Performance Optimization and Load-Balancing Modeling for Superparametrization by 3D LES

Gijs van den Oord
Netherlands eScience Center
Amsterdam, the Netherlands

Maria Chertova
Netherlands eScience Center
Amsterdam, the Netherlands

Fredrik Jansson*
Centrum voor Wiskunde en Informatica
Amsterdam, the Netherlands

Inti Pelupessy
Netherlands eScience Center
Amsterdam, the Netherlands

Pier Siebesma
Delft University of Technology
Delft, the Netherlands

Daan Crommelin†
Centrum voor Wiskunde en Informatica
Amsterdam, the Netherlands

ABSTRACT

In order to eliminate climate uncertainty w.r.t. cloud and convection parametrizations, superpramaterization (SP) [1] has emerged as one of the possible ways forward. We have implemented (regional) superparametrization of the ECMWF weather model OpenIFS [2] by cloud-resolving, three-dimensional large-eddy simulations. This setup, described in [3], contains a two-way coupling between a global meteorological model that resolves large-scale dynamics, with many local instances of the Dutch Atmospheric Large Eddy Simulation (DALES) [4], resolving cloud and boundary layer physics. The model is currently prohibitively expensive to run over climate or even seasonal time scales, and a global SP requires the allocation of millions of cores. In this paper, we study the performance and scaling behavior of the LES models and the coupling code and present our implemented optimizations. We mimic the observed load imbalance with a simple performance model and present strategies to improve hardware utilization in order to assess the feasibility of a world-covering superparametrization. We conclude that (quasi-)dynamical load-balancing can significantly reduce the runtime for such large-scale systems with wide variability in LES time-stepping speeds.

CCS CONCEPTS

• **Applied computing → Earth and atmospheric sciences**; • **Computing methodologies → Multiscale systems**; Massively parallel algorithms; *Massively parallel and high-performance simulations.*

KEYWORDS

Weather & climate simulation, Superparametrization, Multiscale modeling, Load-balancing

*Also with Delft University of Technology.
†Also with Korteweg-de Vries Institute, University of Amsterdam.

PASC '21, July 5–9, 2021, Geneva, Switzerland
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8563-3/21/07.
https://doi.org/10.1145/3468267.3470611

ACM Reference Format:
Gijs van den Oord, Maria Chertova, Fredrik Jansson, Inti Pelupessy, Pier Siebesma, and Daan Crommelin. 2021. Performance Optimization and Load-Balancing Modeling for Superparametrization by 3D LES. In *Platform for Advanced Scientific Computing Conference (PASC '21), July 5–9, 2021, Geneva, Switzerland.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3468267.3470611

Introduction

One of the dominant uncertainties in current climate projections is the estimation of cloud feedback; whereas it is certain that clouds play a crucial role in the development of the Earth's albedo and therefore climate sensitivity, our global circulation models (GCM) do not yet possess the resolution to represent realistically boundary layer turbulence and convection that determine cloud fields and emerging mesoscale organization [5]. Furthermore, there is high demand for realistic meteorological forecasts at sub-kilometer resolutions, ranging from flood risk assessment to the short-term prediction of wind farm yields.

Properly resolving the turbulent overturning motion of air and humidity within the atmospheric boundary layer requires grid resolutions typically of the order of 100 m, which is still orders of magnitude beyond state-of-the art GCM's and non-hydrostatic regional weather models. Hence these models rely on a complex system of process parametrizations, which gives rise to model uncertainties and biases [6].

On the other end of the spectrum of model resolutions, the large eddy simulations (LES) reside. At these scales, turbulence is well represented due to self-similarity in the inertial sub-range, convective up- and downdrafts arise from explicitly resolved dynamics and mesoscale phenomena such as cold pools emerge naturally. It is, however, computationally not feasible to cover the globe with a single LES.

One strategy to overcome this barrier is *superparametrization* (SP) [1], a technique where GCM parametrizations have been replaced by independent sub-models within each grid column. The basic idea is to force the sub-models by the GCM state and let them re-distribute momentum, heat, and humidity vertically. The forcing on the LES models has a relaxation period equal to the GCM time step and acts purely with respect to the horizontal slab average $\langle q \rangle$,

$$f_q(t_n) = \frac{Q(t_n) - \langle q(t_n) \rangle}{\Delta t}, \tag{1}$$

where Q and q denote the corresponding prognostic state variables in respectively the GCM and small-scale models. After progressing the latter towards the arrival point of the global model, the models are coupled via a local tendency proportional to the change in the horizontally-averaged state,

$$P_Q^{n+1} = \frac{\langle q(t_n + \Delta t)\rangle - \langle q(t_n)\rangle}{\Delta t} \tag{2}$$

The advantage of this approach is firstly practical: with moderate adjustments to the weather model code we can replace its physics parametrization scheme with a more realistic cloud-resolving submodel. Secondly, there are performance benefits as the LES models can run completely independently –they usually have periodic boundary conditions in the horizontal domain– which drastically limits inter-node communication[1].

From a performance modeling perspective, it is expected (and observed) that the resulting performance of the SP model is determined by the LES models, which have much smaller time steps than the global model. It is therefore essential that the implementation of SP allows the LES instances to run in parallel while the GCM awaits the time stepping and determines the collective synchronization point.

The purpose of this paper is to study the performance scaling of this model with growing number of superparametrizing LES instances, describe optimizations that we have implemented and explore new strategies to reduce the time-to-solution or consumed resources. Our parallel coupling implementation, load-balancing strategy and performance modeling may apply to other multiscale systems where micro-models run concurrently and independently to simulate sub-grid features of a large-scale model.

Superparametrization of OpenIFS by DALES

In our setup [3], we have coupled the ECMWF OpenIFS code to DALES through the coupling framework OMUSE [7]. This Python package is based upon the AMUSE platform [8] which provides automatic communication between a master Python script and the model components [9], wrapped into Python classes. This interface is nonrestrictive and provides the flexibility to implement the lockstep procedure of SP. The OMUSE framework provides a number of services to expedite coupled Earth system models, e.g. grid-data structures, unit conversions and model state handling, and we will use this acronym to refer to this entire software layer from here on. Using Python as the coupling code language has enabled us to obtain a scientifically sound coupling and dedicated diagnostics in a relative short development period. Nevertheless, some unexpected bottlenecks have emerged too, which we elaborate on later.

Our SP implementation of OpenIFS respects the sequential time step splitting algorithm that governs the physics package[2] of the ECMWF model [10]. To reconcile this numerical scheme with SP and maintain a parallel execution of the LES instances, we had to split the OpenIFS time step into a part before the cloud scheme and after. After exposing the fractional OpenIFS time steps and DALES time steps as Python functions in OMUSE, as well as the appropriate getters and setters of prognostic fields and tendencies, we were able to implement the algorithm 1 as a Python script that drives the GCM model and LES instances over MPI. The parallel loop on the third line of the algorithm was achieved by using the asynchronous execution capabilities for remote functions in OMUSE.

Algorithm 1: Superparametrization time step

```
1 while $t < t_{final}$ do
2     evolve global model with dynamics and pre-cloud
        physics: $Q \rightarrow Q + D(t)\Delta t$;
3     for $i$ in $1, \ldots, n$ do in parallel
4         compute large scale forcings $f_q(t)$ using eq. 1;
5         time step local models until $t + \Delta t$: $q \rightarrow q(t + \Delta t)$;
6         compute local physics tendency $P_Q(t + \Delta t)$ using eq.
            2;
7     apply SP tendencies: $Q \rightarrow Q + P_Q \Delta t$;
8     evolve global model with post-cloud physics:
        $Q \rightarrow Q(t + \Delta t)$;
9     $t \rightarrow t + \Delta t$;
```

Benchmark Cases

We have run the coupled model in multiple configurations and presented a scientific validation of the outcome against observations in [3]. For the work below, four benchmark cases were used, listed in table 1. Two of them were run around the village of Cabauw (Netherlands) and two of them above the sub-tropical island of Barbados, where trade winds and shallow convection yield large-scale cloud systems that are notably hard to simulate correctly[3]. The Cabauw case represents a typical spring day with land-surface fluxes driving the development of the atmospheric boundary layer, but note that these cases contain LES instances over both land and sea, whereas the Barbados setup contains practically only DALES instances over the ocean. The horizontal grid cell size for DALES is in all cases 200 m, which is about the minimal resolution within the 'inertial range' and the vertical discretization contains 160 equidistant levels up to 4 km height.

In all of these cases, the OpenIFS grid has 511 spectral modes and 91 vertical layers and a reduced gaussian grid with 348528 points and a corresponding horizontal grid spacing of about 40 km. The SP time step has been chosen to be 15 minutes, corresponding to the maximal recommended timestep of OpenIFS at this resolution.

SP load imbalance

The concurrent execution of LES instances appears to be quite unbalanced in our benchmark cases. Figure 1a depicts the waterfall plot of a few time steps of the Cabauw-200 run that displays a particularly unbalanced situation. The large spread in the LES run time distribution, which has been drawn in fig. 1b, can primarily be attributed to the adaptive time step size: DALES has an explicit time integration scheme which limits the time step by the CFL condition. This results in instances with strong up- and downdrafts

[1]Provided the LES MPI layouts do not cross nodes.

[2]In this context we define 'physics' as all processes that are parametrized purely in the vertical direction or within a cell, such as convection, boundary layer turbulence, cloud physics, the surface scheme, orographic drag etc.

[3]These locations also host to measurement campaigns, which is useful for assessing the simulation outcome, but of no specific relevance here.

case	N_{xy}	les	steps	np
Cabauw-64	64^2	72	69	4
Cabauw-200	200^2	42	193	8
Barbados-64	64^2	208	26	4
Barbados-200	200^2	180	26	8

Table 1: Listing of the benchmark cases. N_{xy} denotes the horizontal number of cells, 'les' denotes the number of DALES instances, 'steps' the number of SP time steps, 'np' the number of MPI tasks per DALES model.

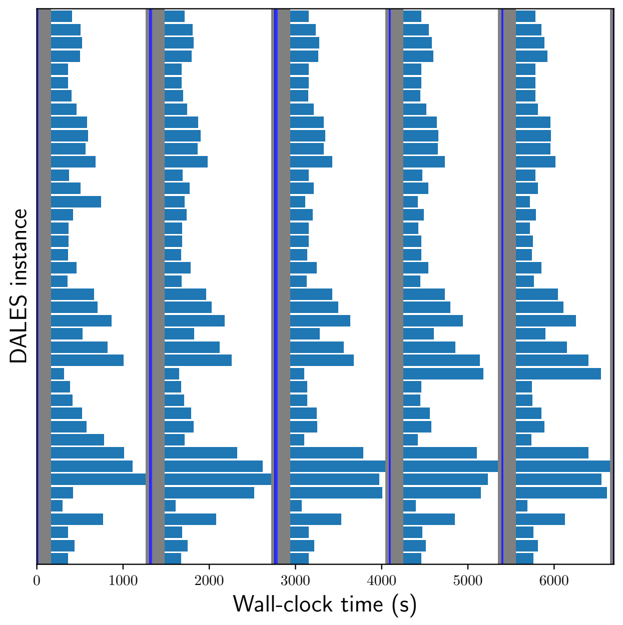
and horizontal fluctuations being heavily limited in their time step. Secondly, the thermodynamics and microphysics have a significant impact on the DALES performance, and hence cloudy or rainy LES simulations generally take more time to complete a single time step. A global SP will require 0.34 million DALES copies and hence ten thousands of cluster nodes, unless multiple instances share a single CPU core, with a degraded DALES performance as a consequence. One can think of several strategies to reduce the number of LES instances, being either confinement to a limited SP region, as we use for these benchmark cases, or coarsening the global model grid, which in its turn degrades the accuracy of the large-scale circulation[4]. In the remaining part of this analysis we will assume that the number of available cores is sufficient to run the required number of instances, and focus on improving the performance of the coupled system.

The performance optimization of the SP setup has been subject of previous research and the following strategies have been identified:

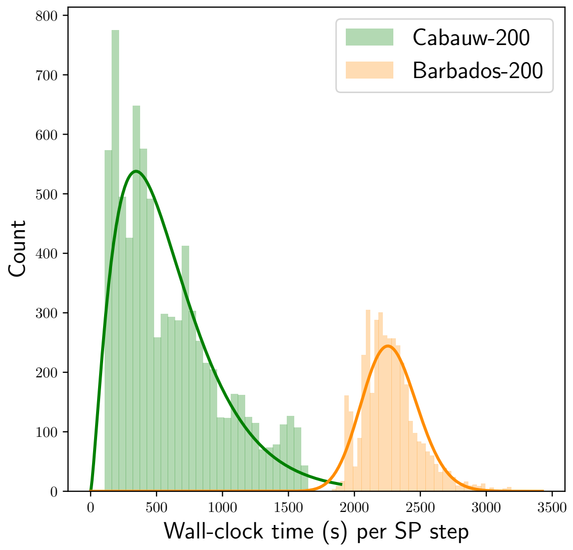
- Increase DALES performance. This is obviously the method of choice, but has limited prospects since DALES is already well-optimized. We have increased single-thread performance by eliminating divisions and optimizing loop orderings.
- Decrease the LES grid extent and reduce the number of cells. There is no reason to let the local models fill up the global model grid cells horizontally, but this strategy is limited by the scale of the emerging cloud patterns that need to be represented by the local models.
- Use mean-state acceleration [11], which allows for shorter run time than the full GCM time step. This method assumes there is a time scale separation between the large eddies and the mean state, allowing the resulting SP tendency to be linearly extrapolated from the mean state after only a fraction of the time steps. The technique slightly deteriorates the accuracy of the SP in favour of shorter runtimes for the DALES models [3].
- Use load-balancing to fill up the white space in fig. 1a. Since it cannot be known initially which LES models will be slow or fast[5], a dynamical load-balancing strategy is needed, where every SP time step we monitor the wall-clock times of the instances and rebalance the amount of CPU cores (and MPI tasks) they will be given. This assumes that the wall-clock

[4]And moreover, if one insists that all the LES instances cover global model grid cells and retain a resolution of about 200 m, this strategy will effectively increase the number of grid points per LES, making them run slower again.

[5]And this behavior may change during the simulation, e.g. as convection is developing the instance time stepping may decelerate.



(a) Waterfall plot of four SP time steps. The blue bars denote the actual wall-clock time of the LES instances.



(b) Overall distribution of DALES wall-clock times of the SP time stepping with a fitted Gamma distribution.

Figure 1: DALES ensemble timing data.

time T_n of a LES instance is good predictor for T_{n+1}. Fortunately this is usually the case, as can be observed from fig. 2a and 2b.

An important technical aspect of the SP algorithm is the requirement that the LES instances start from their previous state: whereas the forcings are discontinuous between SP time steps due to the GCM update, the developed turbulent eddies encoded in the small-scale LES state must be inherited. In the current framework, the DALES instances keep their state in memory while the GCM progresses; assigning more resources to a slow DALES instance however requires a restart of the program from a serialized state. Although this is a feature that the OMUSE framework does support, DALES currently must always restart with the same number of MPI processes as when the snapshot was made. Nevertheless we believe overcoming this technical difficulty is very much feasible, though not the purpose of this paper: we will suffice with a performance model of DALES to study the overall benefits of a quasi-dynamical[6] load-balancing strategy.

Coupler performance

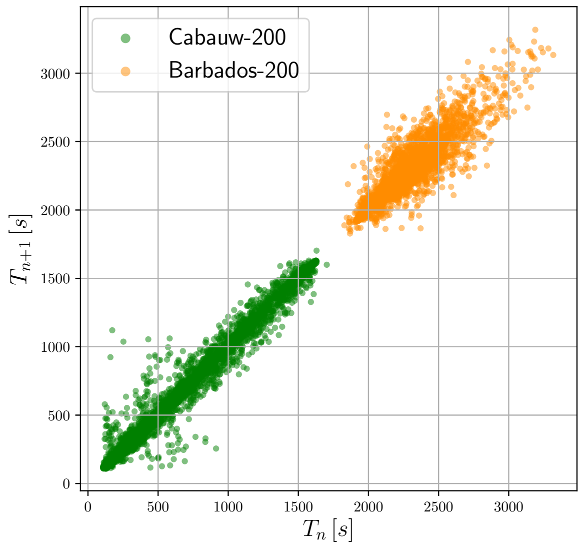
Our efforts to optimize the SP coupled system have not been limited to the DALES code; we have obtained better performance and scalability of the coupling code as well. This was necessary as the coupling code and communication between Python script and LES instances had become a major bottleneck after applying the above optimizations. In our original implementation of the SP algorithm, the computation and dispatching of the large-scale forcings and retrieval of SP tendencies (line 4 and 6 in algorithm 1) had been part of the serial Python code. As a result, many MPI messages were being sent in a blocking way before and after the SP time stepping, resulting in sub-optimal performance and poor scaling behavior of the data exchanges.

In the new version of our SP implementation we have used the asynchronous feature of OMUSE to parallelize the communication of these forcings and tendencies. The former are being sent to all LES instances in parallel whereas the latter are being retrieved whenever the LES has finished time stepping, an attempt to hide MPI latencies and prevent network congestion. As a result, the coupling time has been reduced as can be seen from fig. 3. However, increasing the number of LES models to 1000 reveals that there is still a bottleneck in the implementation: setting the forcings may result in many MPI send calls being issued from the node executing the master script and we suspect that a collective MPI routine[7] would be a far better option here. We will ignore this scaling problem in our considerations on load-balancing below and assume that the communications with the LES instances is a constant overhead on the time-to-solution.

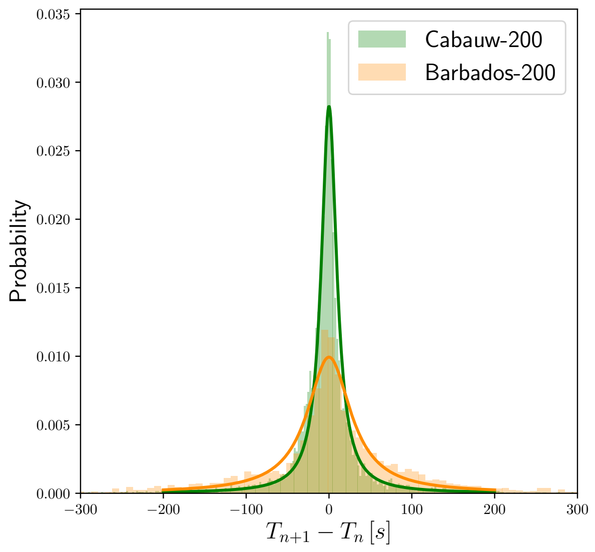
DALES strong scaling

The DALES code is parallellized with MPI, with partitioning in the horizontal directions of the grid[8]. The DALES time step contains only a few inter-process communication points, which are

[6]We rebalance the work at SP time steps.
[7]Which is to our knowledge not (yet) part of the OMUSE framework.
[8]This is a common partitioning in atmospheric models because the vertical dimension is treated vastly differently throughout the code.

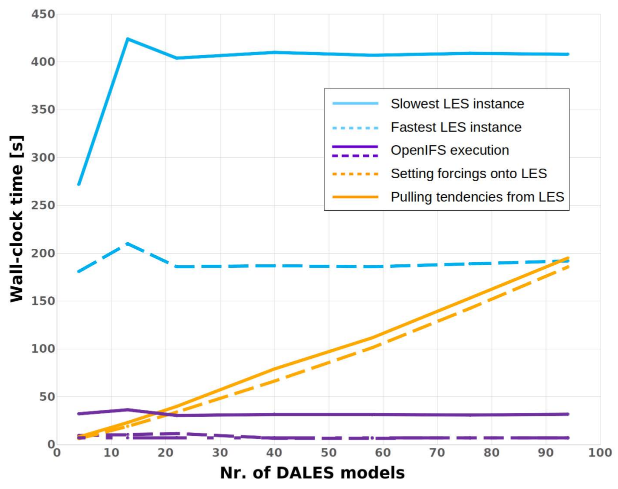


(a) Autocorrelation plot of the DALES wall-clock time T_{n+1} at SP time step $n + 1$ vs. the previous time step.

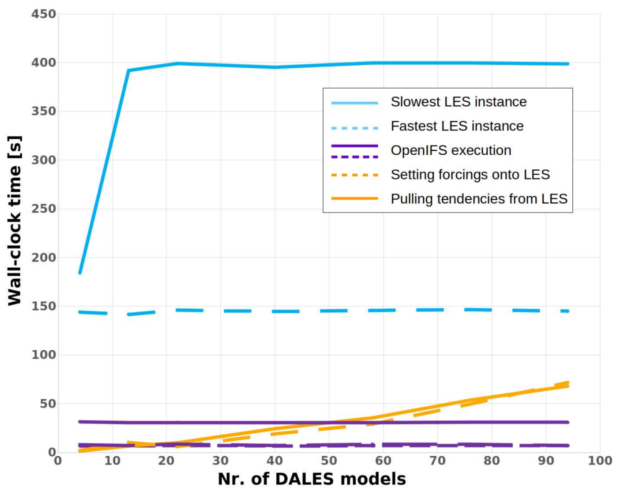


(b) Distribution of the difference of consecutive wall-clock times of SP time steps per LES instance, with a fitted Cauchy distribution.

Figure 2: DALES SP time step autocorrelation.



(a) Original sequential SP coupling implementation timings.



(b) Parallelized coupling optimization result.

Figure 3: Scaling of various algorithm components with the number of superparametrized gridpoints, using the average over 5 time steps and LES over (a growing region over) Barbados, with a horizontal resolution of 64×64 grid cells.

the halo exchanges of its prognostic fields during horizontal advection/diffusion and collective communication for the fast Fourier transform in the pressure equation solver. For our purpose, we are only interested in DALES configurations with up to 200×200 cells horizontally, and it is sufficient to consider the strong scaling of the program within a single node of a compute cluster. The inter-node communication overhead makes larger partitions at these resolutions inefficient. In fig. 4 we fitted the DALES runtime on a single node of the ECMWF Cray XC40[9] with a simple Amdahl scaling

[9]This machine has two 18-core Intel Xeon EP E5-2695 V4 Broadwell processors per compute node.

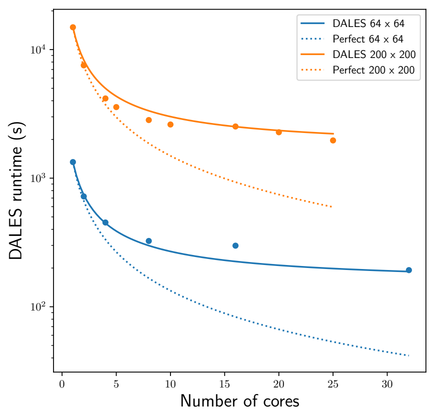


Figure 4: Strong scaling behavior of DALES for two resolutions (64^2 and 200^2 grid points horizontally) on a single node. Samples were obtained with a 5-run average using integer divisors of the grid resolution in each dimension.

[12],

$$T(N) \propto (1 - p) + p/N \tag{3}$$

where we fitted $p = 0.89$ to be the parallel fraction of the code and the proportionality factor is the single-thread wall-clock time $T(1)$ which depends upon the DALES grid resolution. We assume this constant is universal across our simulations. Its potential limiting factors are halo exchanges, collectives during the solution of the pressure equation and local load imbalances in thermodynamics and microphysics. The former two are dependent upon DALES configuration (which is identical for all our instances), the latter is negligible due to periodic boundary conditions.

Exploring Dynamic Load Balancing

The aim of this section is to explore possible benefits of dynamic load balancing for the SP system by optimizing the number of CPU cores used by each of the DALES instances. We will assume that all other technical bottlenecks have been cleared and the SP time stepping of the slowest LES instance dominates the time-to-solution. Furthermore, we will make the following assumptions:

(1) It is always possible to find a configuration of DALES MPI tasks such that no instance has to run across multiple nodes. We constrain the simulated load-balancing to disallow more than 36 cores for a single DALES instance –the number of cores per node at the machine where we have collected our timing data– but ignore the problem of placing the processes efficiently.
(2) Load-balancing and process placement algorithms take a negligible amount of time compared to the SP time stepping. In reality, a restart of the entire system does introduce some overhead.
(3) Assigning more MPI tasks for certain DALES instances does not significantly impact the communication time with the

master Python script. This is a safe assumption considering every DALES will be localized within a single node.

(4) The performance of a DALES instance is independent of the DALES models it is sharing a compute node with, even if the number of processed grid cells becomes inhomogeneously distributed amongst MPI tasks. This assumption ignores the fact that a memory-bound code like DALES is heavily dependent upon L3 cache usage and thus will feel the presence of other instances on the CPU, but it does not make a lot of sense to model this penalty whilst the process placement problem has not been solved.

To explore the impact of load-balancing we apply the scaling rule eq. 3 to the four benchmark cases presented above to optimize the SP wall-clock time, which we assume to be the slowest LES. The algorithm consists of an iterative hypothetical reallocation of resources, reassigning a core from the currently fastest LES to the slowest one until the predicted run time decreases no more. This yields a configuration that can be applied to the measured wall-clock time of the next time step. We denote the result with the *persistence assumption*, as it is based upon the hypothesis that the performance will be a good predictor for the next time step. The *perfect load-balancing* is the theoretical maximum, constructed by using the actual measured timings of the next time step as input. The results are listed in tab. 2 and reveal a picture where load-balancing can play a significant role depending on the case at hand. For the Cabauw cases, the tail of the distribution is long and the bulk of the models have short run times (see fig. 1a). A speedup of about 18% seems feasible, and this is rather close to the perfect load-balancing because the LES run-times are well predictable, which can be seen from the time series in fig. 5 as well. In the Barbados cases however, the instances are much more lumped together and load-balancing speedup is at best limited to a disappointing 7%. We believe this is due to the fact that the models are all located above sea, with similar surface fluxes, and driven by a rather uniform trade wind. This will result in a homogeneous CFL condition across the DALES models and less potential for load-balancing to accelerate extremely slow instances.

case	s_{pers} [%]	s_{perf} [%]
Cabauw-64	17.94	25.51
Cabauw-200	11.01	15.49
Barbados-64	1.30	4.94
Barbados-200	2.60	6.44

Table 2: Hypothetical load balancing of 4 SP simulations: s_{pers} denotes the speedup achieved by assuming persistent wall-clock time of LES instances, s_{perf} denotes the maximal achievable speedup with perfect prediction.

To draw conclusions beyond the test cases at hand, we can treat the LES run times as an ensemble of independent Markov chains that evolve according to the Cauchy distribution ρ, centered at zero, with scale parameter τ fitted in fig. 2b,

$$T_{n+1} = T_n + \Delta\,, \quad \rho(\Delta) = \frac{1}{\pi\tau}\left[\frac{\tau^2}{\tau^2+\Delta^2}\right] \tag{4}$$

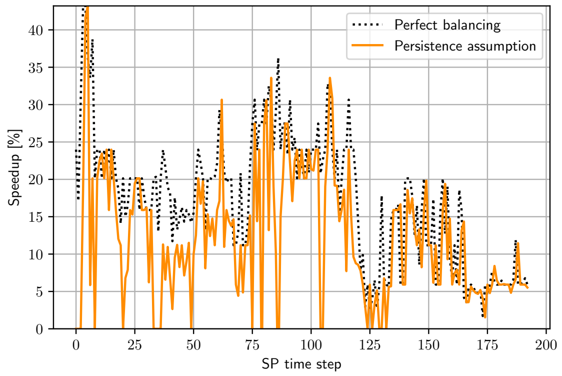


Figure 5: Time series of the expected speedup for the Cabauw-64 case, listing both the maximal achievable speedup (perfect prediction) and the gain for the persistence assumption.

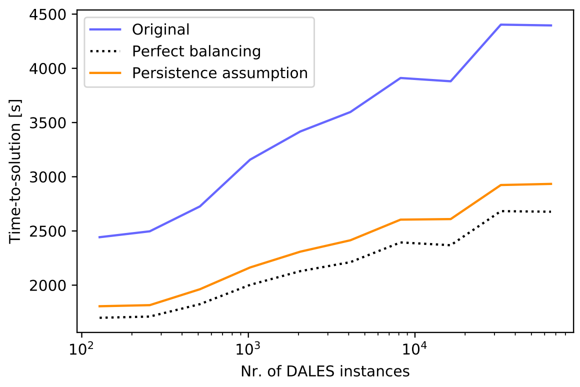
The assumption that the LES wall-clock time depends only on its previous state is not exactly valid because the LES models are forced by a transient large-scale circulation that eventually influences their step size and thermodynamic complexity. Furthermore we note that the scale τ of the distribution in eq. 4 is not universal and depends on the test case at hand: the Barbados cases display a slightly wider distribution than the Cabauw runs, especially comparing against the variance of the distribution of LES run-times themselves. Finally, the distribution is in reality skewed in extreme cases, very slow LES instances tend to accelerate and very fast ones to slow down due to limited wind forcings and surface fluxes. Nevertheless, we believe that the independent ensemble is a sufficient approximation to assess the impact of load-balancing for large number of LES instances. For this it is essential to mimic the long tail of the distributions of SP wall-clock times well, such that the slowest LES has a realistic run time. The PDF of DALES run times has been fitted with a gamma distribution (see fig. 1b),

$$\rho(T) = \frac{x^{k-1}e^{-x}}{\theta\Gamma(k)}\,, \quad x = \frac{T}{\theta} \tag{5}$$

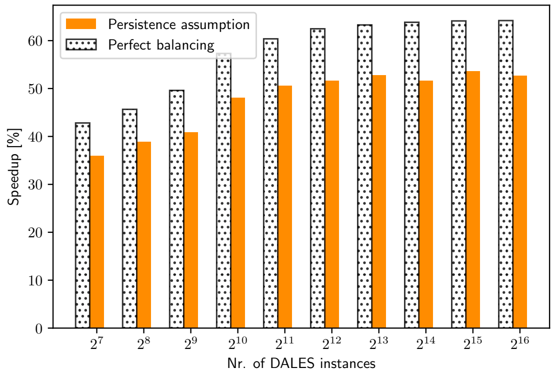
where θ is a typical time scale that determines the standard deviation together with the dimensionless shape parameter k. With the fitted parameters listed in tab. 3 we are able to sample an initial set of run times for any number of DALES models and progress this set using eq. 4. During this we reject sampled SP times beyond 10σ of the original fitted distribution to avoid artificially high run

case	τ [s]	k	θ [s]
Cabauw-64	16.91	2.12	309.90
Cabauw-200	10.99	2.29	256.83
Barbados-64	3.87	100.63	3.24
Barbados-200	35.26	116.78	19.42

Table 3: Fitted parameters from eqs. 4 and 5 for two test cases at two resolutions.



(a) Total wall-clock time per step.



(b) Speedup due to load-balancing.

Figure 6: Synthetic Cabauw-64 based performance with increasing number of LES instances.

times and resulting load-balancing speedup, and also reject jumps in runtime larger than 10τ, which are considered unrealistic. As a starting distribution, we choose the Cabauw-64 fitted PDF because we expect this density to represent the global SP ensemble better than the Barbados case; in a global cover with tens of thousands of LES instances, the large variability among the DALES models will result in a long tail of slow instances and a large reservoir of fast models. The resulting timings are shown in fig. 6a. The original timings increase as more LES models are considered because extremely slow instances are more likely to occur. The effect of load-balancing significantly reduces this growth and its relative impact rises too, with an asymptotic speedup around 50%, which is only about 10% lower than the maximal achievable acceleration, shown in fig. 6b. We note, however, that the increasing run time disqualifies the Cabauw-64 global extension as a viable forecasting system as it takes more than an hour to progress the model over a GCM time step of 15 minutes. Nevertheless, we expect the reported speedup to persist under optimizations of the LES code – at least as long as the time step size in the explicit integration is limited by the CFL number and the strong scaling is not degraded.

Finally we point out that the above results depend upon the parallel efficiency p of the DALES code of eq. 3; increasing the parameter to e.g. 0.99 raises the speedup listed in tab. 2 to resp. 20% and 18% for the Cabauw-64 and Cabauw-200 cases; for the Barbados cases however, the parameter p does not impact the speedup significantly. Hence we expect that also after quasi-dynamical load-balancing has been implemented, it remains beneficial to invest in the scaling optimization of DALES, at least for cases with sufficient spread in the run-times.

Conclusion and Outlook

We have given a detailed description of the various factors that determine the performance of a SP of OpenIFS by a large number of independent LES instances. The scaling of the application has been improved by optimizing single-threaded performance of DALES and increasing parallelism in the coupling routines. We pointed out the load imbalance which occurred especially in our 'Cabauw' runs over the Netherlands, and which we believe to exist in a global SP too. A thought experiment based upon the strong scaling of DALES and the observed distribution and autocorrelation has revealed a significant impact of dynamically balancing the load of each processor. In practice this can be achieved by restarting the system with a balanced number of MPI tasks for the LES instances. However, a more important task in our current implementation is the scaling of the routine that sends the forcings to all DALES instances. Because this happens at a synchronization point in our code, MPI latencies cannot be hidden and this task quickly becomes a bottleneck for large number of LES instances. We believe a collective MPI call could resolve this issue quite easily, which will be part of a future effort. Another option could be to redesign the system to not have a master script at all, and apply the forcings from the OpenIFS MPI tasks in a non-blocking fashion. In such an implementation all unit conversions would have to be carried out in the Fortran library interfaces of the two components.

Even with all of the above optimizations in place, a global SP of the 40 km resolution OpenIFS will need a tremendous amount of resources to achieve a time-to-solution that is shorter than the simulated time range. The performance advantage of SP (over say a global LES) is the fact that neighboring instances do not communicate with each other every time step. The other burden however, the limited LES time step, cannot entirely be mitigated by SP because the global model acts as a synchronization point for all small-scale models and hence the system is bound by the LES with the largest average time-to-solution over the SP time step. In this sense, SP by variable time-stepping LES models remotely resembles local time stepping in the GCM [13] and the predictive load-balancing acts as a work scheduler.

We note that quasi-dynamical load-balancing could be used to optimize a setup with more LES instances than available CPU cores as well. This however requires a task queue approach to the parallel loop in the algorithm 1 and job scheduling provided within the Python coupling code, because a naive overloading of the available resources in OMUSE will lead to multiple MPI tasks sharing a single core, degrading the DALES performance significantly. A possible strategy could consist of a generic non-preemptive multiprocess scheduler dynamically distributing the LES instances, where the requested number of cores and estimated execution time –determined by the load-balancer– are input parameters to the scheduler. Such a worker queue method requires a restart for every LES at the beginning of their SP time loop, possibly incurring I/O and process creation penalties. If this overhead can be kept sufficiently small, we are convinced this approach may well lead to superior scaling (to the above estimation) for all core counts. Another approach is to consider the multiprocess scheduling as part of the cost function of the static optimization problem of the load balancing, and minimize the predicted *makespan* of every SP time step under the constraint that the processes of a single LES model have to be confined to a single compute node. Plenty of policies have been proposed that handle the NP-hard problem of finding the most efficient schedule [14]; the oversubscribed, load-balanced SP presents an interesting application of these algorithms and may be the subject of future research.

Finally, we point out that a multiscale model with many small-scale models will generally have synchronization points between them. Whenever these sub-models display sufficient variability in their time-to-solution, a relatively good strong scaling and sufficient autocorrelated run-times, the SP pattern applies and quasi-dynamic load-balancing on a predictive basis can improve the performance of the entire system. In such cases the Monte Carlo generation of synthetic wall-clock times described above may provide an indication of the expected speedup, and give an estimation of the hardware resources required for running experiments.

Acknowledgments

This work was supported by the Netherlands eScience Center (NLeSC) under grant no. 027.015.G03. Furthermore, we acknowledge the use of ECMWF's computing and archive facilities in the research reported here. Also SURFsara provided computing resources. Plots in this paper were created with matplotlib [15].

REFERENCES

[1] W. W. Grabowski, "Coupling cloud processes with the large-scale dynamics using the cloud-resolving convection parameterization (CRCP)," *Journal of the Atmospheric Sciences*, vol. 58, no. 9, pp. 978–997, 2001.

[2] G. Carver *et al.*, "The ECMWF OpenIFS numerical weather prediction model release cycle 40r1: description and use cases," *in preparation to be submitted to GMDD*, 2019.

[3] F. Jansson, G. van den Oord, I. Pelupessy, J. H. Grönqvist, A. P. Siebesma, and D. Crommelin, "Regional superparameterization in a global circulation model using large eddy simulations," *Journal of Advances in Modeling Earth Systems*, vol. 11, no. 9, pp. 2958–2979, 2019.

[4] T. Heus, C. C. van Heerwaarden, H. J. J. Jonker, A. Pier Siebesma, S. Axelsen, K. van den Dries, O. Geoffroy, A. F. Moene, D. Pino, S. R. de Roode, and J. Vilà-Guerau de Arellano, "Formulation of the Dutch Atmospheric Large-Eddy Simulation (DALES) and overview of its applications," *Geoscientific Model Development*, vol. 3, no. 2, pp. 415–444, 2010.

[5] A. Arakawa, "The cumulus parameterization problem: Past, present, and future," *Journal of Climate*, vol. 17, no. 13, pp. 2493–2525, 2004.

[6] S. Bony and J.-L. Dufresne, "Marine boundary layer clouds at the heart of tropical cloud feedback uncertainties in climate models," *Geophysical Research Letters*, vol. 32, no. 20, 2005.

[7] I. Pelupessy, B. van Werkhoven, A. van Elteren, J. Viebahn, A. Candy, S. Portegies Zwart, and H. Dijkstra, "The oceanographic multipurpose software environment (OMUSE v1.0)," *Geoscientific Model Development*, vol. 10, no. 8, pp. 3167–3187, 2017.

[8] Pelupessy, F. I., van Elteren, A., de Vries, N., McMillan, S. L. W., Drost, N., and Portegies Zwart, S. F., "The astrophysical multipurpose software environment," *A&A*, vol. 557, p. A84, 2013.

[9] G. van den Oord, F. Jansson, I. Pelupessy, M. Chertova, J. H. Grönqvist, A. P. Siebesma, and D. Crommelin, "A python interface to the dutch atmospheric large-eddy simulation." submitted to SoftwareX, 2019.

[10] A. Beljaars, G. Balsamo, P. Bechtold, A. Bozzo, R. Forbes, R. J. Hogan, M. Köhler, J.-J. Morcrette, A. M. Tompkins, P. Viterbo, and N. Wedi, "The numerics of physical parametrization in the ecmwf model," *Frontiers in Earth Science*, vol. 6, p. 137, 2018.

[11] C. R. Jones, C. S. Bretherton, and M. S. Pritchard, "Mean-state acceleration of cloud-resolving models and large eddy simulations," *Journal of Advances in Modeling Earth Systems*, vol. 7, no. 4, pp. 1643–1660, 2015.

[12] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), (New York, NY, USA), pp. 483–485, ACM, 1967.

[13] M. Baldauf, "Local time stepping for a mass-consistent and time-split advection scheme," *Quarterly Journal of the Royal Meteorological Society*, vol. 145, no. 718, pp. 337–346, 2019.

[14] Kasahara and Narita, "Practical multiprocessor scheduling algorithms for efficient parallel processing," *IEEE Transactions on Computers*, vol. C-33, pp. 1023–1029, Nov 1984.

[15] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.