Procedia Computer Science
Volume 80, 2016, Pages 52–62
ICCS 2016. The International Conference on Computational Science

Modeling and Implementation of an Asynchronous Approach to Integrating HPC and Big Data Analysis [*]

Yuankun Fu[1], Fengguang Song[1], and Luoding Zhu[2]

[1] Department of Computer Science
Indiana University-Purdue University Indianapolis
{fuyuan, fgsong}@iupui.edu
[2] Department of Mathematical Sciences
Indiana University-Purdue University Indianapolis
luozhu@iupui.edu

Abstract
With the emergence of exascale computing and big data analytics, many important scientific applications require the integration of computationally intensive modeling and simulation with data-intensive analysis to accelerate scientific discovery. In this paper, we create an analytical model to steer the optimization of the end-to-end time-to-solution for the integrated computation and data analysis. We also design and develop an intelligent data broker to efficiently intertwine the computation stage and the analysis stage to practically achieve the optimal time-to-solution predicted by the analytical model. We perform experiments on both synthetic applications and real-world computational fluid dynamics (CFD) applications. The experiments show that the analytic model exhibits an average relative error of less than 10%, and the application performance can be improved by up to 131% for the synthetic programs and by up to 78% for the real-world CFD application.

Keywords: High performance computing, analytical modeling, integration of exascale computing and big data

1 Introduction

Alongside with experiments and theories, computational modeling/simulation and big data analytics have established themselves as the critical third and fourth paradigms in scientific discovery [3, 5]. Today, there is an inevitable trend towards integrating the two stages of computation and data analysis together. The benefits of combining them are significant: 1) the overall end-to-end time-to-solution can be reduced considerably such that interactive or real-time scientific discovery becomes feasible; 2) the traditional one-way communication (from

[*]This material is based upon research supported by the Purdue Research Foundation and by the NSF Grant No. 1522554.

doi:10.1016/j.procs.2016.05.297

computation to analysis) becomes bidirectional to enable guided computational modeling and simulation; and 3) modeling/simulation and data-intensive analysis are essentially complementary to each other and can be used in a virtuous circle to amplify their collective effect.

However, there are many challenges to integrate computation with analysis. For instance, how to minimize the cost to couple computation and analysis, and how to design an effective software system to enable and facilitate such an integration. In this paper, we focus on building an analytical model to estimate the overall execution time of the integrated computation and data analysis, and designing an intelligent data broker to intertwine the computation stage and the analysis stage to achieve the optimal time-to-solution predicted by the analytical model.

To fully interleave computation with analysis, we propose a fine-grain-block based asynchronous parallel execution model. The execution model utilizes the abstraction of pipelining, which is widely used in computer architectures [11]. In a traditional scientific discovery, a user often executes the computation, stores the computed results to disks, then reads the computed results, and finally performs data analysis. From the user's perspective, the total time-to-solution is the sum of the four execution times. In this paper, we rethink of the problem by using a novel method of fully asynchronous pipelining. With the asynchronous pipeline method (detailed in Section 2), a user input is divided into fine-grain blocks. Each fine-grain block goes through four steps: computation, output, input, and analysis. As shown in Figure 1, our new end-to-end time-to-solution is equal to the maximum of the the computation time, the output time, the input time, and the analysis time (i.e., the time of a single step only). Furthermore, we build an analytical model to predict the overall time-to-solution to integrate computation and analysis, which provides developers with an insight into how to efficiently combine them.

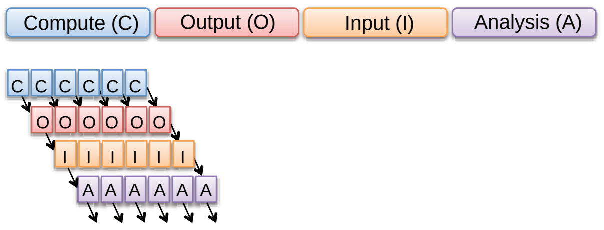


Figure 1: Comparison between the traditional process (upper) and the new fully asynchronous pipeline method (lower).

Although the analytical model and its performance analysis reveal that the corresponding integrated execution can result in good performance, there is no software available to support the online tight coupling of analysis and computation at run time. To facilitate the asynchronous integration of computation and analysis, we design and develop an I/O middleware, named *Intelligent DataBroker*, to adaptively prefetch and manage data in both secondary storage and main memory to minimize the I/O cost. This approach is able to support both in-situ (or in memory) data processing and post-processing where initial dataset is preserved for the entire community (e.g., the weather community) for subsequent analysis and verification. The computation and analysis applications are coupled up through the DataBroker. DataBroker consists of two parts: a DataBroker producer in the compute node to send data, and a DataBroker consumer in the analysis node to receive data. It has its own runtime system to provide dynamic scheduling, pipelining, hierarchical buffering, and prefetching. The paper introduces the design of the current prototype of DataBroker briefly.

We performed experiments on BigRed II (a Cray system) with a Lustre parallel file system at Indiana University to verify the analytical model and compare the performance of the traditional process, an improved version of the traditional process with overlapped data writing and computation, and our fully asynchronous pipeline approach. Both synthetic applications and real-world computational fluid dynamics (CFD) applications have been implemented with the prototype of DataBroker. Based on the experiments, the difference between the actual time-to-solution and the predicted time-to-solution is less than 10%. Furthermore, by using DataBroker, our fully asynchronous method is able to outperform the improved traditional method by up to 78% for the real-world CFD application.

In the rest of the paper, Section 2 introduces the analytical model to estimate the time-to-solution. Section 3 presents the DataBroker middleware to enable an optimized integration approach. Section 4 verifies the analytical model and demonstrates the speedup using the integration approach. Finally, Section 5 presents the related work and Section 6 summarizes the paper and specifies its future work.

2 Analytical Modeling

2.1 The Problem

This paper targets an important class of scientific discovery applications which require combining extreme-scale computational modeling/simulation with large-scale data analysis. The scientific discovery consists of computation, result output, result input, and data analysis. From a user's perspective, the actual time-to-solution is the end-to-end time from the start of the computation to the end of the analysis. While it seems to be a simple problem with only four steps, different methods to execute the four steps can lead to totally different execution time. For instance, traditional methods execute the four steps sequentially such that the overall time-to-solution is the sum of the four times.

In this section, we study how to unify the four seemingly separated steps into a single problem and build an analytical model to analyze and predict how to obtain optimized time-to-solution. The rest of the section models the time-to-solution for three different methods: 1) the traditional method, 2) an improved version of the traditional method, and 3) the fully asynchronous pipeline method.

2.2 The Traditional Method

Figure 2 illustrates the traditional method, which is the simplest method without optimizations (next subsection will show an optimized version of the traditional method). The traditional method works as follows: the compute processes compute results and write computed results to disks, followed by the analysis processes reading results and then analyzing the results.

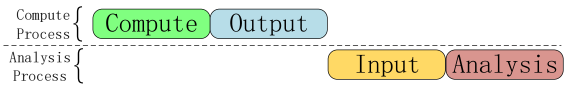


Figure 2: The traditional method.

The time-to-solution ($t2s$) of the traditional method can be expressed as follows:

$$T_{t2s} = T_{comp} + T_o + T_i + T_{analy},$$

where T_{comp} denotes the parallel computation time, T_o denotes the output time, T_i denotes the input time, and T_{analy} denotes the parallel data analysis time. Although the traditional method can simplify the software development work, this formula reveals that the traditional model can be as slow as the accumulated time of all the four stages.

2.3 Improved Version of the Traditional Method

The traditional method is a strictly sequential workflow. However, it can be improved by using multi-threaded I/O libraries, where I/O threads are deployed to write results to disks meanwhile new results are generated by the compute processes. The other improvement is that the user input is divided into a number of fine-grain blocks and written to disks asynchronously. Figure 3 shows this improved version of the traditional method. We can see that the output stage is now overlapped with the computation stage so that the output time might be hidden by the computation time.

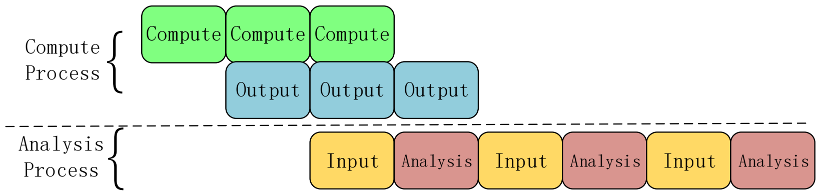


Figure 3: An improved version of the traditional method.

Suppose a number of P CPU cores are used to compute simulations and a number of Q CPU cores are used to analyze results, and the total amount of data generated is D. Given a fine-grain block of size B, there are $n_b = \frac{D}{B}$ blocks. Since scalable applications most often have good load balancing, we assume that each compute core computes $\frac{n_b}{P}$ blocks and each analysis core analyzes $\frac{n_b}{Q}$ blocks. The rationale behind the assumption of load balancing is that a huge number of fine-grain parallel tasks (e.g., $n_b >> P$) will most likely lead to an even workload distribution among a relatively small number of cores.

Our approach uses the time to compute and analyze individual blocks to estimate the time-to-solution of the improved traditional method. Let t_{comp}, t_o, t_i, and t_{anal} denote the time to compute a block, write a block, read a block, and analyze a block, respectively. Then we can get the parallel computation time $T_{comp} = t_{comp} \times \frac{n_b}{P}$, the data output time $T_o = t_o \times \frac{n_b}{P}$, the data input time $T_i = t_i \times \frac{n_b}{Q}$, and the parallel analysis time $T_{analy} = t_{analy} \times \frac{n_b}{Q}$. The time-to-solution of the improved version is defined as follows:

$$T_{t2s} = \max(T_{comp}, T_o, T_i + T_{analy}).$$

The term $T_i + T_{analy}$ is needed because the analysis process still reads data and then analyzes data in a sequence. Note that this sequential analysis step can be further parallelized, which results in a fully asynchronous pipeline execution model (see the following subsection).

2.4 The Fully Asynchronous Pipeline Method

The fully asynchronous pipeline method is designed to completely overlap computation, output, input, and analysis such that the time-to-solution is merely one component, which is either computation, data output, data input, or analysis. Note that the other three components will not be observable in the end-to-end time-to-solution. As shown in Figure 4, every data block goes through four steps: compute, output, input, and analysis. Its corresponding time-to-solution can be expressed as follows:

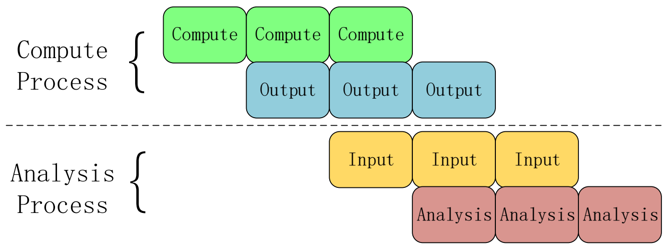


Figure 4: The fully asynchronous pipeline method.

$$\begin{aligned} T_{t2s} &= \max(T_{comp}, T_o, T_i, T_{analy}) \\ &= \max(t_{comp} \times \frac{n_b}{P}, t_o \times \frac{n_b}{P}, t_i \times \frac{n_b}{Q}, t_{analy} \times \frac{n_b}{Q}). \end{aligned}$$

The above analytical model provides an insight into how to achieve an optimal time-to-solution. When $t_{comp} = t_o = t_i = t_{analy}$, the pipeline is able to proceed without any stalls and deliver the best performance possible. On the other hand, the model can be used to allocate and schedule computing resources to different stages appropriately to attain the optimal performance.

3 Design of DataBroker for the Fully Asynchronous Method

To enable the fully asynchronous pipeline model, we design and develop a software prototype called *Intelligent DataBroker*. The interface of the DataBroker prototype is similar to Unix's pipe, which has a writing end and a reading end. For instance, a computation process will call DataBroker.write(block_id, void* data) to output data, while an analysis process will call DataBroker.read(block_id) to input data. Although the interface is simple, it has its own runtime system to provide pipelining, hierarchical buffering, and data prefetching.

Figure 5 shows the design of DataBroker. It consists of two components: a *DataBroker producer component* in the compute node to send data, and a *DataBroker consumer component* in the analysis node to receive data. The producer component owns a producer ring buffer and one or multiple producer threads to process output in parallel. Each producer thread looks up the I/O-task queues and uses priority-based scheduling algorithms to transfer data to destinations in a streaming manner. A computational process may send data to an analysis process via two possible paths: message passing by the network, or file I/O by the parallel file system. Depending on the execution environment, it is possible that both paths are available and used to speed up the data transmission time.

The DataBroker consumer is co-located with an analysis process on the analysis node. The consumer component will receive data from the computation processes, buffer data, and prefetch and prepare data for the analysis application. It consists of a consumer ring buffer and one or multiple prefetching threads. The prefetching threads are responsible for making sure there are always data blocks available in memory by loading blocks from disks to memory. Since we assume a streaming-based data analysis, the prefetching method can use the technique of *read ahead* to prefetch data efficiently.

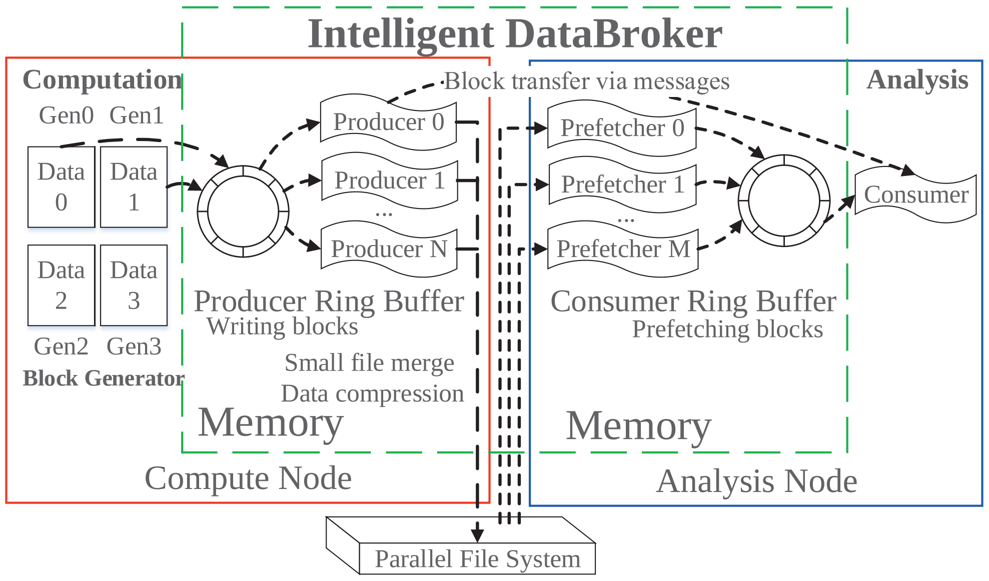


Figure 5: Architecture of the DataBroker middleware for coupling computation with analysis in a streaming pipeline manner. DataBroker consists of a producer component on a compute node, and a consumer component on an analysis node.

4 Experimental Results

We perform experiments to verify the accuracy of the analytical model and to evaluate the performance of the fully asynchronous pipeline method, respectively. For each experiment, we collect performance data from two different programs: 1) a synthetic application, and 2) a real-world computational fluid dynamics (CFD) application. All the experiments are carried out on BigRed II (a Cray XE6/XK7 system) configured with the Lustre 2.1.6 distributed parallel file system at Indiana University. Every compute node on BigRed II has two AMD Opteron 16-core Abu Dhabi CPUs and 64 GB of memory, and is connected to the file system via 56-Gb FDR InfiniBand which is also connected to the DataDirect Network SFA12K storage controllers.

4.1 Synthetic and Real-World Applications

The synthetic application consists of a computation stage and an analysis stage. To do the experiments, we use 32 compute nodes to execute the computation stage, and use two different numbers of analysis nodes (i.e., 2 analysis nodes and 32 analysis nodes) to execute the analysis stage, respectively. We launch one process per node. Each computation process randomly generates a total amount of 1GB data (chopped to small blocks) and writes the data to the DataBroker producer. Essentially the computation processes only generate data, but not perform any computation. At the same time, each analysis process reads data from its local DataBroker consumer and computes the sum of the square root of the received data block for a number of iterations. The mapping between computation processes and analysis processes is static. For instance, if there are 32 computation processes and 2 analysis processes, each analysis process will process data from a half of the computation processes.

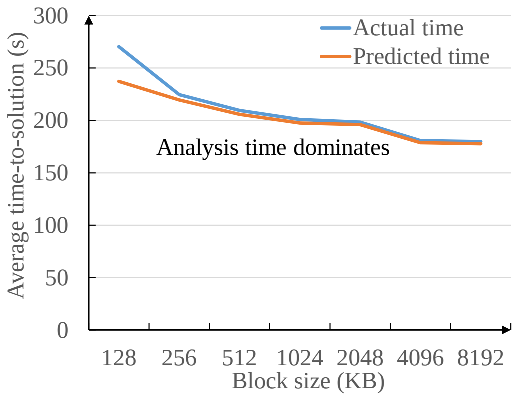
Our real-world CFD application, provided by the Mathematics Department at IUPUI [15], computes the 3D simulations of flow slid of viscous incompressible fluid flow at 3D hydrophobic microchannel walls using the lattice Boltzmann method [8, 10]. This application is written in ANSI C and MPI. We replaced all the file write functions in the CFD application by our DataBroker API. The CFD simulation is coupled with an data analysis stage, which computes a series of statistical analysis functions at each fluid region for every time step of the simulation. Our experiment takes as input a 3D grid of 512 × 512 × 256, which is distributed to different computation processes. Similar to the synthetic experiments, we also run 32 computation processes on 32 compute nodes while running different numbers of analysis processes. For each experiment, we execute it four times and display their average in our experimental results.

4.2 Accuracy of the Analytical Model

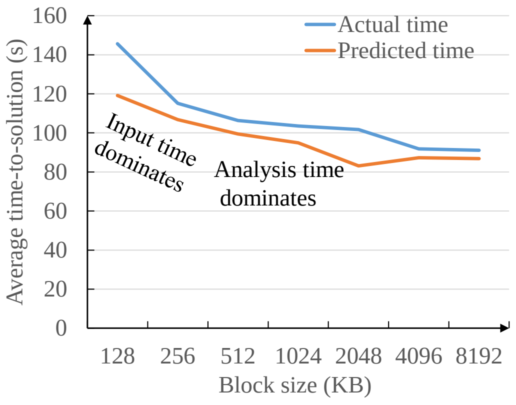
We experiment with both the synthetic application and the CFD application to verify the analytical model. Our experiments measure the end-to-end time-to-solution on different block sizes ranging from 128KB to 8MB. The experiments are designed to compare the time-to-solution estimated by the analytical model with the actual time-to-solution to show the model's accuracy.

Figure 6 (a) shows the actual time and the predicted time of the synthetic application using 32 compute nodes and 2 analysis nodes. For all different block sizes, the analysis stage is the largest bottleneck among the four stages (i.e., computation, output, input, and analysis). Hence, the time-to-solution is essentially equal to the analysis time. Also, the relative error between the predicted and the actual execution time is from 1.1% to 12.2%, and on average 3.9%. Figure 6 (b) shows the actual time and the predicted time for the CFD application. Different from the synthetic application, its time-to-solution is initially dominated by the input time when the block size is 128KB, then it becomes dominated by the analysis time from 256KB to 8MB. The relative error of the analytical model is between 4.7% and 18.1%, and on average 9.6%.

The relative error is greater than zero because our analytical model ignores the pipeline startup and drainage time, and there is also a small amount of pipeline idle time and jitter time during the real execution. Please note that each analysis process has to process the computed results from 16 computation processes.



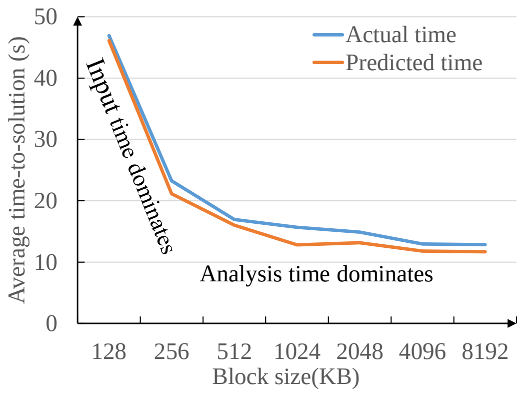
(a) Synthetic experiments



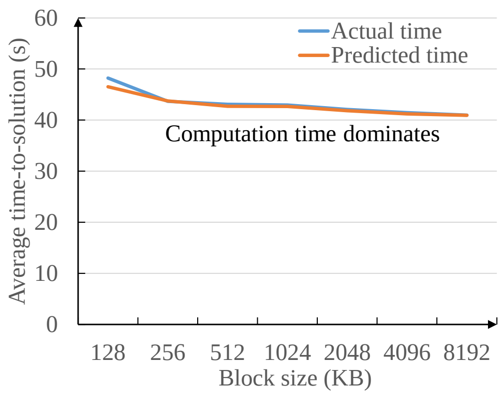
(b) CFD application

Figure 6: Accuracy of the analytical model for the fully asynchronous pipeline execution with 32 compute nodes and 2 analysis nodes.

Figure 7 (a) shows the performance of the synthetic application that uses 32 compute nodes and 32 analysis nodes. When the block size is equal to 128KB, the input time dominates the time-to-solution. When the block size is greater than 128KB, the data analysis time starts to dominate the time-to-solution. The turning point in the figure also verifies the bottleneck switch (from the input stage to the analysis stage). The predicted time and the actual time are very close to each other and have an average relative error of 9.1%. Similarly, Figure 7 (b) shows an relative error of 0.9% for the CFD application that also uses 32 compute nodes and 32 analysis nodes.



(a) Synthetic experiments



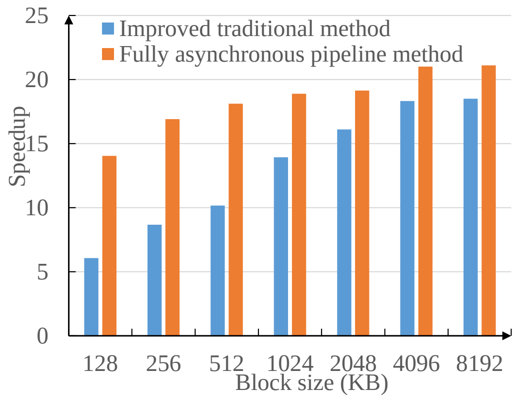
(b) CFD application

Figure 7: Accuracy of the analytical model for the fully asynchronous pipeline execution with 32 compute nodes and 32 analysis nodes.

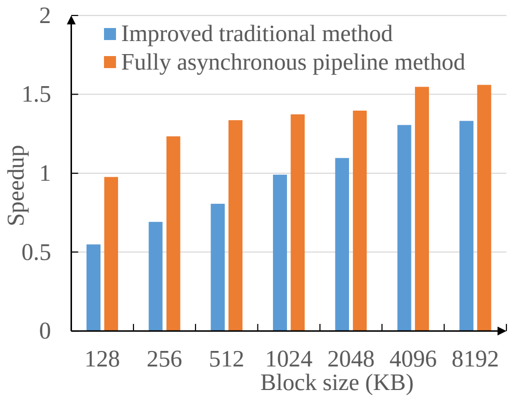
4.3 Performance Speedup

Besides testing the analytical model, we also conduct experiments to evaluate the performance improvement by using the fully asynchronous pipeline method. The experiments compare three different approaches (i.e., three implementations) to executing the integrated computation and analysis: 1) the traditional method, 2) the improved version of the traditional method which builds upon fine-grain blocks and overlaps computation with data output, and 3) the fully asynchronous pipeline method based on DataBroker. Each of the three implementations takes the same input size and is compared with each other in terms of wall clock time.

Figure 8 (a) and (b) show the speedup of the synthetic application and the real-world CFD application, respectively. Note that the baseline program is the traditional method (i.e., speedup=1). The data in subfigure (a) shows that the improved version of the traditional method can be up to 18 times faster than the traditional method when the block size is equal to 8MB. It seems to be surprising, but by looking into the collected performance data, we discover that reading two 16GB files by two MPI process simultaneously is 59 times slower than reading a collection of small 8MB files by the same two MPI processes. This might be because two 16GB files are allocated to the same storage device, while a number of 8MB files are distributed to multiple storage devices. On the other hand, the fully asynchronous pipeline method is faster than the improved traditional method by up to 131% when the block size is equal to 128KB. Figure 8 (b) shows the speedup of the CFD application. We can see that the fully asynchronous method is always faster (up to 56%) than the traditional method whenever



(a) Synthetic experiments



(b) CFD application

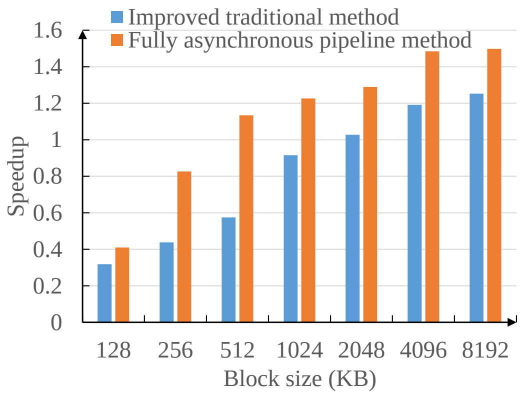
Figure 8: Performance comparison between the traditional, the improved, and the DataBroker-based fully asynchronous methods using 32 compute nodes and 2 analysis nodes.

the block size is larger than 128KB. The small block size of 128KB does not lead to improved performance because writing small files to disks can incur significant file system overhead and cannot reach the maximum network and I/O bandwidth. Also, the fully asynchronous method is consistently faster than the improved traditional method by 17% to 78%.

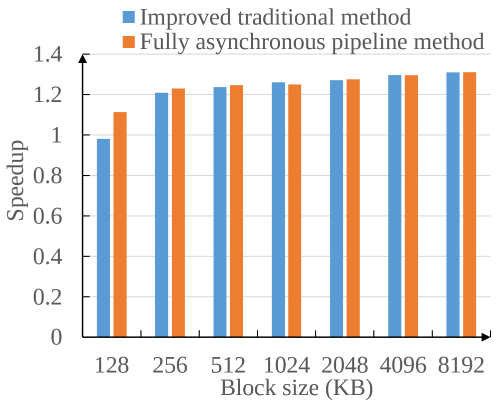
Figure 9 (a) shows the speedup of the synthetic application that uses 32 compute nodes and 32 analysis nodes. We can see that the fully asynchronous pipeline method is 49% faster than the traditional method when the block size is equal to 8MB. It is also 24% faster than the improved transitional method when the block size is equal to 4MB. Figure 9 (b) shows the speedup of the CFD application with 32 compute nodes and 32 analysis nodes. Both the fully asynchronous pipeline method and the improved traditional method are faster than the traditional method. For instance, they are 31% faster with the block size of 8MB. However, the fully asynchronous pipeline method is almost the same as the improved method when the block size is bigger than 128KB. This is because the specific experiment's computation time dominates its time-to-solution so that both methods' time-to-solution is equal to the computation time, which matches our analytical model.

5 Related Work

To alleviate the I/O bottleneck, significant efforts have been made to in-situ data processing. GLEAN [13] deploys data staging on analysis nodes of an analysis cluster to support in-situ analysis by using sockets. DataStager [2], I/O Container [6], FlexIO [14], and DataSpaces [7] use RDMA to develop data staging services on a portion of compute nodes to support in-situ analysis. While in-situ processing can totally eliminate the I/O time, real-world applications can be tight on memory, or have a severe load imbalance between the computational and analysis processes (e.g., faster computation has to wait for analysis that scales poorly). Moreover, it does not automatically preserve the initial dataset for the entire community for further analysis and verification. Our proposed DataBroker supports both in-situ and post processing. It also provides a unifying approach, which takes into account computation, output, input, and analysis as a whole to optimize the end-to-end time-to-solution.



(a) Synthetic experiments



(b) CFD application

Figure 9: Performance comparison between the traditional, the improved, and the DataBroker-based fully asynchronous methods using 32 compute nodes and 32 analysis nodes.

There are efficient parallel I/O libraries and middleware such as MPI-IO [12], ADIOS [1], Nessie [9], and PLFS [4] to enable applications to adapt their I/O to specific file systems. Our data-driven DataBroker is complementary to them and is designed to build upon them to maximize the I/O performance. Depending on where the bottleneck is (e.g., computation, output, input, analysis), DataBroker can perform in-memory analysis or file-system based (out-of-core) analysis adaptively, without stalling the computation processes.

6 Conclusion and Future Work

To facilitate the convergence of computational modeling/simulation and the big data analysis, we study the problem of integrating computation with analysis in both theoretical and practical ways. First, we use the metric of the time-to-solution of scientific discovery to formulate the integration problem and propose a fully asynchronous pipeline method to model the execution. Next, we build an analytical model to estimate the overall time to execute the asynchronous combination of computation and analysis. In addition to the theoretical foundation, we also design and develop an intelligent DataBroker to help fully interleave the computation stage and the analysis stage.

The experimental results show that the analytical model can estimate the time-to-solution with an average relative error of less than 10%. By applying the fully asynchronous pipeline model to both synthetic and real-world CFD applications, we can increase the performance of the improved traditional method by up to 131% for the synthetic application, and up to 78% for the CFD application. Our future work along this line is to utilize the analytical model to appropriately allocate resources (e.g., CPUs) to the computation, I/O, and analysis stages to optimize the end-to-end time-to-solution by eliminating the most dominant bottleneck.

References

[1] H. Abbasi, J. Lofstead, F. Zheng, K. Schwan, M. Wolf, and S. Klasky. Extending I/O through high performance data services. In *IEEE International Conference on Cluster Computing and*

Workshops (CLUSTER'09), pages 1–10. IEEE, 2009.

[2] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. Datastager: Scalable data staging services for petascale applications. *Cluster Computing*, 13(3):277–290, 2010.

[3] G. Aloisioa, S. Fiorea, I. Foster, and D. Williams. Scientific big data analytics challenges at large scale. *Proceedings of Big Data and Extreme-scale Computing*, 2013.

[4] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate. PLFS: A checkpoint filesystem for parallel applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC'09)*, page 21. ACM, 2009.

[5] J. Chen, A. Choudhary, S. Feldman, B. Hendrickson, C. Johnson, R. Mount, V. Sarkar, V. White, and D. Williams. Synergistic challenges in data-intensive science and exascale computing. *DOE ASCAC Data Subcommittee Report, Department of Energy Office of Science*, 2013.

[6] J. Dayal, J. Cao, G. Eisenhauer, K. Schwan, M. Wolf, F. Zheng, H. Abbasi, S. Klasky, N. Podhorszki, and J. Lofstead. I/O Containers: Managing the data analytics and visualization pipelines of high end codes. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '13, pages 2015–2024, Washington, DC, USA, 2013. IEEE Computer Society.

[7] C. Docan, M. Parashar, and S. Klasky. DataSpaces: An interaction and coordination framework for coupled simulation workflows. *Cluster Computing*, 15(2):163–181, 2012.

[8] Z. Guo and C. Shu. *Lattice Boltzmann method and its applications in engineering.* World Scientific, 2013.

[9] J. Lofstead, R. Oldfield, T. Kordenbrock, and C. Reiss. Extending scalability of collective IO through Nessie and staging. In *Proceedings of the sixth workshop on Parallel Data Storage*, pages 7–12. ACM, 2011.

[10] P. Nagar, F. Song, L. Zhu, and L. Lin. LBM-IB: A parallel library to solve 3D fluid-structure interaction problems on manycore systems. In *Proceedings of the 2015 International Conference on Parallel Processing*, ICPP'15. IEEE, September 2015.

[11] D. A. Patterson and J. L. Hennessy. *Computer organization and design: the hardware/software interface.* Newnes, 2013.

[12] R. Thakur, W. Gropp, and E. Lusk. On implementing MPI-IO portably and with high performance. In *Proceedings of the sixth workshop on I/O in parallel and distributed systems*, pages 23–32. ACM, 1999.

[13] V. Vishwanath, M. Hereld, M. E. Papka, R. Hudson, G. C. Jordan IV, and C. Daley. In situ data analysis and I/O acceleration of FLASH astrophysics simulation on leadership-class system using GLEAN. In *Proc. SciDAC, Journal of Physics: Conference Series*, 2011.

[14] F. Zheng, H. Zou, G. Eisenhauer, K. Schwan, M. Wolf, J. Dayal, T.-A. Nguyen, J. Cao, H. Abbasi, and S. Klasky. FlexIO: I/O middleware for location-flexible scientific data analytics. In *IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 320–331. IEEE, 2013.

[15] L. Zhu, D. Tretheway, L. Petzold, and C. Meinhart. Simulation of fluid slip at 3D hydrophobic microchannel walls by the lattice Boltzmann method. *Journal of Computational Physics*, 202(1):181–195, 2005.