EasyChair Preprint

№ 1613

Reproducible Pedagogy for Cognitive Dissonance Reduction

Rohit Goswami, Sonaly Goswami, Pranay Baldev, Shaivya Anand and Debabrata Goswami

October 9, 2019

Dissonance Reduction Tooling for HPC Pedagogy

1st Rohit Goswami
Department of Chemistry
Indian Institute of Technology Kanpur
Kanpur, India
rgoswami@iitk.ac.in

2nd Sonaly Goswami
Department of Chemistry
Indian Institute of Technology Kanpur
Kanpur, India
sonaly@iitk.ac.in

3rd Pranay Baldev
Department of Electrical Engineering
Indian Institute of Technology Kanpur
Kanpur, India
bpranay@iitk.ac.in

4th Shaivya Anand
School of Environmental Science and Technology
Indian Institute of Technology Kharagpur
Kharagpur, India
shaivya.anand@iitkgp.ac.in

5th Debabrata Goswami
Department of Chemistry
Indian Institute of Technology Kanpur
Kanpur, India
dgoswami@iitk.ac.in

***Abstract*—We describe a general work-flow which scales intuitively to high-performance computing (HPC) clusters for different domains of scientific computation. We demonstrate our methodology with a radial distribution function calculation in C++, with mental models for FORTRAN and Python as well. We present a pedagogical framework for the development of guided concrete incremental techniques to incorporate domain-specific knowledge and transfer existing expertise for developing high-performance, platform-independent, reproducible scientific software. This is effected by presenting the acceleration of a radial distribution function, a well known algorithm in computational chemistry. Thus we assert that for domain specific algorithms, there is a language-independent pedagogical methodology which may be leveraged to ensure best practices for the scientific HPC community with minimal cognitive dissonance for practitioners and students.**

***Index Terms*—pedagogy, best-practices, tooling, methodology, reproducible-research**

I. Introduction

High performance scalable computing techniques have permeated all fields of science, engineering and technology. Digital literacy has gained traction as an invaluable tool for scientific reproducible research, with basic tools being more than adequately covered by initiatives such as the Carpentries [1], [2] workshops and certifications. In addition to the renewed focus on digital tools and their use, the community has also recognized the necessity of well developed code for research [3]. However, even as open workflows have been recognized by the life sciences [4], [5], high performance tools have not been addressed in terms of their unique pedagogical issues. Given that distributed computing in general is considered to be a complex topic, it is natural that students unused to computational techniques, comfortable in their own scientific domains would not be able to leverage the appropriate compute, as it has been reported that the view of a novice and an expert per domain will differ [6]. Herein we present an overview of contemporary HPC education, and develop a methodology to incorporate best practices while facilitating familiarity by suitable generalizations and variations [7], [8]. We have taken the domain specific example of a standard analysis code for the calculation of the radial distribution function in three dimensions [9]. We enumerate the nature of the algorithm in pseudo-code, as the equivalent steps in both performance oriented C++, FORTRAN, and a "simpler" high level language (python) to show how each may be linked to domain specific representations. Furthermore we then show the natural extension of the mental model developed for C++ to encompass concepts for the parallelism on distributed systems and thus present an optimal pedagogical perspective for guided practice [10] of high performance computing.

II. Implementation and Models

That coherent group activities enhance the understanding and nature of scientific endeavors has been long understood in the pedagogical literature [11], [12]. In keeping with the same principles, we have used a striped down set of software development best-practices from life sciences and physical sciences [5], [13]–[15] suitable for novices, which may be described below with an optimal progression of topics:

1) **Logging** This is established by version control at a granular scale, and by the Zenodo archive at a coarse-grained, software-level perspective. The source code available on Github [16] for collaborative development, and to ensure reproducible workflows we implement a Zenodo archive [17] with versioning where the data-sets and history of the code may be tracked by mile-stoning.
2) **Consistency** Software linters and code commit guidelines are a crucial part of scalable computing, and we have found that, for pedagogical purposes, it is sufficient to introduce them as part of the git collaborative guidelines. The principles of guided practice are found to be at odds with the varying levels of comfort learners have while approaching a project to be reworked for distributed execution, so in this instance we have enforced consistency via Clang (`clang-format`) without being tied to an IDE.

3) **Reliability** The build process, consisting of a CMake modular system to maintain homogeneity across machines is also amenable to stricter dependency management via reproducible cryptographic hashes of the nix packaging system [18].
4) **Documentation** Incorporating a suitable self generating documentation tool like Doxygen or Sphinx allows students to couple the code concepts to the underlying science in an efficient manner. For this project, we have implemented a Doxygen workflow, integrated with Travis CI to ensure that the documentation is built and served by default (at https://eduhipc2019.femtolab.science/) as shown in Fig. 1.

Function documentation

```
int gr(double* rdfArray,
       int* nframes,
       double binsize,
       int nbin,
       std::vector<double> box,
       std::vector<std::vector<double>> coord,
       double cutoff,
       int nop,
       int switchVar)
```

#include <code/cpp/include/rdf.hpp>

Parameters	
rdfArray [in]	The array for holding the histogram for the $g(r)$ values
nframes [in]	The total number of frames for which the $g(r)$ is sampled
binsize [in]	The user-specified bin-width or bin size
nbin [in]	The total number of bins in the $g(r)$ histogram
box [in]	The simulation box lengths, required for calculating the total volume
coord [in]	A vector of vectors, holding the coordinates of the particles
cutoff [in]	This is the maximum length upto which the $g(r)$ will be calculated, and should be less than half the box length
nop [in]	The total number of particles in the simulation box
switchVar [in]	Int whose value determines whether initialization (0), sampling (1) or normalization (2) will be performed
Returns	an int value of 0 (success) or 1 (failure)

Function for calculating the RDF or $g(r)$

The is calculated for a bulk system, containing identical particles of a single type. Depending on the value of switchVar, either initialization, sampling or normalization of the $g(r)$ is performed.

Fig. 1. Doxygen generated API level documentation of function descriptors for the RDF namespace

Once serial code has been developed as per the guidelines above, we then assert the following generic progression and tooling for distributed, high performance execution:

1) **Algorithm Analysis** With the documentation in place, it should be relatively easy to determine the pain points for the software. The focus here is to recognize the portions which are amenable to be accelerated and are not limited primarily by I/O.
2) **Memory Management** Identifying the data which is to be transferred to the networked devices is crucial for memory allocation. The networked devices may be explicit, as in connected in a cluster, or implicit, as in between a GPU and CPU on the same machine. For OpenACC directives [19], the ability to allocate memory for the parallel regions is the last step of optimization, before delving into low level library code, like CUDA.
3) **Code Refactor** Once the previous optimizations have been completed, it is possible for advance data structure changes to be implemented with a focus on scalable performance, like the transference of data as space filling curves [15], however, this is typically too advanced for most domain specific practitioners.

Similar to the principles of the carpentries [1], the pedagogical concepts described will not typically allow for exponential performance gains for generic scientific software, however, given the proliferation of frameworks like OpenACC [19], it is reasonable to assume this is the intended goal for most teachers.

III. Radial Distribution Function

The radial distribution function (RDF) $g(r)$, or the first order pair correlation function is an appropriate starting point to generate a domain specific language for high performance computation, as it defined in the form of a deviation function. That is, it is the ratio of the average number density from a given atom to that of the average number density of an ideal gas of the same density [9]. A pictorial schematic of the RDF logic is shown in Fig. 2.

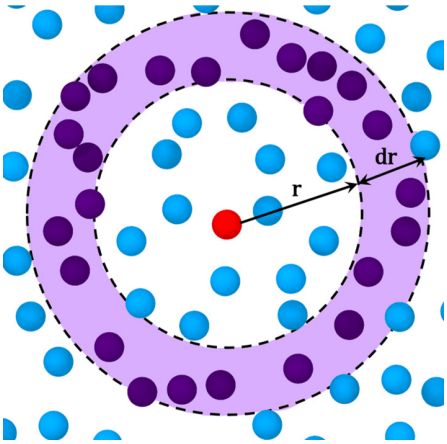

Fig. 2. Schematic of the RDF logic, pictorially describing the binning of pairwise distances from a reference particle.

The algorithm is amenable to distribution in theory, however, in practice the size of the underlying data-set is often a limiting factor without special considerations. The pseudocode for the algorithm is given in Algo. 1 and can be paraphrased as:

1) **Initialization** The $g(r)$ array is initialized to zero.
2) **Sampling** The histogram is added to for a particular bin, if the distance of a pair of atoms falls within the r associated with the bin.
3) **Normalization** Every bin of the $g(r)$ array is normalized by the product of the number of ideal gas particles in that bin, and the number of particles and number of frames.

We have considered a "water box" containing 4096 water molecules with inter-particle interactions defined by the mW force-field [20]. This was simulated using LAMMPS [21] with a 10fs timestep using the velocity verlet algorithm for the integration. We have used the isothermal–isobaric (NPT)

Data: Molecular Dynamics Trajectory
Result: $g(r)$ v/s r in a plain-text file
1 initialization;
2 **if** *Initialization* **then**
  /* Calculate the bin size */
3 numFrames=0; /* Frames processed */
4 numBins= (cutoff/binSize)+1;
  /* Initialize rdf array to zero */
5 **foreach** *element* **do** from 0 to numBins
6 element=0;
7 **end**
8 **end**
9 **if** *Sampling* **then**
  /* Update number of frames */
10 numFrames=numFrames+1;
11 **while** *iatom=1* $\rightarrow$ *nparticles-1* **do**
12 **while** *jatom=1*$\rightarrow$*nparticles* **do**
  /* Calculate distance between iatom and jatom */
13 **if** r_{ij} <*cutoff* **then**
  /* Update Histogram */
14 ibin=Nearest Integer of $(r_{ij}/binSize)$;
15 $g(ibin) = g(ibin) + 2$;
16 **end**
17 **end**
18 **end**
19 **end**
20 **if** *Normalization* **then**
  /* Loop over bins */
21 **while** *ibin=1*$\rightarrow$*numBins* **do**
  /* Calculate the distance r for each ibin */
22 r=binSize$\times$(ibin+0.5);
  /* Volume between bins i+1 and i */
23 binVolume=$((i+1)^3 - i^3) \times (binSize^3)$;
  /* Number of Ideal Gas particles in bin volume */
24 numIdeal=$(4/3) \times \pi \times$ binVolume $\times \rho$;
  /* Normalize by using the product of numFrames, numberOfParticles and numIdeal, called normFactor */
25 $g(ibin) = g(ibin)/$normFactor;
26 **end**
27 **end**

Algorithm 1: Pseudo code implementation of the RDF. In line number 15, the addition of two to the histogram counter is to account for the contribution of both ij and ji pairs in the loop. On line 24, ρ is the average, bulk number density of the system.

ensemble at a pressure of 1 atm and a temperature of 298K implemented by the Nose-Hoovier barostat and thermostat.

IV. Programming Models

A. Python

Python is often the programming language of choice for undergraduate students [22], [23] for its ease of use. However, the mental model for python programming, being as it is heavily dependent on OOP (Object Oriented Programming) is often detrimental to the structuring of data for parallel execution. This is due to the fact that though certain aspects of the code are linked conceptually and might be seen to benefit from being classified together, but this often yields large objects which need to be passed repeatedly, thus proving to be a performance bottleneck. Beyond this, the dependency-hell brought on by conflicting versions of python packages is more often than not unavoidable, and save for specialized container systems or nix derivations, package management systems are not scalable for distributed computing. Furthermore, larger code bases suffer from the lack of dedicated debugging support, and execution pathways are difficult to trace due to the dynamic typing. We describe the mental model in Figure 3.

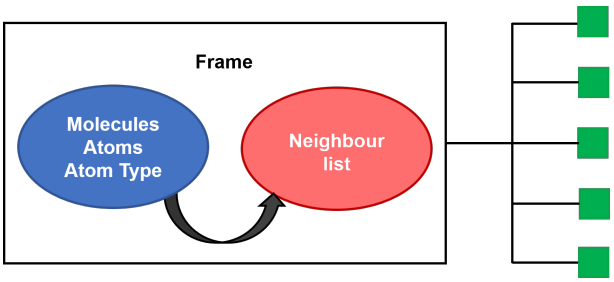

Fig. 3. A naive OOP workflow would be to group the associated coordinates and types as a single object (blue), while keeping the neighbor-list in a separate data-structure (red). The neighbor-list is expensive to generate and is frame dependent. In practice, such conceptually elegant groupings are unwieldy as the entire frame needs to be distributed over the HPC cluster.

B. FORTRAN

FORTRAN is a statically typed, math oriented scientific programming language with a long and distinguished pedigree including well-known libraries like BLAS [24]. It enjoys support for many optimization libraries including OpenACC, and is supported by GDB. FORTRAN is somewhat less flexible in terms of the programming paradigms which may be implemented, and difficulties in terms of having to deal with shared memory regions make it more complicated than required for most domain specific coding novices. It is relatively difficult to design incremental exercises for guided practice sessions towards distributed computing, and hence we have found it to be unsuitable.

C. C++

Given that C++ is the super-set of the C language which forms the basis of many of the system kernels, we have opted

to leverage it for our teaching purposes. C++ is widely used in the scientific community [21], [25], especially at the back-end, where most optimizations take place. Given the debugging ease due to GDB, along with its statically typed nature, namespaces, and build systems like CMake further improve the viability of the language for distributed computing. The ease with which programming paradigms, from object oriented to functional styles may be mixed makes it more viable for practitioners to design incremental exercises. Furthermore, the self documenting support provided by Doxygen (Fig. 1) allows for an API level link between sections of the code and the scientific principles being implemented in a cohesive and visually pleasing whole.

V. Reassigning Language

Without any loss of generality, we demonstrate the development of concepts of high performance computing in linguistic terms amenable to the molecular dynamics community. The naive algorithm described in Algo. 1 is essentially a brute force nested loop over every pair of particles in the system, without repetition. It is immediately evident, that we require each reference atom to have access to every other particle in the system, by virtue of the RDF being a pair correlation function. Furthermore since every particle in the system has to be a reference particle $N(N-1)$ pair calculations are required. To share this expensive calculation over multiple processors to attain the optimal speedup limited by Amdahl's law [26], we therefore understand from the nature of the RDF, that we require the coordinates of every atom to be distributed to each processor. This consideration leads us to reject arbitarily large structures containing information not relevant to the RDF calculation.

VI. Parallelism

The heart of this particular algorithm is the double nested `for` loop. Practically, there are I/O bottlenecks, and the exact data structures used are often too large to be passed effectively. We have leveraged the generalized OpenACC [19] `pragma` preprocesser directives to control the flow of data and direct parallelism. We note that the memory requirements of data structures to be passed must be determined in advance for optimal results. It is also trivial, with the nix packaging system [18] to test multiple compilers including PGI, Clang and GNU compilers. The pseudo-code for the optimization to be generated is given in Algo. 2 while the un-optimized nested loops implementing Algo. 1 is shown in Fig. 4. The OpenACC library defaults to shallow copies, so it is non-trivial to distribute STL containers such as a vector of vectors.

VII. Conclusions

We are able to show by example, a methodology which enables domain specific scientists and programming novices to prepare their code in a manner suited for distributed execution. We assert that a modest understanding of the core concepts of HPC computing is sufficient for performance enhancements for existing software, and have used the principles of guided

Data: Molecular Dynamics Trajectory
Result: $g(r)$ v/s r in a plain-text file

```
1  if Sampling then
       /* Update number of frames                    */
2      numFrames=numFrames+1;
       /* These two loops are to be
          accelerated, sizes of coord and
          g are known                                */
3      #pragma acc data copyin(coord[nparticles][3])
        copy(g[nbins]);
4      while iatom=1 → nparticles-1 do
5          #pragma acc loop while jatom=1→nparticles
            do
               /* Calculate distance between
                  iatom and jatom                    */
6              if r_ij <cutoff then
                   /* Update Histogram              */
7                  ibin=Nearest Integer of (r_ij/binSize);
8                  g(ibin) = g(ibin) + 2;
9              end
10         end
11     end
12 end
```

Algorithm 2: Indicative regions of parallelism. 3 depicts passing in the data-structure explicitly.

```
//  Loop over all pairs of atoms
∀ (ℤ  iatom = 0; iatom < nop - 1; iatom++) {
  // Loop through jatom
  ∀ (ℤ  jatom = iatom + 1; jatom < nop; jatom++) {
    // Get the periodic distance r_ij
    dist.resize(box.size());
    r2 = 0.0;
    ∀ (ℤ  k = 0; k < box.size(); k++) {
      dist[k] = coord[iatom][k] - coord[jatom][k];
      // Apply PBCs
      dist[k] -= box[k] * round(dist[k] / box[k]);
      r2 += pow(dist[k], 2);
    }
    r_ij = sqrt(r2);
    // Only add if r_ij is within the cutoff
    if (r_ij < cutoff) {
      ibin = (ℤ )(r_ij / binsize);
      rdfArray[ibin] += 2;
    } // end of check

  } // end of loop through jatom
}   // end of loop through every iatom

 ⟼ 0;
} // end of accumulation
```

Fig. 4. The bottleneck of Algo. 1 is the nested loop shown here. The pairwise distance and square-root calculation is also expensive.

practice and incremental sub-goal completion for the same. We demonstrate how modern tools are able to bring together concepts of science and HPC computing with minimal dissonance. We utilize the methodology and pedagogical tooling described for the specific case of molecular dynamics radial

distribution function acceleration. It is expected that this work be followed by statistical studies formally affirming the results of our experiences and will also facilitate the development of better, performant, reproducible scientific workflows.

References

[1] G. Wilson, "Software Carpentry: Getting Scientists to Write Better Code by Making Them More Productive," *Computing in Science Engineering*, vol. 8, no. 6, pp. 66–69, Nov. 2006. DOI: 10/bf3n4h.

[2] T. K. Teal, K. A. Cranston, H. Lapp, E. White, G. Wilson, K. Ram, and A. Pawlik, "Data Carpentry: Workshops to Increase Data Literacy for Researchers," *International Journal of Digital Curation*, vol. 10, pp. 135–143, Feb. 18, 2015, ISSN: 1746-8256. DOI: 10/gf5hjw. [Online]. Available: http://www.ijdc.net/index.php/ijdc/article/view/351 (visited on 10/08/2019).

[3] C. Goble, "Better Software, Better Research," *IEEE Internet Computing*, vol. 18, no. 5, pp. 4–8, Sep. 2014. DOI: 10/vjz.

[4] A. Prlić and J. B. Procter, "Ten Simple Rules for the Open Development of Scientific Software," *PLOS Computational Biology*, vol. 8, no. 12, e1002802, Dec. 6, 2012, ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1002802. [Online]. Available: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002802 (visited on 09/22/2019).

[5] S. Altschul, B. Demchak, R. Durbin, R. Gentleman, M. Krzywinski, H. Li, A. Nekrutenko, J. Robinson, W. Rasband, J. Taylor, and C. Trapnell, "The anatomy of successful computational biology software," *Nature Biotechnology*, vol. 31, no. 10, pp. 894–897, Oct. 2013, ISSN: 1546-1696. DOI: 10.1038/nbt.2721. [Online]. Available: https://www.nature.com/articles/nbt.2721 (visited on 09/22/2019).

[6] M. T. H. Chi, R. Glaser, and M. J. Farr, *The Nature of Expertise*. Taylor & Francis Group, Jan. 1, 1988, 480 pp., ISBN: 978-0-89859-711-0.

[7] R. Catrambone, "Generalizing Solution Procedures Learned From Examples," p. 12, DOI: 10/ccxs7m.

[8] D. W. Braithwaite and R. L. Goldstone, "Effects of Variation and Prior Knowledge on Abstract Concept Learning," *Cognition and Instruction*, vol. 33, no. 3, pp. 226–256, Jul. 3, 2015, ISSN: 0737-0008, 1532-690X. DOI: 10/gf9kbq. [Online]. Available: http://www.tandfonline.com/doi/full/10.1080/07370008.2015.1067215 (visited on 10/08/2019).

[9] D. Frenkel and B. Smit, *Understanding Molecular Simulation: From Algorithms to Applications*. Elsevier, Oct. 19, 2001, 661 pp., ISBN: 978-0-08-051998-2.

[10] K. A. Ericsson, R. T. Krampe, and C. Tesch-Romer, "The Role of Deliberate Practice in the Acquisition of Expert Performance," p. 44, DOI: 10/d7gc4g.

[11] L. K. Weir, M. K. Barker, L. M. McDonnell, N. G. Schimpf, T. M. Rodela, and P. M. Schulte, "Small changes, big gains: A curriculum-wide study of teaching practices and student learning in undergraduate biology," *PLOS ONE*, vol. 14, no. 8, e0220900, Aug. 28, 2019, ISSN: 1932-6203. DOI: 10/gf675b. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0220900 (visited on 10/02/2019).

[12] J. Bransford, N. R. C. (U.S.), and N. R. C. (U.S.), Eds., *How People Learn: Brain, Mind, Experience, and School*, Expanded ed, Washington, D.C: National Academy Press, 2000, 374 pp., ISBN: 978-0-585-32107-3.

[13] S. Crouch, N. C. Hong, S. Hettrick, M. Jackson, A. Pawlik, S. Sufi, L. Carr, D. De Roure, C. Goble, and M. Parsons, "The Software Sustainability Institute: Changing Research Software Attitudes and Practices," *Computing in Science & Engineering*, vol. 15, no. 6, pp. 74–80, Nov. 1, 2013, ISSN: 1521-9615. DOI: 10/bzmr. [Online]. Available: https://aip.scitation.org/doi/abs/10.1109/MCSE.2013.133 (visited on 10/08/2019).

[14] R. Goswami, A. Goswami, and J. K. Singh, "D-SEAMS: Deferred Structural Elucidation Analysis for Molecular Simulations," Sep. 21, 2019. arXiv: 1909.09830 [cond-mat, physics:physics]. [Online]. Available: http://arxiv.org/abs/1909.09830 (visited on 10/08/2019).

[15] R. Goswami, A. Goswami, and D. Goswami, "Space Filling Curves: Heuristics For Semi Classical Lasing Computations," in *2019 URSI Asia-Pacific Radio Science Conference (AP-RASC)*, New Delhi, India: IEEE, Mar. 2019, pp. 1–4, ISBN: 978-90-825987-5-9. DOI: 10/gf5mqk. [Online]. Available: https://my.pcloud.com/publink/show?code=XZ8vxr7ZHFxXSXuD2J4ntMcQoyPGXFodmegy (visited on 08/01/2019).

[16] *dgFemtoLab/eduhipc2019*, FemtoLab (IITK), Oct. 9, 2019. [Online]. Available: https://github.com/dgFemtoLab/eduhipc2019 (visited on 10/09/2019).

[17] R. Goswami and A. Goswami, "yodaChill Iterative HPC code," version v0.1-alpha, Oct. 2019. DOI: 10.5281/zenodo.3478180. [Online]. Available: https://doi.org/10.5281/zenodo.3478180.

[18] E. Dolstra, M. de Jonge, and E. Visser, "Nix: A Safe and Policy-Free System for Software Deployment," p. 15, 2004.

[19] R. Farber, *Parallel Programming with OpenACC*. Newnes, Oct. 14, 2016, 328 pp., ISBN: 978-0-12-410459-4.

[20] V. Molinero and E. B. Moore, "Water Modeled As an Intermediate Element between Carbon and Silicon," *J. Phys. Chem. B*, vol. 113, no. 13, pp. 4008–4016, Apr. 2, 2009, ISSN: 1520-6106. DOI: 10/dh2gkk. [Online]. Available: https://doi.org/10.1021/jp805227c (visited on 10/08/2019).

[21] S. Plimpton, “Fast Parallel Algorithms for Short–Range Molecular Dynamics,” p. 42,

[22] T. E. Oliphant, “Python for Scientific Computing,” *Computing in Science Engineering*, vol. 9, no. 3, pp. 10–20, May 2007. DOI: 10/fjzzc8.

[23] K. J. Millman and M. Aivazis, “Python for Scientists and Engineers,” *Computing in Science Engineering*, vol. 13, no. 2, pp. 9–12, Mar. 2011. DOI: 10/dc343g.

[24] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, “An extended set of FORTRAN basic linear algebra subprograms,” *ACM Trans. Math. Softw.*, vol. 14, no. 1, pp. 1–17, Mar. 1, 1988, ISSN: 00983500. DOI: 10.1145/42288.42291. [Online]. Available: http://portal.acm.org/citation.cfm?doid=42288.42291 (visited on 10/09/2019).

[25] D. V. D. Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. C. Berendsen, “GROMACS: Fast, flexible, and free,” *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1701–1718, 2005, ISSN: 1096-987X. DOI: 10/fjqfsm. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.20291 (visited on 10/09/2019).

[26] J. Jeffers and J. Reinders, *High Performance Parallelism Pearls Volume Two: Multicore and Many-Core Programming Approaches*. Morgan Kaufmann, Jul. 28, 2015, 574 pp., ISBN: 978-0-12-803890-1.