The Impact of Work Distribution on In Situ Visualization: A Case Study

Tobias Rau
University of Stuttgart
tobias.rau@visus.uni-stuttgart.de

Patrick Gralka
University of Stuttgart
patrick.gralka@visus.uni-stuttgart.de

Oliver Fernandes
University of Stuttgart
oliver.fernandes@visus.uni-stuttgart.de

Guido Reina
University of Stuttgart
guido.reina@visus.uni-stuttgart.de

Steffen Frey
University of Stuttgart
steffen.frey@visus.uni-stuttgart.de

Thomas Ertl
University of Stuttgart
thomas.ertl@vis.uni-stuttgart.de

ABSTRACT

Large-scale computer simulations generate data at rates that necessitate visual analysis tools to run in situ. The distribution of work on and across nodes of a supercomputer is crucial to utilize compute resources as efficiently as possible. In this paper, we study two work distribution problems in the context of in situ visualization and jointly assess the performance impact of different variants. First, especially for simulations involving heterogeneous loads across their domain, dynamic load balancing can significantly reduce simulation run times. However, the adjustment of the domain partitioning associated with this also has a direct impact on visualization performance. The exact impact of this side effect is largely unclear a priori as generally different criteria are used for balancing simulation and visualization load. Second, on node level, the adequate allocation of threads to simulation or visualization tasks minimizes the performance drain of the simulation while also enabling timely visualization results. In our case study, we jointly study both work distribution aspects with the visualization framework MegaMol coupled in situ on node level to the molecular dynamics simulation ls1 Mardyn on Stampede2 at TACC.

CCS CONCEPTS

• **Human-centered computing** → **Scientific visualization**; • **Computing methodologies** → **Simulation evaluation**; *Ray tracing*; Molecular simulation.

KEYWORDS

load balancing, visualization, HPC, distributed rendering

ACM Reference Format:
Tobias Rau, Patrick Gralka, Oliver Fernandes, Guido Reina, Steffen Frey, and Thomas Ertl. 2019. The Impact of Work Distribution on In Situ Visualization: A Case Study. In *ISAV: In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV'19), November 18, 2019, Denver, CO, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3364228.3364233

ISAV'19, November 18, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-7723-2/19/11…$15.00
https://doi.org/10.1145/3364228.3364233

1 INTRODUCTION

With particle simulation codes scaling well on current supercomputer architectures, complex phenomena can be simulated at high fidelity, revealing new insights into the principles of thermodynamics [11, 25]. The output sizes generated by these simulations can reach several petabytes and beyond, such that storing the results becomes unfeasible due to storage and bandwidth limitations (e.g. [2]). Many of the issues pertaining to this problem are dealt with by employing in situ analysis and visualization techniques that process data as it is generated. In the following, we focus on tightly coupled setups that share the same resources for simulation and visualization.

Many different in situ frameworks have evolved in recent years [1, 3, 5, 6, 23, 24, 27, 30]. Some software packages focus on efficient I/O, either by directly optimizing throughput to underlying persistent storage [16], or by generating optimized data representations [28] to reduce the data. Others efficiently exploit new hardware architectures like accelerators [13, 15, 19], or offer an API incorporating a collection of methods [4, 14]. These techniques aim to strike a balance between delivering an adequate analysis and minimizing the impact on resource usage.

It is a common approach for simulation codes to minimize their time to solution by balancing the computational load across several compute nodes. To achieve this, typically the domain decomposition is adjusted according to some domain-specific metric (e.g., based on molecule density). As it is crucial for performance to keep the volume of data transfers to the required minimum, the size of the domain for visualization directly changes with the adjustments done for the simulation. These changes naturally directly influence the load characteristics of the visualization as well. Here, a metric designed to optimize the solver's time to solution generally deviates from the criteria employed to balance visualization load [7, 8, 17]. Generally, not only the data content but also the shape of the domain have significant performance impact. Another aspect of work distribution is the (node-level) allocation of compute resources to simulation and visualization, respectively. Here, the main goal is generally to minimize the performance impact of the visualization on the simulation, while still achieving timely visualization results (e.g., for interactive exploration or monitoring tasks).

This work evaluates the impact of work distribution in the context of in situ visualization. In particular, we focus (1) on the impact of simulation load balancing across nodes on visualization and (2) the distribution of threads to simulation and visualization on

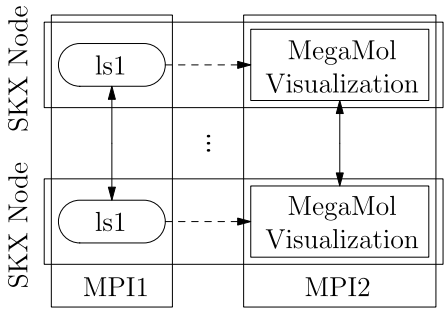


Figure 1: Overview of the deployment on a cluster architecture. Dashed: node-local communication, Solid: on-site network communications.

each node. For this we use the particle solver code ls1 Mardyn [20] and the visualization framework MegaMol [9, 10], which incorporates the ray tracing engine OSPRay [22, 26]. This setup is targeted at providing a loosely coupled, general and robust in situ solution. Data is exclusively communicated to the visualization via shared memory on node level.

2 BASICS AND CASE STUDY SETUP

The setup of this evaluation is as follows: Each rank of the simulation has its own node and is coupled to a MegaMol rank on the same node in a one-to-one manner. The simulation rank transports only its local partition of the data set to the local MegaMol rank. Therefore, each MegaMol rank renders only a world-space data brick that is contributed to a compositing tree (facilitating *IceT* [18]) each MegaMol rank participates in. The final image is gathered to a single MegaMol rank that can send it to any (external) client over TCP/IP. Additionally, MegaMol renders the data set from 256 different view points, which can be combined into an image data base for post hoc exploration, similar to the ParaView Cinema [21] approach. These view points are arranged on a cigar shaped construct around the data set bounding box. On HPC resources, the simulation and MegaMol processes can be launched in separate MPI worlds. Thus, a failure in the visualization software cannot interfere with the simulation. A schematic overview of the setup can be found in Figure 1.

2.1 The ls1 Solver (and its Load-Balancing Scheme)

We use the multi-center Lennard-Jones 6-12 rigid potential [12], which is a very common approach for particle-based solvers. The computational complexity of the simulation originates from molecules being within each other's potential, as the potential generates the forces that alter the state variables of the molecules. Usually, a cutoff radius is employed to neglect the weak long-range effects (often chosen as 2.5σ, with σ being the cross section of a particle collision). The main computational load is thus due to pairs of particles closer than the cutoff, which correlates with the sheer number of particles, but is mainly influenced by the local density. In dynamic systems, a load rebalancing is required at regular intervals to compensate for particle fluctuation across local domains. The solver recalculates its domain decomposition and moves particles between ranks accordingly. In general, balancing leads to regions in the simulation domain with a higher density being refined into more subdivisions than lower density regions.

2.2 Solver-Visualization-Coupling

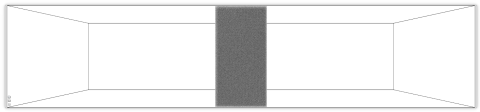
When a simulation job is started, each node ("SKX Node" in Figure 1) of the executing cluster is populated by a process for the simulation, communicating with each other via one MPI communicator, denoted as "MPI1". In the same step, an instance of MegaMol is launched for each node, acting as the visualization component ("MegaMol Visualization"). To facilitate a loose coupling, these MegaMol instances employ their own MPI communicator (denoted "MPI2"), which is, for example, used for compositing the final images via IceT [18]. Each instance merely occupies some resources, usually a few cores, sharing the same memory space as the simulation. There are no memory bottlenecks since ls1 is heavily compute-bound and thus has a very limited memory footprint. Before starting the actual computation, the solver tries to perform a handshake over a ZMQ socket [29]. After this handshake, the solver resumes its remaining setup and goes into the main computation loop. If that handshake fails, it still resumes the computation but without communicating data to MegaMol. From here on out, all coupling communication is only directed *towards* the visualization, yielding robustness against failure of the visualization component. At fixed simulation step intervals, the solver copies relevant simulation data to shared memory, such as particle positions and velocities. The only communication between the processes of the simulation in "MPI1" and the MegaMol instance residing in "MPI2" is a unidirectional update trigger facilitated via a ZMQ socket, activated after ls1 completes the copy to shared memory. This avoids the need for the two separate MPI worlds to directly interact. The MegaMol instance listening on the ZMQ socket catches this trigger. Processing the data chunks is discussed in the following paragraphs.

2.3 Visualization Data Flow

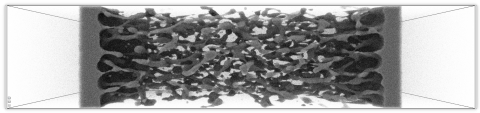
The final visualization is generated in two steps, a ray tracing step producing an intermediate frame buffer containing color and depth information, and a compositing step, aggregating the individual frame buffers into a final image, which is sent to the view node.

Raytracing Step:
For the rendering backend we use the Intel's ray tracing engine OSPRay [26], as the Stampede2 cluster at the Texas Advanced Computing Center (TACC), with which we are evaluating our approach, does not provide GPUs. Our approach is meant to be as general as possible to support other rendering backends as well, so we are explicitly not utilizing OSPRay's built-in distributed rendering. Note that for the measurements, progressive rendering has been changed to a fixed number of rays, to generate comparable results for the runtime measurements.

Compositing Step:
When the individual frame buffers on each node are ready, IceT [18] composites the final image. In general, MegaMol is able to connect to a remote instance and transfer image data and camera interaction to the distributed visualization, but we do not require any particular interaction to perform this case study.

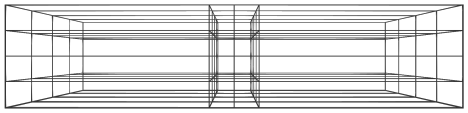


(a) step 0

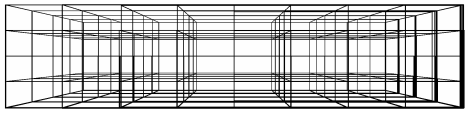


(b) step 25

Figure 2: Rendering of the examined system using OSPRay at 0 (a) and $25 \cdot 10^5$ (b) solver iterations.



(a) DIF step 0



(b) DIF step 25

(c) DOM

Figure 3: The decomposition adopts directly to the particle distribution. Here, (a) and (b) directly compare to (a) and (b) of Figure 2 with the DIF decomposition for 128 nodes. The domain decomposition (c) shows the standard spacial subdivision of a not load balanced simulation.

3 MEASUREMENTS AND DISCUSSION

To obtain our measurements, we deploy ls1 on the Stampede2 cluster at TACC. Each node features a two-way Intel Xeon Platinum 8160 with a total of 48 cores. We configured the solver and the visualization to run in three different per-node parallelization configurations. This per-node parallelization varies the amount of threads allocated to each of the simulation and the ray tracing engine. In all configurations, 46 cores are used and 2 cores are left for the OS, I/O and other background tasks to run. The simulation time steps used in our measurements are shown in Figure 2. Each time step contains $66{\cdot}10^6$ particles in total. For each simulation time step, we render 256 different configurations varying in camera position and orientation, as described above. The generated images can be used for post hoc analysis of the visualized data similar to O'Leary et al. [21]. This enables interaction with the data set just based on images, which is completely independent of the actual data size. The rendering resolution is 1920×1200 across all our measurements.

The distribution of particle data is performed according to the selected load balancing scheme. In the following, we denote the measurements pertaining to the static regular domain decomposition as "DOM", while the diffusion based dynamic load balancing is depicted with "DIF". In Figure 3 the boxes exemplify the domain decomposition resulting from different schemes at different simulation times. Every 20 steps the load is rebalanced, which has a direct influence on the simulation's per-step performance. The rebalancing changes the decomposition scheme (local bounding boxes) slightly to adopt for best performance. However, this performance optimization levels out and does not lead to further benefits, which is an indicator that the frequency where the load is rebalanced can be lowered. However, the rebalancing has to stay active due to the dynamic behavior of the simulation. The number of steps depends on the number of nodes, because the grid of local domain bounding boxes is much more fine-grained. Approximately, for eight nodes this takes only one or two iterations, and for 128 nodes this can take up to 40 iterations.

In Figure 4, we compare the run times per step of a simulation running with and without visualization running alongside on the same node. In addition, we show the performance impact of assigning different numbers of threads to the simulation and the visualization's ray tracing engine (OSPRay), respectively. For this, different configurations are used: 2/44, 4/42, and 8/38 (OSPRay threads vs. simulation threads). For the DIF load balancing scheme, we also take into account that the rebalancing occurs every 20^{th} step.

This is done by filtering the peaks in the simulation duration, and also discarding non-equilibrated values (1 to 40 iterations of the rebalancing). Therefore, the simulation times displayed in Figure 4 are consistent with numbers that actually occur in a production simulation run. An additional effect of the load balancing is a stronger thread scaling behavior. In comparison to the DOM simulation times that can be seen in Figure 4, the load-balanced results have a slightly steeper slope towards a smaller number of threads. This should be considered when tuning the coupled system for performance. In the case of normal domain decomposition (DOM), reducing the simulation's resources has no significant impact on the solver performance. We assume this is an indicator that without load balancing the simulation is not using the full capacity of its assigned resources. In reverse, this means that the performance overhead of assigning more resources to the visualization slightly increases in the case of dynamic load balancing.

Additionally, Figure 4 shows an anomaly. In the case of eight nodes and the DIF load balancing scheme, the simulation runs faster coupled to the visualization framework. We assume, this is due to the case that we allocate the same number of threads for the simulation, whether a visualization framework is coupled to the simulation or not. However, without the visualization framework, cores on a node remain unoccupied. The OS scheduler could switch threads to these unoccupied cores, which would generate an overhead. However, this assumption requires further investigation.

Figure 5 shows the rendering performance of the ray tracer evaluated over all camera positions using the median as value (the respective figure with error bars with the 10 % decile as lower and the 90 % decile as the upper bound, can be found in the appendix). While the impact of less threads on the simulation is very small, those extra threads that can be used for rendering increase the performance significantly. Additionally, there is no effect of the decomposition scheme for step 0 on the ray tracing performance, whereas there is a noticeable performance increase for step 25. The performance of the compositor shows a similar behavior with the exception that for step 0 there is already a noticeable speedup. However, the speedup already levels out for eight threads, which means that the benefit of larger number of threads for rendering is almost at its maximum in our specific case. Note that the time for writing images to disk is not included in our provided measurements (approximately 0.3 - 0.7 sec per image according to our experiments).

Each data point in Figure 5 is the median of a set of measurements that in fact show a large range of measured render times due to the varying camera configurations (plots with error bars shown in the appendix). To investigate this more closely, the ray tracing computation duration is plotted for every camera angle in Figure 6(a). In the case where the simulation is starting at time step 0, the bounding box is not completely filled with particles, which leads to many camera angles that do not capture any particle. We also observed a similar impact on compositing performance (see Figure 6(b)). However in some cases the view is completely

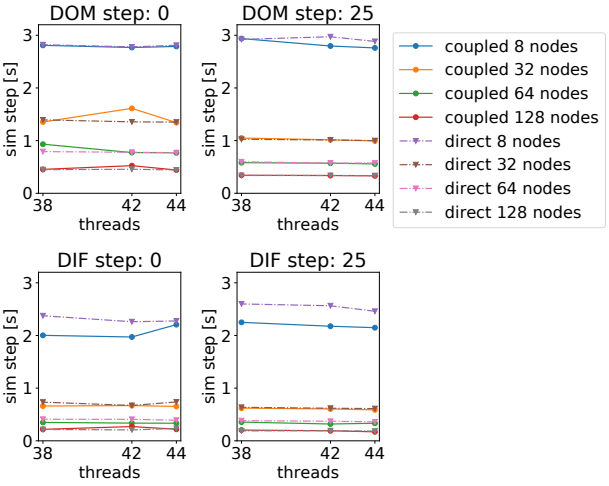
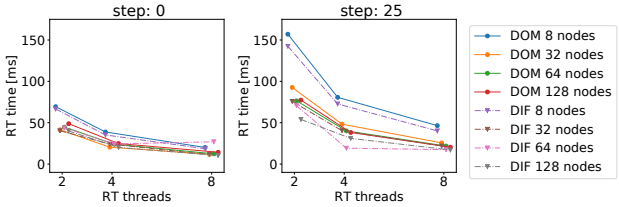
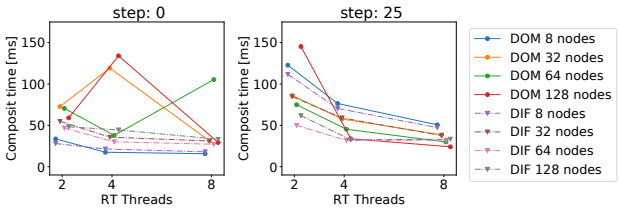


Figure 4: Influence of the visualization on the simulation. The left column shows the simulation starting at simulation time 0, in the right the simulation was resumed at a later point in time ($25 \cdot 10^5$). The rows differentiate between the two different decomposition schemes (DOM and DIF) that were used. The full lines are the runs with the simulation and the visualization running in situ (coupled), and the dotted lines show the results of just the simulation running (direct).



(a) Ray tracing



(b) Compositing

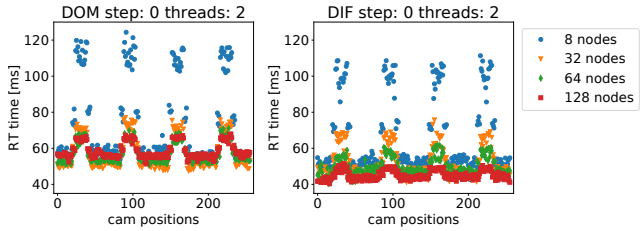
Figure 5: Plot (a) shows the ray tracing performance over the amount of assigned threads. Graph (b) represents the same measurement this time for the duration of the compositor. The plotted value represents the median of the measured data. The two domain decomposition schemes are plotted in the same graph for comparison. On the left the ray tracing performance for simulation iteration 0 is shown, on the right the iteration is started at $25 \cdot 10^5$ iterations. The data points are distributed around the actual value of threads (2, 4 and 8) to avoid visual clutter. Graphs with their corresponding error bars are found in the appendix.

filled with particles which leads to significantly longer computation time. In time step $25 \cdot 10^5$ the bounding box is filled more homogeneously with particles, and accordingly the variance in the rendering times is reduced. In addition, it can be seen that both the performance of ray tracing and compositing benefit from dynamic load balancing.

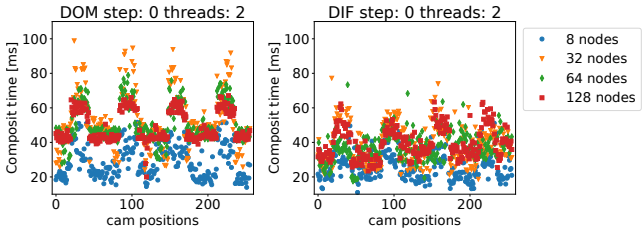
4 CONCLUSION AND LESSONS LEARNED

In this paper, we analyzed the performance impact of different work distribution schemes in the context of in situ visualization. In particular, we evaluated the effects of simulation load balancing across nodes as well as the assignment of threads to simulation and visualization on each node. We studied these effects for an in situ setup featuring a particle solver coupled to a ray tracing visualization system.

In general, we found that the investigated in situ coupling approach is quite lightweight and only induces relatively low overhead. According to our measurements, reducing the node level parallelization of the simulation only has a minor negative effect on the simulation's performance, while the visualization performance significantly benefits from accordingly increased computation resources. However, we could see that the simulation is generally more capable of efficiently utilizing more threads when load balancing is used. As a result of this, the performance overhead of



(a) Ray tracing



(b) Compositing

Figure 6: Diagram (a) plots the ray tracing times for the 256 different camera angles. Graph (b) shows the duration for image composition. Here, only the data for step 0 and 2/44 threads is plotted.

assigning more resources to the visualization slightly increases. Furthermore, in the examined case, the simulation's load balancing not only leads to a performance benefit for the solver but for the visualization as well. We conclude that distributing regions of dense particles both helps the particle solver as well as the sphere ray tracing. However, we assume that this characteristic is highly dependant on the solver and the employed distribution criteria, and balancing could even have harmful effects for visualization performance in other cases. This is a direction that we plan to investigate in more detail in future work.

ACKNOWLEDGMENTS

This research was partially funded by the German Bundesministerium für Bildung und Forschung (BMBF) as part of project "TaLPas" (Task-based Load Balancing and Auto-tuning in Particle Simulations), by the Baden-Württemberg Stiftung as part of project "DiHu" (High Performance Computing II grant) and by the Intel® Graphics and Visualization Institutes of XeLLENCE program. The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper. Additionally, the authors would like to thank the ls1 Mardyn development team for their support and Matthias Heinen for providing the simulation configurations.

REFERENCES

[1] Hasan Abbasi, Matthew Wolf, Greg Eisenhauer, Scott Klasky, Karsten Schwan, and Fang Zheng. 2010. DataStager: scalable data staging services for petascale applications. *Cluster Computing* 13, 3 (September 2010), 277–290. https://doi.org/10.1007/s10586-010-0135-6

[2] Utkarsh Ayachit, Andrew Bauer, Earl P. N. Duque, Greg Eisenhauer, Nicola Ferrier, Junmin Gu, Kenneth E. Jansen, Burlen Loring, Zarija Lukić, Suresh Menon, Dmitriy Morozov, Patrick O'Leary, Reetesh Ranjan, Michel Rasquin, Christopher P. Stone, Venkat Vishwanath, Gunther H. Weber, Brad Whitlock, Matthew Wolf, K. John Wu, and E. Wes Bethel. 2016. Performance Analysis, Design Considerations, and Applications of Extreme-scale in Situ Infrastructures. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'16)*. IEEE, 921–932. https://doi.org/10.1109/SC.2016.78

[3] 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 *First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV '15)*. ACM, New York, NY, USA, 25–29. https://doi.org/10.1145/2828612.2828624

[4] Utkarsh Ayachit, Brad Whitlock, Matthew Wolf, Burlen Loring, Berk Geveci, David Lonie, and E. Wes Bethel. 2016. The SENSEI Generic in Situ Interface. In *Second Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization (ISAV '16)*. IEEE, 40–44. https://doi.org/10.1109/ISAV.2016.013

[5] Hank Childs, Eric Brugger, Brad Whitlock, Jeremy Meredith, Sean Ahern, David Pugmire, Kathleen Biagas, Mark Miller, Cyrus Harrison, Gunther H. Weber, Hari Krishnan, Thomas Fogal, Allen Sanderson, Christoph Garth, E. Wes Bethel, David Camp, Oliver Rübel, Marc Durant, Jean M. Favre, and Paul Navrátil. 2012. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*. 357–372.

[6] Jai Dayal, Drew Bratcher, Greg Eisenhauer, Karsten Schwan, Matthew Wolf, Xuechen Zhang, Hasan Abbasi, Scott Klasky, and Norbert Podhorszki. 2014. Flexpath: Type-Based Publish/Subscribe System for Large-Scale Science Analytics. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 246–255. https://doi.org/10.1109/CCGrid.2014.104

[7] Thomas Fogal, Hank Childs, Siddharth Shankar, Jens Krüger, R Daniel Bergeron, and Philip Hatcher. 2010. Large Data Visualization on Distributed Memory Multi-GPU Clusters. In *High Performance Graphics*. The Eurographics Association, 57–66. https://doi.org/10.2312/EGGH/HPG10/057-066

[8] Steffen Frey and Thomas Ertl. 2011. Load Balancing Utilizing Data Redundancy in Distributed Volume Rendering. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'11)*. The Eurographics Association, 10. https://doi.org/10.2312/EGPGV/EGPGV11/051-060

[9] Patrick Gralka, Michael Becher, Matthias Braun, Florian Frieß, Christoph Müller, Tobias Rau, Karsten Schatz, Christoph Schulz, Michael Krone, Guido Reina, and Thomas Ertl. 2019. MegaMol – a comprehensive prototyping framework for visualizations. *The European Physical Journal Special Topics* 227, 14 (March 2019), 1817–1829. https://doi.org/10.1140/epjst/e2019-800167-5

[10] Sebastian Grottel, Michael Krone, Cristoph Müller, Guido Reina, and Thomas Ertl. 2015. MegaMol – A Prototyping Framework for Particle-Based Visualization. *IEEE Transactions on Visualization and Computer Graphics* 21, 2 (February 2015), 201–214. https://doi.org/10.1109/TVCG.2014.2350479

[11] Matthias Heinen, Jadran Vrabec, and Johann Fischer. 2016. Communication: Evaporation: Influence of heat transport in the liquid on the interface temperature and the particle flux. *The Journal of Chemical Physics* 145, 8 (August 2016), 081101. https://doi.org/10.1063/1.4961542

[12] J. E. Jones. 1924. On the Determination of Molecular Fields. II. From the Equation of State of a Gas. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 106, 738 (October 1924), 463–477. https://doi.org/10.1098/rspa.1924.0082

[13] Mark Kim, Tom Evans, Scott Klasky, and David Pugmire. 2017. In Situ Visualization of Radiation Transport Geometry. In *In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization (ISAV'17)*. ACM, New York, NY, USA, 7–11. https://doi.org/10.1145/3144769.3144770

[14] 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 *In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization (ISAV'17)*. ACM, New York, NY, USA, 42–46. https://doi.org/10.1145/3144769.3144778

[15] Nick Leaf, Bob Miller, and Kwan-Liu Ma. 2017. In Situ Video Encoding of Floating-Point Volume Data Using Special-Purpose Hardware for a Posteriori Rendering and Analysis. In *IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV'17)*. IEEE, 64–73. https://doi.org/10.1109/LDAV.2017.8231852

[16] Qing Liu, Jeremy Logan, Yuan Tian, Hasan Abbasi, Norbert Podhorszki, Jong Youl Choi, Scott Klasky, Roselyne Tchoua, Jay Lofstead, Ron Oldfield, Manish Parashar, Nagiza Samatova, Karsten Schwan, Arie Shoshani, Matthew Wolf, Kesheng Wu, and Weikuan Yu. 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. https://doi.org/10.1002/cpe.3125

[17] Stéphane Marchesin, Catherine Mongenet, and Jean-Michel Dischler. 2006. Dynamic Load Balancing for Parallel Volume Rendering. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'06)*. The Eurographics Association, 8. https://doi.org/10.2312/EGPGV/EGPGV06/043-050

[18] Kenneth Moreland, Wesley Kendall, Tom Peterka, and Jian Huang. 2011. An Image Compositing Solution at Scale. In *International Conference for High Performance*

Computing, Networking, Storage and Analysis (SC'11). ACM, New York, NY, USA, Article 25, 10 pages. https://doi.org/10.1145/2063384.2063417

[19] Kenneth Moreland, Christopher Sewell, Will Usher, Li-ta Lo, Jeremy Meredith, David Pugmire, James Kress, Hendrik Schroots, Kwan-Liu Ma, Hank Childs, Matthew Larsen, Chun-Ming Chen, Robert Maynard, and Berk Geveci. 2016. VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. *IEEE Computer Graphics and Applications* 36, 3 (May 2016), 48–58. https://doi.org/10.1109/MCG.2016.48

[20] Christoph Niethammer, Stefan Becker, Martin Bernreuther, Martin Buchholz, Wolfgang Eckhardt, Alexander Heinecke, Stephan Werth, Hans-Joachim Bungartz, Colin W. Glass, Hans Hasse, Jadran Vrabec, and Martin Horsch. 2014. ls1 mardyn: The Massively Parallel Molecular Dynamics Code for Large Systems. *Journal of Chemical Theory and Computation* 10, 10 (2014), 4455–4464. https://doi.org/10.1021/ct500169q

[21] Patrick O'Leary, James Ahrens, Sébastien Jourdain, Scott Wittenburg, David H. Rogers, and Mark Petersen. 2016. Cinema image-based in situ analysis and visualization of MPAS-ocean simulations. *Parallel Comput.* 55 (July 2016), 43–48. https://doi.org/10.1016/j.parco.2015.10.005

[22] Tobias Rau, Michael Krone, Guido Reina, and Thomas Ertl. 2017. Challenges and Opportunities using Software-Defined Visualization in MegaMol. In *Workshop on Visual Analytics, Information Visualization and Scientific Visualization (WVIS) in the 30th Conference on Graphics, Patterns and Images (SIBGRAPI'17)*. http://sibgrapi2017.ic.uff.br/e-proceedings/assets/papers/WVIS/WVIS2.pdf

[23] Will Usher, Silvio Rizzi, Ingo Wald, Jefferson Amstutz, Joseph Insley, Venkatram Vishwanath, Nicola Ferrier, Michael E. Papka, and Valerio Pascucci. 2018. libIS: A Lightweight Library for Flexible In Transit Visualization. In *Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV'18)*. ACM, New York, NY, USA, 33–38. https://doi.org/10.1145/3281464.3281466

[24] Venkatram Vishwanath, Mark Hereld, Vitali Morozov, and Michael E. Papka. 2011. Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. ACM, New York, NY, USA, Article 19, 11 pages. https://doi.org/10.1145/2063384.2063409

[25] Jadran Vrabec, Martin Bernreuther, Hans-Joachim Bungartz, Wei-Lin Chen, Wilfried Cordes, Robin Fingerhut, Colin W. Glass, Jürgen Gmehling, René Hamburger, Manfred Heilig, Matthias Heinen, Martin T. Horsch, Chieh-Ming Hsieh, Marco Hülsmann, Philip Jäger, Peter Klein, Sandra Knauer, Thorsten Köddermann, Andreas Köster, Kai Langenbach, Shiang-Tai Lin, Philipp Neumann, Jürgen Rarey, Dirk Reith, Gábor Rutkai, Michael Schappals, Martin Schenk, Andre Schedemann, Mandes Schönherr, Steffen Seckler, Simon Stephan, Katrin Stöbener, Nikola Tchipev, Amer Wafai, Stephan Werth, and Hans Hasse. 2018. SkaSim - Skalierbare HPC-Software für molekulare Simulationen in der chemischen Industrie. *Chemie Ingenieur Technik* 90, 3 (March 2018), 295–306. https://doi.org/10.1002/cite.201700113

[26] I Wald, GP Johnson, J Amstutz, C Brownlee, A Knoll, J Jeffers, J Günther, and P Navrátil. 2017. OSPRay - A CPU Ray Tracing Framework for Scientific Visualization. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (January 2017), 931–940. https://doi.org/10.1109/TVCG.2016.2599041

[27] Brad Whitlock, Jean M. Favre, and Jeremy S. Meredith. 2011. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'11)*. The Eurographics Association, 9. https://doi.org/10.2312/EGPGV/EGPGV11/101-109

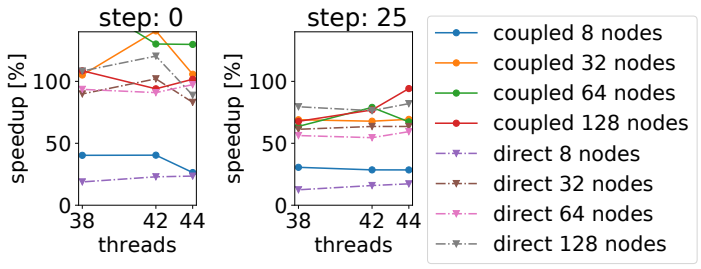
[28] Yucong (Chris) Ye, Tyson Neuroth, Franz Sauer, Kwan-Liu Ma, Giulio Borghesi, Aditya Konduri, Hemanth Kolla, and Jacqueline Chen. 2016. In Situ Generated Probability Distribution Functions for Interactive Post Hoc Visualization and Analysis. In *Symposium on Large Data Analysis and Visualization (LDAV'16)*. IEEE, 65–74. https://doi.org/10.1109/LDAV.2016.7874311

[29] ZeroMQ. 2019. ZeroMQ Messaging Library. Retrieved 2019-10-20 from https://zeromq.org/

[30] Fang Zheng, Hongbo Zou, Greg Eisenhauer, Karsten Schwan, Matthew Wolf, Jai Dayal, Tuan-Anh Nguyen, Jianting Cao, Hasan Abbasi, Scott Klasky, Norbert Podhorszki, and Hongfeng Yu. 2013. FlexIO: I/O Middleware for Location-Flexible Scientific Data Analytics. In *International Symposium on Parallel and Distributed Processing (IPDPS'13)*. IEEE, 320–331. https://doi.org/10.1109/IPDPS.2013.46

APPENDIX

For reference, we have included the full range of performed measurements and results below.



Plot of the reached speedup of the simulation due to load balancing in percent.

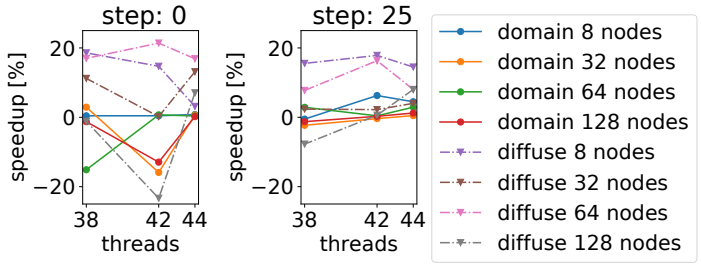
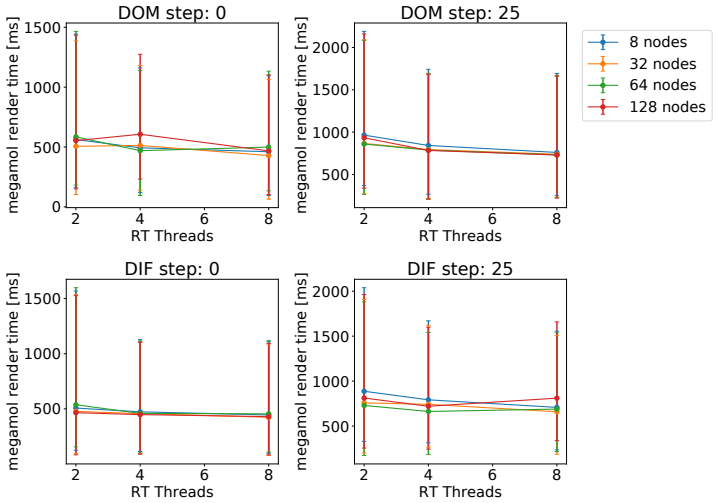
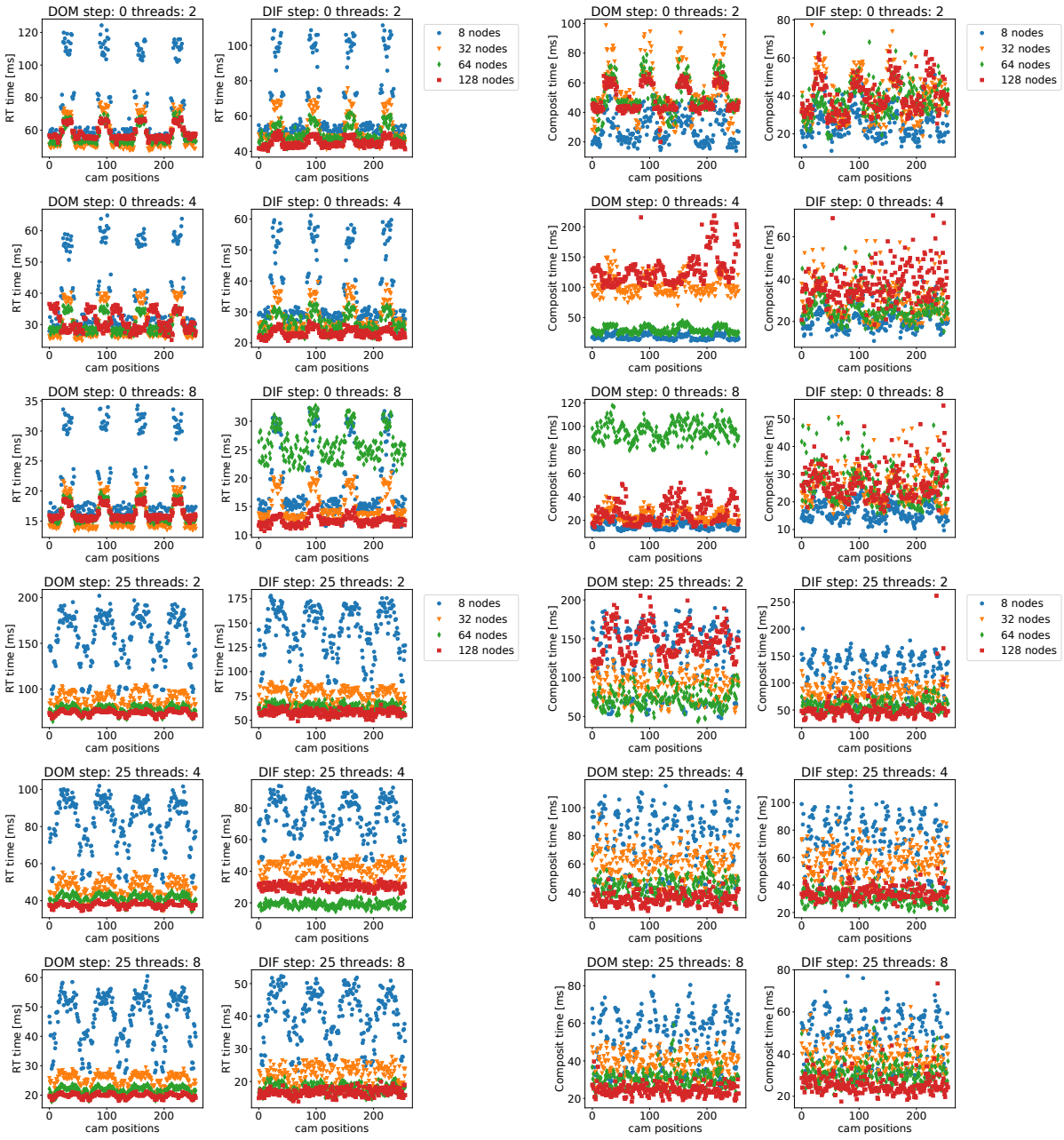


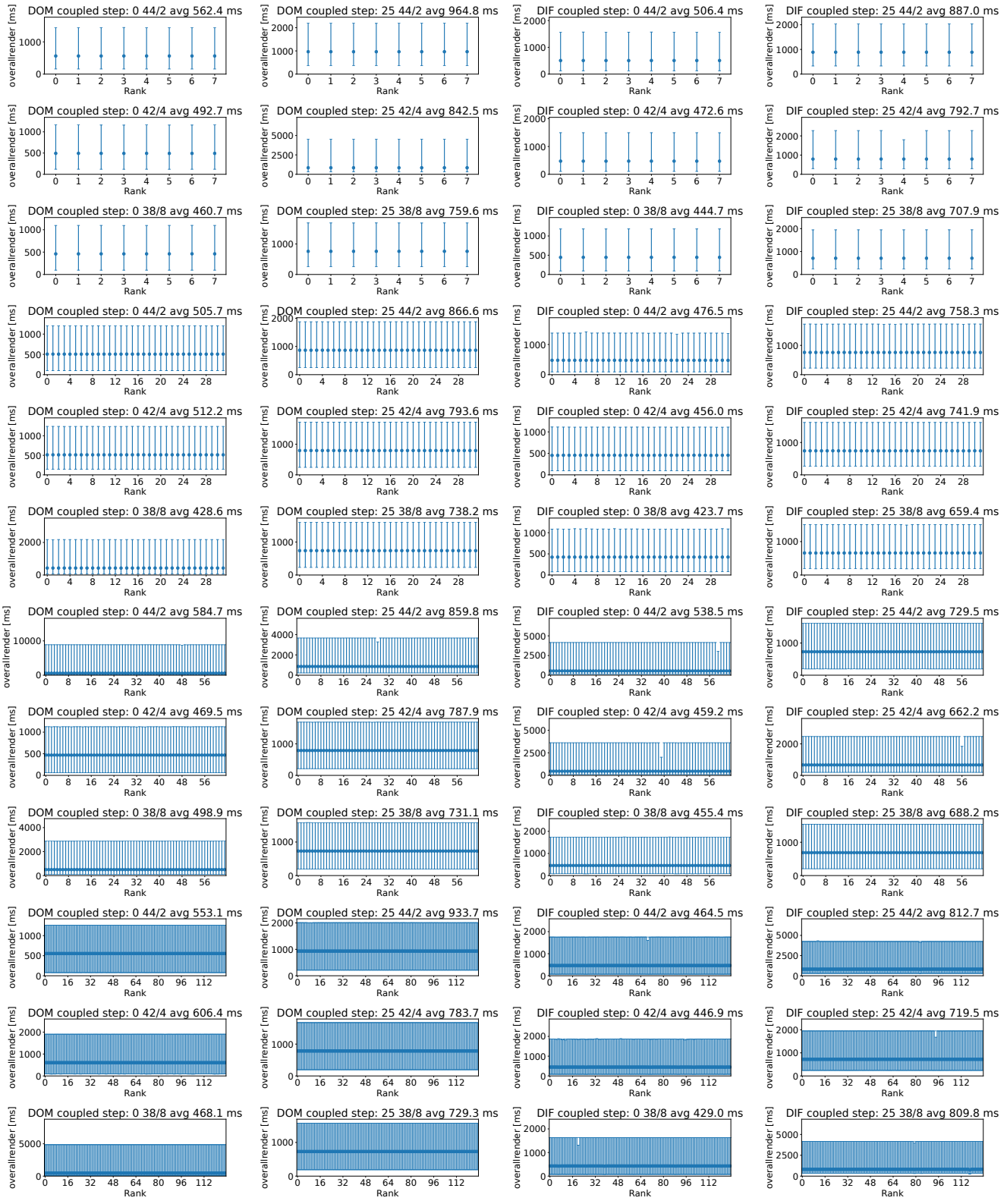
Figure of the relative speedup reached by coupling the simulation with the visualization. For time step 0 the accumulated speedup is 2.5% **and for time step 25 the speedup is** 4.3%**.**



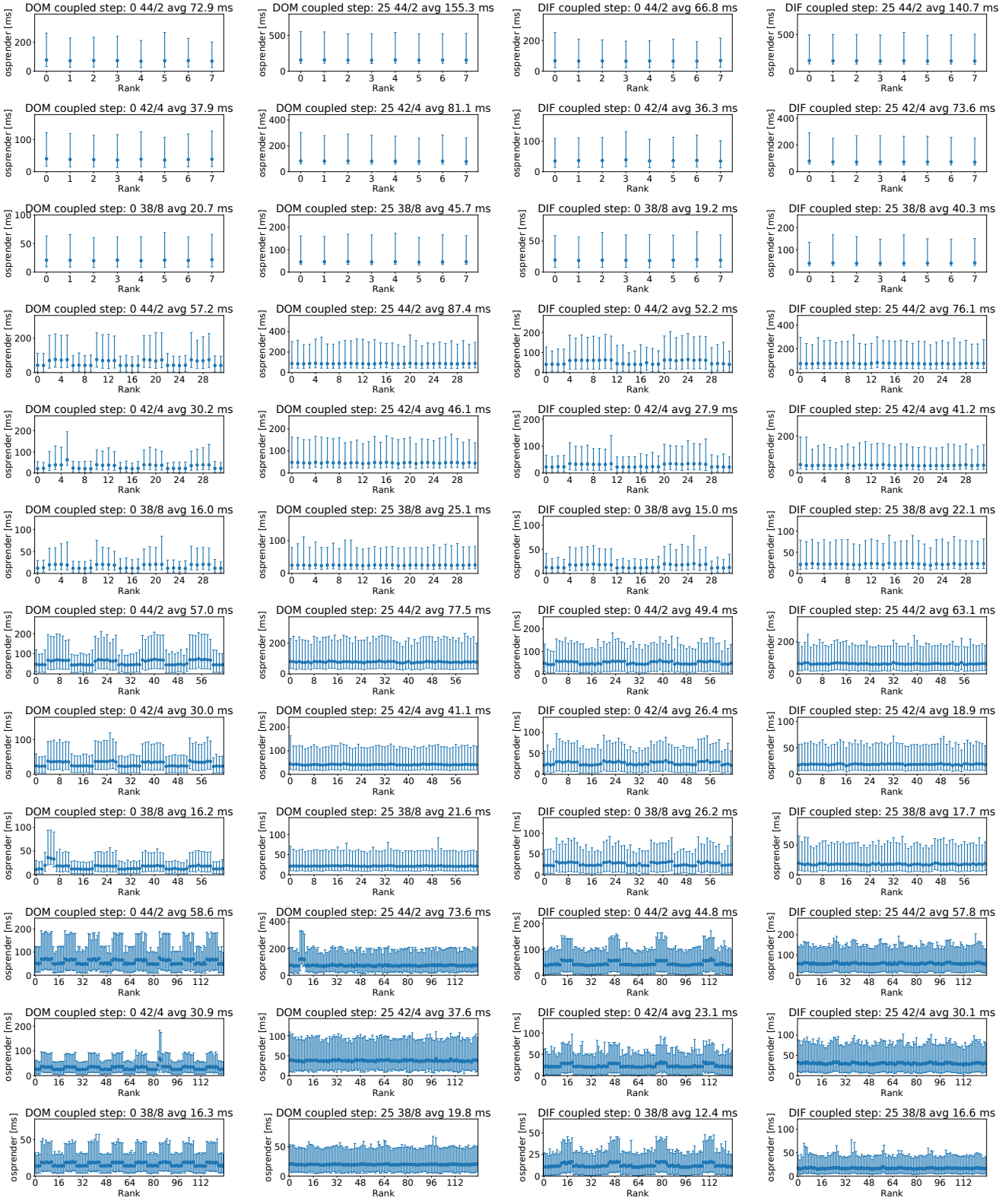
Depiction of the overall time for one visualization step for the two different domain decompositions and the two different iteration steps. Note that at each frame an image is encoded and stored to disk.



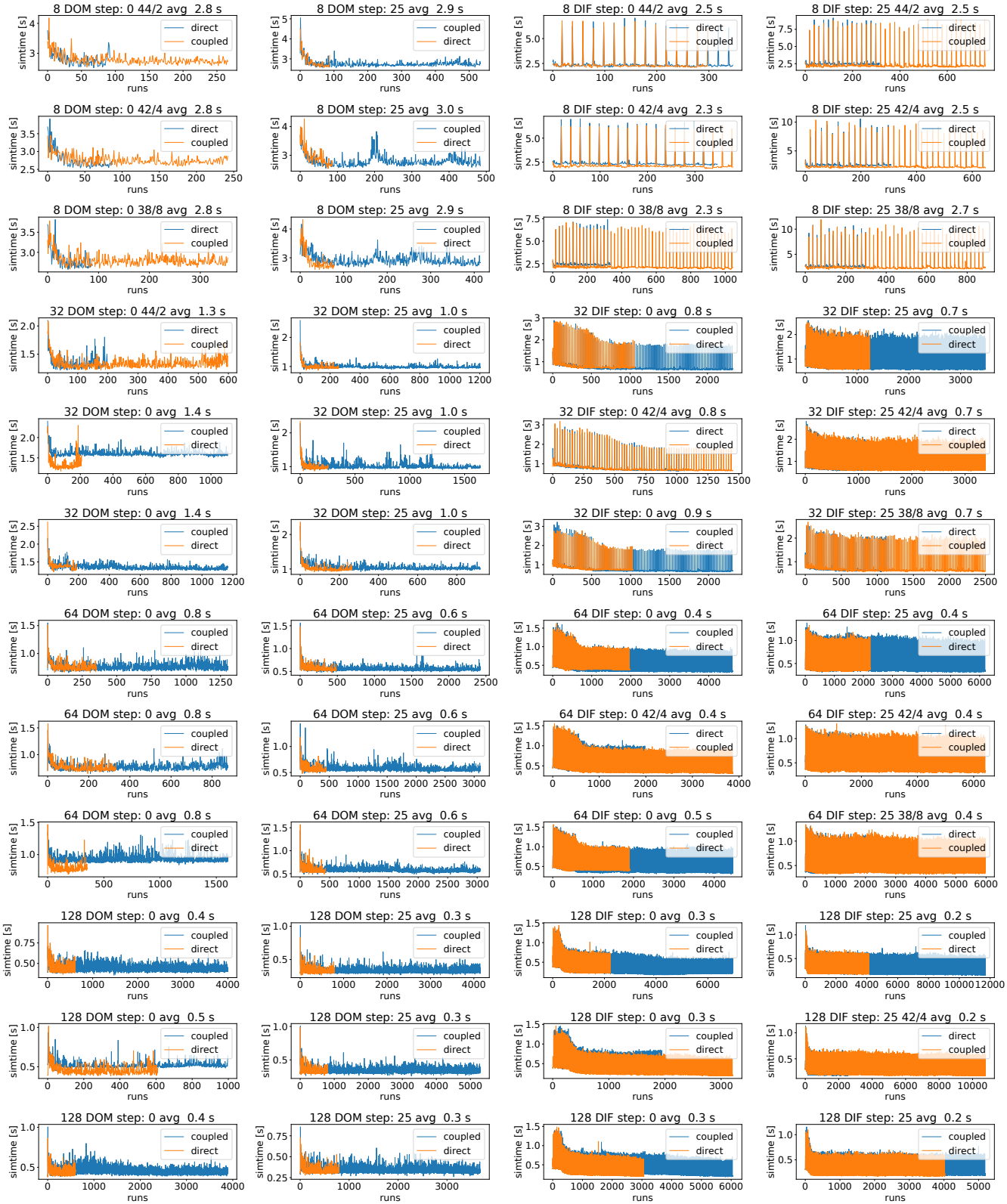
Plots of the ray tracing (left) and compositing (right) times for all used camera configurations.



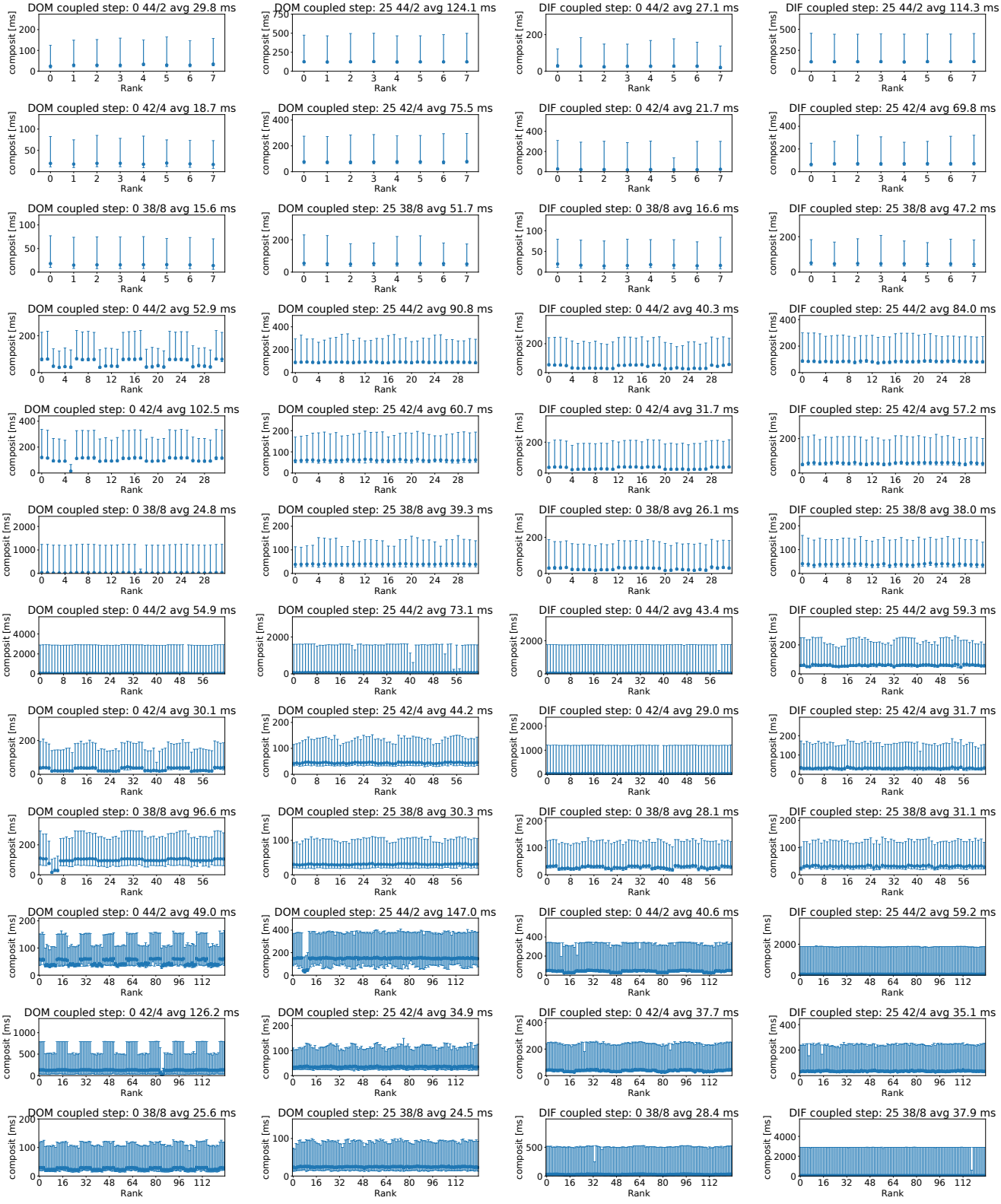
Plots of the overall time of one visualization step for each incorporated rank. Every plot represents a unique configuration of parameters.



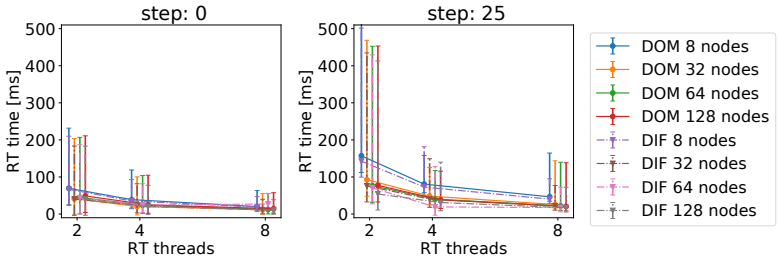
Graph collection of the ray tracing time for every single rank. Note that especially for step 0 an imbalance of load causes some ranks to use significantly more time for rendering.



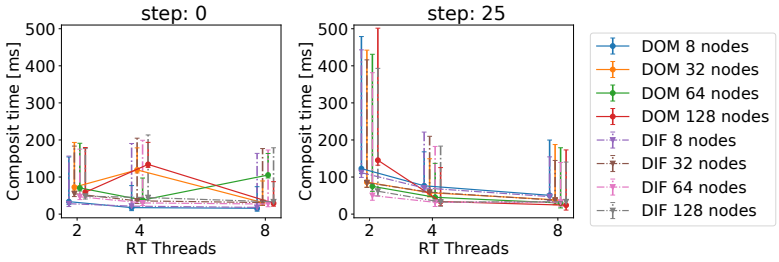
Plots of the simulation time of every configuration for each individual step. The varying length is due to measurements using a fixed time limit. Also, coupled runs should run longer to sample each camera configuration.



Plot of the time the image compositing is using during visualization.



(a) Ray tracing



(b) Compositing

Figure 7: Plot (a) shows the ray tracing performance over the amount of assigned threads. Graph (b) represents the same measurement this time for the duration of the compositor. As a representation we chose the median as value and the 10 % **decile and** 90 % **decile as error bar.**