Reproducible Pedagogy for Cognitive Dissonance Reduction

Rohit Goswami, Sonaly Goswami, Pranay Baldev, Shaivya Anand and Debabrata Goswami
Dissonance Reduction Tooling for HPC Pedagogy

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 milestoneing.

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 re-worked 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/](https://eduhipc2019.femtolab.science/)) as shown in Fig. 1.

![Fig. 1. Doxygen generated API level documentation of function descriptors for the RDF namespace](image)

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.](image)

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 10 fs timestep using the velocity verlet algorithm for the integration. We have used the isothermal–isobaric (NPT)
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, \(\rho\) is the average, bulk number density of the system.
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 arbitrarily 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 preprocessor 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. 1 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 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


