Peta-Scale Hierarchical Hybrid Multigrid Using Hybrid Parallelization

Björn Gmeiner(✉) and Ulrich Rüde

Department of Computer Science 10,
University of Erlangen-Nürnberg, Erlangen, Germany
{bjoern.gmeiner,ulrich.ruede}@fau.de

Abstract. 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 finite element mesh is refined in a structured way, resulting in semi-structured meshes. Within this article we compare and analyze the efficiencies of the stencil-based code on those clusters.

Keywords: Parallel multigrid · Performance analysis · HHG

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 [7,11]. 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 [12]. 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 Ω. 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. 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.

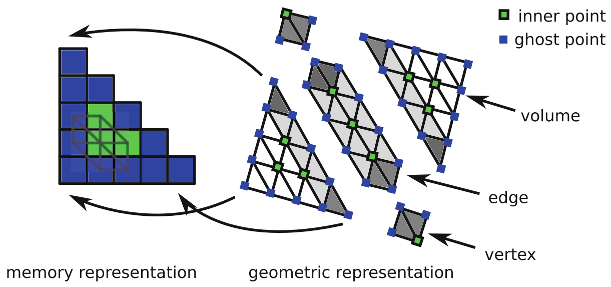

Fig. 1. Splitting of two triangle input elements into HHG grid primitives after two steps of refinement. Additionally, the memory representation of a refined triangle with a 7-point stencil for the lower left inner point is sketched [9].

1.1 Parallel Multigrid with Hierarchical Hybrid Grids

For solving partial differential equations (PDEs), finite elements (FE) methods are a popular discretization scheme, since they allow flexible, unstructured meshes. The framework HHG [2,8] is designed to combine the flexibility of the FE method and the superb performance of geometric multigrid [3,10] 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. 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 between six multigrid levels, parallel Conjugated Gradient 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. This can have significant performance benefits since it reduces memory traffic, possibly at the expense of redundant computations.

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

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

Table 1. System overview of the IBM clusters JUGENE, JUQUEEN, and SuperMUC.

	JUGENE	JUQUEEN	SuperMUC
System	BlueGene/P	BlueGene/Q	System x iDataPlex
Processor	IBM PowerPC 450	IBM PowerPC A2	Intel Xeon E5-2680
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

SuperMUC is located in the LRZ supercomputing center in Garching. Table 1 presents a system overview of these clusters.

JUGENE, was the largest BlueGene/P installation with 294 912 compute cores and 1 petaflop/s peak performance. Each node was equipped with a PowerPC 450 quadcore processor running at a low clock frequency. The architecture provided a very high main memory performance. A three-dimensional torus network in combination with a tree-based collective network was available for 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 5.9 petaflop/s. Although the clock-frequency still remains relatively low, it is nearly doubled. Each of the 16 cores available for user applications has four hardware (HW) threads. 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 was fused into 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 moderately parallel, memory intensive 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. The Xeon processors deliver a significantly higher core and node performance than the PowerPCs in the IBM architectures, 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.

2 Porting Hierarchical Hybrid Grids to BlueGene/Q

For the substantial changes that were necessary to use HHG on more than 30 000 parallel threads, we refer to [9]. 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.

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. A hybrid parallelization helps to limit the total number of MPI processes and this helps to maintain scalability for 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 Table 2. All runs up to the last two are executed by four threads per compute core. From the timings we 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.

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

MPI processes	OpenMP threads	Runtime	Efficiency (%)	MPI processes	OpenMP threads	Runtime	Efficiency (%)
4 096	1	3.09		64	64	5.22	59
2 048	2	3.10	99				
1 024	4	3.21	96	2 Threads/Core:			
512	8	3.45	90	64	32	5.77	54
256	16	3.95	78	1 Threads/Core:			
128	32	4.33	71	64	16	8.36	37

[2] www.openmp.org

Table 3. Weak scaling experiment on JUQUEEN solving a problem on the full machine

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

2.1 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. 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 a challenging setup for maintaining the parallel scalability as we will show in the performance study in the next section. Table 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 s. 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 TB 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 Conjugated Gradient (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 been discussed in the older literature on parallel multigrid methods.

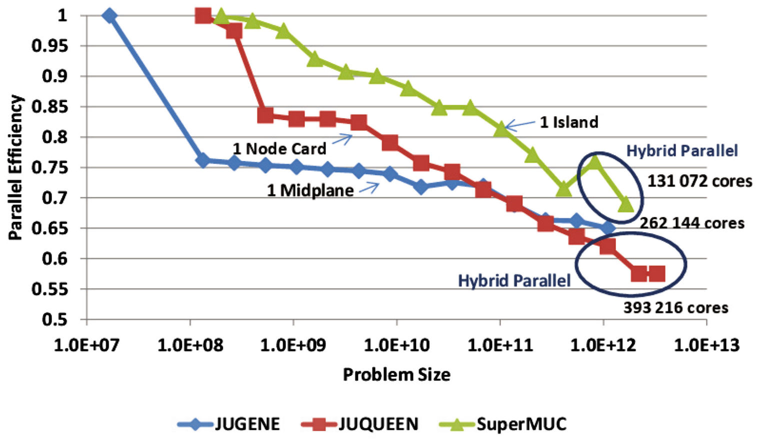

Fig. 2. Parallel efficiencies on different supercomputers in a weak scaling on three different supercomputers. The largest runs required a hybrid parallelization strategy. Some hardware structures (node card, midplane, island) of the clusters can be identified by the gradient of the parallel efficiency.

2.2 Comparison of Scalability Results on Other Peta-scale Clusters

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. Figure 2 shows strong efficiency drops when advancing from one node to several nodes. This is especially prominent on both BlueGene systems. However, from there 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 sub-portions of the architecture, induce again more significant performance drops. On SuperMUC the efficiencies up to a quarter 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 quarter of an island to half 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. SuperMUC presents the best parallel efficiency. 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

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

	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 (const.)	1	6.6	4.7
Energy improvement compared to JUGENE (var.)	1	6.4	3.2

in the timings of up to 15 s between the single V-cycles compared to runs with 32 768 MPI processes. Figure 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.

Table 4 shows the single node performance, parallel efficiencies, and energy consumptions relatively to JUGENE. The runs were carried out for a node allocation providing $\approx$ 0.8 Pflop/s nominal 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 it does not require such a high degree of parallelism from the application.

2.3 Single Node Performance Analysis

This section will analyze the single node performance as given in Table 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 vectorized 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 leads to further performance limitations and eventually result in a node performance of only 6.1 % of the peak performance (see Table 4). This is for a complete multigrid cycle.

For a comparison, we refer to results of Datta et al. who present autotuning results for an averaging 29-point stencil on this architecture [5,6]. 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 (e.g. padding, core blocking, software pre-fetching) techniques can achieve a significant speedup: 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 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 not the case for the averaging stencil.

On JUQUEEN, we assign one MPI process to each HW thread. 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 gives 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 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 the characteristics of our code. Thus we achieve with a better flop/s performance than on JUQUEEN. However, hyperthreading improves the performance only insignificantly by at most a few percent.

3 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 compensate memory limitations and maintain scalability. Recently, we designed a Stokes solver for Earth mantle convection simulations [1,4] within HHG utilizing the presented multigrid performance.

Acknowledgements. The work was supported by the International Doctorate Program (IDK) within the Elite Network of Bavaria. The authors gratefully acknowledge the Gauss Centre for Supercomputing (GCS) for providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS share of the supercomputer JUQUEEN at Jülich Supercomputing Centre (JSC). GCS is the alliance of the three national supercomputing centres HLRS (Universität Stuttgart), JSC (Forschungszentrum Jülich), and LRZ (Bayerische Akademie der Wissenschaften), funded by the German Federal Ministry of Education and Research (BMBF) and the German State Ministries for Research of Baden-Württemberg (MWK), Bayern (StMWFK) and Nordrhein-Westfalen (MIWF).

References

1. Baumgardner, J.R.: Three-dimensional treatment of convective flow in the earth's mantle. J. Stat. Phys. **39**(5/6), 501–511 (1985)
2. Bergen, B., Gradl, T., Hülsemann, F., Rüde, U.: A massively parallel multigrid method for finite elements. Comput. Sci. Eng. **8**(6), 56–62 (2006)
3. Brandt, A.: Multi-level adaptive solutions to boundary-value problems. Math. Comput. **31**(138), 333–390 (1977)
4. Burstedde, C., Ghattas, O., Gurnis, M., Stadler, G., Tan, E., Tu, T., Wilcox, L.C., Zhong, S.: Scalable adaptive mantle convection simulation on petascale supercomputers. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, p. 62. IEEE Press (2008)
5. Datta, K., Williams, S., Volkov, V., Carter, J., Oliker, L., Shalf, J., Yelick, K.: Auto-tuning the 27-point stencil for multicore. In: Proceedings of the Fourth International Workshop on Automatic Performance Tuning (iWAPT2009) (2009)
6. Datta, K., Yelick, K.: Auto-tuning stencil codes for cache-based multicore platforms. Ph.D. thesis, University of California, Berkeley (2009)
7. Fattebert, J.L., Gygi, F.: Density functional theory for efficient ab initio molecular dynamics simulations in solution. J. Comput. Chem. **223**, 662–666 (2002)
8. Gmeiner, B., Gradl, T., Köstler, H., Rüde, U.: Highly parallel geometric multigrid algorithm for hierarchical hybrid grids. In: Binder, K., Münster, G., Kremer, M. (eds.) NIC Symposium 2012. Publication series of the John von Neumann Institute for Computing, vol. 45, pp. 323–330. Jülich (2012)
9. Gmeiner, B., Köstler, H., Stürmer, M., Rüde, U.: Parallel multigrid on hierarchical hybrid grids: a performance study on current high performance computing clusters. Concurr. Comput. Pract. Exp. (2012). http://dx.doi.org/10.1002/cpe.2968
10. Hackbusch, W.: Multi-Grid Methods and Applications. Springer, Heidelberg (1985)
11. Sanchez, V., Sued, M., Scherlis, D.: First-principles molecular dynamics simulations at solid-liquid interfaces with a continuum solvent. J. Chem. Phys. **131**(17), 174108 (2009)
12. Schmid, R., Tafipolsky, M., König, P.H., Köstler, H.: Car-Parrinello molecular dynamics using real space wavefunctions. Phys. Status Solidi B **243**(5), 1001–1015 (2006)