Analysing the Scalability of Climate Codes Using New Features of Scalasca

M. Harlacher, A. Calotoiu, J. Dennis, F. Wolf

published in

NIC Symposium 2016
K. Binder, M. Müller, M. Kremer, A. Schnurpfeil (Editors)

Forschungszentrum Jülich GmbH,
John von Neumann Institute for Computing (NIC),
Schriften des Forschungszentrums Jülich, NIC Series, Vol. 48,
ISBN 978-3-95806-109-5, pp. 343.
http://hdl.handle.net/2128/9842

Analysing the Scalability of Climate Codes Using New Features of Scalasca

Monika Harlacher[1], **Alexandru Calotoiu**[2], **John Dennis**[3], **and Felix Wolf**[2]

[1] Universität Siegen, 57068 Siegen, Germany
E-mail: monika.luecke@uni-siegen.de

[2] Technische Universität Darmstadt, 64293 Darmstadt, Germany
E-mail: {calotoiu, wolf}@cs.tu-darmstadt.de

[3] National Center for Atmospheric Research, Boulder, CO 80307, USA
E-mail: dennis@ucar.edu

This paper shows how recently developed features of the performance analysis tool Scalasca helped gain important insights into the performance behaviour of state-of-the-art climate codes in the CESM (Community Earth System Model) ensemble. Particular emphasis is given to the load balance of the sea-ice model and the scaling behaviour of the atmospheric model. The presented work is a result of the project *Enabling Climate Simulation at Extreme Scale*, which has been funded through the G8 Research Councils Initiative on Multilateral Research Funding.

1 Introduction

Policy decisions for mitigating climate change or adapting to it are subjects of great discussion throughout the world. Uninformed decisions will impose a heavy cost on future generations, both financial and human. Therefore, it is essential to reduce the current uncertainties about future climate changes and their impact by running climate simulations at 1,000 times larger scales than today. Exascale supercomputers are expected to appear around 2020, featuring a hierarchical design and gathering 100 millions of computing cores. The numerical models of the physics, chemistry, and biology affecting the climate system need to be improved to run efficiently on these extreme systems. Without improvement, these codes will not produce simulations results required to respond to the societal and economical challenges of climate change.

The objective of the G8 ECS (Enabling Climate Simulation at Extreme Scale, 2011-2014) project was to investigate how to run efficiently climate simulations on future exascale systems and get correct results. The project gathered researchers in climate and computer science from Canada, Germany, Japan, Spain, and USA to focus on three main topics: (i) how to complete simulations with correct results despite frequent system failures, (ii) how to exploit hierarchical computers with hardware accelerators close to their peak performance and (iii) how to run efficient simulations with very high numbers of threads.

This article concentrates on the third aspect – the scalability. Subject of the study is the CESM (Community Earth System Model)[1], a fully-coupled ensemble of climate codes maintained at the National Center for Atmospheric Research. It provides state-of-the-art computer simulations of the Earth's past, present, and future climate states. In this study, we analyse the performance of selected CESM codes using new technologies developed in the framework of the Scalasca project[2], a performance analysis toolset designed for

highly scalable applications. All performance experiments were carried out on the IBM BlueGene/Q system JUQUEEN located at the Jülich Supercomputing Centre.

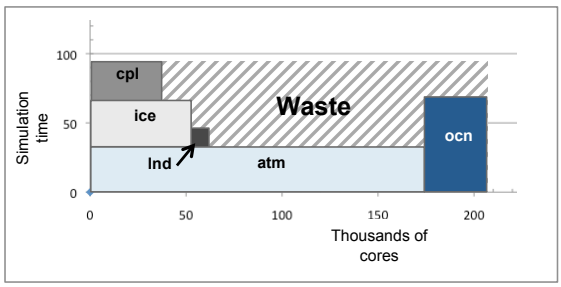


Figure 1. A common configuration of running CESM.

The scalability of CESM is predominantly influenced by the spatial resolution and the performance of the most time-consuming modules, the atmospheric model CAM and the ocean model POP. CAM features very good scalability, while POP scales merely up to a few thousands of cores and is therefore considered as the most severe scalability bottleneck. Since in a common configuration (Fig. 1), the sea-ice model CICE is run alongside POP, one way of compensating for POP's limited scalability is to expand CICE's. This is why major efforts concentrated on this module of CESM. In this article, we focus on CICE and CAM, two studies in which we demonstrate new performance-analysis methods. Whereas we looked at the full model for CICE, we studied CAM using its dynamical core, which is called HOMME[3].

Initially, we used only the vanilla version of Scalasca[2]. With the help of Scalasca, we identified critical performance factors and established causal relations between them. During the project, we also used recently added features, such as delay analysis[4], and added new features on our own whenever necessary. Since load imbalance emerged as a major theme from this initial study, we finally came up with the new idea of a load-balancing simulator that can be used to compare the benefits of different load-balancing strategies without changing the code. We created a design of the simulator, implemented a prototype, and present preliminary results for CICE, which suggest significant optimisation potential. Finally, over the course of the project we developed a novel tool for the automatic detection of scalability bugs. The tool, which is based on automatically generated performance models, was developed in collaboration with the DFG-funded project Catwalk. We applied it to assess HOMME's potential for running at very large scales. More details about the three analysis methods and their results are given below.

2 CICE – The Sea Ice Model

Our study of the sea ice model revealed significant load imbalance. In an early version of the code, all processes receive roughly the same amount of grid cells, which implies huge computational imbalances across processes because the distribution of ice is not uniform. This load imbalance entails wait states in communication sections.

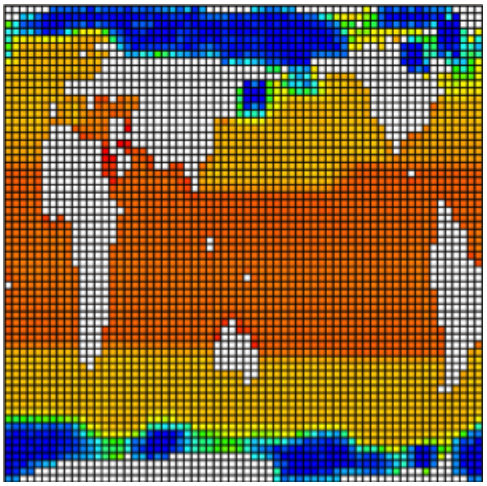


Figure 2. Indirect waiting time for one communication call path in the CESM sea-ice model. Indirect waiting time is waiting time caused by previous wait states as opposed to direct delay. The higher the colour temperature the larger the amount of indirect waiting time.

With our delay analysis extension, we were able to detect a phenomenon caused by the nearest-neighbour exchange pattern used in the application: the propagation of wait states. Near the poles, at the boundary between computationally intensive and less intensive regions, processes with less work wait for messages from processes with more work. This lets them appear overloaded to their neighbours closer to the equator, leading to a chain reaction along which wait states propagate all the way from the poles to the equatorial regions. Fig. 2 depicts the indirect waiting time, which is the waiting time caused by previous wait states and not direct delay through computation overload. It increases with increasing distance from computationally intense regions (poles).

To visualise the performance characteristics with a more advanced domain decomposition, as shown in Fig. 4, we recorded the required decomposition information at runtime. We extended our study to the at that time latest CESM release, 1.0.4, with a decomposition based on space-filling curves as a method for load balancing. This configuration is used by NCAR for high-resolution climate simulations. Already by comparison of the application profiles gained with Scalasca, we were able to measure the performance impact of the different domain decomposition strategies. We experienced a very strong dependence of the application's execution time on the choice of domain distribution parameters, such as block size and aspect ratio, maximum number of blocks that can be assigned to a process and the kind of the space-filling curve.

Another functionality recently added to Scalasca is the reconstruction of the application's critical path[5], the longest execution sequence without wait states in a parallel program. The critical path identifies the activities that determine the overall program runtime, and are therefore preferable candidates for performance optimisation. The higher-level function "evp_haloupdate3dr8" in the dynamics, including communication setup and nearest neighbour data exchange, contributes 27%. Jointly with the computational routines "evp" in the dynamics (with 17%) and "compute_dedd" in the radiation step (16%), these

functions have the largest share of the critical path. The latter two also show high critical imbalance, the time difference between a call path's contribution to the critical path and the average time spent in the call path across all processes. In a perfectly balanced program, their impact should only amount to 12.5% and 2.9% of the total runtime, respectively. This indicates that the highest potential for performance improvement via better load balancing rests in the function "compute_dedd".

2.1 Load-Balancing Simulation

In our studies on the scalability and efficiency of CICE, we were able to trace poor scalability of some communication routines back to load imbalance in associated computation phases. Although load balance is crucial for efficient resource utilisation, it is often very difficult for application developers to find a suitable load-balancing strategy that fits their specific problem. The effectiveness of a strategy depends not only on the specific combination of balancing criteria, e.g., computational weight and communication load, but also on the application behaviour, e.g., the communication pattern and sequence of computation and communication phases. Moreover, implementation and test of different load balancing strategies are usually not possible without major code surgery. Therefore, we designed and implemented a load-balancing simulator as a software engineering tool that enables developers to easily test and experiment with different load-balancing strategies. The load-balancing simulator facilitates experiments with different load-balancing strategies and communication patterns without cumbersome analytical comparison or time consuming modifications of the real code and subsequent tests. This information aids developers in choosing a specific method and gives them a guideline whether or not the per-

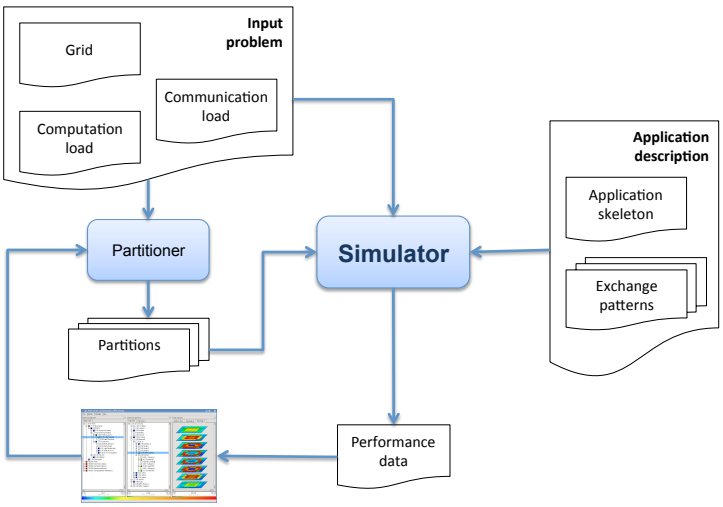
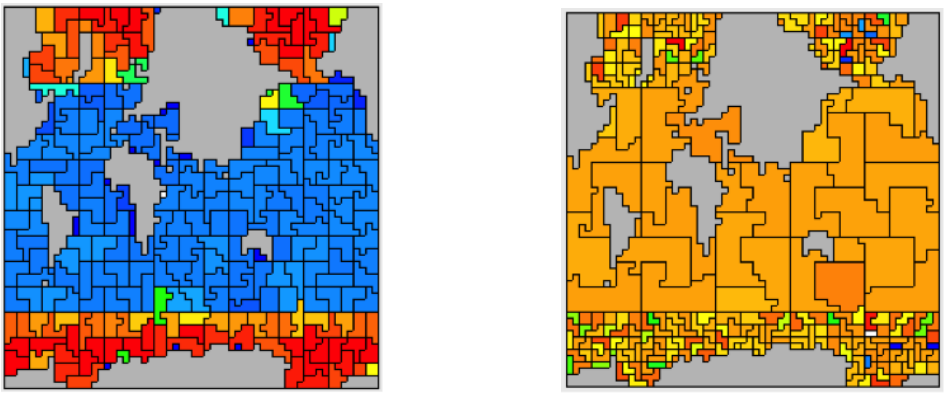


Figure 3. Architecture of the load-balancing simulator. Based on an abstract problem and application description, the simulator re-enacts the application's communication behaviour under the assumption of different partitions, facilitating an easy comparison in experiments.



(a) Original partition (b) Alternative partition

Figure 4. The CESM sea ice model shows severe computational load imbalance between regions of sea ice and open ocean, resulting in large MPI wait time. Using the load-balancing simulator, we replayed this behaviour (left image) and compared it to an alternative partition (right image). The improved load balance reduces MPI wait time and suggests a speedup of up to 2.7 on 256 cores.

formance improvement justifies the effort of software re-engineering. As an example, we applied the load-balancing simulator to CICE and examined its behaviour under different load-balancing strategies.

The architecture of the simulator is depicted in Fig. 3. The simulator operates on a graph structure, where vertices store the computational weight of a geometry item, and edges describe the communication links and volumes between them. This enables representations of arbitrary static grid types, e.g., structured or unstructured meshes of any shape, dimension, and element outline. The simulator also maintains an abstract description of the application's computation and communication phases to re-enact its behaviour. An application usually has multiple computation phases with communication phases in between. A computation phase is a recurring part in the application where a process executes some local operations. The time required for these operations is specified in the form of vertex weights, one constant weight for each computation phase. Computation is simulated by performing dummy computations lasting the specified amount of time. During communication phases, the simulator re-enacts the specified communication operation, e.g., a stencil operation, with dummy messages of realistic size. The simulator replays the predefined sequence of phases to show how the application would perform under a given input partition. The user can directly compare the runtime of simulations with different partitions or investigate the simulated application behaviour in more detail with performance analysis tools such as Scalasca.

The modular framework of the simulator not only provides pre-defined implementations of common communication patterns, but also allows the user to extend the simulator by adding more complex phase types or by exchanging the underlying communication library. The load-balancing simulator enables scientific application developers to study their application's behaviour and the associated potential performance yield without cumbersome analytical comparison or time consuming modifications of the real code and subse-

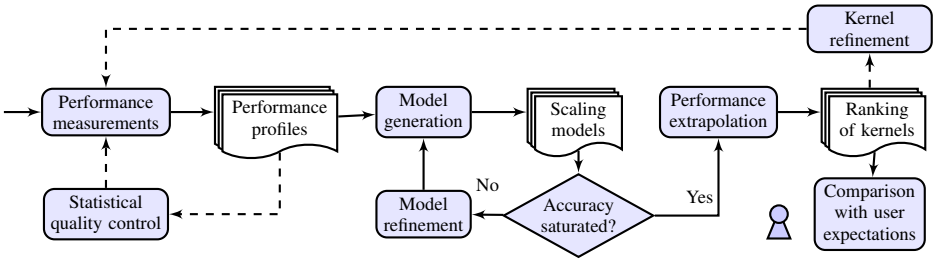


Figure 5. Workflow of scalability-bug detection. Solid boxes represent actions or transformations, and banners their inputs and outputs. Dashed arrows indicate optional paths taken after user decisions.

quent tests. We initially used the simulator to compare the current partition of the sea ice simulation to an alternative partition. In contrast to the current partition, the alternative partition does not restrict block sizes and is based on measured computational weights instead of the probability of sea ice. We estimated a speedup after reconfiguration of 2.7 for 256 cores. Note that the speedup numbers do not yet take into account that to lift the limit on block sizes it would be necessary to allocate the memory dynamically, which might prohibit some compiler optimisations. An investigation of this issue is still in progress.

3 HOMME – The Dynamical Core of the Atmospheric Model

Many parallel applications suffer from latent performance limitations that may prevent them from scaling to larger machine sizes. Often, such scalability bugs manifest themselves only when an attempt to scale the code is actually being made - a point where remediation can be difficult. However, creating analytical performance models that would allow such issues to be pinpointed earlier is so laborious that application developers attempt it at most for a few selected kernels, running the risk of missing harmful bottlenecks. We designed a lightweight performance modelling tool that improves both coverage and speed of this scalability analysis[6]. Generating an empirical performance model automatically for each part of a parallel program, we can easily identify those parts that will reduce performance at larger core counts. Using the HOMME climate simulation as an example, we demonstrated that scalability bugs are not confined to those routines usually chosen as kernels.

Fig. 5 gives an overview of the different steps necessary to find scalability bugs using our method, whose details we explain further below. To ensure a statistically relevant set of performance data, profile measurements may have to be repeated several times - at least on systems subject to jitter. This is done in the optional statistical quality control step. Once this is accomplished, we apply regression to obtain a coarse performance model for every possible program region. These models then undergo an iterative refinement process until the model quality has reached a saturation point. To arrange the program regions in a ranked list, we extrapolate the performance either to a specific target process scale or to infinity, which means we use the asymptotic behaviour as the basis of our comparison. Finally, if the granularity of our program regions is not sufficient to arrive at an actionable recommendation, performance measurements, and thus the kernels under investigation,

can be further refined via more detailed instrumentation.

Going beyond models for just runtime, we now also started to generate empirical models that allow projections for application requirements. Application requirements can be anything such as the required number of floating-point operations, network messages, transmitted bytes, or even memory consumption. System designers can use process-scaling models in tandem with problem-scaling models and the specification of a candidate system to determine the resource usage of an application execution with a certain problem size. Once analytical models are established for an interesting set of requirements, the designer can use them to "play" with configurations such as the amount of memory per node or the network injection speed etc..

To showcase how our tool helps to find hidden scalability bugs in a production code for which no performance model was available, we applied it to HOMME. Being designed with scalability in mind, it employs spectral element and discontinuous Galerkin methods on a cubed sphere tiled with quadrilateral elements. While experiences in the past did not indicate any scalability issues at up to 100,000 processes, HOMME has never been subjected to a systematic scalability study. The result of our analysis was the identification of two scalability issues, one of which was previously unknown[6]. The unknown issue was found in the initialisation and is a reduction needed to funnel data to dedicated I/O processes. It is of the kind usually overlooked when modelling manually. Fig. 6 shows the projected execution times of these two problematic kernels. The initialisation problem is expected to become serious already before hitting one million processes.

The requirements analysis of HOMME revealed some potential future bottlenecks that would need to be mitigated or removed to achieve extreme scaling potential. Specifically, two MPI-collective call paths ending in Allreduce and Bcast show prohibitively growing message sizes ($O(p)$ and $O(p \cdot log(p))$, respectively).

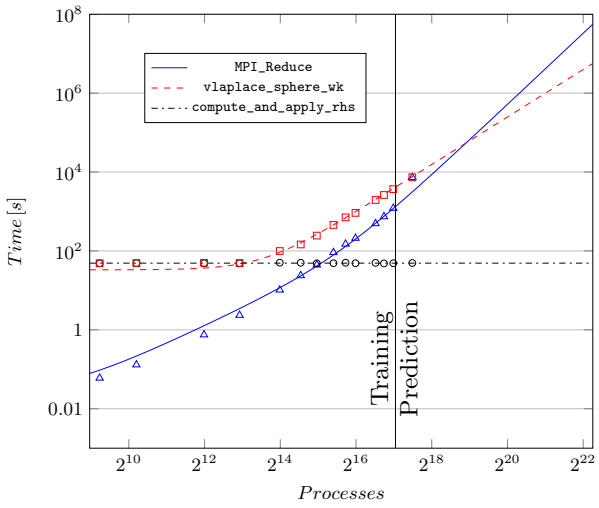


Figure 6. Projected execution times of selected functions in HOMME, two of them representing scalability bottlenecks.

4 Outlook

In the future, we plan to continue our work on load-balance optimisation and requirements analysis for extreme scalability. With respect to the former, we plan to enhance the design of the load-balancing simulator mainly with simplified usage in mind. This includes a more automated way of extracting computation and communication weights from the application and utilities for calibration and validation of the simulation's accuracy. We also plan to enrich the current portfolio of supported load-balancing strategies. Finally, we want to facilitate the simulation of dynamic strategies by modelling them as a sequence of (static) balancing steps.

Acknowledgements

This work received funding through the G8 Research Councils Initiative on Multilateral Research, Interdisciplinary Program on Application Software towards Exascale Computing for Global Scale Issues. Furthermore, the authors gratefully acknowledge the computing time granted by the JARA-HPC Vergabegremium and provided on the JARA-HPC Partition part of the supercomputer JUQUEEN at Forschungszentrum Jülich.

References

1. P. H. Worley, A. A. Mirin, A. P. Craig, M. A. Taylor, J. M. Dennis, and M. Vertenstein, *Performance of the community earth system model*, in: Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC11), ACM, 2011.
2. M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, and B. Mohr, *The Scalasca performance toolset architecture*, Concurrency and Computation: Practice and Experience, **22**, no. 6, 702–719, Apr. 2010.
3. J. M. Dennis, J. Edwards, K. J. Evans, O. Guba, P. H. Lauritzen, A. A. Mirin, A. St-Cyr, M. A. Taylor, and P. H. Worley, *CAM-SE: A scalable spectral element dynamical core for the community atmosphere model*, Intl. Journal of High Performance Computing Applications, **26**, no. 1, 74–89, 2012.
4. D. Böhme, M. Geimer, F. Wolf, and L. Arnold, *Identifying the root causes of wait states in large-scale parallel applications*, in: Proc. of the 39th International Conference on Parallel Processing (ICPP), San Diego, CA, USA, pp. 90–100, IEEE Computer Society, Sept. 2010, Best Paper Award.
5. D. Böhme, B. R. de Supinski, M. Geimer, M. Schulz, and F. Wolf, *Scalable Critical-Path Based Performance Analysis*, in: Proc. of the 26th IEEE International Parallel & Distributed Processing Symposium (IPDPS), Shanghai, China, pp. 1330–1340, IEEE Computer Society, May 2012.
6. A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, *Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes*, in: Proc. of the ACM/IEEE Conference on Supercomputing (SC13), Denver, CO, USA, pp. 1–12, ACM, November 2013.