Check for updates

International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

Special Issue Paper

High-throughput virtual laboratory for drug discovery using massive datasets

The International Journal of High Performance Computing Applications
1–17
© The Author(s) 2021
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/10943420211001565
journals.sagepub.com/home/hpc
SAGE

Jens Glaser[1], Josh V Vermaas[1], David M Rogers[1], Jeff Larkin[2], Scott LeGrand[2], Swen Boehm[3], Matthew B Baker[3], Aaron Scheinberg[4], Andreas F Tillack[5], Mathialakan Thavappiragasam[6], Ada Sedova[6] and Oscar Hernandez[3]

Abstract

Time-to-solution for structure-based screening of massive chemical databases for COVID-19 drug discovery has been decreased by an order of magnitude, and a virtual laboratory has been deployed at scale on up to 27,612 GPUs on the Summit supercomputer, allowing an average molecular docking of 19,028 compounds per second. Over one billion compounds were docked to two SARS-CoV-2 protein structures with full optimization of ligand position and 20 poses per docking, each in under 24 hours. GPU acceleration and high-throughput optimizations of the docking program produced 350× mean speedup over the CPU version (50× speedup per node). GPU acceleration of both feature calculation for machine-learning based scoring and distributed database queries reduced processing of the 2.4 TB output by orders of magnitude. The resulting 50× speedup for the full pipeline reduces an initial 43 day runtime to 21 hours per protein for providing high-scoring compounds to experimental collaborators for validation assays.

Keywords

High-throughput virtual screening, drug discovery, GPU acceleration, high-performance database query

1. Justification for prize

Record time for molecular docking 1.37 billion compounds from the Enamine database to SARS-CoV-2 proteins and for machine-learning based rescoring: over 50× time to solution speedup over current CPU-based methods. A further 10× speedup for analyzing the massive output dataset by parallel database methods on GPUs to complete the screening.

2. Performance attributes

Performance attribute	Our submission
Category of achievement	Time to solution, scalability
Type of method used	Numerical optimization
Results reported for	Whole application including I/O
Precision reported	Single precision
System scale	Measured on full-scale system (Summit)
Measurement mechanism	Timers

3. Overview of the problem

SARS-CoV-2, the virus responsible for the development of COVID-19, attacks its human host through the actions of its viral proteins. Many drug discovery efforts against viruses aim to interfere with viral protein function. For HIV, antiretrovirals targeting essential viral reverse transcriptase and protease enzymes substantially improve life expectancy (The Antiretroviral Therapy Cohort Collaboration, 2008),

[1] National Center for Computational Sciences, Oak Ridge National Laboratory, Oak Ridge, TN, USA
[2] NVIDIA Corporation, Santa Clara, CA, USA
[3] Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA
[4] Jubilee Development, Cambridge, MA, USA
[5] Scripps Research, San Diego, CA, USA
[6] Biosciences Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA

Corresponding author:
Jens Glaser, National Center for Computational Sciences, Oak Ridge National Laboratory, Oak Ridge, TN 37831-2008, USA.
Email: glaserj@ornl.gov

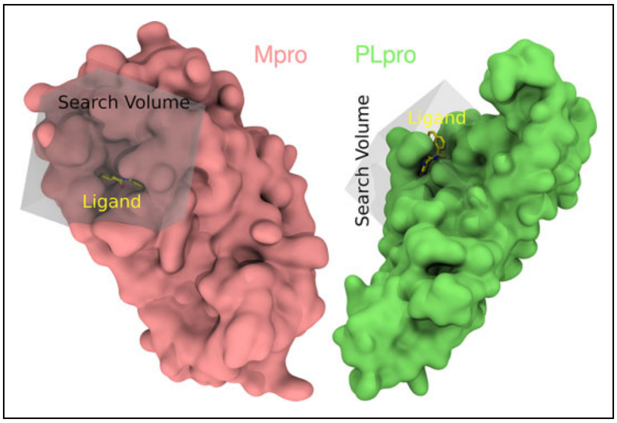


Figure 1. Protein-ligand examples from our results, highlighting the binding cleft of the protein (pink or green surfaces for Mpro and PLpro, respectively) used to define the search volume (transparent gray cube) where docking programs search to find good docked poses for the ligand (colored sticks, where carbons are yellow, oxygens are red, and nitrogens are blue) within the protein context.

and the multidrug cocktail Harvoni that targets the NS5A and NS5B proteins in Hepatitis C effectively cures the disease (Gritsenko and Hughes, 2015). These solutions took many years to reach the patient, in part due to the large chemical search space where potential drugs may be found. There are estimated to be at least 10^{60} compounds in the small-molecule universe that pharmaceutical companies mine for potential drug candidates (Virshup et al., 2013). Verifying activity for a significant fraction of chemical space against a specific viral protein target in a traditional benchtop setting is intractable. Sorting through compound sets *in silico* to identify ligands that may interact strongly with viral proteins is therefore imperative to reducing the *in vitro* search space within a larger drug discovery pipeline (Parks and Smith, 2020). Structure-based computational screening of small molecules (ligands) use simulations of their ability to bind to a protein, known as docking. Docking calculations aim to find a set of ligands that interfere with protein function by rapidly evaluating the most favorable pose that a small molecule will adopt within a protein binding pocket, using a three-dimensional atomic model of both the protein and the ligand to evaluate the interactions between the two (Figure 1). Molecules can change shape by rotating chemical functional groups, and thus even for a single compound the number of possible atomic configurations is unbounded.

3.1. Physical model

3.1.1. Optimizing ligand pose. Through optimization algorithms over simulated potential energy surfaces, different atomic configurations are evaluated within the protein environment. Favorable atomic poses and their energies, or scores, are returned for sorting during output processing and for possible further refinement in subsequent screening iterations. These optimization algorithms vary from iterative gradient-based methods to genetic algorithms and hybrid methods. The potentials which define the optimization space may be physics-based, or partially statistical. We focus on the Lamarckian Genetic Algorithm (LGA) implemented in AutoDock for optimizing docking poses (Norgan et al., 2011). Optimizing the position of a single ligand within a protein's binding pocket is fast. However, when faced with billions of possible ligands, docking the small molecules to specific protein targets and interpreting the output rapidly is a significant challenge only feasible in a high-performance computing environment. For rapid responses to new pathogens such as SARS-CoV-2, it is desirable to target a number of different viral proteins and then select compound lists based on which molecules are predicted to be exceptional binders, pruning the number of candidates to a manageable number for *in vitro* assays to validate experimentally.

3.1.2. Machine learning for accurate pose ranking and data analytics. The chemical complexity of a binding event is such that the most accurate simulation approaches for estimating small molecule binding affinity to a protein would require days for a single binding calculation. Molecular docking is a low-resolution method that finds possible poses for the ligands rapidly, at the expense of accuracy. To ameliorate limitations of the low-resolution model that high-throughput docking requires, machine-learning based methods to re-rank the most favorable poses generated by the docking calculation have become a popular second step in this type of screening technique (Adeshina et al., 2020; Ballester, 2020; Gorgulla et al., 2020; Shen et al., 2020). These models, trained on experimental datasets, take as input some key features of the final poses generated by the docking calculations and output a re-ordering of the estimated ligand binding strengths to provide a more accurate set of the most likely binders. The total computational cost for both steps remains much less than a full-accuracy calculation.

3.2. Computational challenges

The overarching computational challenge is to search the vast chemical space rapidly with a drug discovery workflow suitable for today's HPC environments, balancing accuracy with efficiency gained by lower-resolution, high-throughput methods. In particular, the ability to utilize the entire Summit supercomputer in this effort provides a valuable computational resource in the urgent effort to discover therapeutics for COVID-19. With this ability comes the need for fault-tolerance strategies that can handle hardware failures and maintaining a persistent workflow state for restart. In modern HPC settings, maximizing GPU utilization within the molecular docking process is essential. Many docking codes operate on a single protein-ligand pair at a time, which result in significant setup and finalization times where the GPU is idle. Furthermore, the docking procedure has many levels of internal parallelism that can

Table 1. State of the art docking codes, sorted by citations as counted by Google Scholar on September 24, 2020.

Program	Reference	License	GPU	2020 Citations
AutoDock Vina	Trott and Olson (2010)	Apache		2330
AutoDock4	Morris et al. (2009)	GNU		1670
GLIDE	Friesner et al. (2004)	Commercial		569
DOCK6	Allen et al. (2015)	Academic		59
rDock	Ruiz-Carmona et al. (2014)	GNU		53
FRED	McGann (2011)	Commercial		43
DOCK3	Coleman et al. (2013)	Academic		17
PLANTS	Korb et al. (2006)	Academic	✓ Korb et al. (2011)	16
QuickVina-W	Hassan et al. (2017)	Apache		15
QuickVina 2	Alhossary et al. (2015)	Apache		12
BUDE	McIntosh-Smith et al. (2015)	Unavailable	✓	8
GeauxDock	Fang et al. (2016)	Academic	✓	5
AutoDock-GPU	Santos-Martins et al. (2019b)	GNU	✓	4
GOLD		Commercial		
LeDock		Commercial		
MOE-dock		Commercial		

be potentially exploited by GPUs. At the highest level, individual runs, or statistical samples with different starting conditions, are independent, and low-level parallelism can also be optimized specific to the docking algorithm used.

Chemical compound datasets such as those used in screening are generally composed of millions of individual files for each compound. Compounds can be deposited by a variety of contributors and may be subject to removal or modification, and this single-file-based database system is therefore ubiquitous in the field. Operating on the scale of a billion molecules with billions of input files is a challenge, straining filesystems and network architectures if their limitations are not taken into account when designing workflows.

Repeating large-scale screens for multiple SARS-CoV-2 protein targets drives a commensurate increase in output data scales. State-of-the-art methods employ an extensive set of analytical components known collectively as cheminformatics, which can include machine learning, to parse these data streams. Feature calculations for these methods then must be performed on each element in the input and output datasets. Addressing the efficient calculation of these features with the incorporation of GPU acceleration is essential for performance.

Finally, deriving meaning from the output at the billion-molecule (gigadocking) scale creates major new challenges, as queries to sort, merge, or histogram output data must operate on TB-scale databases that challenge standard data analytics approaches. With every application of a new machine-learned model for assessing scores of docked poses or the addition of a new chemical descriptor to each output ligand comes the need to process again a set of database queries to sort the top compounds and obtain information about the feature distribution. Again, as the campaign expands to include more of the viral proteins, the size of the dataset that must be repeatedly queried grows. Therefore, a scalable, accelerated distributed database query processing infrastructure that can make use of high-performance technology such as GPUs and InfiniBand networks becomes the crucial last component in this full-scale screening pipeline to drive down time to solution.

4. Current state of the art

4.1. Docking large compound datasets

Molecular docking has developed substantially since the earliest computational applications to drug discovery in the 1980s and 1990s (DesJarlais et al., 1988; Dempcy and Skibo, 1991; von Itzstein et al., 1993), and many different academic and commercial codes are now available (Table 1). A handful of existing docking codes have previously been ported to GPUs. Where applied, GPUs have substantially improved performance, with speedups realized for PLANTS (60×) (Korb et al., 2011), BUDE (21×) (McIntosh-Smith et al., 2015), and GeauxDock (4×) (Fang et al., 2016). However, these accelerated options have been embraced by only a minority of the docking community, perhaps due to ease of use or cost reasons, or a lack of long-term support for the code base. The AutoDock family of programs, by contrast, are extensively used today (Table 1), despite performance limitations originating from a CPU-driven design. Therefore the recently developed AutoDock-GPU program (Santos-Martins et al., 2019b) was a compelling choice for billion-ligand docking calculations, combining familiar AutoDock ligand preparation workflows (Table 1) with a well-understood potential function and run parameters, long-standing support and development by Scripps Research, and the performance offered by a GPU implementation.

The advent of GPUs can bring about a revolution in docking in both speed and scope. In particular, in order to take advantage of the full computational capacity of Summit, using all 27,648 GPUs is essential, as the 193,536 CPU cores on Summit represent only 2.25% of the

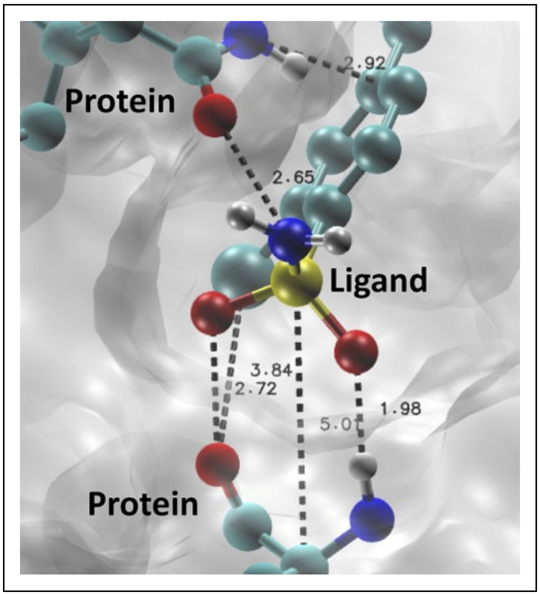


Figure 2. Examples of contacts calculated as features for machine-learning based rescoring, with distances in Å.

total FLOPS of the machine. Recent large scale docking calculations utilizing CPUs have used simpler physical models and additional computational time to dock hundreds of millions of compounds to well defined binding sites (Lyu et al., 2019), accelerating the effort to evaluate large ligand sets in a docking campaign.

A key finding from these studies is that it is critical to test the largest ligand set possible: subsampling a large ligand set by selecting representative molecular scaffolds does not pick out all effective ligands, as subtle substitutions along the scaffolds can dramatically change the docking output (Lyu et al., 2019). Nevertheless, Deep Docking (Gentile et al., 2020) techniques promise to leverage deep learning and a small set of explicit docking calculations to screen a larger set of ligands implicitly, and have been applied to the SARS-CoV-2 Mpro enzyme, one of the proteins studied here (Ton et al., 2020).

4.2. Rescoring ligand poses

Research has indicated that binding affinity rankings that emerge from the simplified physical models used during docking can be improved by applying rescoring functions based on machine learning to poses determined from docking studies (Adeshina et al., 2020; Ballester, 2020; Shen et al., 2020; Wójcikowski et al., 2017; Yasuo and Sekijima, 2019). These rescoring functions depend on calculating a set of descriptors for each ligand, including features derived from the protein-ligand pose generated through docking. Thus, rather than using docking itself as the only filter prior to experiment, rescoring techniques can improve the accuracy of the result when applied to a large fraction of the docking results. Previous methods have used CPU-based compiled executables or Python scripts, or a combination of the two. In many cases a large assortment of features is calculated (Adeshina et al., 2020) which then becomes the bottleneck in the calculation, rather than the inference using the machine-learned model itself. Especially in the case where the rescoring program is purely Python, this step in the structure-based virtual screening pipeline can become as computationally expensive as the docking portion.

4.2.1. The contacts calculation. Figure 2 illustrates one feature used in machine-learned models for rescoring, the close contacts calculation. Variants of this feature type are used as a component (Adeshina et al., 2020; Wójcikowski et al., 2017), or the only feature (Wójcikowski et al., 2017) in rescoring models. Distances between certain atoms in the ligand and those in the receptor that make contact within some cutoff are tabulated. In the RFScore-VS (version 2) model (Wójcikowski et al., 2017), one of the rescoring models applied in our pipeline, a set of concentric interaction spheres of close contacts is created and separated into bins using the C, N, O, and S receptor atoms, and C, N, O, F, P, S, Cl, Br, and I ligand atoms.

4.3. Analyzing outputs for final selection

In the past, docking campaigns were limited enough in scope that the results could be reasonably expected to fit into the main memory of a modern workstation. In recent large docking calculations, only the top thousand compounds are examined in great detail to accommodate this workflow, representing the top 0.001% of scored molecules (Lyu et al., 2019); however, this top portion must be sorted out of the large output dataset. When sorting, joining, and performing analyses such as histograms on terabytes of docking output data, handling the output becomes a big data problem with solutions that can be adapted from other data science domains. The state-of-the-art in CPU-based big database query processing is provided by technologies such as Apache Spark[1] and Hadoop,[2] with the use of the columnar Parquet[3] or ORC[4] file formats storing compressed data for rapid processing. Deployment of these methods for the analysis of very large docking-based screening outputs has not become standard; however, benchmarks such as TCPx-BB[5] measure general performance of a number of different analysis tasks on large datasets.

5. Innovations realized

At the outset of the pandemic, docking and analyzing a billion compounds in a single day as part of a drug discovery campaign seemed fanciful. Through adaptation of AutoDock-GPU, GPU offloading feature calculations in rescoring, and the use of GPU-accelerated database query software, we created a rapidly realized infrastructure that can take full advantage of leadership computing facilities such as the OLCF to analyze billions of potential pharmaceutical therapeutics for COVID-19. Operating at this scale acts as a filter to downselect from the billions of compounds that

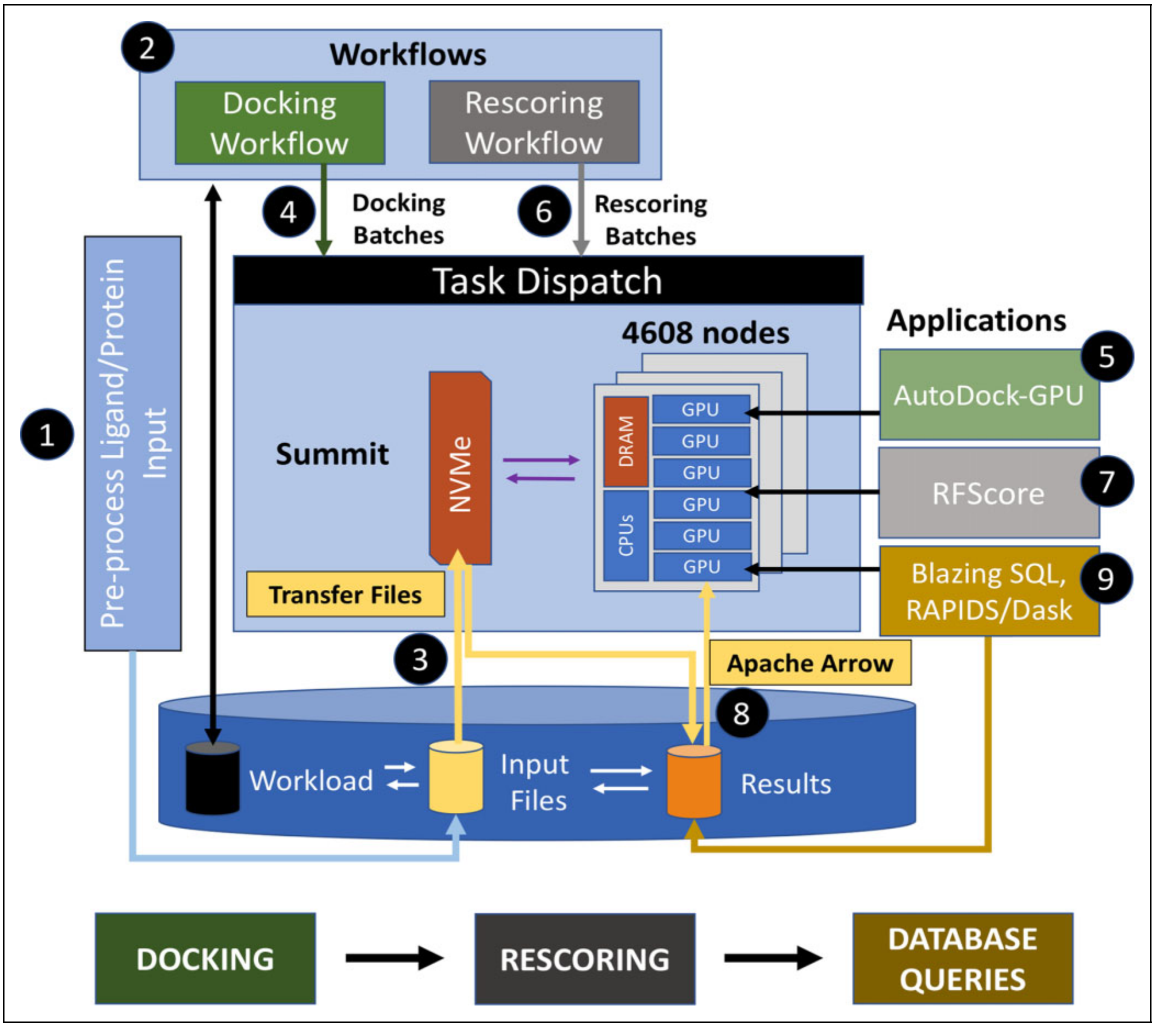


Figure 3. Architecture of the Virtual Laboratory, showing the details of the pipeline as it passes between docking, rescoring, and analytics phases. Input files are pre-processed ❶ and transferred to the filesystem on Summit where the workload ❷ is created, files are transferred to Summit's NVMe ❸, docking batches are deployed ❹, running AutoDock-GPU ❺ on the GPUs, then results are rescored using the GPU-accelerated contacts calculation ❻ ❼ and finally, outputs are converted to Apache parquet format ❽ for the database query processing analytics portion using the RAPIDS, Dask, and BlazingSQL libraries ❾.

chemical synthesis companies can produce to those with the greatest promise of efficacy. The filter reduces the workload on high-throughput experimental screens that can only test thousands of compounds at a time, and may require weeks of leadtime before these screens can be performed. Our contribution has allowed a time-to-solution of under 24 hours for the structure-based virtual screening of 1.37 billion compounds from the Enamine REAL database[6] against each viral protein structure, including docking, rescoring, and post-processing the terabytes of output required to reduce the calculation to a final solution.

5.1. Interactive queries using the virtual lab

In this project, we are combining Summit and other OLCF facilities into a larger "virtual laboratory" for drug discovery (Figure 3). By integrating facilities in this manner, we provide a unified user experience for performing large-scale docking calculations and analyzing their results. Real-time analysis is driven by Jupyter notebooks, and offers the ability to check on remote running analysis tasks or scientific queries, thus offering interactive HPC capabilities to internal and external collaborators and improving researcher productivity.

5.2. High-throughput optimizations for GPU docking of massive compound datasets

AutoDock-GPU was originally designed to dock a single ligand to a single receptor, specified as command-line inputs, and using OpenCL to utilize available GPU resources (Santos-Martins et al., 2019a, 2019b). To dock many ligands to one receptor (or vice versa), the executable was run separately for each combination. For very large ligand sets, this is not only tedious, but also inefficient; therefore, several changes were made to maximize

Table 2. GPU overheads docking 100 compounds. H2D and D2H are host to device and device to host memory copies, respectively. Free is time spent in `cudaFree`, which includes 100 calls used strictly to initialize a GPU context. Reset is time spent in `cudaDeviceReset` at the end of each compound.

	Mem. Op.	No. calls	Bytes	Sec.	Speedup (×)
Orig.	H2D	15515	4470049	0.34	
	D2H	15678	990276	0.05	
	Free	2000		241.9	
	Reset	100		136.6	
	Total	33293	5460325	378.9	
Optim.	H2D	15203	1199360	0.07	**4.78**
	D2H	15649	988368	0.05	**1.03**
	Free	20		3.04	**79.60**
	Reset	0		0	(136.6 s)
	Total	30872	2187729	3.16	**119.88**

performance of multiple independent docking calculations. In addition, OpenCL is not supported on the Summit supercomputer as drivers are not available for the POWER9 architecture. Therefore, to be able to make use of the computational power of Summit, a different programming model for AutoDock-GPU was required.

After a translation to both CUDA (LeGrand et al., 2020) and Kokkos[7] to allow for testing on Summit, we enabled the program to perform multiple docking calculations sequentially when provided with a list of protein and ligand files by the user. To optimize the performance for high-throughput docking of thousands of ligands to a single receptor on the GPU, we enabled the program to store and reuse the 16 receptor maps containing the interaction grid information for atoms in ligands instead of re-reading the same map files for each new ligand and re-transferring this data to the GPU repeatedly. Once on the large pool of global memory available on Summit's V100 GPUs, this receptor information is not evicted until all docking calculations specified in the file list have been completed. In addition, we reused CUDA contexts between sequential dockings. CUDA context creation proved to be the biggest bottleneck in repeated dockings of ligands to the same receptor. These two optimizations effectively amortize large startup costs over many docking calculations and substantially improve the aggregate runtime to dock sets of ligands (Table 2), with data structure reuse accounting for a 5× performance improvement beyond the prior state of the art on the GPU (Figure 4).

To improve performance further, overlap of GPU and CPU computation is achieved by utilizing multiple threads. For portability to OpenCL versions of AutoDock-GPU, rather than using streams, the overlap is orchestrated by OpenMP directives creating CPU-based threads that run in parallel to load and process ligand input files, run GPU-based docking kernels when the GPU is free, and process and write results to disk. To ensure that jobs do not conflict, all CPU-GPU communication and the docking algorithm itself, consisting of several CUDA or OpenCL kernels, are placed in an `omp critical` section. As a result, OpenMP threads perform the setup and post-processing steps of each job in parallel, while sequentially using the GPU for the docking calculation itself (Figure 4, bottom). This approach hides almost all setup and post-processing time, ensuring that the GPU is fully utilized during the docking process.

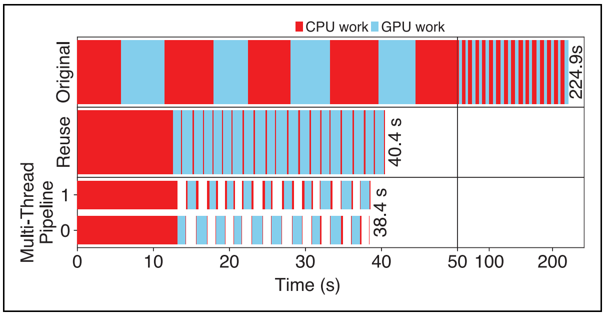


Figure 4. Execution timeline for three AutoDock-GPU configurations, running either as a set of single executables before optimizations (top), a single executable with a filelist on a single CPU thread with file and context reuse optimizations (middle), or a multi-threaded executable on a single GPU to emphasize CPU and GPU work independence (bottom).

5.3. Gigadocking workflow optimization

In order to scale out the large set of single-GPU calculations to parallelize over available resources, some form of workflow management system is required to orchestrate docking tasks. For our two gigadocking calculations to Mpro on Summit, which differ only in the protein structure used as input, we experimented with different workflow models. We performed gigadocking calculations with both FireWorks (Jain et al., 2015) and a simple Redis database feeding workers with tasks. Each task was composed of several thousand ligand docking calculation inputs for docking against a single Mpro receptor structure.

The task structure aims to optimize for I/O performance. Placing all ligand input files in a single directory is impractically slow, and so the ligand set was divided among a set of different directories that each comprised a task. The tarred directory was unpacked on the local NVMe burst buffer of each Summit node (Figure 3), sparing the metadata server on the shared GPFS filesystem (Schmuck and Haskin, 2002) from becoming overburdened. After processing the ligands within the directory, the outputs were repackaged for further analysis.

5.3.1. Fireworks and the slate resource at OLCF.

The FireWorks workflow management software (Jain et al., 2015) was deployed via an OpenShift computing cluster at the OLCF called Slate. Slate provides container orchestration for users and consists of two user-facing clusters, with Marble being the cluster that interfaces with Summit.

Marble is used to deploy workflows on Summit that are controlled externally, providing a persistent state for the workflow in case of Summit failure and allowing system configurations and services that are not possible on Summit. FireWorks required libraries and configurations that proved easier to deploy on containerized resources. This solution also allows workflows to be run simultaneously on Summit and other computing facilities, creating a potential for even more computing power using collective resources across institutions. Task managers called FireWorkers are deployed on Summit's compute nodes, and contact the MongoDB on Marble for queuing and data storage to retrieve sets of tasks to execute on that compute node. For docking the dataset to one of the Mpro structures, 4602 compute nodes were used simultaneously on Summit, each running 6 FireWorkers per node, bringing the total number of FireWorkers to 27,612; to the best of our knowledge this is currently the largest-scale FireWorks deployment.

5.4. Rescoring optimization by GPU offloading

Rescoring procedures depend on input descriptors calculated from docking outputs (Section 4.2). In evaluating performance, we realized that calculating the close contacts between protein and ligand represented a large fraction of the total rescoring runtime. Thus, we reimplemented the close contact calculation used in RFScore-VS (Wójcikowski et al., 2017) with a custom CUDA kernel designed to calculate a contact histogram in shared memory.[8] The kernel uses the atomic position and atom types to compute the number of contacts per atom type pair (9 per ligand, 4 per protein) according to 6 spatial bins, for a total descriptor length of 216 integer numbers. To optimize throughput, the kernel operates on a list of ligands to stream through the GPU. Furthermore, parsing the human-readable input coordinate files in PDBQT format was a significant bottleneck. Using a `cudf` based parser in Python that operates over all entries of all PDBQT inputs per partition of $\sim 50,000$ in parallel, we read in the input elements directly from disk in parallel. These elements can be passed directly into the contact kernel, implemented as a `RawKernel()` in CuPy,[9] to realize substantial acceleration in rescoring when compared with CPU-based alternatives.

5.5. Accelerated distributed database queries on massive output datasets

Sorting and analyzing a 2.4 TB dataset with 1.37B entries is a challenge, and goes beyond the use case for conventional persistent databases implemented on CPU nodes, such as MongoDB. On the other end of the spectrum are software frameworks that support the MapReduce paradigm, such as Apache Hadoop and Spark. The nature of such distributed queries is task-based and massively parallel. Driven by the demand for faster data processing in corporate contexts, GPU-accelerated database query engines are quickly becoming a popular option due to the substantial speedups they realize over their CPU-based counterparts (Chu et al., 2020). We hypothesized that the incorporation of a recently developed GPU-based query engine based on NVIDIA RAPIDS would offer these performance improvements and enable interactive analysis of our data sets through Jupyter notebooks. RAPIDS uses GPU parallelism and leverages high-bandwidth memory to accelerate data set manipulations.

Realizing this speedup required porting BlazingSQL[10] and the NVIDIA RAPIDS framework to the POWER9 platform. BlazingSQL is a GPU-accelerated SQL query engine, with a `C++`, a Python and a Java component, that operates on a large set of structured input data. However it is not a traditional persistent relational database management system, because it does not change the files it operates on. Previous versions of RAPIDS have been supported on POWER9 by IBM through the Watson ML framework.[11] BlazingSQL is currently not a part of any vendor-provided distribution.

`BlazingSQL` utilizes the RAPIDS `cudf` library and the `dask-distributed` scheduler at the low-level to launch compute tasks on many workers. However, in contrast to purely task-based, stateless MapReduce-type workflows, BlazingSQL needs the scheduler only to initiate a persistent communication layer between its workers and for returning the query results to the user. It therefore is able to benefit from Summits' high-performance EDR InfiniBand interconnect as well as fast I/O. The current 0.15 version uses TCP as a transport, and a UCX-based version leveraging Summit's NVLINK and infiniband (IB) network is currently under testing.

To compile the BlazingSQL and `cudf` libraries and their dependencies on Summit, we submitted code changes in the form of pull requests to the Apache Arrow,[12] Google Abseil,[13] and `cudf`[14] libraries to enable initial support for POWER9 through their open-source repositories. Adapting these applications to HPC to minimize the time to solution of a scientific problem demonstrates a step toward the convergence of big data and HPC workflows.

6. Performance measurement

6.1. Systems used

Billion-ligand scale docking calculations for two different Mpro structures were performed on the ORNL IBM AC922 Summit system. The Summit system has 4,608 compute nodes, each with two 22-core IBM POWER9 processors and six Nvidia Volta V100 GPUs. The GPUs are connected to the CPUs by NVLINK-2, with peak performance of 150 GB/sec in each direction, and between each other. Each V100 GPU has approximately 15.7 TFlops of single precision performance fed by 900 GB/s peak memory bandwidth to 16 GB HBM2 memory. The Summit compute nodes are equipped with a 1.6 TB NVMe burst buffer device and 512 GB main memory. For database applications, we used 54 high-memory nodes on Summit, where the V100 GPUs

Table 3. Version numbers of essential codes used in this research as installed on Summit and Marble.

Application	Compiler/Library	Version
AutoDock-GPU*	GCC	6.4.0
	CUDA	10.1.243
FireWorks		1.9.5
	MongoDB	4.2.8
Redis		6.0.3
JupyterHub		2.2.6
	BlazingSQL	0.15

*https://github.com/scottlegrand/AutoDock-GPU/tree/relicensing.

have 32 GB HBM2 memory, and are paired with 2 TB of RAM.

The FireWorks database infrastructure for full system runs on Summit, as well as the JupyterHub interface to the docking results, was deployed via Slate at the OLCF. Slate provides a mechanism for database or containerized applications to interact with other OLCF resources from two user-facing clusters, with the Marble cluster interfacing with Summit. Marble itself is a heterogeneous cluster of 30 nodes with 10 Gigabit Ethernet connectivity. The backend for FireWorks was 9 MongoDB instances deployed on Marble in a ReplicaSet, each allocated 28 CPU and 32 GB of memory. Libraries and applications used on Summit and Marble are provided in Table 3.

6.2. Applications benchmarked

6.2.1. High-throughput optimized AutoDock-GPU. As part of a larger scientific campaign to find non-covalent inhibitors for the SARS-CoV-2 proteins, we began with the Main Protease (Mpro) and Papain-like Protease (PLpro). We docked the 2018 Enamine REAL database (1.37 billion ligands) prepared by VirtualFlow (Gorgulla et al., 2020) to two different crystallographic structures for Mpro (PDBIDs 5R84 and 6WQF (Kneller et al., 2020)), with an additional crystallographic structure for PLpro (PDBID 7JIR) prepared for synthetic benchmarks (Figure 1). In each case, the inputs needed for each protein were prepared with Python scripts accompanying AutoDockTools (Morris et al., 2009). The search volume for each protein was a cube with side length 24.75 Å, with a 0.375 Å grid spacing (Figure 1). All runs were carried out with `nruns` set to 20, generating 20 independent docking solutions from independent starting configurations, and `autostop` engaged, which stops the search when predicted energy improvements converge. Other settings were left as default. We used our development fork of AutoDock-GPU with CUDA and pipelining optimizations (Table 3) to carry out the docking.

Full-scale runs on Summit: Summit carried out docking calculations of the Enamine REAL database against the two Mpro structures. The 5R84 structure, selected due to its comparatively accessible active site, used FireWorks as the workflow manager. The 6WQF Mpro structure was solved at room temperature (Kneller et al., 2020), providing a thermalized snapshot of the protein. The task list for 6WQF was managed by a Redis database. Tasks for each GPU were dequeued from the database using a Python script. Each gigadocking calculation utilized all 4602 nodes of Summit for under 24 hours, and the aggregate computational cost is measured through observed walltimes reported by the queuing system.

Synthetic Benchmarks: In order to facilitate runtime comparisons between AutoDock-GPU and the original AutoDock4 single-threaded CPU implementation, the ligands in the Enamine diversity set were docked to both PLpro and Mpro active sites (Figure 1) using both codes on Summit, with the same docking regions used for the full-scale runs. The Enamine diversity set consists of 10,561 representative ligands of the larger Enamine database, and is tractable to compute with the slower AutoDock4 implementation. Runtime settings were left as default, except for the `nruns` parameter, which was set to 20 for consistency with the gigadocking runs performed on Summit. Crucially, the `autostop` parameter was not set for AutoDock-GPU, as AutoDock lacks a comparable feature, and so the AutoDock-GPU per ligand timings are slightly longer on average than they would be in the production runs. Based on the variability of runtimes observed in the synthetic benchmark, we do not attempt to estimate performance in FLOPs, as different ligands create very different performance profiles.

6.3. Rescoring benchmark

For the rescoring benchmark, we focus on the RFScore-VS (Wójcikowski et al., 2017) family of machine-learning based scoring functions, for which the Python `oddt` package[15] (Wójcikowski et al., 2015) provides a reference CPU implementation. The scoring function relies on descriptors characterizing the set of interacting atoms between ligand and receptor. While the scoring function, as implemented in `oddt`, combines the descriptor calculation and inference into a single operation, they can be separated, and the pair distance computation takes over 95% of the runtime. The inference part of the calculation implements a random-forest model in `scikit-learn` (Pedregosa et al., 2011), which is optimized for batch-processing on the CPU. Here we benchmark the contact calculation only, using our custom CUDA kernel,[16] and compare against the baseline implementation.

An important optimization consists in leveraging the full parallel throughput of the GPU by parsing input text files on the device. We achieve this by providing the inputs as string columns in a `cudf` dataframe. Using string splitting, tokenization and regular expressions in parallel, we extract the coordinates into a device buffer. The kernel then operates on this buffer in a batched fashion. The reuse of a single GPU allocation for an entire batch of

$\sim 50,000$ ligands amounts to a significant improvement in runtime.

6.4. Database query processing

The deployment on Summit uses BlazingSQL version 0.15, `cudf` version 0.15, and dask version 2.25.0. We measure the timings for typical counting, sorting, histogram, and inner join queries on our docking data set using BlazingSQL on GPUs. Some of these queries are more difficult than others, because they involve string columns of several kB width holding the docked conformations. Support for string columns in parallel query processing on GPUs is a novel capability. As the deployment of big data analytics tools on Summit is also new, and in the absence of a scalable CPU-based solution on this platform, we use an additional, synthetic benchmark to establish improvements in the time to solution over state-of-the-art, albeit non-HPC, platforms. Therefore, we benchmarked BlazingSQL on an unofficial GPU port of the TPCX-BB benchmark,[17] a standard set of queries that has been adapted to RAPIDS.[18] This benchmark runs on dedicated CPU clusters, and the possibility of a 19.5× speedup of the RAPIDS version on 128 A100 GPUs over the official top contender, a 19 node cluster with two Intel(R) Xeon(R) Gold 6244 CPUs, 38 cores and 384 GB RAM per node running the Hortonworks Data Platform 3.0, had been previously demonstrated for a dataset of 10 TB size (scale factor 10,000).[19] The total reported CPU execution time is 4.7 h for 30 queries. Here, instead of a comparison within the same node with and without GPUs, and only for the purpose of reporting GPU-CPU speedup, we compare the BlazingSQL time against the Hortonworks time for the same set of queries and input data, assuming that the performance difference thus measured is reasonably representative of the performance difference that we would have obtained had we executed our docking dataset queries on a CPU cluster.

The input data for the queries resides in parquet files on the high-performance file system (GPFS) in separate directories, comprising of two sets, one in which only the top conformation per ligand are retained (1.3 TB), and one that holds all `nruns=20` docking results per ligand (25 TB). The sets are split into partitions of 5,000 and 25,000 Parquet files, respectively, per target.

7. Performance results

Table 4 shows a summary of the total time required to run each of the components of the pipeline on a single viral protein, using the full Enamine REAL database, emphasizing the aggregate improvement in time to solution.

7.1. Docking workflow

The aggregate computational cost for each gigadocking calculation was comparable between the two runs with different scheduling schemes (Figure 5), because they shared parameters that most influence the docking cost, `nruns` and the search volume size. The observed runtime difference in Figure 5 reflects both variations in the geometry of the protein's binding site as well as initial challenges with thread and connection limits using the FireWorks strategy. Thus, Figure 5 constitutes an upper bound on the cost of a gigadocking campaign once scaling limits for managing these workflows are overcome.

Managing thousands of simultaneous independent runs to use all 27,648 GPUs on Summit is a task unto itself. We used two methods for dispatching individual compute tasks to all GPUs, one based on FireWorks, and one from a simple Redis[20] database, and thus can compare the relative overhead between the two methods (Table 5). Launch overheads for preparing the batches (dequeuing from the database), as well as file movement onto and off of the local node filesystems, are minimal when compared against typical batch runtimes for a set of docking calculations. Comparison requires noting a few differences in the methods. Redis batch sizes were larger, reducing slightly the impact

Table 4. Time to solution (TTS) acceleration summary for GPU implementations of the docking pipeline on Summit using up to 27,612 GPUs, per viral protein target.

Step	CPU TTS	Nodes	GPU TTS	Speedup
1.37B Docking	41.67 days	4,602	20 hours	50×
Rescoring	1.35 days	72	0.32 hours	100×
Database Queries	1.15 hours	27	7.5 min	10×
Total	43.07 days		21 hours	**50.5**×

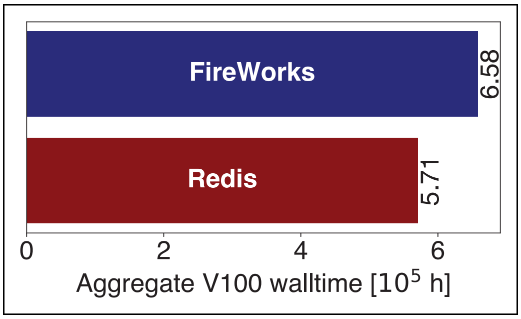


Figure 5. Aggregate GPU-hours spent on up to 27,612 V100 GPUs of Summit for conducting the 1.37B docking calculations with AutoDock-GPU, for two different Mpro structures. The GPU-hours reported here include unproductive utilization due to node failures and fizzled FireWorkers.

Table 5. Mean Task Dispatch Time (seconds) on Summit. FireWorks and Redis average batch sizes were 500 and 1,030, respectively.

Phase	FireWorks	Redis
Dequeue	0.93 ± 4.97	0.17 ± 0.03
Copy In	1.88 ± 7.87	0.99 ± 0.61
Docking	632 ± 142	1490 ± 295
Copy Out	0.81 ± 1.81	3.83 ± 0.03

of overhead but increasing file-staging times. FireWorks was able to overlap docking and file-staging steps. Increased variability in execution times for different phases was observed for FireWorks when compared with Redis. Both setups required making thousands of simultaneous outgoing connections to the database, especially at startup, which caused stability problems with FireWorks's underlying MongoDB and the nameserver. Our network setup, which placed Redis closer to the compute nodes, clearly improved the situation. In addition, the FireWorks deployment happened first, and thus provided lessons in the form of engineering workarounds that were incorporated into the subsequent Redis-based workflow.

Although the Redis-based solution turned out to be more robust and also more efficient in production use, we hesitate to say that one workflow solution is, in general, preferable to the other for launching high-throughput workflows at scale on leadership systems. Because on Summit, FireWorks has to be deployed using an external cluster via containers, there are many more components that must be tuned and an increased chance for failures. Potential benefits of FireWorks over the Redis-based solution include its graphical interface to monitor the workflow and plot statistics, the ability to maintain the state of the workflow even if all of Summit were to fail (as it is launched externally), and an API that can be used to program more complex workflow tasks. Ongoing research in workflows at the OLCF promises improvements for this solution.

7.2. Per-node performance

GPU-accelerated docking with AutoDock-GPU is cheaper considering hardware and electricity costs than comparable CPU approaches because it is significantly faster. The aggregate runtime for AutoDock4 on a representative set of ligands and receptors is ∼350 times larger than the comparable AutoDock-GPU run (Figure 6). Accounting for the 7:1 CPU-core: GPU ratio on Summit (42 usable physical cores vs. 6 GPUs), per-node performance acceleration is 50×. Each billion-ligand calculation would have taken nearly 2 months utilizing all of Summit's CPUs rather than a single day on its GPUs. The overall improvement in the aggregate runtime is driven primarily by the reduction in runtime for the fastest cases where I/O and setup constituted a larger fraction of the total runtime, which are over 600× faster for AutoDock-GPU compared with AutoDock4. By comparison, the most difficult ligands whose performance is limited by algorithmic constraints are only approximately 300× faster (Figure 6).

The speedup differential emphasizes how critical the context and receptor reuse optimizations truly were. Improving mundane setup steps by removing unneeded data movement (Table 2) by pipelining the computation delivered a nearly 4× speedup compared with previous versions of AutoDock-GPU that operate on a single ligand/receptor pair (Figure 6). Further improvement in accelerating AutoDock-GPU for difficult ligands may be possible. The average performance for AutoDock-GPU utilizes less than 10% of the peak FLOPs available on a V100 due to the considerable heterogeneity within the genetic optimization algorithm, and in particular the energy evaluation function in AutoDock-GPU.

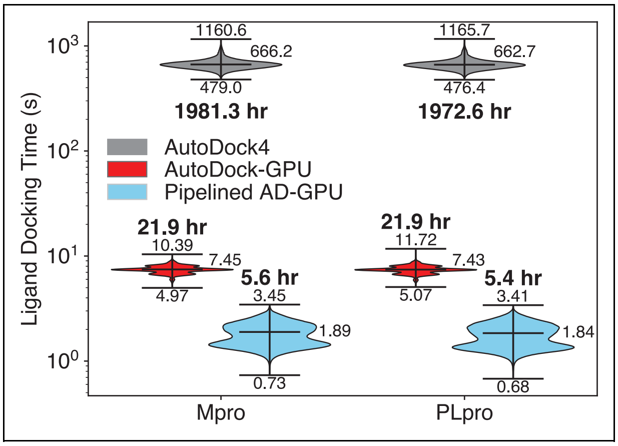


Figure 6. Runtime comparison for the CPU AutoDock4 code (gray), AutoDock-GPU without pipelining (red), and the pipelined AutoDock-GPU code (blue), running on Summit to compute optimal poses for the 10.5 k ligand Enamine diversity dataset against both PLpro and Mpro structures. The violin plots show the distribution of individual runtimes for receptor-ligand combinations, with the minimum, maximum, and median runtimes for each combination (in seconds) reported adjacent to the violin plots. The aggregate runtime for the full 10.5 k ligand set is provided in bold text, reported in core-hours or GPU-hours as appropriate.

Table 6. Contacts calculation for rescoring, showing per ligand time, per node throughput and time to solution on $N = 72$ Summit nodes. Mean time per ligand is measured by timing a single set of 5, 252, 741 and 263, 563 ligands on the GPU and the CPU, respectively.

	T_{ligand} [ms]	ligs/Ns	$T_{\text{parse},N=72}$	TTS
CPU	271.4	154.76	3.216 h	32.407h
GPU	0.387	15,494	0.314 h	0.323h
Speedup	**700**×		**10.24**×	**100.12**×

7.3. Rescoring

Following docking, rescoring must take place, which had become the productivity bottleneck given the rapidity of the docking. Rescoring, limited by the contact calculation, had a longer walltime on a modest number of CPUs than the docking calculation itself (Table 6). Contact calculations are a better fit for GPU architectures, as the pairwise distance map can make use of shared memory with fast 32 bit atomics. Thus, the 100× larger throughput compared against the original CPU-based implementation in `oddt` leverages the GPUs on Summit. We anticipate backporting this feature in the near future as a service to the community.

We benchmark the individual and per node throughput using 42 CPUs or 6 GPUs per node, respectively, measured

Table 7. Complexity of query types.

Query Type	complexity
Init	$\mathcal{O}(1)$
Count	$\mathcal{O}(N)$
Histogram	$\mathcal{O}(N)$
Sort	$\mathcal{O}(N\log N)$
Hash join (partition)	$\mathcal{O}(N+M)$
Hash join (join)	$\mathcal{O}\left[(N/N_{\text{part},N}) \times (M/N_{\text{part},M})\right]$

in ligands per second per Summit node. The speedup achieved per ligand calculation is 700×, and 100× per node. Streaming the I/O to load the PDBQT coordinates directly into the GPU substantially reduced the time T_{parse} to parse the coordinate files. This unavoidable I/O now is the remaining bottleneck after the custom GPU kernels substantially accelerate the contact calculations at the heart of the rescoring algorithm.

7.4. Database query processing

7.4.1. Docking data set. In analyzing the docking results, we have established a workflow that involves joining together the per-ligand scores over the ligand name, a 12–20 character string column used as a common index. Then we sort the scores to filter out the top 100,000 hits according to a given scoring method, compute the histogram and add structural information, i.e. the 3D conformation and the SMILES string representing the 2D chemical structure of the molecule as extra columns. We repeat these steps for a total of four different scoring methods.

Table 7 shows the complexity of different types of database queries required for completion of the analysis portion of the drug discovery pipeline. Among those, joins represent the greatest computational challenge because of the reshuffling of sub-partitions, and thus because of all-to-all communication. The workers use a heuristic to calculate the size of the sub-partitions, which are more fine-grained than the partitions used on the level of the filesystem.

Thanks to GPU-acceleration on multiple nodes, the query times are generally on the order of tens of seconds, and therefore short enough for interactive use. Table 8 summarizes the measured query times on a typical configuration of 27 high-memory nodes with six 32 GB GPUs per node. The `inner_join_conf` operation involves the ligand conformations stored as wide string columns and takes the longest time, as expected. The pure query execution time for analyzing the entire data set for a single protein target is only 7.46 minutes. The computational cost of the data analysis stage now becomes insignificant to the overall computational cost for the pipeline, and these steps can be performed repeatedly if needed. Moreover, the flexibility offered by the SQL query language allows the researcher to refine the analysis and elucidate different aspects of the data set in an interactive manner, therefore maximizing scientific productivity.

Table 8. Summary of query times, per target, on 27 nodes.

Query Name	Type	number	TTS
`create_context`	Init	1	1.61 s
`create_table`	Init	1	22.48 s
`total_count`	Count	1	5.67s
`inner_join_dcoid`	Hash join	4	35.98 s
`select_top100 k`	Sort	4	43.36 s
`hist_1d`	Histogram	4	47.64 s
`hist_2d`	Histogram	4	47.00 s
`inner_join_conf`	Hash join	4	136.71 s
`inner_join_smiles`	Hash join	4	107.39 s
Total		27	7.46m

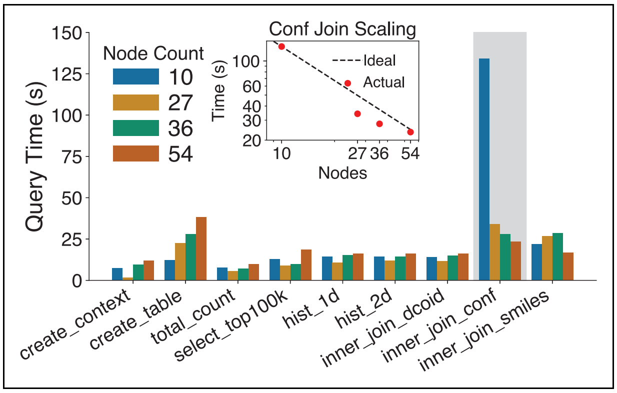


Figure 7. Runtime of GPU-accelerated ligand database queries with BlazingSQL/RAPIDS. We show the total query time for initialization statements, and eight typical queries over the entire 1.37B rows data set of Mpro (6WQF structure) as function of the number of Summit nodes, using the 32 GB high-memory GPUs. The `inner_join_conf` query (gray background) demonstrates strong scaling, as indicated by the results within the inset.

Figure 7 shows the query times as a function of the number of nodes used, with $N = 10$ being the minimum number of nodes for which the data queries successfully execute on the GPUs given memory constraints. For the most expensive query, we observe speedups consistent with linear strong scaling as we increase resources from 10 to 54 nodes (Figure 7, inset), highlighting the importance of taking full advantage of Summit's GPU resources for data processing.

7.4.2. Synthetic data set. To investigate the speed-up obtained for our deployment of BlazingSQL compared to a conventional, CPU-based cluster, we run the NVIDIA RAPIDS GPU-Big Data benchmark on 36 Summit nodes (Figure 8). This benchmark represents a typical data set for an online retail website, and the size of the data set can be scaled arbitrarily because the synthetic data is generated algorithmically. Here, we use a 10 TB dataset. The different queries are written in SQL language and exercise different capabilities of the engine on 24 different tables. Therefore, some variability in the query run times is expected and we report the aggregate time for executing 28 out of 30 queries. The walltime for 27 Summit nodes,

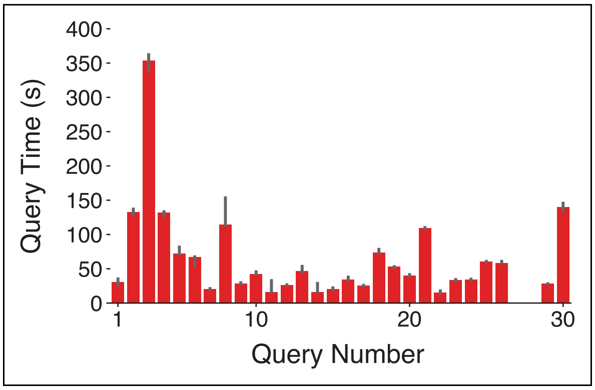


Figure 8. Median runtime of 28 representative database queries on 36 high-memory nodes (with 32 GB V100 GPUs) of Summit for a synthetic big data benchmark (derived from TPCX-BB, scale factor 10k, creating a 10 TB dataset) using BlazingSQL. Queries 27 and 28 are not supported on the POWER9 platform. The total query execution time is 30.37 min.

$T = 30.37\text{min}$ (Figure 8), represents a speed-up of 9.28× over an optimized CPU cluster. We also carried out a weak-scaling test of database performance by generating a 100 TB data set. Preliminary results (not shown) suggest that communication latencies become limiting, and we anticipate improvements in using UCX communication over Infiniband and NVLINK in upcoming versions of BlazingSQL to allow us to analyze datasets at that scale.

8. Implications

The COVID-19 pandemic has created changes in the way we do science, bringing together dedicated scientists and engineers from across disciplines to tackle challenges we had not previously foreseen. For docking calculations generally, the inherently parallel nature of the problem coupled to relatively long runtimes on previous CPU implementations had hidden inefficient I/O and startup overhead patterns for high-throughput HPC settings. Thus, optimizations to reuse data that could be implemented quickly under the pressure of the pandemic proved to be most consequential to improve docking performance and substantially reduce calculation walltimes (Table 4).

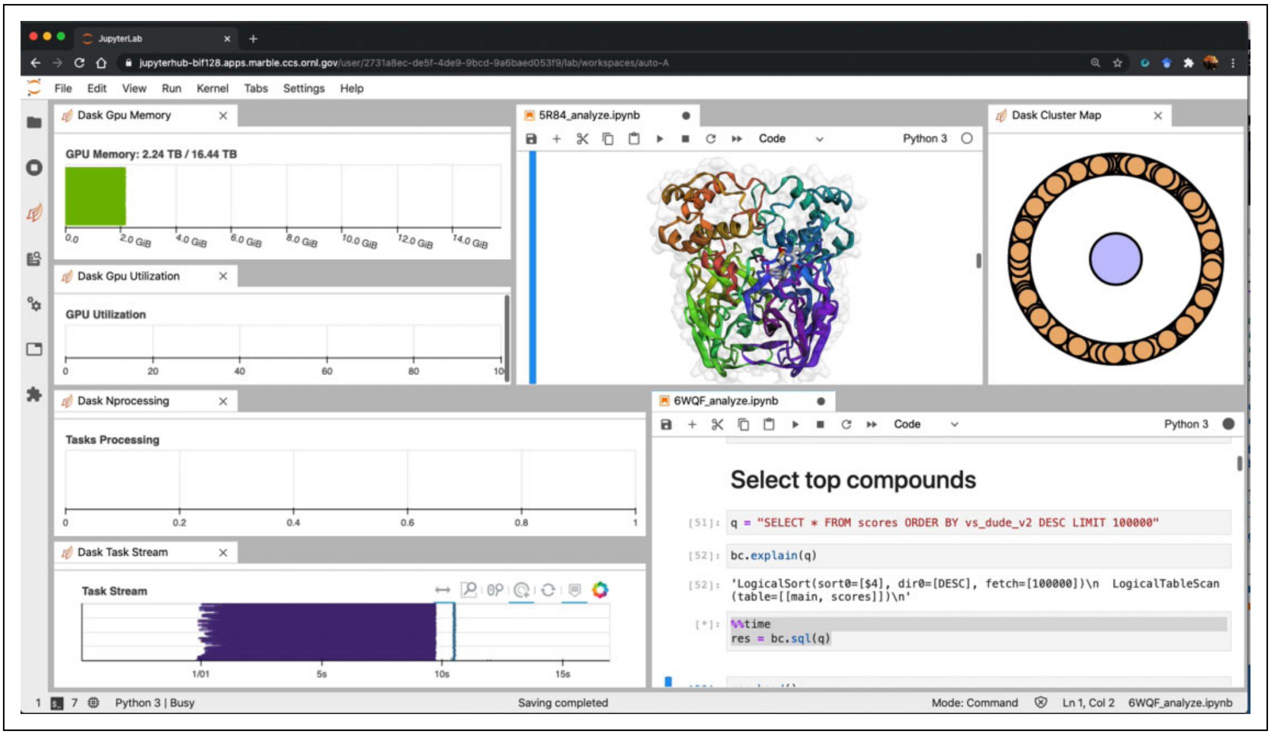
In a similar vein, use of technologies and solutions involving industrial partners to provide insight and advice was critical to accelerating other components of the overall drug discovery workflow. The resulting custom CUDA kernels and GPU-accelerated database queries powered by RAPIDS, Dask, and BlazingSQL speak to the close working relationships forged by the pandemic that were critical to bringing this drug discovery pipeline online rapidly. We strongly advocate for the community to consider what incentives outside of public health emergencies might facilitate such intensive cross-pollination of expertise to address scientific challenges.

A striking example of this synergy is how we faced rescoring and database query challenges. These steps became rate limiting as the docking calculation accelerated (Table 4), and overcoming barriers to performance in these stages of the workflow demanded consultation with domain experts to assess tools and approaches. The final solution, which uses the IB network on Summit, is able to perform distributed queries using the GPUs and shows the promise of weak scaling for queries on much larger dataset sizes. Screening against the other protein targets in the viral proteome and the inclusion of more than one final pose generated by each docking calculation will continue to grow the output dataset size. This future analysis will require additional computational infrastructure for the analysis portion of the pipeline, initiating a convergence of big data analysis and HPC for structure-based computational pharmaceutical research.

From the chemistry perspective, accelerating each step of the docking pipeline by leveraging GPU acceleration has had a stunning impact on the time it takes to screen through vast chemical databases of synthesizable compounds. The aggregate time to solution for the docking of more than a billion potential ligands against a protein target drops from months to days; reducing time to solution by more than an order of magnitude is a paradigm-shifting result for drug discovery. Furthermore, this represents a substantial cost savings, with the per ligand cost of virtual screening amounting to pennies in electricity and hardware. By contrast, the synthesis costs alone for novel compounds are hundreds of dollars each, disregarding the non-negligible costs of the activity assay itself, and can take hours or even days real-time to arrive at an answer. Naturally, computational predictions will need to be validated. However, the cost differential between the approaches will serve to tighten the loop between benchtop and laptop scientists in drug discovery workflows beyond current practice, facilitated by accessible visualization tools such as the Jupyter notebooks used here (Figure 9) to enable interactive exploration and discussion.

We see this virtuous cycle ourselves in the ongoing experimental work currently in progress to test candidates iteratively from our structure-based virtual screening pipeline with an eye toward novel therapeutics to interfere with SARS-CoV-2 proteins and treat COVID-19. Given the unprecedented parallel research efforts across the scientific community aimed at developing both pharmacological treatments and vaccines, ours is among many in the search for therapeutic solutions for COVID-19. Moving forward, we see the long-term value for the computational pipeline developed here in its application to other pharmacological challenges. While COVID-19 was the catalyst to bring together scientists and engineers with different and complementary skill sets, no part of this docking and analysis workflow is COVID-specific.

We therefore envision a future where this type of large-scale docking and statistical analysis workflow is commonplace in drug discovery, and making a real difference in selecting promising drug candidates from a large pool of synthetically available compounds. The field has been making steps in this direction for many years, with recent



Select top compounds

```
[51]: q = "SELECT * FROM scores ORDER BY vs_dude_v2 DESC LIMIT 100000"

[52]: bc.explain(q)

[52]: 'LogicalSort(sort0=[$4], dir0=[DESC], fetch=[100000])\n  LogicalTableScan
      (table=[[main, scores]])\n'

[*]:  %%time
      res = bc.sql(q)
```

Figure 9. Web-interface to the virtual lab using JupyterHub, highlighting visualization of both scientific and computational details of interest through accessible tools.

multi-million ligand virtual screens generating considerable excitement and a serving as a template for future work (Gorgulla et al., 2020; Lyu et al., 2019). By substantially reducing both the walltime (Table 4) and hardware costs for conducting large-scale structure-based virtual screening calculations, the improvements made here are an enabling technology to bring this type of comprehensive drug discovery pipeline up to date with GPU computing trends in the wider HPC landscape.

Author's Note

Josh V Vermaas is also affiliated with MSU-DOE Plant Research Laboratory, Michigan State University, East Lansing, MI, USA.

Acknowledgements

The authors thank Rupesh Agarwal for extensive support and assistance with input preparation and scientific and methodological discussions, Jeremy C. Smith and his group for support and inspiration. We also thank Duncan Poole, Geetika Gupta, Jon Lefman, and the rest of the NVIDIA team as well as Diogo Santos-Martins and the Scripps Research team for support and coordination. We thank developers of BlazingSQL, Rodrigo Aramburu, William Malpica, and Felipe Aramburu, and NVIDIA RAPIDS for their support in deploying their database solutions to Summit. We thank Jason Kincl for help with the deployment of FireWorks on Marble. We thank Omar Demerdash for initial exploration of rescoring methods. We thank Jamey Kinney, Usman Qureshi, and Miles Euell (Google).

Copyright statement

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research was sponsored by the Laboratory Directed Research and Development Program at Oak Ridge National Laboratory (ORNL), which is managed by UT-Battelle, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC05-00OR22725. This work also used resources, services, and support provided via the COVID-19 HPC Consortium (https://covid19-hpc-consortium.org/), which is a unique private-public effort to bring together government, industry, and academic leaders who are volunteering free compute time and resources in support of COVID-19 research, and used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

ORCID iDs

Jens Glaser https://orcid.org/0000-0003-1852-3849
Josh V Vermaas https://orcid.org/0000-0003-3139-6469
David M Rogers https://orcid.org/0000-0002-5187-1768

Notes

1. https://spark.apache.org.
2. https://hadoop.apache.org.
3. https://parquet.apache.org.
4. https://orc.apache.org.
5. http://www.tpc.org/tpcx-bb/default5.asp.
6. https://enamine.net/library-synthesis/real-compounds/real-database.
7. https://github.com/ascheinb/AutoDock-GPU/tree/kokkos_develop.
8. https://github.com/scottlegrand/contacts.
9. https://cupy.dev.
10. https://github.com/BlazingDB.
11. https://www.ibm.com/support/pages/get-started-ibm-wml-ce.
12. https://github.com/apache/arrow.
13. https://github.com/abseil/abseil-cpp.
14. https://github.com/rapidsai/cudf.
15. https://github.com/oddt/oddt.
16. https://github.com/scottlegrand/contacts.
17. http://www.tpc.org/tpcx-bb/default5.asp.
18. https://github.com/rapidsai/gpu-bdb.
19. https://blogs.nvidia.com/blog/2020/06/22/big-data-analytics-tpcx-bb/.
20. https://redis.io/.

References

Adeshina YO, Deeds EJ and Karanicolas J (2020) Machine learning classification can reduce false positives in structure-based virtual screening. *Proceedings of the National Academy of Sciences of the United States of America* 117(31): 18477–18488.

Alhossary A, Handoko SD, Mu Y, et al. (2015) Fast, accurate, and reliable molecular docking with QuickVina 2. *Bioinformatics* 31(13): 2214–2216.

Allen WJ, Balius TE, Mukherjee S, et al. (2015) DOCK 6: impact of new features and current docking performance. *Journal of Computational Chemistry* 36(15): 1132–1156.

Ballester PJ (2020) Selecting machine-learning scoring functions for structure-based virtual screening. *Drug Discovery Today: Technologies* 32–33: 81–87.

Chu H, Kim S, Lee JY, et al. (2020) Empirical evaluation across multiple GPU-accelerated DBMSes. In: *Proceedings of the 16th International Workshop on Data Management on New Hardware*. New York, NY: ACM. ISBN 9781450380249, pp. 1–3.

Coleman RG, Carchia M, Sterling T, et al. (2013) Ligand pose and orientational sampling in molecular docking. *PLoS One* 8(10): e75992.

Dempcy RO and Skibo EB (1991) Rational design of quinazoline-based irreversible inhibitors of human erythrocyte purine nucleoside phosphorylase. *Biochemistry* 30(34): 8480–8487.

DesJarlais RL, Sheridan RP, Seibel GL, et al. (1988) Using shape complementarity as an initial screen in designing ligands for a receptor binding site of known three-dimensional structure. *Journal of Medicinal Chemistry* 31(4): 722–729.

Fang Y, Ding Y, Feinstein WP, et al. (2016) GeauxDock: accelerating structure-based virtual screening with heterogeneous computing. *PLoS One* 11(7): e0158898.

Friesner RA, Banks JL, Murphy RB, et al. (2004) Glide: a new approach for rapid, accurate docking and scoring. 1. Method and assessment of docking accuracy. *Journal of Medicinal Chemistry* 47(7): 1739–1749.

Gentile F, Agrawal V, Hsing M, et al. (2020) Deep docking: A deep learning platform for augmentation of structure based drug discovery. *ACS Central Science* 6(6): 939–949.

Gorgulla C, Boeszoermenyi A, Wang ZF, et al. (2020) An open-source drug discovery platform enables ultra-large virtual screens. *Nature* 580(7805): 663–668.

Gritsenko D and Hughes G (2015) Ledipasvir/Sofosbuvir (Harvoni): improving options for hepatitis C virus infection. *Pharmacology & Therapeutics* 40(4): 256–276.

Hassan NM, Alhossary AA, Mu Y, et al. (2017) Protein-ligand blind docking using QuickVina-W with inter-process Spatio-temporal integration. *Scientific Reports* 7(1): 15451.

Jain A, Ong SP, Chen W, et al. (2015) FireWorks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience* 27(17): 5037–5059.

Kneller DW, Phillips G, O'Neill HM, et al. (2020) Structural plasticity of SARS-CoV-2 3CL Mpro active site cavity revealed by room temperature X-ray crystallography. *Nature Communications* 11(1): 3202.

Korb O, Stützle T and Exner TE (2006) PLANTS: application of ant colony optimization to structure-based drug design. In: *International Workshop on Ant Colony Optimization and Swarm Intelligence*, Brussels, Belgium, 5–8 September 2004, pp. 247–258.

Korb O, Stützle T and Exner TE (2011) Accelerating molecular docking calculations using graphics processing units. *Journal of Chemical Information and Modeling* 51(4): 865–876.

LeGrand S, Scheinberg A, Tillack AF, et al. (2020) GPU-accelerated drug discovery with docking on the Summit supercomputer: porting, optimization, and application to COVID-19 research. In: *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*. Virtual Event, USA, 21–24 September 2020. DOI:10.1145/3388440.3412472.

Lyu J, Wang S, Balius TE, et al. (2019) Ultra-large library docking for discovering new chemotypes. *Nature* 566(7743): 224–229.

McGann M (2011) FRED pose prediction and virtual screening accuracy. *Journal of Chemical Information and Modeling* 51(3): 578–596.

McIntosh-Smith S, Price J, Sessions RB, et al. (2015) High performance in silico virtual drug screening on many-core processors. *International Journal of High Performance Computing Applications* 29(2): 119–134.

Morris GM, Huey R, Lindstrom W, et al. (2009) AutoDock4 and AutoDockTools4: automated docking with selective receptor flexibility. *Journal of Computational Chemistry* 30(16): 2785–2791.

Norgan AP, Coffman PK, Kocher JPA, et al. (2011) Multilevel parallelization of AutoDock 4.2. *Journal of Cheminformatics* 3(1): 12.

Parks JM and Smith JC (2020) How to discover antiviral drugs quickly. *The New England Journal of Medicine* 382(23): 2261–2264.

Pedregosa F, Varoquaux G, Gramfort A, et al. (2011) Scikit-learn: machine learning in Python. *Journal of Machine Learning Research* 12: 2825–2830.

Ruiz-Carmona S, Alvarez-Garcia D, Foloppe N, et al. (2014) rDock: a fast, versatile and open source program for docking ligands to proteins and nucleic acids. *PLOS Computational Biology* 10(4): e1003571.

Santos-Martins D, Eberhardt J, Bianco G, et al. (2019a) D3R Grand Challenge 4: prospective pose prediction of BACE1 ligands with AutoDock-GPU. *Journal of Computer-Aided Molecular Design* 33(12): 1071–1081.

Santos-Martins D, Solis-Vasquez L, Koch A, et al. (2019b) Accelerating AutoDock4 with GPUs and gradient-based local search. *ChemRxiv*. DOI: 10.1021/acs.jctc.0c01006.

Schmuck F and Haskin R (2002) GPFS: a shared-disk file system for large computing clusters. In: *Proceedings of the FAST 2002 Conference on File Storage Technologies*. Monterey, CA, USA, 28–30 January 2002. Monterey, CA: USENIX Association, pp. 19–es.

Shen C, Ding J, Wang Z, et al. (2020) From machine learning to deep learning: advances in scoring functions for protein–ligand docking. *Wiley Interdisciplinary Reviews: Computational Molecular Science* 10(1): e1429.

The Antiretroviral Therapy Cohort Collaboration (2008) Life expectancy of individuals on combination antiretroviral therapy in high-income countries: a collaborative analysis of 14 cohort studies. *Lancet* 372(9635): 293–299.

Ton AT, Gentile F, Hsing M, et al. (2020) Rapid identification of potential inhibitors of SARS-CoV-2 main protease by deep docking of 1.3 billion compounds. *Molecular Informatics* 39(8): 2000028.

Trott O and Olson AJ (2010) AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of Computational Chemistry* 31(2): 455–461.

Virshup AM, Contreras-García J, Wipf P, et al. (2013) Stochastic voyages into uncharted chemical space produce a representative library of all possible drug-like compounds. *Journal of the American Chemical Society* 135(19): 7296–7303.

von Itzstein M, Wu WY, Kok GB, et al. (1993) Rational design of potent sialidase-based inhibitors of influenza virus replication. *Nature* 363(6428): 418–423.

Wójcikowski M, Ballester PJ and Siedlecki P (2017) Performance of machine-learning scoring functions in structure-based virtual screening. *Scientific Reports* 7(1): 46710.

Wójcikowski M, Zielenkiewicz P and Siedlecki P (2015) Open drug discovery toolkit (ODDT): a new open-source player in the drug discovery field. *Journal of Cheminformatics* 7: 26.

Yasuo N and Sekijima M (2019) Improved method of structure-based virtual screening via interaction-energy-based learning. *Journal of Chemical Information and Modeling* 59(3): 1050–1061.

Author biographies

Jens Glaser Jens is a computational scientist at the National Center for Computational Sciences at Oak Ridge National Laboratory, where his research involves developing novel algorithms for biological and soft matter simulation, with application to nanoparticles and synthetic biomaterials made from protein building blocks, and drug discovery. Jens obtained his doctorate in physics from Leipzig University in Germany, worked as a postdoc at the University of Minnesota, and as assistant research scientist at the University of Michigan. A common thread in his research connects the simulation of collective phenomena in many-particle systems and the search for parallel approaches to molecular simulation, through the use of GPUs.

Josh V. Vermaas Josh was a computational scientist in the National Center for Computational Sciences Division at Oak Ridge National Laboratory (ORNL) in Oak Ridge, TN, working within the scientific engagement group. He is now applying interested in large-scale molecular dynamics simulations of biological systems as part of the Plant Research Laboratory at Michigan State University. Josh obtained his PhD work in Biophysics at the University of Illinois at Urbana-Champaign, and his postdoctoral tenure at the National Renewable Energy Laboratory was spent advancing molecular simulation analysis tools and GPU-accelerated workflows to address sustainability and renewable energy questions.

David Rogers David is a computational scientist in the National Center for Computational Sciences Division at ORNL, where he works to develop mathematical and computational theory jointly with methods for multiscale modeling using HPC. He obtained his Ph.D. in Physical Chemistry from University of Cincinnati in 2009 where he worked with Prof. Thomas Beck on applying Bayes' theorem to the free energy problem in multiscale modeling of fluids and interface chemistry. David Rogers completed his postdoctoral work at Sandia National Labs, and was Assistant Professor at University of South Florida before joining ORNL.

Jeff Larkin Jeff is a Senior Developer Technologies (DevTech) Software Engineer with NVIDIA, where he specializes in porting and optimizing HPC applications to NVIDIA platforms. Since 2013 he has been an active contributor to the OpenMP and OpenACC specification committees, where he has provided his application-development expertise in the development of these specifications. Jeff has a M.S. in Computer Science from the University of Tennessee, where he was a member of Jack Dongarra's Innovative Computing Laboratory. Before that he received a B.S. in computer science from Furman University. Prior to joining NVIDIA, Jeff was a member of the Cray Supercomputing Center of Excellence at ORNL.

Scott LeGrand Scott Le Grand is a distinguished engineer at NVIDIA. At various points in his life, he was the lead author of DSSTNE, the Deep Scalable Sparse Tensor Network, an AI framework that revolutionized Amazon's product recommendation system, he was granted 14 patents for innovative GPU algorithms, accelerated Molecular Dynamics by 1000x by porting Folding@Home and AMBER to CUDA, and demonstrated all of this can be made bitwise-reproducible without loss of performance or efficiency.

Swen Boehm Swen is a research and development staff member in the Computer Science and Mathematics Division at ORNL. Swen's research is focused on runtime environments, tools and programming models for the HPC environment. He graduated from The University of Reading, UK with a masters in Network Centered Computing.

Matthew B. Baker Matthew is a technical staff member at ORNL where his research interests include programming environments and run times for massively distributed systems. Matthew graduated from East Tennessee State University (ETSU) with a masters in applied computer science. Matthew was instrumental in the development of the UCX platform and has been developing ways to improve the use of Python interfaces for UCX such as used in Dask. Matthew's current work includes monitoring systems are large scale and allowing applications to make decisions at runtime based on the measurements from the running system.

Aaron Scheinberg Aaron is a computational scientist at Jubilee Development focusing on exascale computing, scientific application performance, particle-based methods, magnetic fusion simulations, and GPU programming. Following his doctoral thesis in planetary science at the Massachusetts Institute of Technology, he worked in computational plasma physics at the Swiss Federal Institute of Technology (EPFL) and the Princeton Plasma Physics Laboratory (PPPL).

Andreas F. Tillack Andreas is a statistical mechanician and quantum theoretician with a life-long passion for code development who happens to like organic systems. After his Diplom-Physics degree from Humboldt-University Berlin (building a flow cytometer from the optical pickup unit of a DVD burner) he obtained a PhD in Chemistry from the University of Washington, Seattle (discovering design rules for organic non-linear optical chromophores and developing an ellipsoidal coarse-grained force field). He did his post-doc at ORNL speeding up Quantum Monte-Carlo on Summit and is currently at Scripps Research developing the new generation of AutoDock.

Mathialakan Thavappiragasam Mathi is a postdoctoral research associate in the Biophysics Group in the Biological Sciences Division at ORNL. He is a computational scientist working on GPU programming for simulation and informatics in biology and biophysics, as well as on programming models and performance portability for the OLCF's exascale supercomputer, Frontier. He received his dual PhD in Electrical and Computer Engineering and Computational Mathematics, Science, and Engineering at Michigan State University where he worked on electromagnetic simulation and HPC parallel programming for numerical solutions of differential equations.

Ada Sedova Ada is a biophysicist and research and development staff member in the Biophysics Group in the Biological Sciences Division at ORNL. She received her PhD from the University at Albany, SUNY/NYS Department of Health Wadsworth Center Biomedical Sciences program with a focus on biophysics and structural biology. She worked as a postdoctoral research associate at the University of Albany's Chemistry department and at ORNL's National Center for Computational Sciences. She works on simulation, bioinformatics, and applying HPC to these areas, as well as performing experiments such as neutron scattering and electrochemistry. Ada is interested in achieving more accurate models of physical systems with HPC and connecting experiments to simulation.

Oscar Hernandez Oscar is a senior staff member of the Programming Systems research group which supports the programming environment for the Oak Ridge Leadership Computing Facility (OLCF) systems. He represents ORNL in many specification organizations such as OpenACC and

OpenMP, has represented ORNL in SPEC/HPG and OpenSHMEM initiatives and helped to start the OpenUCX project. He is currently part of the SOLLVE ECP team and is the application liaison for the project. At ORNL he works closely with application teams including the CAAR and INCITE efforts and constantly interacts with them to address their programming model and tools needs via HPC software ecosystems.