Automated performance modeling of the UG4 simulation framework

A. Vogel, A. Calotoiu, A. Nägel, S. Reiter, A. Strube, G. Wittum, and F. Wolf

Abstract Many scientific research questions such as the drug diffusion through the upper part of the human skin are formulated in terms of partial differential equations and their solution is numerically addressed using grid based finite element methods. For detailed and more realistic physical models this computational task becomes challenging and thus complex numerical codes with good scaling properties up to millions of computing cores are required. Employing empirical tests we presented very good scaling properties for the geometric multigrid solver in [28] using the UG4 framework that is used to address such problems. In order to further validate the scalability of the code we applied automated performance modeling to UG4 simulations and presented how performance bottlenecks can be detected and resolved in [38]. In this paper we provide an overview on the obtained results, present a more detailed analysis via performance models for the components of the geometric multigrid solver and comment on how the performance models coincide with our expectations.

Andreas Vogel
Goethe Universität Frankfurt, Germany e-mail: andreas.vogel@gcsc.uni-frankfurt.de

Alexandru Calotoiu
Technische Universität Darmstadt, Germany e-mail: calotoiu@cs.tu-darmstadt.de

Arne Nägel
Goethe Universität Frankfurt, Germany e-mail: arne.naegel@gcsc.uni-frankfurt.de

Sebastian Reiter
Goethe Universität Frankfurt, Germany e-mail: sebastian.reiter@gcsc.uni-frankfurt.de

Alexandre Strube
Forschungszentrum Jülich, 52425 Jülich, Germany e-mail: a.strube@fz-juelich.de

Gabriel Wittum
Goethe Universität Frankfurt, Germany e-mail: wittum@gcsc.uni-frankfurt.de

Felix Wolf
Technische Universität Darmstadt, Germany e-mail: wolf@cs.tu-darmstadt.de

1 Introduction

The mathematical description for many important scientific and industrial questions is given by a formulation in terms of partial differential equations. Numerical simulations of the modeled systems via finite element and finite volume discretizations (e.g., [9, 13, 19]) can help to better understand the physical behavior by comparing with measured data and ideally provide the possibility to predict physical scenarios. Using detailed computational grids the discretization thereby leads to large sparse systems of equations and these matrix equations can be resolved using advanced methods of optimal order — such as the multigrid method (e.g., [9, 18]).

Looking at the variety of applications and the constantly growing computing resources on modern supercomputers the efficient solution of partial differential equations is an important challenge and it is advantageous to address the numerous problems with a common framework. Ideally, the framework should provide scalable and reusable components that can be applied in all of the fields of interest and serve as a common base for the construction of applications for concrete problems. To this end the UG software framework has been developed and a renewed implementation has been given in the current version 4.0 ([3, 36]) that pays special attention to parallel scalability.

In order to validate the scaling properties of the software framework on such architectures we carried out several scalability studies. Starting with a hand-crafted analysis we presented close to optimal weak scaling properties of the geometric multigrid solver in [28]. However, the study focussed only on a few coarse-grained aspects leaving room for potential performance bottlenecks, that are not visible at current scales due to a small execution constant, but may become dominant at largest scales due to bad asymptotic behavior. Therefore, in a subsequent study we analyzed entire ***UG*** 4 runs in [38] applying an automated performance modeling approach by Calotoiu et al. [10] to ***UG*** 4 simulations. The modeling approach creates performance models at a function level granularity and uses few measurement runs at smaller core counts in order to predict the asymptotic behavior of each code kernel at largest scales. By detecting bad asymptotic behavior for code kernels in the grid setup phase we were able to detect and remove a performance bottleneck.

In this paper we focus on more detailed models for the geometric multigrid solver and explain how the observed performance models meet our expectations. Since the geometric multigrid solver is one of the crucial aspects for simulation runs in terms of scalability, we have evaluated in more depth the models for fine-grained kernels of the employed geometric multigrid solver and compare the observed behavior to the intended implementation.

The main aspects of this report are:

- Summarize the automated modeling approach and obtained results for its application to the simulation framework ***UG*** 4.
- Provide a detailed analysis for components of the geometric multigrid solver.
- Validation of the scaling behavior for the multigrid solver components.

The remainder of this paper is organized as follows. In Sect. 2, the ***UG*** 4 simulation environment is presented with focus on the parallelization aspects of the parallel geometric multigrid (Sect. 2.2) and the skin permeation problem (Sect. 2.3) used in the subsequent studies. Sect. 3 outlines the performance modeling approach. In Sect. 4 we briefly summarize previously obtained analysis results for entire simulation runs and then present a detailed performance modeling for the geometric multigrid solver used in a weak scaling study for the skin problem. Sect. 5 and Sect. 6 are dedicated to related work and concluding remarks.

2 The UG4 Simulation Framework

As a real world target application code for the performance modeling approach we will focus on the simulation toolbox ***UG*** 4 ([36]). The software framework is written in C++ and uses grid-based methods to numerically address the solution of partial differential equations via finite-element or finite-volume methods. With the main goal to address questions from biology, technology, geology and finance with one common effort, several components are reused in all types of applications and thus the performance modeling for those program parts provides insight into the performance of all these applications. In the following we give a brief overview on the used numerical methods and especially comment on the parallelization aspects.

2.1 Concepts and numerical methods

In order to construct the required geometries the meshing software ProMesh ([2, 27]) is used that shares code parts with the ***UG*** 4 library. Meshes can be composed of different element types (e.g., tetrahedron, pyramid, prism and hexahedron in 3d) and subset assignment is used to distinguish parts of the domain with different physics or where boundary conditions are to be set. Once loaded in ***UG*** 4 meshes are further processed to create distributed, unstructured, adaptive multigrid hierarchies with or without hanging nodes. Implemented load-balancing strategies ([27]) range from simple but fast bisection algorithms to more advanced strategies including usage of external algorithms such as ParMetis ([1]). In this study, however, we restrict ourselves to a 3d hexahedral grid hierarchy generated through globally applied anisotropic refinement (cf. [38]). A study for adaptive hierarchies with hanging nodes is work in progress and will be considered in a subsequent study.

A flexible and combinable discretization module allows to combine different kinds of physical problems discretized by finite-element and finite-volume methods (e.g., [9, 13, 19]) and boundary conditions in a modular way to build a new physical problem selecting from basic building blocks ([36, 37]). As algebraic structures for the discretized solutions and associated matrices , block vectors and a *CSR* (compressed sparse row) matrix implementations are provided. For the parallel solution

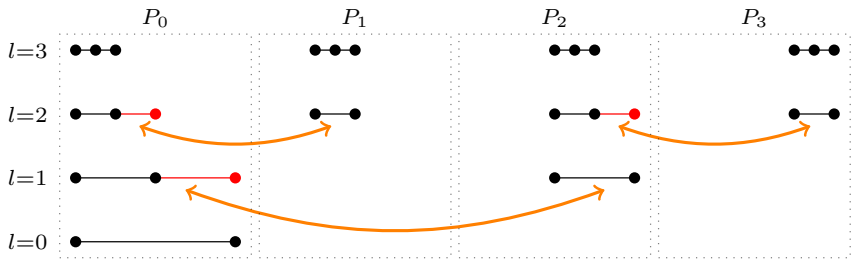


Fig. 1 Illustration for a 1d process hierarchy on 4 processes. Ghost elements (*red*) are sent during redistribution. Data is communicated between ghosts and actual elements through vertical interfaces (*orange*) (cf. [27, 28])

.

of such matrix equations several solvers are implemented, including Krylov methods such as CG and BiCGStab and preconditioners such as Jacobi, Gauss-Seidel, incomplete LU factorization, ILUT and block versions of these types (e.g., [29]). In addition, a strong focus is on multigrid methods (e.g. [18]) and geometric and algebraic multigrid approaches ([20, 28]).

The parallelization for the usage on massively parallel computing clusters with hundred thousands of cores is achieved using `MPI`. The separate library called PCL (parallel communication layer, [28]) builds on top of `MPI` and is used to ease the graph-based parallelization. Both, the parallelization of the computing grid – assigning a part of the multigrid hierarchy to each process – and of the algebraic structures are programmed based on the PCL. By storing parallel copies on each process in a well-defined order in interface containers identification is performed in an efficient way [27, 28, 36], and global IDs are dispensable.

In order to hide parallelization aspects and ease the usage for beginners the scripting language Lua ([21]) is used as end-user interface. A flexible plugin system allows to add additional functionality if required.

2.2 Parallel hierarchical geometric multigrid

The multigrid method ([19]) is used to solve large sparse systems of equations that arise typically by the discretization of some partial differential equation. We briefly recap the idea of the algorithm and our modifications and implementation ([28]) for the parallel version. Given the linear equation system $\mathbf{A}_L\mathbf{x}_L = \mathbf{b}_L$ on the finest grid level L, the desired solution $\mathbf{x}_L$ is computed iteratively: Starting with some arbitrary initial guess $\mathbf{x}_L$, in every iteration the defect $\mathbf{d}_L = \mathbf{b}_L - \mathbf{A}_L\mathbf{x}_L$ is used to compute a multigrid correction $\mathbf{c}_L = \mathbf{M}_L(\mathbf{d}_L)$, where $\mathbf{M}_L$ is the multigrid operator, that is added to the approximate solution $\mathbf{x}_L := \mathbf{x}_L + \mathbf{c}_L$. In order to compute the correction $\mathbf{c}_L$ not only the fine grid matrix $\mathbf{A}_L$ is used but several auxiliary coarse grid matrices $\mathbf{A}_l, L_B \leq l \leq L$, are employed, where L_B denotes the base level. The multigrid cy-

cle is then defined in a recursive manner: given a defect $\mathbf{d}_l$ on a certain level l the correction is first partly computed via a smoothing operator (e.g. Jacobi iteration). Then the defect is transferred to the next coarser level, where the algorithm is applied to the restricted defect $\mathbf{d}_{l-1}$. The thereby computed coarse grid correction $\mathbf{c}_{l-1}$ is then prolongated to the finer level and added to the correction on level l, followed by some postsmoothing. Once the algorithm reaches the base level L_B, the correction is computed exactly as $\mathbf{c}_l = \mathbf{A}_l^{-1}\mathbf{d}_l$ by, e.g., using LU factorization. Algorithm 1 summarizes this procedure.

Algorithm 1 $\mathbf{c}_l = \mathbf{M}_l(\mathbf{d}_l)$ ([19, 28])

Requirement: $\mathbf{d}_l = \mathbf{b}_l - \mathbf{A}_l\mathbf{x}_l$
if $l = L_B$ **then**
  Base solver: $\mathbf{c}_l = \mathbf{A_l}^{-1}\mathbf{d}_l$
  return $\mathbf{c}_l$
else
  Initialization: $\mathbf{d}_l^0 := \mathbf{d}_l$, $\mathbf{c}_l^0 := 0$
  (Pre-)Smoothing for $k = 1, \ldots, \nu_1$:
    $\mathbf{c} = S_l(\mathbf{d}_l^{k-1})$,
    $\mathbf{d}_l^k = \mathbf{d}_l^{k-1} - \mathbf{A}_l\mathbf{c}, \quad \mathbf{c}_l^k = \mathbf{c}_l^{k-1} + \mathbf{c}$
  Restriction: $\mathbf{d}_{l-1} = \mathbf{P}_l^T\mathbf{d}_l^{\nu_1}$
  Coarse grid correction: $\mathbf{c}_{l-1} = \mathbf{M}_{l-1}(\mathbf{d}_{l-1})$
  Prolongation:
    $\mathbf{c}_l^{\nu_1+1} = \mathbf{c}_l^{\nu_1} + \mathbf{P}_l\mathbf{c}_{l-1}$,
    $\mathbf{d}_l^{\nu_1+1} = \mathbf{d}_l^{\nu_1} - \mathbf{A}_l\mathbf{P}_l\mathbf{c}_{l-1}$
  (Post-)Smoothing for $k = 1, \ldots, \nu_2$:
    $\mathbf{c} = S_l(\mathbf{d}_l^{\nu_1+k})$,
    $\mathbf{d}_l^{\nu_1+1+k} = \mathbf{d}_l^{\nu_1+k} - \mathbf{A}_l\mathbf{c}, \quad \mathbf{c}_l^{\nu_1+1+k} = \mathbf{c}_l^{\nu_1+k} + \mathbf{c}$
  return $\mathbf{c}_l^{\nu_1+1+\nu_2}$
end if

The matrix equations for complex problems can easily grow beyond the size of billions of unknowns. In order to solve such problems, massively parallel linear solvers with optimal complexity have to be used. The multigrid algorithm only depends linearly on the number of unknowns and therefore good weak scaling properties are to be expected. As demonstrated in [28] geometric multigrid solvers can exhibit nearly perfect weak scalability when employed in massively parallel environments with hundred thousands of computing cores.

To this end, the components of the algorithm must be parallelized. The basic idea is to construct a distributed multigrid hierarchy as follows:

1. Start with a coarse grid on a small number of processes.
2. Refine the grid several times to create additional hierarchy levels.
3. Redistribute the finest level of the hierarchy to a larger set of processes.

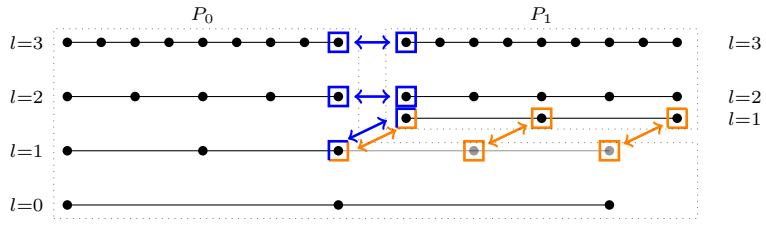


Fig. 2 Illustration for a 1d parallel multigrid hierarchy distributed onto two processes. Parallel copies are identified via horizontal (*blue*) and vertical interfaces (*orange*) (from [38], cf. also [28])

4. Repeat at (2) until the desired grid resolution is obtained. At this point all active processes should contain a part of the finest level of the multigrid hierarchy.

Refining the grid, new levels of the multigrid hierarchy are created and after some refinements the finest grid level is distributed to a larger set of processes and communication structures (called *vertical interfaces*) are established. This process can be iterated, successively creating a tree structure of processes holding parts of the hierarchical grid. Fig. 1 shows a process hierarchy for a distributed multigrid hierarchy on four processes (cf. [27, 28]). The communication structures in vertical direction are used to parallelize the transfer between the grid levels, i.e. to implement the transfer of data between grid levels at restriction and prolongation phases within a multigrid cycle. However, if no vertical interface is present the transfer operators act completely process-locally. For the communication within multigrid smoothers on each grid level additional *horizontal interfaces* are required. These interfaces will be used to compute the level-wise correction in a consistent way. An illustration for the resulting hierarchy distribution and interfaces is given in Fig. 2 (cf. [27, 28, 37, 38]). In order to compute the required coarse grid matrices, each process calculates the contribution of the grid part assigned to the process itself. Thus, the matrices are stored in parallel in an additive fashion and no communication is required for this setup.

A Jacobi smoother has very good properties regarding scalability, however it may not be suitable for more complicated problems (e.g. with anisotropic coefficients or anisotropic grids). To handle this issue for anisotropic problems, we use anisotropic refinement in order to construct grid hierarchies with isotropic elements from anisotropic coarse grids: Refining only those edges in the computing grid that are longer than a given threshold, and halving this threshold in each step, the approach yields a grid hierarchy which contains anisotropic elements on lower levels and more and more isotropic elements on higher levels. An illustration for a resulting hierarchy is shown in Fig. 3. The used refinement strategy produces non-adaptive grids, i.e. meshes that fully cover the physical domain. This eases the load-balancing compared to adaptive meshes where huge differences in the spatial resolution and thereby element distribution may occur during refinement and redistribution is necessary. In this work we focused on the non-adaptive strategy only, however, plan to report on the adaptive case in future works.

Reconsidering the hierarchical distribution approach described above, lower levels of the multigrid hierarchy are only contained on a smaller number of processes. This is well suited for fast parallel smoothing, prolongation and restriction operations thanks to maintaining a good ratio between computation and communication costs on all levels. A smoothing operation on coarser levels with only few inner unknowns would be dominated by the communication and thus the work is agglomerated to fewer processes resulting in idle processes on coarse levels. However, the work on finer grid levels is dominating the overall runtime.

2.3 Application: Human skin permeation

As an exemplary application from the field of computational pharmacy we focus on the permeation of substances through the human skin. These simulations consider the outermost part of the epidermis, called stratum corneum, and are used to estimate the throughput of chemical exposures. An overview on different descriptions of the biological and geometric approaches to simulate such a setting can be found in [24] and the references therein. For this study, we use the same setup as used in [38]: The transport in two subdomains $s \in \{cor, lip\}$ (corneocyte, lipid) is described by the diffusion equation

$$\partial_t c_s(t,\mathbf{x}) = \nabla \cdot (\mathbf{D}_s \nabla c_s(t,\mathbf{x})),$$

using a subdomain-wise constant diffusion coefficient $\mathbf{D}_s$. We use a 3d brick-and-mortar model consisting of highly anisotropic hexahedral elements with aspect ratios as bad as $1/300$ in the coarse grid. Employing anisotropic refinements we construct a grid hierarchy with better and better aspect ratios on finer levels. The resulting grid hierarchy is displayed in Fig. 3. For a more detailed presentation we refer to [38].

3 Automated Performance Modeling

The automated modeling approach used to analyze the ***UG*** 4 framework has been presented by Calotoiu et al. in [10, 11]. Here, we give a brief overview on the procedure and ideas. For further details please refer to [10, 31, 38, 40].

Automated performance modeling is used to empirically determine the asymptotic scaling behavior for a large number of fine-grained code kernels. These scaling models can then be inspected and compared to the expected complexity: A discrepancy indicates a potential *scalability bug* that can be addressed and hopefully removed by the code developers. If no such scalability issues are found this can be taken as a strong evidence that no unexpected scalability problems are present. In

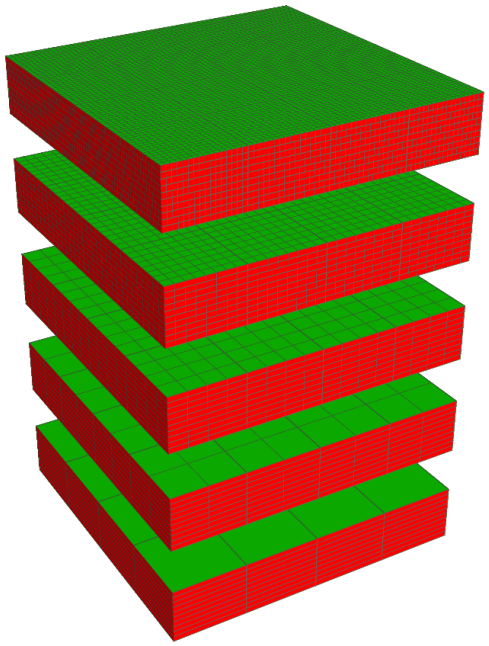


Fig. 3 Grid hierarchy created by anisotropic refinement for the 3d brick-and-mortar model (in exploded view). The aspect ratios of the grid elements improves with every refinement step

addition the created models can also be used to predict the resource consumption at larger core counts if required.

In order to create the models, the simulation framework ***UG*** 4 has been instrumented to measure relevant metrics such as time, bytes sent/received or floating-point operations in program regions at a function level granularity. Running simulations at different core counts now offers the opportunity to determine via cross-validation [26] which choice of parameters in the *performance model normal form* (PMNF, [10])

$$f(p) = \sum_{k=1}^{n} c_k \cdot p^{i_k} \cdot log_2^{j_k}(p),$$

with $i_k, j_k \in I, J \subset \mathbb{Q}$, best fits the measurements. The approach is applicable to strong and weak scaling. In this study, however, we have focused on weak scaling only. In order to account for jitter, several runs for every core count have to executed. The required effort for this approach therefore is to run the application a few times at a few core counts.

For the correct analysis of the multigrid algorithm in weak scaling studies, a more careful approach than just analyzing the code kernels directly has to be taken ([38]). This is due to the following observation: within a weak scaling study the problem size has to be increased and for multigrid approaches this leads to an increase in

the number of grid levels. The multigrid algorithm - traversing the multigrid hierarchy top-down and then again bottom-up, applying smoothers and level transfers for every level - will thus create a different call tree at different core counts for multigrid functions due to the varying numbers of grid levels. However, the performance modeling approach usually assumes the same call tree for all core counts. Therefore, we preprocess the call tree: only kernels present in all measurements remain in the modified call tree and measurement data of code kernels not present in modified call trees is added to the parent kernel. This approach is not only limited to multigrid settings but may be useful for all codes that use recursive calls whose invocation count increases within a scaling study.

4 Results

The automated performance modeling (Sect. 3) has been applied to entire simulation runs of the ***UG*** 4 simulation framework (Sect. 2) and several aspects of the analysis have been reported in [38]. In order to analyze and validate the code behavior the proceeding and reasoning is as follows: We run several simulations at different core counts p and measure detailed metric information (times, bytes sent) at a function level granularity. By this, we receive fine grained information for small code kernels. For all of these kernels and all available metrics we create performance models and then rank these by their asymptotic behavior with respect to the core count. All code kernels with constant or only logarithmical dependency are considered optimal. However, if some code kernel, e.g., in the multigrid method would show a linear or quadratic dependency, this would not match our performance expectations and we consider it a scalability bug that has to be addressed and removed. Inspecting all measured code kernels thus provides us with a fine grained information for different parts of the simulation code. Given that all code kernels show an optimal dependency we finally obtain a validation of the expected scaling properties.

Here, we first briefly give an overview on the results presented in [38] and then show more detailed results focussing on the multigrid kernels and their scaling properties.

4.1 Analysis for grid hierarchy setup and solver comparison

In a first test, we analyzed entire runs in a weak scaling study for the human skin permeation simulating the steady-state concentration distribution on a three-dimensional brick-and-mortar skin geometry. A scalability issue has been detected by the performance modeling that can be explained and resolved ([38]): At the initialization of the multigrid hierarchy an `MPI_Allreduce` operation for an array of length p was used to inform each process about its intended communicator group membership. The resulting $p \cdot O(\texttt{MPI\_Allreduce})$ dependency has been addressed

by using `MPI_Comm_split` instead that can be implemented with a $O(\log^2 p)$ [32] behavior. By this, we were able to remove this potential bottleneck at this stage of code development.

In a second study, we provided a comparison for two different types of solvers: the geometric multigrid solver has been compared in a weak scaling study to the unpreconditioned conjugate gradient (CG) method. The unpreconditioned conjugate gradient method is known to have unpleasant weak-scaling properties due to the increase by a factor of two for the iteration count resulting in a $O(\sqrt{p})$ dependency (see [38] for a detailed theoretical analysis). Due to the created models we confirmed that the theoretical expectations are met by our implementation of the parallel solvers.

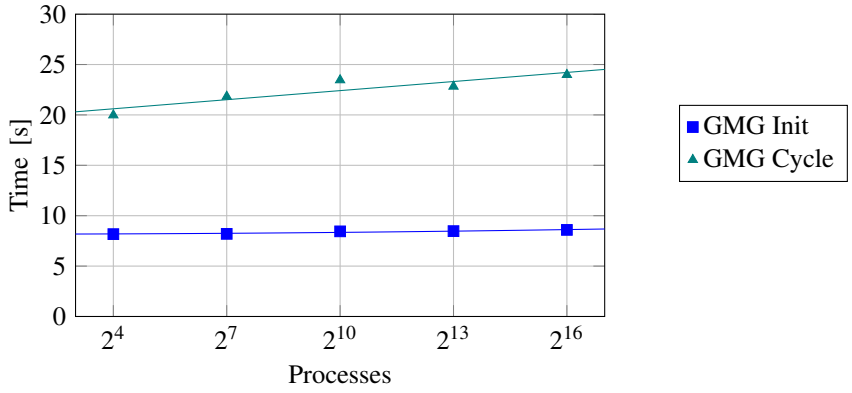
4.2 Scalability of code kernels in the geometric multigrid

In this section we give a more detailed view on the code kernels for the geometric multigrid. For the analysis of the multigrid solver we consider the human skin permeation model: We compute the steady state of the concentration distribution for the brick-and-mortar model described in Sect. 2.3 and choose the diffusion parameter to $\mathbf{D}_{cor} = 10^{-3}$ and $\mathbf{D}_{lip} = 1$. For the solution of the arising linear system of equations, the geometric multigrid solver is used. As acceleration an outer conjugate gradient method is applied. For the smoothing a damped Jacobi is employed with three smoothing steps. As cycle type the V-cycle is used and as base solver we use a LU factorization. The stopping criterion for the solver is the reduction of the absolute residuum to 10^{-10}. The anisotropic refinement as laid out in Sect. 2.2 is used to enhance the aspect ratios of the hierarchy from level to level. Once satisfactory ratios are reached, this level is used as base level for the multigrid algorithm.

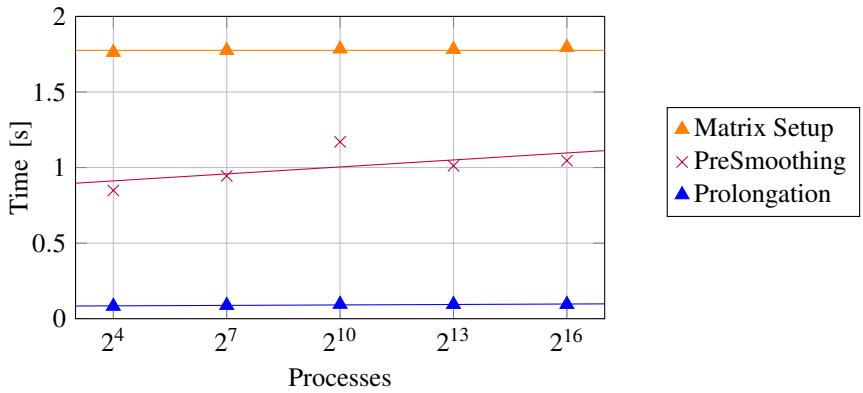
In Fig. 4 we present the accumulated wallclock times for exemplary coarse-grain kernels of the multigrid method and provide information on the number of used cores and the size of the solved matrix system (degrees of freedom). Please note that the iteration counts are bounded as expected for a multigrid method. Since the assembling for the matrix is an inherent parallel process without any communication it can be performed - given an optimal load-balancing - with constant wall-clock time in the weak scaling. This is confirmed by the generated performance model. All other shown aspects of the multigrid method show a logarithmical dependency. This is due to the fact that the number of involved coarse grid levels $L = O(\log p)$ depends on the number of processes in a weak scaling. We consider this logarithmical dependency still as optimal since even allreduce operations implemented in a tree-like fashion will show the same behavior and are used to check for convergence.

The performance models for several code kernels are shown in Tab. 1. Please note that all code kernels in our measurements have shown constant or logarithmical dependency with respect to the number of processes. Here, we show some selected kernels in order to give more details on the parallelization aspects of the

(a)



(b)



(c)

p	L	DoF	n_{gmg}
16	6	290,421	25
128	7	2,271,049	27
1024	8	17,961,489	29
8192	9	142,869,025	29
65536	10	1,139,670,081	29

(d)

Kernel	Model for time [s]
GMG Init	$8.17 + 0.002 \cdot \log_2 p$
GMG Cycle	$19.41 + 0.30 \cdot \log_2 p$
Matrix Setup	1.78
PreSmoothing	$0.850 + 0.015 \cdot \log_2 p$
Prolongation	$0.081 + 0.001 \cdot \log_2 p$

Fig. 4 Measured accumulated wallclock times (*marks*) and models (*lines*) for the skin 3d problem (self time and subroutines). Shown are: **(a)** (cf. [38]) initialization of the multigrid solver and time spent in the multigrid cycle, **(b)** times for coarse matrix assembling, smoothing and prolongation. **(c)** (from [38]) Number of grid refinements (L), degrees of freedom (DoF) and number of iterations of the solver (n_{gmg}). **(d)** Performance models for the kernels shown in the graphs

Table 1 Skin 3d study: Models for self-time and bytes sent for selected kernels of the geometric multigrid method with outer CG iteration. $|1-R^2|$, the absolute difference between R^2 and the optimum scaled by 10^{-3}, can be considered a normalized error

Kernel	Time		Bytes sent	
	Model time $= f(p)[ms]$	$\lvert 1-R^2\rvert$ $[10^{-3}]$	Model bytes $= f(p)$	$\lvert 1-R^2\rvert$ $[10^{-3}]$
`CG` → `GMG` → `PreSmooth` → `Jacobi` → `apply` → `step`	$18.9+0.4\cdot\log p$	42.6	0	0.0
`CG` → `GMG` → `PreSmooth` → `Jacobi` → `apply` → `AdditiveToConsistent` → `MPI_Isend`	$1.51+1.12\cdot\log p$	36.0	$5.77\cdot 10^5+0.95\cdot 10^5\cdot\log p$	53.2
`CG` → `GMG` → `Restrict` → `init`	510	0.0	0	0.0
`CG` → `GMG` → `Restrict` → `apply`	$51.0+0.05\cdot\log p$	378	0	0.0
`CG` → `norm`	$3.52+0.002\cdot\log^2 p$	544	0	0.0
`CG` → `norm` → `AdditiveToUnique` → `MPI_Isend`	$0.52+0.45\cdot\log p$	38.9	$1.95\cdot 10^5+0.34\cdot 10^5\cdot\log p$	45.5
`CG` → `norm` → `allreduce` → `MPI_Allreduce`	$1.67+0.92\cdot\log^2 p$	7.5	O(`MPI_Allreduce`)	0.0

multigrid method. For a more detailed description on the mathematical algorithm and parallelization aspects we refer to Sect. 2 and [28].

The presmoothing is performed in a two step fashion: First, the Jacobi iteration is applied on process-wise data structures resulting in no data transfer (`CG` → `GMG` → `PreSmooth` → `Jacobi` → `apply` → `step`). In a second phase update information is exchanged between nearest neighbors in order to gain a consistent update resulting in data transfer (`PreSmooth` → `Jacobi` → `apply` → `AdditiveToConsistent` → `MPI_Isend`). All behaviors are found to depend logarithmically due to the increase in grid levels that are using this method.

The grid transfer is performed process-wise as well (`Restrict` → `apply`). No communication is needed unless vertical interfaces are present. The setup phase (`Restrict` → `init`) simply assembles the transfer operators into a matrix structure on each process and a constant time within a weak scaling is thus observed.

Finally, we show some kernels for the outer CG iteration. In order to check for convergence, the norm of a defect vector is computed in each iteration step. After a nearest neighbor communication in order to change the storage type of the vector (`CG` → `norm` → `AdditiveToUnique` → `MPI_Isend`), the norm is first computed on each process (`CG` → `norm`) and then summed up globally (`CG` → `norm` → `allreduce` → `MPI_Allreduce`).

This way our expectations for the code kernels of the multigrid solver are confirmed and we have strong evidence that only logarithmical complexity with respect to the core count (or better) occurs.

5 Related Work

Numerous analytical and automated performance modeling approaches have been proposed and developed. The field ranges from manual models [8, 25], capable to effectively describe characteristics of entire tool chains, over source-code annota-

tions [35] to specialized languages [33]. Automated modeling methods are developed based on machine-learning approaches [22], and via extrapolating trace measurements in [42] (extraplating from single-node to parallel architectures), in [41] (predicting communication costs at large core counts) and in [12] (extraplating based on a set of canonical functions). The Dimemas simulator provides tools for performance analysis in message-passing programs [16].

Various frameworks to solve partial differential equations use multigrid methods. Highly scalable multigrid methods are presented, in [7], [17], [30], [34], and [39] for geometric multigrid, and in [4], [5], and [6] for algebraic multigrid methods. Work on performance modeling for multigrid can be found in [15, 38] for geometric and in [14] for algebraic multigrid. For an overview for the numerical treatment of skin permeation, we refer to [23] and the references therein.

6 Conclusion

The numerical simulation framework UG4 consists of half a million lines of code and is used to address problems formulated in terms of partial differential equations employing multigrid methods to solve arising large sparse matrix equations. In order to analyze, predict and improve the scaling behavior of UG4 we have conducted a performance modeling at code kernel granularity. Inspecting automated performance models we validated the scalability of entire simulations and presented the close to optimal weak scaling properties for the components of the employed geometric multigrid method that only depend logarithmically on the core count.

Acknowledgment

Financial support from the DFG Priority Program 1648 *Software for Exascale Computing* (SPPEXA) is gratefully acknowledged. The authors also thank the Gauss Centre for Supercomputing (GCS) for providing computing time on the GCS share of the supercomputer JUQUEEN at Jülich Supercomputing Centre (JSC).

References

1. Parmetis: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview.
2. Promesh: http://www.promesh3d.com.
3. UG4: https://github.com/UG4.
4. A.H. Baker, R.D. Falgout, T.V. Kolev, and U.M. Yang. Multigrid smoothers for ultra-parallel computing. *SIAM J. Sci. Comput*, 33:2864–2887, 2011.
5. Allison H Baker, Robert D Falgout, Todd Gamblin, Tzanio V Kolev, Martin Schulz, and Ulrike Meier Yang. Scaling algebraic multigrid solvers: On the road to exascale. In *Competence in High Performance Computing 2010*, pages 215–226. Springer, 2012.

6. P. Bastian, M. Blatt, and R. Scheichl. Algebraic multigrid for discontinuous galerkin discretizations of heterogeneous elliptic problems. *Numerical Linear Algebra with Applications*, 19(2):367–388, 2012.
7. B. Bergen, T. Gradl, U. Rude, and F. Hulsemann. A massively parallel multigrid method for finite elements. *Computing in Science & Engineering*, 8(6):56–62, 2006.
8. Eric L. Boyd, Waqar Azeem, Hsien-Hsin Lee, Tien-Pao Shih, Shih-Hao Hung, and Edward S. Davidson. A hierarchical approach to modeling and improving the performance of scientific applications on the KSR1. In *Proc. of the Intl. Conference on Parallel Processing (ICPP)*, pages 188–192, 1994.
9. Dietrich Braess. *Finite Elemente*. Springer, 2003.
10. Alexandru Calotoiu, Torsten Hoefler, Marius Poke, and Felix Wolf. Using automated performance modeling to find scalability bugs in complex codes. In *Proc. of the ACM/IEEE Conference on Supercomputing (SC13), Denver, CO, USA*. ACM, November 2013.
11. Alexandru Calotoiu, Torsten Hoefler, and Felix Wolf. Mass-producing insightful performance models. In *Workshop on Modeling & Simulation of Systems and Applications, University of Washington*, Seattle, Washington, August 2014.
12. Laura Carrington, Michael Laurenzano, and Ananta Tiwari. Characterizing large-scale HPC applications through trace extrapolation. *Parallel Processing Letters*, 23(4), 2013.
13. Philippe G. Ciarlet and J.L. Lions. *Finite element methods (part 1)*. North-Holland, Amsterdam, 1991.
14. Hormozd Gahvari, Allison H Baker, Martin Schulz, Ulrike Meier Yang, Kirk E Jordan, and William Gropp. Modeling the performance of an algebraic multigrid cycle on HPC platforms. In *Proc. of the International Conference on Supercomputing*, pages 172–181. ACM, 2011.
15. Hormozd Gahvari and William Gropp. An introductory exascale feasibility study for FFTs and multigrid. In *International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–9. IEEE, 2010.
16. S. Girona, J. Labarta, and R. M. Badia. Validation of dimemas communication model for mpi collective operations. In *Proc. of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 39–46, London, UK, UK, 2000. Springer-Verlag.
17. Björn Gmeiner, Harald Köstler, Markus Stürmer, and Ulrich Rüde. Parallel multigrid on hierarchical hybrid grids: a performance study on current high performance computing clusters. *Concurrency and Computation: Practice and Experience*, 26(1):217–240, 2014.
18. Wolfgang Hackbusch. *Multi-grid methods and applications*, volume 4. Springer, 1985.
19. Wolfgang Hackbusch. *Theorie und Numerik elliptischer Differentialgleichungen: mit Beispielen und Übungsaufgaben*. Teubner, 1996.
20. Ingo Heppner, Michael Lampe, Arne Nägel, Sebastian Reiter, Martin Rupp, Andreas Vogel, and Gabriel Wittum. Software framework ug4: Parallel multigrid on the hermit supercomputer. In *High Performance Computing in Science and Engineering 12*, pages 435–449. Springer, 2013.
21. Roberto Ierusalimschy, Luiz Henrique De Figueiredo, and Waldemar Celes Filho. Lua-an extensible extension language. *Softw., Pract. Exper.*, 26(6):635–652, 1996.
22. Benjamin C. Lee, David M. Brooks, Bronis R. de Supinski, Martin Schulz, Karan Singh, and Sally A. McKee. Methods of inference and learning for performance modeling of parallel applications. In *Proc. of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '07)*, pages 249–258, 2007.
23. A. Nägel, M. Heisig, and G. Wittum. The state of the art in computational modelling of skin permeation. *Advanced Drug Delivery Systems*, 2012.
24. Arne Nägel, Michael Heisig, and Gabriel Wittum. A comparison of two- and three-dimensional models for the simulation of the permeability of human stratum corneum. *European Journal of Pharmaceutics and Biopharmaceutics*, 72(2):332–338, 2009.
25. Fabrizio Petrini, Darren J. Kerbyson, and Scott Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Proc. of the ACM/IEEE Conference on Supercomputing (SC '03)*, page 55, 2003.

26. Richard R. Picard and R. Dennis Cook. Cross-validation of regression models. *Journal of the American Statistical Association*, 79(387):575–583, 1984.
27. Sebastian Reiter. *Efficient algorithms and data structures for the realization of adaptive, hierarchical grids on massively parallel systems*. PhD thesis, University of Frankfurt, Germany, 2014.
28. Sebastian Reiter, Andreas Vogel, Ingo Heppner, Martin Rupp, and Gabriel Wittum. A massively parallel geometric multigrid solver on hierarchically distributed grids. *Comp. Vis. Sci.*, 16(4):151–164, 2013.
29. Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.
30. R.S. Sampath and G. Biros. A parallel geometric multigrid method for finite elements on octree meshes. *SIAM Journal on Scientific Computing*, 32:1361–1392, 2010.
31. Sergei Shudler, Alexandru Calotoiu, Torsten Hoefler, Alexandre Strube, and Felix Wolf. Exascaling your library: Will your implementation meet your expectations? In *Proc. of the International Conference on Supercomputing (ICS), Newport Beach, CA, USA*, pages 1–11. ACM, June 2015.
32. Christian Siebert and Felix Wolf. Parallel sorting with minimal data. In *Recent Advances in the Message Passing Interface*, pages 170–177. Springer, 2011.
33. Kyle L. Spafford and Jeffrey S. Vetter. Aspen: A domain specific language for performance modeling. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 84:1–84:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.
34. Hari Sundar, George Biros, Carsten Burstedde, Johann Rudi, Omar Ghattas, and Georg Stadler. Parallel geometric-algebraic multigrid on unstructured forests of octrees. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 43. IEEE Computer Society Press, 2012.
35. Nathan R. Tallent and Adolfy Hoisie. Palm: easing the burden of analytical performance modeling. In *Proc. of the Inter. Conf. on Supercomputing (ICS)*, pages 221–230, 2014.
36. A. Vogel, S. Reiter, M. Rupp, A. Nägel, and G. Wittum. UG 4: A novel flexible software system for simulating PDE based models on high performance computers. *Comp. Vis. Sci.*, 16(4):165–179, 2013.
37. Andreas Vogel. *Flexible und kombinierbare Implementierung von Finite-Volumen-Verfahren höherer Ordnung mit Anwendungen für die Konvektions-Diffusions-, Navier-Stokes- und Nernst-Planck-Gleichungen sowie dichtegetriebene Grundwasserströmung in porösen Medien*. PhD thesis, Universität Frankfurt am Main, 2014.
38. Andreas Vogel, Alexandru Calotoiu, Alexandre Strube, Sebastian Reiter, Arne Nägel, Felix Wolf, and Gabriel Wittum. 10,000 performance models per minute – scalability of the UG4 simulation framework. In Jesper Larsson Träff, Sascha Hunold, and Francesco Versaci, editors, *Euro-Par 2015: Parallel Processing*, volume 9233 of *Theoretical Computer Science and General Issues*, pages 519–531. Springer International Publishing, 2015.
39. Samuel Williams, Mike Lijewski, Ann Almgren, Brian Van Straalen, Erin Carson, Nicholas Knight, and James Demmel. s-step Krylov subspace methods as bottom solvers for geometric multigrid. In *28th International Parallel and Distributed Processing Symposium*, pages 1149–1158. IEEE, 2014.
40. Felix Wolf, Christian Bischof, Torsten Hoefler, Bernd Mohr, Gabriel Wittum, Alexandru Calotoiu, Christian Iwainsky, Alexandre Strube, and Andreas Vogel. Catwalk: A quick development path for performance models. In *Euro-Par 2014: Parallel Processing Workshops*. Springer, 2014.
41. X. Wu and F. Mueller. ScalaExtrap: trace-based communication extrapolation for SPMD programs. In *Proc. of the 16th ACM symposium on Principles and practice of parallel programming (PPoPP '11)*, pages 113–122, 2011.
42. Jidong Zhai, Wenguang Chen, and Weimin Zheng. Phantom: predicting performance of parallel applications on large-scale parallel machines using a single node. *SIGPLAN Notices*, 45(5):305–314, 2010.