# PetaScale Finite Element Methods for Hierarchical Hybrid Grids using Hybrid Parallelization

Björn Gmeiner, Daniel Ritter, Ulrich Rüde*

In this article we present a performance study of our finite element package Hierarchical Hybrid Grids (HHG) on current European supercomputers. HHG is designed to close the gap between the flexibility of finite elements and the efficiency of geometric multigrid by using a compromise between structured and unstructured grids. A coarse input FE mesh is refined in a structured way, resulting in semi-structured meshes. First we compare and analyze the efficiencies of the stencil-based code on those clusters. In a second step we discuss the $\tau$-extrapolation multigrid technique to improve the FE accuracy for an exemplary electrochemistry application.

## 1 Introduction

In electro-chemistry, *density functional theory* (DFT) plays an important role as a class of models to calculate the electrical potential imposed by the charges of an ensemble of atom nuclei and electrons [1, 2]. One essential step in the DFT is the solution of the potential equation that reduces to Poisson's equation in the case of a homogeneous dielectricity coefficient [3, 4]. However, often effects of an ionic solvent with varying dielectricity cannot be neglected. The governing equation in this case is given as

$$-\nabla \cdot k(x,y,z)\nabla u(x,y,z) = f(x,y,z), \tag{1}$$

where $k$ denotes the dielectricity constant, $u$ the potential field and $f$ the right-hand side. For the sake of simplicity, we assume Dirichlet conditions at the boundaries of the simulation domain $\Omega$.

While the values of the coefficient $k$ at the simulation domain boundaries can be determined by experiments, the exact shape of the transitions at the interfaces within the domain is not known precisely. This fact justifies to study both smooth and non-smooth transition models for the dielectricity coefficient. Note, that a jump in the coefficient leads to a continuous, but non-smooth solution $u(x,y,z)$ of the equation in our application.

In this application, a high accuracy is required, and additionally, the conservation of energy is especially important. The total energy in the system must stay constantly and the shape of the potential has to be exact: Within the overall simulation, not only the potential, but also the forces, i.e. the derivatives of the potential must be approximated as well as possible. In order to meet this requirement, we use a $\tau$-extrapolation enhanced finite element (FE) discretization.

Alternatively, a discretization with compact higher-order finite difference (FD) stencils [5, 6] on a uniform grid could be used. Both methods are advantageous for computations on massively parallel systems, since they maintain a simple communication structure between sub-domains that have been partitioned onto a distributed memory architecture. The analysis of higher-order FD methods for variable coefficient and interface problems is the subject of another submission to the CMCMM 2013. This paper also includes a systematic comparison with the $\tau$-extrapolation based method that is in the focus of the remainder of this paper.

The paper is structured as follows:

- The remaining part of this section introduces the software package HHG (Hierarchical Hybrid Grids), and three peta-scale class HPC systems, as well as the $\tau$-extrapolation technique.

*Chair for Computer Science 10, University of Erlangen-Nürnberg, Germany, bjoern.gmeiner@cs.fau.de

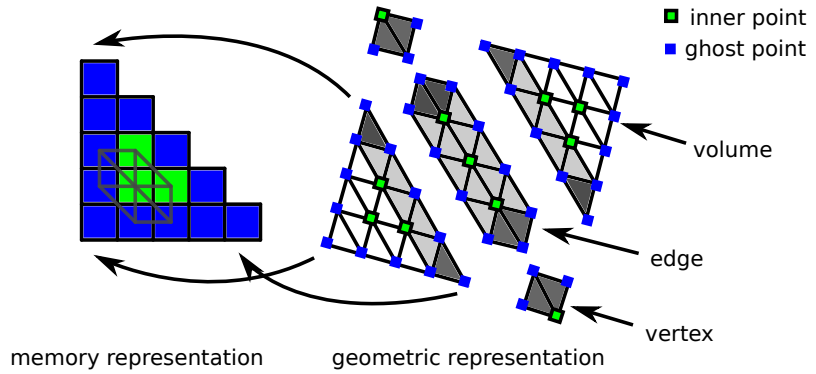


Figure 1: Splitting of two triangle input elements into HHG grid primitives after two steps of refinement. Additionally, the memory representation of a refined triangle with a 7-point stencil for the lower left inner point is sketched.

- The second section describes a novel hybrid parallelization strategy implemented within HHG to allow extreme scale simulations on the clusters JUQUEEN and SuperMUC. Scalability experiments and performance analysis on different clusters are presented.
- In the third section, we perform numerical experiments with $\tau$-extrapolation to improve the order of accuracy of the original basic FE discretization with linear tetrahedral elements for variable and jumping coefficients. Finally, we conclude the paper.

## 1.1 Parallel multigrid with Hierarchical Hybrid Grids

For solving partial differential equations (PDEs), FE methods are a popular discretization scheme, since they allow flexible, unstructured meshes. Applications for FE can be found in a wide range including propagation of sound or heat, electro-chemistry, electrostatics, electrodynamics, fluid flow, geophysics, or elasticity.

The framework HHG [7, 8, 9] is designed to close this gap between the flexibility of the FE method and the superb performance of geometric multigrid [10, 11] by using a compromise between structured and unstructured grids. A coarse input FE mesh is organized into the grid primitives vertices, edges, faces, and volumes. The primitives are then refined in a structured way (see fig. 1), resulting in semi-structured meshes. The regularity of the resulting grids may be exploited in such a way that it is no longer necessary to explicitly assemble the global discretization matrix. In particular, given an appropriate input grid, the discretization matrix may be defined implicitly using stencils for each structured patch. Here a stencil represents a row of the global stiffness matrix. Within HHG, we have implemented an MPI[1]-parallel geometric multigrid method that operates on the resulting block-structured grid hierarchy.

If not stated differently, the settings of the multigrid components and parameters used in this paper are:

- three Gauss-Seidel iterations for pre- and post-smoothing steps,
- linear interpolation,
- six multigrid levels,
- parallel Conjugated Gradient (CG) algorithm to solve the coarsest grid problem,
- and direct coarse grid approximation with coefficient averaging.

The stencils can be stored in registers when the dielectricity is piecewise constant, or it can be assembled on-the-fly for a variable dielectricity. In both cases this results in a so-called *matrix-free* implementation, where the stiffness (or mass) matrix are never explicitly formed. This can have significant performance benefits since it reduces memory traffic, possibly at the expense of redundant computations.

[1] `www.mcs.anl.gov/mpi`

| | JUGENE | JUQUEEN | SuperMUC |
|---|---|---|---|
| System | IBM BlueGene/P | IBM BlueGene/Q | IBM System x iDataPlex |
| Processor | IBM PowerPC 450 | IBM PowerPC A2 | Intel Xeon E5-2680 8C |
| Clock Frequency | 0.85 GHz | 1.6 GHz | 2.8 GHz |
| Number of Nodes | 73 728 | 24 576 | 9 216 |
| Cores per Node | 4 | 16 | 16 |
| HW Threads per Core | 1 | 4 | 2 |
| Memory per HW Thread | 0.5 GB | 0.25 GB | 1 GB |
| Network Topology | 3D Torus | 5D Torus | Tree |
| Gflop/s per Watt | 0.44 | 2.54 | 0.94 |

Table 1: System overview of the supercomputers JUGENE, JUQUEEN, and SuperMUC.

## 1.2 Architectures

Within this article we compare the performance of HHG on three European supercomputers: JUGENE, JUQUEEN are both located at FZ Jülich, and SuperMUC is located in the LRZ supercomputing center in Garching. Tab. 1 presents a system overview of these clusters.

JUGENE, which is not in operation anymore, was the largest BlueGene/P installation with 294 912 compute cores. Each node was equipped with a PowerPC 450 quadcore processor running at a low clock frequency of 850 MHz to achieve a low power consumption. The architecture provided a very high main memory performance, as measured with the stream benchmark. The overall peak performance was 1 petaflop/s. A three-dimensional torus network in combination with a tree-based collective network was available for parallel communication. We were able to solve FE systems with in excess of $10^{12}$ degrees of freedom on JUGENE with the HHG package. We use these three year old performance results obtained on JUGENE as reference for our new results.

The BlueGene/Q system JUQUEEN is the successor of the JUGENE with a peak performance of more than 5.0 petaflop/s. Although the clock-frequency still remains relatively low, it is nearly doubled and the number of processors per node quadrupled. Beneath the 16 cores available for user applications, an additional 17th core is reserved for the operation system. Each core has four hardware (HW) threads that have to be utilized for best execution performance. The memory bandwidth has not scaled up accordingly, but in order to compensate this disadvantage in part, e.g. the prefetching and speculative execution facilities have been improved. The torus network is extended to five-dimensions for shorter paths, and the collective network is now implemented within the torus network. The ratio of peak network bandwidth node performance and peak floating point performance is only 50% of that of BlueGene/P. On the other hand, the cores within each node and consequently the intra-node communication performance has drastically increased.

SuperMUC is a 3.2 petaflop/s IBM x iDataPlex cluster. This machine consists of 18 thin islands, carrying 97.5% of total performance, and one fat island for memory intensive, but only moderately parallel applications. Each thin island is equipped with 512 compute nodes. Two sockets with Sandy Bridge-EP Intel Xeon E5-2680 8C provide 16 physical cores (32 logical cores with Hyperthreading). The Intel Xeon processors deliver a significantly higher core and node performance than the PowerPCs in the IBM architectures, however, at the price of higher power consumption. The nodes within an island are linked by an Infiniband non-blocking tree, whereas a pruned 4:1 tree connects all islands.

## 1.3 $\tau$-extrapolation

The $\tau$-extrapolation (see [11, 12]) is a modification of the multigrid method to improve the convergence order of the solution by an extrapolation process. In contrast to a classical Richardson extrapolation, the extrapolation works not explicitly on the solution, but implicitly on the residual. However, both methods have in common that they are motivated by an expansion of the truncation error. In two dimensions, the $\tau$-extrapolation was applied to the FE framework [13] and to variable coefficient problems [14]. Investigations for one-dimensional linear and non-linear problems and for the Navier-Stokes equations are presented in [15]. Within this work, we implemented $\tau$-extrapolation in parallel into the current version of HHG and use it for 3D tetrahedral elements on block-structured grids.

| MPI-Processes | OpenMP-Threads | Time | Efficiency | |
|---|---|---|---|---|
| 4 096 | 1 | 3.09 | | |
| 2 048 | 2 | 3.10 | 99% | |
| 1 024 | 4 | 3.21 | 96% | |
| 512 | 8 | 3.45 | 90% | |
| 256 | 16 | 3.95 | 78% | |
| 128 | 32 | 4.33 | 71% | |
| 64 | 64 | 5.22 | 59% | |
| 64 | 32 | 5.77 | 54% | 2 Threads/Core |
| 64 | 16 | 8.36 | 37% | 1 Threads/Core |

Table 2: Efficiency of the hybrid parallelization compared to a pure MPI parallel approach on JUQUEEN for moderate problem sizes.

# 2 Porting HHG to BlueGene/Q

For the substantial changes that were necessary to use HHG on more than 30 000 parallel threads, we refer to [16]. This includes the design of data structures for generating tetrahedral input grids efficiently in parallel. However, for the current and upcoming systems, this alone proved not to be sufficient and thus this paper will present a new hybrid parallelization strategy.

## 2.1 Hybrid parallelization

The new system architectures with more powerful and complex compute nodes make a hybrid parallelization approach especially attractive and potentially profitable, since they provide better opportunities for a shared memory parallelization via OpenMP[2]. Thus a hybrid parallelization strategy, including message passing for coarse grain parallelism, and shared memory parallelism within a node for finer scale parallel execution, has been found essential for exploiting the full potential of architectures like JUQUEEN or SuperMUC.

- In a pure MPI parallel setting, the available main memory per process is only 256 MB per process on JUQUEEN. This is too small for the three largest runs described in the next sections. In contrast, a hybrid parallelization increases the available main memory for each process.
- On SuperMUC, the scaling breaks down when too many MPI processes are being used. Here, a hybrid parallelization helps to limit the total number of MPI processes and this helps to maintain scalability when going to extreme size simulations

The current OpenMP implementation in HHG supports parallelism inside kernel executions and copy of ghost layers on several primitives. However, the MPI instructions are executed asynchronously, but not explicitly OpenMP-parallel. Further, OpenMP introduces an additional overhead for spawning threads, which is especially critical on the coarsest grids, where the workloads per thread is small. The quality of the MPI/OpenMP parallel execution is reflected in tab. 2. All runs up to the last two are executed by four threads per compute core. The timings conclude that serial fraction of the code is still between $1 - 2\%$. We will use a hybrid parallelization with up to eight OpenMP-threads for the largest parallel run on JUQUEEN in the following scaling experiment as the performance loss is still not too high.

## 2.2 Weak scaling on JUQUEEN

This section shows the scalability of the HHG approach on a current cluster. The program is compiled with the IBM XL compiler suite on both BlueGene clusters. Only basic optimization levels (*-O2* or *-O3 -qstrict*) are possible to maintain correct program execution. More aggressive optimization (*-qhot*) causes a performance drop of around a factor two.

As a test case we use a piecewise constant dielectricity, and thus can use constant stencils within each HHG block and each geometric primitive. Consequently, the numerical efficiency is extremely high and in a relative sense, the communication is very intensive. Therefore, this is quite

[2] `www.openmp.org`

| Number of Threads | Number of Unknowns | Time per V-cycle | Number of Threads | Number of Unknowns | Time per V-cycle |
|---|---|---|---|---|---|
| 64 | $1.33 \cdot 10^8$ | 2.34 s | 16 384 | $3.43 \cdot 10^{10}$ | 3.15 s |
| 128 | $2.67 \cdot 10^8$ | 2.41 s | 32 768 | $6.87 \cdot 10^{10}$ | 3.28 s |
| 256 | $5.35 \cdot 10^8$ | 2.80 s | 65 536 | $1.37 \cdot 10^{11}$ | 3.39 s |
| 512 | $1.07 \cdot 10^9$ | 2.82 s | 131 072 | $2.75 \cdot 10^{11}$ | 3.56 s |
| 1 024 | $2.14 \cdot 10^9$ | 2.82 s | 262 144 | $5.50 \cdot 10^{11}$ | 3.68 s |
| 2 048 | $4.29 \cdot 10^9$ | 2.84 s | 524 288 | $1.10 \cdot 10^{12}$ | 3.76 s |
| 4 096 | $8.58 \cdot 10^9$ | 2.96 s | 1 048 576 | $2.20 \cdot 10^{12}$ | 4.07 s |
| 8 192 | $1.72 \cdot 10^{10}$ | 3.09 s | 1 572 864 | $3.29 \cdot 10^{12}$ | 4.03 s |

Table 3: Weak scaling experiment on JUQUEEN solving a problem on the full machine with up to $3.29 \cdot 10^{12}$ unknowns.

a challenging setup for maintaining the parallel scalability as we will show in the performance study in the next section. Tab. 3 shows the run-time results of a scaling experiment. The smallest test run already solves a system of slightly more than $10^8$ unknowns and one V-cycle takes approximately 2.3 seconds. Note that this is performed on a single compute node on JUQUEEN, demonstrating the high efficiency of the HHG approach.

In each further row of the table, the problem size is doubled as well as the number of nodes. This is a classical weak scalability test. The full machine could eventually solve a linear system with $3.3 \cdot 10^{12}$ unknowns, corresponding to more than $10^{13}$ tetrahedral finite elements. In total, this computation uses 300, out of the almost 400 terabytes of main memory during the solution process.

Four hardware threads are necessary to saturate the performance of one processor core, leading to a parallel execution of more than one million threads. Although, the computational time increases only moderately, we note that the coarse grid solver is only a straightforward CG iteration. Therefore in large runs, more than half a second of the V-cycle execution time is spent in the increasing number of CG iterations on the coarsest grid, that is caused by larger and larger coarse grids. This shows clearly, that for perfect asymptotic scalability a better coarse grid solver would be necessary. Nevertheless, we believe that our results with the CG solver indicate clearly that the coarse grid solver performance is not as critical for scalability, as has has been discussed in the older literature on parallel multigrid methods. In our experience, a careful and well-designed implementation of the communication routines can reduce the negative effect of the coarse grid bottleneck significantly. Based on these results, we see no reason why multigrid should be considered a-priory inferior to other iterative methods in terms of parallel efficiency.

## 2.3 Comparison with other peta-scale clusters and discussion of scalability results

In this section, we compare the parallel efficiency of our code on different HPC clusters. In contrast to the BlueGene systems, the program is compiled with the Intel compiler suite and IBM MPI for SuperMUC with *-O3 -xavx* compiler flags. As reference, one V-cycle takes 4.25 s for JUGENE, and 1.18 s on SuperMUC on one compute node.

Fig. 2 shows strong efficiency drops when advancing from one node to several nodes. This is especially prominent on both BlueGene systems. However, form then onwards to larger parallel runs, the parallel efficiency stays nearly constant. Only the transition from a single Midplane on BlueGene/P, or one Node Card on BlueGene/Q to larger subportions of the architecture, induce again more significant performance drops.

On SuperMUC the efficiencies up to a $\frac{1}{4}$ island ($2.6 \cdot 10^{10}$ unknowns) differ between the multigrid cycles. We believe that this is caused by perturbations due to other applications running simultaneously on the same island. From $\frac{1}{4}$ of an island to $\frac{1}{2}$ of an island ($5.2 \cdot 10^{10}$ unknowns) the performance even improves. However, when leaving a single island of the architecture, the parallel efficiency drops significantly. This is likely caused by the reduced communication performance beyond each island in the pruned 4:1 tree. For more than two islands we also disable hyperthreading to obtain substantially more reproducible run-times. In contrast to this observation, the run-times of both BlueGene machines remain more stable for all problem sizes. The parallel efficiency is

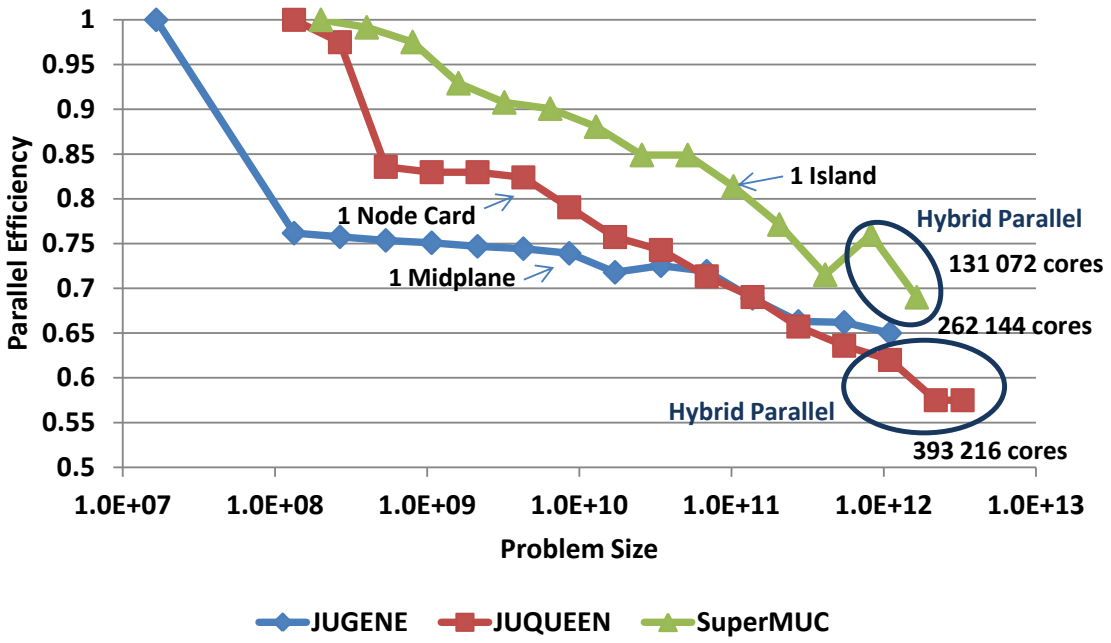


Figure 2: Parallel efficiencies on different supercomputers in a weak scaling on three different supercomputers. The largest runs required a hybrid parallelization strategy. Some hardware structures (node card, midplane, island) of the clusters can be identified by the gradient of the parallel efficiency.

comparable to be best results on SuperMUC. However, we could not map our mesh onto the torus networks, since the coarsest mesh is basically unstructured.

First scaling experiments on SuperMUC showed a breakdown at 65 536 MPI processes, resulting in roughly four times longer run-times, as well as fluctuations in the timings of up to 15 seconds between the single V-cycles compared to runs with 32 768 MPI processes. Fig. 2 displays that there have already been problems on 32 768 MPI processes (corresponding to $4.1 \cdot 10^{11}$ unknowns or 4 islands). The results on larger machine sizes use the hybrid parallelization that allows us to execute the largest two runs with only 16 384 MPI processes, leading to a significantly improved parallel efficiency. The largest run was carried out on 16 islands of the cluster.

Different from the behavior of SuperMUC, the hybrid parallelization on JUQUEEN, as used for the largest runs, clearly decreases the parallel efficiency. However a hybrid parallelization is still necessary as explained above in order to have enough main memory available.

Tab. 4 shows the single node performance, parallel efficiencies, and energy consumptions relatively to JUGENE. The runs were carried out a node allocation providing $\approx 0.8$ Pflop/s nominal

| | JUGENE | JUQUEEN | SuperMUC |
|---|---|---|---|
| **Single Node** | | | |
| Peak flop/s (constant dielectricity) | 6% | 7% | 12% |
| Peak flop/s (variable dielectricity) | 9% | 10% | 13% |
| Peak bandwidth (constant dielectricity) | 11% | 53% | 60% |
| **Parallel Efficiencies** (at $\approx 0.8$ Pflop peak) | | | |
| Scaling (constant dielectricity) | 65% | 64% | 72% |
| Scaling (variable dielectricity) | 94% | 93% | 96% |
| Scaling – without CG (constant dielectricity) | 75% | 70% | 79% |
| Number of processes | 262 144 | 262 144 | 32 768 |
| Energy improvement compared to JUGENE (constant dielectricity) | 1 | 6.6 | 4.7 |
| Energy improvement compared to JUGENE (variable dielectricity) | 1 | 6.4 | 3.2 |

Table 4: Single node and parallel efficiencies (scaling), as well as power consumptions of used parts of the clusters while running HHG.

peak. Even though a major design goal of the BlueGene/P was to have a low energy consumption, the next generation could improve the energy consumption by a factor between six to seven for our application. The SuperMUC turns out not to be as energy efficient as JUQUEEN, however does not require such a high degree of parallelism from the application.

### 2.4 Single node performance analysis

This section will analyze the single node performance as given in tab. 4. This is for the case of constant dielectricity.

On JUGENE one MPI process is assigned to each compute core. Since the processors provide high memory bandwidth, codes tend to be more limited by instruction throughput than by memory bandwidth. However, the kernel that applies the stencil is affected on JUGENE from a serialization within the PowerPC multiply-add instructions. Additionally, a correct memory alignment for vectroized loads (for the SIMD units) is not assured due to the varying loop sizes that are caused by the tetrahedral macro elements. The limited-issue width and the in-order architecture of the processor is leading to further performance limitations and eventually result in a node performance of only 6.1% of the peak performance (see tab. 4). This is for a complete multigrid cycle.

For a comparison, we refer to results of Datta et. al. who present auto-tuning results for an averaging 29-point stencil on this architecture [17, 18]. Their baseline implementation achieves about 0.035 GStencil/s updates which corresponds to 7.7% of the peak performance for a reference (in-cache) implementation. Basically two of eleven optimization techniques (including e.g. padding, core blocking, software pre-fetching) techniques can achieve a significant speedup. These are common subexpression elimination and register blocking. While the first inherently cannot be applied for our stencil, since we do not have redundant calculations, the register blocking results in our case roughly in a speedup of two. In principle we could use this code this optimization, but it leads to very small sub-blocks that will suffer from non-constant loop sizes. Moreover, the issue with serialization remains a bottleneck, which is is not the case for the averaging stencil.

On JUQUEEN, we assign one MPI process to each thread (64 per node). The stream benchmark shows that it is possible to run at a high fraction of $\approx 85\%$ of the effective maximal memory bandwidth of 27.8 $GB/s$ by using one process per node. Two or four threads per node saturate the effective bandwidth completely. Going from one to two threads per core, HHG features a factor of two improvement in performance. In these cases, the code is still instruction bound like on BlueGene/P. Going from two to four threads per core, the additional speedup is only a factor 1.3. Overall, in this case, a multigrid cycle utilizes in average about 18.1 $GB/s$ of the main memory bandwidth. Only by reducing the main memory footprint and possibly improving the the core performance itself, we see a chance for further reductions of the execution time.

Similarly to the situation on JUQUEEN, the code is mainly memory bandwidth limited on the SuperMUC node architecture. However, the nodes can saturate the bandwidth better and its machine balance suits better to the characteristics of our code. Thus we can achieve with a somewhat better flop/s performance than on JUQUEEN. However, hyperthreading for single node improves the performance only insignificantly by at most a few percent.

## 3 Example application motivated by ab-initio molecular dynamics

In the following, we describe a numerical example computation originating from a molecular dynamics simulation. We use this application to evaluate and discuss the convergence features of our method. Since our goal is to study the approximation order of our discretization, we choose a test problem that is complex enough to exhibit many of the relevant characteristics of the actual application. However, it is constructed such that there exists an analytical solution.

### 3.1 Smooth, variable dielectricity

In the first experiment, the dielectricity is chosen variable, i.e.:

$$k(x, y, z) = \sin(6x)\sin(6y)\sin(6z) + 2. \tag{2}$$

| Resolution | Error (inf.) without $\tau$-extr. | Error (inf.) with $\tau$-extr. | Consistency without $\tau$-extr. | Consistency with $\tau$-extr. |
|---|---|---|---|---|
| $9^3$ | $1.27 \cdot 10^{-1}$ | $7.35 \cdot 10^{-2}$ | | |
| $17^3$ | $3.50 \cdot 10^{-2}$ | $7.80 \cdot 10^{-3}$ | 1.86 | 3.24 |
| $33^3$ | $9.08 \cdot 10^{-3}$ | $4.20 \cdot 10^{-4}$ | 1.95 | 4.21 |
| $65^3$ | $2.32 \cdot 10^{-3}$ | $3.09 \cdot 10^{-5}$ | 1.97 | 3.77 |
| $129^3$ | $5.80 \cdot 10^{-4}$ | $2.04 \cdot 10^{-6}$ | 2.00 | 3.92 |

Table 5: Error and order of consistency with and without $\tau$-extrapolation for a variable coefficient problem.

The offset excludes a zero value in the coefficient $k$. The solution $u(x, y, z)$ and corresponding right hand side $f(x, y, z)$ are:

$$u(x, y, z) = \sin(6x)\sin(6y)\sin(6z), \tag{3}$$

and

$$\begin{aligned} f(x, y, z) = &-36\cos^2(6x)\sin^2(6y)\sin^2(6z) \\ &+ 108(\sin(6x)\sin(6y)\sin(6z) + 2)\sin(6x)\sin(6y)\sin(6z) \\ &- 36\sin^2(6x)\cos^2(6y)\sin^2(6z) - 36\sin^2(6x)\sin^2(6y)\cos^2(6z). \end{aligned} \tag{4}$$

We consider the cubic domain $\Omega = (0, 1) \times (0, 1) \times (0, 1)$. For the right hand side $f$ a quadratic quadrature rule according to [19] is used. The element-wise integrals over the coefficients $k$ for the operator are evaluated with a lower approximation formula at the barycentric coordinates $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ with weight $\frac{1}{6}$. We use no post-smoothing steps to keep the improved solution on the finest grid, which is done typically for $\tau$-extrapolation. The coefficients $k$ are linearly averaged for a direct coarse grid correction. Tab. 5 shows that the consistency is enhanced from second to fourth order by $\tau$-extrapolation.

## 3.2 Jumping, piecewise constant dielectricity

In contrast to the last paragraph, we assume that

$$k(x, y, z) = \begin{cases} k_1 = 1, & x < 0, \\ k_2 = 100, & x \geq 0. \end{cases} \tag{5}$$

In a typical setup from molecular dynamics (see [2]), the jump in $k$ is aligned with one of the coordinate axes which simplifies the difficulty of the problem significantly. Therefore, we directly inject the coefficients $k$ for $\tau$-extrapolation. We consider the cubic domain $\Omega = (-0.5, 0.5) \times (0, 1) \times (0, 1)$, and harmonic functions (see fig. 3)

$$u(x, y, z) = (a \cdot \sinh(\pi\sqrt{2}x) + b \cdot \cosh(\pi\sqrt{2}x)) \cdot \sin(\pi y) \cdot \sin(\pi z). \tag{6}$$

Clearly, these functions satisfy the Laplace equation $\Delta u = 0$ and they satisfy homogeneous Dirichlet boundary conditions on four of the six faces of $\partial\Omega$, i.e. we have $u = 0$ for $x = 0, 1$ and for $y = 0, 1$. Noting that

$$\sinh(\pi\sqrt{2}(x + 1)) = \cosh(\sqrt{2}\pi)\sinh(\sqrt{2}\pi x) + \sinh(\sqrt{2}\pi)\cosh(\sqrt{2}\pi x) \tag{7}$$

we now choose in the left half of $\Omega$ for $x < 0$ a solution in the representation

$$u(x, y, z) = \sinh(\sqrt{2}\pi(x + 1)) \cdot \sin(\pi y) \cdot \sin(\pi z). \tag{8}$$

In the right hand half of $\Omega$, for $x \geq 0$ we use the form of eq. (6) and determine the parameters $a$ and $b$ by requiring continuity of $u$ across the interface and the continuity of $k\vec{j} \cdot \vec{n}$. This leads to the conditions

$$b = \sinh(\sqrt{2}\pi), \quad a = \frac{\sqrt{2}\pi\cosh(\sqrt{2}\pi)}{100\sqrt{2}\pi\cosh(\sqrt{2}\pi 0)} = \frac{\cosh(\sqrt{2}\pi)}{100}. \tag{9}$$

Similarly to the variable dielecticity test case, tab. 6 shows that the consistency is enhanced to fourth order by $\tau$-extrapolation.

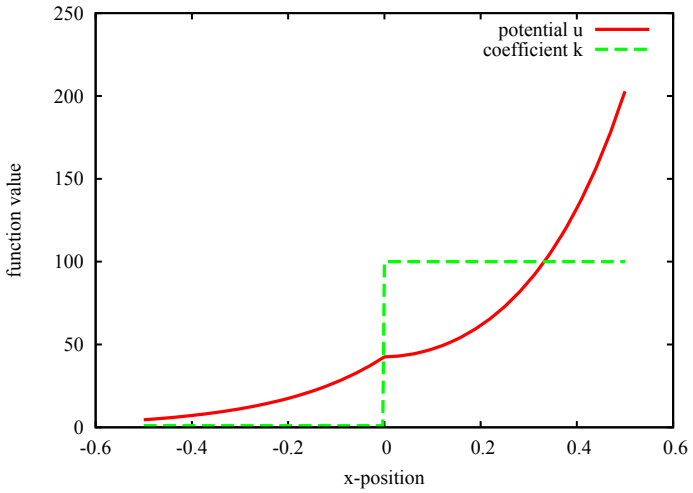


Figure 3: Illustration of the solution $u$ and the coefficient $k$ of the test problem along the line $(y, z) = (0.5, 0.5)$.

| Resolution | Error (inf.) without $\tau$-extr. | Error (inf.) with $\tau$-extr. | Consistency without $\tau$-extr. | Consistency with $\tau$-extr. |
|---|---|---|---|---|
| $9^3$ | $1.77 \cdot 10^{0}$ | $6.41 \cdot 10^{-1}$ | | |
| $17^3$ | $4.48 \cdot 10^{-1}$ | $5.53 \cdot 10^{-2}$ | 1.98 | 3.53 |
| $33^3$ | $1.13 \cdot 10^{-1}$ | $4.00 \cdot 10^{-3}$ | 1.99 | 3.79 |
| $65^3$ | $2.82 \cdot 10^{-2}$ | $2.69 \cdot 10^{-4}$ | 2.00 | 3.90 |
| $129^3$ | $7.05 \cdot 10^{-3}$ | $1.73 \cdot 10^{-5}$ | 2.00 | 3.96 |

Table 6: Error and order of consistency with and without $\tau$-extrapolation for a jumping piecewise constant coefficient problem.

## 4 Conclusion and future work

We presented a weak-scaling comparison of HHG on three different HPC petaflop clusters. To reach the full potential of the recent architectures, a hybrid parallelization approach turned out to be necessary for the growing node-level parallelism to resolve memory limitations and maintain scalability. In addition to improve the resolution by increasing the number of elements, $\tau$-extrapolation shows up as a HPC-confirm MG method to enhance the approximation order for variable and jumping coefficient test-cases.

In a next step, we plan to compare $\tau$-extrapolation and compact higher-order finite differences in terms of accuracy and computational cost.

## References

[1] J.-L. Fattebert and F. Gygi. Density functional theory for efficient ab initio molecular dynamics simulations in solution. *J Comput Chem*, 223:662–666, 2002.

[2] V. Sanchez, M. Sued, and D. Scherlis. First-principles molecular dynamics simulations at solid-liquid interfaces with a continuum solvent. *J Chem Phys*, 131(17):174108, 2009.

[3] H. Köstler, R. Schmid, U. Rüde, and C. Scheit. A parallel mutigrid accelerated Poisson solver for ab initio molecular dynamics applications. *Computing and Visualization in Science*, 11(2):115–122, 2008.

[4] R. Schmid, M. Tafipolsky, P. H. König, and H. Köstler. Car-Parrinello molecular dynamics using real space wavefunctions. *physica status solidi (b)*, 243, Issue 5:1001–1015, 2006.

[5] L. Collatz. *The Numerical Treatment of Differential Equations*. Springer Verlag, Berlin, 3rd edition, 1966.

[6] Z. Li and K. Ito. *The Immersed Interface Method: Numerical Solutions of PDEs Involving Interfaces and Irregular Domains*. Frontiers in applied mathematics. SIAM, Philadelphia, 2006.

[7] B. Bergen, T. Gradl, F. Hülsemann, and U. Rüde. A massively parallel multigrid method for finite elements. *Computing in Science and Engineering*, 8(6):56–62, 2006.

[8] T. Gradl, C. Freundl, H. Köstler, and U. Rüde. Scalable Multigrid. In S. Wagner, M. Steinmetz, A. Bode, and M. Brehm, editors, *High Performance Computing in Science and Engineering. Garching/Munich 2007*, pages 475–483. LRZ, KONWIHR, Springer-Verlag, Berlin, Heidelberg, New York, 2008.

[9] B. Gmeiner, T. Gradl, H. Köstler, and U. Rüde. Highly parallel geometric multigrid algorithm for hierarchical hybrid grids. In K. Binder, G. Münster, and M. Kremer, editors, *NIC Symposium 2012*, volume 45 of *Publication series of the John von Neumann Institute for Computing*, pages 323–330, Jülich (Germany), February 2012.

[10] A. Brandt. Multi-Level Adaptive Solutions to Boundary-Value Problems. *Mathematics of Computation*, 31(138):333–390, 1977.

[11] W. Hackbusch. *Multi-Grid Methods and Applications*. Springer-Verlag, Berlin, Heidelberg, New York, 1985.

[12] A. Brandt. Multigrid techniques: 1984 guide with applications to fluid dynamics. *GMD-Studie Nr. 85, Sankt Augustin, West Germany*, 1984.

[13] M. Jung and U. Rüde. Implicit extrapolation methods for multilevel finite element computations. *SIAM Journal on Scientific Computing*, 17(1):156–179, 1996.

[14] M. Jung and U. Rüde. Implicit extrapolation methods for variable coefficient problems. *SIAM Journal on Scientific Computing*, 19(4):1109–1124, 1998.

[15] K. Bernert. $\tau$-extrapolation—theoretical foundation, numerical experiment, and application to Navier–Stokes equations. *SIAM Journal on Scientific Computing*, 18(2):460–478, 1997.

[16] 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*, 2012. Accepted.

[17] K. Datta and K.A. Yelick. *Auto-tuning stencil codes for cache-based multicore platforms*. PhD thesis, University of California, Berkeley, 2009.

[18] K. Datta, S. Williams, V. Volkov, J. Carter, L. Oliker, J. Shalf, and K. Yelick. Auto-tuning the 27-point stencil for multicore. In *In Proc. iWAPT2009: The Fourth International Workshop on Automatic Performance Tuning*, 2009.

[19] VV Shaæidurov. *Multigrid methods for finite elements*, volume 318. Kluwer Academic Publishers (Dordrecht and Boston), 1995.