3735 pN per step (2 fs) during the simulations. This value is quite high in regards to SMD simulations. However, this is the first step of the search process as these simulations allowed us to identify potential trajectories for the complex which will be further studied. Decreasing the forces applied per step requires more iterations, i.e. more time, to follow a similar trajectory, but increasing the overall simulation time impairs interactivity.

This experiment showed us that it is possible to drive large simulations in an interactive session. However, for a 15 min interactive session, only nano second reactions are observable interactivly. This is a major obstacle as a large number of chemical reactions take place in micro seconds or more. In the case of the FepA, we chose to use important forces allowing us to speed up the reaction process at the price of realism. Using more CPU should help us to run longer simulation but the micro second simulation seems unreachable in interactive time. We are currently looking at intermediate solutions between IMD and SMD. The goal is to provide the user a way to describe a trajectory for selected atoms, run the simulation for several days and, from time to time, connect the interactive tools to the simulation to check the state of the simulation and adjust the trajectory if necessary.

6. Conclusion

We presented an effort to enable interactive, haptic based, steering of large molecular dynamics simulations. Our main contribution is the efficient integration of a parallel Gromacs simulation within a virtual reality frame-

work associating haptics and high performance 3D rendering. This framework enables us to simulate molecular systems of 1.7M atoms at 25 iterations per second relying on 384 computing cores. This framework was used to explore an open biology problem, enabling us to find two plausible paths for the transport of an iron-siderophore complex through a FepA channel. This is one of the few examples of a virtual reality application that directly contributes to solving an open problem in structural biology. However, the FepA experiment remains a mid-size molecular system compared to the capabilities of our framework. Future work will explore its effectiveness for tackling molecular systems at an even larger scale.

Acknowledgments

This work has been partly funded by the ANR, project FVNANO ANR-07-CIS7-003 and EXAVIS ANR-11-MONU-003. Experiments were performed on the Grid'5000 experimental testbed (https://www.grid5000.fr).

References

[1] J. Allard, J.-D. Lesage, and B. Raffin. Modularity for Large Virtual Reality Applications. *Presence: Teleoperators and Virtual Environments*, 19(2):142–162, April 2010.

[2] A. Anderson and Z. Weng. VRDD: applying virtual reality visualization to protein docking and design. *Journal of Molecular Graphics and Modelling*, 17(34):180 – 186, 1999.

[3] P. Bivall Persson, L. Tibell, M. Cooper, A. Ynnerman, and B.-H. Jonsson. Evaluating the Effectiveness of Haptic Visualization in Biomolecular Education - Feeling Molecular Specificity in a Docking Task. In *12th IOSTE Symposium*, pages 745–752. Universiti Science Malaysia, 2006.

[4] F. P. Brooks, Jr., M. Ouh-Young, J. J. Batter, and P. Jerome Kilpatrick. Project GROPEHaptic displays for scientific visualization. In *Proceedings of Siggraph '90*, pages 177–185, New York, NY, USA, 1990. ACM.

[5] M. Chavent, A. Vanel, A. Tek, B. Levy, S. Robert, B. Raffin, and M. Baaden. GPU-accelerated atom and dynamic bond visualization using hyperballs: A unified algorithm for balls, sticks, and hyperboloids. *Journal of Computational Chemistry*, 32(13):2924–2935, 2011.

[6] J. Esque, M. Sansom, M. Baaden, and C. Oguey. A case study of protein topology: how enterobactin modifies the water network through fepA. *submitted to PLoS One*.

[7] S. Grottel, G. Reina, C. Dachsbacher, and T. Ertl. Coherent Culling and Shading for Large Molecular Dynamics Visualization. *Computer Graphics Forum*, 29(3):953–962, 2010.

[8] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *Journal of Chemical Theory and Computation*, 4(3):435–447, 2008.

[9] W. Humphrey, A. Dalke, and K. Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

[10] J. Insley, M. Papka, S. Dong, G. Karniadakis, and N. Karonis. Runtime Visualization of the Human Arterial Tree. *Visualization and Computer Graphics, IEEE Transactions on*, 13(4):810 –821, july-aug. 2007.

[11] S. Izrailev, S. Stepaniants, B. Isralewitz, D. Kosztin, H. Lu, F. Molnar, W. Wriggers, and K. Schulten. Steered Molecular Dynamics. In *Computational Molecular Dynamics: Challenges, Methods, Ideas*, pages 39–65. Springer-Verlag, 1998.

[12] M. Koutek, J. van Hees, F. H. Post, and A. F. Bakker. Virtual spring manipulators for particle steering in molecular dynamics on the responsive workbench. In *Proceedings of EGVE '02*, pages 53–ff, Aire-la-Ville, Switzerland, 2002. Eurographics Association.

[13] M. Krone, J. E. Stone, T. Ertl, and K. Schulten. Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories. In *EuroVis 2012 Short Papers*, volume 1, 2012.

[14] J.-D. Lesage and B. Raffin. A Hierarchical Component Model for Large Parallel Interactive Applications. *Journal of Supercomputing*, 60:389–409, June 2012.

[15] M. D. Mazzeo, S. Manos, and P. V. Coveney. In situ ray tracing and computational steering for interactive blood flow simulation. *Computer Physics Communications*, 181(2):355 – 370, 2010.

[16] S. Parker, D. Weinstein, and C. Johnson. The SCIRun Computational Steering Software system. In E. Arge, A. Bruaset, and H. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 1–40. Birkhauser Press, Boston, 1997.

[17] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kal, and K. Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005.

[18] L. Renambot, H. E. Bal, D. Germans, and H. J. W. Spoelder. CAVEStudy: An Infrastructure for Computational Steering and Measuring in Virtual Reality Environments. *Cluster Computing*, 4(1):79–87, Mar. 2001.

[19] Schrödinger, LLC. The PyMOL Molecular Graphics System, Version 1.3r1. August 2010.

[20] J. E. Stone, J. Gullingsrud, and K. Schulten. A system for interactive molecular dynamics simulation. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, I3D '01, pages 191–194, New York, NY, USA, 2001. ACM.

[21] J. E. Stone, A. Kohlmeyer, K. L. Vandivort, and K. Schulten. Immersive molecular visualization and interactive modeling with commodity hardware. In *Proceedings ISVC'10*, pages 382–393, Berlin, Heidelberg, 2010. Springer-Verlag.

[22] R. M. Taylor, II, R. M. T. Ii, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser. VRPN: A Device-Independent, Network-Transparent VR Peripheral System, 2001.

[23] R. van Liere, J. D. Mulder, and J. J. V. Wijk. Computational Steering, 1996.