A task-based directed acyclic graph implementation of additive AMG

Amani AlOnazi
Extreme Computing Research Center
King Abdullah University of
Science and Technology
Saudi Arabia
Email: amani.alonazi@kaust.edu.sa

George S. Markomanolis
KAUST Supercomputing Laboratory
King Abdullah University of
Science and Technology
Saudi Arabia
Email: georgios.markomanolis@kaust.edu.sa

David Keyes
Extreme Computing Research Center
King Abdullah University of
Science and Technology
Saudi Arabia
Email: david.keyes@kaust.edu.sa

***Abstract*—Algebraic multigrid is the solver of choice in many and growing applications in today's petascale environments and scales well in a weak, distributed-memory sense. Exascale architectures exacerbate the challenges of increased strong SIMT concurrency, reduced memory capacity and bandwidth per core, and vulnerability to global synchronization. The recent Vassilevski-Yang (2014) "mult-additive" AMG nearly preserves the convergence rate of the multiplicative form of AMG, while reducing communication and synchronization frequency and controlling memory growth. However, it remains bulk-synchronous. We extend the algorithm further towards exascale environments with a task-based directed acyclic graph implementation. The algorithm can then be represented as a task-based DAG where vertices represent tasks, and edges are dependencies among them. We implement a tiling approach for decomposing the grid hierarchy into parallel units within task containers. The distribution of tiles is left to the compiler and OmpSs runtime system to expose task-level parallelism.**

I. Introduction

Since the 1980's, many researchers have proposed new multigrid algorithms, such as additive variants [1], to adapt them to evolving massively parallel systems. The common idea among these algorithms is trading communication for computation at the price of extra computation and slower convergence. Due to the robustness of classical multigrid and Moore's law, some of the proposed solutions were kept on the shelf for more than two decades. However, as single-core speeds are no longer doubling, hardware trends are that the number of cores increases, memory capacity per core shrinks, and on-node memory hierarchy becomes more complex. This trend imposes challenges for classical multigrid algorithms and implementations.

In general, there are three main components in the algebraic multigrid solve phase: restriction, smoothing, and prolongation kernels. As shown in Figure 1, the main difference between multiplicative and additive multigrid V-cycle is that in the multiplicative method, smoothing and restriction routines at the next coarser level are executed sequentially on one level after the other, whereas in the additive one smoothing on the different levels can be executed in parallel. Restriction and prolongation are sequential in both methods. Mathematically, the work counts of both methods are equivalent. The

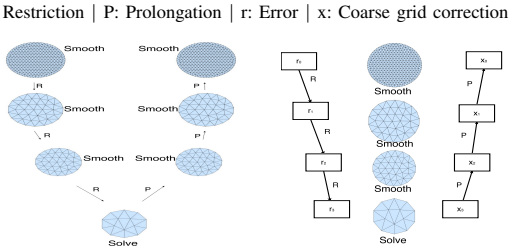

Fig. 1. Outline of AMG V-cycle both multiplicative (left) and additive (right) variants.

aforementioned work of Vassilevski-Yang [1] introduces an additive algebraic multigrid that exhibits convergence behavior similar to the classical algebraic multigrid. This takes important steps in reducing communication, but remains bulk-synchronous. We extend it further towards exascale environments with a two-level parallel implementation.

In this work, we present a parallel approach based on data-driven execution to speed up the on-node performance of AMG. This approach takes advantage of the extra factor of parallelism between grid hierarchy in the additive AMG and data flow programming that express the application as a directed acyclic graph (DAG), where vertices represent computational tasks and edges are the dependencies among them. This approach is a fine-grained task-based implementation to deal with intra-node concurrency. Therefore, a classical parallel approach of multigrid domain decomposition using MPI exploits inter-node communication while a task-based DAG exploits node parallelism. At this stage of the project, we focus on single node performance, which is the more challenging of the two issues, and multithreaded-MPI is noted as ongoing work.

The main idea consists of breaking the main three kernels of additive multigrid: restriction, smoothing, and prolongation, into tasks. We decompose the tasks by decomposing the grid hierarchy into parallel units within task containers using a loop tiling approach. Then, the whole solve phase of additive AMG can be represented as a DAG. The execution of the AMG

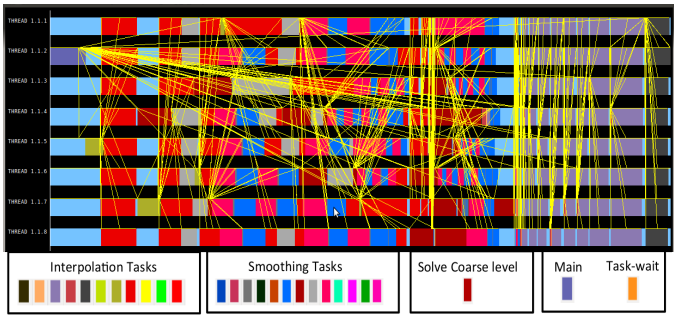

Fig. 2. A trace of 8 cores executing a DAG of single iteration of additive AMG V-cycle used as preconditioner for conjugate gradient to solve the test problem on 128×128×128 grid, generated by Extrae and Paraver tools of BSC.

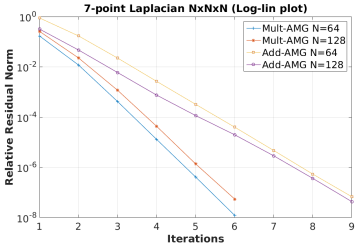

Fig. 3. Convergence results of BoomerAMG using multiplicative cycle and our implementation of additive cycle with task-based parallelism

solve phase is performed by asynchronously and dynamically scheduling the tasks using OmpSs runtime system of the Barcelona Supercomputing Center (BSC) over available cores within the node. While respecting data dependencies, executing the set of tasks in AMG solve phase results in an out-of-order execution, and a potential approach for hiding the latency of MPI-communication calls. We implement our approach within the *hypre* library of Lawrence Livermore National Laboratory. The test problem is a 3D Laplace problem with a 7-point finite difference stencil on a Cartesian grid and uniform Dirichlet boundary condition. Figure 2 illustrates a DAG of single iteration of additive AMG V-cycle executed by 8 cores.

II. Results and Discussion

The parallel V-cycle is implemented using OmpSs in the BoomerAMG code in *hypre* library. AMG is used to solve the formerly mentioned test problem as a preconditioner for conjugate gradient. The setup phase was done using *hypre* library with Falgout coarsening and a Jacobi relaxation method. The test runs were performed on Shaheen II, a Cray XC40 supercomputer at KAUST.

Figure 3 shows the convergence of BoomerAMG using multiplicative cycle and our implementation of additive cycle with task-based parallelism while solving the 3D Laplace test problem. Figure 4 shows more than 4× speedup on-node performance compared to single core execution. As we increase the number of cores, cache misses increase and for

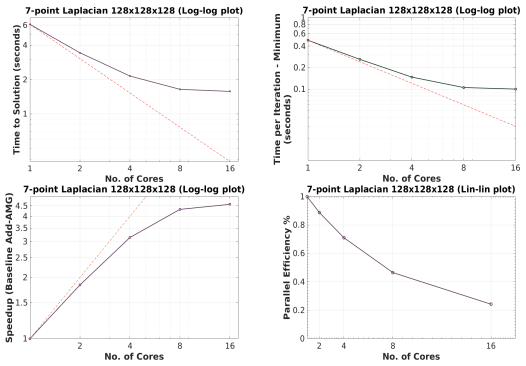

Fig. 4. Strong scalability data of additive AMG with task-based parallelism

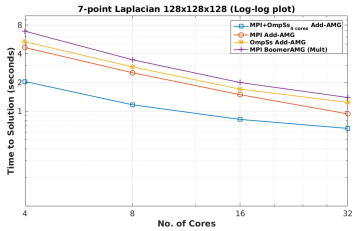

Fig. 5. Strong scaling of BoomerAMG (MPI), Add-AMG (OmpSs), and Add-AMG (MPI+OmpSs)

this reason the performance and parallel efficiency begins to degrade after 4 cores. We compared the strong scaling of BoomerAMG using MPI within a node and additive AMG using OmpSs and MPI+OmpSs. As shown in Figure 5, the hybrid MPI+OmpSs additive AMG outperforms flat MPI and flat OmpSs alternatives.

III. Conclusion and Future Work

In this work, we presented a parallel approach based on data-driven execution to speedup the on-node performance of AMG. We extended the Vassilevski-Yang mult-additive AMG algorithm with a new implementation as an asynchronous DAG, where vertices represent tasks, and edges are the data dependencies between them. We implemented a tiling approach for decomposing the grid hierarchy into parallel units within task containers. The distribution of tiles is left to the compiler and runtime to expose task-level parallelism. We are working on extending this work to include inter-node communication, which means a hybrid distributed-shared memory parallelism. MPI exploits inter-node parallelism while a task-based directed acyclic graph exploits node parallelism. We plan to further investigate the performance of multithreaded-MPI (MPI+OmpSs) additive AMG implementation. So far, only Jacobi relaxation method is implemented with task-based programming model. We also plan to investigate the performance of taskifying other relaxation methods.

References

[1] P. S. Vassilevski and U. M. Yang, "Reducing communication in algebraic multigrid using additive variants," *Numerical Lin. Alg. with Applic.*, vol. 21, no. 2, pp. 275–296, 2014. [Online]. Available: http://dx.doi.org/10.1002/nla.1928