FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
INSTITUT FÜR INFORMATIK (MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG)

Lehrstuhl für Informatik 10 (Systemsimulation)

Analysis of a flat Highly Parallel Geometric Multigrid Algorithm for Hierarchical Hybrid Grids

Björn Gmeiner, Tobias Gradl, Harald Köstler, Ulrich Rüde

Technical Report 03-2011

Analysis of a flat Highly Parallel Geometric Multigrid Algorithm for Hierarchical Hybrid Grids

Björn Gmeiner, Tobias Gradl, Harald Köstler, Ulrich Rüde

April 19, 2011

Abstract

While multicore architectures are becoming usual on desktop machines, supercomputers are approaching million cores. The amount of memory and compute power on current clusters enable us e.g. to obtain a resolution of in excess $(10\,000)^3=10^{12}$ degrees of freedom. However, on the downside we are forced to partition our domain into extremely many sub-problems. Portions of the algorithm that do not permit such degrees of parallelism can easily become a bottleneck. Additionally the performance analysis and debugging of programs at this scale become challenging tasks in themselves. We present scaling results of a quite flat multigrid algorithm (MG) for Hierarchical Hybrid Grids (HHG) and discuss its performance.

1 Introduction

A major aim of our paper is to show that it is still possible to design a relatively flat multigrid algorithm on current highly parallel supercomputers.

Thus the next section introduces the problems of the coarsest grids and ways to treat it. Further some parallel issues, which arised in our implementations are discussed.

The third section covers the single-core performance of our framework. We show and analyze the impact of the single multigrid components of our discretization by comparing predicted work with measurements.

Section four presents strong and weak scaling results. In the strong scaling, we try to push our MG to its limits on the underlying computer architecture and recieved a parallel speedup of 50 on already high core numbers. The weak scaling shows a good parallel efficiency to 292 912 cores, while the coarse grid overhead stays resonable.

Finally we will conclude the paper.

1.1 Multigrid Algorithm

Discretizing a differential equation with finite elements leads to a huge system of equations. An important property of the system matrix is its sparse form. The sparse pattern reduces the number of operations *per iteration* over the domain for iterative solvers to a complexity of O(N) for N unknowns. However, an increasing number of iterations are required for solvers like Gauss-Seidel (GS) or Conjugated Gradient (CG) with growing problem size. Multigrid is a strategy to change this behaviour, so that the number of operations to solve a system is linear dependent on the number of unknowns. This allows solving large systems in reasonable time. We will give a quick overview of the multigrid method. For a full introduction, we refer the reader to [2, 5, 6, 7].

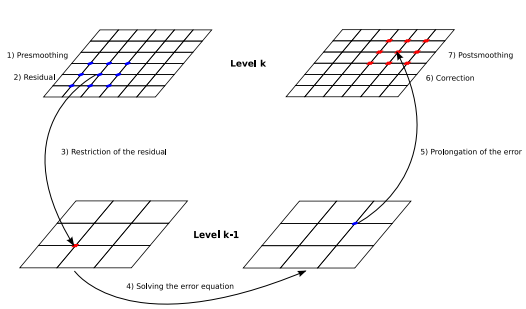
An observation of local acting iterative smoothers, like Gauss-Seidel or Jacobi solvers, is that they can smooth high error frequencies very well. However, low error frequencies are reduced extremly slowly. To resolve this shortcoming, multigrid uses a second effect: Let us consider a function on a fine grid, which has low frequencies. When we transfer this function to a coarser

grid, the frequencies of the function increase with respect to the meshsize. Thus we have changed the frequency of a function by changing its discretization. In multigrid this idea is applied to low frequencies of the error by transfering the error to coarser grids.

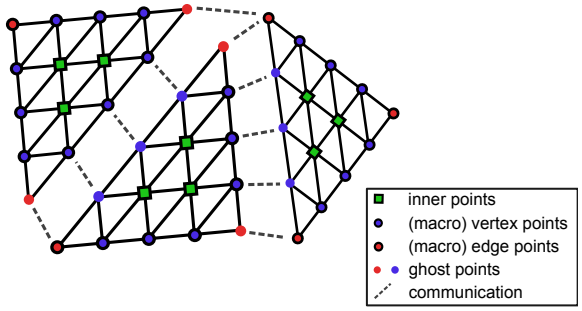
For the linear case we can use the *error equation* (1) to calculate the error e_k on the coarser grids, where N_k is the discretization matrix, r_k the residual, f_k the right hand side, and v_k the approximated solution. The subindex kl denotes the level of the grid. For a two-grid case k is the fine grid and $k-1$ is the coarse grid. Afterwards the solution is corrected by the error on the fine grid.

$$N_k e_k = r_k = f_k - N_k v_k \tag{1}$$

Figure 1: HHG space partitioning and Geometric multigrid.



(a) Two-grid cycle.



(b) Space partitioning in HHG Primitives.

With these thoughts we can roughly define a two-grid cycle starting with a start solution v_k on the fine grid (see Figure 1 (a)):

1. Apply some steps of an iterative smoother to v_k in order to reduce the high frequencies on the fine grid:

$$v_k \leftarrow S(v_k)$$

2. Compute the residual:

$$r_k = f_k - L_k v_k$$

3. Transfer the residual from the fine grid to the coarse grid. This process is called restriction:

$$r_{k-1} = R(r_k)$$

4. Define a new variable e on the coarse grid:

$$e_{k-1} = 0$$

5. Set up a new system, according to the error equation:

$$N_{k-1} e_{k-1} = r_{k-1}$$

6. Solve the system $N_{k-1} e_{k-1} = r_{k-1}$ to reduce the high frequencies (from the perspective of the finer grid, these are lower frequencies):

$$e_{k-1} \leftarrow S(e_{k-1})$$

7. Transfer the error e_{k-1} back to the fine grid. This process is called interpolation. The interpolation changes the high frequencies to lower ones:

$$e_k = I(e_{k-1}).$$

8. Correct the solution v_k by the error, which was obtained on the coarse grid:

$$v_k \leftarrow v_k + e_k$$

1.2 Hierarchical Hybrid Grids

The HHG framework [1, 3] is designed to close the gap between Finite Element's (FE) flexibility and geometric MG's performance 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, resulting in semi-structured meshes (see Figure 1 (b)). 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 that are constant for each structured patch. Thus, HHG is designed to have a low memory consumption as well as hardware efficient execution and a high degree of parallelism. This approach allows to solve elliptic partial differential equations with a very high resolution. HHG supports different point- and line-wise relaxation schemes for the smoothing procedure.

2 Grid partitioning and the coarsest grids

From the theoretical point of view, Multigrid has a linear complexity $O(N)$ with respect to the number of unknowns for sparse systems. This is also reflected in practice for serial runs. In parallel settings, we are also able to achieve this complexity, at least to a certain degree without special care. However the reason is an increasing communication requirement at the coarsest levels. Estimates for the decreasing volume to surface ratio for multigrid hierarchy are given in [4]. To give an example, let us assume we want to utilize 300 000 cores, having one process per core. With static grid partitioning the coarsest grid would consist of 300 000 elements. At the latest at this stage there are two possibilities to treat this grid: Proceed with the construction of new coarser grid levels by collocating elements on fewer number of cores (agglomeration) or stop at this stage and apply any iterative or direct solver.

In our multigrid algorithm, we decided to evaluate the second approach. This strategy is also known as flat multigrid, truncated cycle, or U-cycle. For a regular tetrahedral grid is a bit drastic to end up with one element per process, like in our example. This would mean, there are up to four local unknowns per process and at least 44 ghost points. HHG refines each input element two times to generate the coarsest multigrid level. Thus the minimal size per process are 64 elements (up to 35 local unknowns), which provide a reasonable volume to surface ratio. For grid partition two observeration have to be considered, when dealing with a very high degree of parallelism.

While constructing the grids in a setup phase, often global acting algorithms have to be used e.g. to find communication neighbors or to perform global numbering. Let us assume an evaluation of a relation costs 10 processor cycles and let n_P is the number of processors. Algorithms of complexity of $O(n_P)$ and $O(n_P log_2(n_P))$ would need $3 \cdot 10^6$ and $5.5 \cdot 10^7$ cycles for $n_P = 3 \cdot 10^5$, respectively. A modern processor can perfom these operations in far below one second. However, while for an $O(n_P^2)$ class it often is still possible with $n_P = 10 \cdot 10^4$ to treat our example in 10^9 cycles (around one second), $n_P = 3 \cdot 10^5$ processors would require 10^{11} cycles. This is in the range of a minute on current hardware. It can be acceptable, but our assumption are quite optimistic. If there are more one sub-grids per processor, an evaluation of a relation is more expensive, or thinking on coming generation of supercomputers a complexity of $O(n_P^2)$ has to be avoided by choosing an other algorithm or parallelizing, if possible.

Table 1: Number of arithmetic operations and measured cycles/FLOP ratio per fine-grid point.

Component	Stencil	Add	Mult	Total	Cycles/FLOP
Residual	1	1	–	30	2.1
Restriction	–	$\frac{14}{8}$	$\frac{1}{8}$	$\frac{15}{8}$	6.0
Smoothing	1	–	–	29	3.7
ProlongationAdd	–	$2 \cdot \frac{7}{8}$	$1 \cdot \frac{7}{8}$	$3 \cdot \frac{7}{8}$	11.4

The second observation is similar to the first one, but considering main memory. Since we read in an unstructured input mesh to allow geometric flexibility, basic logical information of the mesh can easily grow up to the order of 100 MB or more. This is already the case for very few million elements, which the coarsest grid might have using a flat multigrid algorithm. Thus one should think about, if it is possible to held the coarse grid structure in main memory of each processor, just in the setup phase, or if it is avoidable at all. In HHG we had to reduce the complexity for the mesh construction to $O(n_P log_2(n_P))$ and do not store the whole structure of the coarsest grid in the solving phase at each process.

3 BlueGene/P Single-Core Performance

All our calculations of this paper were performed on the supercomputer Jugene in Jülich. Each socket of Jugene is equipped with IBM's Blue Gene/P (BGP) quad/core processor. The PowerPC 450 cores run at a quite low clock frequency of 850 MHz to achieve a very low power consumption. With two multiply-add instruction per cycle, one core has an theoretical peak performance of 3.4 GFlop/s in double precision. The L1 data cache has a size of 32 KB per core, while the L2 cache is shared between four cores with a total size of 8 MB. A compute node, wich is equivalent to a socket, is capable to deliver 13.6 GB/s bandwidth. Thus, the double precision flop to byte ratio is 1.0.

Having this said, the most expensive part of a simple multigrid code are matrix-vector operations. From a performance point of view, it is even better in our case, since we are able to use 15-point stencils inside a stuctured region. In addition to that we need some simple vector operation. Table 1 shows the number of arithmetic operations per fine grid points. In our experiments, we use V-cycles with three Gauss-Seidel pre- and post-smoothing steps. Thus we perform six smoothing steps per cycle in total. For five additional coarser level, we have some additional overhead, roughly:

$$W_{total} = W_l + \frac{1}{8}W_{l-1} + \frac{1}{64}W_{l-2} + \frac{1}{512}W_{l-3} + .. \approx 1.142 \tag{2}$$

The coarse grid solving step takes below 1% of the total time and thus can be neglected in the serial case.

The finest mesh of our first two measurements consists of $4.12 \cdot 10^6$ unknowns per core. This is the same ratio, which we use for the weak scaling in the next section. The time on one core of one V-cycle is 4.25 seconds. On a full node of four core, one V-cycle takes 4.62 seconds. Since compared to the first case, only $\frac{1}{4}$ of main memory bandwidth is available, main memory bandwidth seems not be the limiting factor. Table 1 shows the required cycle per operation for the single multigrid components. The ratios already include time spend on memory copy operations and local MPI communication at the interfaces between the structured regions. Residual and smoothing consists of similar type of arithmetic operations. However their performance differ. The residual calculation need considurable less cycle per flop, because source and destination arrays are not the same. Thus better optimization is can be achieved by the compiler, even dependencies in a line of a structured region are broken by row-wise red-black Gauss-Seidel coloring. Restriction and prolongation have much worse flop to byte ratio. In addition to that, the grid is traversed two times for the prolongation, and intermediate results are stored in a temporary array. On the other hand,

both inter-grid transfer components consume substantially less arithmetic operations per fine-grid point.

Alltogether HHG achieves 6.12% peak performance for one processor in terms of double precision performance.

4 Scaling on Jugene

In this section includes weak and strong scalability results of our multigrid algorithm. Hereby we try to push HHG to limits of current CPU parallelism. Following results are measured by using VN (virtual node) mode. This means each node executes 4 tasks, sharing 2 GB of main memory.

4.1 Strong Scaling

For e.g. solving hyperbolic equations it might be interesting, not to utilize the full main memory in order to have smaller runtimes per time step. Fixing the problem size, but increasing the number of processors is known as strong scaling. For computational steering it can be useful to reduce the runtime of one time step to the order of 0.05 seconds to achieve real-time behavior. In an implicit time steping, one time step per frame would be feasable. In this section, we want to evaluate the potential of a multigrid algorithm to reach this constraint.

In an experimental setting, we solved $2.14 \cdot 10^9$ unknowns using 512 to 49 152 cores. For the strong scaling experiment, we use following multigrid components:

- V(3,3) row-wise red-black GS smoothing
- 60 steps of a CG solver to approximate the coarsest grid
- Five multigrid levels
- Direct coarse grid approximation
- Linear interpolation

Figure 2 shows, HHG achieves a good strong scaling over a wide range of cores.

Choosing 97 times more number of cores, we reduce the initial time of 7.95 seconds per V-cycle to 0.16 seconds. A perfect strong scaling would result in 0.08 seconds per V-cycle. Thus we can clearly observe, that the communication overhead impacts the performance. For the largest run, we have quite small memory arrays of around 383 KB for each variable (right hand side, unknowns, residual) including ghost points. From this data volume, the ghost points require 69 KB. Thus the volume to surface ratio is quite small. Further latency of the messages have a larger impact, since more messages per time have to be send. Significant fraction of the time (about 38%) is spend for the approximation of the coarsest grid.

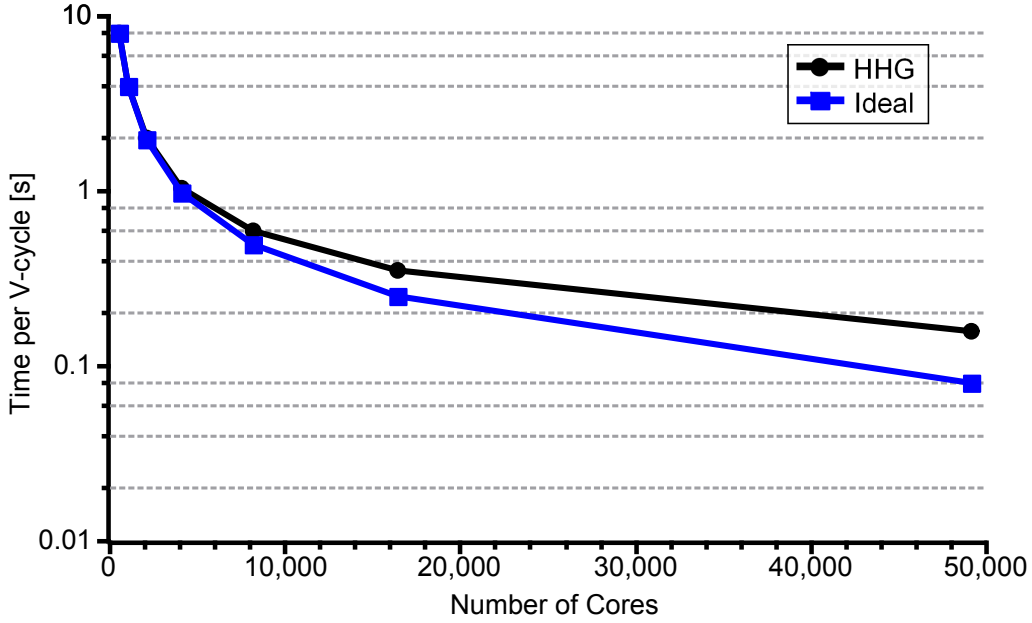
With similar setup, but using V(1,1) cycles and 40 CG steps on the coarsest grid, one cycle takes 0.07 seconds. For an implicit real-time application, additional time is required for a visualization pipeline (post-processing) and the time steping itself. However, we think it is possible to tune the solving further e.g. by optimizing further the coarsest grid solver and multigrid components, having less unknowns per core, or utilizing stronger processors. Thus highly parallel real-time simulations with a multigrid algorithm seems to be challenging but feasible on current hardware.

4.2 Weak Scaling

This section discusses some effects of solving large linear systems by MG with up to 294 912 compute cores. The setup of this scaling experiments are similar to the strong scaling of the previous section except of:

- Number of CG steps depend on the size of the coarsest grid

Figure 2: Strong Scaling behavior of HHG on PowerPC 450 cores of a Blue Gene/P at Jülich. This test case was performed starting from 512 cores, solving $2.14 \cdot 10^9$ unknowns.



- Constant numbers of MG levels:
  Six for up to 262 144 cores
  Seven for 294 912 cores

- 12 structured regions for six MG levels
  1 structured region for seven MG levels

The parallel efficiency is reflected in Table 2 by the Time (s) per V-cycle. In the limit HHG could solve up to 10^{12} degrees of freedom (DoF). Fixing the number of levels, the work spend on the coarstest grid increases from 15 to 180 CG steps. However for the largest runs, the time on the coarstest mesh is around 12.5% of the total time for one V-cycle. In general, we observe that the require number of CG steps is proportional to the diameter of the domain. For a pure CG without MG "on top" this is shown e.g. in cite:Schewchuck. The full machine run uses one level more. In our semi-structured approach, a structured region cannot be shared by multiple processors. Thus, we are not able to utilize full main memory. The additional level reduces the number of CG iterations. Further the performance increases by 14% due to larger inner loops for the finest grids.

MG achieves its efficiency by the reduced work in the coarsest grids. Theoretically the work is reduce by a factor of eight in three dimensions with standard coarsening. However, it does not fully hold in a parallel run for different reasons, especially for the coarser grids:

- Latency and number of messages ($> 26 * 4$ messages per processor, smoothing step, and level, even with message-combining)

- Bad ratio between communication and computation

- Very small memory arrays (on the coarest level: around 10 unknowns per processor)

- More memory copy operations of boundary points, also due to a bad ratio between communication and computation

Table 3 tries to capture the impact of those effects to the performance. This should be similar for other (semi-) structured linear FE programs. The level corresponds to the MG level. Level

Table 2: Weak Scaling behavior of HHG on PowerPC 450 cores of a Blue Gene/P at Jülich.

Cores	**Struct. Regions**	**DoF** ($\cdot 10^6$)	**CG**	**Time (s)**
128	1 536	535	15	5.64
256	3 072	1 071	20	5.66
512	6 144	2 142	25	5.69
1024	12 288	4 287	30	5.71
2048	24 576	8 577	45	5.75
4096	49 152	17 159	60	5.92
8192	98 304	34 326	70	5.86
16384	196 608	68 669	90	5.91
32768	393 216	137 355	105	6.17
65536	786 432	274 744	115	6.41
131072	1 572 864	549 555	145	6.42
262144	3 145 728	1 099 176	180	6.52
294912	294 912	824 365	110	3.80

Table 3: Performance of communication and computation on different coarse grid levels. Efficiencies η are normalized to the finest grid. *Total* and *computation* are measured in updated points per time with and without communication, respectively. *Communication* states the number of transfered points per time. λ is the ratio between computation and communication time.

Level	Line size	η_{total}	$\eta_{computation}$	$\eta_{communication}$	$\lambda_{comp/comm}$
6	129	1	1	1	7.59
5	65	0.72	0.91	0.61	2.56
4	33	0.42	0.57	0.53	1.78
3	17	0.15	0.21	0.31	1.41
2	9	0.03	0.03	0.06	1.13

1 is created by refining each input element two times. This results in the coarsest mesh in our MG hierarchy, and consequently CG has to solve the arising linear system. The line size n are the number of grid points in one irection. Ghost and boundary points are included. In a straight forward (semi-) structured hexahedral FE domain partitioning we have n^3 grid points in total and n^2 for one face on one process. However, with our tetrahedral elements we end up with $n(n+1)(n+2)/6$ and $n(n+1)/2$ grid points respectively. η_{total} shows how efficiently the unknowns are updated on the different levels. While this is not so much severe for the finest grids, we have a relative factor of three between the coarsest two. However, one have to keep in mind that also the unknowns per grid are drastically decreasing. For example, the total time spend on level 2 is still a factor of 110 less compared to the finest one. $\lambda_{comp/comm}$ gives the ratio between computation and communication time. On the finest grid, the calculations are clearly dominant w.r.t. time. Due to increasing surface to volume ratio, we theoretically should see a relative factor of 2 between two consequent grid levels. In measurements, we observe a factor above two for the finer grids and below two between the coarser grids. The splitting into $\lambda_{computation}$ and $\lambda_{communication}$ shows this in more detail. For large structured regions, the penalty is larger for the communication, while small memory arrays have a stronger impact to the computations.

5 Conclusion

In our presentation, we addressed scalability problems and communication overhead created by the coarsest grids in a multigrid hierarchy. Surprisingly, a careful implementation can result in excellent scalability results, i.e. the coarse grids do not seriously effect overall parallel performance. To this end we explored and analyzed the weak scaling of numerical experiments with up to 10^{12} unknowns.

A strong scaling was investigated the possibilities to solve parabolic equations in real-time by an implicit scheme. To achieve real time behavior, strong scaling is necessary, i.e. for a fixed problem size more and more processors must be used. There the parallel efficiency on current architectures reaches its limits.

References

[1] B. Bergen, T. Gradl, F. Hülsemann and U. Rüde. *A massively parallel multigrid method for finite elements.* Comput. Sci. Eng., 8 (2006), pp. 56–62.

[2] W. Briggs V. E. Henson, S. McCormick. *A Multigrid Tutorial*, Society for Industrial and Applied Mathematics, (2000).

[3] T. Gradl and U. Rüde. *High Performance Multigrid on Current Large Scale Parallel Computers* 9th Workshop on Parallel Systems and Algorithm (PASA), Bonn, (2008).

[4] F. Hülsemann, M. Kowarschik, M. Mohr and U. Rüde. *Parallel Geometric Multigrid.* Numerical Solution of Partial Differential Equations on Parallel Computers, Springer, 2005, pp. 165–208.

[5] W. Hackbusch. *Multi–grid methods and applications.* Springer, Berlin, 1985.

[6] U. Trottenberg, C.W. Oosterlee and A. Schüller. *Multigrid* Academic Press, New York, 2001.

[7] P. Wesseling. *An introduction to multigrid methods.* John Wiley, Chichester, UK, 1992.