LOOM: Interweaving tightly coupled visualization and numeric simulation framework

João Barbosa
INESC-TEC and University of Minho
Braga, Portugal

Luis Paulo Santos
INESC-TEC and University of Minho
Braga, Portugal

Paul Navratil
TACC - University of Texas at Austin
Austin, Texas, USA

Donald Fussell
University of Texas at Austin
Austin, Texas, USA

ABSTRACT

Traditional post-hoc high-fidelity scientific visualization (HSV) of numerical simulations requires multiple I/O check-pointing to inspect the simulation progress. The costs of these I/O operations are high and can grow exponentially with increasing problem sizes. In situ HSV dispenses with costly check-pointing I/O operations, but requires additional computing resources to generate the visualization, increasing power and energy consumption. In this paper we present LOOM, a new interweaving approach supported by a task scheduling framework to allow tightly coupled in situ visualization without significantly adding to the overall simulation runtime. The approach exploits the idle times of the numerical simulation threads, due to workload imbalances, to perform the visualization steps. Overall execution time (simulation plus visualization) is minimized. Power requirements are also minimized by sharing the same computational resources among numerical simulation and visualization tasks. We demonstrate that LOOM reduces time to visualization by 3× compared to a traditional non-interwoven pipeline. Our results here demonstrate good potential for additional gains for large distributed-memory use cases with larger interleaving opportunities.

CCS CONCEPTS

• **Computing methodologies** → *Parallel algorithms.*

KEYWORDS

scientific visualization, tightly-coupled in situ, parallel scheduling

ACM Reference Format:
João Barbosa, Luis Paulo Santos, Paul Navratil, and Donald Fussell. 2021. LOOM: Interweaving tightly coupled visualization and numeric simulation framework. In *ISAV 2021: In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization, Nov. 15th 2021, St. Louis, MO.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3490138.3490139

1 INTRODUCTION

Today high-performance computing (HPC) has become pervasive across all society. Science, health care, industry, and even entertainment rely on considerable computing resources, especially on HPC.

Numerical simulation is the cornerstone for discovery in science and engineering, with increasing demands for computational resources and producing massive datasets for analysis. In addition to the computation itself, most of these applications and workflows involve some graphical capabilities, either as part of the application itself (e.g., user interface) or for analysis of the resulting data (e.g., scientific visualization) [10].

Scientific visualization is typically the last step in the scientific numerical analysis pipeline and, until a decade ago, was only performed using a *post-hoc* approach. A collection of frameworks [4, 8, 9, 12, 14, 15, 22] are considered tightly-coupled, and many also offer an off-node, "loosely coupled" mode which requires additional computational resources. The loosely coupled approach uses a collection of techniques to intercept the data, for instance, ADIOS [13], an *in-transit* framework that intercepts the simulation I/O operations to transport the data the visualization nodes. Two other examples of co-processing approach are ParaView Catalyst[4] from Kitware, and VisIt libsim[22] from Lawrence Livermore National Laboratory (LLNL) and Intelligent Light.

We observe that since a simulation and the analysis of that simulation represent separate workflow phases, there is an opportunity to exploit simulation workload imbalance to insert visualization operations on "ready" simulation data before global conclusion of a simulation iteration. This paper presents LOOM, a new interweaving approach to in situ numeric simulation and scientific visualization resource sharing. The new interweaving approach aims to integrate the two workloads seamlessly by classifying each as high and low priority, respectively, and schedule the task execution such that the low priority task execution can fully exploit the potential resource idle due to the numeric simulation load unbalance and minimize or even hide the visualization workload cost. We demonstrate that for both a single-node, multi-threaded use case and a small multi-node use case, LOOM achieves time to visualization 3 × faster than non-interwoven simulation-visualization workflow. We expect LOOM to achieve similar performance for large multi-node use cases with greater work imbalance across nodes.

2 RELATED WORK

For the last decade, a significant effort has been made to migrate from a post-hoc visualization to an in situ data visualization, and analysis [7]. These efforts stem from the need to overcome some of the significant I/O bottlenecks [2, 19] that large numeric simulations incur when dumping their state for post-analysis, especially as they scale up to exascale. Furthermore, it is critical to allow computational steering as the numeric simulations grow up to exascale.

Although a collection of frameworks [4, 8, 9, 12, 14, 15, 22] are considered tightly-coupled often requiring additional computational resources. A more complete description of in situ terminology and frameworks can be found by Kress [11] and Childs et al. [7].

Zheng et. al. [23] proposed the GoldRush framework that performs analysis during the serial portions of OpenMP based scientific applications. GoldRush is able to predict the serial region length and execute the analysis payload when the interval is long enough. This approach throws out any serial region deemed small and so does not recover the idle time of the cores waiting for the end of the parallel region.

Barbosa et al. [5, 18] proposed allowing the developer to define the partition method for the data and rely on a performance model and scheduler to *dice* the tasks to be decomposed into smaller ones when needed. This mechanism can be leveraged to enable the interweaving of simulation and visualization tasks, primarily because they operate in different time intervals, t and $t - 1$, respectively. We leverage this property in our approach.

Our work is a tightly coupled in situ, focusing on balancing the simulation and the visualization workloads by exploiting the idle time in the simulation threads. This approach allows us to reduce the overall simulation and visualization time (T_{s+v}) saving at least static energy and our goal is to achieve a solution time ($T_{s||v}$) equal to the simulation execution time (T_s).

3 OUR APPROACH

Assuming that the simulation is irregular, we also assume that it will generate idle time that we can exploit to hide a portion of the visualization time. Our goal is then to interweave the simulation and the visualization in such a way that: $T_s \leq T_{s\|v} < T_{s+v}$, i.e., such that the execution $T_{s\|v}$ is equal to the simulation time (T_s) and smaller than the traditional approach (T_{s+v}).

This requires as little impact on the simulation as possible, thus we use a priority based policy for our scheduling approach. We schedule tasks (τ) as high or low priority depending on if they are simulation or visualization tasks. Using this approach, we ensure that for the scheduling interval that corresponds to the interval between I/O checkpoints, the scheduler serves primarily *ready to execute* simulation tasks (τ_s), while serving τ_v tasks, if available, when simulation threads idles. The solution gives rise to a granularity of τ_v, i.e., if the workload from τ_v ($W(\tau_v)$) is too large, it will delay the execution of subsequently scheduled τ_s, on the other hand, if $W(\tau_v)$ is too small it will increase the scheduling overhead. To address this problem, we follow the approach proposed by Barbosa et al. [5, 18] referred to as *dicing*. The dicing strategy enables the developer to define a generic workload applied to a partition created by the scheduler using a developer-defined partitioning method called *dice*. In other words, the scheduler is free to *dice* the original task τ_v to fit a specific time slot, in our case, an expected idle time between consecutive τ_s. As a consequence, the scheduler will begin sim timestep $t + 1$ immediately after sim timestep t and attempt to render vis timestep t within the gaps of sim timestep $t + 1$. This approach offers more opportunities to interleave the computation compared to rendering vis t within the gaps of sim t, at the cost of maintaining both timesteps sim t and sim $t + 1$ in memory.

We use a simple moving average model to predict both the idle time (T_i) and visualization task execution time (T_{v,τ_v}). Such values are fed to the scheduler, which estimates the diced task ($\tau_v^{'}$) workload size ($W(\tau_v^{'})$) using the following equation:

$$W(\tau_v^{'}) = \frac{T_i * W(\tau_v)}{T_v} \tag{1}$$

where $W(\tau_v^{'})$ is going to be passed to the developer defined partition method to obtain $\tau_v^{'}$. The partitioned task is then submitted for execution and the remaining $W(\tau_v)$ is pushed for rescheduling.

To maintain and ensure that all task dependencies are met and to ensure execution in a timely fashion, we maintain a hierarchical set of queues that serve different purposes. At the top level, we maintain a *Global Not Ready Queue* (GNRQ) implemented as a directed acyclic graph (DAG) that contains all the tasks submitted to the scheduler that are not yet ready for execution due to dependencies. Whenever a task is executed, all of its successors are checked to verify if any became ready for execution. If the successor has become ready (all dependencies are met), it is moved to a *Global Ready Queue* (GRQ). We implement the priority scheduling policy described above at this level. Ultimately a *Local Ready Queue* (LRQ) is used at each core to reduce access contention to GRQ and to implement a simple work-stealing mechanism to improve load balancing among the cores: i) Pull from GRQ a set of tasks (high and low according to the GRQ policy) to fill the LRQ; ii) If GRQ is empty, steal work from the core siblings if possible; and iii) Idle the core if no more work exists in the scheduler GRQ and LRQ's until a task released from GNRQ awakes the core/thread.

Using this strategy, we can interweave the simulation and visualization as shown in Figure 1 using real data from the scheduler.

4 NUMERIC SIMULATION AND SCIENTIFIC VISUALIZATION INTERWEAVING

We assume that the numeric simulation performs its own partition (decomposition) and that such partitioning is efficient, and thus the task implements an operation over each partition. For the visualization, we implement *diceble* tasks that the scheduler can shape to the best granularity to fit in the numeric simulation idle gaps.

4.1 Generic Task and Diceble Task

A *Generic Task* is an object that encodes a workload using state attributes and an *execute* method, i.e., it encodes the operation to perform and the application state to manipulate within the class attributes.

A *Diceable Task* is a generic task that adds a user-defined *dice* method allowing the scheduler to request specific sized partitions. A generalized prediction model for the work generated by a certain partition size remains an open research question beyond the scope of this paper. Rather than a generalized model, we use a *per-task user-defined work unit* (Wu) that allows the user to specify the number of Wu's per task, enabling them to define work by the concept or measure most appropriate for the task. The scheduler uses the observed execution time of these work units to create a statistical model of expected execution time that determines the number of work units to schedule into an available time slot, as described in Equation 1.

4.2 Execution and Scheduling

The task system uses a simple work-stealing strategy that balances workloads across the threads issued to each CPU core. The scheduler does not over-subscribe the CPU cores except for the first core that runs a *working* thread and the main application thread. Each thread has its individual *Local Ready Queue* (LRQ) from where it pulls a single task for execution. All tasks in the LRQ are ready for execution, i.e., it is expected that all tasks in LRQ have their dependencies met. Whenever the LRQ is empty it either: i) steals a group of tasks from the the *Global Ready Queue* (GRQ); ii) Steal a group of tasks from the neighbors LRQ if GRQ is empty; and iii) Idle/Sleep on a conditional variable until a task is posted to GRQ.

The system keeps a DAG with all the tasks in *flight*, i.e., all the tasks submitted for execution but have not yet been executed. When a task is submitted, the set of dependencies must be attached by the developer. These dependency sets are used to build the DAG, which is used to ensure that all dependencies are met before placing the task in the GRQ.

When a *diceble* task is pulled from the GRQ, the *dice* method of the task is invoked after computing the value of equation 1 using the performance bookkeeping. If a new task is returned, it is placed in LRQ, and the remaining task (the part that was not assigned to the diced task) is placed back in the GRQ; otherwise, it is assumed that the task cannot be diced anymore and placed in the LRQ.

4.3 Time bookkeeping

A crucial step is to keep a reasonable time bookkeeping with the scheduler since the visualization tasks' dice strategy needs it. We achieve such a goal by measuring the time a working thread sleeps, the time a task takes to execute, and the scheduler's scheduling overhead. For the idle time information, we maintain two simple atomic variables per worker thread that store: the total amount of idle time of the thread and a weighted moving average of the idle time interval.

The only tasks that we track are the *diceble* tasks through the *user-defined* method to partition the task into smaller ones. For each *diceble task* type, we keep a collection of ten samples uniformly random selected containing: the average idle time of the thread when the dice was requested; the total number of work units (Wu) as supplied by the developer; the total execution time of the sample; and estimated execution time as predicted by the scheduler. Whenever a *dice* request is going to be performed, the samples are averaged, and the values obtained are supplied to equation 1.

5 RESULTS

5.1 VADIS

We test our assumption on a Computational Fluid Dynamics (CFD) model VADIS (pollutant DISpersion in the atmosphere under VAriable wind conditions) developed as a numerical tool to assess local scale air pollutants dispersion in complex urban morphologies [3, 16, 17, 20], by considering multi-obstacle and multi-source description, as well as time-varying flow fields and time-varying emissions. The VADIS functioning is based on two modules: the FLOW and the DISPER modules. The FLOW is an Eulerian module that uses the numerical solution of the 3D Reynolds averaged Navier-Stokes equations to calculate the wind velocity components, turbulent viscosity, pressure, turbulence kinetic energy, and temperature fields. The DISPER module applies the Lagrangian approach to the computation of the 3D concentration field of the air pollutant dispersion using the wind field previously estimated by the FLOW.

5.2 DGSWEM

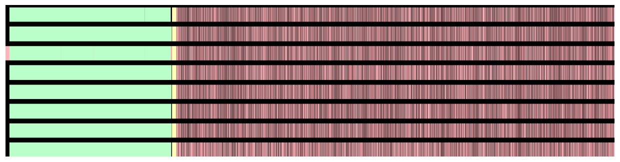
We use the discontinuous Galerkin (DG) shallow water equation model (SWEM) as an example of an application that requires scientific visualization to monitor the progress of the simulation. DGSWEM is a hurricane storm surge prediction model which takes into account the effects of wind stress that pushes water on to land using a DG kernel which has achieved widespread popularity due to its stability and high-order convergence properties. For details of the model and the parallelization method see [6].

5.3 Testing environment

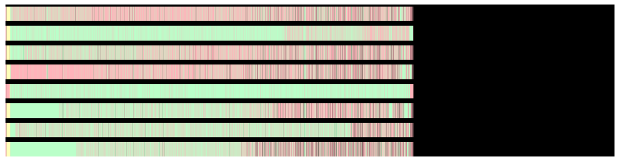
The interweaving solution was tested on an Intel(R) Core(TM) i7-8809G CPU @ 3.10GHz, with four cores plus two-way hyper-threading (8 core targets), 32GB of RAM, and an SSD storage unit. The machine was running Fedora Core 33 with kernel 5.10.13-200.fc32. The software was compiled with GCC version 11.0.0 with options "-O3 -arch=native". Frontera [21] at the Texas Advanced Computing Center (TACC) was also used. We used up to 4 compute nodes each with two Intel(R) Xeon(TM) Platinum 8280 ("Cascade Lake") processors for 56 cores per node, without hyper-threading, running CentOS 7.8. The software was compiled with GCC version 8.3.0 with options "-O3 -arch=native". Each version of the application and the different combinations was run 10 times, and the average time was taken as the reference value.

5.4 Results Discussion

Figure 1 shows how the interweaving approach works on the DISPER application from VADIS, with the simulation tasks in red, the I/O tasks for the simulation to read data from storage in yellow, and the visualization tasks for the previous simulation results in green. Figure 1 (a) shows the result of performing the visualization and simulation using the traditional *in situ* approach, where all the tasks from $t-1$ are executed prior to the simulation tasks of t. We can see that between the simulation tasks black blocks represent idle time that is wasted. In Figure 1 (b) we can see that the visualization tasks from $t-1$ are inter-weaved with the simulation tasks of t leading to a much lower percentage of overall idle time. Furthermore, since the time axis of both images are the same length



(a) Traditional *in situ* approach



(b) LOOM approach

Figure 1: DISPER - VADIS task timeline plot for visualization at time t and simulation at time $t+1$ on a 8 core machine: the visualization tasks are in green and the simulation tasks are in red. Time grows from left to right and time axis has the same length. By interleaving simulation and analysis, our approach completes the workflow faster than traditional post-hoc analysis.

Δ	T_s	T_{s+v}	$T_{s\|v}$	$\downarrow$ TTV	T_{s+IO}
3	187.29	487.61	402.2	**1.39x**	246.74
1.5	636.02	1097.94	929.89	**1.57x**	1366.63
1	1723.18	2382.19	1953.83	**2.85x**	4740.81

Table 1: DISPER simulation time, sequential time for simulation and visualization, parallel execution of simulation and visualization, improved time to visualization (TTV), and simulation with I/O (sim cost for post-hoc analysis). Δ is the decomposition delta of the $1248 \times 1248 \times 120$ grid using 3m, 1.5m and 1m cell size.

Nodes	T_s	T_{s+v}	$T_{s\|v}$	$\downarrow$ TTV	T_{s+IO}
2	2141	2273	2189	**2.75x**	2991
4	1219	1298	1247	**2.82x**	1603

Table 2: DGSWEM simulation time, time for simulation and visualization, parallel execution of simulation and visualization using 2 and 4 nodes of TACC Frontera

we can see that there is an total time advantage of interweaving the tasks; this overall time reduction is due to the recovery of the idle time of Figure 1 (a). This advantage can be seen on Table 1.

Table 1 shows the average results from DISPER application of VADIS in seconds for each of the configurations for different grid sizes obtained by defining the decomposition delta of the grid as 3^3m, 1.5^3m, and 1^1m respective to the full grid size of $1248 \times 1248 \times 120$ meters of an urban environment. Besides defining the fluid field size for the advection space, the grid also defines the particle concentration volume that is checkpointed (I/O) and the size of the volume used for volume ray tracing through the Intel OpenVKL library.

Each column of Table 1 represents a different configuration of the application. The T_s column shows the result of executing the numerical simulation and is our base point for the complete analysis. T_{s+v} column presents the average execution of the simulation code followed by the visualization code sequentially, i.e., without overlapping with the simulation step of the next time step. The column $T_{s\|v}$ is our proposed approach that overlaps the execution of the next simulation step with the *in situ* visualization of the previous time-step. The T_{s+IO} is similar to our approach because we overlap IO and simulation instead of waiting for the write operation to finish; however, instead of performing the visualization, it writes the final pollutant particle concentration into the disk.

The "$\downarrow$ TTV" values show the reduction of time to visualization using our approach $T_{s\|v}$ when compared to the traditional approach T_{s+v} using the following formula:

$$\text{Improvement} = \frac{T_{s+v} - T_s}{T_{s\|v} - T_s} \qquad (2)$$

Table 1 shows that as simulation size increases, the cost of checkpointing data for post-hoc analysis also increases (as expected) and that even at these modest simulation sizes, in situ methods provide faster time to visualization. Although the rendering cost is higher as resolution increases, it does not increase in a cubic fashion but the worst case as the norm of the diagonal of the volume. As seen in Figure 1 the higher cost of the simulation allows for the entire visualization step to finish before the simulation step. This characteristic also explains why the improvement increases with the grid size; we have more time to hide the visualization.

Table 2 shows the execution of DGSWEM for different MPI processes count. The visualization of DGSWEM generates per checkpoint three 4k (4096×2160) images showing three different properties. The images are generated in each process for the local meshes and gathered in rank 0 using a last sort approach. Table 2 shows that we were able to hide a significant portion of the visualization cost to achieve almost a 3x acceleration. However, we can clearly see that the visualization still has a significant weight in the overall execution which implies that the idle time predictor needs improvement to be able to reduce the impact of visualization and better estimate the vis workload.

6 CONCLUSION

In this paper we present a interweaving simulation and in situ visualization workloads able to exploit the simulation idle time to perform the lower priority task (visualization). The interweaving is supported by a scheduling strategy that includes an automatic granularity adjustment of the visualization workload using a user-defined *dicing function*. This strategy allows the scheduler to choose a granularity that matches the expected idle time with a low overhead.

The proposed approach in this paper does not yet target a production-scale in situ visualization pipeline, most obviously due to the lack of a cross-node scheduler (the DGSWEM results interleave only locally to each node). Nevertheless, these results demonstrate overall efficiency for both execution time and resource allocation requirements. Being able to efficiently interleave the simulation and the visualization pipeline without performance degradation on the simulation side and without the need for additional resources is a significant step towards efficient exascale computing. As future work, we will explore further optimizations and scheduling opportunities enabled by using a distributed task scheduling framework such as Galaxy [1].

Our approach degenerates into a traditional approach when the simulation workload is perfectly balanced among the cores of a single node and thus, in the worst-case, increments the visualization's execution time to the overall execution time. However, we are interested in the other side of the spectrum, or the middle range, when applications are not well balanced between the threads/cores or even computational nodes. In such cases, the approach can efficiently exploit the idle simulation time to perform the pipeline and generate the visualization outputs, minimizing the overall execution time, i.e., minimal performance impact.

ACKNOWLEDGMENTS

This work was supported in part by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020. This research was also supported in part by the Frontera computing project at the Texas Advanced Computing Center. Frontera is made possible by National Science Foundation award OAC-1818253.

REFERENCES

[1] Greg Abram, Paul Navrátil, Pascal Grossett, David Rogers, and James Ahrens. 2018. Galaxy: Asynchronous Ray Tracing for Large High-Fidelity Visualization. In *2018 IEEE 8th Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE, 72–76.

[2] S. Ahern, A. Shoshani, K.L. Ma, A. Choudhary, T. Critchlow, S. Klasky, and V. Pascucci. 2011. Scientific Discovery at the Exascale: Report from the (DOE) (ASCR) 2011 Workshop on Exascale Data Management, Analysis, and Visualization. https://doi.org/10.2172/1011053 Office of Scientific and Technical Information (OSTI).

[3] J.H. Amorim, V. Rodrigues, R. Tavares, J. Valente, and C. Borrego. 2013. CFD modelling of the aerodynamic effect of trees on urban air pollution dispersion. *Science of The Total Environment* 461-462 (2013), 541–551. https://doi.org/10.1016/j.scitotenv.2013.05.031

[4] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. 2015. 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* (Austin, TX, USA) *(ISAV2015)*. ACM, New York, NY, USA, 25–29. https://doi.org/10.1145/2828612.2828624

[5] Joao Barbosa, Donald Fussell, and Calvin Lin. [n. d.]. Simple Heterogeneous Programming Using Multigranular Tasks.

[6] Maximilian Bremer, Kazbek Kazhyken, Hartmut Kaiser, Craig Michoski, and Clint Dawson. 2019. Performance Comparison of HPX Versus Traditional Parallelization Strategies for the Discontinuous Galerkin Method. *J. Sci. Comput.* 80, 2 (Aug. 2019), 878–902. https://doi.org/10.1007/s10915-019-00960-z

[7] Hank Childs, Sean D Ahern, James Ahrens, Andrew C Bauer, Janine Bennett, E Wes Bethel, Peer-Timo Bremer, Eric Brugger, Joseph Cottam, Matthieu Dorier, et al. 2020. A terminology for in situ visualization and analysis systems. *The International Journal of High Performance Computing Applications* 34, 6 (2020), 676–691.

[8] Nathan Fabian, Kenneth Moreland, David Thompson, Andrew C Bauer, Pat Marion, Berk Gevecik, Michel Rasquin, and Kenneth E Jansen. 2011. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *2011 IEEE symposium on large data analysis and visualization*. IEEE, 89–96.

[9] Robert Haimes, David Edwards, David Edwards, and Robert Haimes. 1997. Visualization in a parallel processing environment. In *35th Aerospace Sciences Meeting and Exhibit*. 348.

[10] Christopher Johnson and Charles Hansen. 2004. *Visualization Handbook*. Academic Press, Inc., Orlando, FL, USA.

[11] James Kress. 2017. In situ visualization techniques for high performance computing. *Eugene OR* (2017).

[12] Matthew Larsen, James Ahrens, Utkarsh Ayachit, Eric Brugger, Hank Childs, Berk Geveci, and Cyrus Harrison. 2017. The alpine in situ infrastructure: Ascending from the ashes of strawman. In *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization*. 42–46.

[13] Qing Liu, Jeremy Logan, Yuan Tian, Hasan Abbasi, Norbert Podhorszki, Jong Youl Choi, Scott Klasky, Roselyne Tchoua, Jay Lofstead, Ron Oldfield, et al. 2014. Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. *Concurrency and Computation: Practice and Experience* 26, 7 (2014), 1453–1473.

[14] Steven G Parker and Christopher R Johnson. 1995. SCIRun: a scientific programming environment for computational steering. In *Supercomputing'95: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*. IEEE, 52–52.

[15] Brad Peterson, Harish Dasari, Alan Humphrey, James Sutherland, Tony Saad, and Martin Berzins. 2015. Reducing overhead in the uintah framework to support short-lived tasks on gpu-heterogeneous architectures. In *Proceedings of the 5th International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*. 1–8.

[16] Sandra Rafael, Luís P Correia, Diogo Lopes, Jorge Bandeira, Margarida C Coelho, Mário Andrade, Carlos Borrego, and Ana I Miranda. 2020. Autonomous vehicles opportunities for cities air quality. *Science of the Total Environment* 712 (2020), 136546.

[17] S Rafael, B Vicente, V Rodrigues, AI Miranda, C Borrego, and M Lopes. 2018. Impacts of green infrastructures on aerodynamic flow and air quality in Porto's urban area. *Atmospheric Environment* 190 (2018), 317–330.

[18] Roberto Ribeiro, João Barbosa, and L. Santos. 2015. A Framework for Efficient Execution of Data Parallel Irregular Applications on Heterogeneous Systems. *Parallel Process. Lett.* 25 (2015), 1550004:1–1550004:30.

[19] Alejandro Ribés and Bruno Raffin. 2020. The Challenges of In Situ Analysis for Multiple Simulations. In *ISAV'20 In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (Atlanta, GA, USA) *(ISAV'20)*. Association for Computing Machinery, New York, NY, USA, 32–37. https://doi.org/10.1145/3426462.3426468

[20] Vera Rodrigues, Sandra Rafael, Sandra Sorte, Sílvia Coelho, Hélder Relvas, Bruno Vicente, Joana Leitão, Myriam Lopes, Ana Isabel Miranda, and Carlos Borrego. 2018. Adaptation to climate change at local scale: a CFD study in Porto urban area. In *Computational Fluid Dynamics-Basic Instruments and Applications in Science*. InTech, 163–186.

[21] Dan Stanzione, John West, R Todd Evans, Tommy Minyard, Omar Ghattas, and Dhabaleswar K Panda. 2020. Frontera: The evolution of leadership computing at the national science foundation. In *Practice and Experience in Advanced Research Computing*. 106–111. https://doi.org/10.1145/3311790.3396656

[22] Brad Whitlock, Jean M. Favre, and Jeremy S. Meredith. 2011. Parallel in Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization* (Llandudno, UK) *(EGPGV '11)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 101–109. https://doi.org/10.2312/EGPGV/EGPGV11/101-109

[23] Fang Zheng, Hongfeng Yu, Can Hantas, Matthew Wolf, Greg Eisenhauer, Karsten Schwan, Hasan Abbasi, and Scott Klasky. 2013. GoldRush: Resource efficient in situ scientific data analytics using fine-grained interference aware execution. In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 1–12. https://doi.org/10.1145/2503210.2503279