Space and Time Parallel Multigrid for Optimization and Uncertainty Quantification in PDE Simulations

Lars Grasedyck, Christian Löbbert, Gabriel Wittum, Arne Nägel, Volker Schulz, Martin Siebenborn, Rolf Krause, Pietro Benedusi, Uwe Küster, and Björn Dick

Abstract In this article we present a complete parallelization approach for simulations of PDEs with applications in optimization and uncertainty quantification. The method of choice for linear or nonlinear elliptic or parabolic problems is the geometric multigrid method since it can achieve optimal (linear) complexity in terms of degrees of freedom, and it can be combined with adaptive refinement strategies in order to find the minimal number of degrees of freedom. This optimal solver is parallelized such that weak and strong scaling is possible for extreme scale HPC architectures. For the space parallelization of the multigrid method we use a tree based approach that allows for an adaptive grid refinement and online load balancing. Parallelization in time is achieved by SDC/ISDC or a space-time formulation. As an example we consider the permeation through human skin which serves as a diffusion model problem where aspects of shape optimization, uncertainty quantification as well as sensitivity to geometry and material parameters are studied. All methods are developed and tested in the UG4 library.

L. Grasedyck (✉) • C. Löbbert
IGPM, RWTH Aachen, Aachen, Germany
e-mail: lgr@igpm.rwth-aachen.de; loebbert@igpm.rwth-aachen.de

G. Wittum • A. Nägel
G-CSC, University of Frankfurt, Frankfurt, Germany
e-mail: wittum@gcsc.uni-frankfurt.de; naegel@gcsc.uni-frankfurt.de

V. Schulz • M. Siebenborn
University of Trier, Trier, Germany
e-mail: volker.schulz@uni-trier.de; siebenborn@uni-trier.de

R. Krause • P. Benedusi
ICS, University of Lugano, Lugano, Germany
e-mail: rolf.krause@usi.ch; pietro.benedusi@usi.ch

U. Küster • B. Dick
HLRS, University of Stuttgart, Stuttgart, Germany
e-mail: dick@hlrs.de; kuester@hlrs.de

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing – SPPEXA 2013-2015*, Lecture Notes in Computational Science and Engineering 113,
DOI 10.1007/978-3-319-40528-5_23

1 Introduction

From the very beginning of computing, numerical simulation has been the force driving the development. Modern solvers for extremely large scale problems require extreme scalability and low electricity consumption in addition to the properties solvers are always expected to exhibit—like optimal complexity and robustness. Naturally, the larger the system becomes, the more crucial is the asymptotic complexity issue. In this article, in order to get the whole picture, we give a brief review of recent developments towards optimal parallel scaling for the key components of numerical simulation. We consider parallelization in space in Sect. 2, in time in Sect. 4, and with respect to (uncertain) parameters in Sect. 6. These three approaches are designed to be perfectly compatible with each other and can be combined in order to multiply the parallel scalability. At the same time they are kept modular and could in principle also be used in combination with other methods. We address the optimal choice of CPU frequencies for the components of the multigrid method in Sect. 3. This serves as a representative first step for the general problem of finding an energy optimal solver, or respectively energy optimal components. Finally, in Sect. 5 the whole simulation tool is embedded in a typical optimization framework.

2 Parallel Adaptive Multigrid

To accommodate parallel adaptive multigrid computation, we developed the simulation system UG [2], which is now available in the fourth version, UG4 [21, 28]. UG4 is a solver for general systems of partial differential equations. It features hybrid unstructured grids in one, two and three space dimensions, a discretization toolbox using finite element and finite volume schemes of arbitrary order and geometric and algebraic multigrid solvers. It allows for adaptive grid refinement. Furthermore, UG4 features a flexible and self adaptive graphical user interface based on VRL [13].

In our first test we investigate the scaling properties of the geometric multigrid method in UG4 by a weak scaling test for the simple 3d-Laplace model problem (cf. Sect. 5 for strong scaling tests). As can be seen from Table 1 and Fig. 1 we achieve almost perfect weak scaling.

In our second numerical test we consider the weak scaling efficiency for a slightly more involved structural mechanics problem, 3d linear elasticity. The results in Table 2 show the same almost perfect weak scaling.

Adaptivity is a key tool for HPC, the larger the problem becomes, the more important adaptive grid resolution becomes. This can be seen from the following numerical test example computed by Arne Nägel, Sebastian Reiter and Andreas Vogel, see also [29]. To compute diffusion across human skin, we model the main barrier, i.e. the uppermost skin layer, the stratum corneum. The stratum corneum

Table 1 Weak scaling on JUQUEEN. 3d-Laplacian, uniform grid, finite volumes with linear ansatz functions, geometric multigrid V-cycle, damped Jacobi smoother ($\nu_1 = \nu_2 = 2$). We denote by p the number of processors, by dofs the number of degrees of freedom, by N_{iter} the number of multigrid iterations, and by $T_{\text{ass}}, T_{\text{setup}}$, and T_{solve} the elapsed time for the assembly, setup, and solve, respectively

p	L	dofs	N_{iter}	T_{ass}	(eff.)	T_{setup}	(eff.)	T_{solve}	(eff.)
64	8	4,198,401	10	4.46	–	2.22	–	3.04	–
256	9	16,785,409	10	4.47	99.6	2.17	102.2	3.08	98.6
1,024	10	67,125,249	10	4.46	99.9	2.32	95.6	3.13	97.0
4,096	11	268,468,225	10	4.40	101.3	2.26	98.3	3.17	95.8
16,384	12	1,073,807,361	10	4.42	100.9	2.38	98.3	3.27	93.0
65,536	13	4,295,098,369	10	4.42	100.9	2.47	89.7	3.40	89.5
262,144	14	17,180,131,329	10	4.47	99.7	2.62	84.9	3.55	85.5

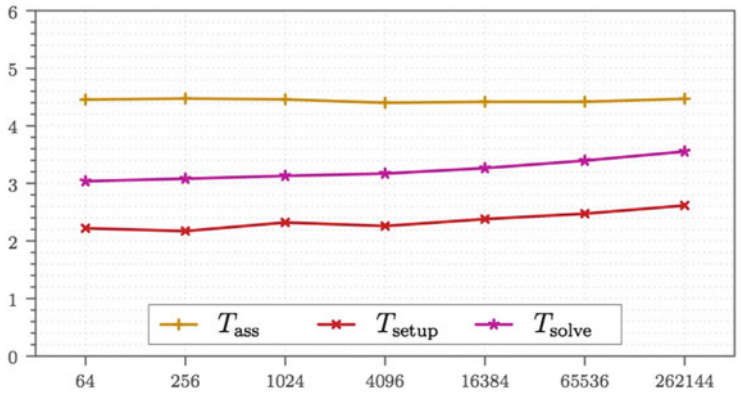

Fig. 1 Weak scaling on JUQUEEN. 3d-Laplacian, uniform grid, finite volumes with linear ansatz functions, geometric multigrid V-cycle, damped Jacobi smoother ($\nu_1 = \nu_2 = 2$). Plotted is the elapsed time $T_{\text{ass}}, T_{\text{setup}}$, and T_{solve} for the assembly, setup and solve phase, respectively, for $p = 64, \ldots, 262{,}144$ processors (From [21])

consists of dead horn cells, the corneocyctes, which are glued together by lipid bilayers. As geometry model we use the so-called cuboid model as shown in Fig. 2. To compute diffusion of a substance across stratum corneum, we use a diffusion equation with constant diffusivities in the two different materials, corneocytes and lipids, and add a transmission condition for the interior material boundaries

$$\frac{\partial c(\mathbf{x}, t)}{\partial t} = \operatorname{div}\left(k(\mathbf{x}) \nabla c(\mathbf{x}, t)\right)$$

with the diffusivities

$$k(\mathbf{x}) = \begin{cases} k_{lip}, & \mathbf{x} \text{ in lipid layer} \\ k_{cor}, & \text{otherwise} \end{cases}$$

Table 2 Weak scaling on JUQUEEN. 3d linear elasticity, uniform grid, finite volumes with linear ansatz functions, geometric multigrid V-cycle, damped Jacobi smoother ($\nu_1 = \nu_2 = 2$). Legend as in Table 1

p	L	dofs	$T_{\text{ass}} + T_{\text{setup}} + T_{\text{solve}}$	(eff.)
1	3	14,739	7.33	–
8	4	107,811	7.42	98.8
64	5	823,875	7.58	96.8
512	6	6,440,067	7.79	94.2
4,096	7	50,923,779	7.90	92.8
32,768	8	405,017,091	8.08	90.7
262,144	9	3,230,671,875	8.21	89.4

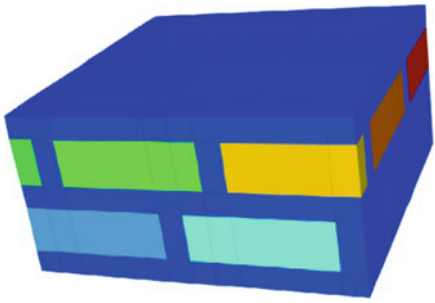

Fig. 2 Cuboid model of human stratum corneum. The corneocytes are modeled by cuboids, measuring $30 \times 30\mu$m horizontally and 1μm in vertically, the lipid layer is assumed to be 100 nm thick

Table 3 Weak scaling on JUQUEEN, skin problem 3d cuboid, uniform refinement, geometric multigrid V-cycle, damped Jacobi smoother ($\nu_1 = \nu_2 = 3$, $\omega = 0.6$), base level 4, base solver LU. Legend as in Table 1

p	L	dofs	N_{iter}	T_{ass}	T_{setup}	T_{solve}
16	6	290,421	25	1.76	8.17	20.23
128	7	2,271,049	27	1.77	8.20	22.31
1,024	8	17,961,489	29	1.78	8.45	24.10
8,192	9	142,869,025	29	1.78	8.48	23.35
65,536	10	1,139,670,081	29	1.79	8.59	24.79

and the transmission condition

$$K_{cor/lip} \cdot c_{lip}(\mathbf{x}, t) \mid_{\mathbf{n}_-} = c_{cor}(\mathbf{x}, t) \mid_{\mathbf{n}_+}$$

for the interior material boundaries. The transmission condition describes the so-called partitioning effect caused by the lipophilicity, respectively hydrophilicity, of the diffusing substance. The problem here is the extreme anisotropy, which is combined with the jumping diffusivities of the material. Together, these features cause an optimal barrier effect as described in [12, 20]. We used this model for a scaling study with uniform refinement. The results are shown in Table 3. The same model was used to study the influence of adaptive grid refinement in parallel. To that end, we did a weak scaling study of this problem with adaptive refinement using a residual error estimator as refinement criterion and compared this with the uniform refinement results. Plotting this into one graph, we obtain the results in Fig. 3.

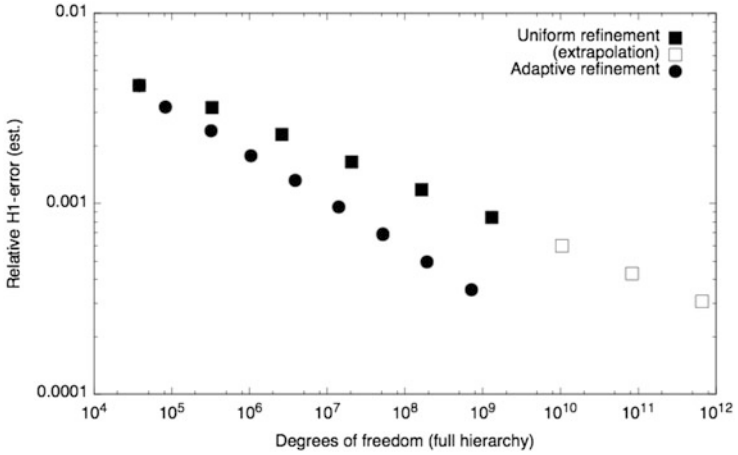

Fig. 3 Error reduction per degrees of freedom for uniform and adaptive refinement in parallel (From [29])

From that we conclude:

- Using the full machine with the adaptive approach, we gain an accuracy comparable to the one with a computer 512 times as large.
- Adaptivity is a leading method for power saving. To reach the same error with the adaptive method, you need just 1024 CPUs instead of 65,536 CPUs in the uniform case or using 65,536 CPUs for the adaptive computation, we would need a computer 512 times larger, i.e. with 33,554,432 CPUs, to reach the same accuracy on a uniform grid. This means saving 99.5 % in CPU time and in power consumption.

3 Empirically Determined Energy Optimal CPU Frequencies

Besides improved scaling properties also energy efficiency poses a challenge that needs to be tackled in order to enable exascale computing. This is due to rising energy costs, a limited availability of electric power at many sites as well as challenges for heat dissipation. From our point of view, increasing energy efficiency requires approaches on multiple fields. The most important one will be efficient algorithms as addressed in the previous and following sections. Furthermore, efficient implementations of these algorithms are required and the resulting codes need to be executed on energy efficient hardware. Moreover, the CPU's clock frequency may be adjusted according to the current load. In this section, we will give an overview of our approach to do the latter. A detailed description of this method has already been published in [6]. Hence, we just give a summary here and refer to the aforementioned document regarding further details.

3.1 Approach

Our purpose is to figure out the *maximum* energy saving potentials and corresponding runtime impacts achievable by adjusting the CPU's clock frequency. We hence try to minimize the energy required to solve a given problem. This energy can be determined by $E = \int_{t_1}^{t_2} P(t)\,\mathrm{d}t$ with $P(t)$ denoting the present power consumption of the corresponding code, running from time t_1 to t_2. According to [4], P can be approximated by $P = CV^2(t)f(t)$ where C denotes the semiconductor's capacity and $V(t)$ respectively $f(t)$ denote the time dependent supply voltage as well as clock frequency of the CPU. Hence, reducing $f(t)$ and $V(t)$ by so-called dynamic voltage and frequency scaling (DVFS), decreases power but may increase the runtime and therefore energy consumption.[1] In phases with intensive memory access, however, one may observe only a slight increase of runtime because the CPU is anyway forced to wait on the memory subsystem most of the time. However, predicting memory access characteristics in complex codes is a challenging task. Thus, it is also hard to predict the optimal clock frequency and we therefore deploy an empirical approach. Linux also does this in its standard configuration but clock frequency decisions are based on an idle time analysis. In contrast to this, we take advantage of knowledge about potential phase boundaries and adjust the clock frequency immediately to the optimal value instead of spending time for a runtime analysis first.

In order to do so, we employ preparatory measurements to figure out energy optimal clock frequencies and utilize them in subsequent production runs. The overhead induced by this method is negligible if it is possible to determine optimal frequencies within a single node or timestep and use them within plenty of those. The core of our approach is to run the entire target code at a *fixed* clock frequency, measure the resulting power consumption over time, and repeat this procedure with all the available frequencies. The resulting energy consumption of a routine can be determined by integrating the measured power over the routine's runtime. Since all routines have been run and profiled at *all* available frequencies, one can now—per routine—pick out the optimal ones in terms of energy. Hence, phases with varying memory access characteristics can be reflected by adapting the clock frequency per routine to its optimal value.

We emphasize that—while minimizing energy—this method may significantly increase the runtime. Nevertheless we deploy it because we are interested in the maximum energy saving potentials and corresponding runtime impacts of DVFS, as stated above.

[1] We use the commonly utilized term "energy consumption" despite the fact that electrical energy is *converted* to thermal energy.

3.2 Implementation Details

In order to implement the aforementioned approach, a measurement method is required that yields highly reliable and time correlated results. We measure the actual supply voltage V and current I of the used CPU and its associated memory modules as close to these components as possible. The current flow I can be determined according to Ohm's law as $I = \frac{V_R}{R}$ with V_R denoting the voltage drop over a high precision shunt $R = 0.01\Omega$ in the CPU's supply line. Based on these values, one can calculate the present power consumption by $P = VI$. The required measurements are performed by an A/D converter in a separate machine with high accuracy ($\epsilon_{relative} < \pm 1.5\,\%$) and a time resolution of $80\,\mu$s.

As already mentioned before, integrating P over time yields the energy spent in a particular routine. The corresponding time interval is determined by calls to `gettimeofday()` from within the code to be evaluated. This method requires a precise synchronization between the real time clocks of the compute node and the measurement hardware. The Precision Time Protocol (PTP) is employed for this purpose via a separate ethernet link. By this method, an average time deviation of about $20\,\mu$s can be achieved, which is below the time resolution of the used A/D converter and therefore admissible.

Unfortunately it is not possible to separate the power consumption of distinct components—especially CPU cores—with the described method. Parallel runs on multiple cores without tight synchronization and perfect load balancing will hence blur the measured power consumption. We therefore restrict our method to serial runs for a start. Parallel runs might nevertheless be regarded in future research.

In order to compensate for OS jitter as well as other transients, five runs of every test case are evaluated and their median is used in further processing.

Since the time resolution of our measurement system is $80\,\mu$s, it is not possible to make reasonable statements on the energy consumption of routines with a runtime in or below this order of magnitude. We therefore take into account only routines with a runtime of at least 1 ms. Since the used profiler solely provides accumulated times, $\frac{t_{ac}}{n} \geq 1\,\text{ms}$ is used as the selection criterion, where t_{ac} denotes the accumulated runtimes (including subroutine calls) of all routine calls and n denotes their number. In analogy to this, optimal clock frequencies are selected based on the average energy consumption of entire calls (i.e., including subroutine calls) to the respective routine. These criteria are just one of many possible choices and other ones will be investigated in future research.

According to [14], the frequency transition latency of current CPUs is substantial. In case of a rapid series of frequency transition requests it is hence not reasonable to immediately set the new frequency in the target production runs. As a consequence, we wait for a period of $10\,\mu$s after the first request of the series, track further ones, and serve only the latest. The choice of this value will also be subject to future research.

Unfortunately, the library call used for setting the frequencies[2] is blocked until completion of the transition (which may be a substantial amount of time), although the CPU can be used in the usual manner during this period. Because of the possibly large number of frequency transitions, this may decrease the overall performance and at the same time raise the energy consumption to an extent that may diminish the benefits of optimal frequency usage. We therefore trigger the actual frequency transition from within a concurrent helper thread, cf. [6].

3.3 Evaluation

In order to evaluate the described approach, it has been applied to the already mentioned numerical simulation code UG4. As a representative application, UG4 has been deployed to solve a time dependent convection–diffusion problem using the vertex centered finite volume method on a two dimensional grid with the geometric multigrid solver and several combinations of setup parameters, particularly different smoothers. Every timestep involves several phases with differing characteristics, i.e., discretization and system assembly (memory-bounded), system solution (CPU-bounded), as well as output of results (potentially I/O-bounded), which may be exploited by the approach by means of differing clock frequencies. The corresponding runs have been executed on an Ivy Bridge compute node, cf. [6] for technical details.

To quantify the effects of the approach, we compared the resulting runtime and energy consumption of entire runs to those resulting from Linux' default clock frequency management. In our first experimental measurements (cf. [6]) we have found an average energy saving potential of about 10 %, which was, however, contrasted by an average runtime penalty of about 19 %. By further investigating our method since publishing those data, we have found that results are not fully reproducible in between different runs of the approach. Despite this fact, it still seems to be possible to reduce the energy requirements by allowing an increased runtime.

One shortcoming of the approach is the limitation to systems with special measurement equipment. Hence, it will be important to investigate the precision of power estimation by means of hardware performance counters with respect to our approach, in order to use them for the preparatory measurements on conventional systems.

In future research, we will tackle these problems in order to enable full reproducibility. We will, moreover, try to reduce the induced runtime penalty and expand the method to multiple active cores within a socket.

[2] `cpufreq_set_frequency()`

4 Parallel in Time Multigrid

Firstly we mention an inexact variant of the well known time integrator Spectral Deferred Correction (SDC); SDC is commonly used as a smoother for multilevel algorithms in time. The Inexact SDC (ISDC) algorithm can reduce significantly the computational cost of SDC. In fact in SDC, a full system solution is required for an implicit, or semi-implicit strategy. On the other hand in ISDC few multigrid V-cycles are used to get an approximate solution. The effectiveness of this technique is due to the iterative nature of SDC that provides an accurate initial guess for the multigrid cycles. This method has been tested on the heat equation (see Table 4) and Viscous Burgers' equations in [27].

The natural usage of the ISDC time stepper is in the context of multilevel time-parallel algorithms, e.g. MLSDC [26] or PFASST [7], based originally on SDC. Both schemes perform SDC sweeps in a hierarchy of levels and use a FAS correction term for the spatial representation of the problem on different levels. ISDC can further improve parallel efficiency of those parallel-in-time methods [16].

Secondly we mention the results in the context of a multigrid space–time solution method for the Navier-Stokes equations with periodic boundary conditions in time [3]. The Navier-Stokes equations are discretized in space–time with high order finite differences on a staggered grid. The discretization leads to a large, ill-conditioned, non-linear system that has to be solved in parallel. Picard iterations are used to treat the non-linearity and we design a block Gauss-Seidel smoother for a space–time multigrid algorithm. A local Fourier analysis is used to analyze the smoothing property of such a method on a staggered grid. The space–time domain is fully decomposed resulting in a parallel-in-time method. Convergence and weak/strong scaling were successfully tested (see Fig. 4).

Table 4 Accumulated Multigrid V-cycles over all sweeps to reduce the SDC or ISDC residual below $5 \cdot 10^{-8}$ for different values of the diffusion coefficient k in the heat equation and the number of quadrature nodes M

k	M	SDC	ISDC	Savings (%)
1	3	16	12	25
	5	23	20	13
	7	32	28	13
k	M	SDC	ISDC	Savings (%)
10	3	36	20	44
	5	61	40	34
	7	79	47	41
k	M	SDC	ISDC	Savings (%)
100	3	106	52	51
	5	150	104	31
	7	187	167	11

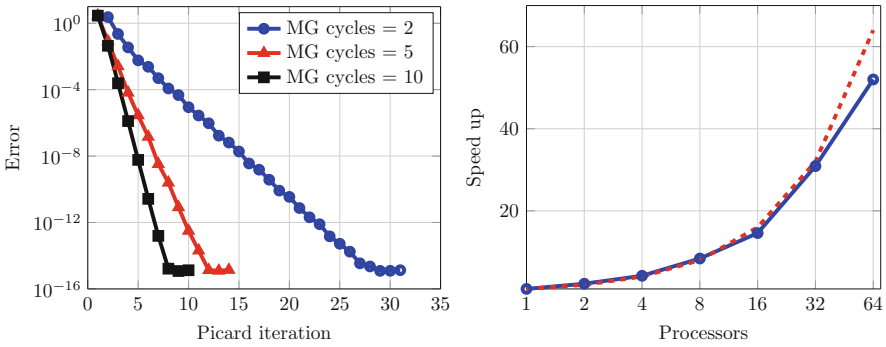

Fig. 4 *Left*: convergence of the Picard iteration with different numbers of multigrid cycles per iteration. *Right*: strong scaling results with processors equally distributed in space and time

5 Scalable Shape Optimization Methods for Structured Inverse Modeling in 3D Diffusive Processes

We consider the inverse modeling of the shape of cells in the outermost layer of the human skin, the so-called stratum corneum. For this purpose we present a novel algorithm combining mathematical shape optimization and high performance computing. In order to show the capabilities of this method, we assume that we have an experiment providing a time-series of data describing the spatial distribution of a tracer in a skin sample. Based on this information, we aim at identifying the structure and the parameters matching the experimental results best. The starting point is a common computational model for the so-called stratum corneum based on tightly coupled tetrakaidecahedrons. For a review, the reader is referred, e.g., to [17, 18].

From a computational point of view, this means to evaluate the model equations, compute the defect to the measurements, evaluate sensitivities of this defect with respect to the shape of the parameter distribution and finally update the shape in order to minimize the defect. A special focus is on the scalability of the optimization algorithm for large scale problems. We therefore apply the geometric multigrid solver UG4 [28].

In this particular application, we are dealing with flows dominated by diffusion. We thus choose the classical parabolic model equation for the simulation together with standard finite elements. By c we denote the concentration of the quantity of interest in the domain $\Omega = \Omega_1 \cup \Omega_2$ over the time interval $[0, T]$. At the initial time $t = 0$, the concentration c is fixed to homogeneously zero in the entire domain and one at the upper boundary Γ_{top}. The other boundaries denoted by Γ_{out} are modeled such that there is no flux across them. The permeability of the domain Ω is given by a jumping coefficient k taking two distinct values k_1 and k_2 in Ω_1 and Ω_2. It can thus be thought of as a homogeneous material with inclusions of different permeability

separated by the interior boundary Γ_{int}. The underlying model is given by

$$\min\ J(\Omega) := \frac{1}{2}\sum_{i=1}^{M}\int_{\Omega}(c(t_i)-\bar{c}(t_i))^2 dx + \mu\int_{\Gamma_{int}} 1\, ds \tag{1a}$$

$$\text{s.t.}\ \frac{\partial c}{\partial t} - \operatorname{div}(k\nabla c) = f \quad \text{in } \Omega\times(0,T] \tag{1b}$$

$$c = 1 \quad \text{on } \Gamma_{top}\times(0,T] \tag{1c}$$

$$[\![c]\!] = 0\,, \quad \left[\!\!\left[k\frac{\partial c}{\partial \mathbf{n}}\right]\!\!\right] = 0 \quad \text{on } \Gamma_{int}\times(0,T] \tag{1d}$$

$$\frac{\partial c}{\partial \mathbf{n}} = 0 \quad \text{on } \Gamma_{out}\times(0,T] \tag{1e}$$

$$c = c_0 \quad \text{in } \Omega\times\{0\}\,. \tag{1f}$$

The first term in (1a) tracks the observations and the second term is a perimeter regularization. Thus, the optimization tends to shapes Γ_{int} with minimal surface area. Equations (1d) describe the continuity of the concentration and of the flux across Γ_{int}.

The corresponding adjoint equation, which is obtained by deriving the Lagrangian (cf. [25]) of problem (1a), (1b), (1c), and (1d) with respect to the state c, then reads as

$$-\frac{\partial p}{\partial t} - \operatorname{div}(k\nabla p) = \begin{cases} -(c-\bar{c}) & \text{in } \Omega\times\{t_1,\dots,t_M\} \\ 0 & \text{in } \Omega\times[0,T)\setminus\{t_1,\dots,t_M\} \end{cases} \tag{2a}$$

$$p_2 = k_1\frac{\partial p}{\partial n}\,, \quad p = 0 \quad \text{in } \Omega\times\{T\} \tag{2b}$$

$$[\![p]\!] = 0\,, \quad \left[\!\!\left[k\frac{\partial p}{\partial n}\right]\!\!\right] = 0 \quad \text{on } \Gamma_{\text{int}}\times[0,T) \tag{2c}$$

$$p_1 = -k_1 p\,, \quad \frac{\partial p}{\partial n} = 0 \quad \text{on } \Gamma_{\text{out}}\times[0,T) \tag{2d}$$

$$p = 0 \quad \text{on } \Gamma_{\text{top}}\times[0,T)\,. \tag{2e}$$

In order to derive the derivative with respect to the shape, first the space of feasible shapes has to be defined. For more details on the connection of shape calculus and shape manifolds, see [22]. We consider the manifold

$$B_e(S^2,\mathbb{R}^3) := \text{Emb}(S^2,\mathbb{R}^3)/\text{Diff}(S^2) \tag{3}$$

of smooth embeddings $\text{Emb}(S^2,\mathbb{R}^3)$ of the unit sphere S^2 into $\mathbb{R}^3$. Let $b \in B_e(S^2,\mathbb{R}^3)$ be a feasible shape, then the tangent space to the manifold in b is given

by all smooth deformations in normal direction

$$T_b B_e = \{h \mid h = \alpha \mathbf{n},\ \alpha \in C^\infty(S^2, \mathbb{R})\}\ . \tag{4}$$

Additionally, we need to equip the tangential space with an inner product. Here we take the so-called Sobolev metric for a constant $\gamma > 0$ given by

$$g^1 : T_b B_e \times T_b B_e \to \mathbb{R},\ (u, v) \mapsto \int_b \langle (\mathrm{id} - \gamma \Delta_b) u,\ v \rangle\, ds\ . \tag{5}$$

The symbol Δ_b denotes the tangential Laplace or Laplace-Beltrami operator along b. This inner product determines the representation of the shape gradient which is then the actual update to the shape in each optimization step. The Sobolev inner product with the Laplace-Beltrami operator and a proper parameter γ ensure smooth shape deformations such that the optimized shape remains in B_e.

The shape derivative in direction of a smooth vector field $V : \Omega \to \mathbb{R}^3$ is defined as

$$dJ(\Omega)[V] := \lim_{h \to 0} \frac{J(\Omega_h) - J(\Omega)}{h} \tag{6}$$

where $\Omega_h = \{x + h \cdot V(x) \mid x \in \Omega\}$ is perturbed according to V. For the underlying model equations the shape derivative is derived in [25] and is given by

$$dJ(\Omega)[V] = \int_{\Gamma_{int}} \left(\int_0^T \langle V, \mathbf{n} \rangle \left[\!\!\left[-2k \frac{\partial c}{\partial \mathbf{n}} \frac{\partial p}{\partial \mathbf{n}} + k \nabla c^T \nabla p \right]\!\!\right] dt + \langle V, \mathbf{n} \rangle\, \mu \kappa \right) ds \tag{7}$$

where $\kappa : \Gamma_{int} \to \mathbb{R}$ denotes the sum of the principle curvatures of the variable surface Γ_{int}.

In most applications, the measurements $\bar{c}$ are not available as a continuous function. There is rather a set of discrete measurements in space. We thus apply radial basis functions in order to interpolate $\bar{c}$ to the finite element nodes where c is given.

The next step is to obtain a descent direction which can be applied as a deformation to the mesh. On each triangle $\tau \subset \Gamma_{int}$ we evaluate the quantity

$$\delta_0 := \left[\!\!\left[-2k \frac{\partial c}{\partial \mathbf{n}} \frac{\partial p}{\partial \mathbf{n}} + k \nabla c^T \nabla p \right]\!\!\right] \tag{8}$$

i.e., the jump of the value in brackets between in two opposing tetrahedra on Γ_{int} sharing a common triangle. Rescaling $\mathbf{n}$, we define the vector

$$\mathbf{g}_0 := \delta_0 \mathbf{n}. \tag{9}$$

For linear finite elements, both δ_0 and $\mathbf{g}_d$ are piecewise constant on each surface triangle. Thus, in order to be consistent with the curvature, which is available in each surface node, we project $\mathbf{g}_d$ onto a vector $\mathbf{g}_c$ in the space of piecewise linear basis functions via the L_2 projection, i.e.,

$$\int_{\Gamma_{int}} \mathbf{g}_c \, \mathbf{v} \, ds = \int_{\Gamma_{int}} \mathbf{g}_d \, \mathbf{v} \, ds \tag{10}$$

for all piecewise linear trial functions $\mathbf{v}$ on Γ_{int}. By solving $(\mathrm{id} - \gamma \Delta_b)\mathbf{g} = \mathbf{g}_c$ with a discretization of the Laplace-Beltrami operator as derived in [15] we finally obtain the representation of the shape gradient $\mathbf{g}$.

One optimization iteration can be summarized in the following steps:

1. Evaluate measurements on current grid via radial basis function representation,
2. solve parabolic and its adjoint PDE with geometric multigrid,
3. compute δ_0 and integrate over time,
4. L^2 projection of piecewise constant gradient to linear basis function space and add curvature for regularization,
5. solve Laplace-Beltrami equation for the representation of the gradient in the Sobolev metric,
6. solve linear elasticity equations with $\mathbf{g}$ as Dirichlet condition on Γ_{int} and deform the mesh.

The algorithm described here is implemented within the software toolbox UG4 [28]. This software is known to be scalable and features parallel multigrid solvers [21]. Numerical experiments were conducted on the HERMIT[3] supercomputer.

The investigation of the scalability is depicted in Fig. 5a–d. The computations shown are based on a coarse grid with 9923 elements and, due to uniform refinements, a fine grid with 325,156,864 elements on the 5th level. For strong scalability (cf. Fig. 5a–c), one observes that most timings decrease, when p increases. All operations not involving any solver, show this decrease. We can explain the saturation for larger number of cores by the time the coarse grid solver requires, which is a natural behavior. Also the weak scalability, which can be seen in Fig. 5d, reflects our expectations. In the decreasing times, for the gradient computation one clearly sees the difference in the asymptotic behavior of volume cells and surface cells. A more detailed analysis can be found in [19].

In our future work we will focus especially on two issues of the presented method. First, due to the incorporation of the shape gradient as a Dirichlet condition in the mesh deformation, the iterated finite element grids tend to have overlapping elements. In [23] we present shape metrics which circumvent this issue and additionally lead to good mesh qualities. Second, the scalability of the presented

[3] HLRS, Stuttgart, Germany, http://www.hlrs.de/systems/platforms/cray-xe6-hermit/

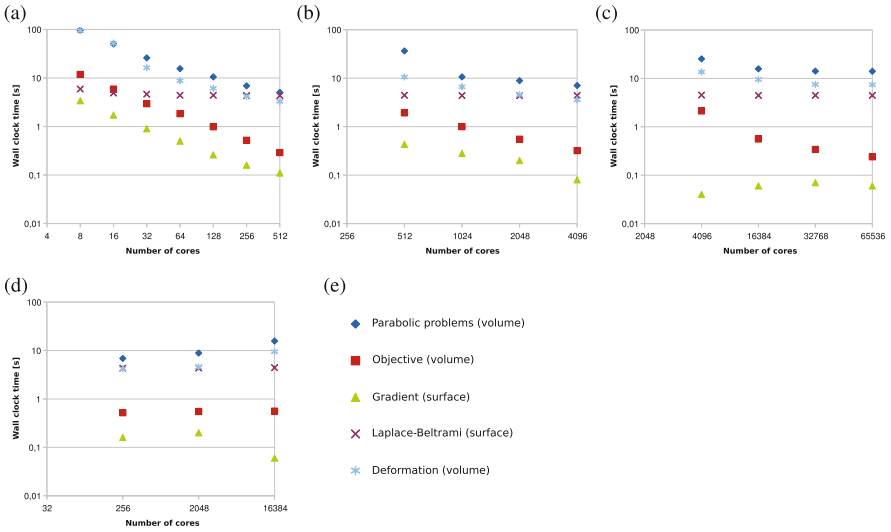

Fig. 5 Scaling of different components of the algorithm in first optimization iteration. (**a**) Strong scaling on level 3. (**b**) Strong scaling on level 4. (**c**) Strong scaling on level 5. (**d**) Weak scaling, increment factor 8 for cells and processors. (**e**) Legend (From [19])

approach is affected by the necessity to solve PDEs on surfaces only. In [24] equivalent formulations for (7) using volume formulations are investigated which overcome the effect on the scalability.

6 Uncertainty Quantification

As a typical parameter-dependent extreme scale problem we consider a PDE involving diffusion coefficients that are parametrized by $\mathbf{p} \in P = [0,1]^d$,

$$\text{div}\ (k(\mathbf{x},\mathbf{p})\nabla c(\mathbf{x},\mathbf{p})) = f(\mathbf{x},\mathbf{p}), \quad \mathbf{x} \in \Omega(\mathbf{p}) \quad + b.c. \tag{11}$$

and assume that for fixed parameters $\mathbf{p} \in P$ a solution $c(\mathbf{x},\mathbf{p})$ is computed in parallel by the UG4 library. The parameters could, e.g., be piecewise diffusion coefficients in each corneocyte. However, we are not interested in the whole solution c of (11) itself but rather in a quantity $\phi : P \to \mathbb{R}$, e.g. the integral mean of the solution $c(\cdot,\mathbf{p})$ over a subset Ω_ϕ:

$$\phi(\mathbf{p}) = \frac{1}{|\Omega_\phi|} \int_{\Omega_\phi} c(\mathbf{x},\mathbf{p})\, d\mathbf{x}\,. \tag{12}$$

Since the parameter set P is d-dimensional, even a discretization with 5 parameter values for each component gives rise to 5^d possible combinations, which exceeds the estimated number of particles in the observable universe already for $d \approx 150$ parameters. Therefore, the full tensor cannot be stored or computed, but rather an extremely data-sparse approximation of it. This approximation is sought in the hierarchical low rank Tucker format [8, 11].

In [1, 10] we have devised a strategy for parallel sampling of tensors in the hierarchical Tucker format, i.e. we compute a few of the values $\phi(\mathbf{p})$ and derive all others from these—based on the assumption that $\phi(\cdot)$ can be approximated in the data sparse low rank hierarchical Tucker format (cf. [5]). In this context sampling means that only certain entries of the tensor are required as opposed to intrusive methods that require us to solve the underlying system of PDEs in the tensor format. The sampling strategy that we propose is guided by the idea that samples are taken one after the other and that later samples can be adapted to the already obtained information of prior samples. This is in contrast to tensor completion strategies [9] where the samples are taken randomly (perfectly parallelizable) and the tensor is completed afterwards.

As a result of [10], parallelization of the (adaptive) sampling process is possible with an almost optimal speedup. Since the method is only a heuristic, it would be helpful to obtain an a posteriori estimate of the approximation quality. For this, we require

- a representation of the underlying (discrete) operator A, the right-hand side b, and the solution c in the hierarchical Tucker format,
- to approximately compute the (discrete) residual $r = b - Ac$,
- to estimate the accuracy by relating it to the residual.

This, however, is still under development. As a first step into this direction, we have distributed the hierarchical low rank tensor according to the dimension tree layout over $2d - 1$ nodes (here we use a complete binary tree and consider only powers of 2 for d). For such a distributed tensor the parallel tensor arithmetic has to be developed. One key ingredient is the evaluation of the tensor, i.e. extracting a single entry from the compressed representation. This procedure has been parallelized and gives the results in Table 5.

Table 5 Parallel weak scaling of the tensor evaluation for distributed tensors. The tensor is of size $100{,}000^d$, the number of processors used is $2d - 1$, the internal rank is $k = 500$ for every node in the dimension tree

d	Parallel time (s)	Serial time (s)	Speedup
4	0.127	0.246	1.9
8	0.261	0.777	3.0
16	0.433	1.880	4.3
32	0.627	4.206	6.7
64	0.882	8.673	9.8
128	0.869	18.82	21.6
256	1.057	38.09	36.0

We observe that the parallel speedup is roughly 36 for a tensor in dimension $d = 256$ with 511 processors. The loss is due the fact that the nodes in the dimension tree have to be processed sequentially one level after the other (which was expected). In addition to distributing the data of the tensor over several nodes, we also gain a considerable speedup.

7 Conclusion

We have presented the development of a parallel multigrid based solver for complex systems and tasks such as shape optimization or uncertainty quantification within the unified UG4 software library. The modular parallelization in space, time, and with respect to parametric dependencies allows us to provide the software for computing way beyond exascale.

Acknowledgements All ten authors gratefully acknowledge support from the DFG (Deutsche Forschungsgemeinschaft) within the DFG priority program on software for exascale computing (SPPEXA), project Exasolvers.

References

1. Ballani, J., Grasedyck, L.: Hierarchical tensor approximation of output quantities of parameter-dependent PDEs. SIAM/ASA J. Uncertain. Quantif. **3**(1), 852–872 (2015)
2. Bastian, P., Wittum, G.: Robustness and adaptivity: the UG concept. In: Hemker, P., Wesseling, P. (eds.) Multigrid Methods IV, Proceedings of the Fourth European Multigrid Conference. Birkhäuser, Basel (1994)
3. Benedusi, P., Hupp, D., Arbenz, P., Krause, R.: A parallel multigrid solver for time-periodic incompressible Navier–Stokes equations in 3d. In: Karasözen, B., Manguoglu, M., Tezer-Sezgin, M., Göktepe, S., Ugur, Ö. (eds.) Numerical Mathematics and Advanced Applications – ENUMATH 2015. Springer, Ankara (2016)
4. Corporation, I.: Enhanced Intel® SpeedStep® Technology for the Intel® Pentium® M Processor. White Paper (2004). http://download.intel.com/design/network/papers/30117401.pdf
5. Dahmen, W., DeVore, R., Grasedyck, L., Süli, E.: Tensor-sparsity of solutions to high-dimensional elliptic partial differential equations. Found. Comput. Math. 1–62 (2015). http://dx.doi.org/10.1007/s10208-015-9265-9
6. Dick, B., Vogel, A., Khabi, D., Rupp, M., Küster, U., Wittum, G.: Utilization of empirically determined energy-optimal CPU-frequencies in a numerical simulation code. Comput. Vis. Sci. **17**(2), 89–97 (2015). http://dx.doi.org/10.1007/s00791-015-0251-1
7. Emmett, M., Minion, M.L.: Toward an efficient parallel in time method for partial differential equations. Commun. Appl. Math. Comput. Sci. **7**, 105–132 (2012)
8. Grasedyck, L.: Hierarchical singular value decomposition of tensors. SIAM J. Matrix Anal. Appl. **31**, 2029–2054 (2010)
9. Grasedyck, L., Kluge, M., Krämer, S.: Variants of alternating least squares tensor completion in the tensor train format. SIAM J. Sci. Comput. **37**(5), A2424–A2450 (2015)

10. Grasedyck, L., Kriemann, R., Löbbert, C., Nägel, A., Wittum, G., Xylouris, K.: Parallel tensor sampling in the hierarchical tucker format. Comput. Vis. Sci. **17**(2), 67–78 (2015)
11. Hackbusch, W., Kühn, S.: A new scheme for the tensor representation. J. Fourier Anal. Appl. **15**(5), 706–722 (2009)
12. Heisig, M., Lieckfeldt, R., Wittum, G., Mazurkevich, G., Lee, G.: Non steady-state descriptions of drug permeation through stratum corneum. I. The biphasic brick-and-mortar model. Pharm. Res. **13**(3), 421–426 (1996)
13. Hoffer, M., Poliwoda, C., Wittum, G.: Visual reflection library: a framework for declarative gui programming on the java platform. Comput. Vis. Sci. **16**(4), 181–192 (2013)
14. Mazouz, A., Laurent, A., Benoît, P., Jalby, W.: Evaluation of CPU frequency transition latency. Comput. Sci. **29**(3–4), 187–195 (2014). http://dx.doi.org/10.1007/s00450-013-0240-x
15. Meyer, M., Desbrun, M., Schröder, P., Barr, A.H.: Discrete differential-geometry operators for triangulated 2-manifolds. In: Hege, H.C., Polthier, K. (eds.) Visualization and Mathematics III, pp. 35–57. Springer, Berlin (2003)
16. Minion, M.L., Speck, R., Bolten, M., Emmett, M., Ruprecht, D.: Interweaving PFASST and parallel multigrid. SIAM J. Sci. Comput. **37**, S244–S263 (2015)
17. Mitragotri, S., Anissimov, Y.G., Bunge, A.L., Frasch, H.F., Guy, R.H., Hadgraft, J., Kasting, G.B., Lane, M.E., Roberts, M.S.: Mathematical models of skin permeability: an overview. Int. J. Pharm. **418**(1), 115–129 (2011)
18. Naegel, A., Heisig, M., Wittum, G.: Detailed modeling of skin penetration – an overview. Adv. Drug Delivery Rev. **65**(2), 191–207 (2013). http://www.sciencedirect.com/science/article/pii/S0169409X12003559. Modeling the human skin barrier – towards a better understanding of dermal absorption
19. Nägel, A., Schulz, V., Siebenborn, M., Wittum, G.: Scalable shape optimization methods for structured inverse modeling in 3D diffusive processes. Comput. Vis. Sci. **17**(2), 79–88 (2015)
20. Nägel, A., Heisig, M., Wittum, G.: A comparison of two- and three-dimensional models for the simulation of the permeability of human stratum corneum. Eur. J. Pharm. Biopharm. **72**(2), 332–338 (2009)
21. Reiter, S., Vogel, A., Heppner, I., Rupp, M., Wittum, G.: A massively parallel geometric multigrid solver on hierarchically distributed grids. Comput. Vis. Sci. **16**(4), 151–164 (2013). http://dx.doi.org/10.1007/s00791-014-0231-x
22. Schulz, V.: A Riemannian view on shape optimization. Found. Comput. Math. **14**, 483–501 (2014)
23. Schulz, V., Siebenborn, M.: Computational comparison of surface metrics for PDE constrained shape optimization. Comput. Methods Appl. Math. (submitted) (2015). arxiv.org/abs/1509.08601
24. Schulz, V., Siebenborn, M., Welker, K.: A novel Steklov-Poincaré type metric for efficient PDE constrained optimization in shape spaces. SIAM J. Optim. (submitted) (2015). arxiv.org/abs/1506.02244
25. Schulz, V., Siebenborn, M., Welker, K.: Structured inverse modeling in parabolic diffusion problems. SIAM J. Control Optim. **53**(6), 3319–3338 (2015). arXiv.org/abs/1409.3464
26. Speck, R., Ruprecht, D., Emmett, M., Minion, M.L., Bolten, M., Krause, R.: A multi-level spectral deferred correction method. BIT Numer. Math. **55**, 843–867 (2015)
27. Speck, R., Ruprecht, D., Minion, M., Emmett, M., Krause, R.: Inexact spectral deferred corrections. In: Domain Decomposition Methods in Science and Engineering XXII. Lecture Notes in Computational Science and Engineering, vol. 104, pp. 127–133. Springer, Cham (2015)
28. Vogel, A., Reiter, S., Rupp, M., Nägel, A., Wittum, G.: UG4: a novel flexible software system for simulating PDE based models on high performance computers. Comput. Vis. Sci. **16**(4), 165–179 (2013). http://dx.doi.org/10.1007/s00791-014-0232-9
29. Wittum, G.: Editorial: algorithmic requirements for HPC. Comput. Vis. Sci. **17**(2), 65–66 (2015)