Deploying and optimizing performance of a 3D hydrodynamic model on cloud

Fearghal O'Donncha*, Srikumar Venugopal*, Scott C. James†, Emanuele Ragnoli*

*IBM Research - Ireland

Email: feardonn@ie.ibm.com

†Baylor University

Abstract—Container-based cloud computing, as standardised and popularised by the open-source docker project has many potential opportunities for scientific application in high-performance computing. It promises highly flexible and available compute capabilities via cloud, without the resource overheads of traditional virtual machines. Further, productivity gains can be made by easy repackaging of images with additional developments, automated deployments, and version-control integrations. Nevertheless, the impact of container overhead and overlay network implementation and performance are areas that requires detailed study to allow for well-defined quality of service for typical HPC applications. This paper presents details on deploying the Environmental Fluid Dynamics Code (EFDC) on a container-based cloud environment. Results are compared to a bare metal deployment. Application-specific benchmarking tests are complemented by detailed network tests that evaluate isolated MPI communication protocols both at intra-node and inter-node level with varying degrees of self-contention. Cloud-based simulations report significant performance loss in mean run-times. A containerised environment increases simulation time by up to 50%. More detailed analysis demonstrates that much of this performance penalty is a result of large variance in MPI communication times. This manifests as simulation run-time variance on container cloud that hinders both simulation run-time and collection of well-defined quality-of-service metrics.

I. INTRODUCTION

High performance computing (HPC) is a central component of many academic and industrial institutions for both research and application-specific studies. Initially driven by increase in chip processor speeds, modern scientific and industrial codes rely on multi-core processing to achieve desired performance levels. Multi-core processors developed in conjunction with codes written to take advantage of parallel processing power. Consequently, HPC platforms evolved to suit the needs of scientific computing applications, including optimised network communication protocols, OS-specific libraries, and optimised hardware tuning based on communication, memory, and processor speed for a given application.

More recently, a shift to the utility computing model through cloud has greatly increased the availability of compute resources without the expense of setting up and maintaining a dedicated cluster. However, traditional cloud deployments have focused predominantly on web- and analytics-based applications whose resource requirements are different from HPC applications. HPC applications typically require low latency and high bandwidth inter-processor communication to max-

imise performance. This is usually provided by Infiniband, the most commonly used interconnect in modern HPC systems. In the case of cloud, the presence of a commodity interconnect (and the effects of OS-level virtualisation techniques) leads to communication becoming the barrier to achieving targeted parallel performance metrics.

Cloud computing and the requisite flexibility of deployment are typically provided through either virtual machines (VMs) or container-based virtualisation. VMs are the most common choice for Infrastructure as a service (IaaS) through, for example, Amazon EC2 and IBM SoftLayer. These allow customers to run their applications within a hosted VM environment. Further, the majority of Platform as a Service (PaaS) and Software as a Service (SaaS) solutions are built on IaaS with applications running inside VMs.

Container-based virtualisation has seen a rapid rise in popularity in the past three years. Largely driven by technologies such as the Docker project, containerisation aims to accelerate development and ease distribution and deployment of applications. Docker is an open-source platform for the management of Linux containers. Docker containers can be seen as extremely lightweight virtual machines that allow code to be run in isolation from other containers. Every Docker image starts from a base image, such as Ubuntu or RedHat. When users make changes to a container, instead of directly writing the changes to the image of the container, Docker adds an additional layer containing the changes to the image. By wrapping software in a complete filesystem that contains everything needed, the process ensures that the software will run the same regardless of the environment. The main advantage of containers over virtual machines is that they are typically much more lightweight. VMs operate by instantiating a guest operating system on which the application including the necessary binaries and libraries run. Containers, on the other hand, include the application and all its dependencies, but share the kernel with other containers, running as isolated processes in user space on the host operating system. This avoids many of the overheads of VMs because each VM has its own instance of the kernel. These overheads manifest through increased memory requirements and much longer start up times, because VMs require booting an entire kernel compared to containers that only involve launching a process on the host kernel.

A large body of research considers the performance of VMs for scientific computing applications. Younge et al. [1] investigated the performance of a number of virtualisation technologies using a set of standard HPC benchmarking tests. Results demonstrate performance comparable to bare metal but with significantly higher variance. They did not however consider inter-node computation with MPI-based simulations restricted to intra-node level. Jackson et al. [2] conducted a more comprehensive evaluation comparing conventional HPC platforms to Amazon EC2 when running a range of synthetic and industrial application codes. Results indicate that EC2 was between 6 and 20 times slower than running on bare metal. Performance was strongly correlated with communications with communication-intensive codes experiencing the most performance degradation. Further, results demonstrated large variability in performance that the authors attributed to the shared nature of the virtualised environment, the network interconnect, and differences in the underlying non-virtualised environment.

Considering that the widespread adoption of container-based virtualisation is more recent, the literature is not as rich. Felter et al. [3] conducted the most comprehensive comparison of performance achieved on container and VMs. They used a suite of benchmark workloads that stressed CPU, memory, storage, and networking resources. Results demonstrated that in almost all cases containers provided equal or better performance than VMs. The performance of containers on top of hypervisor-based virtual machines in a distributed cloud system was investigated by Kratzke [4] through a series of simple data-exchange experiments. In particular, the paper analysed the effects of overlay networks and encryption layers on performance. Overlay networks serve as a lookup database to provide a logical IP address for containers on top of the infrastructure. The additional hypervisor layer impacted performance significantly. Containers deployed on bare metal introduced a performance drop of approximately 10% in data transfer, while the additional overlay and encryption layer incurred a 75% drop in performance.

II. MOTIVATION

Container-based virtualisation as a replacement for VMs has seen an explosion in interest in the last few years, having seen adoption in some key enterprises and recognition from major software vendors. Containers have many advantages to support both PaaS and SaaS solutions, through reduced resource consumption (as opposed to VMs or other virtualisation technologies) and much easier deployment and iteration [5]. In the scientific community, efficient container deployments make collaboration and extending results easier by allowing flexible porting of software to any platform through a self-contained docker image [6]. Emanating from this, there is real potential for cloud-based deployments of traditional HPC-based applications without the punitive overheads that VMs impose on both performance (computational throughput) and productivity (ease of deployment).

Many organisations have found that using the cloud has clear benefits over in-house hardware. Indeed, shifting what was a capital expense to an operational expense has many advantages, including instant availability and the ability to rapidly scale up (and down) [7]. Cloud offers scientists the possibility of almost unlimited storage and instantly available and scalable computing resources.

In this paper, we interrogate the performance of a widely used hydro-environmental code, Environmental Fluid Dynamics Code (EFDC), in a container-based cloud environment. We quantify performance both intra- and inter-node, considering both the floating point rate achieved and the effects of communication at both the shared-memory intra-node level and between nodes. The fundamental objective of this paper is to determine the viability of deploying a scientific code on a cloud platform while achieving performance comparable with a commodity nodes cluster.

III. METHODOLOGY

The paper focuses on performance and scalability of EFDC on a cloud platform. EFDC is a public-domain, open-source, modelling package for simulating three-dimensional flow, transport, and biogeochemical processes in surface-water systems. The model is specifically designed to simulate estuaries and subestuarine components (tributaries, marshes, wet and dry littoral margins) and has been applied to a wide range of environmental studies including surface-current processes [8], [9], suspended sediment transport [10], [11], water-quality investigations [12], [13], marine renewable energy [14], [15], and canopy flow processes [16], [17]. It is currently used by universities, research organisations, governmental agencies, and consulting firms.

The equations that form the basis for the EFDC hydrodynamic model are based on the continuity and Reynolds-averaged Navier-Stokes equations. These equations are resolved on a discretised grid using a combination of finite volume and finite difference techniques. A notable feature of the numerical scheme is the separation of the solution scheme into external- and internal-mode equations: the external mode equations solve for surface elevation and depth-averaged velocities using a semi-implicit numerical scheme, while the internal mode solves for the fully three-dimensional velocity components using a fractional-step scheme combining an implicit step for the vertical shear terms with an explicit discretisation for all other terms; the depth-averaged velocities computed in the external mode serve as boundary conditions to the computation of the layer-integrated velocities. This approach solves the two-dimensional, depth-averaged momentum equations implicitly in time, thereby allowing the models barotropic time step to equal the baroclinic time step. The primary limitation of this semi-implicit method is the reliance on an elliptic solver (preconditioned conjugate gradient) to solve implicitly for the free-surface elevation. This has traditionally posed a problem for efficient projection of model codes onto parallel computers due to the inherent non-

local conditions of the solver [18]; this issue is addressed in further detail in the next section.

Parallelisation of the code using both the MPI and OpenMP paradigms is presented elsewhere [19], [20] and will be briefly discussed here with emphasis on the challenges faced. EFDC is a Fortran 77 code originally designed for deployment on vector computers as opposed to distributed systems. The code was configured to achieve a degree of parallelisation on shared-memory processors through directives specific to vectorised architectures. Parallelisation was achieved using a rectilinear domain-decomposition approach that decomposed the full domain into a number of subdomains for parallel processing. A novel load-balancing technique seeks equal distribution of land/water cells in any domain (within the constraints of rectilinear partitioning) along with the ability to mark some tiles as inactive. By adding ghost layers of halo points to each of the subdomains, the communication step can be accomplished by sending values to the ghost layers of neighbouring processors. Each subdomain then proceeds independently with synchronisation of the solution achieved through ghost layers at the end of each timestep. Two types of inter-machines communications keep the computations consistent with the sequential code: (1) point-to-point communication to update halo values among neighbouring machines and (2) global communication at each iteration of the preconditioned conjugate gradient solver required for computing surface elevations.

Performance tests were conducted on an IBM idataplex compute server. Each node consists of two 2.93-GHz six-core Intel Westmere processors, twelve cores total, forming a single NUMA (Non-Uniform Memory Architecture) unit with 128 GB of RAM and 1 GigE network interconnect. Cloud-based scaling tests were conducted by creating a RedHat 7.1 image from the Docker repository and instantiating on the Linux server. The EFDC application and its dependencies were built inside this image and deployed on the server.

Experiments were conducted on an idealised test-case scenario to reduce application-specific performance issues that may not translate to other case studies (in particular, processor load imbalances that may arise in case studies with irregular land/water mixes making it difficult to balance processors under a different number of MPI process configurations). The test case consisted of a rectangular domain of $50 \times 50 \times 20$ grid cells assigned to each processor; i.e. the global problem size increased as number of processors increased to maintain an equal problem size for each processor to solve. This avoids the strong scaling dilemma where communication swamps computation as number of processors continue to increase and the size of the problem to be solved remains fixed. Weak scaling provides insight into the complexity of translating an application to many-core parallelism. In an embarrassingly parallel application, where there is no communication between adjacent domains, an application could scale to any number of cores without performance degradation. In many practical applications, however, simulation time increases as communication overhead increases if computation expense remains fixed.

Fig. 1: EFDC scaling plot when deployed on both bare-metal (BM) and container-based (CR) infrastructures. (a) presents mean time per timestep of the simulation while (b) presents median time per timestep. Results are average from 200-timestep simulations for allocations across the 5 available nodes investigated for each of the two Configurations (BM1, CR1 and BM2, CR2).

IV. RESULTS

The first stage of the analysis considered a direct comparison of model run times when deploying on a container-based versus a Linux environment. To provide a comprehensive assessment of intra- and inter-node performance, we allocate MPI processes across nodes according to two different configurations. Configuration 1 deploys processes across as few nodes as possible with a maximum of 12 processes on any node. This minimises inter-node, network communication. Configuration 2 distributes processes equally across all available nodes, i.e., when running 5 MPI processes, then a single process was run on each node, etc. Figure 1 presents run-time metrics for these configurations in containerised and bare-metal environments. We present both mean and median time-per-timestep for a 200-timestep simulation. A cursory analysis indicates significant differences between median and mean time metrics for the cloud simulations. At the higher core count (greater than 40 cores) the mean time of the cloud simulations is 40-60% greater than for bare metal. These punitive effects are not present if we consider median simulation times with differences of less than 10% between cloud and bare metal simulations. Clearly, this discrepancy is a result of higher variability in simulation time for the cloud simulation as expressed in the mean values.

Figure 2 shows boxplots of the 200 timestep simulation metrics for both bare metal and container-based simulations across the range of cores. These statistics corroborate what is observed in the preceding figure, namely, that container-based simulations are subject to large timing variations. Analysis

Fig. 2: Boxplots of simulation time per timestep versus number of cores for (a) bare-metal and (b) container-based deployments for a 200-timestep simulation. The red line indicates the median, the blue rectangle represent the 25th and 75th percentile, and red crosses indicate outliers from the interquartile range.

suggests this to be a result of large variation in the time it takes for MPI operations to conclude. As discussed above, the application contains two communication intensive routines: the majority of data exchange occurs at the end of each timestep when each process does a neighbour-to-neighbour exchange of ghost-point data. The second communication-intensive section is during the solution for surface elevation, which is done by a semi-implicit numerical discretisation, using an iterative conjugate gradient scheme that requires repeated MPI reduction calls to converge.

Figure 3 presents timing metrics for individual components of the application that contain MPI-intensive routines, namely MPI_send/MPI_rcv and MPI_reduce. These plots indicate

Fig. 3: Time metrics for communication-intensive components of the application along with the total run-time. Bare metal timings are denoted by the black curves while red represents cloud metrics. The plus symbol denotes timings for the neighbour-to-neighbour exchange that is necessary at the end of each timestep for synchronisation, while the dashed curves represent the computation time of the conjugate gradient solver that incorporates repeated MPI-reduction operations.

that the neighbour-to-neighbour exchange is a major component of the total run-time at higher core counts. Further, it is a source of much of the slowdown evident in the container-based deployments compared to bare metal simulations. This is predominantly due to high variability in neighbour-to-neighbour exchange timing as evident by comparing mean and median time metrics.

The high cost of neighbour-to-neighbour exchange routines is a result of both data transfer rates and application packaging of variables to be transferred. To minimise MPI latency, all variables (a total of 8 2D and 3D arrays) are packaged into a single array before sending to a neighbour. This allows the data exchange to proceed through a single MPI call. A relatively expensive component of the data exchange is a result of the data storage scheme used in EFDC. To reduce array size, the 2D arrays in the I and J directions are mapped to a single vector representing only cells that contain water (i.e., land cells are excluded from computation). This reduces all 2D arrays to 1D with reduced dimensions (land cells are excluded). The alternative is to maintain a (I, J) indexing that loops over all cells (both water and land) with appropriate masking applied to the land cells. The vectorised approach adopted in EFDC is more efficient in terms of memory storage and computational FLOPS; however, the indirect addressing storage scheme is inefficient when accessing variables that are not stored adjacently in the vectorised array, as the data in the ghost points on the edge of each domain will be.

Figure 4 presents equivalent metrics when deploying the

Fig. 4: Time metrics for communication-intensive components of the application along with the total run times in the bare-metal environment up to 200 timesteps. The plus symbol denotes timings for the neighbour-to-neighbour exchange that is necessary at the end of each timestep for synchronisation, while the dashed curve represent the computation time of the conjugate gradient solver that incorporates repeated MPI reduction operations. The total solution time is denoted by the solid black curve.

application at higher core counts. Interestingly, the ratio of communication-to-computation time does not change significantly beyond approximately 50 cores. This again is a result of the neighbour-to-neighbour exchange being a highly localised routine characterised by a five-point stencil of domains (i.e., domain to the East, West, North and South of the one being considered). The addition of extra model domains at a spatially removed point does not impact performance of that domain (assuming computation allocation remains fixed).

To further investigate the cost of communication within the simulation, we conducted a range of tests of network latency and bandwidth for both the bare-metal and container environments. Figure 5 shows the performance of a point-to-point communication, i.e. a communication between two nodes, with MPI in the test cluster for different message sizes. The latency and bandwidth were obtained using a Ping-Pong test with two nodes by sending packets of data back and forth to quantify latency and throughput of the interconnector. Communication is restricted to two MPI processes on two different nodes. Figure 5(a) presents timing metrics for MPI communication in both a bare-metal and container environments while 5(b) presents the associated network bandwidths.

To quantify performance with increasing core counts, we used standard network benchmarking tests from the HPCC benchmarking suite to understand variability in network performance as a function of both core count and environment (bare metal or container). The effective bandwidth benchmark

Fig. 5: Comparison of point-to-point communication performance between two nodes in bare-metal and cloud environments. Network latency versus message size is presented in (a), while (b) presents bandwidth plotted against message size.

(b_eff) measures the accumulated bandwidth of the communication network of distributed computing systems. Several message sizes, communication patterns, and methods are used [21] and the results averaged to provide a best estimate of actual performance in typical application code. It differs from the Ping-Pong test in that all processes are sending messages to neighbours in parallel. The algorithm uses an average to take into account that short and long messages are transferred with different bandwidth values in real application scenarios. The result of this benchmark is a single number called the effective bandwidth. Figure 6 presents the computed effective bandwidth for the bare-metal and cloud environments. To provide further insight into the source of performance discrepancies, Table I presents the average Ping-Pong latency and bandwidth and the randomly ordered ring latency and bandwidth.

Fig. 6: The recorded effective bandwidth of the bare metal and cloud environment against different core counts as computed by the HPCC benchmark algorithm.

TABLE I: Interconnect and data-exchange performance comparisons of bare-metal and cloud environments.

System	Latency μ s	Bandwidth MB/s	RandRing Lat. μ s	RandRing bw MB/s
Bare metal	3.38	2664.8	27.2	1233.5
Container	21.4	1531.7	48.8	744.2

These results demonstrate a clear difference between interconnect and data-exchange performance of both platforms. Focusing on a simple Ping-Pong test between two nodes, containers report significantly lower bandwidth capacity; this is particularly the case beyond 10^5 bytes (100 kB). Both latency and bandwidth in the container environment are reduced to about 25% of the bare-metal environment. Bandwidth and latency performance metrics from the HPCC benchmarks are arguably more representative of actual performance in a typical application characterised by multiple type of MPI calls, both neighbour-to-neighbour and global, exacerbated by self contention. Metrics from Table I present latency considering only a simple Ping-Pong test and also a random-ring Ping-Pong test that replicates simultaneous communication both in random and natural ring patterns (each process sends to a direct neighbour). Latency benchmarks show slowdowns of between 75% and 500% in contested and uncontested Ping-Pong tests, respectively, while reported bandwidth is reduced by 40%.

V. DISCUSSION & CONCLUSIONS

The objective of this paper is an assessment of the viability of deploying a typical HPC application code on a container-based cloud environment with a typical configuration. The application presented is typical of many advection-diffusion-based codes used in a wide variety of geoscience

and probabilistic-based computational domains. It is characterised by complex numerical routines combining explicit and implicit schemes while parallelisation requires intensive, periodic communication to maintain fidelity of solution.

Results demonstrate a considerable penalty attached to deploying on cloud. In almost all scenarios investigated, interconnect performance was poorer on cloud. Simple neighbour-to-neighbour send/receive tests reported increased latency and decreased network bandwidth. Introducing self-contention by increasing the number of MPI processes did not change performance trends with latency and bandwidth performance trailing bare metal by up to 75% and 40%, respectively.

Performance tests when deploying EFDC on bare-metal and cloud environments report non-negligible performance drops due to cloud virtualisation. The containerised environment increases simulation time by up to 50%. More detailed analysis demonstrates that much of this performance penalty results from large variance in MPI communication times. This manifests in large variance in simulation run times on container cloud. Further, it impacts on quality-of-service statistics that are central to cloud computing provision based on specific and predefined performance metrics that allows a user to select the optimal platform for their needs.

Cloud computing has many opportunities and advantages for scientific applications and tightly coupled HPC codes. Our analysis suggests that achieving performance comparable to a bare-metal cluster is possible intermittently. However, high performance variability does not make it possible to guarantee quality of service comparable to bare metal. Nevertheless, container technology is a relatively new addition to the general cloud sector. The Docker project is the recipient of considerable funding and is investing in furthering the technology, including improving network capabilities. Any improvement in network performance will manifest as improved metrics for the type of application code assessed in this paper

ACKNOWLEDGMENT

Elements of this research has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No. 671603

REFERENCES

- [1] A. J. Younge, R. Henschel, J. T. Brown, G. Von Laszewski, J. Qiu, and G. C. Fox, "Analysis of virtualization technologies for high performance computing environments," in *Cloud Computing (CLOUD), 2011 IEEE International Conference*, 2011, pp. 9–16.
- [2] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference*, 2010, pp. 159–168.
- [3] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium*, 2015, pp. 171–172.
- [4] N. Kratzke, "About microservices, containers and their underestimated impact on network performance," in *CLOUD COMPUTING 2015 : The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization*, 2015, pp. 165–169.

- [5] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support PaaS," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, 2014, pp. 610–614.
- [6] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.
- [7] D. Eadline, "Moving HPC to the cloud AD-MIN magazine." [Online]. Available: <http://www.admin-magazine.com/HPC/Articles/Moving-HPC-to-the-Cloud>
- [8] F. O'Donncha, E. Ragnoli, S. Zhuk, F. Suits, and M. Hartnett, "Surface flow dynamics within an exposed wind-driven bay: Combined HF radar observations and model simulations," in *OCEANS 2012*, 2012, pp. 1–8.
- [9] F. O'Donncha, M. Hartnett, S. Nash, L. Ren, and E. Ragnoli, "Characterizing observed circulation patterns within a bay using HF radar and numerical model simulations," *J. Mar. Syst.*, vol. 142, pp. 96–110, 2015.
- [10] P. Thanh, M. D. Grace, and S. C. James, "Sandia National Laboratories Environmental Fluid Dynamics Code: Sediment transport user manual," SAND2008-5621, Livermore, CA., Tech. Rep., 2008.
- [11] S. C. James, C. A. Jones, M. D. Grace, and J. D. Roberts, "Advances in sediment transport modelling," *J. Hydraul. Res.*, vol. 48, no. 6, pp. 754–763, 2010.
- [12] S. C. James and V. Boriah, "Modeling algae growth in an open-channel raceway," *J. Comput. Biol.*, vol. 17, no. 7, pp. 895–906, 2010.
- [13] S. C. James, V. Janardhanam, and D. T. Hanson, "Simulating PH effects in an algal-growth hydrodynamics model," *J. Phycol.*, vol. 49, no. 3, pp. 608–615, 2013.
- [14] S. C. James, E. Seetho, C. Jones, and J. Roberts, "Simulating environmental changes due to marine hydrokinetic energy installations," in *OCEANS, 2010*, Seattle, Washington, 2010, pp. 1–10.
- [15] S. C. James, J. Barco, E. Johnson, J. Roberts, and S. Lefantzi, "Verifying marine-hydro-kinetic energy generation simulations using SNL-EFDC," in *OCEANS, 2011*, Kona, Hawaii, 2011, pp. 1–9.
- [16] F. O'Donncha, M. Hartnett, and D. R. Plew, "Parameterizing suspended canopy effects in a three-dimensional hydrodynamic model," *J. Hydraul. Res.*, vol. 53, no. 6, pp. 714–727, 2015.
- [17] F. O'Donncha, S. C. James, N. O'Brien, and E. Ragnoli, "Numerical modelling study of the effects of suspended aquaculture farms on tidal stream energy generation," in *OCEANS, 2015*, Genova, Italy, 2015, pp. 1–7.
- [18] S. Griffies, C. Boning, F. Bryan, E. Chassignet, R. Gerdes, H. Hasumi, A. Hirst, A. Treguier, and D. Webb, "Developments in ocean climate modelling," *Ocean Modelling*, vol. 2, no. 3, pp. 123–192, 2000.
- [19] F. O'Donncha, E. Ragnoli, and F. Suits, "Parallelisation study of a three-dimensional environmental flow model," *Comput. Geosci.*, vol. 64, pp. 96–103, 2014.
- [20] F. O'Donncha, S. C. James, N. O'Brien, and E. Ragnoli, "Parallelisation of hydro-environmental model for simulating marine current devices," in *OCEANS, 2015*, Washington D.C., 2015, pp. 1–7.
- [21] R. Rabenseifner and A. E. Koniges, "The parallel communication and I/O bandwidth benchmarks: b_eff and b_eff_io," in *Proceedings of 43rd Cray User Group Conference*, Indian Wells, California, USA, 2001.