ARMer: A Python Library for Adaptive Resource Allocation in High-Throughput Workflow

Qianzhen Shao[1] and Zhongyue J. Yang[1-4,*]

[1]*Department of Chemistry, Vanderbilt University, Nashville, Tennessee 37235, United States*

[2]*Center for Structural Biology, Vanderbilt University, Nashville, Tennessee 37235, United States*

[3]*Vanderbilt Institute of Chemical Biology, Vanderbilt University, Nashville, Tennessee 37235, United States* [4]*Data Science Institute, Vanderbilt University, Nashville, Tennessee 37235, United States*

ABSTRACT: High-throughput modeling requires allocation of different types of computing resources (e.g., GPU/CPU) for various computational sub-tasks in high-performance computing (HPC) clusters. To enhance efficiency of resource consumption, here we developed an adaptive resource allocation strategy to dynamically request computing resources based on the specific need of a certain modeling sub-task in the workflow. We implemented the strategy as a new Python library, i.e., adaptive resource manager (ARMer). As a proof of concept, we employed ARMer for allocating computing resources during the high-throughput enzyme modeling of fluoroacetate dehalogenase using EnzyHTP. The workflow involves four sequential sub-tasks, including: mutant generation, molecular dynamics simulation, quantum mechanical calculation, and data analysis. Compared to fixed resource allocation where both CPU and GPU are on-call for use during the entire workflow, the use of ARMer in the workflow can save up to 87% CPU hours and 14% GPU hours. In addition, ARMer allows parallel submission of multiple computational jobs in a job array and provides customized environment settings for each software used in the workflow.

Keywords: resource allocation, high throughput simulation, Python library

1. Introduction

High-throughput molecular modeling integrates multiple manual operations of simulation, such as input file preparation, software running, and data analysis, into one automatic workflow. In modern computational chemistry, high-throughput computation emerges as a new paradigm to address challenges in studying reaction mechanisms,[1-3] screening catalysts,[4-8] designing functional materials,[9-13] discovering drug candidates,[14, 15] and modeling enzymes[16-18]. With the advent of machine learning, these workflows allow facile collection of molecular features, or even provide high-accuracy computational data for model training.[19-22] Boosted by advances of computer hardware and software, high-throughput molecular modeling workflows enable the operation of large amount and diverse types of computational tasks.

As a common feature, high-throughput workflows involve multiple computational sub-tasks that are conducted by interfacing external software. As such, the workflow needs to allocate different computing resources for different sub-tasks. For example, a workflow of enzyme modeling (e.g., EnzyHTP[16]) needs to perform CPU-based structure model construction, GPU-based molecular dynamics (MD) simulation, CPU-based quantum mechanics (QM) calculation, and CPU-based data analysis. The diverse requirement of computing resources presents a challenge to operate high-throughput workflow in high-performance computing (HPC) clusters where a job scheduler is used to manage resource allocation.

To address this, the most straightforward and commonly used strategy is to request sufficient computing resources in one-time job submission for the whole workflow. The resource demand will be estimated based on the CPU and GPU needs of the most computationally demanding sub-tasks, regardless of how much resource remains idle in other sub-tasks in the

workflow. With fixed resource allocation, significant resource waste is expected. This issue will become more severe as new hardware added to the workflow, such as tensor processing unit and quantum computing units.

To enhance resource efficiency in HPC clusters, we developed an adaptive resource allocation strategy to dynamically request and distribute computing nodes for each sub-task in the high-throughput modeling workflow. The strategy was implemented as a Python application programming interface, known as adaptive resource manager (ARMer). As a proof of concept, we integrated ARMer with EnzyHTP[16] to adaptively allocate CPUs and GPU for sub-tasks in the high-throughput enzyme modeling of fluoroacetate dehalogenase (FAcD)[23-27]. Finally, we benchmarked resource and time consumption for adaptive resource allocation against fixed resource allocation.

2. Design and Implementation

The adaptive resource allocation strategy employs a "workflow script" that runs a single-CPU thread to manage sub-tasks for the entire high-throughput workflow. Using commands implemented in the ARMer Python library, the workflow script configures, submits, and monitors new jobs in HPC clusters that pertain to the actual need of computing resources in a sub-task of the workflow. This is in sharp contrast to the fixed resource allocation scheme where maximal computing resources are requested.

The ARMer Python library consists of two classes: the Job class and HPC class (Figure 1). The Job class (called ClusterJob in the code) defines variables and functions that are associated with job configuration, submission, and dynamic monitoring of job completion (Figure 2). The HPC class (subclasses of ClusterInterface in the code) supports the Job class with variables and functions to mediate external input/output in a local HPC cluster where ARMer is deployed.

Similar to *subprocess.run* in the Python standard library, ARMer library enables the "workflow script" to run shell commands - these commands are wrapped in the job scripts in the HPC clusters. Defined in the Job class, a job object is instantiated using the constructor *config_job()*, where the arguments *commands*, *cluster*, *env_settings*, and *res_keywords* are provided by the user (Supporting Information, Figure S1). Specifically, *commands* refers to the target shell commands for running external software for a specific enzyme modeling sub-task (e.g., *commands* = "g16 < filename.gjf > filename.out"); *cluster* refers to an HPC class object that contains miscellaneous details about the local HPC (e.g., *cluster* = Accre(), in which ACCRE refers to our local HPC at Vanderbilt University); *env_settings* states environment settings of external software (e.g., *env_settings* = '"module load Gaussian/16.B.01"'). Even for the same external software (e.g., Gaussian 16), the environment settings likely differ in different HPC clusters. Finally, *res_keywords* configures computing resources for the job (e.g., *res_keywords* = {'core_type' : 'cpu', 'nodes':'1', 'node_cores' : '8', 'job_name' : 'EnzyHTP_QMCluster', 'partition' : 'production', 'mem_per_core' : '3G', 'walltime' : '24:00:00'}), in which the parameters depend on the computational task (e.g., 8-24 CPUs for QM calculation, 1 GPU for MD calculation, etc.).

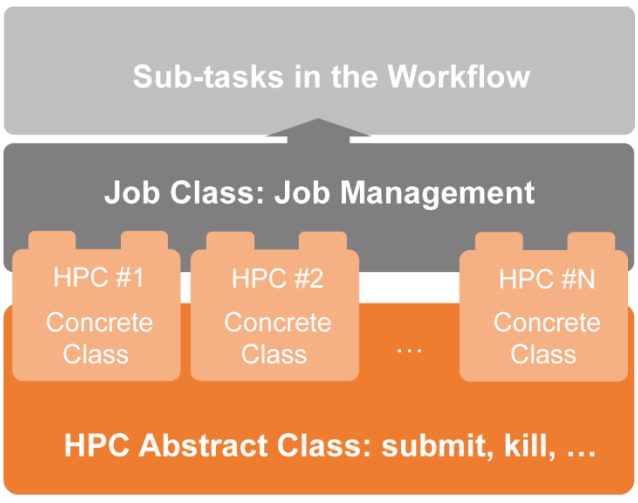

Figure 1. Two classes in the ARMer Python library: Job class and HPC class.

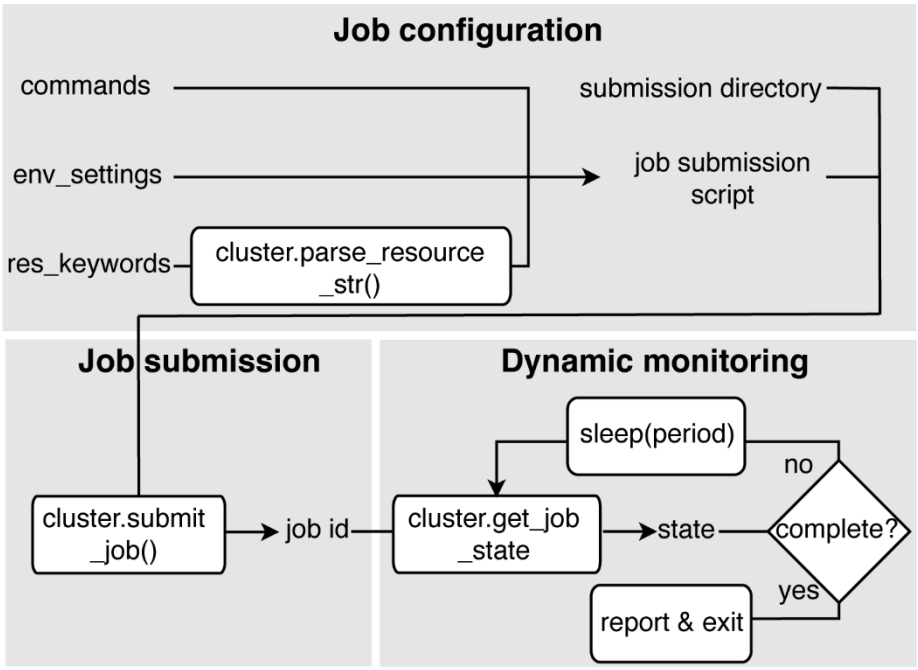

Figure 2. Variables and functions for job configuration, submission, and dynamic monitoring defined in the Job class.

With the job object instantiated, a job script for the required task can be generated (Figure 3) and then submitted by the *submit()* method (Figure 2 and Supporting Information, Figure S1). Notably, the format of the job script, the submission commands, and other HPC-dependent information are obtained from the HPC class object that is instantiated and passed to the *cluster* argument described above. Once the job has been submitted, a job ID is added to the object by the function. By tracing the job ID, the "workflow script" can monitor the status of a job object in the queue, and mediate the status by killing, holding, or releasing the job (Figure 2). Furthermore, the "workflow script" can dynamically detect the timing of the job completion by retrieving error or completion messages from the output file. Notably, the capability of dynamically monitoring the job completion status is vital to high-throughput modeling workflow. This is because the workflow

involves multiple different types of simulation sub-tasks that must be sequentially operated. In the case of running enzyme modeling workflow using EnzyHTP, after submitting an MD sampling task, the "workflow script" must put the rest of the sub-tasks on hold and wait for the conformational ensemble to generate before submission of the subsequent QM calculations.

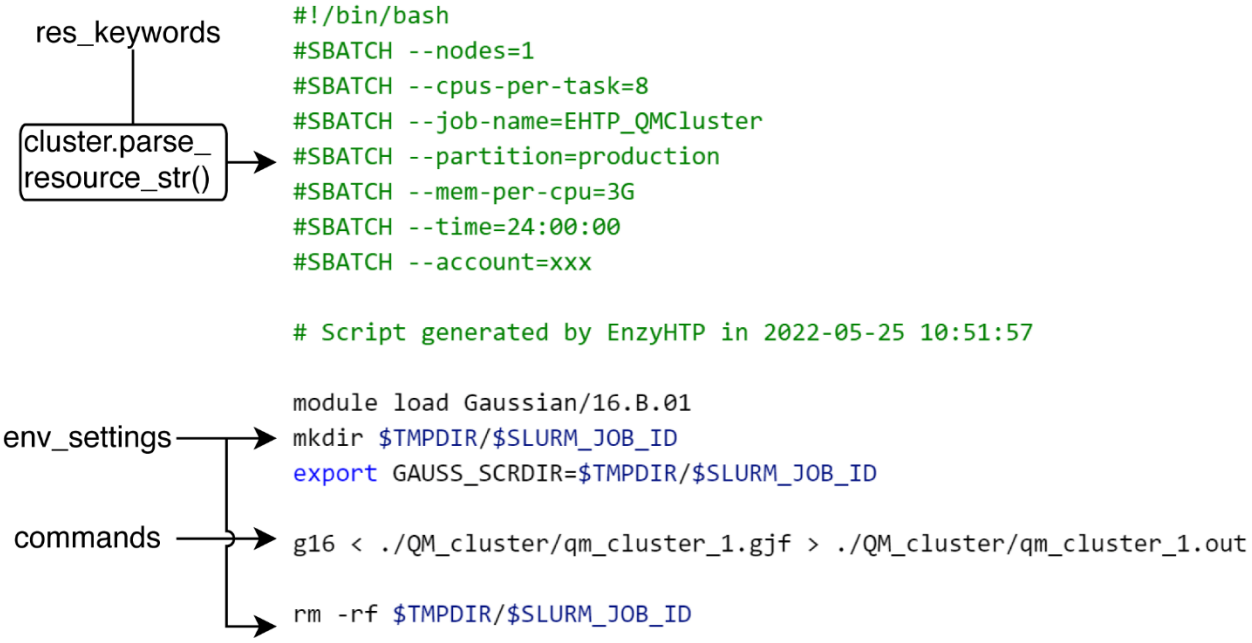

Figure 3. A sample job script that performs QM calculation using Gaussian 16. The job script is generated by the "workflow script" using commands implemented in ARMer.

Two methods have been implemented to achieve dynamic monitoring – they are: *wait_to_end()*and *wait_to_array_end()* method. The *wait_to_end()* method checks the status of a job in the job queue with a certain period of time (i.e., every 30 seconds) and exits upon the detection of messages that indicate job completion, error, or cancellation. The *wait_to_array_end()* method takes multiple job objects and submits them in one job array. Similarly, the method monitors the status of all jobs in the array regularly, and dynamically append new jobs to the array up to the maximal capacity (i.e., array size).

The HPC class files are stored in a folder named "cluster". These files allow users and developers to easily modify ARMer Python library to be compatible with their local HPC cluster. The instances of the HPC class are used as input for generating the Job instance. The methods of the HPC class are used by the Job instance to interface with a local HPC cluster. To make the Job class work properly with the HPC classes, an abstract HPC class, that defines code interfaces, is designed to generate concrete HPC classes. For example, a classmethod *submit_job()* defines the specific syntax for job submission in a local HPC cluster. If the user wants to customize the HPC class for their use, they need to refer to requirements in the abstract class. The inheritance of classmethods from the abstract HPC class will be automatically confirmed during the instantiation.

3. Results and Discussion

3a. Model System and Workflow. We employed fluoroacetate dehalogenase (FAcD)[23-27] as a model system and conducted a high-throughput workflow of enzyme simulations using EnzyHTP.[16] The workflow consists of four sequential sub-tasks, namely, 1) mutant structure construction, 2) MD simulation, 3) QM calculation, and 4) post-analysis. Notably, the model system and workflow have been applied in our previous work to test the throughput capability of EnzyHTP.[16] However, unlike the previous workflow that sampled 100 variants with minimalist resource cost, the current workflow only tested one enzyme variant (i.e, K83D) with MD and QM simulations set up to meet resource demand in actual computational research (Supporting Information .zip).

The first sub-task of the workflow uses the mutant generation module in EnzyHTP to generate a mutant structure (i.e, K83D, Supporting Information Figure S2a) and then optimizes the structure with molecular mechanics using the PMEMD module in AMBER[28]. The time cost

for mutant structure construction is nearly negligible and that for minimization takes ~30 seconds on 1 GPU. The second sub-task samples 100 conformers from a 100 ns MD trajectory simulated by the PMEMD module. The time cost of this step is ~14 hours on 1 GPU. Similarly, the computational cost for automatic preparation of MD input files can be omitted. Based on each of the sampled snapshot, the third sub-task conducts QM calculations of single point energy on the active site cluster (i.e., substrate + Asp110, Supporting Information Figure S2b) at the level of PBE0/def2TZVP using Gaussian 16.[29] In total, 100 independent QM calculations should be conducted. With 8 CPUs, each calculation takes ~2 minutes to complete. The last sub-task involves post electronic structure analysis to obtain electrostatic stabilization energy[30, 31], which is derived from the projection of protein electric field (computed by the point charges of enzyme residues with built-in EnzyHTP function) to the dipole of reacting bond (computed by Multiwfn[32]). The whole analysis take ~0.1 hours using 1 CPU. Overall, the entire workflow involves different types of modeling sub-tasks with diverse requirement of computing resources.

3b. Fixed Resource Allocation versus Adaptive Resource Allocation. We benchmarked the resource and time consumption of the workflow for two strategies: fixed resource allocation versus adaptive resource allocation, on our local HPC at Vanderbilt, i.e., advanced computing center for research and education (ACCRE).

Using fixed resource allocation, all computing resources (i.e., 1 GPU and 8 CPU) involved in the workflow are requested by a single submission script in one shot. Figure 4 shows the simulation type, time cost (i.e., vertical axis), and resource demand (i.e., GPU in orange and CPU in blue) associated with each sub-task in the workflow. Both CPU and GPU are requested for the entire workflow (Figure 4). However, resource waste is observed. Specifically, in the ~14 hours of MD simulation, only one GPU is used but the CPUs are primarily in idle mode (time-waste:

~8x14 = ~112 CPU hours); in the 3 hours of QM calculations, CPUs are used but the GPU is not (time-waste: 3 GPU hours). In the minimization and post-analysis sub-tasks, CPU and GPU are also not exploited, albeit the resource cost is trivial. Apparently, with fixed resource allocation, the collective resource waste is significant.

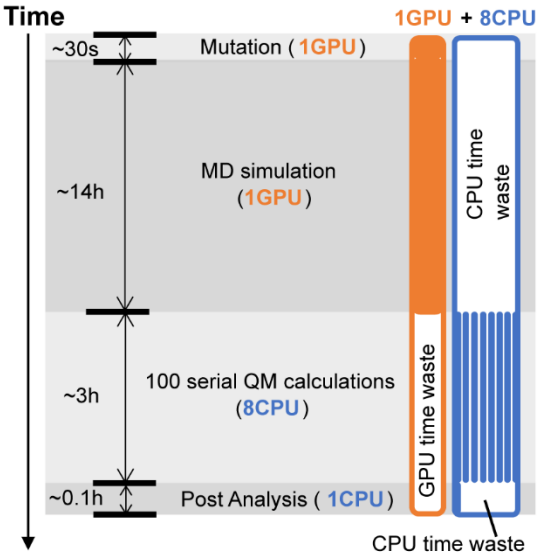

Figure 4. The high-throughput workflow of FAcD modeling using fixed resource allocation strategy, in which 1 GPU (in orange) and 8 CPUs (in blue) are requested in the beginning of the workflow. The type of modeling sub-tasks, time cost, and resource consumption are noted on the Figure.

Using adaptive resource allocation, a 96-hour wall-clock time, single CPU job is submitted that operates a Python "workflow script" to allocate resources for sequential sub-tasks involved in the workflow (Figure 5). The "workflow script" manages the sub-tasks using commands implemented in the ARMer library (detailed in the Design and Implementation section). Compared to fixed allocation strategy that directly execute sub-tasks using the allocated CPU or GPU, this

workflow script configures resource-demanding sub-tasks (i.e., need >1 CPU or ≥1 GPU) in a new job script and then submits the job to the queue (i.e., setting *ifcluster* = 'True' in the code).

In the MD simulation sub-task, the workflow script configures shell commands that run AMBER simulations in a job script along with the GPU request and environment settings. The workflow script then submits the MD job and regularly monitors the completion status of the job. After confirming the completion of MD, the workflow script will continue operating the QM calculation sub-task in the workflow. Since the 100 QM calculations are independent, the workflow script can submit multiple QM jobs (8 CPU each) simultaneously to the job array so that they can run in parallel up to the size limit of job array (i.e., 25 jobs) in local HPC cluster (Figure 5). New jobs will be submitted once the "workflow script" detects open slots on the array (see discussion of parallel computing using Python subprocess module, Supporting Information Text S1). With an array size of 25 jobs, one would expect an ideal time acceleration by a factor of 25 given the ideal condition of no job queueing time (i.e., ~2.4 hours as compared to 60 hours with all job running serially). Overall, with adaptive resource allocation, the resource waste can be minimized.

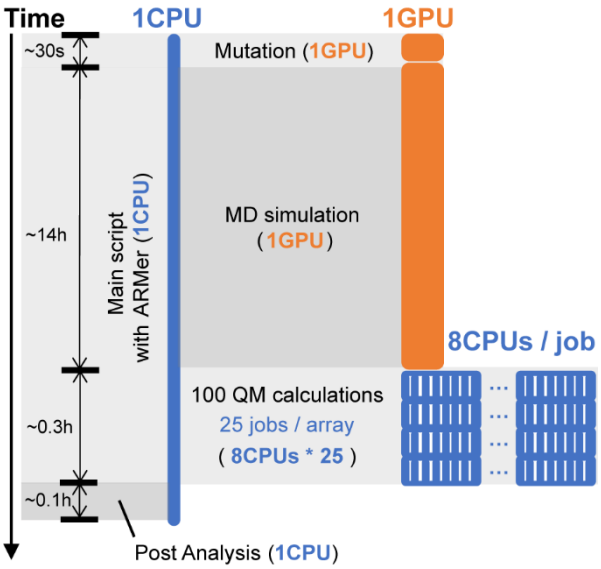

Figure 5. The high-throughput workflow of FAcD modeling using adaptive resource allocation strategy, in which a "workflow script" the runs on a single-CPU thread operates the modeling sub-tasks (i.e., mutation, MD, and QM) by configuring, submitting, and monitoring new job scripts. The MD job requests 1 GPU (in orange) and each QM job 8 CPUs (in blue). To submit and run individual QM calculations in parallel, a job array with a size of 25 is employed. The type of modeling sub-tasks, time cost, and resource consumption are noted on the Figure.

3c. Comparison of Computational Cost. We compared the computational cost (Figure 6) and wall clock time (Figure 7) for different sub-tasks of the workflow using fixed resource allocation (in red) versus adaptive resource allocation (in green) on ACCRE. The computational cost is represented by core-hours or SUs. To normalize the computational cost across CPU and GPU, a weighting factor (i.e., 1 SU of GPU = 54 SUs of CPU) is applied based on the high-performance linpack benchmark. In contrast, fixed resource allocation involves 14% resource waste in GPU-based MD simulation and 87% in CPU-based QM calculations, while adaptive resource allocation makes full use of these computing resources in every sub-task of the workflow (Figure 6). For wall

clock time, both strategies differ most significantly in the QM calculation sub-task (Figure 7). Enabled by parallel submission of individual QM jobs to a job array (i.e., 25 jobs per array), adaptive resource allocation strategy completes 100 QM calculations by 0.35 hours. With fixed resource allocation (i.e., 8 CPUs), the calculations are completed by 2.8 hours. Noticeably, the magnitude of efficiency acceleration (8-fold) is less than the ideal condition with parallel job submission (25-fold). The discrepancy is mainly caused by difference of CPU performance in ACCRE and the queueing time (Supporting Information Text S2). However, with larger QM system size and higher theory level, the QM resource consumption will be more significant. As such, the waste of resource and time for fixed resource allocation is expected to inflate.

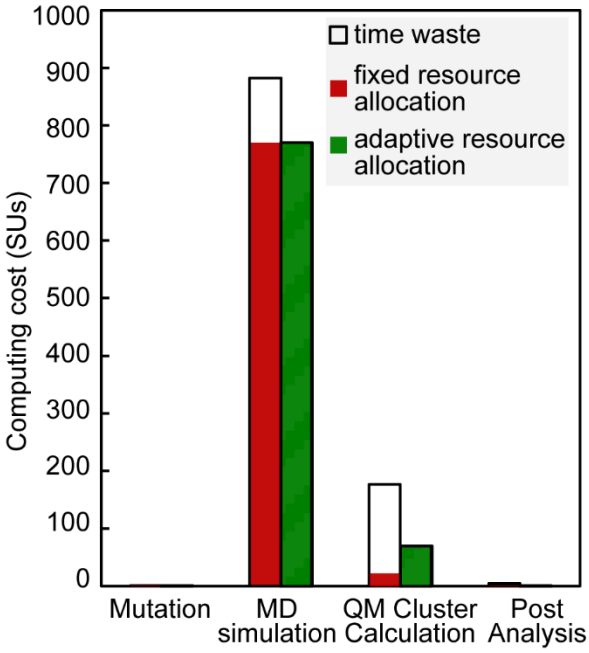

Figure 6. The comparison of computing resource cost (represented by core-hours) between fixed resource allocation (in red) and adaptive resource allocation (in green) for each sub-task in the enzyme FAcD modeling workflow using EnzyHTP. The resource time-waste is colored in white.

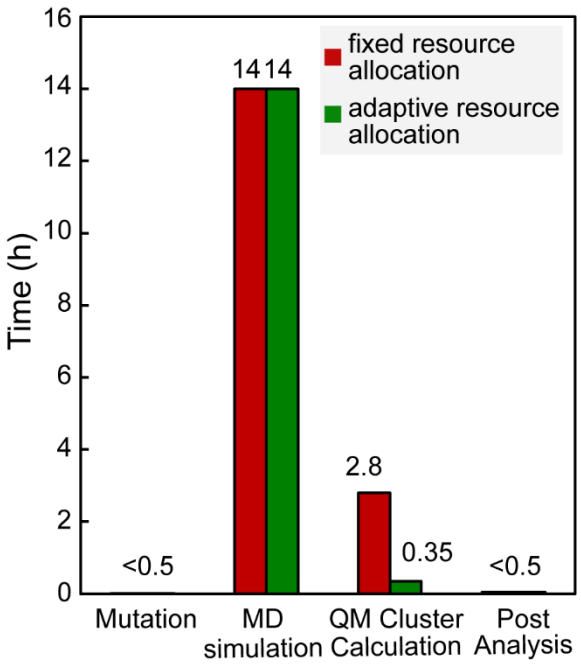

Figure 7. The comparison of wall-clock time cost between fixed resource allocation (in red) and adaptive resource allocation (in green) for each sub-task in the enzyme FAcD modeling workflow using EnzyHTP.

With new jobs generated, submitted, and monitored on the fly by adaptive resource allocation strategy, the resource waste observed by fixed resource allocation is no long expected because resources are requested and distributed based on the need of individual suc-task in the workflow. We should note that one potential caveat of adaptive resource allocation strategy is the time spent for queueing. In our test, the total amount of time spent for job queueing is about 12 minutes, which is relatively trivial. On extremely crowded HPC, we assume job queueing time to be much longer. The fixed resource allocation scheme suffers less from the queueing, but the overall cost of time and resource is substantial.

Besides being resource-efficient, adaptive resource allocation scheme also prevents conflict of software environment setting. Under the adaptive resource allocation scheme, the

workflow script can customize specific environment setting based on the requirement of an individual job. In contrast, the fixed resource allocation scheme will have to load all environmental variables in the beginning of the submission job, which can cause conflict for different software as the workflow proceeds.

4. Conclusion

We developed a Python library, ARMer, to adaptively allocate resources for computational sub-tasks in a high-throughput molecular modeling workflow in HPC clusters. The ARMer Python library consists of two classes: the Job class that defines variables and functions for job configuration, submission, and monitoring; the HPC class that supports the Job class to mediate external input/output in a local HPC cluster. Using commands implemented in ARMer, a "workflow script" can run on a single CPU thread to generate, submit, and monitor new jobs that call external software to execute sub-tasks.

As a proof of concept, we employed ARMer to manage the sub-tasks in the enzyme modeling workflow for FAcD using EnzyHTP. The sub-tasks include enzyme mutant structure construction, 100 ns MD simulation, 100 QM calculations, and electronic structure analysis. We compared the consumption of time and resource between two allocation strategies: fixed resource allocation versus adaptive resource allocation. Fixed resource allocation involves significant resource waste: in the 14 hours of MD simulation, only one GPU is used but the CPUs are primarily in idle mode (time-waste: 8x14 = 112 CPU hours); in the 3 hours of QM calculations, CPUs are used but the GPU is not (time-waste: 3 GPU hours). In contrast, the adaptive resource allocation makes full use of both computing resources in the corresponding sub-tasks of the workflow.

Moreover, ARMer allows parallel submission of multiple smaller computational jobs in a job array and provides customized environment settings for each software used in the workflow.

With the development of high-throughput molecular modeling workflow and the advent of more heterogenies HPC architecture (e.g., CPU, GPU, QPU, etc.), ARMer provides a general, easy-to-use, and easy-to-extend python interface to achieve adaptive resource allocation in HPC clusters.

ASSOCIATED CONTENT

Supporting Information. Example code of ARMer API; Structure of FAcD K83D mutant; Structure of the active site cluster; Choice of parallel strategy; Discrepancy of expected speed up and actual speed up from ARMer parallelization (PDF)

Input files for the fixed/adaptive resource allocation test: input structure; workflow python script; job submission script of the workflow script (ZIP).

Data and Software Availability. The code and sample input for ARMer is publically available at https://github.com/ChemBioHTP/ARMer. The input files and structures are provided as part of the SI files. EnzyHTP is available from https://github.com/ChemBioHTP/EnzyHTP/tree/develop. AMBER 19 is available from http://ambermd.org/. Gaussian 16 is available from https://gaussian.com/.

AUTHOR INFORMATION

Corresponding Author

*Email: zhongyue.yang@vanderbilt.edu phone: 615-343-9849

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENT

This research was supported by the startup grant from Vanderbilt University and the fellowship of Vanderbilt Institute of Chemical Biology. This work was carried out in part using computational resources from the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number TG-BIO200057.[33]

References

1. Young, T. A.; Silcock, J. J.; Sterling, A. J.; Duarte, F., autodE: Automated Calculation of Reaction Energy Profiles— Application to Organic and Organometallic Reactions. *Angewandte Chemie* **2021,** *133* (8), 4312-4320.
2. St. John, P. C.; Guan, Y.; Kim, Y.; Etz, B. D.; Kim, S.; Paton, R. S., Quantum chemical calculations for over 200,000 organic radical species and 40,000 associated closed-shell molecules. *Scientific Data* **2020,** *7* (1), 244.
3. Hruska, E.; Gale, A.; Huang, X.; Liu, F., AutoSolvate: A toolkit for automating quantum chemistry design and discovery of solvated molecules. *The Journal of Chemical Physics* **2022,** *156* (12), 124801.
4. An, Q.; Shen, Y.; Fortunelli, A.; Goddard, W. A., QM-Mechanism-Based Hierarchical High-Throughput in Silico Screening Catalyst Design for Ammonia Synthesis. *Journal of the American Chemical Society* **2018,** *140* (50), 17702-17710.
5. Nandy, A.; Duan, C.; Goffinet, C.; Kulik, H. J., New Strategies for Direct Methane-to-Methanol Conversion from Active Learning Exploration of 16 Million Catalysts. *JACS Au* **2022,** *2* (5), 1200-1213.
6. Janet, J. P.; Ramesh, S.; Duan, C.; Kulik, H. J., Accurate Multiobjective Design in a Space of Millions of Transition Metal Complexes with Neural-Network-Driven Efficient Global Optimization. *ACS Central Science* **2020,** *6* (4), 513-524.
7. Janet, J. P.; Duan, C.; Nandy, A.; Liu, F.; Kulik, H. J., Navigating Transition-Metal Chemical Space: Artificial Intelligence for First-Principles Design. *Accounts of Chemical Research* **2021,** *54* (3), 532-545.
8. Guan, Y.; Ingman, V. M.; Rooks, B. J.; Wheeler, S. E., AARON: An Automated Reaction Optimizer for New Catalysts. *Journal of Chemical Theory and Computation* **2018,** *14* (10), 5249-5261.
9. Colón, Y. J.; Snurr, R. Q., High-throughput computational screening of metal–organic frameworks. *Chem. Soc. Rev.* **2014,** *43* (16), 5735-5749.
10. Gan, Y.; Miao, N.; Lan, P.; Zhou, J.; Elliott, S. R.; Sun, Z., Robust Design of High-Performance Optoelectronic Chalcogenide Crystals from High-Throughput Computation. *Journal of the American Chemical Society* **2022,** *144* (13), 5878-5886.

11. Shen, J.; Hegde, V. I.; He, J.; Xia, Y.; Wolverton, C., High-Throughput Computational Discovery of Ternary Mixed-Anion Oxypnictides. *Chemistry of Materials* **2021,** *33* (24), 9486-9500.
12. Jain, A.; Hautier, G.; Moore, C. J.; Ping Ong, S.; Fischer, C. C.; Mueller, T.; Persson, K. A.; Ceder, G., A high-throughput infrastructure for density functional theory calculations. *Computational Materials Science* **2011,** *50* (8), 2295-2310.
13. Abreha, B. G.; Agarwal, S.; Foster, I.; Blaiszik, B.; Lopez, S. A., Virtual Excited State Reference for the Discovery of Electronic Materials Database: An Open-Access Resource for Ground and Excited State Properties of Organic Molecules. *The Journal of Physical Chemistry Letters* **2019,** *10* (21), 6835-6841.
14. McInnes, C., Virtual screening strategies in drug discovery. *Current Opinion in Chemical Biology* **2007,** *11* (5), 494-502.
15. Ekins, S.; Puhl, A. C.; Zorn, K. M.; Lane, T. R.; Russo, D. P.; Klein, J. J.; Hickey, A. J.; Clark, A. M., Exploiting machine learning for end-to-end drug discovery and development. *Nature Materials* **2019,** *18* (5), 435-441.
16. Shao, Q.; Jiang, Y.; Yang, Z. J., EnzyHTP: A High-Throughput Computational Platform for Enzyme Modeling. *Journal of Chemical Information and Modeling* **2022,** *62* (3), 647-655.
17. Amrein, B. A.; Steffen-Munsberg, F.; Szeler, I.; Purg, M.; Kulkarni, Y.; Kamerlin, S. C. L., CADEE: Computer-Aided Directed Evolution of Enzymes. *IUCrJ* **2017,** *4* (1), 50-64.
18. Doerr, S.; Harvey, M. J.; Noé, F.; De Fabritiis, G., HTMD: High-Throughput Molecular Dynamics for Molecular Discovery. *Journal of Chemical Theory and Computation* **2016,** *12* (4), 1845-1852.
19. Lewis‐Atwell, T.; Townsend, P. A.; Grayson, M. N., Machine learning activation energies of chemical reactions. *WIREs Computational Molecular Science* **2021**.
20. Li, X.; Zhang, S. Q.; Xu, L. C.; Hong, X., Predicting Regioselectivity in Radical C–H Functionalization of Heterocycles through Machine Learning. *Angewandte Chemie International Edition* **2020,** *59* (32), 13253-13259.
21. Yan, B.; Ran, X.; Jiang, Y.; Torrence, S. K.; Yuan, L.; Shao, Q.; Yang, Z. J., Rate-Perturbing Single Amino Acid Mutation for Hydrolases: A Statistical Profiling. *The Journal of Physical Chemistry B* **2021,** *125* (38), 10682-10691.
22. Jiang, Y.; Yan, B.; Chen, Y.; Juarez, R. J.; Yang, Z. J., Molecular Dynamics-Derived Descriptor Informs the Impact of Mutation on the Catalytic Turnover Number in Lactonase Across Substrates. *The Journal of Physical Chemistry B* **2022,** *126* (13), 2486-2495.
23. Makinen, M. W.; Fink, A. L., Reactivity and Cryoenzymology of Enzymes in the Crystalline State. *Annual Review of Biophysics and Bioengineering* **1977,** *6* (1), 301-343.
24. Schulz, E. C.; Mehrabi, P.; Müller-Werkmeister, H. M.; Tellkamp, F.; Jha, A.; Stuart, W.; Persch, E.; De Gasparo, R.; Diederich, F.; Pai, E. F.; Miller, R. J. D., The hit-and-return system enables efficient time-resolved serial synchrotron crystallography. *Nature Methods* **2018,** *15* (11), 901-904.
25. Mehrabi, P.; Di Pietrantonio, C.; Kim, T. H.; Sljoka, A.; Taverner, K.; Ing, C.; Kruglyak, N.; Pomès, R.; Pai, E. F.; Prosser, R. S., Substrate-Based Allosteric Regulation of a Homodimeric Enzyme. *Journal of the American Chemical Society* **2019,** *141* (29), 11540-11556.
26. Kim, T. H.; Mehrabi, P.; Ren, Z.; Sljoka, A.; Ing, C.; Bezginov, A.; Ye, L.; Pomès, R.; Prosser, R. S.; Pai, E. F., The role of dimer asymmetry and protomer dynamics in enzyme catalysis. *Science* **2017,** *355* (6322), eaag2355.

27. Mehrabi, P.; Schulz, E. C.; Dsouza, R.; Müller-Werkmeister, H. M.; Tellkamp, F.; Miller, R. J. D.; Pai, E. F., Time-resolved crystallography reveals allosteric communication aligned with molecular breathing. *Science* **2019,** *365* (6458), 1167-1170.
28. D.A. Case, H. M. A., K. Belfon, I.Y. Ben-Shalom, S.R. Brozell, D.S. Cerutti, T.E. Cheatham, III, G.A. Cisneros, V.W.D. Cruzeiro, T.A. Darden, R.E. Duke, G. Giambasu, M.K. Gilson, H. Gohlke, A.W. Goetz, R. Harris, S. Izadi, S.A. Izmailov, C. Jin, K. Kasavajhala, M.C. Kaymak, E. King, A. Kovalenko, T. Kurtzman, T.S. Lee, S. LeGrand, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, M. Machado, V. Man, M. Manathunga, K.M. Merz, Y. Miao, O. Mikhailovskii, G. Monard, H. Nguyen, K.A. O'Hearn, A. Onufriev, F. Pan, S. Pantano, R. Qi, A. Rahnamoun, D.R. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, J. Shen, C.L. Simmerling, N.R. Skrynnikov, J. Smith, J. Swails, R.C. Walker, J. Wang, H. Wei, R.M. Wolf, X. Wu, Y. Xue, D.M. York, S. Zhao, and P.A. Kollman Amber 2021.
29. Frisch, M. J.; Trucks, G. W.; Schlegel, H. B.; Scuseria, G. E.; Robb, M. A.; Cheeseman, J. R.; Scalmani, G.; Barone, V.; Petersson, G. A.; Nakatsuji, H.; Li, X.; Caricato, M.; Marenich, A. V.; Bloino, J.; Janesko, B. G.; Gomperts, R.; Mennucci, B.; Hratchian, H. P.; Ortiz, J. V.; Izmaylov, A. F.; Sonnenberg, J. L.; Williams; Ding, F.; Lipparini, F.; Egidi, F.; Goings, J.; Peng, B.; Petrone, A.; Henderson, T.; Ranasinghe, D.; Zakrzewski, V. G.; Gao, J.; Rega, N.; Zheng, G.; Liang, W.; Hada, M.; Ehara, M.; Toyota, K.; Fukuda, R.; Hasegawa, J.; Ishida, M.; Nakajima, T.; Honda, Y.; Kitao, O.; Nakai, H.; Vreven, T.; Throssell, K.; Montgomery Jr., J. A.; Peralta, J. E.; Ogliaro, F.; Bearpark, M. J.; Heyd, J. J.; Brothers, E. N.; Kudin, K. N.; Staroverov, V. N.; Keith, T. A.; Kobayashi, R.; Normand, J.; Raghavachari, K.; Rendell, A. P.; Burant, J. C.; Iyengar, S. S.; Tomasi, J.; Cossi, M.; Millam, J. M.; Klene, M.; Adamo, C.; Cammi, R.; Ochterski, J. W.; Martin, R. L.; Morokuma, K.; Farkas, O.; Foresman, J. B.; Fox, D. J. *Gaussian 16 Rev. C.01*, Wallingford, CT, 2016.
30. Fried, S. D.; Boxer, S. G., Electric Fields and Enzyme Catalysis. *Annual Review of Biochemistry* **2017,** *86* (1), 387-415.
31. Fried, S. D.; Boxer, S. G., Measuring Electric Fields and Noncovalent Interactions Using the Vibrational Stark Effect. *Accounts of Chemical Research* **2015,** *48* (4), 998-1006.
32. Lu, T.; Chen, F., Multiwfn: A multifunctional wavefunction analyzer. *Journal of Computational Chemistry* **2012,** *33* (5), 580-592.
33. Towns, J.; Cockerill, T.; Dahan, M.; Foster, I.; Gaither, K.; Grimshaw, A.; Hazlewood, V.; Lathrop, S.; Lifka, D.; Peterson, G. D.; Roskies, R.; Scott, J. R.; Wilkins-Diehr, N., XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering* **2014,** *16* (5), 62-74.

Table of Contents Graphic