Research Article

Prediction and reduction of runtime in non-intrusive forward UQ simulations

Florian Künzner[1] · Tobias Neckel[1] · Hans-Joachim Bungartz[1]

Abstract
To foster predictive simulations, a variety of methods have recently been developed to efficiently tackle uncertainty quantification (UQ) in complex, computational intensive problems. Many of these methods are non-intrusive and, thus, result in a (large) number of embarrassingly parallel black-box evaluations of the underlying simulation codes. While the focus of development is typically on the number of black-box evaluations, which represents the bulk of the computational workload, an additional level of potential performance gains exists. In many scenarios, uncertain input leads not only to uncertain outputs, but also to a varying and thus stochastic runtime of the simulation codes. For scheduling the individual black-box runs, this information is typically not taken into account, resulting in non-negligible idling times on parallel systems. In this contribution, we compare a variety of different scheduling strategies for non-intrusive UQ scenarios using the non-intrusive polynomial chaos approach. In particular, we propose to construct a surrogate model for the runtime of the application using the identical UQ methodology as for the original problem. Using this model to predict the runtimes for subsequent black-box runs allows for (heuristical) optimization of the scheduling. The method has been tested for the forward quantification of uncertainty on academic models and on a pedestrian simulation in the context of evacuation scenarios. This approach allows speed-up factors of about two for the total runtime and can be generalised to a large variety of applications that incorporate parameter-dependent runtime.

Keywords Uncertainty quantification · High-performance computing systems · Scheduling · Runtime prediction · Runtime reduction

1 Introduction

Uncertainty quantification (UQ) has become an established technique in computational science and engineering. The goal is to understand the general behaviour of the investigated model under uncertainties. There exist different types of uncertainties, a prominent example are uncertainties contained in the input parameters of a computer model.[1] For a general introduction to the UQ methods, see [1–3].

In this work, the focus is on frequently used non-intrusive forward UQ methods which consist of three phases (see Fig. 1): in the assimilation phase, the parameters are investigated and suitable probability distributions are used for the uncertain parameters. With this information, the computer model is called multiple times with different values, which is known as the propagation phase. In the certification phase, the output of interest (OoI, mostly physical output values of a model) are statistically evaluated. Usually, the statistical moments such as mean and

[1] A computer model in this paper is defined as an implementation of a mathematical/numerical model in order to simulate certain phenomena.

✉ Florian Künzner, florian.kuenzner@tum.de | [1]Department of Informatics – Chair of Scientifc Computing, Technical University of Munich, Boltzmannstrasse 3, 85748 Garching, Germany.

SN Applied Sciences (2019) 1:1038 | https://doi.org/10.1007/s42452-019-1066-3

Received: 13 June 2019 / Accepted: 8 August 2019 / Published online: 16 August 2019

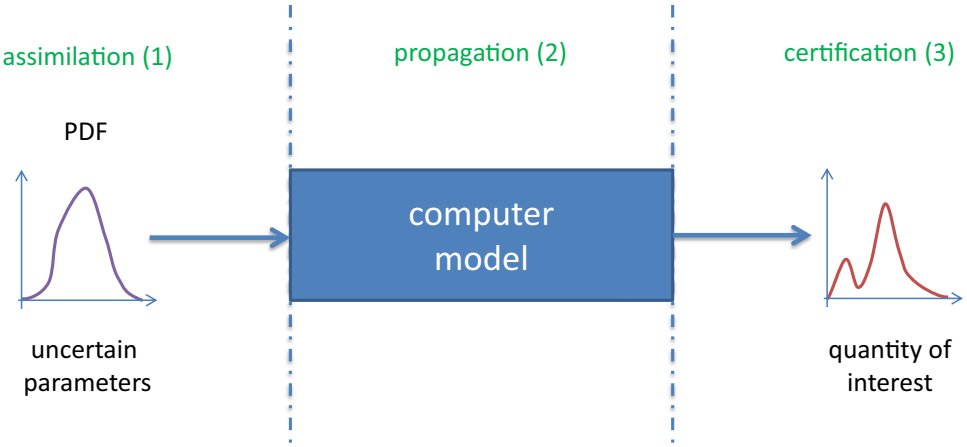

Fig. 1 Illustration of the forward UQ method with its three phases: assimilation, propagation, and certification

variance are calculated for every output of interest and represent the corresponding quantity of interest (QoI).

In non-intrusive UQ simulations, the computer model that simulates the behaviour of the process under investigation is used as a black-box, i.e. the computer model and its equations are not modified for the UQ analysis. The UQ methods typically require many samples (black-box computer model runs) to compute the statistical moments with a certain accuracy.

Due to the black-box approach, the runs are independent and, thus, can be parallelised "embarrassingly" [4–6]. In cases where the runtime of the computer model is sensitive to the actual values of the uncertain parameters, a straightforward mapping of simulation runs to processors or computing nodes typically results in considerable performance losses due to idling times. Such situations arise e.g. when uncertain parameters affect the initial condition, boundary condition, time step sizes, or stopping criteria of the simulation.

We observed exactly this variation of a computer model's runtime in combination with an evacuation scenario of pedestrian dynamics [7, 8] using VADERE [9]. In the context of UQ simulations, a similar variation of the runtime could be observed in [10, 11]. The corresponding research questions for this paper are: (1) Is it possible to predict the runtime of such black-box computer models where its parameters influence the runtime? (2) Is it possible to reduce the idling and, thus, speed-up the execution of the whole UQ analysis?

Several options exist to speed-up a UQ analysis. One option is to reduce the runtime of the computer model itself by using other mathematical equations, smarter algorithms to solve the problem, or other techniques to reduce the computational cost for the spatial or time discretisation. Another option is the use of performance engineering and parallelisation techniques such as vectorisation, multithreading, or distributed computing on a high-performance computing (HPC) system. In the context of UQ, additional options arise to address the outer loop optimisation, because many black-box computer model runs are required. There exist already various techniques such as surrogate models [12–14], multifidelity models [15], model order reduction [16], sparse grid interpolation or cubature [17–19]. But all these techniques at the end require numerous black-box computer model runs where idling can occur.

HPC systems usually provide a job scheduling system such as SLURM [20] to distribute work dynamically to computing units. These job scripts however, are not very interchangeable to other systems, and the whole UQ simulation process is more complicated, because the program is than split into several jobs which makes collecting the outputs at the end more complicated. Users usually want to have an all-in-one solution from the UQ assimilation to propagation and certification.

In this contribution, we benchmark three different parallelisation and scheduling techniques to reduce the idling time. Our main contribution is to use the runtime of a computer model as a synthetic output of interest and create a surrogate model for the runtime via UQ methodology. Hence, in subsequent UQ simulations, we are able to predict the runtime of each single black-box run and to enhance the three standard techniques with this knowledge and dynamic scheduling approaches to make UQ simulations more efficient.

To our knowledge, such an approach has not yet been explored in detail: Only in [21, 22] a synthetic output of interest from the computer model is used to predict the number of internal iterations one run takes. With this information, the authors group the runs with similar iterations together to an so called ensemble group. The runs in the ensemble group are then executed in parallel with C++ template techniques and parallel SIMD instructions to reduce the over-all runtime. But this requires to modify the source code of the computer model. We focus here on non-intrusive methods where we use the computer model

only as black-box. We test and analyse the runtime prediction in combination with different scheduling strategies on academic test models as well with a real-world evacuation scenario of pedestrians dynamics similar to [8].

The remainder of the paper is organised as follows. Section 2 contains theoretical background and modelling details. In Sect. 3, the idling problem is analysed for the three standard scheduling techniques. Section 4 describes the approach of measuring and predicting the runtime of black-box computer model runs. Section 5 shows how to use runtime prediction to reduce the idling by changing the propagation order. Numerical results for an academic example as well for a real-world evacuation scenario in pedestrian dynamics are presented in Sect. 6, before concluding the work in Sect. 7.

2 Theoretical background and modelling details

This section describes some theoretical aspects of the used UQ method, the specific pedestrian simulation computer model, and computational timing measurements that are used to compare different scheduling methods.

2.1 Non-intrusive polynomial chaos

In recent years, a broad spectrum of UQ methods has been developed (see [1, 2] for a general overview). Each method has specific properties and provides different advantages and disadvantages. Since the models considered in this contribution have only a moderate number of uncertain parameters and are considered as black-box, we rely on the non-intrusive polynomial chaos approach (also known as non-intrusive spectral projection approach (NISP, see [3]) or pseudo-spectral approach [2]) which provides a reasonably good accuracy at moderate computational costs.

The model $f(x,t,\boldsymbol{\zeta})$ that is investigated typically depends on space x, time t, and M independent random variables $\boldsymbol{\zeta} = (\zeta_1, \dots, \zeta_M)$ described by a probability distribution. For the non-intrusive polynomial chaos approach, the general polynomial chaos expansion (gPCE) of the solution $U(x,t,\boldsymbol{\zeta})$ reads

$$U(x,t,\boldsymbol{\zeta}) = \sum_{j=0}^{\infty} \underbrace{c_j(x,t)}_{\substack{\text{spatio-}\\ \text{temporal}}} \cdot \underbrace{\boldsymbol{\Phi}_j(\boldsymbol{\zeta})}_{\text{random}}. \tag{1}$$

It separates the spatio-temporal coefficients $c_j(x,t)$ from the random $\boldsymbol{\Phi}_j(\boldsymbol{\zeta})$ base functions. $\boldsymbol{\Phi}_j(\boldsymbol{\zeta})$ are orthogonal polynomials that fit to the probability distribution (see [23]) of the uncertain input parameters $\boldsymbol{\zeta}$. To numerically evaluate (1), it is truncated after $N+1$terms resulting in

$$U(x,t,\boldsymbol{\zeta}) \approx u_N(x,t,\boldsymbol{\zeta}) := \sum_{j=0}^{N} c_j(x,t) \cdot \boldsymbol{\Phi}_j(\boldsymbol{\zeta}). \tag{2}$$

The non-intrusive polynomial chaos approach computes the coefficients $c_j(x,t)$ by an integral

$$c_j(x,t) := \frac{1}{\gamma_j} \int f(x,t,\zeta)\boldsymbol{\Phi}_j(\zeta)W(\zeta)d\zeta \tag{3}$$

which is approximated via a cubature rule. Typically, Gaussian cubature of the form

$$c_j(x,t) \approx \frac{1}{\hat{\gamma}_j} \sum_{i=1}^{Q} f(x,t,z_i)\boldsymbol{\Phi}_j(z_i)w_i, \tag{4}$$

is used due to its accuracy. The z_i are cubature points, and w_i represents the corresponding weights. Q denotes the number of cubature points z_i, where the computer model $f(x,t,z_i)$ is evaluated. Because the Gaussian cubature is the tensor product of all number of cubature points q for each parameter, Q is computed as $Q = q^M$. To make the $\boldsymbol{\Phi}_j(z_i)$ orthonormal, the coefficients are divided by the normalisation constant $\hat{\gamma}_j$ (see [2, p. 84] for details).

The statistical moments such as the mean μ

$$\mu \approx \mu(u_N) = c_0 \tag{5}$$

and the variance σ^2

$$\sigma^2 \approx \sigma^2(u_N) = \sum_{j=1}^{N} c_j^2 \tag{6}$$

can directly be approximated via the coefficients c_j [24]. These statistical moments are in the focus of interest for UQ analysis to support decision making. We compute also these moments but we additionally use the full representation of the general polynomial chaos expansion of Eq. (2) as a surrogate to estimate a specific output of interest (the runtime, see Sect. 4).

For the implementation of the uncertainty analysis, we rely on the software framework Chaospy [25, 26]. It is easy to use, allows fast prototyping, is open for changes and contributions, and has excellent support for developers. Chaospy provides a lot of functionality for the technical part in the assimilation phase (compare Fig. 1): many different probability distributions that can easily be configured and joined to a multivariate distribution. Furthermore, the certification phase is supported to a large extend with the calculation of the statistical moments, sensitivity analysis and the generation of the

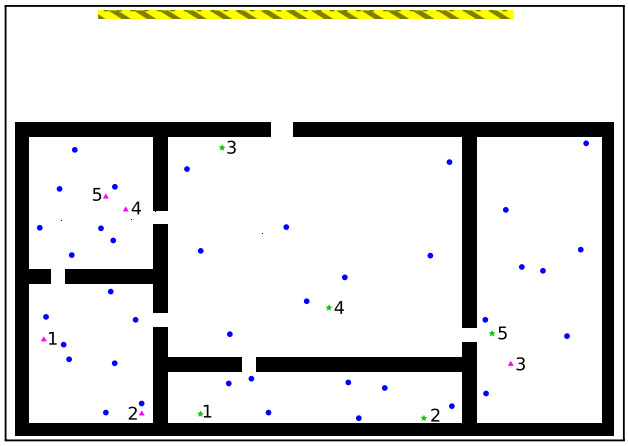

Fig. 2 Illustration of the evacuation scenario with separated families. The yellow striped area outside of the building is the safe zone. Adults without children are represented by blue nodes, adults with children by green stars, and children as pink triangles. For this visualisation, the number of agents has been reduced compared to the actual scenario in Sect. 6.3

complete gPCE u_N (see Eq. (2)). The propagation phase, however, is very much up to the developer: In this paper we address this phase and support the execution on HPC systems with different scheduling strategies.

2.2 Pedestrian simulation: evacuation scenario

The simulation of pedestrians is an active field of research. It is very challenging to validate corresponding results based on individual persons, because people do not always behave predictably and their movement—their speed and path—depends on a considerable amount of unknown and/or uncertain data. We successfully used UQ approaches in [7, 8] to investigate the general behaviour of a group of people, resulting in valuable information on the quantity of interest for the researches in the field.

In this work, we use the computer model VADERE to simulate an evacuation scenario of a building under uncertain conditions in Sect. 6.3. VADERE [9] is an open source framework for pedestrian simulation. It is a microscopic pedestrian dynamics simulator, where each "simulated" person (called agent) is considered individually. The framework works with scenarios: A scenario is a description of the topography (e.g. a building), the parameters of the agents (e.g., the number and the positions of the agents with their walking speed), and the movement and behaviour models (e.g. the optimal steps model (OSM) [27–30]).

The considered scenario analyses the evacuation of a building with the behaviour of separated family members [8]. Figure 2 illustrates the evacuation scenario which consists of a building with one floor (the ground floor). Inside the building, there are adults and young children (pink triangles). Some adults are accompanied by a child (green stars), some not (blue nodes). The challenge is to simulate the behaviour of the pedestrians with separated families when a fire alarm occurs. In such a case all agents want to run out of the building into the save zone. The adults (without children) immediately leave the building. The parents search their children, and the children in this scenario are considered very young and tend to freeze (see [31])—they wait until their parents have found them and then run together out of the building.

The scenario configuration uses the OSM for the basic movement of agents. This model has numerous parameters, which we assume to be deterministic. On top of the OSM, we use a family affiliation model [8] which has three main parameters: (1) $perc_{fam}$ is the percentage of family members, (2) v_{parent} is the speed (in [m/s]) of parents searching children, and (3) v_{child} is the speed (in [m/s]) of the parent-child-pair. In this work, these three parameters are assumed to be uncertain and are further investigated in Sect. 6.3.

The runtime of a VADERE simulation depends strongly on the specified scenario, especially on the number of agents. The evacuation scenarios in this paper use a fixed size of 100 agents. Typical runtimes for a single VADERE simulation in this evacuation scenario vary between 63 and 1062 s depending on the values of the uncertain input parameters. This runtime variation is hard to predict in general and needs to be further investigated to address the research questions (1)–(2) of Sect. 1.

2.3 Runtime definitions for UQ simulations

The following notation denotes runtime measurements by T and runtime estimations by $\mathbb{T}$ when comparing different scheduling strategies. The time for a complete UQ simulation T_{UQsim} is

$$T_{UQsim} = T_{Ass} + T_{Prop} + T_{Cert}, \tag{7}$$

where the runtime for each separate UQ phase is denoted by T_{Ass} for the technical part of the assimilation, T_{Prop} for propagation, and T_{Cert} for the technical part of the certification, respectively (see Fig. 3 for an illustration).

For a fine-grained analysis of the scheduling strategies, it is necessary to understand how the workload is distributed to the computing units. Figure 4 illustrates the concept of work packages: In the propagation phase, the computer model is called for every cubature point z_i, $i = 1, \dots, Q$. The scheduling strategy decides which computing unit works on which set of cubature points. In that context, a set of cubature points is called work package WP_j, $j = 1, \dots, J$, and there is a bidirectional mapping between a work package and an individual computing unit. It is not always guaranteed that the sizes of the work

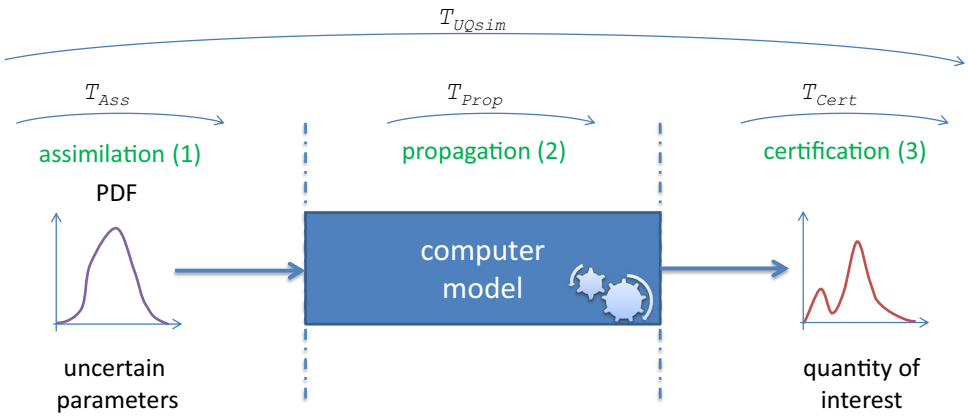

Fig. 3 Visualisation of the UQ phases and the time measurements for simulations on HPC systems. T_{UQsim} is the time for a complete UQ simulation, and T_{Ass}, T_{Prop}, T_{Cert} for the three different UQ phases

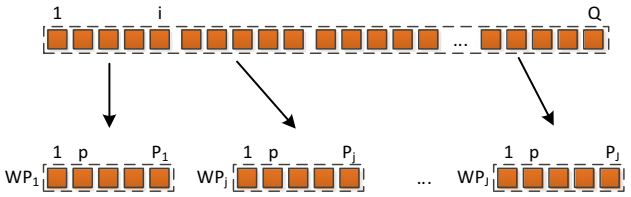

Fig. 4 Illustration of a set of cubature points and the resulting work package

packages are identical. Hence, a mapping between the index i (coming from the non-intrusive polynomial chaos approach, see Eq. (4)) and the local index $p = 1, \dots, P_j$ for a specific work package has to be realised, to allow a backward mapping which is required in the certification phase.

Depending on the configuration of the computing environment, a computing unit can be a single core on a CPU, a set of cores on a CPU, a complete node in a cluster, or a set of cluster nodes.

In Table 1, all time denotations are listed that are relevant for the different scheduling strategies in the following. Specific additional times may be defined for specific strategies locally in the corresponding sections. By taking the maximum of the runtime consumed by actually solving the work packages, communicating with the master computing unit, and idling over all work packages $j = 1, \dots, J$, we define T_S, T_C and T_I, respectively:

$$T_S := \max_j(T_{WP_j}) \tag{8}$$

$$T_C := \max_j(T_C^j) \tag{9}$$

$$T_I := \max_j(T_I^j) \tag{10}$$

Note that T_C comprises all communication to distribute and collect work from or to the computing units. In the considered UQ simulations, only low amounts of data

Table 1 Listing of used time measurements to compare different scheduling strategies

Denotation	Meaning
T_{UQsim}	Time of the whole UQ simulation
T_{Ass}	Time of the assimilation phase (only the technical part)
T_{Prop}	Time of the propagation phase
T_{Cert}	Time of the certification phase
$\hat{T}_{Prop}$	Theoretical optimal propagation time (no idling, cannot get faster)
T_S	Maximum time of executions of black-box computer model runs over all work packages
T_S^i	Time of solving the black-box computer model run i
T_C	Maximum total time of communication
T_C^j	Time of communication with work package j
T_I	Maximum total time of idling over all work packages
T_I^j	Time of idling of each work package
T_{WP_j}	Time of solving work package j (including idling)
$T_{WP_j}^p$	Time of solving black-box computer model run with index p in work package j

For predicted or estimated times later in this paper, the symbol $\mathbb{T}$ will be used

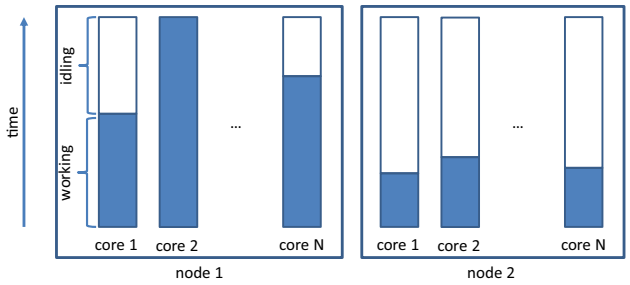

Fig. 5 Illustrative example of idling for two computing nodes. The solid (blue) area shows the time a core is working and the white area marks the time a core is idling

need to be transferred, which does not take much time. Therefore, $T_C << T_I$ and, thus, is not subject to optimisation in this paper. The idling time of each resulting work package T_I^j limits the performance and will be reduced by different approaches. The problem of idling is described in Sect. 3 in more detail.

The actual runtime of the whole propagation phase then consists of the two components T_S and T_C,

$$T_{Prop} = T_S + T_C \,, \tag{11}$$

where typically $T_C << T_S$ holds. The time a computing unit in charge of work package j is actually requiring to solve all of the P_j black-box computer model runs in its work package is denoted by T_{WP_j}. This is the sum of all individual black-box computer model runtimes $T^p_{WP_j}$, $p = 0, \ldots, P_j$ and the idling time of the corresponding work package j:

$$T_{WP_j} = \sum_{p=0}^{P_j} T^p_{WP_j} \; + \; T_I^j \,. \tag{12}$$

The optimal runtime $\widehat{T}_{Prop}$ for the propagation phase would be achieved if and only if no idling takes place in any work package, i.e., if a perfect load balancing could be achieved:

$$T_{Prop} \to \widehat{T}_{Prop} \quad \Longleftrightarrow \quad T_I^j \to 0 \quad \forall j = 1, \ldots, J \,. \tag{13}$$

3 Idling with standard scheduling techniques

A computing unit is idling in a considered time period, if it is not 100% utilised. Figure 5 illustrates a situation where idling happens for the example of a computing unit consisting of at least two cluster nodes: Core 2 on node 1 is working the whole time and is therefore fully utilised—but core 1 to N (without core 2) do not work the whole time and idle after they have finished their work. On node 2, the cores finished their work soon (compared to node 1) and also idle for a considerable time.

Table 2 List of investigated scheduling strategies with their abbreviation

Abbreviation	Strategy	Section
SWP	Static work packages	Sect. 3.1
SWPT	Static work packages with thread pool on node level	Sect. 3.2
DWP	Dynamic work packages	Sect. 3.3

As indicated above, such an idling can happen in UQ simulations in particular if uncertain parameters affect the initial conditions, boundary conditions, time step sizes, or stopping criteria of the simulation. In such a situation, the workload is not optimally balanced between the available computing units.

The consequences of the idling are that (1) the execution takes longer than expected, (2) the execution time of the whole UQ simulation is not really predictable,[2] and (3) it wastes resources because often the computing units are exclusively assigned to a certain job until it has finished.[3]

The amount of idling depends on the specific scheduling strategy. In the following sections, we address the three different scheduling strategies that are listed in Table 2, analyse their idling behaviour, and try to reduce it. All of the three scheduling strategies rely on the initial order of the cubature points z_i. If the runs associated with the z_i are computed serially one after another in the order as they are constructed in the assimilation phase, then this strategy is known in the field of scheduling as first come first served (FCFS). In [33], the authors show that FCFS can have a varying or poor utilisation.

For the sake of simplicity, we assume in this work that the computer models under investigation run on one core (all concepts and acknowledgements presented in the following translate to more general cases but are considerably more complicated to visualise and describe). We further assume that all nodes on a cluster are homogeneous (i.e. have the same hardware configuration).

[2] This may be tedious or disadvantageous in situations where one wants or needs to run scenarios on a larger compute cluster. Access policies of such larger systems typically require the specification of an upper limit of the total runtime which considerably influences the start time of the whole job. If a job exceeds the specified time limit then usually the job is cancelled from the job scheduling system.

[3] A different interesting approach to manage resources is the field of invasive computing [32], where a job can request and release resources dynamically while it is running. This helps to share resources while executing many jobs in parallel.

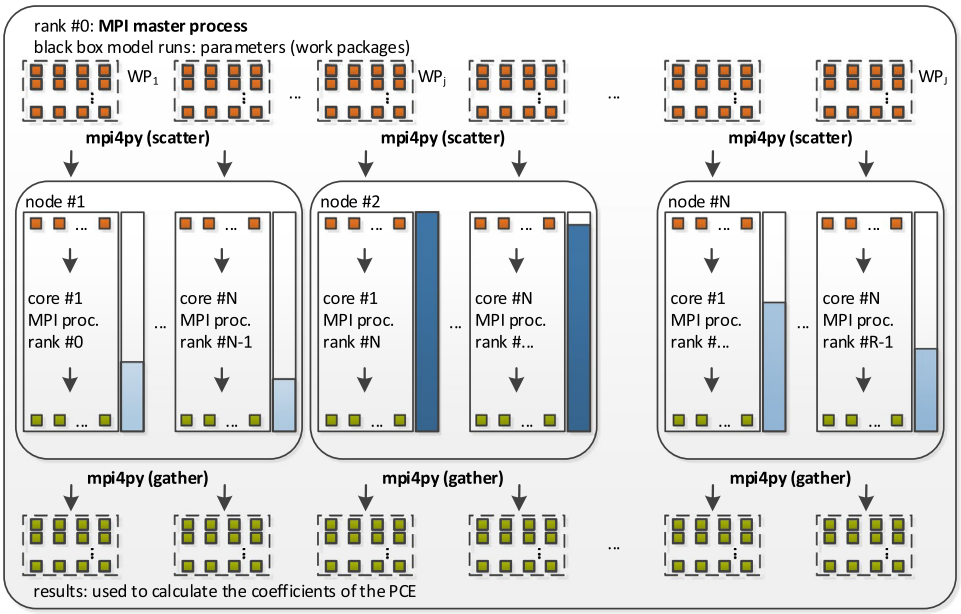

Fig. 6 Illustration of the static work packages (SWP) strategy. The whole cubature points are equally arranged to work packages. Each core works on one work package and for the data transfer MPI is used. The filled (blue) bar beside each core visualises the time a core is working, and the white area denotes the idling time

3.1 Static work packages

For the scheduling approach using static work packages (SWP), all the work packages WP_j are prepared and distributed to the computing units at the beginning of the propagation phase. After all have finished their work, the results are sent back at the end of the propagation phase. While all computing units are working, no data transfer takes place.

Figure 6 illustrates such a situation: the entire set of cubature points $(1, \ldots, Q)$ is evenly assigned to the work packages WP_j. Each work package contains the same number of cubature points (except the last ones in cases where $Q \bmod J \neq 0$). To transfer data on the HPC system between the computing units we use the message passing interface (MPI) standard [34] via the mpi4py [35] Python library. On each core, exactly one MPI process runs, and each MPI process works on one work package. MPI process rank 0 is defined to be the master MPI process, which realises the technical part of the assimilation phase, controls the data transfer, and completes the whole certification phase. In the SWP approach, rank 0 also participates in the propagation phase and works on its own work package. Hence, the number of work packages is $J = \#nodes \times \#cores\_per\_node$. For the distribution of the work packages and the collection of the results, the MPI commands `scatter` and `gather` are used.

In such a static work package setting, idling can happen easily: All the WP_j have the same number of cubature points but each black-box computer model run may have a different runtime $T^p_{WP_j}$, resulting in different runtimes T_{WP_j} for different work packages. If the black-box computer model runs with the long runtimes are contained only in a few work packages then most of the computing units will idle for a long time T^j_I.

3.2 Static work packages with thread pool on node level

The static work packages with a thread pool on node level (SWPT) is similar to SWP, but it only has one MPI process per cluster node. At the beginning of the propagation phase, the work packages WP_j are prepared and distributed to the computing units, and after all have finished their work, the results are sent back. There is, again, no data transfer between the MPI processes during the propagation phase.

In Fig. 7 such a SWPT situation is illustrated. The cubature points z_i are evenly assigned to the work packages WP_j, similar to SWP. For the MPI communication, also the `scatter` and `gather` commands are used. In SWPT, rank 0 also participates in the propagation phase and works on its own work package. The difference to SWP is that there are only $J = \#nodes$ work packages. On each node, there is a thread pool that dynamically distributes the work to the cores. As soon as one core has finished one black-box run it immediately sends the data back to the thread pool and receives the next parameters to proceed with another black-box computer model run.

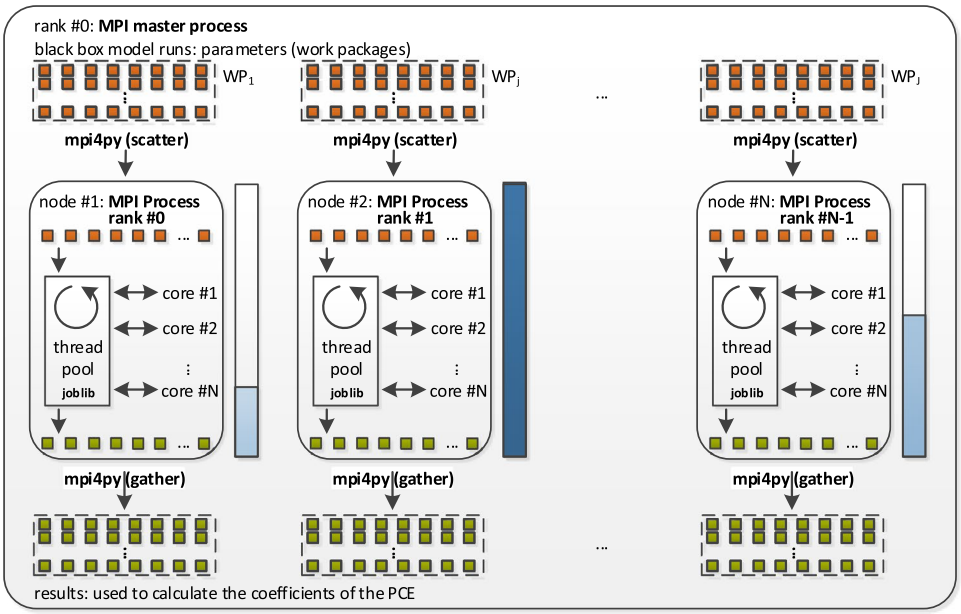

Fig. 7 Illustration of the static work packages with thread pool on node level (SWPT) strategy. The whole cubature points are equally arranged to work packages. Each node works on one work package and for the data transfer MPI is used. On the node level, a thread pool is used to utilise the CPU cores. The filled (blue) bar beside each node visualises the time a node is working, and the white area denotes the idling time

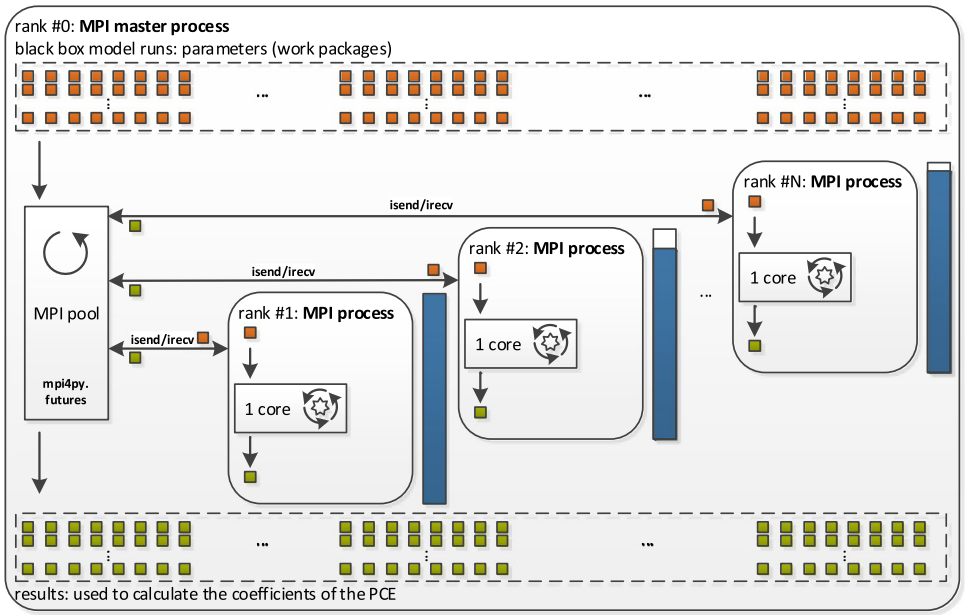

Fig. 8 Illustration of the dynamic work packages (DWP) strategy with a thread pool on MPI level. On every core one MPI worker process is instantiated. Rank 0 dynamically distributes the work to the MPI workers and collects their results. The filled (blue) bar beside each node visualises the time a core is working, and the white area denotes the idling time

To setup the thread pool, the Python library joblib [36] is used.

SWPT also tends to idle easily: if the black-box computer model runs with the long runtimes are only contained in a few work packages, a few nodes have to work for longer periods while the others idle. But due to the thread pool on node level, the idling is usually not that prominent as in SWP.

The problem with idling in SWP and SWPT is that (1) there is no dynamic update of new work to a computing unit after it has finished its initial work package, and (2) the work packages contain the same number of cubature points—but not the same amount of work (runtime).

3.3 Dynamic work packages

To tackle the problem of idling, the dynamic work package (DWP) strategy may represent a suitable approach. The idea is to not split the work into fixed work packages at the beginning of the propagation phase. Instead, the cubature points z_i are distributed dynamically, and therefore the work packages are build dynamically during the propagation phase.

DWP uses a pool on MPI level (MPI pool) to distribute the work to the computing units, as it is illustrated in Fig. 8. On each core, one MPI worker process runs. Rank 0, the master process, does the management and controls the

distribution of the work as well as the collection of the results. The idea is similar to a thread pool, but beyond cluster node boundaries. The master MPI process takes one cubature point z_i after another and sends the data via the MPI command `isend` to the next free MPI worker process. As soon as an MPI worker process has finished its work, the results are send back to the master process via `irecv`, and the next cubature point is send to the worker. In this strategy, rank 0 does not participate in black-box computer model runs: Its job is solely to manage the MPI worker processes.

To implement an MPI pool, we use the `mpi4py.futures` Python package [37] which offers the `MPICommExecutor` class. The `MPICommExecutor` is an elegant way to instantiate a pool on MPI level, because the mpi4py package handles the complete communication part.

The DWP strategy can improve (see Sect. 6) the overall runtime of a UQ simulation by significantly reducing the idling compared to SWP and SWPT. Due to the dynamic distribution, almost all cores are continuously working until everything is done. Only at the very end when no more cubature points are available, the MPI workers tend to idle.

DWP still has some problems: If the black-box computer model run with the longest runtime is started last, then all other cores will still idle for a relatively long period. Furthermore it is still not possible to predict the runtime of a UQ simulation.

4 Runtime prediction

In Sect. 1 we described that it is hard to predict the runtime of a computer model if its runtime varies, because it depends on the values of its input parameters. In the non-intrusive polynomial chaos approach (Sect. 2.1) we do already measure physical values as the output if interest of a computer model.

To tackle the questions (1)–(2), we propose to measure the runtime as an additional synthetic output of interest and use this data to create a runtime predictor. Therefore, the function $f(x,t,z_i)$ in Eq. (4) is extended to additionally comprise the runtime of the executed computer model run. The additional component regarding the runtime is denoted by $\hat{f}(x,t,z_i)$ in the following. In the certification phase, the runtime predictor rp_N is constructed and is defined as the whole gPCE for the runtime,

$$rp_N(x,t,\boldsymbol{\zeta}) := \sum_{j=0}^{N} \hat{c}_j(x,t)\cdot \Phi_j(\boldsymbol{\zeta}), \tag{14}$$

and is calculated using the runtime coefficients $\hat{c}_j(x,t)$ approximated as

$$\hat{c}_j(x,t) \approx \frac{1}{\hat{\gamma}_j}\sum_{i=1}^{Q} \hat{f}(x,t,z_i)\Phi_j(z_i)w_i. \tag{15}$$

The runtime predictor rp_N is a function which depends, in particular, on the uncertain parameters $\boldsymbol{\zeta}$. For each specific choice of parameter values, rp_N returns the corresponding predicted runtime of the computer model. The workflow to create and use the runtime predictor rp_N is as follows: In the training phase, a UQ simulation is executed as usual—but the extended $\hat{f}(x,t,z_i)$ is computed. Additionally, the rp_N is created with Eq. (14) in the certification phase and saved into a file. For the next UQ simulation, the rp_N is loaded from a file. In the assimilation phase, the cubature points z_i are then generated as usual. For each z_i, the rp_N is called once to predict the corresponding runtime $\mathbb{T}_S^i$. Depending on the scheduling strategy that is used in the propagation phase, $\mathbb{T}_{WP_j}^p$ and therefore $\mathbb{T}_{WP_j}$ can be predicted. Due to $T_C << T_S$ it is possible to predict $\mathbb{T}_{Prop}$. Even the whole $\mathbb{T}_{UQsim}$ may now be predicted if T_{Ass} and T_{Cert} are known.

Remark 1 Obviously, this approach involving training requires multiple or adaptive evaluations.[4] Frequently, such techniques are the way to proceed to generate accurate and reliable UQ results. Under restrictions on the computational budget, different UQ simulations are performed which differ in their level of fidelity (e.g., w.r.t. the number of cubature points Q or gPCE terms N).

To determine the quality of a runtime predictor rp_N, we measure the error between the real runtime T_S^i for a cubature point z_i and the corresponding predicted runtime $\mathbb{T}_S^i$. The absolute error is defined as

$$\epsilon r_i := |T_S^i - \mathbb{T}_S^i|, \tag{16}$$

and the relative error as

$$\epsilon r_{i,\text{rel}} := \frac{\epsilon r_i}{\max(|T_S^i|,|\mathbb{T}_S^i|)} = \frac{|T_S^i - \mathbb{T}_S^i|}{\max(|T_S^i|,|\mathbb{T}_S^i|)}. \tag{17}$$

To compare different runtime predictors, we additionally use $\mu(\epsilon r)$ for the mean error, and the discrete $L^2(\epsilon r)$ error, which are defined as:

$$\mu(\epsilon r) := \frac{1}{Q}\sum_{i=1}^{Q} \epsilon r_i \tag{18}$$

[4] See also adaptive sparse-grids-based approaches such as [38] or [11].

Table 3 List of improved scheduling strategies with their abbreviation

Abbreviation	Strategy	Basis	Section
SWP_OPT	Static work packages with rp_N	SWP	Sect. 5.2
SWPT_OPT	Static work packages with thread pool and rp_N	SWPT	Sect. 5.2
DWP_OPT	Dynamic work packages with rp_N	DWP	Sect. 5.1

$$L^2(\epsilon r) := \sqrt{\frac{1}{Q}\sum_{i=1}^{Q} \epsilon r_i^2} \tag{19}$$

In the next section, we discuss how we can use the gPCE runtime predictor rp_N to reduce idling in combination with the three different scheduling strategies.

5 Optimal execution or propagation order

Idling occurs due to the lack of knowledge about the runtime and the FCFS order (see Sect. 3). But with the rp_N from Sect. 4 we do now have a tool to predict the runtime of a computer model—even under uncertain conditions.

The idea is to predict the runtime with rp_N for each cubature point z_i and use the runtime information to control the propagation of the static and dynamic work package strategies. An execution or propagation order is almost optimal if $T_l \to 0$. However, an optimal propagation order depends on the specific scheduling strategy: Therefore, we enhanced the standard scheduling strategies with the runtime information from rp_N resulting in improved scheduling strategies, which are listed in Table 3.

5.1 Dynamic work packages with rp_N runtime predictor

The DWP strategy of Sect. 3.3 builds the work packages dynamically, by assigning the next z_i from the list of cubature points to the next free computing unit. To solve the problem that the black-box computer model runs with longest runtime are started last, we use the well known longest process time first (LPT) order. All cubature points z_i are first sorted by their estimated runtime in descending order. Then, the propagation phase is started with this ordered list. After all black-box computer model runs have finished, the results have to be reordered to match the original FCFS order. Finally, the whole certification phase can be handled as usual.

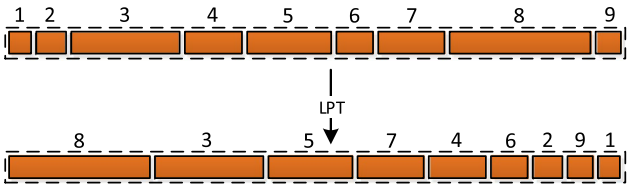

Fig. 9 Illustration of the LPT strategy. The first row shows some unordered entries. The relative runtime of each entry is the width of its rectangle: broader entries take longer. In the second row, the entries are ordered with LPT, where the entries are in descending order according to their runtime

Figure 9 shows an example with nine entries. The first row is unordered, and the width of the rectangles visualises their relative runtime: the bigger the rectangle, the longer the runtime. LPT reorders the entries to descending order, as can be seen in the second row. The quality of a scheduling strategy is often stated with its worst case factor R_m. For DWP using LPT ordering, the $R_m(LPT)$ is (see [39])

$$R_m(LPT) = \frac{4}{3} - \frac{1}{3m}, \tag{20}$$

with m being the number of individual computing units. In our case, $R_m(LPT)$ multiplied by $\hat{T}_{Prop}$ is the longest runtime that we have to assume in the worst case.

The dynamic work packages with the rp_N runtime predictor and the usage of the LPT strategy (DWP_OPT) allow for tackling the questions (1)–(2): the runtime is now predictable with worst case factor $R_m(LPT)$ and the idling is reduced significantly (compare Sect. 6) due to the dynamic scheduling. DWP_OPT additionally has the advantage that it scales automatically to arbitrary numbers of computing units without any code changes.

5.2 Static work packages with rp_N runtime predictor

If the work packages are build as described in Sect. 3.1 (SWP) and Sect. 3.2 (SWPT), each work package WP_j has the same amount of cubature points z_i. This is also known as distributing the work in FCFS order.

The goal is to distribute the cubature points z_i in such a way that each work package WP_j contains the same amount of work (runtime), but not necessarily the same number of cubature points. In scheduling theory, this kind of problem belongs to the *Pm* category [40, p. 14] with m identical processors or computing units. Unfortunately, the preparation of optimal work packages, where each $T_l^j \to 0$, is NP-hard [40, p. 114].

Preparing the work packages in this manner is known as the bin-packing problem in the context of multiprocessors. In [39], the authors introduce a heuristic—the MULTIFIT

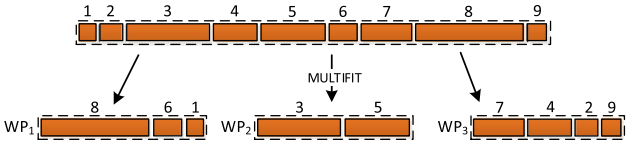

Fig. 10 Illustration of the MULTIFIT strategy. The first row contains the initial unordered entries. The relative runtime of each entry is the width of its rectangle: Broader entries take longer. In the second row, the work packages $WP_1, \ldots, WP_3$ are created with MULTIFIT

algorithm—which is able to produce work packages that are well balanced. In general, the worst case factor $R_m(MF)$ is estimated with

$$R_m(MF) <= 1.222, \tag{21}$$

where further investigations and improvements can be found in [39, 41].

MULTIFIT starts with the initial list of cubature points z_i and applies the LPT to order the list in descending order by the runtime. At the beginning, it is unknown how much work (runtime) should ideally be in one work package WP_j. In our case, the number of work packages J is a fixed constant for MULTIFIT. The algorithm iteratively tries to find with k iterations the work package size capacity C, so that all T_{WP_j} are fairly equal. In each iteration, it applies the first fit decreasing (FFD) algorithm. FFD iterates over all cubature points and fills a work package as long as C is not exceeded. If a work package is full, it goes to the next one. But for each z_i it tests all work packages again, which has the advantage that the work packages can be filled up with cubature points that result in a small runtime.

Figure 10 illustrates the distribution of nine entries into three work packages. The first row starts with an unordered list of entries, and in the second row are the resulting work packages $WP_1, \ldots, WP_3$ and their assigned entries. It can be seen that MULTIFIT produces work packages with similar total runtime T_{WP_j}, but with typically different number of entries. Inside each work package, the entries are ordered by their runtime due to the initial LPT ordering step.

To use the MULTIFIT algorithm within SWP and SWPT, we implemented MULTIFIT and FFD in python. The integration is fairly easy, because it directly replaces the FCFS work package creation code. Besides a reordering of the results to meet the initial order of the cubature points, all other implementations of the scheduling strategies can be reused. For SWPT, the MULTIFIT approach has the additional advantage that the thread pool operates on the LPT ordered list within its work package with $R_m(LPT)$ of Eq. (20).

We denote SWP and SWPT together with the MULTIFIT algorithm by SWP_OPT and SWPT_OPT respectively, because they can reduce the idling significantly with the worst case factor $R_m(MF)$. Via the runtime predictor rp_N and the balanced work packages, the whole propagation runtime $\mathbb{T}_{Prop}$ of a UQ simulation is now predictable.

6 Numerical results

In the following sections, we present numerical results for three different examples: Example 1 uses a simple academic and smooth test function, Example 2 an academic test function with a discontinuity, and Example 3 simulates an evacuation scenario for pedestrian dynamics. For all examples, the quality of the runtime prediction is analysed together with its effect on the scheduling strategies and their resulting propagation time T_{Prop}. In order to compare the examples, all have three uncertain parameters. The number of cubature points per uncertain parameter q was varied between 4 and 12, and different numbers of cluster nodes $cn = 2, 3, 4, 5$ where used for the propagation.

All simulations have been executed on the Linux-Cluster CooLMUC2 of the Leibniz Supercomputing Centre [42]. Each cluster node is equipped with an Intel Xeon E5-2697 v3 ("Haswell") CPU which has 28 cores, and 64 GB of DDR4 RAM. The cluster nodes are interconnected with the FDR14 InfiniBand network.

6.1 Example 1: Simple test function (academic example)

In this example, the forward model is, analytically defined as

$$\hat{f} = 2(e^{5\cdot|x|} + \max(y, 0) + 0.2 \cdot |z|). \tag{22}$$

The motivation for this example is to have a model with an analytical solution to determine the accuracy of the runtime prediction. It has been designed to reproduce a similar behaviour as Example 3 in Sect. 6.3. The function $\hat{f}$ incorporates three parameters $x, y,$ and z which are defined to be uncertain following the probability distributions $U(0.1, 0.5)$, $U(0.8, 1.2)$, and $U(1.4, 1.8)$, respectively.

The value of $\hat{f}$ in Eq. (22) is interpreted as the runtime (in seconds) and the software implementation performs a sleep command of exactly this value to simulate the runtime. This produces different waiting times and therefore different runtimes for the model function.

Figure 11a visualises the real runtimes T_S^i (blue circles) for every cubature point z_i with $q = 7$ resulting in 343 cubature points z_i. The runtime varies from about 5 to 26 s and only a few of the cubature points z_i results in long runtimes. In Fig. 11b, the predicted runtimes $\mathbb{T}_S^i$ (green filled dots) are plotted on top of the real runtime

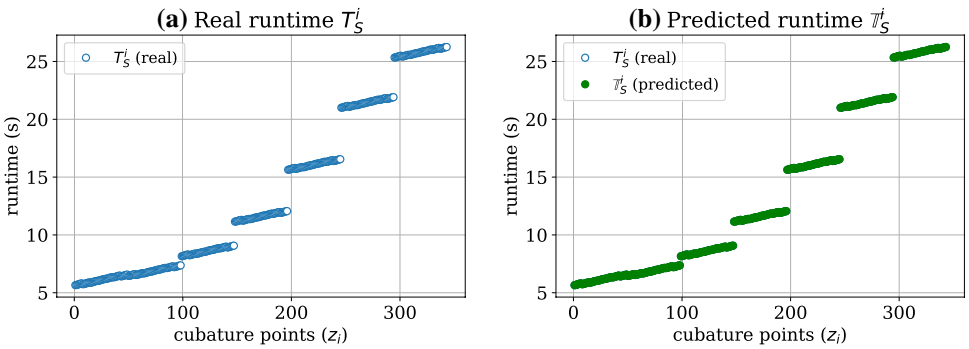

Fig. 11 Example 1: **a** Real runtime T_S^i and **b** predicted runtime $\mathbb{T}_S^i$ versus T_S^i in seconds of $\hat{f}$ (see Eq. (22)) with $q = 7$ cubature points for each of the three uncertain parameters

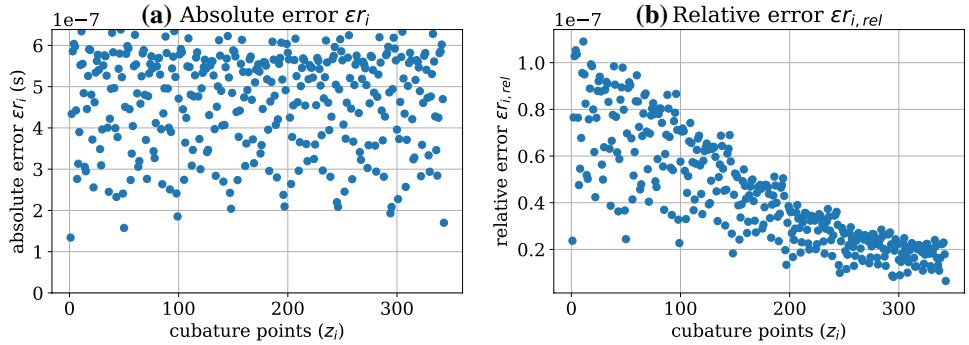

Fig. 12 Example 1: **a** Absolute error ϵr_i of runtime in seconds and **b** relative error $\epsilon r_{i,\text{rel}}$ for $\hat{f}$ (see Eq. (22)) with $q = 7$ cubature points for each of the three uncertain parameters

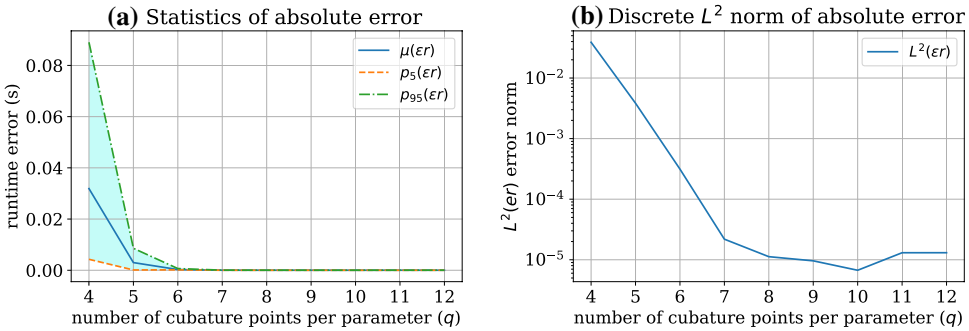

Fig. 13 Example 1: **a** Statistics of the absolute error with mean $\mu(\epsilon r)$, as well as 5th ($p_5(\epsilon r)$) and 95th ($p_{95}(\epsilon r)$) percentiles, and **b** the corresponding discrete $L^2(\epsilon r)$ norm of the absolute error $q = 4, 5, \ldots, 12$ cubature points per parameter

T_S^i (actually hiding it in this particular case due to the high accuracy).

Figure 12a shows the absolute error ϵr_i of the prediction compared to the analytical solution of Eq. (22) for the case $q = 7$. The order of ϵr_i of about 10^{-7} seconds is comparably small. Figure 12b displays the corresponding relative error $\epsilon r_{i,\text{rel}}$. The impression that the relative error is decreasing with higher indices of the corresponding cubature points z_i is only due to higher runtimes together with similar values of the absolute error.

In Fig. 13a, the error statistics with the mean error $\mu(\epsilon r)$, as well as 5th ($p_5(\epsilon r)$) and 95th ($p_{95}(\epsilon r)$) percentiles are shown for $q = 4, 5, \ldots, 12$. For $q = 4, 5$ the error is non-negligible since the predictor has too less information for an accurate prediction. Starting from $q = 6$, the error is comparably small. This is also confirmed by the discrete L^2 error norm in Fig. 13b which is decreasing to about 10^{-5} with increasing number of cubature points per parameter (q).

To compare the different scheduling strategies, all simulations were performed for $q = 4, 5, \ldots, 12$ and with different number of cluster nodes $cn = 2, 3, 4, 5$. The results for $q = 2$ and $q = 5$ are shown in Figs. 14a and 14b, respectively, and indicate that the choice of the scheduling strategy has a huge impact on the overall runtime. The SWP scheduling results in a runtime of about 891 s for $cn = 2$ and $q = 12$, whereas SWPT takes 632 s, and DWP only needs 440 s. SWP_OPT, SWPT_OPT, and DWP_OPT produce runtimes similar to DWP. For $cn = 5$, the runtimes of the different scheduling approaches behave qualitatively similar to $cn = 2$ (and actually also to $cn = 3, 4$), except for the solid red dot and the dashed purple triangle curves (SWP_OPT and SWPT_OPT, respectively) which are not

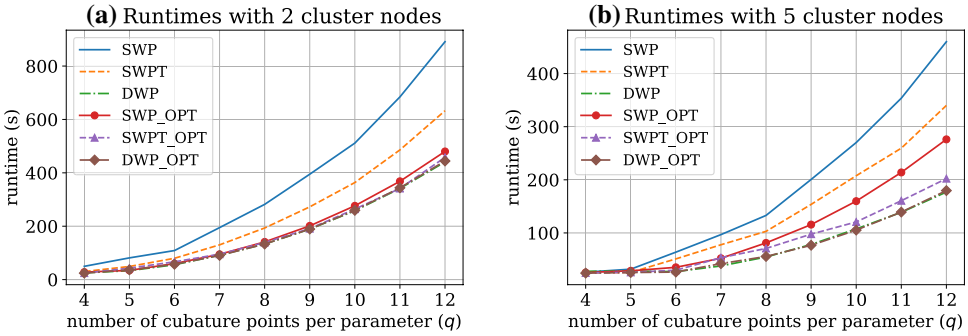

Fig. 14 Example 1: Measured propagation runtimes T_{Prop} in seconds for the six scheduling strategies for **a** 2 cluster nodes, and **b** 5 cluster nodes

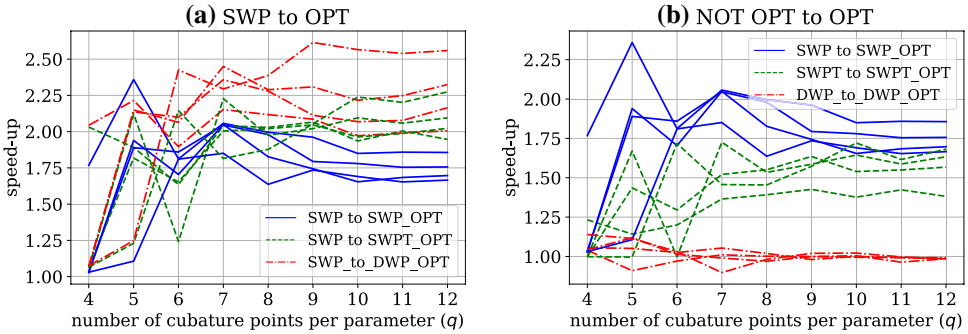

Fig. 15 Example 1: **a** Speed-up for T_{Prop} of SWP compared to SWP_OPT, SWPT_OPT, and DWP_OPT. For each scheduling strategy, the plot contains 4 speed-up lines (of identical colour and line style) corresponding to $cn = 2, 3, 4, 5$. **b** Speed-up of the standard scheduling strategies compared to its optimised counterparts: SWP to SWP_OPT, SWPT to SWPT_OPT, and DWP to DWP_OPT. Again, 4 speed-up lines of identical colour and line style are given for $cn = 2, 3, 4, 5$

competitive to DWP and DWP_OPT anymore for larger values of q.

Figure 15 further helps to investigate the propagation runtimes T_{Prop} of the scheduling strategies: Fig. 15a shows the speed-up of SWP_OPT, SWPT_OPT, and DWP_OPT compared to SWP. For each scheduling strategy, four lines are plotted, corresponding to $cn = 2, 3, 4, 5$. DWP_OPT versus SWP produces the highest speed-up (ranging from about 2.0 to 2.5 for $q = 12$), followed by SWPT_OPT (ranging from about 1.9 to 2.2 for $q = 12$) and lastly by SWP_OPT (ranging from about 1.6 to 1.8 for $q = 12$). To see the improvements from the standard scheduling strategy with its improved version, Fig. 15b is helpful. The improvements from SWP to SWP_OPT are the highest (with about 1.6–1.8 for $q = 12$). Also, SWPT can be significantly improved by SWPT_OPT (with about 1.3–1.6 for $q = 12$). In the case of DWP to DWP_OPT (with about 0.98–0.99 for $q = 12$) there are no significant runtime improvements visible; a theoretical advantage is that there is the worst case boundary $R_m(LPT)$ which cannot be exceeded. In both plots of Fig. 15, the speed-up is unstable for $q = 4, 5, 6$ since there are only a few or no black-box computer model runs for parts of the computing units and a few outlier can have a huge impact on the overall propagation runtime T_{Prop}.

The choice of the scheduling strategy has a high impact on the whole propagation time T_{Prop} for this simple and smooth test function. The constructed runtime predictor rp_N produces a high accuracy and it can predict the runtime $\mathbb{T}_S^i$ very well. Therefore we can successfully use the runtime predictions $\mathbb{T}_S^i$ to improve the standard scheduling strategies and, thus, reduce the overall runtime of the whole UQ simulation T_{UQsim} in this example.

6.2 Example 2: Function with discontinuity (academic example)

This example uses a function with a discontinuity to study how the prediction and the scheduling strategies behave in a more challenging non-smooth setup.[5] We took a similar test function as proposed in [43], but extended it by a third parameter. Hence, $\hat{f}$ reads

$$\hat{f} = e^{-x^2+2\,\mathrm{sign}\,(y)} + z. \tag{23}$$

[5] If it is known in advance that the model is non-smooth, the non-intrusive polynomial chaos approach Eq. (1) is not the method of choice—for that other suitable approaches exist. But for our use case it cannot be assumed that $\hat{f}$ and f are correlated and often the runtime behaviour is not known, that is why a runtime predictor rp_N is built here and we reuse the already applied UQ approach.

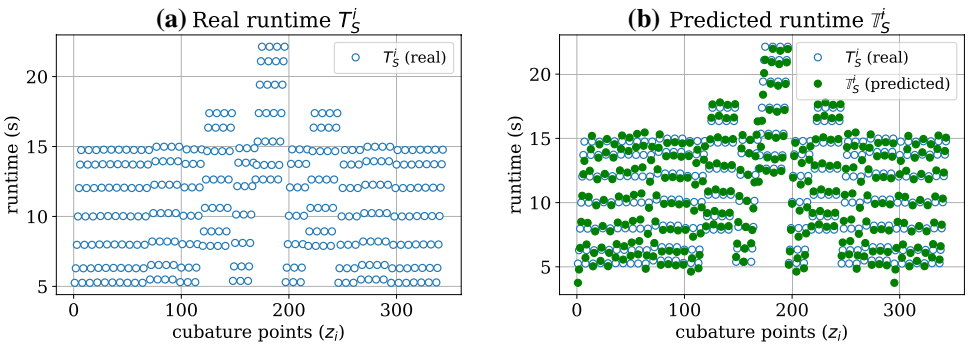

Fig. 16 Example 2: **a** Real runtime T_S^i and **b** predicted runtime $\mathbb{T}_S^i$ versus T_S^i in seconds of $\hat{f}$ (see Eq. (23)) with $q = 7$ cubature points for each of the three uncertain parameters

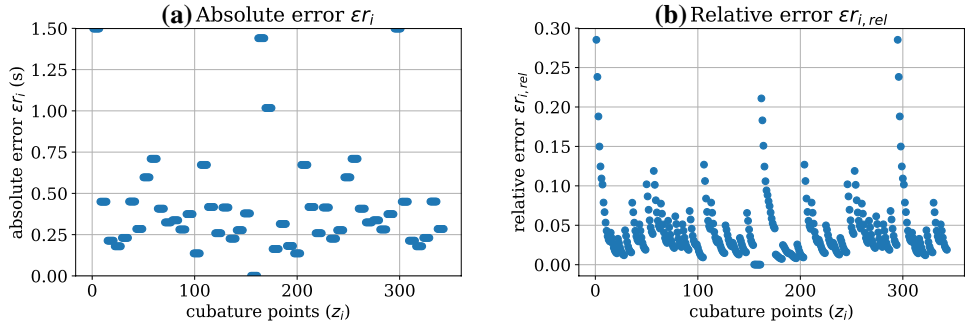

Fig. 17 Example 2: **a** Absolute error ϵr_i of runtime in seconds and **b** relative error $\epsilon r_{i,\text{rel}}$ for $\hat{f}$ (see Eq. (23)) with $q = 7$ cubature points for each of the three uncertain parameters

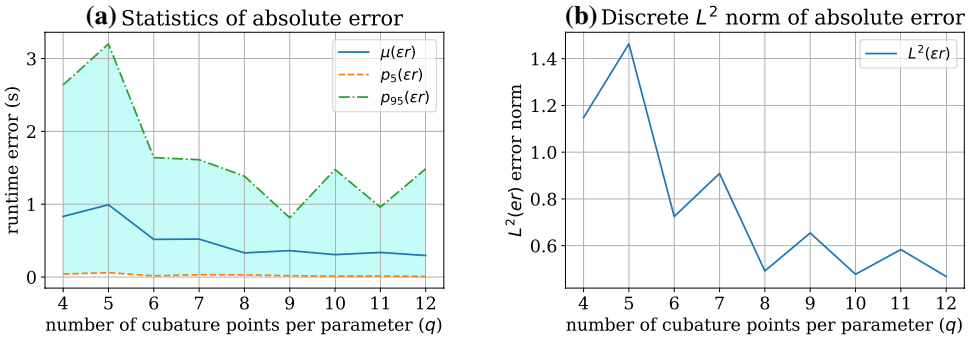

Fig. 18 Example 2: **a** Statistics of the absolute error with mean $\mu(\epsilon r)$, as well as 5th ($p_5(\epsilon r)$) and 95th ($p_{95}(\epsilon r)$) percentiles, and **b** the corresponding discrete $L^2(\epsilon r)$ norm of the absolute error $q = 4, 5, \ldots, 12$ cubature points per parameter

The three parameters x, y, and z are defined to follow the uniform distributions $U(-2.5, 2.5)$, $U(-2.0, 2.0)$, and $U(5.0, 15.0)$, respectively. The value of $\hat{f}$ is, again, interpreted as the runtime in seconds and for simulation purposes the implementation executes a sleep command of exactly this value.

Figure 16a visualises the measured real runtime T_S^i (blue circles) which ranges from about 5.4 to 21.9 s, and Fig. 16b shows the predicted runtime $\mathbb{T}_S^i$ (green filled dots). As expected, the green dots do not perfectly match the blue dots due to prediction errors.

The absolute and relative prediction error is shown in Fig. 17. The absolute error er_i in Fig. 17a is clustered mainly around 0.2 and 0.75 s, and only a few outliers exist at about 1.5 s. The relative error $er_{i,rel}$ in Fig. 17b ranges mainly from 0.01 to 0.1 and reaches around 0.3 for a few cubature points.

The statistics of the absolute error in Fig. 18a confirms the impression of higher errors compared to Example 1, with a mean $\mu(\epsilon r)$ of about 0.3–1.0 for different values of q. The discrete L^2 norm of the absolute error in Fig. 18b is slightly decreasing with higher values of q but remains in the order of one.

Concerning the scheduling strategies, Fig. 19a visualises the propagation runtimes T_{Prop} for $cn = 2$ cluster nodes: SWP takes about 587 s for $q = 12$, SWP_OPT 424 s, and all other scheduling strategies perform relatively similar with about 351-359 s. In the case of $cn = 5$ (Fig. 19b) SWP needs about 322 s, SWP_OPT 249 s, SWPT 183 s, and the others are roughly similar at about 150 s.

The T_{Prop} speed-ups of to the optimised scheduling strategies versus pure SWP are given in Fig. 20a (again, four lines of identical colour and line style represent the different amount of cluster nodes $cn = 2, 3, 4, 5$ per

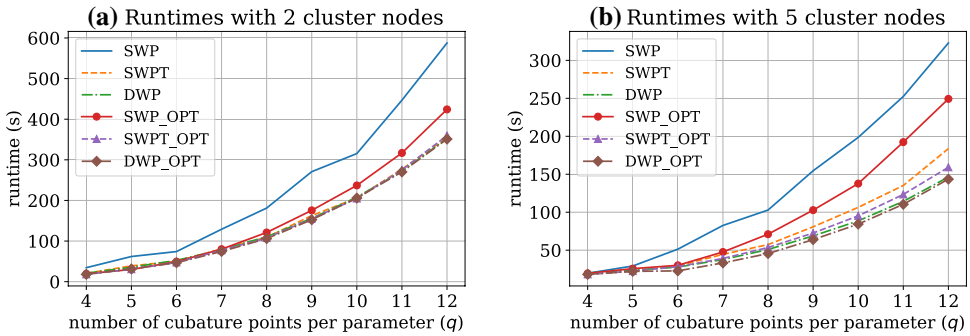

Fig. 19 Example 2: Measured propagation runtimes T_{Prop} in seconds for the six scheduling strategies for **a** 2 cluster nodes, and **b** 5 cluster nodes

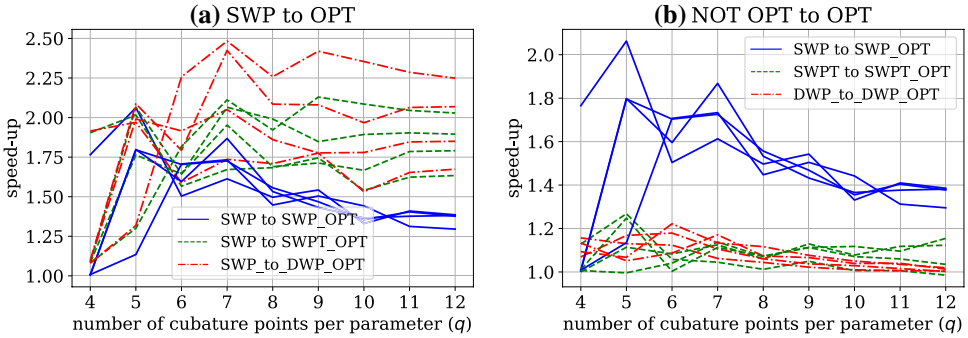

Fig. 20 Example 2: **a** Speed-up for T_{Prop} of SWP compared to SWP_OPT, SWPT_OPT, and DWP_OPT. For each scheduling strategy, the plot contains 4 speed-up lines (of identical colour and line style) corresponding to $cn = 2, 3, 4, 5$. **b** Speed-up of the standard scheduling strategies compared to its optimised counterparts: SWP to SWP_OPT, SWPT to SWPT_OPT, and DWP to DWP_OPT. Again, 4 speed-up lines of identical colour and line style are given for $cn = 2, 3, 4, 5$

strategy): SWP_OPT improves the runtime by a factor of about 1.3, SWPT_OPT by about 1.6–2.0, and DWP_OPT has the highest speed-up rates of about 1.6–2.2. The speed-up comparison of the standard scheduling methods with their improved counterparts is plotted in Fig. 20b: SWP to SWP_OPT has the highest speed-up, followed by SWPT to SWPT_OPT with about 0.99–1.15. Again, there is no significant speed-up in the case of DWP to DWP_OPT. Starting from $q = 7$, the speed-up rates stabilise, due to the then sufficient amount of workload for the computing units. In addition, the larger number of runs reduces the effect of few outliers on the overall results.

To summarize this second example including a discontinuity: Predicting the runtime T_S^i for Eq. (23) is possible but with a noticeable absolute and relative error. Despite these errors, it is possible to significantly improve the propagation runtime T_{Prop} by either using DWP or the optimised scheduling strategies (SWP_OPT, SWPT_OPT, or DWP_OPT). Hence, the scheduling strategy does also have a huge impact on the overall simulation runtime T_{UQsim}.

Table 4 List of uncertain parameters and their distributions for the evacuation scenario in pedestrian dynamics (cf. [8])

Parameter	Description	Distribution
$perc_{fam}$	Percentage of family members (%)	$U(0.1, 0.5)$
v_{parent}	Speed of parent-agents searching their child-agents (m/s)	$U(1.4, 1.8)$
v_{child}	Speed of the parent-child-pair (m/s)	$U(0.8, 1.2)$

6.3 Example 3: Evacuation of pedestrians with separated families

Example 3 consists of the evacuation scenario in pedestrian dynamics with separated families described in Sect. 2.2. This example has the longest absolute runtimes, and $\hat{f}$ represents the measured runtime of a single VADERE simulation run. The three parameters $perc_{fam}$, v_{parent}, and v_{child} are uniformly distributed as listed in Table 4. A UQ analysis concerning the evacuation time has been performed in [8]. In the following, we analyse the runtime and the performance of the different scheduling strategies, similar to example 1 and 2.

The runtime analysis starts with the measured real runtimes T_S^i (blue circles) in Fig. 21a for $q = 7$, which clusters in

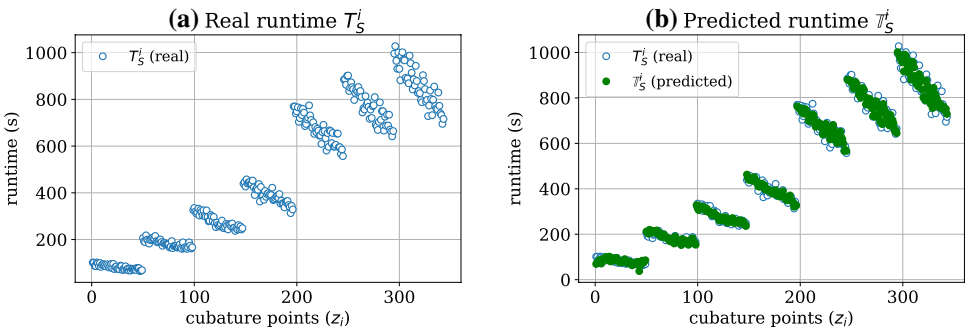

Fig. 21 Example 3: **a** Real runtime T_S^i and **b** predicted runtime $\mathbb{T}_S^i$ versus T_S^i in seconds for the evacuation scenario in pedestrian dynamics (Sect. 2.2) with $q = 7$ cubature points for each of the three uncertain parameters

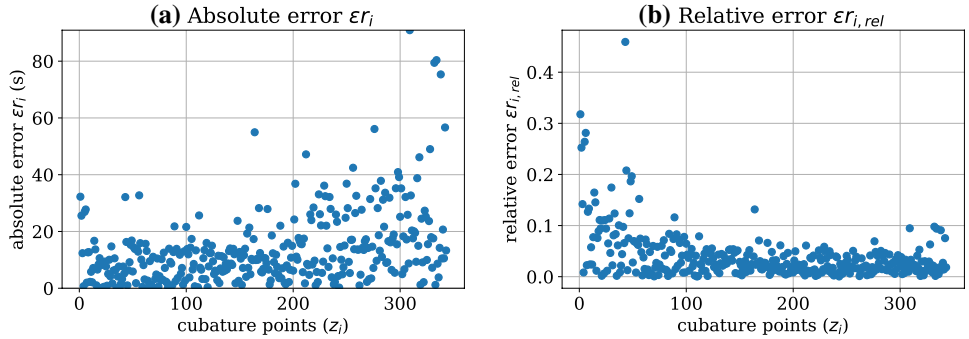

Fig. 22 Example 3: **a** Absolute error ϵr_i of runtime in seconds and **b** relative error $\epsilon r_{i,\text{rel}}$ for the evacuation scenario in pedestrian dynamics (Sect. 2.2) with $q = 7$ cubature points for each of the three uncertain parameters

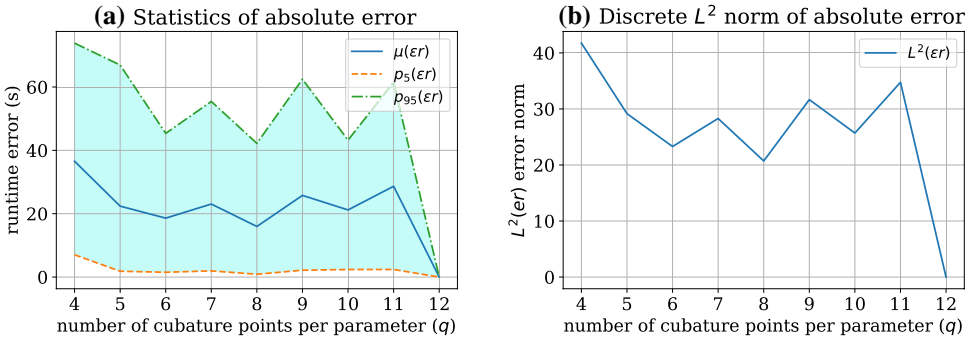

Fig. 23 Example 3: **a** Statistics of the absolute error w.r.t. $q = 12$ with mean $\mu(\epsilon r)$, as well as 5th ($p_5(\epsilon r)$) and 95th ($p_{95}(\epsilon r)$) percentiles, and **b** the corresponding discrete $L^2(\epsilon r)$ norm of the absolute error $q = 4, 5, \ldots, 12$ cubature points per parameter

a wide range of about 64–1028 s. In Fig. 21b, the predicted runtime T_S^i (green filled dots) reproduces the clusters but does not perfectly match every single (blue) circle due to some prediction error.

Figure 22a shows the absolute error ϵr_i for each T_S^i (for $q = 7$), which ranges from about 0.1 to 40 s. For a few black-box computer model runs, the absolute error reaches up to 90 s. The corresponding relative error is shown if Fig. 22b: $\epsilon r_{i,\text{rel}}$ spreads mainly from 0 to 0.1. Only for the first cubature points z_i, the relative error reaches up to 0.45.

The quality of the runtime prediction $\mathbb{T}_S^i$ for different $q = 4, 5, \ldots, 12$, is visible in Fig. 23. For this example, the rp_N for $q = 12$ is taken as the reference model, because no analytical model is available. The mean error $\mu(\epsilon r)$ is high at about 15.9–36.5 s (i.e. Fig. 23a). Furthermore, Fig. 23b shows this with high values of about 20.7–41.7 for the L^2 norm of the absolute error. Another observation is that the $L^2(\epsilon r)$ does not decrease with higher values of q. This is most probably due to the additional intrinsic uncertainty inside VADERE (w.r.t. route planning etc.) that is not controlled by one of the three uncertain parameters, and, thus, resulting in slightly different runtimes in subsequent identical calls.

The propagation runtimes T_{Prop} for the different scheduling strategies are plotted in Fig. 24a for $cn = 2$. For $q = 12$, SWP needs 29,109 s, SWPT 23,696 s, followed by SWP_OPT, SWPT_OPT, DWP, and DWP_OPT which are closely together between 14,623 and 17,055 s. With $cn = 5$ computing nodes (Fig. 24b), the propagation time for SWP is 12,646 s, SWPT 11,133 s, SWP_OPT 10,364 s, and SWPT_OPT, DWP, as well as DWP_OPT are between 5819 and 6406 s. All data on the propagation runtime T_{Prop} ranging

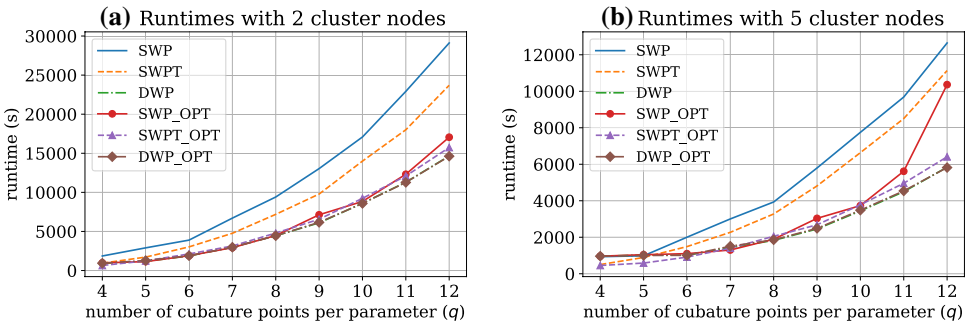

Fig. 24 Example 3: Measured propagation runtimes T_{Prop} in seconds for the six scheduling strategies for **a** 2 cluster nodes, and **b** 5 cluster nodes

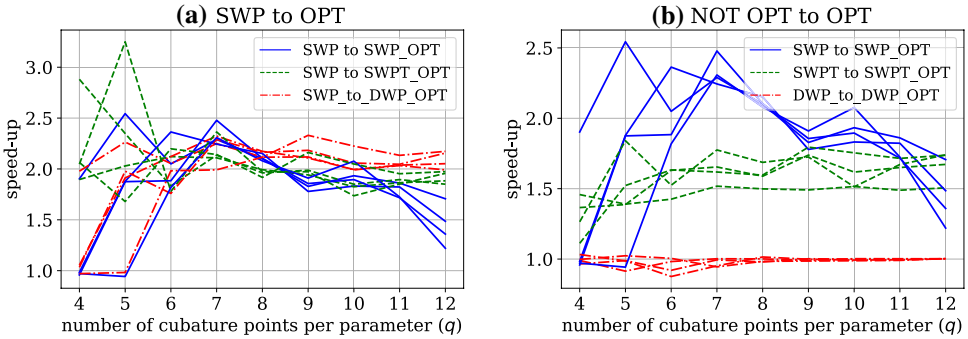

Fig. 25 Example 3: **a** Speed-up for T_{Prop} of SWP compared to SWP_OPT, SWPT_OPT, and DWP_OPT. For each scheduling strategy, the plot contains 4 speed-up lines (of identical colour and line style) corresponding to $cn = 2, 3, 4, 5$. **b** Speed-up of the standard scheduling strategies compared to its optimised counterparts: SWP to SWP_OPT, SWPT to SWPT_OPT, and DWP to DWP_OPT. Again, 4 speed-up lines of identical colour and line style are given for $cn = 2, 3, 4, 5$

from $q = 4, 5, \ldots, 12$ for each $cn = 2, 3, 4, 5$ are additionally available in Table 5.

The speed-up for T_{Prop} of SWP compared to the optimised versions with 4 identical colours and line styles for $cn = 2, 3, 4, 5$ are plotted in Fig. 25a: SWP to SWP_OPT shows a factor of 1.2–1.7, SWP to SWPT_OPT of 1.8–1.9, and SWP to DWP_OPT ranges from 1.9 to 2.1. The speed-ups of the optimised scheduling strategy compared to its base is shown in Fig. 25b: SWP_OPT can optimise its base scheduling strategy by a factor of 1.2–1.7, SWPT_OPT of 1.5–1.7, and DWP_OPT does not improve its base with a relative constant factor of about 1.0.

The results for Example 3 shows that the runtime prediction for $\mathbb{T}_S^i$ results in a comparably high absolute prediction error ϵr with a considerable relative error $\epsilon r_{i,\text{rel}}$ of about 10%. A constantly high value for the $L^2(\epsilon r)$ norm is observed, even with increasing q. Despite this challenging situation, the choice of a suitable (optimised) scheduling strategy still significantly improves the overall propagation time T_{Prop}.

6.4 Summary of the numerical results

In the previous sections, the runtime prediction and the scheduling behaviour of three different examples were analysed: For Example 1 which uses a smooth function for $\hat{f}$, the prediction quality was good with a discrete L^2 error that decreases to the order of 10^{-5} for increasing q. The runtime for the $\hat{f}$ function with the discontinuity in Example 2 is harder to predict: With increasing q, the discrete $L^2(\epsilon r)$ norm decreases but still remains high at the order of one. The runtime of Example 3 is even harder to predict which is indicated by the discrete L^2 error with values up to 41.7.

All three examples demonstrate that the choice of the scheduling strategy has a huge impact on the whole propagation time T_{Prop}. Among the standard scheduling strategies, DWP excelled in all cases to be the one with the smallest runtime, followed by SWPT and lastly by SWP. Due to the dynamic scheduling, DWP and SWPT can, compared to SWP, reduce the idling and therefore the propagation time T_{Prop}. With the help of the runtime predictor rp_N, the standard scheduling strategies SWP and SWPT can be significantly improved for $q = 12$ by speed-up factors up to 1.8 and 1.7, respectively. The DWP_OPT scheduling does not reduce the propagation time, but it guarantees to not exceed the worst case factor $R_m(LPT)$ of Eq. (20). Compared to the runtimes of the SWP scheduling, the propagation was speed-up by a factor of about 2.5 for $q = 12$.

Table 5 List of propagation runtimes T_{Prop} (in seconds) for the evacuation scenario in pedestrian dynamics in subsection 6.3 with the six scheduling strategies and their variations: cn denotes the number of used cluster nodes (each cluster node has 28 CPU cores), and q is the number of cubature points for each uncertain parameter

#cn	q	Scheduling strategies (time in seconds)					
		SWP	SWPT	DWP	SWP_OPT	SWPT_OPT	DWP_OPT
2 cn	4	1858.2	940.3	930.2	976.6	644.9	939.4
	5	2900.5	1716.8	1171.0	1140.3	1234.9	1281.0
	6	3889.8	3024.7	1861.2	1897.7	2122.0	1894.5
	7	6704.8	4764.9	2987.7	2928.0	3139.2	2981.2
	8	9405.9	7187.4	4450.6	4476.5	4796.0	4445.4
	9	13,038.7	9783.5	6096.4	7130.9	6561.3	6156.8
	10	17,083.2	13,996.0	8602.8	8836.6	9236.4	8581.0
	11	22,917.8	18,006.1	11,296.0	12,315.3	12,091.7	11,271.5
	12	29,109.4	23,696.7	14,650.4	17,055.0	15,734.7	14,623.4
3 cn	4	973.3	701.1	890.9	988.6	513.3	922.7
	5	1978.4	1352.2	1023.6	1053.2	974.8	1034.7
	6	3011.7	2314.8	1306.3	1274.2	1419.7	1419.6
	7	4637.1	3636.5	1984.4	2066.3	2198.3	1996.0
	8	6562.2	5231.0	2957.3	3049.2	3293.0	3019.5
	9	8510.6	7621.0	4046.4	4784.5	4383.4	4038.2
	10	11,420.9	10,125.0	5723.0	6234.9	6258.0	5726.4
	11	15,295.9	13,539.5	7508.7	8391.2	8203.7	7507.2
	12	19,916.5	17,684.1	9729.6	13,407.3	10,573.8	9707.0
4 cn	4	967.6	597.8	967.0	1009.8	472.7	936.6
	5	1999.8	1132.5	997.9	1066.5	615.2	1011.2
	6	2000.4	1714.2	996.2	1061.7	1124.5	1137.0
	7	3810.4	2861.6	1583.3	1537.6	1612.1	1663.4
	8	4819.5	4118.0	2261.0	2290.6	2441.1	2227.9
	9	6644.0	5811.9	3039.2	3579.0	3368.8	3043.2
	10	8913.2	7768.9	4300.5	4702.8	5133.8	4331.2
	11	11,598.1	10,476.6	5636.6	6755.5	6265.9	5668.2
	12	15,681.1	13,943.8	7287.1	11,533.5	8013.7	7274.3
5 cn	4	934.5	503.4	960.5	962.3	452.8	962.7
	5	983.6	891.4	1025.1	1043.5	585.5	1001.6
	6	2001.7	1487.2	1014.8	1100.8	910.7	1009.8
	7	3003.8	2269.8	1424.9	1301.6	1402.4	1508.6
	8	3932.6	3275.6	1829.1	1890.8	2051.8	1858.9
	9	5795.6	4812.7	2452.0	3034.5	2681.0	2486.7
	10	7749.3	6636.6	3441.1	3732.0	3781.9	3483.1
	11	9683.7	8504.8	4496.3	5616.4	4958.5	4538.5
	12	12,646.0	11,133.1	5839.9	10,364.2	6406.7	5819.3

Loading and saving the runtime predictor rp_N from and to a file takes about 1.6 ms, for all scheduling strategies in all of the three examples. The runtime prediction for all cubature points z_i takes between 0.04 (for $q = 4$) and 4.76 s (for $q = 12$), and the maximum time for resorting the results back to the original order is about 0.26 ms. This makes the use of the predictor possible in less than 0.1 second for $q = 4$, and less than 5 s for $q = 12$ compared to the overall propagation runtime T_{Prop}. For example 3, this is less then 0.08% of the propagation runtime for DWP_OPT scheduling and $q = 12$. Due to the relatively small overhead for predicting and optimising the runtime, it is almost always possible to use it without noticeable performance losses.

Table 6 contains a list of situations and the corresponding recommended scheduling strategy to reduce the idling for an uncertainty analysis. If some kind of a pool mechanism across computing units is available,

Table 6 List of situations and their recommended scheduling strategy

Situation	Recommended scheduling
Only one production run to quantify the uncertainty is possible	DWP
The runtime prediction has an acceptable or good quality	DWP_OPT, DWP, SWPT_OPT
The runtime prediction has very poor quality	DWP
There is no (MPI) pool across computing nodes available	SWPT_OPT, SWP_OPT
No information about the runtime of a computer model is available	DWP
The computer model does not vary in runtime with different input parameters values	DWP, SWPT, SWP

DWP and DWP_OPT scheduling should be the first choice. If such a pool is not available, SWPT_OPT and SWPT should be used over SWP_OPT and lastly SWP.

7 Conclusion and outlook

In this paper, we studied the runtime and scheduling behaviour of non-intrusive polynomial chaos with a full grid approach for computer models whose runtime depends on the input parameter values. The results show that it is possible to predict the runtime T_S^i of black-box computer model runs for academic examples as well as for an evacuation scenario in pedestrian dynamics. For a smooth model function, a good prediction behaviour with a small relative error down to 10^{-5} could be observed. For the academic test function with a discontinuity and the evacuation scenario, a high relative error of about 10% is observed. Research question (1) can therefore be answered affirmatively: Yes, it is possible to predict the runtime of a black-box computer model run.

In all investigated examples, the choice of the scheduling strategy has a huge impact in the overall propagation time T_{Prop}. Three standard scheduling strategies SWP, SWPT, and DWP have been studied. The more dynamically the black-box computer model runs are scheduled, the lower the propagation time. Therefore, DWP is the best scheduling strategy among the standard scheduling strategies. By using the runtime prediction information to control the scheduling behaviour, the optimised versions SWP_OPT, SWPT_OPT, and DWP_OPT have been developed. SWP_OPT and SWPT_OPT significantly improved its basic version, whereas DWP_OPT could not reduce the runtime of DWP, but has the advantage of not exceeding the worst case boundary $R_m(LPT)$. Compared to the SWP scheduling, which produced the longest propagation times in all examples, it is possible to speed-up the runtime with DWP_OPT and DWP by a factor of about 2.5. For this reason, research question (2) can also be answered with yes, it is possible to reduce the idling and the propagation time.

It is obvious that the proposed approach only makes sense for computer models where runtimes are sensitive on parameter values. Using the prediction to optimise the scheduling is not always possible, especially for computational intensive computer models where only one production run can be afforded due to limited computing units or time. Creating the predictor only for using it once, results in significant overhead. If it can be used multiple times in a forward UQ setting, as in the case of UQ analysis scenarios or adaptive approaches, there is a considerable benefit.

Our proposed approach of predicting and reducing the runtime due to the improved scheduling can be generalised to a large variety of applications that show a parameter-dependent runtime behaviour. It is not limited to the non-intrusive polynomial chaos approach, e.g. also the point collocation method is possible. Once the predictor is built, it is further possible to use it in sampling based UQ approaches. The standard scheduling strategies as well as the improved scheduling strategies can also be implemented with other programming languages and be used on different computing clusters.

Extensions to the presented approach may consist in optimising the scheduling not only w.r.t. the runtime but also to the memory usage or other performance relevant parameters. In particular in multilevel forward UQ and in Bayesian inversion scenarios, the prediction of and optimisation w.r.t such parameters may pay off due to the iterative approach.

Acknowledgements We would like to thank the team around Prof. Dr. Gerta Köster and, in particular Dr. Isabella von Sivers, from the Munich University of Applied Sciences for providing VADERE and supporting its usage, as well as sharing the interest on applying UQ on evacuation scenarios in the context of pedestrian dynamics. We also thank Jonathan Feinberg, the author of Chaospy [26], for the maintaining software and the good support. The authors gratefully acknowledge the compute and data resources provided by the Leibniz Supercomputing Centre [44].

Compliance with ethical standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Smith RC (2014) Uncertainty quantification: theory, implementation, and applications. Computational science and engineering. Society for Industrial and Applied Mathematics, Philadelphia
2. Xiu D (2010) Numerical methods for stochastic computations: a spectral method approach. Princeton University Press, Princeton
3. Sullivan T (2015) Introduction to uncertainty quantification, 1st edn. Springer, Berlin. https://doi.org/10.1007/978-3-319-23395-6
4. Meeds E, Welling M (2015) Optimization Monte Carlo: efficient and embarrassingly parallel likelihood-free inference. In: NIPS
5. Neiswanger W, Wang C, Xing EP (2014) Asymptotically exact, embarrassingly parallel MCMC. In: UAI
6. Heuveline V, Schick M, Webster C, Zaspel P (2017) Uncertainty quantification and high performance computing (Dagstuhl Seminar 16372). Dagstuhl Rep 6(9):59–73. https://doi.org/10.4230/DagRep.6.9.59
7. von Sivers I, Templeton A, Künzner F, Köster G, Drury J, Philippides A, Neckel T, Bungartz HJ (2016) Modelling social identification and helping in evacuation simulation. Saf Sci 89:288–300. https://doi.org/10.1016/j.ssci.2016.07.001
8. von Sivers I, Künzner F, Köster G (2016) Pedestrian evacuation simulation with separated families. In: Proceedings of the 8th international conference on pedestrian and evacuation dynamics (PED2016)
9. Crowd simulation team at Munich University of Applied Sciences: openVADERE Simulation Framework (2016). www.vadere.org. Accessed 13 June 2019
10. Barnes M, Abel IG, Dorland W, Görler T, Hammett GW, Jenko F (2010) Direct multiscale coupling of a transport code to gyrokinetic turbulence codes. Phys Plasmas. https://doi.org/10.1063/1.3323082
11. Farcaş IG, Görler T, Bungart HJ, Jenko F, Neckel T (2018) Sensitivity-driven adaptive sparse stochastic approximations in plasma microinstability analysis. arXiv e-prints arXiv:1812.00080
12. Peherstorfer B, Willcox K (2015) Dynamic data-driven reduced-order models. Comput Methods Appl Mech Eng 291:21–41. https://doi.org/10.1016/j.cma.2015.03.018
13. Peherstorfer B, Willcox K (2016) Dynamic data-driven model reduction: adapting reduced models from incomplete data. Adv Model Simul Eng Sci 3(1):11. https://doi.org/10.1186/s40323-016-0064-x
14. Dietrich F, Künzner F, Neckel T, Köster G, Bungartz HJ (2018) Fast and flexible uncertainty quantification through a data-driven surrogate model. Int J Uncertain Quantif 8(2):175–192
15. Peherstorfer B, Willcox K, Gunzburger M (2018) Survey of multifidelity methods in uncertainty propagation, inference, and optimization. SIAM Rev 60(3):550–591. https://doi.org/10.1137/16M1082469
16. Schilders WHA, van der Vorst HA, Rommes J (2008) Model order reduction: theory, research aspects and applications. Springer, Berlin. https://doi.org/10.1007/978-3-540-78841-6
17. Franzelin F, Pflüger D (2016) From data to uncertainty: an efficient integrated data-driven sparse grid approach to propagate uncertainty. In: Garcke J, Pflüger D (eds) Sparse grids and applications—Stuttgart 2014. Springer, Berlin, pp 29–49
18. Franzelin F, Diehl P, Pflüger D (2015) Non-intrusive uncertainty quantification with sparse grids for multivariate peridynamic simulations. In: Griebel M, Schweitzer MA (eds) Meshfree methods for partial differential equations VII. Springer, Berlin, pp 115–143
19. Winokur J, Kim D, Bisetti F, Le Maître OP, Knio OM (2016) Sparse pseudo spectral projection methods with directional adaptation for uncertainty quantification. J Sci Comput 68(2):596–623. https://doi.org/10.1007/s10915-015-0153-x
20. SLURM: Slurm. https://github.com/SchedMD/slurm. Accessed 13 June 2019
21. Phipps E, D'Elia M, Edwards HC, Hoemmen M, Hu J, Rajamanickam S (2017) Embedded ensemble propagation for improving performance, portability, and scalability of uncertainty quantification on emerging computational architectures. SIAM J Sci Comput 39(2):162–193. https://doi.org/10.1137/15M1044679
22. D'Elia M, Phipps E, Rushdi A, Ebeida M (2017) Surrogate-based Ensemble Grouping Strategies for Embedded Sampling-based Uncertainty Quantification. arXiv e-prints
23. Xiu D, Karniadakis GE (2002) The Wiener–Askey polynomial chaos for stochastic differential equations. SIAM J Sci Comput 24(2):619–644. https://doi.org/10.1137/S1064827501387826
24. Xiu D (2009) Fast numerical methods for stochastic computations: a review. Commun Comput Phys 5(2):242–272
25. Feinberg J, Langtangen HP (2015) Chaospy: an open source tool for designing methods of uncertainty quantification. J Comput Sci 11:46–57. https://doi.org/10.1016/j.jocs.2015.08.008
26. Feinberg J. Chaospy. https://github.com/jonathf/chaospy. Accessed 13 June 2019
27. Seitz MJ, Köster G (2012) Natural discretization of pedestrian movement in continuous space. Phys Rev E 86(4):046108. https://doi.org/10.1103/PhysRevE.86.046108
28. Seitz MJ, Köster G (2014) How update schemes influence crowd simulations. J Stat Mech Theory Exp 2014(7):P07002. https://doi.org/10.1088/1742-5468/2014/07/P07002
29. Seitz MJ, Dietrich F, Köster G (2015) The effect of stepping on pedestrian trajectories. Phys A Stat Mech Appl 421:594–604. https://doi.org/10.1016/j.physa.2014.11.064
30. von Sivers I, Köster G (2015) Dynamic stride length adaptation according to utility and personal space. Transp Res Part B Methodol 74:104–117. https://doi.org/10.1016/j.trb.2015.01.009
31. Leach J (2004) Why people freeze in an emergency: temporal and cognitive constraints on survival responses. Aviat Space Environ Med 75(6):539–542
32. Schreiber M, Riesinger C, Neckel T, Bungartz HJ, Breuer A (2015) Invasive compute balancing for applications with shared and hybrid parallelization. Int J Parallel Program 43(6):1004–1027. https://doi.org/10.1007/s10766-014-0336-3
33. Hamscher V, Schwiegelshohn U, Streit A, Yahyapour R (2000) Evaluation of job-scheduling strategies for grid computing. In: Buyya R, Baker M (eds) Grid computing—GRID 2000. Springer, Berlin, pp 191–202
34. Message Passing Interface Forum: MPI: a message-passing interface standard, version 3.1. Specification (2015). https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf. Accessed 13 June 2019
35. Dalcín L, Paz R, Storti M (2005) MPI for python. J Parallel Distrib Comput 65(9):1108–1115. https://doi.org/10.1016/j.jpdc.2005.03.010
36. Varoquaux G. joblib. https://github.com/joblib/joblib. Accessed 13 June 2019
37. Dalcín L. mpi4py. https://pypi.org/project/mpi4py. Accessed 13 June 2019
38. Gerstner T, Griebel M (2003) Dimension-adaptive tensor-product quadrature. Computing 71(1):65–87. https://doi.org/10.1007/s00607-003-0015-5
39. Coffman EG, Garey MR, Johnson DS (1978) An application of bin-packing to multiprocessor scheduling. SIAM J Comput 7(1):1–17. https://doi.org/10.1137/0207001

40. Pinedo ML (2016) Scheduling: theory, algorithms, and systems. Springer, Berlin. https://doi.org/10.1007/978-3-319-26580-3
41. Kunde M (1982) A multifit algorithm for uniform multiprocessor scheduling. In: Cremers A, Kriegel HP (eds) Theoretical computer science. Springer, Berlin, pp 175–185
42. Linux-Cluster of Leibniz Supercomputing Centre. https://www.lrz.de/services/compute/linux-cluster. Accessed 13 June 2019
43. Ganapathysubramanian B, Zabaras N (2007) Sparse grid collocation schemes for stochastic natural convection problems. J Comput Phys 225(1):652–685. https://doi.org/10.1016/j.jcp.2006.12.014
44. Leibniz Supercomputing Centre. https://www.lrz.de. Accessed 13 June 2019