A Performance Model of In-Situ Techniques

Yi Ju *, Nicolas Vidal‖, Adalberto Perez †, Ana Gainaru ‖, Fred Suter‖,
Stefano Markidis †, Philipp Schlatter †¶, Scott Klasky‖, Erwin Laure *
*Max Planck Computing and Data Facility {yi.ju, erwin.laure}@mpcdf.mpg.de
†KTH Royal Institute of Technology {adperez, markidis, pschlatter}@kth.se
¶Friedrich-Alexander-Universität Erlangen-Nürnberg philipp.schlatter@fau.de
‖ Oak Ridge National Laboratory {vidaln, gainarua, suterf, klasky}@ornl.gov

***Abstract*—The computational capacity of High-Performance Computing (HPC) systems increases continuously with the rapid development of central processing units (CPUs) and graphic processing units (GPUs), while the in-/output (IO) subsystem develops relatively slowly and storage capacity is also limited. Data-intensive applications, which are designed to leverage the high computational capacity of HPC resources, typically generate a considerable amount of data for post-processing visualizations and data analytics. The limited IO speed and storage space could lead to constraints in the actual performance of these applications and, therefore, scientific discovery. In-situ techniques, where data is visualized/analysed while still in memory rather than through disk, can contribute to alleviating these problems as they can reduce or even fully avoid data writing/reading through the IO subsystem to/from storage. However, the overall efficiency of in-situ techniques crucially depends on the characteristics of both the in-situ tasks and the applications, and the resource distribution among them. Therefore, choosing the right in-situ approach (synchronous, asynchronous, or hybrid) and resource allocation is essential to minimize overhead and maximize the benefits of concurrent execution. In this paper, we present a performance model of in-situ techniques to find the most beneficial in-situ approach and the preferred resource configuration. We verify the high accuracy of our approach with over 6800 measurements and provide use cases with different applications.**

***Index Terms*—performance model, HPC, in-situ, CPU, GPU**

I. Introduction

The peak performance of High-Performance Computing (HPC) systems increases continuously with the rapid development of central processing units (CPUs) and graphic processing units (GPUs). Data-intensive research can benefit from this fast development by leveraging the computational capacity for large-scale simulation or data analytics. For instance, for Computational Fluid Dynamics (CFD), HPC systems implement computationally expensive numerical methods to analyse fluid flow problems. Molecular Dynamics (MD) is another crucial research that uses the high computational power of HPC systems to simulate and uses the results to describe the physical movements of atoms and molecules. However, the input/output (IO) subsystem is also developing relatively slowly compared to the computational power. Conventionally, the results generated by an application are stored via the IO subsystem to the storage on the HPC systems, and data analysis applications read these data back via the IO subsystems from storage. As storage space is usually restricted, this may limit the frequency of storing the results for further analysis, and the performance of the IO subsystem may limit the actual performance of the applications and, thus, scientific discovery. In-situ techniques, where data is processed while still in memory rather than written to disk, can help to reduce or even avoid data transfer through the IO subsystem to/from storage so they can potentially solve these problems. In-situ techniques can be categorized into three types based on whether executing the in-situ task would interrupt the application. The *synchronous* approach (Fig. 1a) pauses the application during the execution of the in-situ task, which runs on the same resources as the application and then continues to execute the application. In the *asynchronous* approach (Fig. 1b), separate computing resources are allocated to the application and the in-situ task, and the application transfers the required data to the in-situ task, allowing both to proceed concurrently. Lastly, in the *hybrid* approach (Fig. 1c), part of the in-situ task is executed in a synchronous manner, while the remaining parts are executed on distinct computing resources in an asynchronous manner.

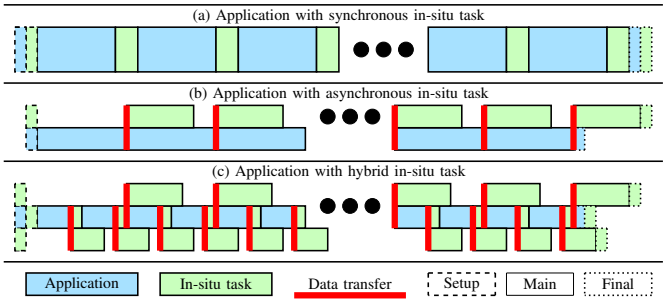


Fig. 1: Illustration of applications with synchronous, asynchronous, and hybrid in-situ tasks.

In-situ techniques can reduce the data traffic and allow frequent in-situ visualization and analysis to be integrated into both GPU-accelerated and CPU-based applications, but one needs to know the behaviour of both, the application and in-situ task to choose the most beneficial approach. [16], [17] For CPU-based applications with computationally cheap in-situ task(s), the synchronous approach is preferred; for CPU-based applications with computationally expensive and poorly scalable in-situ tasks, the asynchronous approach is preferred as it allows different resource allocations tailored to the respective needs and scalability; for CPU-based applications with in-situ tasks consisting of different parts with various frequencies and computational cost, the hybrid approach is preferred; for GPU-based applications, asynchronous and hybrid approaches could significantly improve performance and optimize the

2377-5750/25/$31.00 ©2025 IEEE
DOI 10.1109/PDP66500.2025.00036

use of system resources by executing the in-situ task on the (often) underutilized CPUs on GPU nodes. However, it is still necessary to conduct experiments to explore the design and configuration space to further quantify the most beneficial in-situ approach and runtime configuration.

Performance models have valuable insights into parallel applications. They can help to avoid laborious and expensive experiments to analyze the application [20]. In this paper, we develop a performance model of in-situ techniques, which we use to select the most beneficial in-situ approach and configuration. The paper's specific contributions are:

1) performance models of synchronous, asynchronous, and hybrid in-situ approaches derived from performance models of the applications and the integrated in-situ task;
2) high accuracy of the performance model of in-situ techniques verified with over 6800 experimental data;
3) a method to predict the preferred approach and runtime configuration based on the performance models and the example prediction in real-world use cases.

The remainder of the paper is organized as follows: Section II contains a summary of related work on in-situ techniques and performance models; Section III introduces the performance models of in-situ techniques; Section IV presents the experimental setups and accuracy of the performance models; Section V includes real use cases; and finally, Section VI summarizes and discusses the results of this paper.

II. Related Work

In-situ visualization is one of the most common in-situ tasks and usually based on the Visualization Toolkit (VTK) data format [21], with tools like VisIt with Libsim [10], [18], ParaView with Catalyst [4] and SENSEI [5]. As the name suggested, VTK focuses mainly on visualization but might require a deep copy. The Adaptable IO System (ADIOS/ADIOS2) [13], [19], a higher-level IO abstraction, also provides functionalities for in-situ processing. In addition to the VTK support, ADIOS/ ADIOS2 also supports various data formats, making it more efficient for in-situ tasks other than visualization. Therefore, we use the ADIOS2 library for our work.

Shu et al. [22] used performance models of in-situ workflows for autotuning but focused on asynchronous in-situ approaches, only. On the contrary, our research also includes synchronous and more complex hybrid in-situ approaches, and our performance model of in-situ techniques can suggest the preferred in-situ approach to decrease the development cost for the in-situ workflow in addition to only predicting the best runtime configuration. *Gainaru et al.* [11] used the total cost in node hours to study the impact of data staging in in-situ techniques. However, their study is only based on theoretical performance models, which could not represent the performance of real-world HPC applications and in-situ tasks. Performance models that better depict real-world cases include e.g.: PALM [23], which generates models by requiring users to annotate the source code with performance expressions; *Jayakumar et al.* [15] compare application kernels with a database of kernels with known behaviour to classify performance characteristics; *Hoefler et al.* [6] generate multi-parameter performance models online, but only with restricted search space and limited diversity of models.

Compared to the performance models mentioned above, we use two general forms of performance models, introduced by *Calotoui et al.* [8], [9] and developed by *Ritter et al.* [20]: ***A Performance Model Normal Form (PMNF)*** is a performance model with a single model parameter (Equation 1). It allows a search space for the function representing the set of measurements. The exponents i_k and j_k are chosen from the rational number set $I,\ J \subset\ \mathbb{Q}$. The number of terms, n, defines the discrete model search space. From the previous studies, a simple selection for I and J to model an application is sufficient. (e.g. $n = 2, I = \{0, 1/4, \cdots, 3, \}, J = \{0, 1, 2\}$) [9].

$$f(x) = \sum_{k=1}^{n} c_k \cdot x^{i_k} \cdot \log_2^{j_k}(x) \tag{1}$$

A Normal Form for Multiple Parameters (NFMP), Equation 2, allows combining the influences of m parameters. It is an expansion from PMNF, allowing the combination of a number of m of parameters in each of the n terms that are summed up to form the model. The exponents i_{k_l} and j_{k_l}, respectively, can be defined the same as in PMNF. [8], [20]

$$f(x_1, \cdots, x_m) = \sum_{k=1}^{n} c_k \cdot \prod_{l=1}^{m} x_l^{i_{k_l}} \cdot \log_2^{j_{k_l}}(x_l) \tag{2}$$

III. Methodology

In this section, we explain how to derive the performance models of in-situ approach from the separate performance models of applications and in-situ tasks.

A. Applications' and In-Situ Tasks' Performance Models

We separate the execution of the application (g) and in-situ task (ψ) into three phases: initialization, main, and finalization phases. The performance models of execution time can then be expressed with Equations 3. The frequency of performing in-situ task is v; the total number of simulation steps is n. The initialization phase ($*_{init}$) includes data distribution and memory allocation. The main phase ($*_{main}$) consists of n simulation steps, where each iteration often takes the same time. Based on this assumption, the execution time of this part is expressed as the sum of the execution time of n steps. The last phase is the finalization phase ($*_{final}$), in which the application performs the closing (e.g. freeing memory). Compared to the main phase, the initialization and finalization phases take a relatively short time.

$$\begin{aligned} g(\vec{x}) &= g_{init}(\vec{x}) + \ \sum^{n} g_{main}(\vec{x}) + g_{final}(\vec{x}) \\ \psi(\vec{x}) &= \psi_{init}(\vec{x}) + \ \sum^{n} v\ \psi_{main}(\vec{x}) + \psi_{final}(\vec{x}) \end{aligned} \tag{3}$$

The input variable $\vec{x}$ depends on the parameters necessary for the performance model. If the performance model (PMNF) only depends on one parameter, $\vec{x}$, is a scalar. If the performance model (NFMP) depends on h parameters, $\vec{x}$ is a h-dimensional vector. e.g., when both the numbers of CPUs and of GPUs influence the performance of GPU-accelerated applications, the input variable is a 2-dimensional vector.

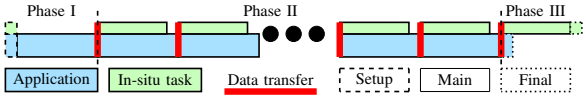


Fig. 2: Illustration of phases of asynchronous in-situ technique.

B. In-Situ Techniques' Performance Models

The performance model of the **synchronous in-situ technique** (φ_{sync}) can be expressed with Equation 4. The application and in-situ tasks are executed by the same computing resources, so only one set of parameters ($\vec{x}$) is sufficient. The 1^{st} line in Equation 4 shows the initialization of the synchronous approach consisting of the initialization of the application (g_{init}) and the in-situ task (ψ_{init}); the 2^{nd} line shows the main phase represented by the sum of the n steps of the simulation (g_{main}) and vn times the in-situ task (ψ_{main}); the 3^{rd} line shows the finalization consisting of the finalization of the application (g_{final}) and the in-situ task (ψ_{final}).

$$\begin{aligned}\varphi_{sync}(\vec{x}) = {} & g_{init}(\vec{x}) + \psi_{init}(\vec{x}) \\ & + \sum^{n}\big(g_{main}(\vec{x}) + v\ \psi_{main}(\vec{x})\big) \\ & + g_{final}(\vec{x}) + \psi_{final}(\vec{x})\end{aligned} \tag{4}$$

Fig. 2 shows the three phases of the **asynchronous in-situ technique**. Its performance model (φ_{async}) can be expressed with Equation 5. We need one parameter set ($\vec{x_1}$) for the application and the other set ($\vec{x_2}$) for the asynchronous in-situ task to represent the corresponding parameters influencing their performance.

$$\begin{aligned}\varphi_{async}(\vec{x_1},\vec{x_2}) = {} & \max\big(g_{init}(\vec{x_1}) + \sum^{1/v} g_{main}(\vec{x_1}), \psi_{init}(\vec{x_2})\big) \\ & + \sum^{n-1/v} \max\big(g_{main}(\vec{x_1}), v\ \psi_{main}(\vec{x_2})\big) \\ & + \max\big(g_{final}(\vec{x_1}),\ \psi_{main}(\vec{x_2}) + \psi_{final}(\vec{x_2})\big) \\ & + \sum^{v\times n} \varphi_{comm}(\vec{x_1},\vec{x_2})\end{aligned} \tag{5}$$

During the initialization phase (Phase I), the first group of computing resources execute the initialization of the application (g_{init}) and the first $1/v$ steps of the application's main phase (g_{main}), while the rest only execute the initialization of the in-situ task (ψ_{init}) (the 1^{st} line of Equation 5). Often, in this phase, the first $1/v$ steps of the application's main phase are dominant. During the main phase (Phase II) represented as the 2^{nd} line of Equation 5, the group of computing resources execute $(n - 1/v)$ steps of the application's main phase (g_{main}), while the rest concurrently execute the $(vn-1)$ steps of the in-situ task's main phase (ψ_{main}). The execution time of this phase is equal to the longer one. During the finalization phase (Phase III) represented as the 3^{nd} line of Equation 5, the first group of computing resources only execute the finalization (g_{final}), while the rest execute the last step of the in-situ task's main phase (ψ_{main}) and the finalization (ψ_{final}). Typically, the last step of the in-situ task's main phase is dominant. In addition to the execution time, we introduce $v \times n$ times of communication between the application and the in-situ task (the 4^{th} line of Equation 5). The communication time, apart from the size of the transferred data, is influenced by the number of resources for the application and the in-situ task, as this will determine the communication pattern. For the application on GPUs and in-situ tasks on CPUs, the communication also includes the additional data transfer between GPUs and CPUs. Among these three phases, the Phase II would often take the most of the total execution time because the in-situ techniques enable frequent in-situ task execution along the long execution. Therefore, this phase becomes the main optimization target to achieve the optimal execution time and resource usage of the asynchronous in-situ approach. When the total computing resources are fixed, especially when the application and the in-situ task are both executed on CPUs, the more computing resources are assigned to the application, the fewer can be assigned to the in-situ task. Thus, the execution time of the in-situ task might increase when the execution time of the application decreases. In this case, the optimal point would appear when $1/v$ steps of the application's main phase take the same amount of time as the one step of the in-situ task's main phase. When we exploit underused CPUs on GPU nodes where the application is mostly executed on GPUs, the best performance would appear when $1/v$ steps of the application's main phase take the same time as or even less time than the one step of the in-situ task. With this, we could predict the number of CPUs required by the in-situ tasks.

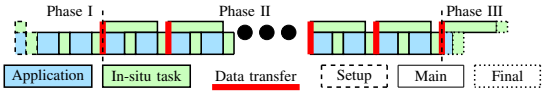


Fig. 3: Illustration of phases of hybrid in-situ technique.

For the **hybrid in-situ technique**, we divide the in-situ task's phase into two parts (ψ_1 and ψ_2). The first part would be executed synchronously with frequency, v_1 , and the second part would be executed asynchronously with frequency v_2 as shown in Equation 6.

$$\begin{aligned}\psi(\vec{x}) = {} & \psi_{1\ init}(\vec{x}) + \psi_{2\ init}(\vec{x}) \\ & + \sum^{n}\Big(v_1\ \psi_{1\ main}(\vec{x}) + v_2\ \psi_{2\ main}(\vec{x})\Big) \\ & + \psi_{1\ final}(\vec{x}) + \psi_{2\ final}(\vec{x})\end{aligned} \tag{6}$$

Typically, the asynchronous part is executed less frequently than the synchronous part. Fig. 3 shows three phases of the simplest hybrid approach. Accordingly, the performance model can be expressed with Equation 7. Similarly, we use two sets of parameters, $\vec{x_1}$ and $\vec{x_2}$, to represent the corresponding parameters influencing the performance.

$$\begin{aligned}&\varphi_{hybrid}(\vec{x_1},\vec{x_2}) = \max\big(g_{init}(\vec{x_1}) + \psi_{1\ init}(\vec{x_1}) \\ &+ \sum^{1/v_2}\big(g_{main}(\vec{x_1}) + v_1\ \psi_{1main}(\vec{x_1})\big), \psi_{2\ init}(\vec{x_2})\big) \\ &+ \sum^{n-1/v_2} \max\big(g_{main}(\vec{x_1}) + v_1\psi_{1main}(\vec{x_1}), v_2\psi_{2main}(\vec{x_2})\big) \\ &+ \max\big(g_{final}(\vec{x_1}) + \psi_{1\ final}(\vec{x_1}), \psi_{2\ main}(\vec{x_2}) + \psi_{2\ final}(\vec{x_2})\big) \\ &+ \sum^{v_2\times n} \varphi_{comm}(\vec{x_1},\vec{x_2})\end{aligned} \tag{7}$$

During the initialization phase (Phase I) (the 1^{st} and 2^{nd} lines of Equation 7), the first part of computing resources execute the initialization of the application (g_{init}) and synchronous part of the in-situ task ($\psi_{1\ init}$), the first $1/v_2$ steps of the application's main phase (g_{main}), and the first v_1/v_2 steps of the in-situ task's synchronous main phase ($\psi_{1\ main}$) (often dominant), while the rest only execute the initialization of the in-situ task ($\psi_{2\ init}$). During the main phase (Phase II) (the 3^{rd} line of Equation 7), the first part of computing

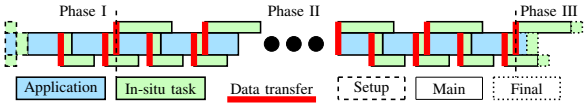


Fig. 4: Illustration of phases of hybrid in-situ technique with three parts.

resources execute $(n - 1/v_2)$ steps of the application's main phase (g_{main}) and $(v_1 n - v_1/v_2)$ steps of the in-situ task's synchronous main phase ($\psi_{1\ main}$), while the rest concurrently execute the $(v_2 n - 1)$ steps of the in-situ task's asynchronous main phase ($\psi_{2\ main}$). During the finalization phase (Phase III) represented as the 4^{th} line of Equation 7, the first part of computing resources only execute the finalizations (g_{final} and $\psi_{1\ final}$), while the rest execute the last step of the in-situ task's asynchronous main phase ($\psi_{2\ main}$) and the finalization ($\psi_{2\ final}$) (often dominant). In addition to the execution, we also introduce $v_2 n$ times extra communication between the in-situ task's synchronous and asynchronous parts, including GPU-CPU data exchange (the 5^{th} line of Equation 7). Similar to the asynchronous approach, among these three phases of the hybrid approach, the main phase (Phase II) typically consumes the majority of the total execution time. As a result, optimizing this phase becomes paramount to achieving optimal execution time and resource utilization with the hybrid in-situ approach.

We use Equation 8 to generalize the performance model of the in-situ task. The in-situ task are separated into m parts (ψ_k). The k^{th} part of the in-situ task can be executed with frequency v_k synchronously or asynchronously.

$$\begin{aligned}\psi(\vec{x}) = &\textstyle\sum_{k=1}^{m} \psi_{k\ init}(\vec{x}) \\ &+ \textstyle\sum_{k=1}^{m} \sum^{n} v_k\ \psi_{k\ main}(\vec{x}) + \sum_{k=1}^{m} \psi_{k\ final}(\vec{x})\end{aligned} \tag{8}$$

Fig. 4 illustrates one in-situ task with three parts with different frequencies integrated. One part is executed synchronously, and the other two asynchronously. We summarize the performance model of the in-situ technique with Equation 9. We separated the computational resources into $\lambda + 1$ groups: one group for the application and m_s synchronous parts of the in-situ task, and λ groups for asynchronous parts. The l^{th} resources ($l \in \{1, \cdots, \lambda\}$) can be represented with $\vec{x_{a_l}}$, and the numbers of asynchronous parts of in-situ tasks executed on the corresponding resources are m_1.

$$\begin{aligned}\varphi(\vec{x_s},&\vec{x_{a_1}}, \cdots, \vec{x_{a_\lambda}}) = \varphi_{init}(\vec{x_s}, \vec{x_{a_1}}, \cdots, \vec{x_{a_\lambda}}) \\ &+ \textstyle\sum^{n-1/v_{min}} \max\big(g_{main}(\vec{x_s}) + \sum_{k=1}^{m_s} v_k\ \psi_{k\ main}(\vec{x_s}), \\ &\quad \textstyle\sum_{k=1}^{m_1} v_k \psi_{k\ main}(\vec{x_{a_1}}), \cdots, \sum_{k=1}^{m_\lambda} v_k \psi_{k\ main}(\vec{x_{a_\lambda}})\big) \\ &+ \varphi_{final}(\vec{x_s}, \vec{x_{a_1}}, \cdots, \vec{x_{a_\lambda}}) \\ &+ \textstyle\sum_{l=1}^{\lambda} \sum_{k=1}^{m_\lambda} \sum^{v_k \times n} \varphi_{comm}(\vec{x_s}, \vec{x_{a_l}})\end{aligned} \tag{9}$$

The relation between the numbers of in-situ task parts in Equation 8 and Equation 9 should be $m = m_s + \sum_{l=1}^{\lambda} m_l$ For the asynchronous parts of the in-situ task, the separate computing resources have to wait for the application to generate the necessary data. The last steps of in-situ tasks cannot overlap with the application. However, these initialization (φ_{init} on the 1^{st} line and finalization φ_{final} on the 4^{th} line of Equation 9) are often neglectable compared to the total execution time. It is beneficial to optimize the runtime configuration to improve the main phase (Phase II in Fig. 4), where all the computing resources for the asynchronous part of in-situ tasks receive the data. The main phase (the 2^{nd} and 3^{rd} lines of Equation 9) is calculated by summing up the maximal execution time among all m_l asynchronous parts of in-situ task ($\sum_{k=1}^{m_l} v_k\ \psi_{k\ main}(\vec{x_s})$) executed on the l^{th} computing resources and the sum of the application (g_{main}) and all m_s synchronous part(s) of in-situ task ($\sum_{k=1}^{m_s} v_k\ \psi_{k\ main}(\vec{x_s})$). For the sake of simplicity, we do not consider the asynchronous parts of in-situ tasks to communicate with each other, so the communication in this performance model can be expressed as the sum of all the communication between the application and asynchronous part(s) of in-situ task or the communication between the synchronous part(s) and asynchronous part(s) of in-situ task (the last line of Equation 9).

IV. Evaluation

We evaluate our performance models with three simulation codes and three typical in-situ tasks.

A. Experimental setups

The used applications are: 1. Nek5000 is a spectral element method (SEM) based CFD code with excellent scalability. [1] 2. NEKO is a successive portable framework of Nek5000, allowing simulation on GPUs and CPUs. [14] 3. Quantum-Espresso (QE) can perform Car-Parrinello MD simulation for complex electronic interactions in large molecules. [12] The integrated in-situ tasks are: 1. Image generation with Paraview/Catalyst [4] is one of the most common in-situ tasks. 2. Physics-based lossy data compression systematically discards data in CFD with low energy spectral coefficients. [17] 3. General lossless data compression is widely used and already offered e.g. as operators in ADIOS2.

In the asynchronous and hybrid in-situ approaches, we use ADIOS2 to communicate the data. Resource allocation is specified in run time configurations. We use Extra-P [7] to construct the performance models. We used five data points to generate the PMNF of the applications and in-situ tasks. [7] We used the cost-effective sampling strategy [20] to generate NFMP for the applications and in-situ tasks with multiple parameters influencing the performance. Assuming that h parameters would influence the performance, we need $(6h - 1)$ data points. Then we evaluated the accuracy of the performance model of in-situ technique with the coefficient of determination (R^2) [7]. It is calculated from the predicted performance and the measured performance with Equation 10, where y_i is the measured performance, $\bar{y}$ is the average of the measured performance, and φ_i is the performance predicted by the performance models. The number of samples used for the accuracy evaluation (z) is much larger than the number of samples used to generate the performance models.

$$R^2 = 1 - \frac{\sum_i^z (y_i - \varphi_i)^2}{\sum_i^z (y_i - \bar{y})^2} \left(\bar{y} = \frac{1}{z} \textstyle\sum_{i=1}^{z} y_i\right) \tag{10}$$

All experiments were performed on the Raven [3] supercomputer at the Max Planck Computing and Data Facility

TABLE I: Performance models of application

Application	Parameter(s)	Performance model of one step in the main part	R^2
NEKO (CPU)	r number of ranks	$7.4203 \times 10^{-3} + 29.0435\ r^{-1}$	0.991
NEKO (GPU)	g number of GPUs r number of ranks	$2.6042 \times 10^{-2} + 1.78424\ g^{-1}$	0.960
Nek5000 (CPU)	r number of ranks	$6.4 \times 10^{-3} + 2.7110 \times 10^{3}\ r^{-1}$	0.995
QE (CPU)	r number of ranks t number of threads per rank	$2.5930 \times 10^{2} - 32.7340 \log_2 r - 5.9391 \times 10^{-1}\ \log_2 t$	0.967
QE (GPU)	g number of GPUs r number of ranks t number of threads per rank	$30.705 + 1.3981\sqrt{\log_2 g} \log_2 r \sqrt{\log_2 t} - 2.8021 \log_2 r \sqrt{\log_2 t}$ $-13.7530\sqrt{\log_2 g}$	0.950

TABLE II: Performance models of in-situ task

In-Situ Task	Parameter(s)	Performance model of one step in the main part	R^2
Lossy data compression for 32^3 NEKO fluid elements	r number of ranks	$O(r)$ (insignificant)	
Lossless data compression for 32^3 NEKO fluid elements	r number of ranks	$O(r)$ (insignificant)	
Lossy data compression for 64^3 NEKO fluid elements	g number of GPUs r number of ranks	$2.5573 \times 10^{-3} + 2.4793 \times 10^{-2}\ g^{-1} + 0.86584\ r^{-1}$	0.941
Lossless data compression for 64^3 NEKO fluid elements	r number of ranks	$0.1384 + 55.3399\ r^{-1}$	0.999
Image generation for 32^3 NEKO fluid elements	r number of ranks	$1.0148 + 2.1720\ r^{-1}$	0.992
Image generation for 64^3 NEKO fluid elements	r number of ranks	$0.3934 + 15.0021\ r^{-1}$	0.985
Lossy data compression for Nek5000 fluid elements	r number of ranks	$O(r)$ (insignificant)	
Lossless data compression for Nek5000 fluid elements	r number of ranks	$O(r)$ (insignificant)	
Image generation for Nek5000 fluid elements	r number of ranks	$3.8060 + 8.2481 r^{-0.5}$	0.999
Data compression for wave function coefficients	r number of ranks	$-11.605 + 205.01318 r^{-0.8}$	0.804

(MPCDF). One CPU node contains two Intel Xeon IceLake-SP 8360Y processors, each with 36 cores and 256 GB RAM. One GPU compute node has four Nvidia A100 GPUs (4×40 GB HBM2 memory per node and NVLink) and two Intel Xeon CPUs with 512 GB RAM. The GPU-accelerated nodes use Nvidia Multi-Process Service (MPS) [2] to allow multiple CPU cores to share access to the same GPU and to use the GPUs more efficiently, and we allocate cores for the original application and in-situ tasks evenly on two CPUs.

B. Accuracy of the performance models

Performance models of applications are shown in Table I. All of them have good accuracy ($R^2 \approx 1$). The CPU-based NEKO and Nek5000 (the 1^{st} and 3^{rd} rows in Table I) are the applications executed only by CPUs with MPI support, so only the number of ranks would influence the performance. For the GPU-accelerated NEKO (the 2^{nd} row in Table I), both the numbers of GPUs and of MPI ranks should influence the performance. For GPU-accelerated NEKO, MPS only brings a slight performance benefit, so the performance model becomes one PMNF, although we provide eleven data points. The 4^{th} row in Table I shows the CPU-based QE and the numbers of MPI ranks and of thread(s) per MPI rank would influence the performance because of its openMP and MPI support; the 5^{th} row in Table I shows the GPU-accelerated QE and the numbers of GPUs, of MPI ranks and thread(s) per MPI rank would all influence the performance.

Most of ***Performance models of in-situ tasks*** (Table II) have good accuracy ($R^2 \approx 1$). We separated the data compression integrated into Nek5000 and NEKO into two parts and tested the performance model of these two parts separately. . The first part is the lossy compression, which reuses functions and parameters from the simulation, i.e., it is deeply coupled with the simulation, and, therefore, always synchronous. The data compression integrated into CPU-based NEKO, and Nek5000 (the 1^{st}, 2^{nd}, 7^{th} and 8^{th} rows in Table II) take only an insignificant amount of time, so we notated the infinitesimal asymptotic in the performance models with a single big O tern ($O(\vec{x})$). For GPU-accelerated NEKO, the data compression is no longer insignificant (the 3^{rd} and 4^{th} rows in Table II) because the NEKO is fast on GPUs. The performance models of image generation (the 5^{th}, 6^{th}, and 9^{th} rows in Table II) do not include the problem size, so we have one performance model for each case.

Performance models of communication with ADIOS2 (Table III) are also accurate. The bandwidth of the communication between the cores on the same node is higher than between cores on different nodes. The 1^{st} row in Table III shows that the bandwidth within one node increases with the number of cores to send data, the number of cores to receive data, and the size of the data transferred. For the communication between nodes (the 2^{nd} row in Table III), when the message size is not large enough, the communication cannot take full advantage of the hardware network bandwidth. We also observe an increase in bandwidth with the message size. However, we cannot saturate the bandwidth of the hardware network bandwidth within one node, which is much larger than the bandwidth between nodes. So users should allocate part of the cores on each node to the application, while the rest to in-situ tasks to ensure data communication within the node. Compared to the

TABLE III: Performance models of communication

Communication	Parameter(s)	Performance models of communication	R^2
"InsituMPI" on the same node	w number of writer ranks r number of reader ranks s message size	$s/(0.2181 + 0.03678 \log_2 w \log_2 r \log_2 s$ $+0.4795 \log_2 w + 0.048928 \log_2 s)$	0.974
"InsituMPI" between nodes	w number of writer ranks r number of reader ranks s message size	$s/0.1506$	0.847

TABLE IV: Performance models of in-situ techniques

Application		In-Situ Task		Execution		Synchronous		Asynchronous		Hybrid	
Name	R^2	Name	R^2	# of nodes	Time range	# of tests	R^2	# of tests	R^2	# of tests	R^2
NEKO (CPU)	0.991	Data compression		1-8	100-800s	80	0.999	No tests		480	0.990
NEKO (CPU)	0.991	Image generation	0.992	1-8	400-2500s	80	0.999	480	0.998	No tests	
NEKO (GPU)	0.960	Data compression	0.941	2-8	300-3300s	250	0.984	No tests		1250	0.953
NEKO (GPU)	0.960	Image generation	0.985	2-8	200-3800s	250	0.995	1250	0.948	No tests	
Nek5000(CPU)	0.991	Data compression		12-28	1500-3500s	80	0.998	No tests		250	0.999
Nek5000(CPU)	0.991	Image generation	0.999	12-28	2000-5500s	80	0.999	250	0.994	No Test	
QE (CPU)	0.967	Data compression	0.804	1-5	250-4500s	450	0.978	600	0.977	No Test	
QE (GPU)	0.950	Data compression	0.804	1-5	150-800s	450	0.910	600	0.844	No Test	

communication between ranks, the data transfer between CPU and GPU is insignificant, so we also denote it with $O(\vec{x})$.

All the R^2 reported in Table. I to Table. III are offered by Extra-P. We derived the ***Performance models of in-situ techniques*** from the performance models of the application and the in-situ task according to the in-situ technique it used. Then, we evaluated the performance with R^2 using Equation 10. The number of nodes used for each execution, the execution time range, and the number of tests to evaluate the accuracy of the performance are listed together with the accuracy in Table IV. In general, the accuracy of the performance models depends on the accuracy of the application's and in-situ task's performance models and the workload difference between the application and the in-situ task. For instance, the last row of Table IV shows that the R^2 of lossless compression on QE data is only 0.804. For the CPU-based QE, data compression takes only neglectable time compared to the total execution time; for the GPU-accelerated QE, the simulation time of QE is on the same exponential level as data compression time. So, low accuracy of data compression has less influence on the derived performance models than GPU-accelerated QE. The 1^{st} - 6^{th} rows in Table IV show the Nek5000 and NEKO with in-situ tasks, and the accuracies of their performance models are, in general, high. The performance models of GPU-accelerated NEKO with in-situ tasks (the 3^{rd} and 4^{th} rows in Table IV) have slightly lower accuracies compared to CPU-based NEKO (the 1^{st} and 2^{nd} rows in Table IV) because the accuracy of its performance is slighter lower. The data compression integrated into Nek5000 and CPU-based NEKO (the 1^{st}, 3^{rd}, and 5^{th} rows in Table IV) has an insignificant influence on the total execution time so we notated the performance models with a single big O tern and calculated it as a zero tern in evaluation. This does not decrease the accuracies of the performance models. Because different applications have different numbers of parameters influencing performance and allow different runtime configurations, the number of samples used for the accuracy evaluation of each application varies.

V. Use Cases

In this section, we provide two use cases of the performance model to predict the performance and preferred configuration to illustrate its potential integration into the production platform. We first use the cost-effective sampling strategy [20] to decide on the data points to measure, generate the performance model, and finally use it for prediction.

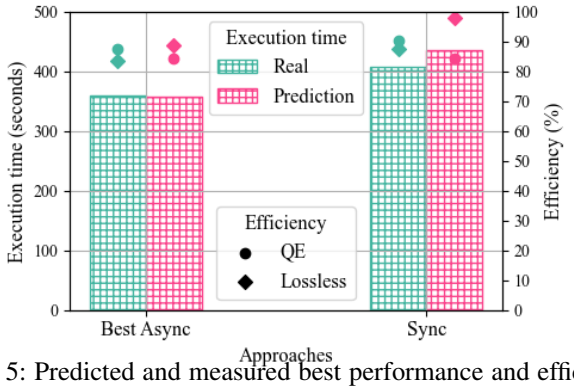


Fig. 5: Predicted and measured best performance and efficiencies of GPU-accelerated QE with compression using four cores per MPI rank, three ranks per GPU and four GPUs for QE.

A. GPU-accelerated QE with lossless data compression

In this use case, we predict the preferred in-situ approach for the GPU-accelerated QE with lossless data compression every ten simulation steps. We use Extra-P to generate the performance models of the GPU-accelerated QE, data compression, and communication. As the case is rather small, we execute it on one GPU node (four GPUs in total). As QE has both Massage Passing Interface (MPI) and OpenMP (allowing multiple CPU cores per MPI rank) support, for each GPU, a different number of CPU cores per MPI rank and a different number of MPI ranks for the CPU-parts of QE can be chosen. Our experiments showed the best performance with four cores per MPI rank and three ranks per GPU.

With this setup (four cores per MPI rank and three ranks per GPU) our model suggests to use the asynchronous in-situ method for lossless compression. Our model predicts that by using eight cores for the lossless compression, we get the best efficiency, as with this setup the time spent in lossless compression is about the same as the time of the simulation steps and the synchronous lossy compression - thus we achieve a good asynchronous overlap with good resource

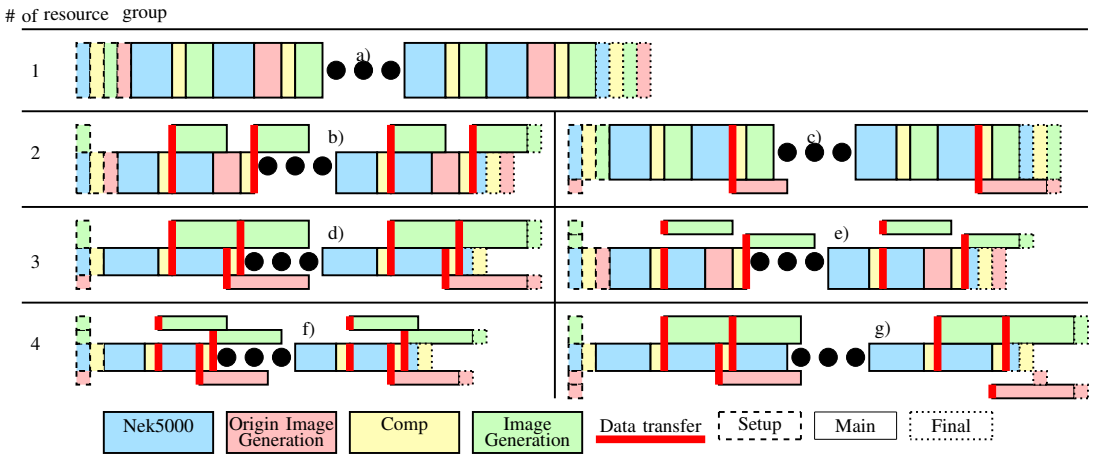
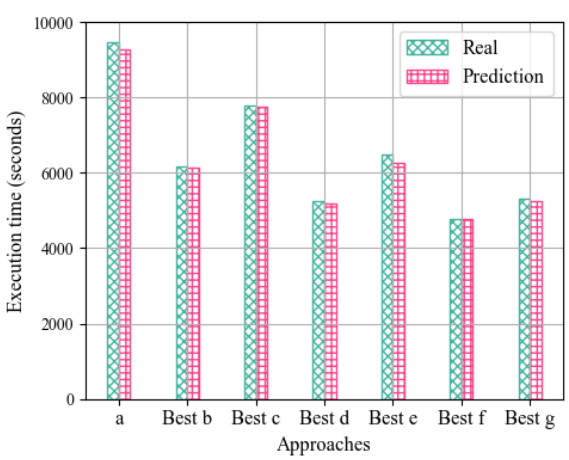
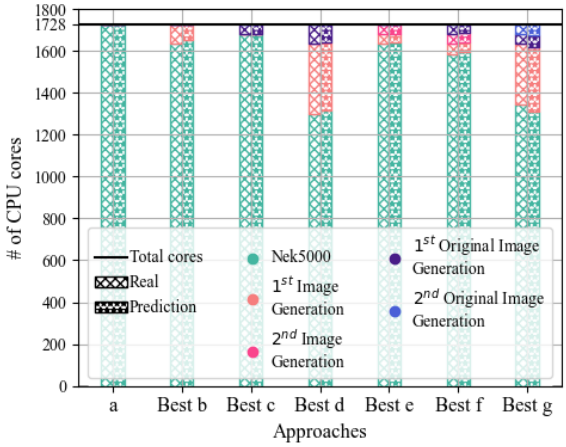


Fig. 6: Illustration of image generation from original and compressed Nek5000 data.



(a) The predicted and measured best performances.



(b) The predicted and measured best configurations (how many CPU cores are being allocated to the simulation and the different asynchronous resource groups).

Fig. 7: The predicted and measured best performances of approaches a to g and their configurations.

usage. All components (simulation, lossy compression, and lossless compression) show a parallel efficiency of over 80%.

Fig. 5 shows the predicted and measured performance and efficiencies of our use case, both in an asynchronous and synchronous manner. Our model correctly predicted the asynchronous manner to outperform the synchronous.

B. Image generation from original and compressed CFD data

In our second use case, we used our performance model to predict the best configuration for a more complex case: we generate images from CFD simulation data using Nek5000 before and after lossy compression, which can then be used, e.g. for visual comparison. We compress data every second simulation step and generate an image from this lossy-compressed data. Every fourth simulation step we also generate an image from the original data. Such a setup can, e.g. be useful if a movie should be made out of the images from the compressed data and less frequent quality checks are performed.

We decompose the in-situ task in our stress test into three parts: lossy data compression and image generation every second simulation step, and image generation from original data every fourth simulation step. The lossy data compression reused the functions from the simulation, so it was performed synchronously. Without in-situ techniques, such an experiment would generate over 130 TB of data for post-processing.

Fig. 6 shows the possible in-situ approaches, with up to three different resource groups for the asynchronous in-situ method where all sets share the same node. When one part of the in-situ task is executed asynchronously, one or two group(s) of resources are used; when one part is executed synchronously, it is executed by the same resources as the simulation. When we test g resource groups for the asynchronous in-situ method, 5^g configurations are tested. For instance, in approach b, we test 2,4,12,18,36 cores per node for asynchronous image generation from compressed data.

Fig. 7 shows the best performing execution times (Fig. 7a) and configurations (Fig. 7b) for each of these approaches. We report the average of five test runs in Fig. 7a, and all the configurations of the best runs are the same and illustrated in Fig. 7b. As can be seen, the predicted and actual execution times are very close, and also the resource configurations are very similar. The overall best performance can be achieved through approach f), with two groups of 48 CPU cores (two cores per node) for image generation from compressed data and one group of 48 CPU cores for image generation from original data having the best performance.

VI. Conclusion and Discussion

In this paper, we developed a performance model for synchronous, asynchronous, and hybrid in-situ approaches using the performance models of the applications and integrated in-situ tasks. We performed systematic performance measurements of three different codes executing both, on CPUs and GPUs: Nek5000 and NEKO (CFD simulations) and QE (MD simulations) with common in-situ tasks: lossy and lossless data compression as well as image generation. We used over 6800 experiments to verify the accuracy of our performance model, which showed a good agreement with the experiments. We also provided two real-world use cases to demonstrate how to use the performance models to predict the best performance and corresponding runtime configurations of the different in-situ approaches and to propose the preferred in-situ technique. This helps to reduce the development and experimental efforts of integrating in-situ tasks into large-scale applications. Thanks to our performance models, we could also identify the potential for further optimization to achieve a better usage of the computational resources. With the general performance model of in-situ techniques (Equation 9), we can also predict the performance of more complex in-situ workflow and suggest the best resource allocation.

In future work, we will design a system for automatic performance model generation and also include more performance metrics in addition to execution time to identify underused resources to provide further suggestions to developers. We will also use our performance models in malleable in-situ approaches. This will allow to add resources for in-situ tasks dynamically during runtime and help with resource underutilization of static resource assignments in cases where e.g. the in-situ tasks only start after a significant runtime of the simulation (e.g. long startup/stabilization phase).

Acknowledgment

The authors gratefully acknowledge the EuroHPC Joint Undertaking (JU) (956748 "ADMIRE" and 101083261 "Plasma-PEPSC") for funding support. The authors would appreciate the Max Planck Computing and Data Facility (MPCDF) for providing compute time on the Raven Supercomputer.

References

[1] Nek5000, a fast and scalable high-order solver for computational fluid dynamics. [Online]. Available: https://nek5000.mcs.anl.gov/

[2] Nvidia nsight systems. [Online]. Available: http://docs.nvidia.com/deploy/mps/index.html

[3] Supercomputer Raven at Max Plank Computing and Data Facility. [Online]. Available: https://www.mpcdf.mpg.de/services/supercomputing/raven

[4] U. Ayachit, A. Bauer, B. Geveci, P. O'Leary, K. Moreland, N. Fabian, and J. Mauldin, "Paraview catalyst: Enabling in situ data analysis and visualization," in *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, 2015, pp. 25–29.

[5] U. Ayachit, B. Whitlock, M. Wolf, B. Loring, B. Geveci, D. Lonie, and E. W. Bethel, "The sensei generic in situ interface," in *2016 Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*. IEEE, 2016, pp. 40–44.

[6] A. Bhattacharyya, G. Kwasniewski, and T. Hoefler, "Using compiler techniques to improve automatic performance modeling," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*. IEEE, 2015, pp. 468–479.

[7] S. Biersdorff, C. Bischof, K. Diethelm, D. Eschweiler, M. Gerndt, A. Knüpfer, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, C. Rössel, P. Saviankou, D. Schmidl, S. Shende, W. M, B. Wesarg, and F. Wolf, "Score-p: A unified performance measurement system for petascale applications." in *HPC Status Konferenz der Gauß-Allianz e.V., Schwetzingen*. Springer, 2011.

[8] A. Calotoiu, D. Beckinsale, C. W. Earl, T. Hoefler, I. Karlin, M. Schulz, and F. Wolf, "Fast multi-parameter performance modeling," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2016, pp. 172–181.

[9] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, "Using automated performance modeling to find scalability bugs in complex codes," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.

[10] H. Childs, "Visit: An end-user tool for visualizing and analyzing very large data," 2012.

[11] A. Gainaru, L. Wan, R. Wang, E. Suchyta, J. Chen, N. Podhorszki, J. Kress, D. Pugmire, and S. Klasky, "Understanding the impact of data staging for coupled scientific workflows," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4134–4147, 2022.

[12] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo *et al.*, "Quantum espresso: a modular and open-source software project for quantum simulations of materials," *Journal of physics: Condensed matter*, vol. 21, no. 39, p. 395502, 2009.

[13] W. F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, P. Davis, J. Choi, K. Germaschewski, K. Huck *et al.*, "Adios 2: The adaptable input output system. a framework for high-performance data management," *SoftwareX*, vol. 12, p. 100561, 2020.

[14] N. Jansson, M. Karp, A. Podobas, S. Markidis, and P. Schlatter, "Neko: A modern, portable, and scalable framework for high-fidelity computational fluid dynamics," *arXiv preprint arXiv:2107.01243*, 2021.

[15] A. Jayakumar, P. Murali, and S. Vadhiyar, "Matching application signatures for performance predictions using a single execution," in *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2015, pp. 1161–1170.

[16] Y. Ju, M. Li, A. Perez, L. Bellentani, N. Jansson, S. Markidis, P. Schlatter, and E. Laure, "In-situ techniques on gpu-accelerated data-intensive applications," in *2023 IEEE 19th International Conference on e-Science (e-Science)*. IEEE, 2023, pp. 1–10.

[17] Y. Ju, A. Perez, S. Markidis, P. Schlatter, and E. Laure, "Understanding the impact of synchronous, asynchronous, and hybrid in-situ techniques in computational fluid dynamics applications," in *2022 IEEE 18th International Conference on e-Science (e-Science)*. IEEE, 2022, pp. 295–305.

[18] T. Kuhlen, R. Pajarola, and K. Zhou, "Parallel in situ coupling of simulation with a fully featured visualization system," in *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization (EGPGV)*, vol. 10. Eurographics Association Aire-la-Ville, Switzerland, 2011, pp. 101–109.

[19] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield *et al.*, "Hello adios: the challenges and lessons of developing leadership class i/o frameworks," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 7, pp. 1453–1473, 2014.

[20] M. Ritter, A. Calotoiu, S. Rinke, T. Reimann, T. Hoefler, and F. Wolf, "Learning cost-effective sampling strategies for empirical performance modeling," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 884–895.

[21] W. Schroeder, K. M. Martin, and W. E. Lorensen, *The visualization toolkit an object-oriented approach to 3D graphics*. Prentice-Hall, Inc., 1998.

[22] T. Shu, Y. Guo, J. Wozniak, X. Ding, I. Foster, and T. Kurc, "Bootstrapping in-situ workflow auto-tuning via combining performance models of component applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[23] N. R. Tallent and A. Hoisie, "Palm: Easing the burden of analytical performance modeling," in *Proceedings of the 28th ACM international conference on Supercomputing*, 2014, pp. 221–230.