A Parallel and Distributed Surrogate Model Implementation for Computational Steering

Daniel Butnaru* butnaru@in.tum.de

Gerrit Buse* buse@in.tum.de

Dirk Pflüger* pflueged@in.tum.de

* Institut für Informatik, Technische Universität München, Germany

***Abstract*—Understanding the influence of multiple parameters in a complex simulation setting is a difficult task. In the ideal case, the scientist can freely steer such a simulation and is immediately presented with the results for a certain configuration of the input parameters. Such an exploration process is however not possible if the simulation is computationally too expensive. For these cases we present in this paper a scalable computational steering approach utilizing a fast surrogate model as substitute for the time-consuming simulation. The surrogate model we propose is based on the sparse grid technique, and we identify the main computational tasks associated with its evaluation and its extension. We further show how distributed data management combined with the specific use of accelerators allows us to approximate and deliver simulation results to a high-resolution visualization system in real-time. This significantly enhances the steering workflow and facilitates the interactive exploration of large datasets.**

***Keywords*-computational steering; surrogate models; sparse grids; parallelization; real-time systems**

I. Introduction

Interpreting simulations and their dependency on input parameters is getting more and more difficult with their ever-increasing complexity and overall computational demands of single simulation runs. Recent developments concerning computational power and faster visualization algorithms have made this task more feasible for certain simulation types, as results can now be computed and visualized in real-time. For such simulations, the field of computational steering—usually associated with visual exploration—has offered tools, methodologies and visualization components facilitating a convenient *configure-start-analyze* approach. Ways of influencing the run of a simulation, direct manipulation of parameters, and visual indicators for exploration objectives have also been addressed in the computational steering context.

Other types of simulations such as Navier-Stokes-based computational fluid dynamics (CFD) scenarios or scenarios in computational mechanics are still only steerable in real-time, if the underlying mathematical model is simplified or a coarse discretization is used. For this category of complex and costly simulations, interactive visual exploration (by means of parameter steering) is only possible through surrogate models. Surrogates are approximate representations of the original simulation, but results at arbitrary points in the simulation's parameter space can be obtained significantly faster.

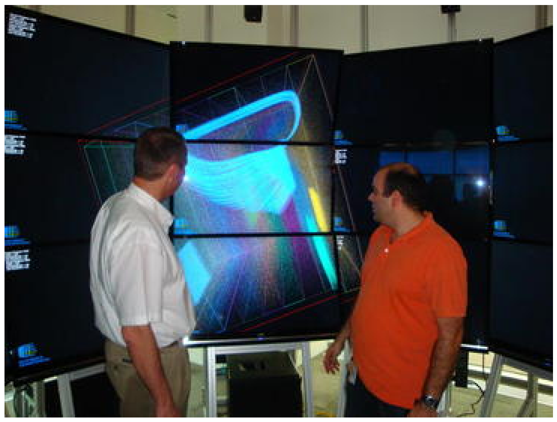


Fig. 1. Researchers investigate a CFD scenario in an interactive and visual manner. In this multi-display multi-node visualization system each display connects to a computing node and renders part of the view frustum.

Surrogate models are most predominant and have shown good results in the area of optimization [1]. In such a setting an objective function is replaced by a cheap surrogate, which is then evaluated repeatedly during an iterative process, until a certain optimality criterion is fulfilled. Research in this area therefore usually focuses on the accuracy of the constructed surrogate.

In this paper however, we employ surrogates to interactively explore a complex simulation in a distributed visualization system (see Fig. 1). We focus on a particular surrogate model based on *sparse grids*, a numerical technique suitable for the approximation of high-dimensional functions. We quantify the performance requirements of a computational steering environment on the basis of the *response time*, and in the same way we identify the computational demands on our surrogate model. We propose a parallel and distributed solution for our surrogate, which guarantees data delivery at interactive rates and with sufficiently high resolution for our target visualization setting. The exploration process is thus significantly enhanced, as a simulation scientist can actually explore the multi-dimensional parameter space by varying parameters and immediately observe the effects on the large screens of a multi-display visualization environment (see Fig. 1).

II. Related Work

Surrogate models for computational steering have also been proposed in [2] with the focus on user steerable optimization and not visual exploration. Surrogate models, also known as meta-models, have been classified in data-fit, multifidelity and reduced-order [3]. Multifidelity models are based on the original physics but perform simplifications (coarser discretization, relaxed solver tolerances, simplified physics) resulting in a lower fidelity model. Reduced-order models are also physics-based and use a reduced basis (eigenmodes of model analysis or singular vectors for POD [4]) and do a projection of the original high-dimensional system onto this much smaller function space. Both multifidelity and reduced order methods are well suited for optimization tasks [5], however, they are more complex to construct and the operations needed to evaluate such models are not easily mapped to hardware for efficient high-resolution and interactive explorations. Sparse grids, as data fit models, do non-physics-based approximation by multi-dimensional piecewise d-linear interpolation, and, while generally not as accurate as the previous methods, they are non-intrusive (no need to modify or know the underlying simulation) and can be formulated for an efficient parallel execution. This motivates their choice for fast surrogate-based visual exploration. In [6], sparse grids have been used as surrogate models, but, again, for optimization purposes and not for visualization.

III. Surrogate Models For Computational Steering

?? The task of understanding the global behavior of a simulation which depends on more than one parameter can easily become quite tedious when faced with long execution times. In a computational steering environment, such a simulation could be started for several parameter combinations in advance, and a user could then draw conclusions based on the results. A surrogate model is used in place of the simulation to enable real-time evaluation, obtaining approximate solutions significantly faster than running the original simulation. This is achieved by investing computational effort in building the surrogate model *offline*, which can then be evaluated very fast during the actual *online* steering. We shall denote with $u(x, \mu)$, $u : \Omega \times \mathcal{P} \to \Gamma$, the field variable we seek to approximate as a function of the physical 3D coordinates $x \in \Omega$ and the vector $\mu \in \mathcal{P}$ of d parameters, where

$$\mathcal{P} = \{\mu \mid \mu \in (0,1)^d \ , \ d \geq 1\}. \tag{1}$$

For a fixed μ the field variable is called a snapshot $u^\mu(x) := u(x, \mu)$ and represents the result of the simulation started with μ as parameter values. The output of the surrogate model $\widetilde{u}(x, \mu)$ is also called a snapshot and will be denoted $\widetilde{u}^\mu(x) := \widetilde{u}(x, \mu)$.

Fig. 2 illustrates the main parts of a surrogate model approach. The construction of the surrogate model is done offline and starts by choosing the parameters and their ranges of interest. This is simulation specific but is also limited by the amount of effort worth investing in the offline phase, as a larger parameter count and range, while desirable, will increase the requirements of the online phase. Next, a sampling is performed within the chosen ranges, and, for each parameter combination, a full simulation is performed and stored. We will denote the set of sampling points $\mathcal{P}_s \subset \mathcal{P}$.

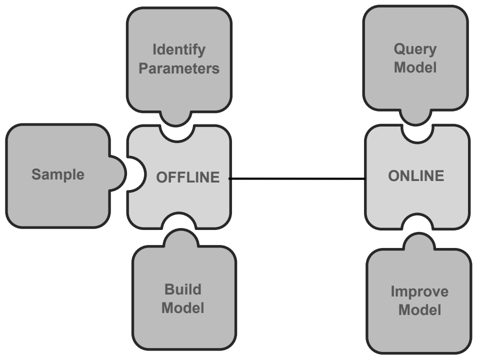


Fig. 2. The two phases of the surrogate process.

For the chosen sampling $\mathcal{P}_s$ all the corresponding simulation snapshots are computed. The offline phase ends with a surrogate-model-specific model reduction technique, reducing the initial high-dimensional problem u to the reduced form $\widetilde{u}$, which is more suitable for fast repeated evaluation. The actual exploration takes place in the online phase, where the user can frequently change parameters, and, guided by an interactive visualization, study the behavior of the original simulation.

In this paper we investigate the requirements of a sparse grid-based surrogate model and present an efficient implementation.

IV. Sparse Grids as Surrogate Models

In previous work [7], we first proposed to use sparse grids as a surrogate model for high-resolution computational steering. They are non-intrusive because the underlying simulation is treated as a black box which delivers snapshots for the requested parameter combinations. Here, we extend our approach to a parallel and distributed processing of the surrogate model. In this section we give an overview of sparse grids and their properties as a surrogate model.

A. Basics of Sparse Grids

Sparse grids help to overcome the curse of dimensionality to a great extent. Interpolating a d-dimensional function u on a regular grid with a resolution of M grid points in each dimension, they enable one to reduce the number of grid points significantly from $\mathcal{O}(M^d)$ to $\mathcal{O}(M(\log M)^{d-1})$ while maintaining a similar accuracy as in the full grid case—at least if u is sufficiently smooth [8]. This typically even holds for functions that do not meet the smoothness requirements if adaptive refinement is employed [9]. The notion *sparse grids* was coined in 1990 for the solution of high-dimensional partial differential equations (PDEs) [10], and they have meanwhile been successfully employed in a whole range of applications, ranging from astrophysics and quantum chemistry to data mining and computational finance, see, e.g., [8], [9] and the references cited there. In the following, we briefly describe sparse grids and the main principles they base upon, a hierarchical representation of the one-dimensional basis and the extension to the d-dimensional setting via a tensor product approach. For further details, we refer to [8].

We consider the representation of a piecewise d-linear function $\tilde{u} : \Omega \to \Gamma$ for a certain mesh-width $h_n := 2^{-n}$ with some discretization level n. We consider rectangular domains Ω which we scale to $\Omega := [0,1]^d$. To obtain an interpolant $\tilde{u}$ as an approximation to some function u, we discretize Ω and employ basis functions ϕ_i which are centered at the grid points stemming from the discretization. $\tilde{u}$ is thus a weighted sum of N basis functions, $\tilde{u} := \sum_{j=1}^{N} v_j \phi_j$, with coefficients v_j.

The underlying principle is a hierarchical formulation of the basis functions. In one dimension, we use the standard hierarchical basis

$$\Phi_l := \left\{\varphi_{l',i} : l' \leq l, i \leq 2^{l'} - 1 \wedge i \text{ odd}\right\}.$$

with piecewise linear ansatz functions $\varphi_{l,i}(\mu) := \varphi\left(\mu \cdot 2^l - i\right)$ and $\varphi(\mu) := \max(1 - |\mu|, 0)$ for some level $l \geq 1$ and an index $1 \leq i < 2^l$. The basis functions are centered at grid points $\mu_{l,i} = 2^{-l} i$ at which we interpolate u, see Figure 3 (top left) for the basis functions up to level 3. Note that all basis functions on one level have pairwise disjoint supports and cover the whole domain.

The hierarchical basis functions can be extended to d dimensions via a tensor product approach as

$$\varphi_{\underline{l},\underline{i}}(\mu) := \prod_{j=1}^{d} \varphi_{l_j,i_j}(\mu_j),$$

with multi-indices $\underline{l}$ and $\underline{i}$ indicating level and index of the underlying one-dimensional hat functions for each dimension. The d-dimensional basis

$$\Phi_{W_{\underline{l}}} := \left\{\varphi_{\underline{l},\underline{i}}(\mu) : \; i_j = 1, \ldots, 2^{l_j} - 1, \; i_j \text{ odd}, \; 1 \leq j \leq d\right\}$$

span hierarchical subspaces $W_{\underline{l}}$. As before, the basis functions for each $W_{\underline{l}}$ have pairwise disjoint, equally sized supports and cover the whole domain. The classical full-grid space of piecewise d-linear functions V_n can be obtained as a direct sum of $W_{\underline{l}}$,

$$V_n := \sum_{l_1=1}^{n} \cdots \sum_{l_d=1}^{n} W_{(l_1,\ldots,l_d)} = \bigoplus_{|\underline{l}|_\infty \leq n} W_{\underline{l}},$$

but the hierarchical scheme of subspaces allows one to choose those subspaces that contribute most to the approximation. By choosing subspaces with respect to their contribution in the L_2-norm, this leads to the sparse grid space $V_n^{(1)}$,

$$V_n^{(1)} := \bigoplus_{|\underline{l}|_1 \leq n+d-1} W_{\underline{l}}.$$

The tableau of subspaces in 2D is shown in Figure 3 (bottom) for $n = 3$.

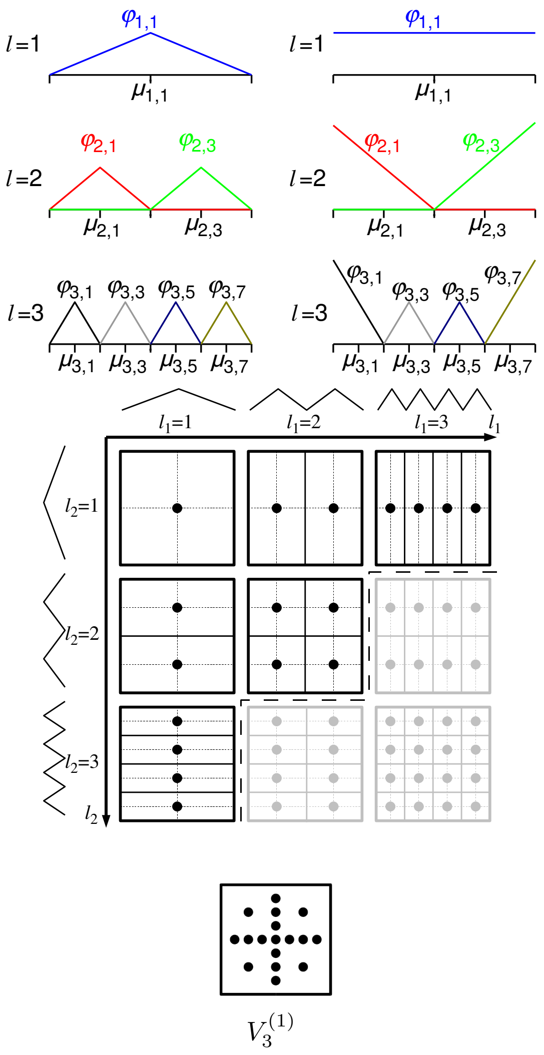


Fig. 3. Classical one-dimensional hierarchical basis functions up to level 3 without boundary basis functions (top left), their modified, extrapolating counterparts (top right), and the tableau of subspaces $W_{\underline{l}}$ up to level 3 in two dimensions (center) together with the resulting sparse grid for $n = 3$ (bottom).

To obtain non-zero values on the boundary, the one-dimensional basis of level 1 can be extended by the two basis functions $\varphi_{0,0}$ and $\varphi_{0,1}$. Unfortunately, even for a very coarse grid with a resolution of $h_1 = 1/2$ this requires to obtain 3^d simulation results—with $3^d - 1$ parameter combinations being located on the boundary of the parameter space Ω. For our application of computational steering, we assume that we start with a reasonable choice of Ω and that these extreme parameter combinations are of less interest than the inner part of Ω. We therefore choose to interpolate only in the inner part and to extrapolate towards the boundary, and use in the following the one-dimensional basis functions depicted in Figure 3 (top right).

B. Construction and Use

Any multi-variate function u can be interpolated on a sparse grid, but the resulting hierarchical and possibly adaptive

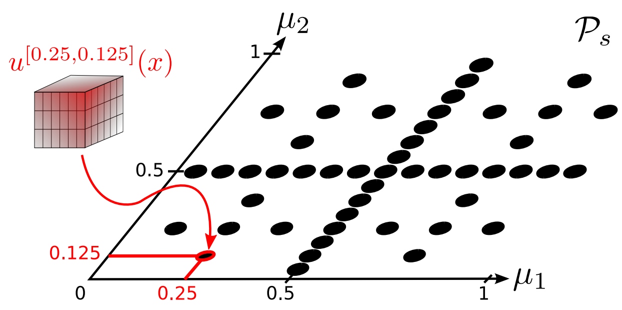


Fig. 4. Sparse grid sampling $\mathcal{P}_s$ for a two-dimensional parameter space. The point coordinates denote the (normalized) parameter combinations for which simulations need to be performed and stored.

structure makes it not suitable for real-time capable interpolation. To enable efficiency, we distinguish the treatment of the parameters in x, the spatial coordinates representing the discretization grid, and μ, the parameters of interest changed during exploration. The sparse grid thus discretizes only the parameter space which has non-regular structure but which, due to its rather small size, can be handled efficiently. The evaluation has to interpolate the whole spatial resolution of the snapshots. Those are stored as large data blocks with a regular structure (compact vectors), which are supported by visualization hardware and which can be dealt with efficiently. This two level approach allows the combination of expensive adaptivity in the parameter space but on a fast small data structure with evaluation involving large snapshots but with high data regularity, enabling ultimately their use as surrogates for interactive steering. The area we most focus our attention on are time-dependent CFD simulations. Such simulations exhibit all the properties that make them ill-suited for a fast direct exploration: each simulation run can easily be very costly and can require many time-steps. Furthermore, the amount of generated data is typically very large, consisting of a 3D vector field for the velocities plus the pressure field. The interesting parameters for steering are fluid parameters (inlet velocities, viscosity, etc.) and time. We discretize them with a sparse grid and leave the space coordinates as full 3D vectors.

For sparse grids, the parameter sampling done in the offline phase is prescribed by the location of the N sparse grid points in the multi-dimensional parameter space $\mathcal{P}$ (see Fig. 4), i.e,

$$\mathcal{P}_s = \{ \ \mu_j \ \}_{j=1,\ldots,N} \, , \tag{2}$$

where the μ_j stem from the sparse grid discretization of the parameter space $\mathcal{P}$. Let $\mathcal{H}^{\mu_j} \subseteq \mathcal{P}_s$ be the set of hierarchical ancestors of sampling point μ_j, including μ_j itself,

$$\mathcal{H}^{\mu_j} = \{ \ \mu_j \ \} \cup \{ \ \mu_k \mid \mu_k \text{ is hier. ancestor of } \mu_j \ \} \, . \tag{3}$$

The sampling $\mathcal{P}_s$ starts with a regular sparse grid and is incrementally refined based on a chosen criterion. For each μ_j, the snapshot has to be obtained offline.

The reduction step that is specific to our sparse grid surrogate model and which concludes the offline phase is called *hierarchization*. It denotes the transformation of all sample snapshots $u^{\mu_j}(x)$, $\mu_j \in \mathcal{P}_s$ into their representation as hierarchical increments $v_j(x) = v^{\mu_j}(x)$, i.e, the surpluses. Hierarchization is a necessary step before the model $\widetilde{u}(x,\mu)$ can be evaluated. The surplus $v_j(x)$ depends on the surpluses of all hierarchical ancestors μ_k in $\mathcal{H}^{\mu_j}$ as

$$v_j(x) := u^{\mu_j}(x) - \sum_{\mu_k \in \mathcal{H}^{\mu_j} \setminus \{\mu_j\}} v_k(x) \cdot \varphi_k(\mu) \, , \tag{4}$$

which corresponds to subtracting the sparse grid interpolation on the next coarser level from the current snapshot.

Once the surrogate $\widetilde{u}$ is constructed, the online phase can start. It can be evaluated on the entire parameter space $\mathcal{P}$ by using information from the set of sampling points $\mathcal{P}_s$. The evaluation at point $\mu \in \mathcal{P}$ is obtained as a linear combination of the N weighted basis functions corresponding to the N grid points,

$$\widetilde{u}(x,\mu) := \sum_{j=1}^{N} v_j(x) \cdot \phi_j(\mu) \, . \tag{5}$$

In order to implement this formula three central aspects need to be considered:

1. Identify affected basis functions: Sparse grid ansatz functions ϕ_j do not have uniformly shaped supports. For an example, see how the support size varies in the illustrations in Fig. 3. Also, only few ϕ_j from different subspaces or levels may have overlapping supports. Thus, many basis functions will not be affected by a certain evaluation. Evaluating the surrogate model at point μ therefore requires in the first step to identify the set $\mathcal{A}^\mu$ of affected basis functions with respect to evaluation point μ,

$$\mathcal{A}^\mu := \{\phi_j \mid j = 1, \ldots, N \wedge \phi_j(\mu) \neq 0\} \, . \tag{6}$$

Fig. 5 qualitatively shows which snapshots (smaller cubes) need to be collected to interpolate a new snapshot at the evaluation point marked with a triangle. Note that working with the full set of basis functions (or grid points, respectively) instead of $\mathcal{A}^\mu$ is not an option as typically $N \gg |\mathcal{A}^\mu|$ which would result in a lot more effort in the next step.

2. Combine weighted surpluses: : This step can and should be completely separated from the first step of identifying the data dependencies. It is the critical operation of the online phase, and its performance is crucial to a smooth user experience. Based on the output of the previous step, efficient gather and reduce strategies need to be implemented to compute the sum (5).

3. Improve the model: Last but not least, the set $\mathcal{P}_s$ of sampling points can be extended. The explorative nature of our setting demands an extensible approach, allowing for incremental improvement of the surrogate model's approximation quality during the online phase. If the initial sampling of the parameter space does not capture the features of the underlying simulation function u, automatic or user driven refinement helps to refine the model's "database" $\mathcal{P}_s$. The acquisition and integration of new simulation data however relies on steps from the offline phase: The reduction operation *hierarchization* needs to be performed for each new simulation

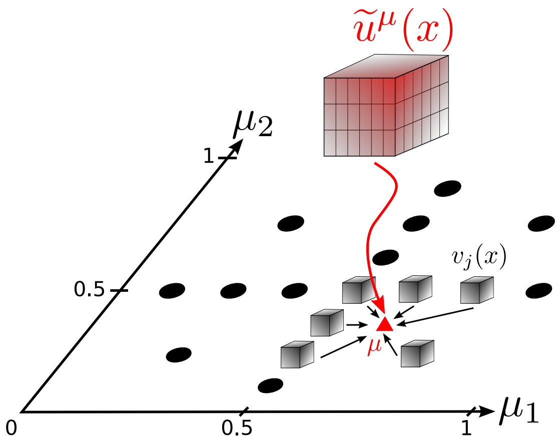
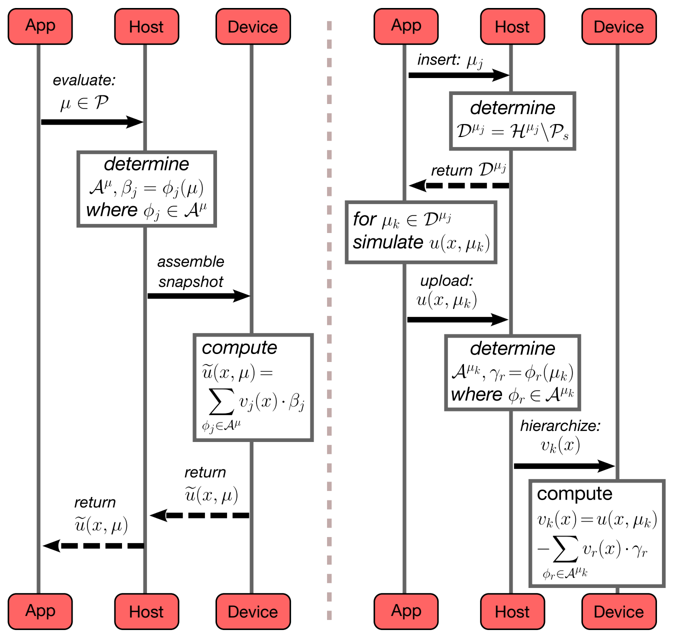


Fig. 5. An example for a sparse grid interpolation: the value $\widetilde{u}^{\mu}(x)$ of the approximated snapshot for the (triangle marked) parameter combination μ of interest is constructed as a sum of the weighted hierarchical coefficients $v_j(x)$, marked with cubes.



App
Host
Device
evaluate: $\mu \in \mathcal{P}$
determine $\mathcal{A}^{\mu}, \beta_j = \phi_j(\mu)$ where $\phi_j \in \mathcal{A}^{\mu}$
assemble snapshot
compute $\widetilde{u}(x,\mu) = \sum_{\phi_j \in \mathcal{A}^{\mu}} v_j(x) \cdot \beta_j$
return $\widetilde{u}(x,\mu)$
return $\widetilde{u}(x,\mu)$
App
Host
Device
App
Host
Device
insert: μ_j
determine $\mathcal{D}^{\mu_j} = \mathcal{H}^{\mu_j} \setminus \mathcal{P}_s$
return $\mathcal{D}^{\mu_j}$
for $\mu_k \in \mathcal{D}^{\mu_j}$ simulate $u(x,\mu_k)$
upload: $u(x,\mu_k)$
determine $\mathcal{A}^{\mu_k}, \gamma_r = \phi_r(\mu_k)$ where $\phi_r \in \mathcal{A}^{\mu_k}$
hierarchize: $v_k(x)$
compute $v_k(x) = u(x,\mu_k) - \sum_{\phi_r \in \mathcal{A}^{\mu_k}} v_r(x) \cdot \gamma_r$
App
Host
Device

Fig. 6. Sequence diagrams for the evaluation of the surrogate model at point $\mu \in \mathcal{P}$ (left), and the extension of the model by a new snapshot $u(x, \mu_j)$, where μ_j is a valid sparse grid sampling point with $\mu_j \notin \mathcal{P}_s$ (right).

result $u^{\mu_j}(x)$ by applying (4). This calls for a hybrid solution that transparently mixes parts of the two phases without interfering with the user experience.

C. Response Time

The optimization of all three steps has to be guided by the performance requirements of the computational steering environment, usually specified by means of system response time. Responses of a system to user action serve as continuity of the exploration process. Depending on the task at hand, different kinds of responses and response delays are acceptable psychologically [11]. For the visual computational steering we consider two main types of exploration tasks. In a *parameter sweep*, the user continuously increments a single parameter being interested in its influence on the behavior of the simulation. A response time of no more than 0.2 seconds was found to be suitable for such user continuous actions [11]. The task of *parameter comparison* involves to switch between two or more parameter combinations and to interprete the differences. For this task, a response time of up to 2 seconds would still allow for an uninterrupted thought process. In the next section, we present the design of a surrogate model realization which fulfills these requirements.

V. System Architecture

The goal we set for surrogate-model-based computational steering is a system which can be easily connected to a distributed visualization environment and delivers results at interactive rates. At the same time, it can be updated with new snapshots on user request during exploration.

A. CPU/GPU Repository

The sparse grid repository is the center piece of the exploration workflow. It is separated into a CPU and a GPU component. The CPU component implements the logic of the surrogate model. The GPU component is responsible for data storage as well as fast data delivery when the model is evaluated. For ease of reference we refer to the components as *device* and *host*, a common terminology in GPU computing.

CPU component (host): The host acts as the frontend of the repository towards the application side. Algorithms for adaptive sparse grids are typically sophisticated and need to be executed on a flexible CPU. Therefore the host maintains the complex structure of the sparse grid surrogate model and delegates tasks to the device, its dedicated worker.

GPU component (device): The device is the backend of the repository, responsible for data and compute intensive tasks. Nearly no knowledge about the surrogate model is necessary to perform the tasks given by the host, which are always related to (4) and (5). In these equations long vectors (the snapshots $u^{\mu_j}(x)$ and surpluses $v_j(x)$) are added or subtracted – the perfect task for a processor with high memory bandwidth. Nvidia's Tesla C2070 offering a specified bandwidth of 144 GB/s is therefore our choice for the backend in charge of these operations. In comparison, the theoretical bandwidth of Intel's Westmere processor (as used in the system) is specified as 32 GB/s, however only around 21 GB/s can actually be achieved (measured with STREAM benchmark [12], [13]).

In the following, the interfaces between device, host and application are explained for the two scenarios in which the repository is accessed (also see Fig. 6 and the functionality specified in Sec. IV-B):

1) The surrogate model is evaluated at position $\mu \in \mathcal{P}$. The host determines the set of affected basis functions $\mathcal{A}^{\mu}$, evaluates them and passes the resulting values $\beta_j := \phi_j(\mu), \phi_j \in \mathcal{A}^{\mu}$ to the device. The interpolated snapshot $\widetilde{u}^{\mu}(x)$ is efficiently assembled on the GPU, from (5).
2) The surrogate model is extended by the inclusion of a

new snapshot $u^{\mu_j}(x)$, where $\mu_j \notin \mathcal{P}_s$ is a valid new sparse grid sampling point. The host first determines all unfulfilled hierarchical dependencies $\mathcal{D}^{\mu_j} = \mathcal{H}^{\mu_j} \backslash \mathcal{P}_s$, i.e., the hierarchical ancestors of μ_j missing in $\mathcal{P}_s$ which are required to ensure a valid grid structure. Once the snapshots for the $\mu_k \in \mathcal{D}^{\mu_j}$ are computed by the application, the host prepares the hierarchization of the new snapshots $u^{\mu_k}(x)$ by computing the respective weights $\gamma_r = \phi_r(\mu_k)$. With the γ_r, the device can efficiently perform hierarchization (4) and integrate the surpluses $v_k(x)$ into the model.

B. Distributed Snapshots and CPU Involvement

The presented repository uses GPUs to deliver fast results to the visualization, but it is limited by the amount of snapshots it can actually store. A current high-end GPU featuring 6 GB of memory can store and combine up to 200 snapshots of size 128^3. Depending on the dimensionality d of the parameter space, this might not be enough to ensure a good approximation quality. To mitigate this limitation, we use the combined memory of all available GPUs to distribute the repository and thus store more snapshots. On each node n we construct the surrogate $\widetilde{u}^{\mu}(x)$ for a slice of the entire domain Ω. Such a distribution allows for a flexible and load balanced computation of the final snapshot but requires a gather operation in order to assemble the full snapshot for visualization.

To allow for even larger snapshot sizes and/or number of snapshots, the CPU can be additionally involved in the collection of affected contributions. The solution we propose is a two step evaluation where first a fast solution is obtained using the contributions mostly on the GPU while in a second step, the remaining contributions are delivered from the CPU.

C. Visualization Requirements and Interface

In our steering environment, the user chooses a parameter μ and is presented in real-time with the visualization of the corresponding full snapshot $\widetilde{u}^{\mu}(x)$. As presented above, the repository is distributed among the visualization nodes in order to utilize all the available GPUs, meaning that each node produces only a slice of the final snapshot. We then use an *all-gather* operation to collect all slices at all visualization nodes. Each of them then renders only the section of the final image corresponding to their view frustum. An explicit *all-gather* operation can be skipped if the visualization environment implements distributed rendering algorithms.

We make use of the described data delivery and identify the minimal interface exposed by the surrogate model to any visualization software:

init(): distribute and load the repository,
evaluate(μ, buffer): place the full snapshot u^{μ} in the visualization buffer via all-gather,
improve(μ): trigger several new simulation runs in order to improve the accuracy around μ.

The presented interface treats the repository as a pure data deliverer which stores no information whatsoever about the simulation scenario or its geometry. This allows for loose coupling between components.

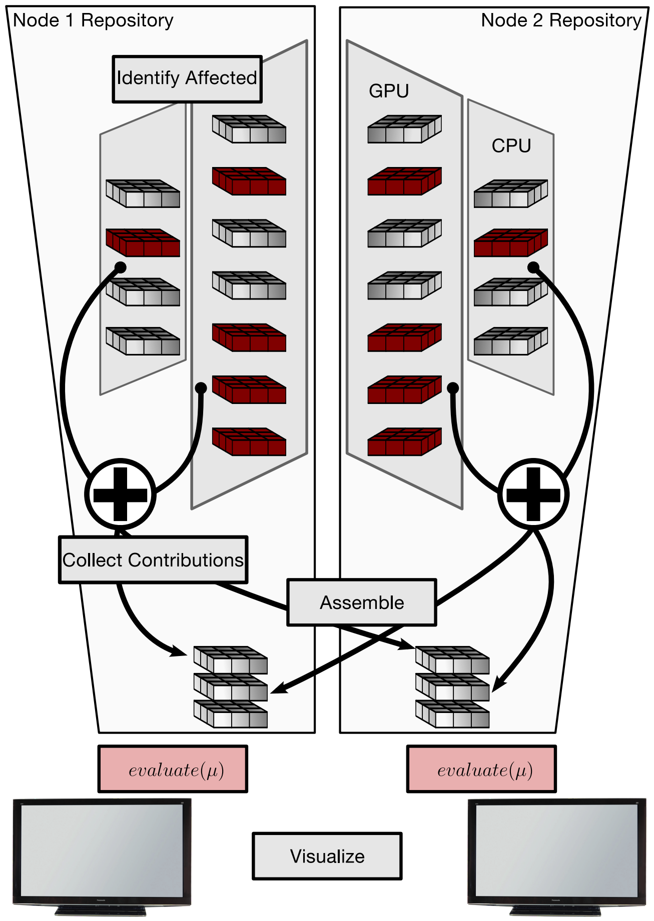


Fig. 7. CPU/GPU repository structure with visualization interface. Hierarchized snapshots v_j are distributed among nodes and stored in GPU and main memory. Each repository evaluates a part of the final snapshot, gathers the rest from the other nodes, and assembles the whole snapshot.

VI. Results

All the measurements presented in this section have been obtained on our FRAVE (Fully Reconfigurable CAVE Environment) system (see Fig. 8)[1]. The FRAVE is a multi-display semi-immersive visualization system organized as a collection of building blocks which enable it to be folded, extended or split up to accommodate a specific type of visualization goal.

A single building block consists of the following components:

- *Displays*: one or two 3D full HD (1920x1080) 65“ plasma screens Panasonic TX-P65VT20E
- *Graphic cards*: an Nvidia QuadroPlex 7000 for graphics (2x 6 GB RAM) and an additional Nvidia Tesla C2070 card (6 GB RAM) for computing purposes

[1]www.mac.tum.de/wiki/index.php/FRAVE

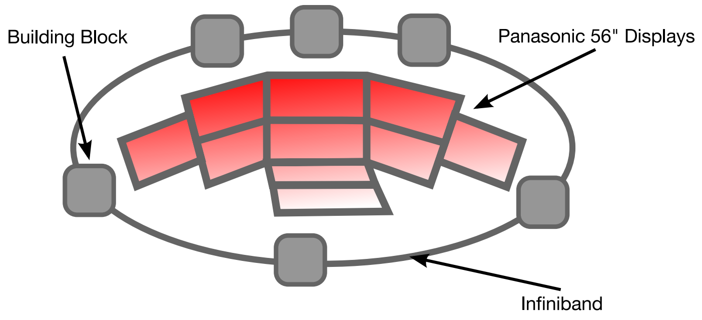


Fig. 8. The FRAVE consists of a series of building blocks connected by Infiniband. The current FRAVE setup has 6 building blocks powering 10 displays (8 for the walls, 2 for the floor).

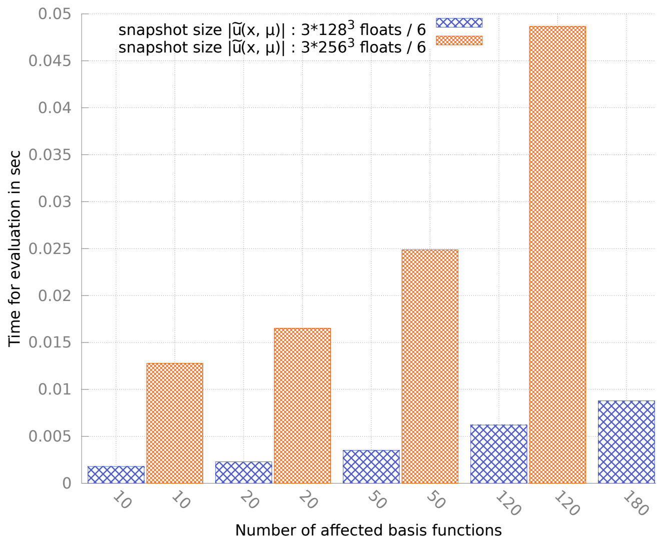


Fig. 9. Time needed locally for evaluation in two different scenarios. $|\mathcal{A}^{\mu}|$ is varied on the horizontal axis to account for different sets $\mathcal{P}_s$.

- *Computer*: a dual-socket Intel Xeon E5630 quad-core system (2.53 GHz) with 24 GB RAM and 8 TB hard drive
- *Frame:* a light aluminum frame on which all the above components are mounted and which can be moved freely

Each building block can be added to the system by connecting it to the Infiniband network, where it is needed.

A. GPU-Based Evaluation and Hierarchization

In this section, evaluation is benchmarked as a local operation. We hereby assume that the repository is running on 6 building blocks, and that the set of all snapshots is equally distributed to the 6 instances of the repository. Triggering evaluation on a node then implies that only one sixth of a snapshot has to be interpolated. We examine the performance of this operation in two scenarios:

- 200 snapshots, each consisting of $3 \cdot 128^3$ floats (24 MB), total amount of data ≈ 4.8 GB, and
- 150 snapshots, each consisting of $3 \cdot 256^3$ floats (192 MB), total amount of data ≈ 28.1 GB.

Even though the first scenario is small enough to be treated locally on each node, there would be almost no storage space left to extend the model. In the second test scenario the repository is put to test for full load. Figure 9 shows for both scenarios and varying number of affected basis functions how long evaluation takes on a single node. For the assumed number of snapshots, $|\mathcal{A}^{\mu}|$ is unlikely to exceed 50 in our scenario. Still, the chart shows that the repository would be able to deliver the partial snapshots within the response time, even for a larger number of affected basis functions.

It is not necessary to benchmark the incremental hierarchization in the same way, as the time needed for this operation during the online phase can be approximated. Hierarchization basically corresponds to a number of evaluations executed successively, as sketched in Fig. 6. Multiplying the time for evaluation by k should thus give an estimate of the time needed to add k new snapshots to the surrogate model.

B. Data Assembly

As illustrated in Fig. 7, each node delivers only a slice of a full snapshot $\widetilde{u}^{\mu}(x)$ to the visualization. However, the visualization requires the full snapshot locally in order to render, and a gather step has to be performed. Figure 10 presents the cost of such a gather operation for different node counts and snapshot sizes on the FRAVE's Infiniband network. We notice that the assembly time scales linearly with snapshot size while being almost constant with respect to the number of nodes involved. We do not expect this to be the case for a much higher number of nodes. But for visualization systems like the FRAVE, which consist of a moderate number of nodes, it can be assumed.

While the evaluation cost allows for a very good response time, the assembly costs for snapshots larger than 3×256^3 prevent an interactive parameter sweep ($\leq 0.2\,s$). A parameter comparison is however still possible ($\leq\ 2\,s$). It is worth mentioning that such snapshot sizes are challenging even for classical visualizations and require special approaches to visualize in an interactive manner. The distributed data availability might actually be more appropriate for such visualization algorithms.

VII. Conclusion

In this paper we analyzed the computational operations of a sparse grid surrogate model. We conclude that the main computational effort of the online phase incurs in the evaluation, a summation of hierarchically weighted vector fields. A first efficient implementation for the evaluation operation was presented which makes use of all available GPU resources of a visualization system to distribute and compute new snapshots for desired parameter combinations. The evaluation operation has then been benchmarked and found to be fast enough for an interactive exploration of a high-resolution simulation. While limitations regarding the scalability of the proposed system exist, solutions to mitigate them have been sketched.

Beside surrogate models based on sparse grids we are currently also considering other types of non-intrusive surrogate

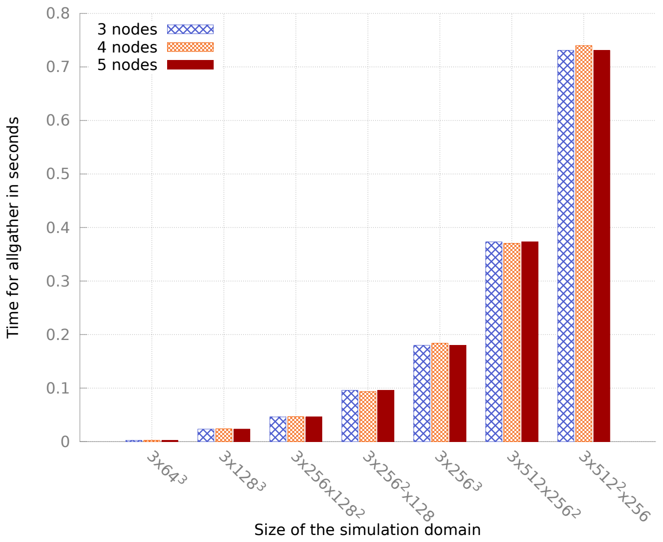


Fig. 10. All-gather cost for different snapshot sizes and numbers of nodes. The given spatial resolution refers to the discretization of the three component velocity vector field Ω.

models [14]. Depending on the setting and visualization task, each of them will have its advantages.

Acknowledgment

This publication is based on work supported by Award No. UK-C0020, made by King Abdullah University of Science and Technology (KAUST).

References

[1] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker, "Surrogate-based analysis and optimization," *Progress in Aerospace Sciences*, vol. 41, no. 1, pp. 1 – 28, 2005.

[2] K. Matkovic, D. Gracanin, M. Jelovic, and Y. Cao, "Adaptive interactive multi-resolution computational steering for complex engineering systems," in *EuroVis Workshop on Visual Analytics*, 2011.

[3] M. S. Eldred and D. M. Dunlavy, "Formulations for surrogate-based optimization with data-fit, multifidelity and reduced-order models," in *Proceedings of the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2006.

[4] B. N. B. D.M. Luchtenburg and M. Schlegel, "An introduction to the POD Galerkin method for fluid flows with analytical examples and MATLAB source codes." Tech. Rep., 2009.

[5] R. Jin, W. Chen, and T. Simpson, "Comparative studies of metamodelling techniques under multiple modelling criteria," *Structural and Multidisciplinary Optimization*, 2001.

[6] A. Klimke and C. J. Pye, "Sparse grid meta-models for model updating," in *Proceedings of the IMAC XXVII Conference*, 2009.

[7] D. Butnaru, D. Pflüger, and H.-J. Bungartz, "Towards high-dimensional computational steering of precomputed simulation data using sparse grids," *Procedia CS*, vol. 4, pp. 56–65, 2011.

[8] H.-J. Bungartz and M. Griebel, "Sparse grids," *Acta Numerica*, vol. 13, pp. 147–269, 2004.

[9] D. Pflüger, *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. München: Verlag Dr. Hut, 2010.

[10] C. Zenger, "Sparse grids," in *Parallel Algorithms for Partial Differential Equations*, ser. Notes on Numerical Fluid Mechanics, W. Hackbusch, Ed., vol. 31. Vieweg, 1991, pp. 241–251.

[11] R. B. Miller, "Response time in man-computer conversational transactions," in *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, 1968, pp. 267–277.

[12] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, 1995.

[13] ——, "Stream: Sustainable memory bandwidth in high performance computers," University of Virginia, Tech. Rep., 1991-2007.

[14] M. Frangos, Y. Marzouk, K. Willcox, and B. van Bloemen Waanders, *Surrogate and Reduced-Order Modeling: A Comparison of Approaches for Large-Scale Statistical Inverse Problems*. John Wiley & Sons, Ltd, 2010, pp. 123–149.