



Partnership for Advanced Computing in Europe

Data I/O Optimization in GROMACS Using the Global Arrays Toolkit

Valentin Pavlov*, Peicho Petkov

NCSA, Acad. G. Bonchev str., bl. 25A, Sofia 1000, Bulgaria

Abstract

In MPI multiprocessing environments, data I/O in the GROMACS molecular dynamics package is handled by the master node. All input data is read by the master node, then scattered to the computing nodes, and on each step gathered back in full and possibly written out. This method is fine for most of the architectures that use shared memory or where the amount of RAM on the master node can be extended (as in clusters and Cray machines), but introduces a bottleneck for distributed memory systems with hard memory limits like the IBM Blue Gene/P. The effect is that even though a Tier-0 Blue Gene/P machine has enough overall computing power and RAM, it cannot process molecular systems with more than 5,000,000 atoms because the master node simply does not have enough RAM to hold all necessary input data. In this paper we describe an approach that eliminates this bottleneck and allows such large systems to be processed on Tier-0 Blue Gene/Ps. The approach is based on using global memory structures which are distributed among all computing nodes. We utilize the Global Arrays Toolkit by PNNL to achieve this goal. We analyze which structures need to be changed, design an interface for virtual arrays and rewrite all routines which deal with data I/O of the corresponding structures. Our results indicate that the approach works and we present the simulation of a large bio-molecular system (lignocellulose) on an IBM Blue Gene/P machine.

1. Introduction

GROMACS is a generic molecular dynamics package which – according to its authors – is “designed to simulate the Newtonian equations of motion for systems with hundreds to millions of particles” [1]. However, due to the way the input and output data is handled, there is a practical upper limit for the size of the simulated system on distributed memory clusters and supercomputing architectures, namely the IBM Blue Gene/P. Even the largest Tier-0 machine in PRACE – JUGENE [2] – which has a total processing power of 1 Petaflop/s and 144 TB RAM cannot simulate a system with more than a couple of million particles due to this bottleneck.

The goal of the project described in this paper is to overcome the memory-related limitations introduced by the input/output data handling and thus **allow GROMACS to perform Peta-scale simulations on distributed-memory supercomputing systems**. Specifically, the technical goal is to be able to successfully run a dynamics **simulation of a system consisting of at least 5,000,000 atoms on a 128-node IBM Blue Gene/P partition**. If this is made possible, then by utilizing larger partitions it would certainly be able to simulate even larger systems.

The rest of this paper is organized as follows. Section 2 outlines the problem at hand, starting with some general information about the IBM Blue Gene/P supercomputing architecture and the GROMACS software package that is relevant to the discussions that follow. Section 3 describes the approach for solving the problem. Section 4 summarizes the results and outlines further work.

* Corresponding author. *E-mail address:* vpavlov@rila.bg

2. Problem Analysis

2.1. IBM Blue Gene/P organization

The IBM Blue Gene/P supercomputer [3] is a hybrid architecture multiprocessing system. It consists of a large number of independent computing nodes interconnected with several networks. Each computing node consists of 4 PowerPC 450 CPUs working at 850 MHz and 2 GB of dedicated physical memory in which applications can run. Each core is capable of performing 4 floating point operations per cycle, thus the total processing power of a computing node amounts to 13.6 GFLOP/s. 32 computing nodes are placed on a node card, and 32 node cards fit into a rack. The system is modular and can be composed of as many as 72 racks, for a total of 1 PFLOP/s and 144 TB RAM. Fig. 1 displays an overview of the system's hardware organization.

A process executes on a block of computing nodes in one of the following three modes:

- Symmetric multiprocessing mode (SMP);
- Virtual node mode (VN);
- Dual mode (DUAL); In **SMP** mode each computing node executes a single process with a maximum of 4 threads. The whole 2 GB of physical memory on the node is available to the process in a shared-memory fashion. The process can utilize the Linux pthreads library [4] or OpenMP [5] to support multi-threading in the node and can utilize Message-Passing Interface (MPI, [6]) to communicate with processes on the other nodes.

In **VN** mode each of the 4 cores on a computing node executes a separate process. There are 4 MPI tasks per node and a single thread per each task. The physical memory on the node is split between the tasks, and thus each process only has direct access to 512 MB of RAM. Multi-threading is not supported in VN mode and the tasks can communicate with each other using MPI.

The **DUAL** mode is a hybrid one; each node executing 2 processes with 2 threads each. In this mode 1 GB of RAM is directly accessible by each process. Like in SMP mode, pthreads, OpenMP and MPI can be used simultaneously.

2.2. GROMACS specifics

GROMACS [1] is a flexible software package for performing molecular dynamics simulations. It is mainly used to study the behavior of bio-chemical structures, but is also quite effective in non-biological setups where non-bonded interactions prevail. The equations of motion are solved using modern scalable parallel algorithms and its realization is multi-platform – it works on a wide range of machines, from single-core laptops to supercomputers with hundreds of thousands of cores. It is licensed under GNU GPL and can be freely used, amended and distributed. The source size exceeds 1,6 million lines of C and FORTRAN code.

GROMACS employs one of two possible communication/synchronization mechanisms for working in parallel environments. It can either use pthreads or MPI, but not both. Each mechanism is better suited for certain multi-computer organization: MPI is applicable for distributed-memory configurations, while threads are applicable for shared-memory configurations. Most of the modern parallel computing environments, including the IBM Blue Gene/P, employ hybrid configurations: they are made up of independent computing nodes (distributed memory setup), which communicate with each other via MPI, but each computing node includes several cores and each core can spawn a thread in a shared memory environment. GROMACS however cannot utilize such hybrid architecture, since **it cannot be compiled to support both MPI and threads**. When such compilation is attempted, the configure script explicitly checks for it and complains that it is not supported.

2.3. Problem definition

The IBM Blue Gene/P cannot rely solely on threads. Its organization does not permit more than 4 threads per task. Some form of inter-process communication like MPI is necessary for high-performance computing. Thus, if GROMACS is to be run on an IBM Blue Gene/P, it needs to be compiled with MPI support. And since it cannot be compiled to support both MPI and threads, threads must be turned off. As a consequence, **the only**

meaningful mode of executing GROMACS on an IBM Blue Gene/P is VN mode, since it is the only purely MPI configuration. In SMP and DUAL modes both MPI *and* threads are required in order to utilize the full processing power of the machine. Running GROMACS in SMP mode for example leads to 3 of the 4 processors on each node being idle, thus wasting 75% of its theoretical performance.

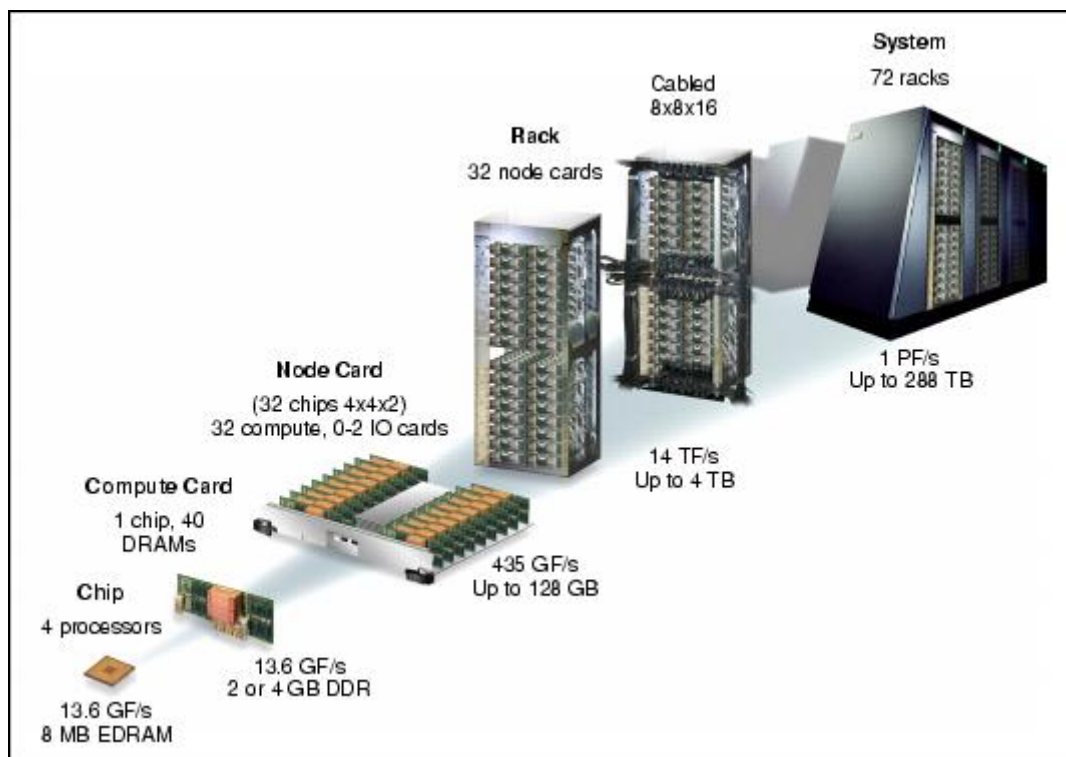


Fig. 1. IBM Blue Gene/P hardware organization (source: IBM, [3])

A profound difficulty when using VN mode is that the amount of memory available to a process is severely limited to only 512 MB. If a job uses memory structures which are not distributed among its parallel tasks, and the size of these structures depend on the problem size, the job will sooner or later run out of memory at certain problem size.

External evidence that GROMACS's master node hosts memory structures that are not distributed is visible on Fig. 2. It pictures the dependency of the memory required for a simulation of a 5,504,000 atom system (lignocellulose), to the number of MPI nodes used. It is clear from the graph, that there is a more or less constant component which offsets the master's node footprint, amounting to about 250MB. It should be noted that the quality of the results is not very good due to the method used to obtain the memory requirements (using GNOME's monitor reading), but they do serve to show the constant component.

For certain problem size this non-distributed global state becomes large enough to fill in the entire process memory, which leads to a system crash even before the actual simulation starts. The exact size at which this happens is hard to estimate, since the memory footprint depends strongly on the complexity of the system, but we have not been able to load any system with more than ~ 2,600,000 atoms on any Blue Gene/P partition in VN mode.

To summarize, GROMACS on Blue Gene/P is unable to simulate even moderately large systems due to the combination of the following factors:

- GROMACS cannot use both MPI and threads simultaneously, which leaves VN mode as the only possible mode of execution. This limits the memory available to each MPI task to 512 MB;
- GROMACS's memory footprint depends on the input size of the problem;
- GROMACS uses memory structures that are not distributed and reside in their entirety on the master MPI node;

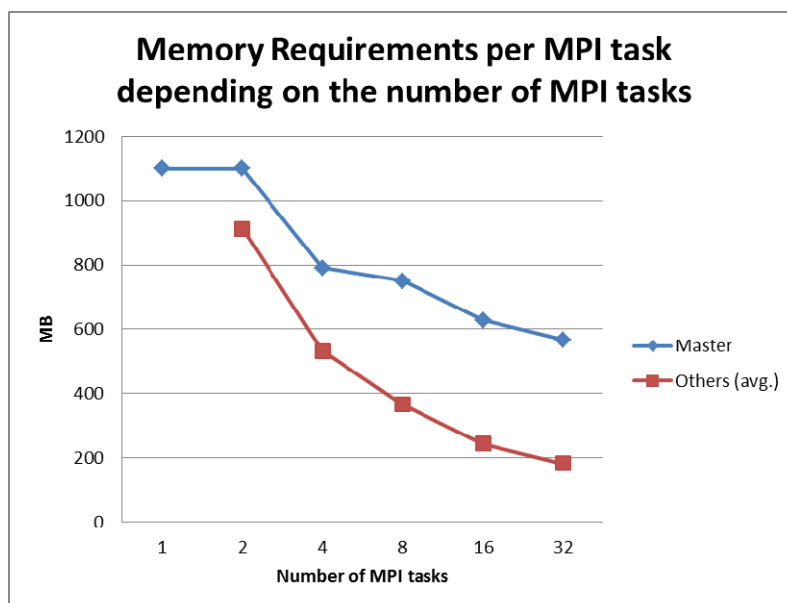


Fig. 2. GROMACS memory requirements per MPI task, depending on the number of MPI tasks. The simulated system contains 5,504,000 atoms worth of lignocellulose biomass.

3. Proposed solution

The problem identified in the previous section cannot be solved by simply adding more computing nodes – it is one of them that cannot fit all input data inside the 512 MB limit. In order to solve it, we need to remove one or more of its contributing factors.

The first factor, GROMACS being unable to use both MPI and threads, is very hard to overcome. It would require to rewrite the parallel core of the simulator, which is not feasible for the duration and the budget of the project. Moreover, even if this is achieved one day, it will not fix the underlying condition, but only shift the problem to a different scale, this time the limit being 2 GB – the maximum amount of memory in SMP mode.

The second factor, namely the memory footprint depending on the input size of the problem is inherent in the nature of the algorithms being used to perform the molecular dynamics simulation. There are no known algorithms in this domain with $O(1)$ memory complexity, so overcoming this factor is not an option.

The third factor is the one we should aim to remove, by **distributing any significantly large memory structures on all nodes**. By “significantly large structure” we mean a structure whose memory complexity is at least $O(N)$, N being the number of atoms in the system. If we treat all nodes in a symmetrical fashion, they will all have similar footprint. Then, if a system cannot fit in the combined RAM of the computing nodes, we can simply bring in more nodes until it does. This, by the very definition of the term scalability [7], is a defining characteristic of a truly scalable solution.

The solution to the problem then boils down to: a) identifying any significantly large non-distributed structures in GROMACS; and b) distributing them across all MPI tasks

3.1. Structures, identified for distribution

After carefully analyzing the source code, we found out that the coordinates and velocities of the atoms comprising the system are not distributed. They are kept in a structure called `t_state`, which contains all the non-static information required to define the state of the system being processed. The structure is defined in `include/types/state.h` and the relevant piece is:

```

rvec      *x;      /* the coordinates (natoms) */

rvec      *v;      /* the velocities (natoms) */

```

Fig. 3. Definition of the arrays, containing the coordinates and velocities of all the atoms within the system

Here, `natoms` is the number of atoms in the system, and the type `rvec` (real vector) is defined as an array of 3 "real" numbers, 3 being the number of spatial dimensions. The precision of the "real" numbers depends on whether GROMACS was compiled with single or double floating point support. Quick calculations show that in double precision for the system with 5,504,000 atoms these two arrays occupy $5,504,000 * 2 * 3 * 8 = 252$ MB, which is consistent with the difference between the memory consumption of the master node and all other nodes.

Further analysis showed that these two arrays are used not only on the master node to contain the global configuration of the system, but also on all other nodes to contain the local state – that part of the global state that is relevant for the partition of the system this node is dealing with. This complicates the task of directly replacing the arrays with their distributed counterpart, since we would now need two different places to keep the global (fully distributed) and local state.

Thus, the `t_state` is amended as follows:

```

rvec      *x;      /* the coordinates (natoms) - local state */

rvec      *v;      /* the velocities (natoms) - local state*/

vrvec_t   *gx;     /* the coordinates (natoms) - global state */

vrvec_t   *gv;     /* the velocities (natoms) - global state*/

```

Fig. 4. Addition of two new 'virtual real vectors' – `gx` and `gv`, that will be used to contain global state

The global state is to be kept in the two new arrays `gx`, and `gv`, which are of new type – `vrvec_t` (virtual real vector type). This new type is defined in an abstract way and is only used through pre-defined interface functions. This hides the details about the exact implementation of the distribution mechanism and allows this mechanism to be easily changed in the future, without changing the rest of GROMACS.

3.2. `vrvec_t` Implementation

For implementing `vrvec_t`, we decided to use the Global Arrays Toolkit library from PNNL [8]. It is an efficient platform-independent API, which provides exactly what we need – a Global Array abstraction – an array, whose storage is distributed in the memory of all MPI processes, but from the application point of view is

accessed uniformly, as if being in the local memory. This practically hides the distributed nature of the memory of the host machine and one can create arrays as large as needed, as long as they fit in the total combined memory of all computing nodes. GAT is successfully incorporated in many packages like NWChem, MOLPRO, UTChem, MOLCAS, TURBOMOLE, etc. It is very stable and mature, open sourced and compatible with MPI without replacing it. All these facts helped us decide on using the GAT to implement our virtual arrays.

3.3. Code Changes

In order to implement the virtual arrays inside GROMACS we had to perform the following activities:

- Change the configuration scripts to allow the user to choose at build time whether he/she wants GROMACS compiled with virtual arrays or not, and if yes, which implementation to use. For now the only supported implementation is Global Arrays Toolkit, but as mentioned earlier, the mechanism is extensible and other implementations are certainly possible in the future. As of version 4.5.4, GROMACS comes with two sets of configuration scripts – one based on `autoconf`, and another based on `cmake`. According to GROMACS's web-site however, this will be the last version in which `autoconf` is to be supported, thus we decided to only change the `cmake` configuration mechanism. The bottom line is that if someone needs virtual arrays, he/she has to use `cmake` to configure GROMACS.
- Change `mdrun`'s (the molecular dynamics simulator) set of command line arguments to include a new argument, `-va`, which instructs the runner to use global arrays where applicable. If this argument is not supplied, the original code is used. This argument is only available if the user built GROMACS with virtual arrays support.
- Specify the `vrvec_t` interface – type definition and function signatures. This is done in a new file, `include/types/vrvec.h`
- Implement the `vrvec_t` functions, using the interface provided by Global Arrays Toolkit. This is done in a new file, `src/gmxlib/vrvec_gat.c`, the whole file being conditionally compiled only if GAT is selected as the virtual arrays driver;
- Replace all references to `x` and `v`, when used in a global context (by the master node) with corresponding functions from the `vrvec_t` interface, acting on `gx` and `gv`. This has to be done recursively and by cloning the original functions in which the references appear, since many of them are also used in local context and moreover, they should continue to work the way they did (with `x` and `v`) in case the `-va` switch is not given on the command line. These changes need to be done recursively – if a function is cloned to use global coordinates/velocities, and it passes them to other functions it calls, they need to be cloned as well. Over 200 functions were changed this way, both for the regular MD simulation and for the energy minimization solver;
- Each occurrence of the global arrays `gx` and `gv` and any functions from the `vrvec_t` interface appear in a block that is conditionally compiled only if the user chose to build GROMACS with virtual arrays supported, and only in a dynamic execution environment where the `-va` switch is given at the command line. This guarantees that GROMACS will continue to work in the original way if any of these two conditions is not met.

4. Results

The primary goal of the project was to overcome the inherent memory limitation in GROMACS on IBM Blue Gene/P, which was due to the master node storing non-distributed global state. After re-implementing the coordinates and velocities storage and handling, we managed to load the 5,504,000 atom system shown on Fig. 5 (lignocellulose biomass, here shown without the solvent) on a 128 node partition and perform simulation with 512 MPI tasks. This was previously impossible with the original GROMACS implementation, regardless of the number of MPI tasks used. This shows that the implementation is scalable and the goal of the project is achieved. GROMACS on IBM Blue Gene/P is now fully scalable in the data sense and can perform Peta-scale simulations, limited only by the total combined amount of memory of the whole machine.

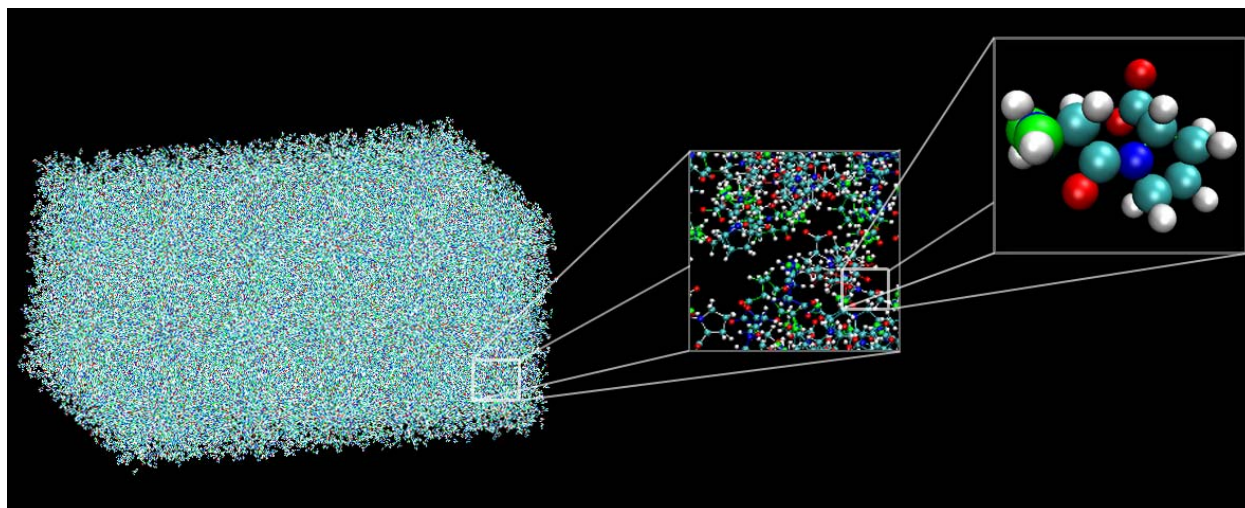


Fig. 5. Lignocellulose biomass system, containing 5,504,000 atoms – 64,000 elements of 26 atoms each for a total of 1,664,000 atoms of bio-chemical compound, plus 3,840,000 atoms of solvent (1,280,000 molecules of water). This system was processed by GROMACS on a 128-node partition (512 MPI tasks in VN mode) at BGSC's IBM Blue Gene/P, Sofia.

5. Future work

Currently, the virtual arrays technique is only applied as a pre/post-processing mechanism. Basically, this means that virtual arrays are only used to load and store the global state on disk. During domain decomposition, the master node decides which parts of the global state to give to which of the slave nodes, and distributes them using standard MPI protocols. This approach leads to unnecessary communication, like in the following scenario – suppose atom A is physically stored on node K. During domain decomposition, master M fetches the coordinates/velocities for A (communication from K to M) and then M decides that A needs to be processed by K; it then sends information for A back to K (communication from M to K), this time K keeping the received coordinates/velocities in its local state. We have two communications and two places where A is permanently stored – in the global state and in one of the local states. There is definitely a room for improvement here.

A better approach would be to revise the domain decomposition entirely and instead of distributing parts of the global state to the slave nodes, the master would just let the slave nodes work with the part of the global arrays stored in their memory, probably rearranging the global array first. It is not clear whether this would reduce communication requirements (rearrangement of the global array also requires communication), but it will cut the total memory requirements by half.

Acknowledgments

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528 and FP7-261557. The work is achieved using the PRACE Research Infrastructure – the Tier-1 IBM Blue Gene/P system at BGSC, Sofia. The authors would like to thank Prof. Stoyan Markov from NCSA for providing the inspiration and for the continuous support and also to Dr. Jeremy Smith from the Centre for Molecular Biophysics at the Oak Ridge National Laboratory for providing the lignocellulose data file.

References

1. GROMACS official web site <http://www.gromacs.org/>
2. JUGENE home page <http://www2.fz-juelich.de/jsc/jugene>
3. C. Sosa and B. Knudson. *IBM System Blue Gene Solution: Blue Gene/P Application Development*. IBM Redbooks Publications, September 2010, <http://www.redbooks.ibm.com/redbooks/pdfs/sg247287.pdf>

4. The Open Group Base Specification Issue 6: IEEE Std 1003.1, 2004 Edition. Technical report, The Open Group, 2004.
<http://pubs.opengroup.org/onlinepubs/009695399/basedefs/pthread.h.html>
5. OpenMP official web site.<http://openmp.org/wp/>
6. MPI: A Message-Passing Interface Standard. Technical report, Message Passing Interface Forum, 2009
<http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf>
7. A. Bondi Characteristics of scalability and their impact on performance, *Proceedings of the 2nd international workshop on Software and performance*, Ottawa, Ontario, Canada, 2000, ISBN 1-58113-195-X, pages 195 - 203
8. Global Arrays Toolkit official site. <http://www.emsl.pnl.gov/docs/global/>