Scalable Composition and Analysis Techniques for Massive Scientific Workflows

Dong H. Ahn*, Xiaohua Zhang†, Jeffrey Mast†, Stephen Herbein*, Francesco Di Natale*, Dan Kirshner†, Sam Ade Jacobs†, Ian Karlin*, Daniel J. Milroy†, Bronis De Supinski†, Brian Van Essen†, Jonathan Allen†, Felice C. Lightstone†
*NVIDIA, Santa Clara, CA, USA, †Lawrence Livermore National Laboratory, Livermore, CA, USA
Email: *{donga, sherbein, fdinatale, ikarlin}@nvidia.com
†{zhang30, mast2, kirshner1, jacobs32, milroy1, desupinski1, vanessen1, allen99, lightstone1}@llnl.gov

2022 IEEE 18th International Conference on e-Science (e-Science) | 978-1-6654-6124-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ESCIENCE55777.2022.00018

***Abstract*—Composite science workflows are gaining traction to manage the combined effects of (1) extreme hardware heterogeneity in new High Performance Computing (HPC) systems and (2) growing software complexity – effects necessitated by the convergence of traditional HPC with data sciences. Composing, analyzing, and optimizing a composite workflow remains highly challenging as the component technologies are generally developed in isolation and often feature widely varying levels of performance, scalability, and interoperability. In this paper, we propose novel workflow composition and analysis techniques to create and optimize a scalable and effective composite workflow for heterogeneous HPC centers, and define the performance space of variables that impact composite workflow performance. We present PerfFlowAspect, an Aspect Oriented Programming (AOP)-based tool to perform cross-cutting performance analysis of composite workflows and better understand the impact of key performance variables on workflows. Our solution directly addresses AOP concerns that can affect workflow performance and covers the full software lifecycle, ranging from the workflow's initial composition through performance analysis and optimization. We use our science workflow composition techniques to implement the American Heart Association Molecule Screening (AHA MoleS) workflow. Through experimentation, we demonstrate that tuning a single performance variable can improve AHA MoleS workflow performance by a factor of up to 2.45x. Our evaluation suggests that our techniques can significantly enhance the ability of a multi-disciplinary research and development team to create a high performance composite workflow.**

I. Introduction

Current economic factors are ushering in a new era of extremely heterogeneous HPC. As Moore's Law is tapering, specialized hardware such as graphics processing units (GPU) [30] is increasingly taking the role previously played by general purpose processors as the main computing workhorse. As cloud computing has become a dominant market force [13, 16], HPC has also begun to embrace a heterogeneous software environment, marrying its software stack to cloud solutions. Indicatively, 2018 was the first year in which new additions to the top 500 supercomputers derived more performance from specialized processors than from general purpose processors. Key cloud solutions including container and container orchestration technologies such as Kubernetes [22] and OpenShift (which is built on top of Kubernetes), are also making inroads into HPC infrastructure [12, 6].

At the same time, the complexity of science workflows on HPC systems is growing sharply. The convergence of traditional HPC and new simulation, analysis, and data science approaches including Machine Learning (ML) and Artificial Intelligence (AI) provides unprecedented opportunities for discovery, but also creates workflows [2, 8, 35] that are far more complex than traditional ones. These workflows often combine many different applications. The different application tasks in the workflow have vastly different computational requirements. Each different task may perform well on one type of computing hardware but not on others. Furthermore, the applications runtimes and execution order can be wildly different. Thus, this complexity must be managed through full automation. A fully automated scientific workflow must carefully map and run application tasks [8, 3, 17] to different specialized hardware (e.g., CPU, GPU, disaggregated AI accelerators [18], I/O storage of certain tiers in a multi-tiered storage subsystem [31]) and/or different system software (e.g., persistent container services such as message-queue and database services running on an on-premises Kubernetes cluster).

With a growing trend toward specialized hardware, large compute facilities will soon feature much higher hardware heterogeneity at all levels. They will not only increasingly field different heterogeneous compute node types, but also different partitions within a large system with different specialized hardware, and vastly different systems with different hardware types across the facilities. Thus, the mapping of application tasks in a workflow will result in going beyond the boundary of a single system and programming the entire center resources. Managing and mapping the tasks on heterogeneous resources across different systems will add far more complexity in multiple dimensions.

In this paper, we categorize some of the key challenges in running automated cross-system workflows that users face on current and future HPC centers. And we provide a solution for each challenge. We specifically identify challenges in three areas. 1) The conceptual challenge of understanding the key performance space of a cross-system workflow that can compose large numbers of domain-specific and general-purpose software components each with different performance charateristics. 2) The performance analysis challenge of gath-

978-1-6654-6124-5/22/$31.00 ©2022 IEEE
DOI 10.1109/eScience55777.2022.00018

ering performance data from large numbers of components to analyze the entire workflow without having to lose the modularity and uniformity in ways which code instrumention is cast across them. 3) The performance optimization challenge of easily re-configuring the workflow to tune the end-to-end workflow turn-around time performance in an iterative fashion. In particular, the main contributions of our paper include:

- Composite science workflow composition techniques that blend domain-specific components with reuseable workflow management components with ease of performance analysis and tuning principles;
- Defining the performance space of composite workflows, effective performance modeling and key variables pertaining to this space;
- An AOP technique to build a cross-cutting performance analysis concern into constituent components as our solution to the performance analysis challenge;
- A workflow software architecture to enable an easy iterative exploration of the performance space as our solution to the performance optimization challenge.

To study the effectiveness of our solutions to these challenges, we identified and closely worked with a multidisciplinary science team that had concrete requirements to create and automate a cross-system workflow at scale. As part of our collaboration, the teams developed the American Heart Association Molecule Screen (AHA MoleS) workflow with the goal of categorizing and solving the major challenges that emerge from running composite workflows across systems at large scale. Using representative AHA MoleS inputs, we ran the workflow across multiple supercomputers with different hardware types as well as an on-premises OpenShift Kubernetes cluster. All of the systems reside at Lawrence Livermore National Laboratory (LLNL), one of the world's largest HPC centers.

Our empirical evaluation using AHA MoleS suggests that our techniques enable easy composition, analysis, and optimization of this workflow in a multi-disciplinary research and software-development environment. To the best of our knowledge, our work is the first to identify specific challenges and solutions required to build and run a fully automated cross-system high-performance workflow combining three completely different types of computing resources at one of the world largest computing facilities.

II. Fast drug molecule screening

To accelerate small molecule drug design and development, HPC is widely used to virtually screen a library of small molecule compounds against a protein target. Each molecular docking calculation consists of two steps where the small molecule is positioned in the protein binding pocket (pose) and an energy score is calculated, predicting the molecular binding. The explosion of chemical libraries from millions of compounds to billions of compounds drastically changed the scale of virtual screening, demanding faster more efficient methods to accurately predict small molecule binding. As such, machine learning algorithms (Fusion) have been trained on molecular binding and can rescore molecules based on given poses. Frequently, multiple methods are used in combination to maximize successful predictions called "hits". Additionally, this same workflow is used to virtually screen molecules across the American Heart Association library of three-dimensional human proteins, composed of over 10,000 proteins, to predict proteins involved in potential side effects.

A. AHA MoleS

The resulting workflow is called AHA MoleS. Figure 1 shows how AHA MoleS builds on and combines two classes of software components: (1) modern general-purpose components including Maestro [7], Flux [10] and RabbitMQ [33] orchestrated by Kubernetes [22]; and (2) domain-specific workflow management and application software including ConveyorLC Docking [38] and Fusion [19], which are two of the main components in the Generative Molecular Design (GMD) [15], particularly in the GMD Structural Core (GMD SC).

B. General-Purpose Workflow Components

The Maestro Workflow Conductor is a lightweight HPC workflow tool that is capable of generically representing and orchestrating multi-step computational workflows [7]. Maestro centers around the concept of a "study": a YAML specification that allows users to mark up their workflow easily in a Bash-like syntax. It can then utilize the specification to construct a Directed Acyclic Graph (DAG) of tasks to be automatically scheduled using a specified scheduler interface. Maestro is written entirely in Python for portability.

RabbitMQ is a high-availability, scalable messaging broker. It supports a host of features, including messaging queues, message prioritization, and delivery acknowledgement. Multiple producers and consumers can send and receive messages simultaneously, making RabbitMQ an ideal platform for coordinating a workflow executed by many workers. Most important to a screening workflow such as AHA MoleS is the ability of RabbitMQ to be run within a datacenter-wide Kubernetes cluster, which is accessible over the network from every HPC cluster. This accessibility enables the workflows to pool resources across multiple heterogeneous clusters.

Flux is a fully hierarchical workload manager for HPC. Its hierarchical capabilities have proven to improve scalability and flexibility significantly through a divide-and-conquer approach that is well-suited for currently-emerging environments. Jobs and resources are divided among the schedulers in the hierarchy and managed in parallel. This approach significantly increases the scalability of Flux over traditional schedulers that rely on a centralized scheme. One of the key features of Flux is that it allows for both single-and multi-user modes. Modern composite workflows can leverage its single-user mode, where hierarchical resource management and job scheduling are provided in the *user space* within a batch allocation created by an HPC *system-level* workload manager, such as Slurm and IBM LSF. This allows researchers to set up their own customized hierarchies within a batch allocation as well as policies of Flux's graph-based job scheduler called Fluxion.

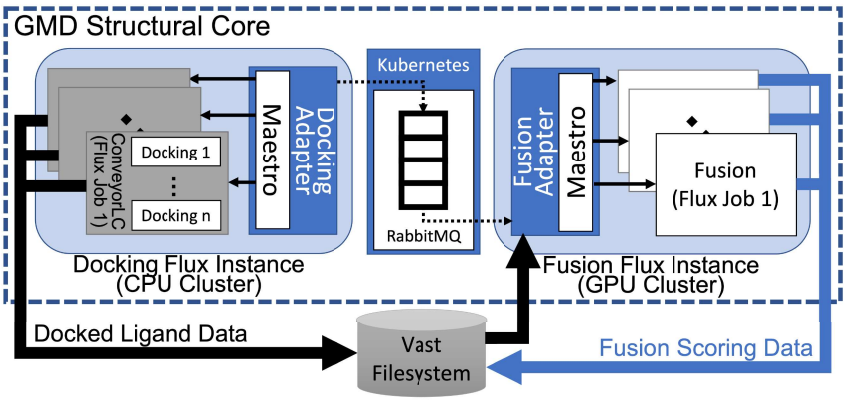


Fig. 1: AHA MoleS workflow architecture

These capabilities also allow users to tune additional scheduling policies such as queuing and resource matching.

C. Domain-Specific Software Components

ConveyorLC is a high-throughput virtual screening pipeline to automate the docking and rescoring of compounds against protein targets [38, 36]. This pipeline mainly includes four Message Passing Interface (MPI)-based parallel applications for protein preparation, ligand preparation, molecular docking, and Molecular Mechanics/Generalized Born-Solvent Accessible Surface Area (MM/GBSA) rescoring. The docking method in the ConveyorLC pipeline is based on Autodock Vina. A mixed MPI and multithreading hybrid parallel scheme is employed in the ConveyorLC docking application [37]. The virtual compounds were converted for the docking simulations by the ConveyorLC ligand preparation application.

Fusion is a structure-based ML model for protein-ligand binding affinity prediction based on the combination of a three-dimensional convolutional neural network (3D-CNN) and spatial graph neural network (SG-CNN) [19]. The 3D-CNN component is designed to capture 3D atomic interactions and implicit geometric configurations using a 3D voxel grid. The SG-CNN component is modified from the PotentialNet [9] architecture based on Gated Graph Sequence Neural Networks [23] and captures explicit pairwise atomic interactions. The fusion of the 3D-CNN and SG-CNN components models two complementary representations of protein-ligand interactions to give more robust prediction accuracy when evaluating new protein targets. The coherent Fusion model has been implemented in the PyTorch ML framework, which provides the ability to accelerate the calculations on GPUs [29].

Finally, GMD is an active learning drug discovery framework. It incorporates the AMPL pipeline [24], GMD SC, and a deep generative network model to iteratively search for small molecule drug candidates. GMD is designed to efficiently search through an exceptionally large chemical space and converge on chemical compounds predicted to have desirable drug-like properties including consideration of efficacy, safety, pharmacokinetics (PK), and synthesizability of the newly proposed compounds. GMD starts with a working compound library and uses AMPL to predict key pharma-relevant parameters of molecules. The GMD SC, including ConveyorLC and Fusion, predicts the binding affinity of compounds. The generative model uses design criteria to guide the generation of new compounds based on the predicted values calculated by AMPL, ConveyorLC, and Fusion. The newly proposed compounds are incorporated into the working compound library and used to inform the next iteration of the GMD loop. Once the design loop converges on novel compounds meeting the design criteria the most promising compounds are selected for experimental validation and to update the ML models. The AHA MoleS workflow adapts the GMD SC infrastructure to enable screening large working compound libraries against an expanded list of protein targets.

D. Challenges Encountered

We create the AHA MoleS workflow based on a purpose-built architectural blueprint. The software engineering of workflow composition is an important development. However, we find that it is difficult to measure the performance of the complete workflow. Gathering data on the workflow execution is a necessary step to measuring and improving performance, but reliance on trial-and-error necessitates a reconfigurable workflow. An iterative approach permits performance analysis and optimization on the entire workflow. In the following, we provide details on the computing environment where we evaluate our engineering developments, and then discuss the solutions to these challenges in the remaining sections. Although we develop the steps and techniques in the context of AHA MoleS, it is important to note that our study allows the techniques to be applied to the broader class of composite scientific workflows.

E. Cross-System Workflow Evaluation Environment

The blueprint of our target workflow is to couple molecular docking and an ML algorithm to score small molecule binding in the protein. Each component has completely different computational requirements, and thus this workflow requires to run

across multiple systems to be effective. This section details the systems we use to explore cross-system workflow challenges. We aim to build our workflow software until we can perform controlled benchmark runs using two supercomputers as well as an on-premises Kubernetes cluster simultaneously: Ruby, Lassen, and the Persistent Data Services (PDS) Red Hat OpenShift cluster at LLNL. Ruby is a CPU machine with a total of 1,512 nodes. Each node contains two Intel® Xeon® CLX-8276L CPUs with a total of 56 cores. Lassen is a machine using CPU-GPU heterogeneous architecture with a total of 795 nodes. Each of the nodes has two IBM® Power9™ CPUs with a total of 44 cores and four Nvidia® Tesla™ V100 (Volta) GPUs. The PDS cluster consists of three Kubernetes servers and five worker nodes, together with a NetApp® AFF A400 storage system providing object and persistent container storage. We targeted the molecule batch creation, ConveyorLC Docking jobs, and batch completion to run on Ruby while targeting Fusion jobs to run on Lassen. We map our applications this way because each of these applications perform best on respective type of computer hardware. We expect that this will be a typical configuration in the future as different applications in a workflow may perform well on a one type of computing hardware but not on others. And how long different applications need to run and in what order are also wildly different from one another. One significant system-level limitation is that our experiment used a Dedicated Application Time (DAT) request to guarantee the allocation of the compute resources on both supercomputers at the same time. This is because there is no center-level scheduler that can schedule across these machines in a coordinated fashion (e.g., allocating both at the same time or allocate Ruby resources first and then Lassen's). The settings and resource allocation for different calculations (e.g., Docking and Fusion) in the workflow are configurable.

III. Performace space of composite workflows

A science workflow is a sequence of tasks that are executed in a specific order with or without dependencies. What distinguishes a composite workflow such as that constructed from the components described in Section II from the traditional approach is that the tasks can be generated, scheduled, and ultimately executed by a set of many domain-specific and general-purpose components in concert and across multiple clusters. In a composite workflow, the domain-specific workflow managers are often interacting closely with a set of general-purpose workflow and resource/job manager components to push the increasingly finer-grained units of work or tasks through their work queues and resource allocators at multiple levels. Moving tasks via a pipeline of two different components often involves well-known operations (e.g., generate, submit, schedule, etc) as well as adaptation procedures between these components.

A. Performance Space

One of the most important goals of any science workflow is to complete its tasks as quickly as possible while being subject to constraints such as amounts of compute resources that can be simultaneously used, fair use of resources that are shared by multiple users, etc. Thus, it is essential to minimize the critical path that workflow tasks must go through in order to achieve this goal. If no performance overhead is incurred in performing scheduling-related operations, identifying those tasks that take longest at each level through the DAG and shortening the time to complete these tasks on the critical path should suffice. In practice however, key workflow scheduling and execution operations can incur significant performance overheads, which are affected by multiple *performance variables* (e.g., sizes of the task and architectures of the scheduler).

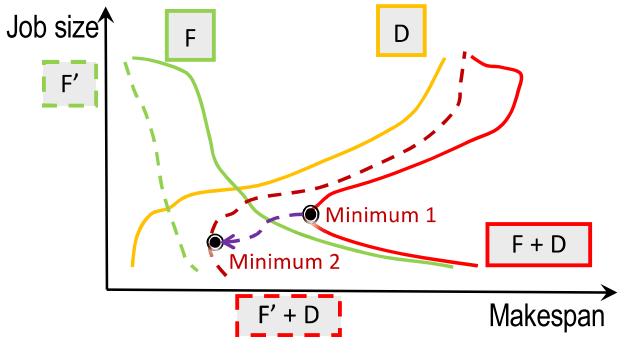
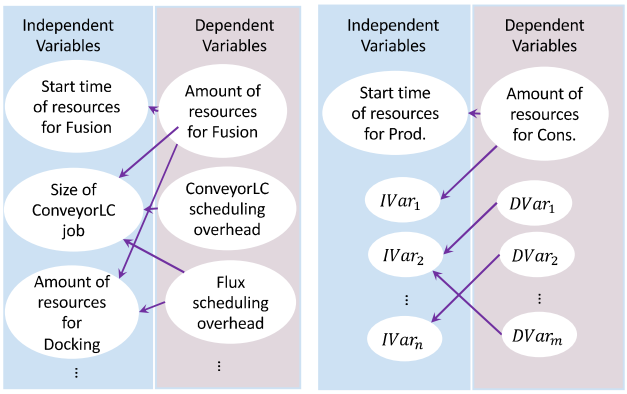


Fig. 2: Optimality relies on composite performance functions; F is Flux scheduling, F' is optimized Flux scheduling, and D is the task scheduler within ConveyorLC.

1) Relative performance: A task in a composite workflow must be adapted between large numbers of components before it can be allocated to a compute resource and executed at increasingly finer granularity. Thus, the relative performance and scalability characteristics of the components are critical for any composition.

Let us consider the performance and scalability of ConveyorLC and GMD SC components described in Section II. ConveyorLC implements its own ad hoc task-level scheduling on top of MPI. As it uses the leader-worker pattern in which each of the worker MPI rank processes fetches new docking tasks from the single leader MPI rank (rank 0) and thus the 0^{th} rank can experience a significant performance bottleneck at scale. To run it at scale, therefore, ConveyerLC demands an ensemble approach whereby a group of ConveyorLC jobs are managed and run by another component such as Flux. This performance characteristic is complicated by the GMD SC software, which is designed to run at varying scales to support a broader set of domain specific applications. One of the implications of this characteristic on the composition is that the number of ConveyorLC jobs that are exposed to GMD SC for coupling cannot be excessively large, a relative performance trait generally opposed to that of ConveyorLC. The final characteristics emerge from Flux: its scheduling performance can also vary depending on the workload.

2) Optimality relies on composite functions: Figure 2 illustrates optimality using the relative performance and scalability of characteristics of the task scheduler within ConveyorLC and



(a) Example (b) General

Fig. 3: Performance Variables

of the job scheduling and execution services within Flux. Here, our performance variable is the size of each ConveyorLC job that is scheduled and executed by Flux. Given a fixed number of compute nodes (N) that are managed by a Flux instance, ConveyorLC can run in many different configurations. It can run as single-node jobs (many jobs run in parallel), as a large N-node job (one job runs at a time), or anything in-between.

The performance of the scheduler within either component varies non-linearly and generally in an inverse manner. As the size of ConveyorLC jobs increases the number of jobs running simultaneously decreases, which lowers the performance overhead of scheduling and execution at the Flux level while increasing the overhead at the ConveyorLC scheduler level. Overall, the minimal makespan can only be determined on the sum of the performance functions of the two schedulers: $F + D$ or $F' + D$ as shown in Figure 2 where F is the performance curve of Flux scheduling, F' is that of its optimized scheduling, and D is the performance curve of the task scheduler within ConveyorLC.

3) Performance variables: An optimization becomes quickly non-trivial; real-world workflows would employ many more components than our rather simple examples and many performance variables beyond the job size variable would affect the performance space. Figure 3 further expands on our concept of performance variables. There are two types of performance variables: 1) independent performance variables and 2) dependent variables. As shown in Figure 3a, the amount of resources allocated to docking simulations is an independent variable. Researchers will choose it according to their science objective. The job size of ConveyorLC is also an independent variable: it will be chosen according to their prior understanding of the scheduler behavior. Likewise, the start time of the resources allocated to Fusion is an independent performance variable. By contrast, the performance overhead of Flux scheduling of ConveyorLC jobs become a dependent variable that depends on both the resource allocation scale for docking simulations and the size of ConveyorLC jobs. The performance overhead of ConveyorLC scheduling becomes a variable that depends on the size of ConveyorLC jobs. Finally, the minimum amount of resources that Fusion needs for the optimal makespan becomes another dependent variable that depends on all three independent variables. Here, the identification of the key performance variables and their dependence relationship is critical for performance optimization. If one component is optimized such that a dependent variable is affected, researchers must account for all other dependent variables correlated with it to understand its true impact.

IV. Manageable cross-component performance analysis

We now highlight our high-level principles for analyzing the performance of composite workflows. First, *each component scheduler must identify the key performance and scalability characteristics as pertaining to the planned composition as early as possible.* The domain experts of each component should be responsible for this identification and cross-discipline communications for later cross-component analysis. Second, *an overall workflow-level performance model must guide the initial composition to allow for easy reasoning of the overall workflow scheduling performance.* This model should be simple, yet rich to capture those events that are on the critical path of the resulting end-to-end workflow. Third, *a well-defined method to monitor the performance of the overall workflow is required.* A composite workflow almost always uses a large set of *preexisting* components, each of which would generally use different logging and built-in performance monitoring. Yet, we need a consistent way to gather performance data for a particular composition consisting of a disparate set of components to capture the critical-path events.

A. Casting Critical Path Analysis (CPA)-based cross-cutting performance concerns

For the first and second principles, we use CPA [34] as the basis for our performance modeling. CPA is a common technique to determine the cause of a workflow's makespan by finding the event path in the task execution history of the workflow that has the longest duration. This critical path identifies where in the workflow we should focus our attention. This helps researchers reason about how various components interact with one another as the tasks are moving through these operations for execution.

For the third principle, we developed a simple AOP [20]-based tool called PerfFlowAspect (PerfFlowAspect) [11]. It is a simple performance analysis tool that can cast a cross-cutting performance-analysis concern or aspect across a heterogeneous set of components used to create a composite science workflow. It is designed specifically to allow researchers to weave the performance aspect into critical points of execution across many workflow components without having to lose the modularity and uniformity of how performance is measured and controlled. It closely follows the AOP design

patterns to minimize code modification needed for the disparate workflow-management technologies and also to modularize the control for managing ways to cast cross-cutting performance-analysis instrumentation across many components.

PerfFlowAspect provides language support most relevant for HPC workflows such as Python, C and C++. For Python, it uses an annotating decorator whereby users can use `@PerfFlowAspect.aspect.critical_path()` to annotate their functions that are likely to be on the critical path of the workflow's end-to-end performance. These annotated functions then serve as the join points that can be weaved together with PerfFlowAspect and acted upon. Once annotated, running this python code will produce a performance trace data files that use Chrome Tracing Format (CTF) in JSON so that it can be loaded into Google Chrome Tracing or Perfetto to render the critical path events on the global tracing timeline.

When these annotated functions, or join points, are weaved together with PerfFlowAspect, users can invoke specific performance-analysis actions, a piece of tracing code, on those points of execution. They are often referred to as *advice* in AOP. One type of advice within PerfFlowAspect that our effort uses is Chrome tracing advice. This particular advice simply logs a performance event data in CTF. PerfFlowAspect's annotation also supports the notion of pointcut: a predicate that matches join points. In fact, `@PerfFlowAspect.aspect.critical_path()` can take an optional keyword argument called pointcut whose value can be around, before, after or their async variations (`around_async`, `before_async`, and `after_async`). `pointcut=around` will invoke the advice before and after the execution of the annotated function whereas `pointcut=before` or `pointcut=after` will only advise either corresponding point of function execution.

Unlike dynamic programming languages such as Python, C/C++ cannot natively and easily weave PerfFlowAspect into the target program (e.g., using language features like Python decorator). To overcome this language-level limitation, PerfFlowAspect introduces a simple LLVM pass which can detect critical-path annotations in the target C/C++ source files and to weave these annotated program points together with PerfFlowAspect's runtime library as part of Clang compilers' optimization pass. This allows C/C++ and Python users alike to use the consistent AOP abstraction of PerfFlowAspect. The consistent abstraction and tool usage flow for both dynamic languages and more traditional high performance computing programming languages like C/C++ significantly facilitate the high level of unity and modularity as required by the AOP paradigm.

Each component developer in a multi-disciplinary team like AHA MoleS can independently use PerfFlowAspect to annotate their functions that are likely to be on the critical path of the resulting workflow's end-to-end performance.

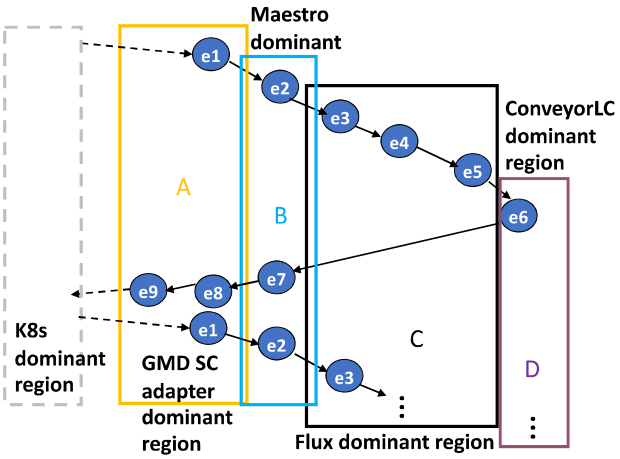


Fig. 4: Critical path of Docking tasks for AHA MoleS

B. Application to AHA MoleS

As shown in Figure 1, AHA MoleS can be broadly categorized as a producer-consumer workflow, where Docking tasks running on a CPU-only supercomputer produce molecular poses to be consumed by Fusion tasks running on a GPU-equipped supercomputer. Figure 4 zooms in on a single Flux instance that schedules and executes ConveyorLC jobs for Docking tasks. We assume a task is generated by GMD SC, which is sent to and enqueued into RabbitMQ before it is dequeued by a GMD SC adapter (e_1) that is responsible for adapting the tasks for next steps. Because the GMD SC adapter does not directly interface with Flux but via a Maestro instance, the task has to be adapted to Maestro first via the invocation of Maestro's submit API (e_2). Maestro then adapts the task to Flux via the invocation of Flux's submit API (e_3). Once the task is submitted to Flux, Flux must go through critical events: its Fluxion scheduler first finds and allocates a set of resources for this task (e_4) and then its execution service must launch and bootstrap the ConveyorLC job (e_5).

The general case is when all of the compute node resources managed by Flux have been utilized and a running job completes e_6. e_6 is a `job-complete` event emitted from Flux and e_7 and e_8 are a `job-complete-detected` events emitted from Maestro and then from the GMD SC adapter. e_9 indicates the GMD SC adapter sends a `job-done` message to RabbitMQ. Between e_5 and e_6, there are many critical-path events emitted from ConveyorLC as pertaining to its task-level scheduling, which we omit. For the general case, e_1 through e_9 represents the critical path of the Docking portion of the workflow. This sequence can repeat many times in particular when the size of each ConveyorLC job gets smaller. Given the relative performance characteristics of Flux and ConveyerLC scheduling, smaller jobs can significantly increase the overall performance overhead of A, B and C regions: $R \times \sum_{i=1}^{4} t(e_i, e_{i+1})$ where R denotes a repeat count and $t(e_i, e_{i+1})$ denotes a duration between two consecutive events. However, they can significantly decrease the overall

performance overhead of D region: $R \times t(e_5, e_6)$. This can provide an insight into how this relative performance varies over the variable: `Size of ConveyorLC job`.

Fusion tasks begin with a RabbitMQ message, which contains a set of molecular poses to be scored by it. The number of molecular poses in each message is controlled by the workflow domain-specific parameter called `Molecule batch size`. Each RabbitMQ message is consumed by a GMD SC adapter, which creates and runs a corresponding Maestro specification. Maestro prepares the run directories for the Fusion jobs and submits the jobs to the enclosing Flux instance running on the GPU-enabled supercomputer. Overall, the critical path of Fusion is similar to Docking. The Docking and Fusion tasks are composed together via the messages enqueued into RabbitMQ by the GMD SC adaptors. To have a Fusion task score a molecule post-Docking, the GMD SC adaptor that consumed the initial Docking task enqueues a Fusion task into the message queue after the Docking task completes.

C. Impacts of Varying Performance Variables

Eight different configurations of our workflow benchmark runs have been performed with the combinations of two versions of Flux, two settings of ConveyorLC Docking jobs, and two settings of Fusion jobs (Table I). One version of Flux is labeled as `Default` (flux-core v0.26 and fluxion v0.15) and the other as `Opt` or `Optimized` (flux-core v0.27 and fluxion v0.16). The latter version has proven far more efficient and scalable when applied not only to our testing workloads but also a large production scientific workflow Multiscale Machine-Learned Modeling Infrastructure (MuMMI) [8, 3]. On Ruby, 981 nodes have been allocated with one node reserved to run our custom scripts that perform batch creation (using 4 threads) and batch completion (using 12 threads). The number of molecules per batch is set to 1,715. The remaining 980 nodes are divided into several Flux jobs; each job is submitted and launched with the `flux mini submit` command. One way is to divide them into four Flux jobs so each run runs across 245 nodes. The other way is to divide them into 196 Flux jobs so each run uses five nodes. For the 245-node Docking job configuration, the number of prefetch Docking jobs is set to 36. The number of prefetch Docking jobs is 72 for the five-node job configuration. We use the larger prefetch number for the five-node job since this setting runs many more jobs simultaneously than the 245-node job configuration. For the Fusion calculations on Lassen, we use two different configurations: one with resource under-provisioning and the other with over-provisioning. Our over-provisioning setting allocates 220 Lassen nodes, which runs 880 Fusion jobs simultaneously, each job running on one of the four GPUs of each Lassen node. With resource under-provisioning, we only allocate 140 Lassen nodes and run 560 Fusion jobs simultaneously.

All eight benchmark runs use the same inputs of 2 million docking calculations, which are divided into 1200 batches. Some Fusion calculation batches crashed due to bad docking structure inputs. The crashes are unpredictable due to randomness in the docking and Fusion calculations. Up to 9 batches failed per benchmark run, so we calculate makespan for each benchmark run based on the starting timestamp of the first batch and the ending timestamp of the 10^{th}-to-last batch, truncating the timestamps for the maximum number of failed batches equally for all benchmark runs. The makespan time per molecule is calculated from the adjusted makespan divided by the adjusted number of the molecules as shown in Table I. One of the most significant differences in performance is the choice of the number of nodes given to each individual ConveyorLC Docking job. The makespan time per molecule for the five-node Docking jobs ranges from 3.95 to 4.90 ms while that for the 245-node Docking jobs ranges from 7.67 to 9.70 ms. We observe about 1.6x to 2.4x speed-up in performance by decreasing the number of Lassen nodes used in individual Docking jobs alone from 245 to five nodes. Another factor affecting performance is the choice of the number of nodes for Fusion calculations. When the number of nodes for Fusion calculations increases by 1.6x from 140 to 220, the speed-up of 1.2x for five-node Docking jobs is more obvious than that of 245-node Docking jobs, which shows no significant change. This is due to the fact that the docking simulations using the 245-node Docking jobs are the bottleneck in the workflow in such a way that we cannot quickly produce the data for Fusion to utilize the over-provisioned nodes. Therefore, the changes in performance for 245-node Docking job configuration are marginal. To our surprise, the default version of Flux systematically performs better than the optimized one. For the five-node Docking job setting, the default version is faster than the optimized one by no more than a tenth millisecond. For the 245-node Docking job setting, the default version is faster than the optimized one by about 2 milliseconds. The version of Flux affects the 245-node docking configuration more than the five-node configuration.

TABLE I: Performance of eight different configurations

Flux Version	**Docking** Nodes/job	**Fusion** Total nodes	**Makespan** (s)	**Per Mol.** (ms)
Default	245	220	15208	7.67
Default	5	220	7841	3.95
Default	245	140	15257	7.69
Default	5	140	9717	4.90
Opt.	245	220	19049	9.60
Opt.	5	220	8041	4.05
Opt.	245	140	19236	9.70
Opt.	5	140	9666	4.87

D. Synthesis and Analysis

Section IV-C suggests that the choice of independent performance variables is critically important for end-to-end workflow performance. With a better choice for a single independent performance variable (`Size of ConveyorLC job`), the workflow achieves up to a 2.45x performance improvement over the slowest configuration in Table I. The fact that the `Default` Flux version outperformed the `Optimized` version is unexpected and suggests the non-linear nature of the

performance space. It also suggests that a suboptimal choice of another independent performance variable (`Start time of resources for Fusion`) can explain a significant underutilizaton of Lassen resources, leading to a poor speedup with Fusion resource scaling. In the best case, a 1.57x Lassen resource increase provides only a 1.24x speedup. In the worse case, the same 1.57x Lassen resource increase is unable to offer any speedup whatsoever.

The identification of dependent performance variables also significantly helps researchers narrow the search space for end-to-end workflow performance tuning. For example, a better choice of a dependent variable (`Amount of resources for Fusion` not only depends on the main independent variable (`Resource amount for Docking`), but also the other variables on which it depends: `Size of ConveyorLC job` and `Start time of resources for Fusion`. The study also suggests that the impacts of a single component optimization must be carefully examined in the overall workflow through this dependency chain.

For instance, the dependence from `Flux scheduling overhead`→`Size of ConveyorLC job` would allow researchers to reconsider the size of ConveyorLC jobs when Flux is optimized in a singleton fashion. In fact, when our study varies the job size, we find an unexpected change in the scheduling overhead curve of the optimized Flux version: it makes the overall workflow run 25% slower over the default version for the larger Docking-job size. We theorized that the additional tuning (e.g., packing multiple job requests into a single Remote Procedure Call (RPC) for resource matching) suited well for extremely high throughput, tiny job size-oriented workloads significantly increased scheduling overhead on much larger job sizes. The transitive dependence among the `Amount of resources for Fusion`, `Size of ConveyorLC job`, `Flux scheduling overhead`, and `ConveyorLC scheduling overhead` variables would also allow researchers to adjust the resource amounts allocated to Fusion, if and when a more effective scheduling optimization will end up improving the overall docking simulation throughput.

E. Using PerfFlowAspect for cross-component performance analysis

Can our PerfFlowAspect performance analysis techniques help researchers better understand the impacts of a key performance variable: `Size of ConveyorLC job`*?* Figure 5a and Figure 6a show the PerfFlowAspect traces, displayed across the global timeline view within the Perfetto tool, of Flux used to run ConvyerLC Docking jobs, either with larger or smaller job size, respectively. These Figures show that the same number of jobs at nearly 50 times larger scale makes everything run slower: a 4.19x longer makespan (14,388 seconds divided by 3,431 seconds) as seen by Flux. Nevertheless, the overall makespan performance difference is only a factor of 1.93 slower from Table I. Therefore, it suggests that more aggressive Fusion resource overprovisioning (`Amount of resources for Fusion`) and/or other Fusion optimizations can further improve this workflow.

Can our PerfFlowAspect analysis help researchers understand the impact of a single component performance overhead change? Figure 5a and Figure 5b show the traces of the top-level Flux instance of two different versions, `Default` and `Optimized`, respectively. It shows that the higher scheduling overhead associated with the optimized Flux on the larger Docking jobs puts docking simulations throughput on the critical path: a 1.27x Flux scheduling slowdown stemming from the optimized Flux almost entirely explains the overall 1.25x slowdown of the end-to-end performance, compared to the corresponding default Flux configuration. With the larger ConveyorLC Docking job configuration, the docking throughput already appears lower than the maximum Fusion throughput (even with Fusion resource under-provisioning).

V. Iteration-based tuning techniques

The architecture of a composite workflow must be flexible and highly re-configurable. Unlike the traditional monolithic workflow approach with a single scheduler of known performance characteristics, a composite workflow's performance optimization is far more complex as it entails multi-dimensional optimization across many components. The architecture must be highly re-configurable to streamline many trial-and-error-based optimization iterations. Furthermore, performance analysis and optimization of a composite workflow must be continuous. As the composite workflow starts to run and performance data are collected and analyzed, various component-wise optimization will be applied. For each iteration, the relative performance can vary, and the critical path for the workflow scheduling can change as a result as well. Not only must the workflow be highly re-configurable but also must it be re-configurable with respect to continuous changes of the critical-path events.

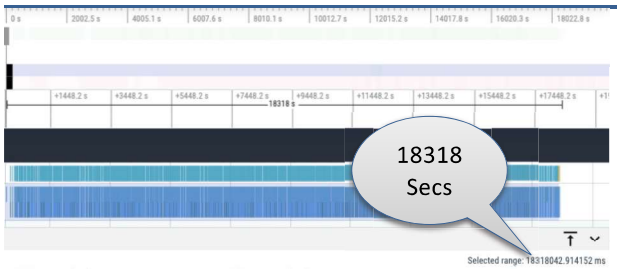
A. Re-configurable base platform and continuous optimization

In addition to domain-specific parameter control, we used the flexibility of Flux to address this challenge. Once the components are blended using Flux, researchers can dial its scheduling policies to reconfigure. A workflow can be configured in such a way that the use of nested Flux instances can hide from GMD SC the complexity and scalability required to run many ConveyorLC jobs. Flux specifically allows GMD SC to treat the group of ConveyorLC jobs as a single unit: GMD SC only needs to run one GMD SC worker to adapt ConveyorLC into its pipeline. Because Flux is a fully featured workload manager, researchers can easily reconfigure other important parameters such as the size of ConveyorLC jobs, use of hierarchical nesting of different topologies, and many other scheduling policies.

B. PerfFlowAspect applied to performance tuning

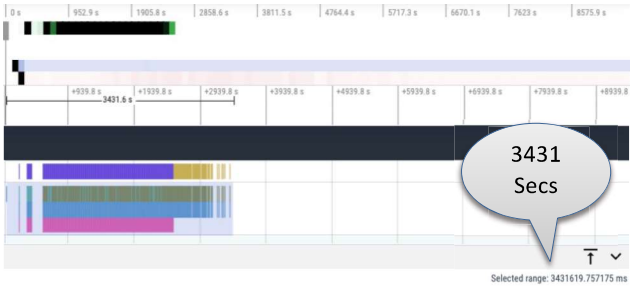
Can an PerfFlowAspect analysis inform researchers of a previously unknown area for performance tuning? Figure 6b

(a) Large ConveyorLC Docking job, large Fusion and default Flux

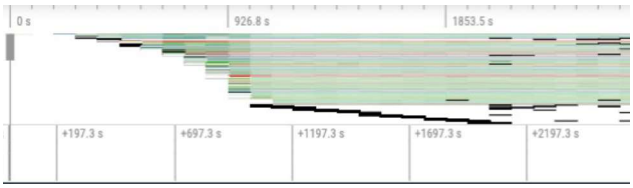


(b) Large ConveyorLC Docking job, large Fusion, optimized Flux

Fig. 5: PerfFlowAspect Traces Visualized in Perfetto



(a) Small ConveyorLC Docking job, large Fusion and default Flux



(b) Fusion traces for the best configuration

Fig. 6: PerfFlowAspect Traces Visualized in Perfetto

shows the traces of the Maestro adapter traces for Fusion. Figure 6b in combination with Figure 6a clearly suggest that there is a significant resource idling due to tight synchronization of Docking and Fusion startup and shutdown. Figure 6a is the best Docking configuration where Docking for all of the compounds completes in 3,341 seconds. Once this is done, however, there is no activity traced from within the Flux instance, indicating Ruby resources idled during the remainder of the time. As shown by the initial staggering patterns of Figure 6b, the Maestro adapter activities also appear to be ramping up slowly as not all Fusion resources are fully utilized during this ramp-up phase. Note that this was not clearly shown from the traces of Flux because from Flux's perspective Fusion jobs were already submitted and running and resources were fully utilized. This signifies the importance of casting the performance analysis concerns across *all* of the critical components as this kind of analysis would be infeasible using single-component traces alone. Overall, it hints that the more imbalanced Docking and Fusion throughput becomes, the worse will this resource idling issue be. We communicated this to the AHA MoleS workflow development team, and they plan to implement a loose synchronization scheme between Docking and Fusion.

VI. Related Work

Many composite workflow approaches have been proposed, which include MuMMI [8, 3], a massive ML-aided discovery for Inertial Confinement Fusion (ICF) energy [26], CASTELO [35], the recent winner [5] and two of the three finalists [17, 25] of the SC20 Gordon Bell Special Award competition, and VirtualFlow [14]. One finalist [32] of the SC21 Gordon Bell Special Award competition developed a multi-resolution composite workflow and ran it across multiple top supercomputers. Our focus of the general design principles, composition, performance analysis and optimization techniques differentiate ours from their focus: advancement of domain sciences. Further, composable and general-purpose workflow management tools such as ExaWorks Software Development Toolkit (SDK) [27] have begun to gain traction to enable more rapid development of this type of workflows. We proposed an approach to rationalizing overall workflow performance and its application to AHA MoleS, but our work is directly applicable to the field at large including these listed composite workflows and software tools that enable them. Similarly, myriad performance tracing and profiling tools exist, including HPCToolkit[1], TAU [28], Caliper [4], Vampir and Score/P [21]. Their primarily focus is on the single application level whereas ours is on gathering a holistic picture of workflow-level performance.

VII. Conclusion

There is a growing consensus that three major characteristics will define the next-generation of HPC centers: extreme hardware heterogeneity at all levels; 2) closer convergence of HPC with cloud computing software; and 3) complex scientific workflows that must automatically and effectively map to and run on next-generation resources across the entire center. To provide a window into the challenges that these next-

generation cross-system workflows will face, we present our multi-disciplinary effort to run a drug screen workflow across three completely different types of center resources, both traditional HPC systems and on-premises cloud software-based (OpenShift Kubernetes) system. To categorize key technical challenges systematically, we direct our effort towards an end-to-end benchmark demonstration of the resulting workflow at massive cross-system scales. We solve each key challenge in succession.

First, while creating a cross-system composite workflow using scalable and portable software components is an important first step, gaining a deep understanding of the performance space of the composite workflow is a necessity. We provide performance insight by introducing the concept of performance variables that capture the interplay of different software components as a result of their relative performance. Second, we find that instrumenting composite workflows and analyzing their end-to-end performance is an unmet challenge which we overcome with PerfFlowAspect. PerfFlowAspect can cast a cross-cutting performance-analysis concern or aspect across a heterogeneous set of resources used in the workflow. Finally, we identify the difficulties of composite science workflow performance tuning and we solve these by building support for iteration-based optimization exploration directly into the workflow architecture itself.

Our evaluation suggests that our solutions can significantly enhance the capacity of a multi-disciplinary team to create, analyze and optimize a high-performance composite workflow. Our experiments show that a better choice for an independent performance variable alone can provide AHA MoleS up to a 2.45x performance improvement. Further, the identification of dependent performance variables significantly helps narrow the search space of end-to-end workflow performance tuning. Our case studies demonstrate that our AOP-based techniques allow researchers to gain insight into principal performance features over these variables. Our work enables AHA MoleS to run in production more efficiently while reducing necessary human intervention in simulation campaigns. Using the AHA MoleS workflow, a single subject matter expert can now handle the simulation campaigns highly efficiently. Overall, our work lights a path for scientific workflows to maximize the use of next-generation compute center resources.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-831502).

References

[1] Laksono Adhianto et al. “HPCToolkit: Tools for performance analysis of optimized parallel programs”. In: *Concurrency and Computation: Practice and Experience* 22.6 (2010), pp. 685–701.

[2] Dong H. Ahn et al. “Flux: Overcoming scheduling challenges for exascale workflows”. In: *Future Generation Computer Systems* 110 (2020), pp. 202–213. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2020.04.006.

[3] Harsh Bhatia et al. “Generalizable Coordination of Large Multiscale Workflow: Challenges and Learnings at Scale”. In: *Proceedings of Supercomputing ’21: The International Conference for High Performance Computing*. SC ’21. 2021. DOI: 10.1145/3458817.3476210. URL: https://doi.org/10.1145/3458817.3476210.

[4] David Boehme et al. “Caliper: Performance Introspection for HPC Software Stacks”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’16. Salt Lake City, Utah: IEEE Press, 2016. ISBN: 9781467388153. DOI: 10.5555/3014904.3014967. URL: https://doi.org/10.5555/3014904.3014967.

[5] Lorenzo Casalino et al. “AI-driven multiscale simulations illuminate mechanisms of SARS-CoV-2 spike dynamics”. In: *The International Journal of High Performance Computing Applications* 35.5 (2021), pp. 432–451.

[6] National Energy Research Scientific Computing Center. *SPIN*. https://www.nersc.gov/systems/spin/. Retrieved July 21, 2021. National Energy Research Scientific Computing Center.

[7] Francesco Di Natale. *Maestro Workflow Conductor*. https://github.com/LLNL/maestrowf. Mar. 2017.

[8] Francesco Di Natale et al. “A massively parallel infrastructure for adaptive multiscale simulations: modeling RAS initiation pathway for cancer”. In: *Proceedings of Supercomputing ’19: The International Conference for High Performance Computing*. SC ’19. 2019. DOI: 10.1145/1122445.1122456. URL: https://doi.org/10.1145/3295500.3356197.

[9] Evan N. Feinberg et al. “PotentialNet for Molecular Property Prediction”. In: *ACS Central Science* 4.11 (2018), pp. 1520–1530. DOI: 10.1021/acscentsci.8b00507.

[10] Flux Framework Community. *Flux Framework: A flexible framework for resource management customized for your HPC site*. http://flux-framework.org. Retrieved June 20, 2021.

[11] Flux Framework Community. *PerfFlowAspect: a tool to analyze cross-cutting performance concerns of composite scientific workflows*. https://github.com/flux-framework/PerfFlowAspect. Retrieved May 31, 2022.

[12] David Fox. *Enabling Workflows in Livermore Computing*. https://hpc.llnl.gov/sites/default/files/Enabling-Workflows-in-Livermore-Computing-LC-User-Meeting-Dec-2020%20Final.pdf. Retrieved July 21, 2021. Lawrence Livermore National Laboratory.

[13] Gartner, Inc. *Gartner forecasts worldwide public cloud end-user spending to grow 23% in 2021*. https://www.gartner.com/en/newsroom/press-releases/2021-04-21-

gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-23-percent-in-2021. Retrieved June 20, 2021.

[14] Christoph Gorgulla et al. "An open-source drug discovery platform enables ultra-large virtual screens". In: *Nature* 580.7805 (2020), pp. 663–668.

[15] Izumi V. Hinkson, Benjamin Madej, and Eric A. Stahlberg. "Accelerating Therapeutics for Opportunities in Medicine: A Paradigm Shift in Drug Discovery". In: *Frontiers in Pharmacology* 11 (2020), p. 770. ISSN: 1663-9812. DOI: 10.3389/fphar.2020.00770. URL: https://www.frontiersin.org/article/10.3389/fphar.2020.00770.

[16] Hyperion Research. *How cloud computing is changing HPC spending*. https://hyperionresearch.com/wp-content/uploads/2021/01/Hyperion-Research-Special-Analysis-Clouds-and-HPC-December-2020.pdf. Retrieved June 20, 2021.

[17] Sam Ade Jacobs et al. "Enabling rapid COVID-19 small molecule drug design through scalable deep learning of generative models". In: *The International Journal of High Performance Computing Applications* (May 2021). DOI: 10.1177/10943420211010930. eprint: https://doi.org/10.1177/10943420211010930. URL: https://doi.org/10.1177/10943420211010930.

[18] Michael James et al. "ISPD 2020 Physical Mapping of Neural Networks on a Wafer-Scale Deep Learning Accelerator". In: *Proceedings of the 2020 International Symposium on Physical Design*. ISPD '20. Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 145–149. ISBN: 9781450370912. DOI: 10.1145/3372780.3380846. URL: https://doi.org/10.1145/3372780.3380846.

[19] Derek Jones et al. "Improved Protein–Ligand Binding Affinity Prediction with Structure-Based Deep Fusion Inference". In: *Journal of Chemical Information and Modeling* 61.4 (2021), pp. 1583–1592.

[20] Gregor Kiczales et al. "Aspect-oriented programming". In: *ECOOP'97 — Object-Oriented Programming*. Ed. by Mehmet Akşit and Satoshi Matsuoka. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 220–242. ISBN: 978-3-540-69127-3.

[21] Andreas Knüpfer et al. "The vampir performance analysis tool-set". In: *Tools for high performance computing*. Springer, 2008, pp. 139–155.

[22] Kubernetes. *Production-Grade Container Orchestration*. https://kubernetes.io. Retrieved July 21, 2021.

[23] Yujia Li et al. "Gated Graph Sequence Neural Networks". In: *arXiv:1511.05493* (2017). arXiv: 1511.05493 [cs.LG].

[24] Amanda J. Minnich et al. "AMPL: A Data-Driven Modeling Pipeline for Drug Discovery". In: *Journal of Chemical Information and Modeling* 60.4 (2020), pp. 1955–1968.

[25] Jonathan Ozik et al. "A population data-driven workflow for COVID-19 modeling and learning". In: *The International Journal of High Performance Computing Applications* 35.5 (2021), pp. 483–499.

[26] J. L. Peterson. *Machine Learning Aided Discovery of a New NIF Design*. Lawrence Livermore National Laboratory, Aug. 2018.

[27] Rob Farber. *Workflow Technologies Impact SC20 Gordon Bell COVID-19 Award Winner and Two of the Three Finalists*. https://www.exascaleproject.org/workflow-technologies-impact-sc20-gordon-bell-covid-19-award-winner-and-two-of-the-three-finalists. 2020.

[28] Sameer S Shende and Allen D Malony. "The TAU parallel performance system". In: *The International Journal of High Performance Computing Applications* 20.2 (2006), pp. 287–311.

[29] Garrett A. Stevenson et al. "High-Throughput Virtual Screening of Small Molecule Inhibitors for SARS-CoV-2 Protein Targets with Deep Fusion Models". In: *Proceedings of Supercomputing '21: The International Conference for High Performance Computing*. SC '21. 2021. DOI: 10.1145/3458817.3476193. URL: https://doi.org/10.1145/3458817.3476193.

[30] Neil C. Thompson and Svenja Spanuth. "The Decline of Computers as a General Purpose Technology". In: *Commun. ACM* 64.3 (Feb. 2021), pp. 64–72.

[31] Tiffany Trader. *Livermore's El Capitan Supercomputer to Debut HPE 'Rabbit' Near Node Local Storage*. https://www.hpcwire.com/2021/02/18/livermores-el-capitan-supercomputer-hpe-rabbit-storage-nodes. HPCwire, 2021.

[32] Anda Trifan et al. "Intelligent Resolution: Integrating Cryo-EM with AI-driven Multi-resolution Simulations to Observe the SARS-CoV-2 Replication-Transcription Machinery in Action". In: *bioRxiv* (2021). DOI: 10.1101/2021.10.09.463779. eprint: https://www.biorxiv.org/content/early/2021/10/12/2021.10.09.463779.full.pdf. URL: https://www.biorxiv.org/content/early/2021/10/12/2021.10.09.463779.

[33] Anthony Wood. *Rabbit MQ: For Starters*. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2016. ISBN: 1540603423.

[34] C.-Q. Yang and B.P. Miller. "Critical path analysis for the execution of parallel and distributed programs". In: *[1988] Proceedings. The 8th International Conference on Distributed*. 1988, pp. 366–373. DOI: 10.1109/DCS.1988.12538.

[35] Chih-Chieh Yang et al. "Design of AI-Enhanced Drug Lead Optimization Workflow for HPC and Cloud". In: *IEEE International Conference on Big Data*. Dec. 2020, pp. 5861–5863.

[36] Xiaohua Zhang, Horacio Perez-Sanchez, and Felice C. Lightstone. "A Comprehensive Docking and MM/GBSA Rescoring Study of Ligand Recognition Upon Binding Antithrombin". In: *Curr. Top. Med. Chem.* 17.14 (2017), pp. 1631–1639. ISSN: 1568-0266.

[37] Xiaohua Zhang, Sergio E. Wong, and Felice C. Lightstone. "Message passing interface and multithreading

hybrid for parallel molecular docking of large databases on petascale high performance computing machines". In: *Journal of Computational Chemistry* 34.11 (2013), pp. 915–927. DOI: https://doi.org/10.1002/jcc.23214.

[38] Xiaohua Zhang, Sergio E. Wong, and Felice C. Lightstone. "Toward Fully Automated High Performance Computing Drug Discovery: A Massively Parallel Virtual Screening Pipeline for Docking and Molecular Mechanics/Generalized Born Surface Area Rescoring to Improve Enrichment". In: *Journal of Chemical Information and Modeling* 54.1 (2014), pp. 324–337. DOI: 10.1021/ci4005145.