2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 979-8-3503-3766-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/IPDPS54959.2023.00063

Optimizing Cloud Computing Resource Usage for Hemodynamic Simulation

William Ladd
Biomedical Engineering
Duke University
Durham, NC, USA
william.ladd@duke.edu

Christopher Jensen
Biomedical Engineering
Duke University
Durham, NC, USA
christopher.w.jensen@duke.edu

Madhurima Vardhan
Biomedical Engineering
Duke University
Durham, NC, USA
mv82@duke.edu

Jeff Ames
Biomedical Engineering
Duke University
Durham, NC, USA
jeff.ames@duke.edu

Jeff R. Hammond
HPC Software
NVIDIA Helsinki Oy
Helsinki, Finland
jeff_hammond@acm.org

Erik W. Draeger
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA, USA
draeger1@llnl.gov

Amanda Randles
Biomedical Engineering
Duke University
Durham, NC, USA
amanda.randles@duke.edu

***Abstract*—Cloud computing resources are becoming an increasingly attractive option for simulation workflows but require users to assess a wider variety of hardware options and associated costs than required by traditional in-house hardware or fixed allocations at leadership computing facilities. The pay-as-you-go model used by cloud providers gives users the opportunity to make more nuanced cost-benefit decisions at runtime by choosing hardware that best matches a given workload, but creates the risk of suboptimal allocation strategies or inadvertent cost overruns. In this work, we propose the use of an iteratively-refined performance model to optimize cloud simulation campaigns against overall cost, throughput, or maximum time to solution. Hemodynamic simulations represent an excellent use case for these assessments, as the relative costs and dominant terms in the performance model can vary widely with hardware, numerical parameters and physics models. Performance and scaling behavior of hemodynamic simulations on multiple cloud services as well as a traditional compute cluster are collected and evaluated, and an initial performance model is proposed along with a strategy for dynamically refining it with additional experimental data.**

***Index Terms*—Cloud computing, hemodynamic simulations, computational fluid dynamics, interconnect, noise variability**

I. Introduction

Non-invasive hemodynamic assessment of cardiovascular pathologies through high-fidelity simulation holds tremendous potential to improve patient outcomes and reduce healthcare costs [1], [2]. State-of-the-art computational fluid dynamic (CFD) models have been US Food and Drug Administration (FDA) approved and are being clinically adopted [3]–[5]. However, even as hemodynamic simulation continues to mature as a diagnostic tool, predicting the necessary computational resources for a given study remains a key challenge. Factors such as anatomic domain size, physical phenomena being modeled, and the numerical precision and spatial resolution required for accurate predictions can vary significantly, which profoundly impact the computational profile and resulting resource requirements. This variation in computational cost is a nontrivial challenge on traditional on-premise high performance computing (HPC) clusters, where the hardware is well-known and predictable. The recent rise of cloud computing as a viable resource for HPC applications introduces additional variables that further necessitate a robust and predictive performance model.

Cloud platforms offer pragmatic advantages for performing hemodynamic simulations, mitigating the limitations of infrastructure and upfront capital costs:

1) On demand provisioning enabled by a usage-based cost model that eliminates the problem of under-provisioned computational resources [6], [7].
2) Flexible architectures customized for a particular workload and lower latency to queue a job versus a traditional HPC center with high utilization [7]–[9].
3) Cloud infrastructures facilitate easy global (location-independent) access and collaboration, enabling standardization of healthcare-compliance approaches [10].
4) Virtualization and application programming interfaces (APIs) for streamlined workload management and interaction [8], [11].

With HIPAA-compliant cloud providers now available, the barriers to patient-specific hemodynamic simulations in the cloud are far lower than with traditional supercomputing center resources [10]. With this accessibility comes the need to simplify run-time decisions that have traditionally been made by computational domain experts.

In this work we make several contributions. First, we assess the suitability of cloud computing infrastructures for performing hemodynamic simulations with real workloads at a scale relevant to practical use cases. We then establish and validate a performance model to accurately guide usage decisions. We determine how cloud hardware features affect HPC performance by comparing benchmark applications to real workloads using massively-parallel CFD code across

1530-2075/23/$31.00 ©2023 IEEE
DOI 10.1109/IPDPS54959.2023.00063

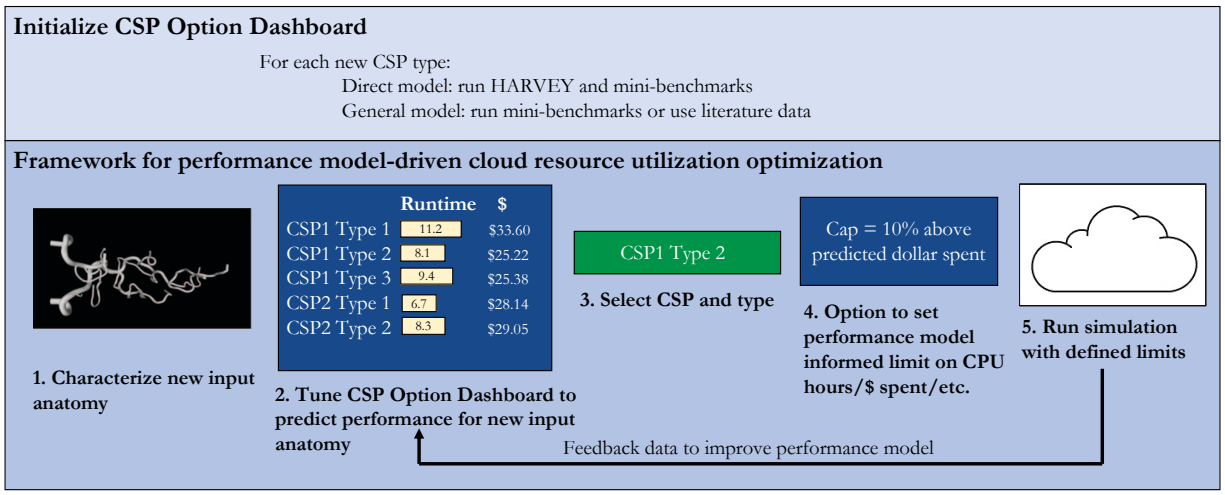

Fig. 1. Overview of proposed framework for performance model-driven optimization of cloud resource usage. In the first phase (top), the Cloud Service Provider (CSP) Option Dashboard is created by characterizing all CSP instance types. In the second phase (bottom), the performance model is tuned to create anatomy-specific predictions for the user to drive cloud use decisions.

multiple cloud computing instances. This work also examines the impact of interconnect speed on HPC instances of cloud infrastructures and predicts the upper bound of performance for applications. The relative value of compute time in terms of raw throughput for different instances is compared and proposed as a metric for cost comparison analysis. Overall, we establish a framework (Figure 1) for optimizing cloud computing resource usage and reducing the risk of inadvertent cost overruns for blood flow simulations.

II. Methods

In order to optimize cloud usage and reduce potential overrun costs, it is critical that we provide the user with a process for assessing the performance of their targeted input file for any available cloud instance. The user can then use an informed performance model to assess key metrics such as time-to-solution, flops/dollar, or total cost to both drive their instance choice and actively put in place limits on the job to bound usage. This framework is laid out in Figure 1. First, it is necessary to assess the performance of the targeted algorithm. In the following sections, we introduce the underlying algorithm and two representative implementations. We investigate both an open source proxy application and the production-scale code, HARVEY [12].

A. Lattice Boltzmann Method for modeling fluid mechanics

In this work, we focus on blood flow models using the lattice Bolzmann method (LBM) to efficiently model hemodynamics in complex, patient-specific anatomies. LBM, an alternative solver of the Navier-Stokes equations, [13] is a stencil-based code that is highly amenable to large-scale parallelization [14]. Derived from the Boltzmann equation of kinetic theory, the LBM models a given fluid as a probability distribution function f_i of particles moving on a regular lattice $\mathbf{x}$ with discrete velocities $\mathbf{c}_i$. Macroscopic fluid variables, such as pressure p and velocity $\mathbf{u}$, are computed locally as moments of $f_i(\mathbf{x},t)$. The time evolution of the distribution $f_i(\mathbf{x},t)$ for timestep Δt is governed by the lattice Boltzmann equation:

$$f_i(\mathbf{x}+\mathbf{c}_i\Delta t, t+\Delta t) - f_i(\mathbf{x},t) = -\Omega(f_i(\mathbf{x},t) - f_i^{eq}(\mathbf{x},t)) \quad (1)$$

wherein the collision operator is $\Omega(f_i(\mathbf{x},t))$ and Maxwell-Boltzmann equilibrium distribution is $f_i^{eq}(\mathbf{x},t)$. A comprehensive explanation of LBM may be found in [15]. The LBM representation uses a discretized approach and overall performance of the algorithm is typically reported in millions of fluid point updates per second (MFLUPS), which quantifies the work done per unit of time. This metric is ideally suited for our purposes, as it is independent of domain size and number of timesteps [12].

B. LBM Proxy Application

We use the open source LBM proxy application `lbm-proxy-app` [16] to test our hypotheses and performance models. `lbm-proxy-app` runs fluid-only LBM in a simple cylindrical geometry, an idealized case designed to isolate and capture the performance of the most common LBM kernels. This can be compared against the benchmark predictions to confirm or rule out the presence of other performance losses for a given LBM code. `lbm-proxy-app` offers both array of structures (AOS) and structure of arrays (SOA) data structures, shown to be faster for LBM on CPUs and GPUs, respectively [17].

C. HARVEY

HARVEY is a massively-parallel circulatory modeling code designed for scalability using realistic arterial geometries [12], [14]. The HARVEY implementation of the LBM uses a standard D3Q19 velocity discretization and the single relaxation time Bhatnagar–Gross–Krook (BGK) collision kernel. While

the LBM proxy application is hardcoded for a cylinder geometry, HARVEY provides accurate flow simulation in complex anatomies. The code has previously been shown to accurately capture hemodynamic flow in cerebral [18], femoral [19], and coronary arterial domains [20]. In this study, velocity boundary conditions were imposed as a Poiseuille profile at the inlets and a zero-pressure boundary condition was applied at the outlets [21]. Though not used in this study, HARVEY also supports both rigid and deformable walls as well as explicit deformable cells modeled with the Immersed Boundary Method and can be run on both CPUs and GPUs at scale.

D. Establishing the performance model

As these fluid dynamics models are meant to be robust and flexible tools for circulatory modeling, predicting performance requires accurately characterizing numerous terms, each of which may change significantly depending on the numerical parameters, parallel decomposition and compute hardware, e.g. a full model of fluid (LBM), cells, and vessel walls in HARVEY running on multiple GPUs would consist of time required for each algorithm (t^{LBM}_{stream}, $t^{LBM}_{collide}$, t^{cells}_{forces}, t^{cells}_{pos}, t^{walls}_{forces}, t^{walls}_{pos}), halo exchanges for parallelism of each algorithm (t^{LBM}_{halo}, t^{cells}_{halo}, t^{walls}_{halo}), and CPU-GPU data transfer ($t_{CPU-GPU}$) [22] thus total runtime can be written as:

$$\begin{aligned} T_{full} =& t^{LBM}_{stream} + t^{LBM}_{collide} + t^{LBM}_{halo} + t^{cells}_{forces} + \\ & t^{cells}_{pos} + t^{cells}_{halo} + t^{walls}_{forces} + t^{walls}_{pos} + \\ & t^{walls}_{halo} + t_{CPU-GPU} \end{aligned} \tag{2}$$

Rather than attempting to describe all possible scenarios with a complex model using Eq. 2, we propose to start from the simplest possible use case of pure LBM fluid on CPUs, eliminating cells runtime, walls runtime, and CPU-GPU data transfer from Eq. 2 giving us Eq. 3 as our starting performance model. Additional terms will be added once the validation of the performance model via iterative refinement is well-established.

$$T = t^{LBM}_{stream} + t^{LBM}_{collide} + t^{LBM}_{halo} \tag{3}$$

Because LBM is known to be bandwidth-bound in most use cases [12], [23], we will replace Eq. 3 with estimates of the time required to access each task's data from memory and communicate each task's halo region to its neighbors hence:

$$t^{mem}_j = t^{LBM}_{stream\text{-}j} + t^{LBM}_{collide\text{-}j} \tag{4}$$

$$t^{comm}_j = t^{LBM}_{halo\text{-}j} \tag{5}$$

$$T \approx \max_j(t^{mem}_j) + \max_j(t^{comm}_j) \tag{6}$$

where t^{mem}_j is the estimated time to access all local fluid data on task j and t^{comm}_j is estimated time to communicate the halo region of task j to its neighbors. This latter estimate necessarily requires some knowledge or estimate of the parallel decomposition that will be used. LBM being a stencil application also leaves this model open to being generalized to other memory-bound stencil applications, specifically ones that consist of memory accesses for each point on the stencil, halo exchanges between tasks, and summing them together as in Eqns. 4, 5, and 6 respectively; however, this generalization would not hold if the stencil application has high computing intensity which would need to be considered as a runtime cost and as a factor in memory bandwidth competition. The other elements to HARVEY described in Eq. 2 can also be added with this framework however CPU-GPU communication time must be considered. This consideration could be implemented by assuming the node's memory bandwidth is saturated (or is at some proportion of saturation) and the amount of memory bandwidth is divided evenly among the cores in use. For the experiments described in the next section, performance was estimated in two ways: directly using the data sizes corresponding to the parallel decomposition used by HARVEY and `lbm-proxy-app` as well as from a general model of the predicted data and halo sizes. The two models are shown together to separate out the error in the underlying performance model form from the error associated with estimating the parallel decomposition.

To measure simulation throughput, we use a standard LBM metric of millions of fluid point updates per second (MFLUPS), defined as:

$$MFLUPS = \frac{n_{timesteps} \cdot n_{points}}{T \cdot 10^6} \tag{7}$$

1) Quantifying memory accesses for kernel fluid point updates: Since LBM algorithms are memory-bound on nearly all general-purpose hardware [24]–[26], we can estimate the time to update all of the fluid points on a task t^{mem}_j as the number of bytes accessed to update all fluid points on that task divided by the sustained memory bandwidth. HARVEY operates in a memory regime where cache effects are negligible due to the amount of memory accessed each timestep. As demonstrated previously [26], runtime of HARVEY is predominately influenced by memory bandwidth. The STREAM benchmark [27] is the canonical method for measuring memory bandwidth on HPC systems. It consists of Copy, Scale, Add and Triad, which access one or two input arrays and write one output array after performing zero, one or two arithmetic operations per element. Despite their simplicity, these kernels are sufficient to understand the behavior of complex HPC workloads, many of which are limited by memory bandwidth and have similar ratios of read and write operations. Even when the read-write ratio is heavily skewed (as in the case of the HPCG benchmark), there is a near-perfect correlation with STREAM, due to the way most memory systems are implemented. We use measurements from the Copy function, as it best reflects the bandwidth achievable by LBM kernels [23].

We observed that the available memory bandwidth on a given node scales with the number of cores in use in two regimes as follows. Each core has inherent memory access speed limits thus adding cores increases memory access speed; however, once a certain number of cores are in use, the memory access speed limit of the node's memory subsystem is reached thus cores compete for memory accesses and the

node's bandwidth (the bandwidth used by the cores on the node) plateaus, increasing much less with further additional cores. Consequently, a node's memory access bandwidth B_{NODE} over a number of cores n, as measured by STREAM over a sweep of OpenMP threads with one thread per core, can be fit to a two-line model (Eq. 8) by adjusting the parameters a_1, a_2, and a_3 to minimize the sum of square errors (SSE) between the curve and the points. The two-line model captures the slope of the bandwidth in each regime one being limited by core speed for $n < a_3$ and the other being limited by memory subsystem speed where $n > a_3$ but still increasing due to the cores competing for memory access.

$$B_{NODE}(n) = \begin{cases} a_1 n & \text{for } n < a_3 \\ a_2 n + a_3(a_1 - a_2) & \text{for } n \geq a_3 \end{cases} \quad (8)$$

Memory bandwidth available to a single task will depend on how many tasks and cores are currently accessing memory on that core and node respectively. For simplicity, we assume that the available memory bandwidth is linearly dependent on the number of tasks per node, but this may require a more sophisticated model on hardware where memory locality and resource contention significantly impacts performance.

For the direct performance model, where the parallel decomposition of each code is used to determine the relative memory sizes on each task, the total memory accesses for each task are computed using:

$$n_{bytes\text{-}j} = \sum_{i=0}^{N_j - 1} \sum_{k=0}^{n_k - 1} (n_{vectors\text{-}j,i,k} \cdot n_{accesses\text{-}k} \cdot d_{size}) \quad (9)$$

wherein:

- $n_{vectors\text{-}j,i,k}$ = the number of vectors in point i of task j that are updated by operation k
- n_k = the number of different fluid point operations
- $n_{accesses\text{-}k}$ = the number of data accesses per vector required by operation k
- d_{size} = data size of each vector in bytes (4 for single precision float, 8 for double precision float, 16 for quadruple precision float)

It must be noted that different types of fluid points (e.g. bulk vs. wall) have different memory requirements which must be taken into account. Different kernel layouts (e.g. AB vs. AA) can also affect the memory layout and data sizes.

For the general performance model, where the data sizes on each task are estimated a priori for a given number of tasks n_{tasks}, the maximum memory on a single task is approximated as:

$$\max_j(n_{bytes\text{-}j}) \approx z \cdot \frac{n_{bytes\text{-}serial}}{n_{tasks}} \quad (10)$$

$$z = c_1 \cdot \ln(c_2(n_{tasks} - 1) + 1) + 1 \quad (11)$$

where z is the deviation from perfect load balance and c_1 and c_2 are empirical parameters derived from fits of Eq. 11 to prior HARVEY decomposition data using Eq. 10 wherein each task's memory accesses were counted for a sweep of task counts then the maximum for each task count was divided by $\frac{n_{bytes\text{-}serial}}{n_{tasks}}$, the number of byte accesses required in the ideal load balancing case wherein the number of bytes accesses required for the single task in a serial run to update the entire simulation domain $n_{bytes\text{-}serial}$ is divided evenly among all tasks n_{tasks}, to get an actual increase factor that z approximates.

2) Quantifying communication time for halo exchanges: The second largest contributor to our performance model is the time to communicate the halo region from one task to its neighbors. While communication costs for LBM codes are often relatively modest due to high surface-to-volume ratios between the local data volume and the surrounding halo, they can become significant at the strong-scaling limit. This effect may be further magnified on some cloud computing instances, where high-end interconnect hardware is less reliably available. This potential variability makes highlights the need for a quantitative performance estimate in order to assess the impact of different hardware options on performance for specific simulation inputs.

We model the inter-node data exchange time using a linear communication model:

$$t_{comm\text{-}event} = \frac{m}{b} + l \quad (12)$$

wherein:

- $t_{comm\text{-}event}$ = runtime of the single communication event
- m = size, in bytes, of the message communicated
- b = bandwidth of the communication
- l = latency of the communication

A linear function has been shown to be a reasonable model for communication time over the ranges addressed here [28].

We measure the communication bandwidth and latency using standard benchmark tools such as the Intel MPI Benchmark [29], which generates scaling reports for sustainable bandwidth for most MPI operations when sharing bandwidth over a network. We use its implementation of the PingPong benchmark to measure MPI communication time between pairs of ranks for both in-node (intranodal) communication, where both the ranks are in the same node, and out-of-node (internodal) communication, where each rank is located in a different node and communication must go through an interconnect between nodes. Results are reported for a range of message sizes to show the effects of latency and bandwidth on communication time to measure the limits of the hardware. The bandwidth is determined from a linear fit using Eq. 12. This is done separately for in-node (intranodal) communication and out-of-node (internodal) communication.

The data derived from the PingPong benchmark were then used to estimate halo communication costs within LBM implementations. Consider a cubic simulation domain containing N fluid points subdivided among n_{tasks} tasks. After domain decomposition, the sub-cube assigned to each task has $\frac{N}{n_{tasks}}$ points and the amount of points each task has that neighbor points contained by another task is proportional the sub-cube's surface area. Adding in the previously estimated load

imbalance z from Eq. 11, we approximate the halo data size as:

$$m_{max\text{-}total} = \frac{w}{6} \cdot \left(z \cdot \frac{N}{n_{tasks}} \right)^{\frac{2}{3}} \cdot 2 \cdot n_{point\text{-}comm\text{-}bytes} \quad (13)$$

where $n_{point\text{-}comm\text{-}bytes}$ is the amount of data to be communicated for each boundary point in bytes and w is the maximum number of neighbors a given task in the cubic domain and subdomains case can have:

$$w = \min(\log_2(n_{tasks}), 6) \quad (14)$$

The additional factor of two in Eq. 13 accounts for the fact that data is both sent and received in an LBM halo exchange. In practice, the maximum number of messages a given task sends and receives will depend on the domain decomposition, the simulation geometry and the total number of tasks, so we also do an empirical fit using:

$$n_{max\text{-}events} = 4 \cdot \log_2((k_1 n_n^{-1} + k_2) \cdot (n_{tasks} - n_n) + 1) \quad (15)$$

We combine these to estimate the maximum time spent in communication:

$$t_{comm\text{-}max} = \frac{m_{max\text{-}total}}{b} + n_{max\text{-}events} \cdot l \quad (16)$$

III. Experiments

A. Representative use cases

In order to assess performance for realistic workloads, we selected three increasingly complex geometries: 1) an idealized cylindrical vessel, 2) an aorta, and 3) a cerebral vasculature (Figure 2). The cylinder geometry represents an idealized vessel domain that is easily divided for parallel simulation, but has high communication cost. The aorta and cerebral models were obtained from the Open Source Medical Software repository [30] and represent anatomically accurate geometries with domains that are challenging to divide evenly but contain more wall points, thus requiring less communication as the domain is spread out with small cross-sections so communication surfaces are small. Identical, high-resolution steady bulk flow simulations were performed in each geometry with the same number of cores on all computational platforms.

B. Targeted infrastructures

We evaluated configurations provided by two cloud service providers (CSPs) with differing interconnects and a traditional compute cluster (TRC).

1) Cloud infrastructures: Each CSP uses units of virtual CPUs (vCPUs); however, there can be multiple vCPUs per physical core, so both units are provided for each cloud instance. CSP-1 was a dedicated cloud instance wherein we used three 16-core nodes and an InfiniBand interconnect. CSP-2 was an on demand service where we used multiple instances with nodes of different sizes and with different interconnects. For nodes, from CSP-2 we used large nodes that have 36 cores totaling 72 vCPUs and small nodes that have 8 cores totaling 16 vCPUs. For interconnects, CSP-2 has unnamed interconnects that are slower and a proprietary, faster node interconnect that will be referred to here as Enhanced Communicator (EC). We used 2 instances from CSP-2: one which used 4 large nodes and the slow interconnect (CSP-2); and one which used 4 large nodes and the EC interconnect (CSP-2 EC). Hardware details of each instance are described in Table I.

2) Traditional cluster: We compared our CSPs to a traditional Intel HPC cluster with dual-socket compute nodes connected with 56Gb/s InfiniBand interconnects (Table I). Each node has two Intel Xeon E5-2695v4 'Broadwell' processors with 40 CPU cores exposed to the job scheduler and a shared 55 MB L3 cache. Our application was compiled with the 2018 versions of the Intel C++ compiler and MPI library.

C. Evaluating different cloud features

We tested the impact of noise variability, memory access speed, and interconnect speed on cloud platforms on the performance of hemodynamic simulations using HARVEY, `lbm-proxy-app`, and the microbenchmarks described in section II. Noise variability was tested by conducting simulations with HARVEY using a complex geometry as input test set, and for the microbenchmarks on different numbers of nodes over the span of 7 days at intervals of 6 hours. Differences in performance were assessed and variability was calculated as mean and standard deviation. The effect of interconnect was also evaluated as some cloud providers use proprietary interconnects, such as CSP-2 with EC, for internodal communication, as compared to traditional HPC systems which mainly rely on InfiniBand [31]. The use of CSP-2 without EC (CSP-2) and CSP-2 with EC (CSP-2 EC), both described in Table I, are to test specifically the interconnect differences.

D. Performance of LBM on Different Cloud Instances

The performance of LBM-based hemodynamic simulations, measured in MFLUPS, is shown in Figure 3. As shown, strong scaling of HARVEY's performance was nearly the same across all investigated geometries.

Figure 3a shows some slight noise or drifting effects where the smoothness of the strong scaling curve is less than that shown in Figures 3b and 3c which is likely due to the high communication load of the idealized cylindrical geometry. Communication load in the cylinder is higher because of the cylinder having a high bulk fluid point to wall fluid point ratio (it being efficiently packed into a space) compared to other geometries therefore when the domain is decomposed there is much greater communication surface area thus more fluid points must be exchanged and more communication events are required. The cerebral vasculature in Figure 3c performs the best of the geometries as updates for wall fluid points require fewer memory accesses to update than bulk fluid points in HARVEY.

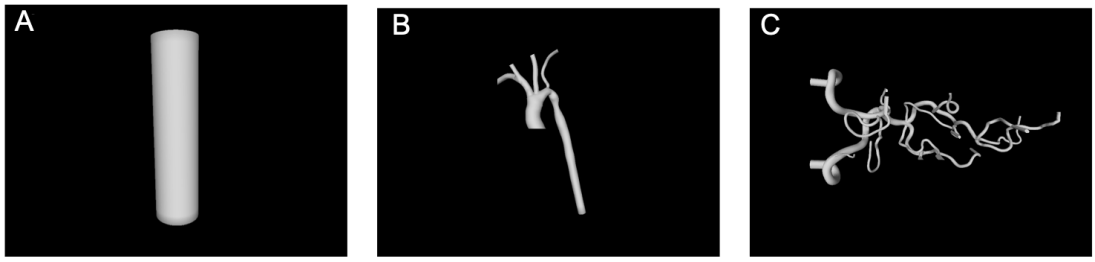

Fig. 2. Arterial geometries used for assessing hemodynamic simulation performance: (A) Idealized cylindrical vessel, a case with high communication but good load balancing, (B) Aorta, a case with typical communication and load balancing, (C) Cerebral vasculature, a case with low communication, many wall points, and typical load balancing.

TABLE I
HARDWARE DETAILS FOR ALL TESTED INSTANCES. (NOTE: THE CSP-2 AND CSP-2 EC INSTANCES ARE REPORTED AS 3.0 GHZ FOR EACH HARDWARE HYPERTHREAD AND 3.4 GHZ ON A SINGLE CORE USING INTEL TURBOBOOST)

System	Traditional Compute Cluster	Cloud 1 - Dedicated	Cloud 2 - Small	Cloud 2 - With EC	Cloud 2 - No EC
Abbreviation	TRC	CSP-1	CSP-2 Small	CSP-2 EC	CSP-2
CPU	Intel Xeon E5-2699 v4	Intel Xeon E5-2667 v3	Intel Xeon E5-2666 v3	Intel Xeon Platinum 8124M	Intel Xeon Platinum 8124M
CPU Clock (GHz)	2.19	3.19	2.42	3.40	3.41
Core Count	2000	48	128	144	144
Cores per Node	40	16	8	36	36
Memory per Node (GB)	471	16	30	192	144
Interconnect (Gbit/s)	56	10	10	100	25

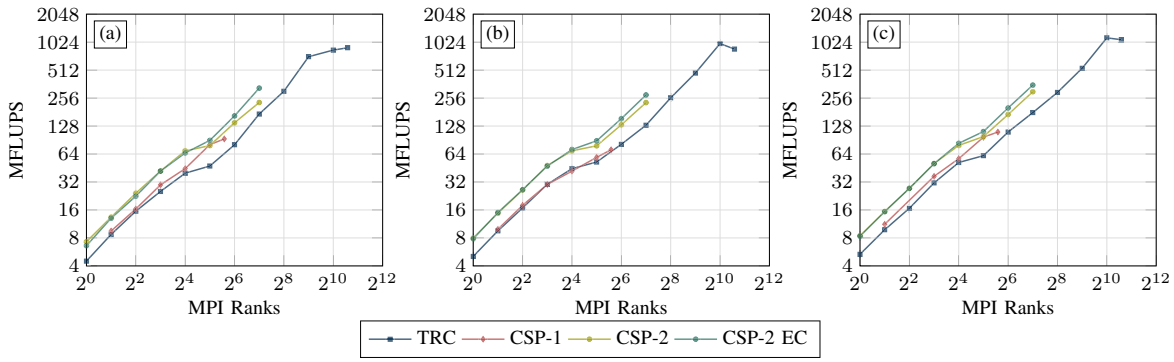

Fig. 3. Strong scaling plots of HARVEY performance for each of the geometries from Figure 2: (a) Idealized cylindrical vessel, (b) Aorta, (c) Cerebral vasculature.

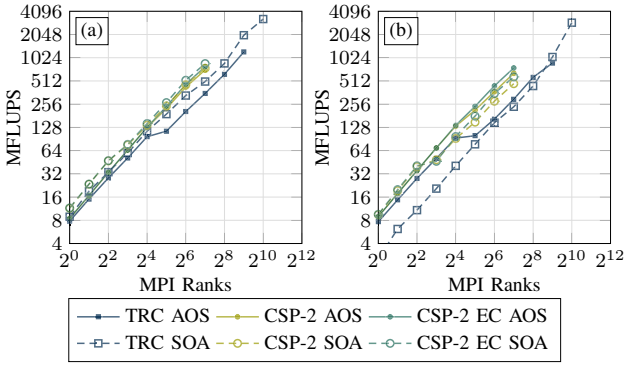

Fig. 4. Strong scaling plots of `lbm-proxy-app` performance for both SOA (with kernel internal for loops unrolled) and AOS data access patterns on each infrastructure for each of the LBM propagation patterns: (a) AA, (b) AB.

Figure 4 shows the strong scaling of four kernels of `lbm-proxy-app` on each computing infrastructure. The AA propagation pattern performance, Figure 4a, is shifted upwards from the AB propagation pattern performance, Figure 4b. This performance increase is expected as the amount of memory accesses is reduced in every other timestep in the AA pattern compared to the AB pattern as the AA pattern uses a single array wherein operations applied alternate each timestep, one timestep applying a single collision operation and the other applying a combined streaming-collision-streaming operation [32] hence accesses to the streaming indexing array are only made every other timestep and only one array of fluid point data is accessed. It is also expected that for CPUs, the AOS memory access pattern performs better than the SOA memory access pattern [17], [33], and this is shown to be true for the

AB propagation pattern but not for the AA pattern.

E. Benchmark Performance of Systems

The sustainable memory access bandwidth of each system, as measured by the STREAM microbenchmark COPY, is shown in Figure 5.

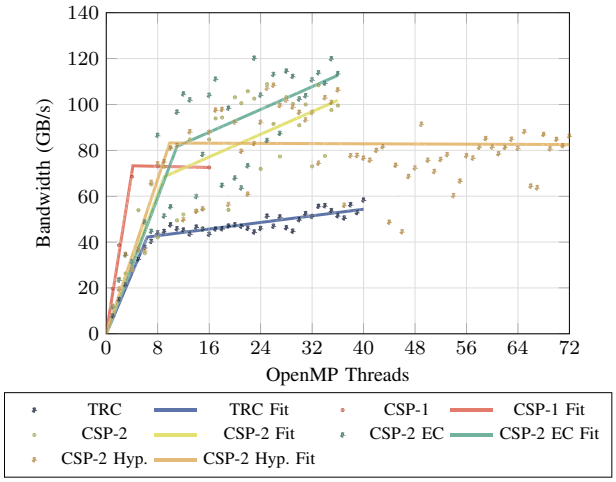

Fig. 5. Memory bandwidth of a single node, as measured by the STREAM microbenchmark COPY, and two-line curve fits using Eq. 8 for a sweep of OpenMP thread counts. Each core on the node is assigned a single OpenMP thread except for CSP-2 Hyperthreaded (Hyp.) which has a thread for each vCPU (two threads to a core).

Figure 5 shows that the two-line model (Eq. 8) replicates the two-phase scaling of memory access bandwidth. For small core numbers, the node memory bandwidth greatly exceeds the limits of individual cores, so there is a steep increase in memory bandwidth per thread. However, increasing the core number will eventually saturate the node's memory access bandwidth, as cores are now competing for the node's limited bandwidth. Adding additional threads thus leads to a shallower increase in memory bandwidth. The fit is very good for the traditional HPC cluster (blue); however, CSP-2 demonstrates large variance after its inflection point, suggesting that not all cores on CSP-2 nodes have separate memory access bandwidth channels, leading to increased competition between cores and decreased overall memory bandwidth.

Cloud service providers tend to list HPC compute nodes by their number of vCPUs, which can exceed the number of physical cores per node via hyperthreading. However, hyperthreading does not increase the amount of available memory bandwidth. This was investigated on a hyperthreaded CSP-2 instance with two vCPUs per physical core and one OpenMP thread per vCPU (CSP-2 Hyp., Figure 5). We observed decreased memory bandwidth on this hyperthreaded instance, with the actual sustainable memory bandwidth, as measured by the STREAM COPY benchmark, tending to be 20-40% lower than the published maximum nodal memory bandwidth (Table II) with the exception of CSP-1.

TABLE II
STREAM COPY BENCHMARK DATA FIT SUSTAINABLE MEMORY BANDWIDTHS FOR ONE NODE ON EACH SYSTEM WHEREIN THE NUMBER OF OPENMP THREADS IS MAXED OUT AT ONE THREAD PER PHYSICAL CORE, PUBLISHED NODE MAXIMUM MEMORY BANDWIDTHS, AND THE PERCENTAGE DIFFERENCE BETWEEN THE TWO BANDWIDTHS.

Bandwidth Type	**TRC**	**CSP-1**	**CSP-2**	**CSP-2 EC**
Published (MB/s)	76,800	68,000	162,720	162,720
STREAM (MB/s)	∼55,625	∼74,273	∼104,259	∼115,413
Difference	-27.57%	+9.23%	-35.92%	-29.07%

Traditional HPCs such as TRC have significantly less latency and higher bandwidth in internodal communication than CSPs, which reuslts in faster communication times (Figure 6 and Table III). As expected, the communication time of CSP-2 EC was less than CSP-2 without EC, albeit only a 2.65 μs decrease in latency and a 211.93 MB/s increase in bandwidth (Table III). Measured communication times demonstrated some nonlinearity, but a linear communication model was sufficient for an acceptable fit (Figure 6). Defining latency as the communication time for a message of zero bytes did underestimate latency at larger message sizes, but this prevented vast overestimations of communication times for smaller message sizes.

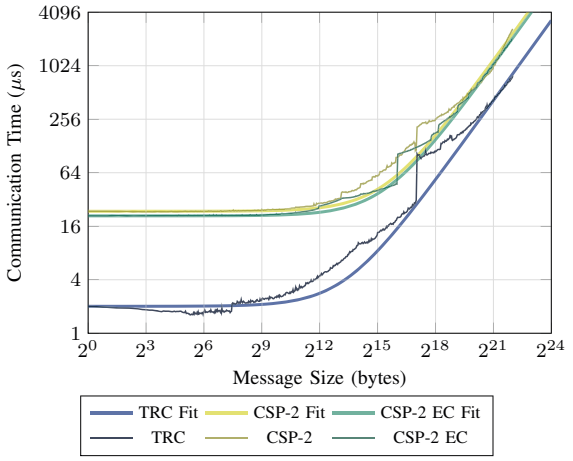

Fig. 6. Measurements and curve fits using Eq. 12 for PingPong communication timings for a large range of message sizes. Curve fits enforce that latency is the communication time for 0 bytes and bandwidth depends on all data points.

Microbenchmark curve fit parameters are provided in Table III. When comparing the TRC, CSP-2, and CSP-2 EC, we see that increasing a node's maximum total memory bandwidth led to an increase in the nodal saturation point a_3, while increased communication times led to decreased communication bandwidth b and increased communication latency l.

TABLE III
MICROBENCHMARK CURVE FIT PARAMETERS FOR CURVES IN FIGURES 5 AND 6 USING EQUATIONS 8 AND 12 RESPECTIVELY TO SHOW PROPERTIES OF STREAM MEMORY BANDWIDTH AND INTERNODAL COMMUNICATION USING PARAMETERS. UNITS ARE MB/SEC-THREADS FOR a_1 AND a_2, THREADS FOR a_3, MB/SEC FOR b_{inter}, AND MICROSECONDS FOR l_{inter}. THE AMOUNT OF CORES IN A SINGLE NODE OF EACH SYSTEM IS PROVIDED FOR REFERENCE. *DENOTES HYPERTHREADING.

System	a_1	a_2	a_3	b_{inter}	l_{inter}	Cores
TRC	6768.24	369.16	6.39	5066.57	2.01	40
CSP-2	7790.02	1264.80	9.00	1804.84	23.59	36
CSP-2 EC	7605.85	1269.95	11.00	2016.77	20.94	36
CSP-2 Hyp.	8629.29	-93.43	9.87	N/A	N/A	72*
CSP-1	18092.64	-62.79	4.15	N/A	N/A	16

F. Noise Variability

Table IV demonstrates that noise variability has little effect on the final simulation performance and is not significantly greater on a cloud cluster than a dedicated cluster.

TABLE IV
HARVEY AORTA PERFORMANCE STATISTICS FROM MEASUREMENTS AT INTERVALS OF 6 HOURS OVER 7 DAYS.

System	MPI Ranks	Mean MFLUPS	Standard Deviation	Variation Coefficient
CSP-1	16	39.04	0.71	0.02
CSP-1	32	54.84	0.51	0.01
CSP-1	48	67.84	0.92	0.01
CSP-2 Small	16	25.53	0.72	0.02
CSP-2 Small	32	42.49	0.85	0.02
CSP-2 Small	64	71.19	0.35	0.004
CSP-2 Small	128	127.99	1.83	0.01

G. Performance Model Predictions

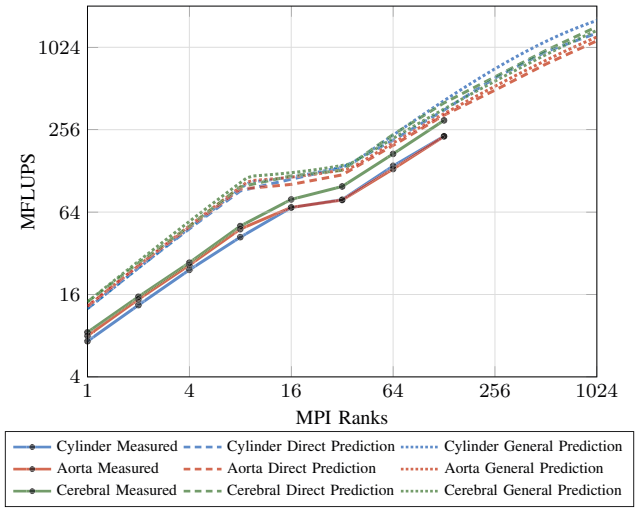

Fig. 7. Performance model predictions and actual performance for all HARVEY geometries on CSP-2 nodes (without EC).

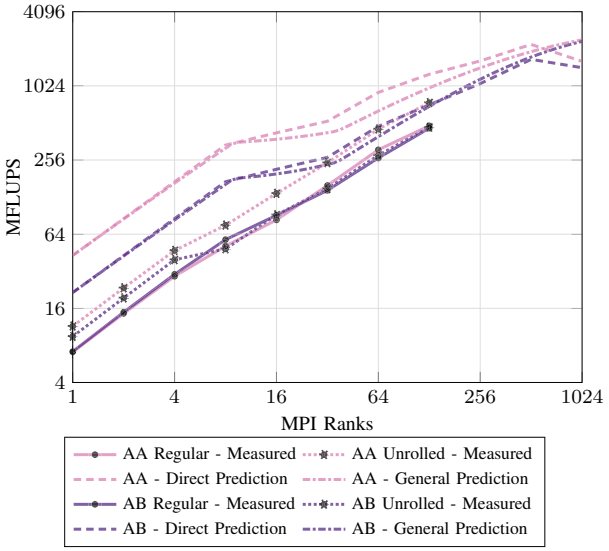

Fig. 8. Performance model predictions and actual performance for `lbm-proxy-app` SOA kernels, including AA and AB propagation patterns and internal for loops with and without unrolling, on CSP-2 nodes (without EC).

Both performance models overpredicted HARVEY (Figure 7) and `lbm-proxy-app` (Figure 8) performances by a consistent amount in all cases. Direct predictions replicate the relative performance for HARVEY being worse for the aorta and cylindrical geometries with the cerebral geometry being better performing and the relative performance for `lbm-proxy-app` being better for AA propagation than AB propagation. The performance improvement of AA over AB in `lbm-proxy-app` is shown to occur only for the unrolled kernels, indicating that the change in for loop structure either changed the data accessing or hid the internal for loop overheads, thereby causing a performance increase. The generalized predictions drift slightly from direct predictions for the cylinder at higher MPI ranks and overestimate performance.

To study the effect of low quality CSP interconnects with low bandwidth and high latency we chose to study the HARVEY cylindrical case on the CSP-2 instance. The high communication and evenly distributed memory accesses on this geometry with the low quality interconnect of CSP-2 allows internodal communication to stand out as it does in a predicted distribution of runtime (Figure 9). To show that the generalized model correctly approximates what is actually happening, as shown in the direct model, and to reveal what attributes of communication, bandwidth and latency, are taking up time, a generalized predicted runtime distribution is shown in Figure 10.

A compromise of the generalized model is that it sums the maximum memory access time and the maximum internodal communication time out of all of the tasks separately. For simulations with high rank counts, this approach can overestimate memory accesses and internodal communication events, while simultaneously underestimating the contribution of in-

tranodal communications. However, the direct performance model (Figure 9) demonstrates that intranodal communications (green) take up much less runtime than memory accesses (red) and internodal communications (purple); therefore, this aspect of the generalized model does not significantly affect its performance predictions.

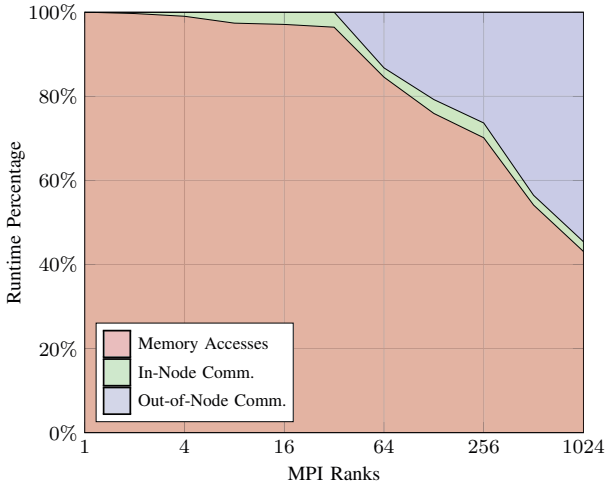

Fig. 9. Composition of maximum task runtimes per core count as predicted by the direct model for a strong scaling case of HARVEY using the cylindrical geometry on CSP-2 without EC.

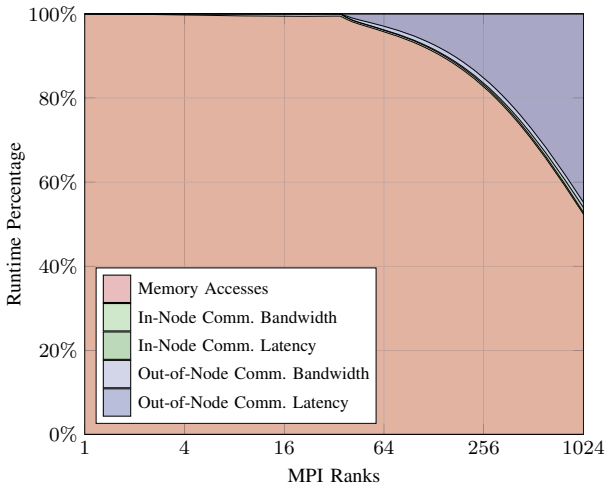

Fig. 10. Composition of maximum task runtimes per core count as predicted by the generalized model for a strong scaling case of HARVEY using the cylindrical geometry on CSP-2 without EC.

Given these runtime compositions, the increased performance of HARVEY and `lbm-proxy-app` observed in the CSP-2 nodes relative to the TRC nodes (Figures 3 and 4 respectively) is likely due to the increased memory bandwidths in the CSP-2 nodes (Figure 5, Table II). The accelerated drop in the performance for higher MPI ranks in Figures 7 and 8 is shown to be a consequence of the high communication time for higher MPI ranks as demonstrated in Figure 9. Direct modeling here interpolates the communication time from PingPong measurement raw data however the generalized performance model uses the linear model for communication events thus revealing in Figure 10 that the bulk of the internodal communication time is due to latency and not due to insufficient bandwidth.

IV. Discussion

These results show that cloud resources can provide comparable or even superior performance than a traditional cluster for realistic LBM workloads. However, a key factor in using these resources efficiently is balancing compute power against cost. In this work, we have laid the groundwork for the framework, shown in Figure 1, to optimize usage of cloud platforms for LBM hemodynamic simulations. When considering an range of different CSPs and instances for each CSP, we have presented two performance models for characterizing LBM compute loads. By quantifying the cost for memory access and halo exchange for a given cloud instance, a CSP Option Dashboard can be created for any new anatomical target. This Dashboard enables the user to select the preferred CSP and Type based on user-defined metrics. Maximizing time-to-solution may be necessary for situations where rapid turnaround is essential, but in most cases the trade-offs of cost and overall throughput must be taken into account. One metric we propose is to compare the relative throughput of different hardware configurations for a given simulation workload using our performance model:

$$r_{B,A} = \frac{T_{sim\text{-}A}}{T_{sim\text{-}B}} = \frac{MFLUPS_B}{MFLUPS_A} \tag{17}$$

2048 Cores - Aorta	TRC	CSP-2	CSP-2 EC
TRC	1.0000	0.8115	0.7282
CSP-2	1.2323	1.0000	0.8973
CSP-2 EC	1.3733	1.1144	1.0000

Fig. 11. A heatmap of relative value $r_{B,A}$ of performance of computing infrastructures using the performance of HARVEY running the aorta geometry on 2048 cores as predicted by the generalized performance model. B is read from left side and A from top.

By plotting the results as a heatmap as in Figure 11 one can quickly see the relative impact of hardware selection on throughput as well as the optimal hardware for a given simulation. If maximizing overall throughput is the goal, this alone could be the decisional metric. If minimizing cost is most important, one could weight these ratios by the relative cost of each instance. It is ultimately up to the end user to determine what is important to them and define an appropriate cost metric to fit.

The performance model prediction can then be optionally used to establish a limit on job specifications. For example,

the user could allow a 10% tolerance on the prediction and set a hard stop on the number of CPU hours allowed for that job or dollars spent or the user could predict the length of potential jobs then plan according to their budget and needs. A performance model-driven limit would help flag simulations that are vastly out of line with the prediction and provide the user some protection against inadvertent cost overruns. The performance model driven approach can also be used as a realistic measure of potential performance for an application such that one may attempt to optimize their code to achieve such performance using a combination of hardware limits and roofline models without aiming for performance bound by a single hardware limit's roofline that cannot actually be met. Our approach to performance modeling can be generalized to other stencil-based codes by counting memory accesses and communication events in those stencil codes. Roofline models for other hardware constraints (such as floating point operations and storage memory speed) can also be considered in the overall performance model either by an approximation such as by adding the theoretical runtime predicted by the roofline model or by a direct counting such as by tracking the simultaneous loads on each hardware unit and extracting runtime based on the roofline for each type of load. Using our model as a baseline, additional elements of runtime can be added then checked for their impact on the model's ability to predict experimental results. Following the results of this check the element can be added or discarded and the model can be deemed a sufficient or insufficient predictor of performance. Together this system of adding and checking can be used as a feedback loop system for refining predictions for stencil code performance.

This performance model's predictions are not exact however as with our assumptions we consider memory accesses and communication but ignore costs including time for floating point operations, and we model each task as having a constant share of the node's memory bandwidth which is not the case with imperfect load balancing. Further steps for improving this model would include integrating rooflines for floating point operations and tracking of concurrency of memory accesses however the model outlined here sufficiently matches the actual performance to act as a proof of concept. Our work included testing on Intel processors and while we expect the same assumptions to be true for AMD processors, future work will also need to be done to confirm this. Such a difference in processor hardware however is not expected to change the prediction power of our performance model.

The final step in the process is to compare the measured simulation times against the predictions of the performance model. There are multiple factors that can result in inaccurate estimates, from needing additional terms in the performance model to capture the computational workload of a given simulation to mismatch between measured hardware performance within simplified benchmarks and a full simulation workflow. It was also assumed here that cloud allocations are node based wherein the user is allocated all cores on a node. We did not encounter a case where we could not reserve all cores on the nodes we used; however, for use cases where cores are on shared nodes, memory bandwidth usage by other users on the node and the increases in internodal communication must be considered. Consideration for memory bandwidth sharing may be an assumption of full or partial usage of the other cores by other users. Storing all measured performance along with the estimated performance model prediction will be critical to iteratively refining the performance models to correctly capture retrospective values as well as predict future behavior with sufficient accuracy. Performance monitoring projects such as SONAR [34] are expected to be extremely useful in helping to automate and track the measured performance against model predictions.

V. Conclusion

We present a framework for optimizing cloud resource utilization for large-scale fluid dynamics simulations. We examined the impact of different cloud instance settings as compared to a conventional on premise system and presented both a direct and general performance model for predicting HPC performance of real workloads. We expect this work to lay important groundwork for efficient utilization of cloud infrastructures for production-scale blood flow simulations in the future.

Acknowledgment

Computing support was provided by the AWS Cloud Credits for Research program and the Microsoft Azure HPC + AI Collaboration Center in partnership with NVIDIA. This work was supported by the American Heart Association Predoctoral Fellowship 20PRE35211158 (M.V.), American Heart Association Postdoctoral Fellowship 903995 (C.J.), NSF 1943036 (J.A.), Microsoft Azure HPC + AI Collaboration Center (W.L.), and NIH U01CA253511 and 1DP1AG082343 (A.R). This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

References

[1] L. M. Itu, P. Sharma, and C. Suciu, Patient-specific hemodynamic computations: application to personalized diagnosis of cardiovascular pathologies. Springer, 2017.

[2] L. Athanasiou, F. R. Nezami, and E. R. Edelman, "Computational cardiology," IEEE journal of biomedical and health informatics, vol. 23, no. 1, pp. 4–11, 2018.

[3] C. A. Taylor, T. A. Fonte, and J. K. Min, "Computational fluid dynamics applied to cardiac computed tomography for noninvasive quantification of fractional flow reserve: scientific basis," Journal of the American College of Cardiology, vol. 61, no. 22, pp. 2233–2241, 2013.

[4] A. J. Graham, M. Orini, E. Zacur, G. Dhillon, H. Daw, N. T. Srinivasan, C. Martin, J. Lane, J. S. Mansell, A. Cambridge, et al., "Evaluation of ecg imaging to map hemodynamically stable and unstable ventricular arrhythmias," Circulation: Arrhythmia and Electrophysiology, vol. 13, no. 2, p. e007377, 2020.

[5] J. E. Betancourt, A. Noheria, D. Cooper, G. Orme, S. Sodhi, C. Steyers, and P. Cuculich, "Accuracy of cardioinsight noninvasive electrocardiographic imaging compared with invasive mapping for determining location of ventricular arrhythmias," Journal of the American College of Cardiology, vol. 73, no. 9S1, pp. 458–458, 2019.

[6] A. Gupta and D. Milojicic, "Evaluation of HPC applications on cloud," in 2011 Sixth Open Cirrus Summit, pp. 22–26, IEEE, 2011.

[7] C. Evangelinos and C. Hill, "Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on Amazons EC2," ratio, vol. 2, no. 2.40, pp. 2–34, 2008.

[8] A. Gupta, L. V. Kale, F. Gioachin, V. March, C. H. Suen, B.-S. Lee, P. Faraboschi, R. Kaufmann, and D. Milojicic, "The who, what, why, and how of high performance computing in the cloud," in 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, vol. 1, pp. 306–314, IEEE, 2013.

[9] P. Mvelase, H. Sithole, S. Masoka, and M. Bembe, "HPC in the cloud environment: Challenges, and theoretical analysis," in Proceedings of the International Conference on Scientific Computing (CSC), pp. 94–101, The Steering Committee of The World Congress in Computer Science, Computer . . . , 2018.

[10] D. Yimam and E. B. Fernandez, "A survey of compliance issues in cloud computing," Journal of Internet Services and Applications, vol. 7, no. 1, p. 5, 2016.

[11] M. A. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. Cunha, and R. Buyya, "HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges," ACM Computing Surveys (CSUR), vol. 51, no. 1, p. 8, 2018.

[12] A. P. Randles, V. Kale, J. Hammond, W. Gropp, and E. Kaxiras, "Performance analysis of the lattice Boltzmann model beyond Navier-Stokes," in Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, pp. 1063–1074, IEEE, 2013.

[13] S. Chen and G. D. Doolen, "Lattice Boltzmann method for fluid flows," Annual review of fluid mechanics, vol. 30, no. 1, pp. 329–364, 1998.

[14] A. Randles, E. W. Draeger, T. Oppelstrup, L. Krauss, and J. A. Gunnels, "Massively parallel models of the human circulatory system," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p. 1, ACM, 2015.

[15] T. Krüger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, and E. M. Viggen, "The lattice Boltzmann method," Springer International Publishing, vol. 10, pp. 978–3, 2017.

[16] J. Gounley and G. Dube, "LBM-Proxy-App." https://code.ornl.gov/j8g/lbm-proxy-app. Accessed: 2022-06-07.

[17] G. Herschlag, S. Lee, J. S. Vetter, and A. Randles, "GPU data access on complex geometries for D3Q19 Lattice Boltzmann method," in 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 825–834, 2018.

[18] M. Dabagh, P. Nair, J. Gounley, D. Frakes, L. F. Gonzalez, and A. Randles, "Hemodynamic and morphological characteristics of a growing cerebral aneurysm," NeurosurgicalFocus, vol. 47, no. 1, p. E13, 2019.

[19] B. Feiger, M. Vardhan, J. Gounley, M. Mortensen, P. Nair, R. Chaudhury, D. Frakes, and A. Randles, "Suitability of lattice Boltzmann inlet and outlet boundary conditions for simulating flow in image-derived vasculature," International Journal for Numerical Methods in Biomedical Engineering, vol. 35, no. 6, p. e3198, 2019.

[20] M. Vardhan, J. Gounley, S. J. Chen, E. C. Chi, A. M. Kahn, J. A. Leopold, and A. Randles, "Non-invasive characterization of complex coronary lesions," Scientific reports, vol. 11, no. 1, pp. 1–15, 2021.

[21] J. Latt, B. Chopard, O. Malaspinas, M. Deville, and A. Michler, "Straight velocity boundaries in the lattice Boltzmann method," Physical Review E, vol. 77, no. 5, p. 056703, 2008.

[22] J. Ames, D. F. Puleri, P. Balogh, J. Gounley, E. W. Draeger, and A. Randles, "Multi-gpu immersed boundary method hemodynamics simulations," Journal of Computational Science, vol. 44, July 2020.

[23] J. Habich, C. Feichtinger, H. Köstler, G. Hager, and G. Wellein, "Performance engineering for the lattice Boltzmann method on GPGPUs: Architectural requirements and performance results," Computers and Fluids, 2013.

[24] G. Wellein, T. Zeiser, G. Hager, and S. Donath, "On the single processor performance of simple lattice Boltzmann kernels," Computers and Fluids, vol. 35, no. 8-9, pp. 910–919, 2006.

[25] N. P. Tran, M. Lee, and D. H. Choi, "Memory-efficient parallelization of 3D lattice Boltzmann flow solver on a GPU," Proceedings - 22nd IEEE International Conference on High Performance Computing, HiPC 2015, pp. 315–324, 2016.

[26] A. P. Randles, V. Kale, J. Hammond, W. Gropp, and E. Kaxiras, "Performance analysis of the lattice boltzmann model beyond navier-stokes," in IEEE 27th International Symposium on Parallel and Distributed Processing, pp. 1063–1074, 2013.

[27] J. D. McCalpin et al., "Memory bandwidth and machine balance in current high performance computers," IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, vol. 1995, pp. 19–25, 1995.

[28] "Lawrence Livermore National Lab MPI performance." https://computing.llnl.gov/tutorials/mpi_performance/#MessageSize. Accessed: 2018-12-05.

[29] "Intel MPI benchmarks Github." https://github.com/intel/mpi-benchmarks. Accessed: 2018-12-02.

[30] N. M. Wilson, A. K. Ortiz, and A. B. Johnson, "The vascular model repository: A public resource of medical imaging data and blood flow simulation results," Journal of Medical Devices, vol. 7, no. 4, p. 040923, 2013.

[31] J. Xu, H. Fu, W. Shi, L. Gan, Y. Li, W. Luk, and G. Yang, "Performance tuning and analysis for stencil-based applications on POWER8 processor," ACM Transactions on Architecture and Code Optimization, vol. 15, no. 4, pp. 1–25, 2018.

[32] P. Bailey, J. Myre, S. D. Walsh, D. J. Lilja, and M. O. Saar, "Accelerating lattice boltzmann fluid flow simulations using graphics processors," in International Conference on Parallel Processing, pp. 550–557, 2009.

[33] G. Herschlag, S. Lee, J. S. Vetter, and A. Randles, "Analysis of gpu data access patterns on complex geometries for the d3q19 lattice boltzmann algorithm," IEEE Transactions on Parallel and Distributed Systems, 2021.

[34] "SONAR HPC performance-monitoring software stack." https://github.com/LLNL/sonar-driver.