Communication overlapping Pipelined Conjugate Gradients for Distributed Memory Systems and Heterogeneous Architectures

Manasi Tiwari and Sathish Vadhiyar

Department of Computational and Data Sciences, Indian Institute of Science, Bangalore {manasitiwari,vss}@iisc.ac.in

Abstract. Preconditioned Conjugate Gradient (PCG) method has been one of the widely used methods for solving linear systems of equations for sparse problems. Pipelined PCG (PIPECG) attempts to eliminate the dependencies in the computations in the PCG algorithm and overlap non-dependent computations by reorganizing the traditional PCG code and using non-blocking allreduces. We have developed a novel pipelined PCG algorithm called PIPECG-OATI (One Allreduce per Two Iterations) which reduces the number of non-blocking allreduces to one per two iterations and provides large overlap of global communication and computations at higher number of cores in distributed memory CPU systems. PIPECG-OATI gives up to 3x speedup over PCG and 1.73x speedup over PIPECG at large number of cores.
For GPU accelerated heterogeneous architectures, we have developed three methods for efficient execution of the PIPECG algorithm. These methods achieve task and data parallelism. Our methods give considerable performance improvements over PCG CPU and GPU implementations of Paralution and PETSc libraries.

Keywords: Preconditioned Conjugate Gradient · Overlapping communication and computations · Distributed Memory Systems · Heterogeneous Architectures

1 PIPECG-OATI for Distributed Memory Systems

1.1 Problem Statement

The main computational kernels in Preconditioned Conjugate Gradient (PCG) [3] are Sparse Matrix Vector Product (SPMV), Preconditioner Application (PC), Vector-Multiply-Adds (VMAs) and Dot Products. For distributed memory systems, the bottleneck in PCG are the three dot products per iteration. They result in synchronization and waiting of the processors which cannot be overlapped with any independent computations. As the number of cores increase, the time taken for allreduce increases and the cores wait for a longer time. The pipelined PCG method (PIPECG)[2] was introduced for distributed memory systems. It reduces the number of allreduces in PCG to one per iteration and overlaps it with one PC and one SPMV. While this is a reasonable strategy for smaller number of cores, executions of the PIPECG code at larger number of cores show that the time taken by the allreduce can not be fully overlapped by the PC and SPMV.

As we are moving to the exascale era, in order to obtain good performance at larger number of cores, we must reduce the number of allreduces in the PCG method even more and remove dependencies between computations so that non blocking allreduce can be used to overlap global communication with more computations.

1.2 Methodology

In order to solve the aforementioned problem, we propose a novel algorithm, PIPECG-OATI (PIPECG-One Allreduce per Two Iterations)[5], which combines two iterations of PIPECG, reduces the number of non-blocking allreduces to one per two iterations and then overlaps them with two PCs and two SPMVs. This is done at the cost of introducing extra VMA operations.

The primary challenge in combining two iterations of PIPECG and pipelining it is that it has dependencies that require an extra PC and an extra SPMV for each combined-iteration. So, a total of three PCs and three SPMVs would be required in a combined-iteration as opposed to two PCs and two SPMVs in two uncombined iterations. Since the PC and SPMV are the most computationally intensive kernels in each iteration, an extra PC and SPMV would degrade the performance of PIPECG-OATI. To deal with this challenge, we introduced new non-recurrence computations in each iteration of PIPECG-OATI which brings down the number of PCs and SPMVs to two per combined-iteration.

For achieving PIPECG-OATI from PIPECG, we follow the below steps:

1. Collect the PCs and SPMVs of two iterations at one point in the combined-iteration by introducing recurrence relations.
2. Collect the dot products of two iterations at one point in the combined-iteration by expressing the vectors as recurrence relations.
3. As the new dot products will need results of PC and SPMV beforehand, introduce recurrence relations for these PC and SPMV.
4. To deal with extra PC and SPMV, introduce new non-recurrence computations.

An elaborate derivation can be found in [5].

1.3 Experiments and Results

We have implemented our PIPECG-OATI method along with optimizations like merged vector operations in the PETSc library[1][1]. We ran tests on our Institute's supercomputer cluster called SahasraT, a Cray-XC40 machine which has 1376 compute nodes. Each node has two CPU sockets with 12 cores each, 128GB RAM and connected using Cray Aries interconnect. We use Jacobi preconditioner in all tests.

Figure 1 shows the strong scaling of different methods on a 125-pt 3D Poisson problem with 2M unknowns on up to 110 nodes (2640 processes). Our

[1] Available as KSPPIPECG2. URL: https://www.mcs.anl.gov/petsc/petsc-master/docs/manualpages/KSP/KSPPIPECG2.html

method, PIPECG-OATI, is compared with PCG, PIPECG and other pipelined variants like PIPECG3 method available in PETSc. We observe from Figure 1 that PCG reaches 21x speedup and PIPECG reaches 26x speedup after which speedup degrades due to increased allreduce costs which are not overlapped with either any or enough computations. We see that PIPECG3 reaches 34x speedup. PIPECG-OATI reaches 36x speedup. It performs better than PCG and PIPECG because at higher number of cores, the overlap provided by PIPECG-OATI becomes more than the overlap provided by PIPECG. It performs better than PIPECG3 because of lesser number of FLOPS.

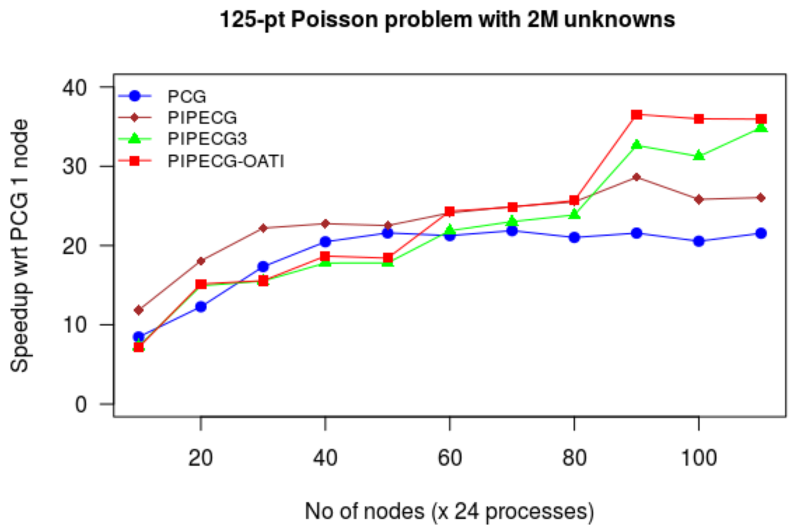

Fig. 1. Strong scaling of different methods on a 125-pt Poisson problem with 2M unknowns.

2 PIPECG executions for GPU accelerated architectures

We have developed three methods for efficient execution of PIPECG method for GPU accelerated systems so that we can make use of all the resources available in the GPU node. The first two methods, Hybrid-PIPECG-1 and Hybrid-PIPECG-2, achieve task parallelism by executing dot products on the CPU while executing the PC and SPMV kernels on the GPU. The third method, Hybrid-PIPECG-3 achieves data parallelism by decomposing the workload between CPU and GPU based on a performance model. The performance model takes into account the relative performance of CPU and GPU using some initial executions and performs 2D data decomposition. Our methods give up to 8x speedup over PCG CPU implementation of Paralution[4] and PETSc libraries for different matrices as shown in Figure 2. Our methods give up to 5x speedup speedup over PCG GPU implementation of Paralution and PETSc libraries. Hybrid-PIPECG-3 method also provides an efficient solution for solving problems that cannot be fit into the GPU memory and gives up to 2.5x speedup for such problems. Further details can be found in [6].

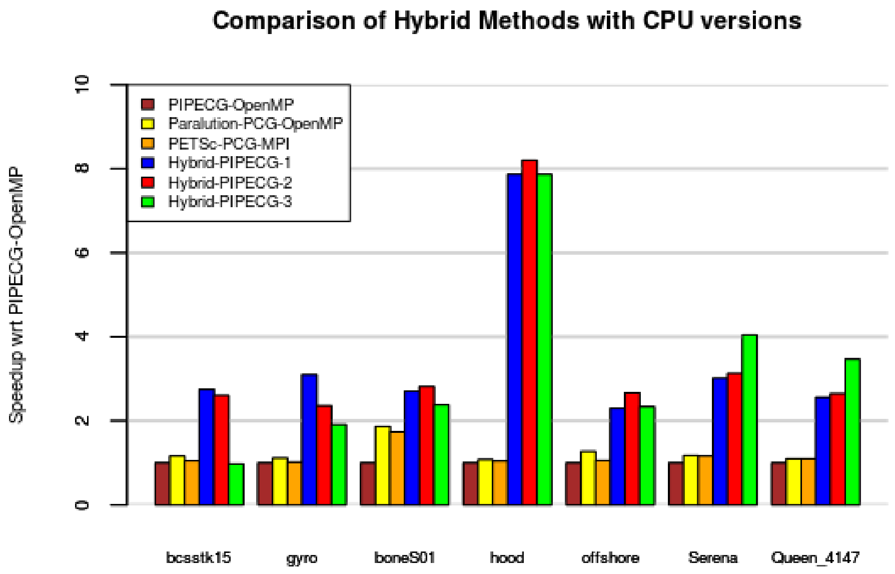

Fig. 2. Comparison of Hybrid methods with various CPU versions.

3 Conclusion and Future Work

In this work, we have presented PIPECG-OATI, which reduces the number of allreduces to one per two iterations and overlaps the allreduce with two PCs and two SPMVs. We have also presented Hybrid PIPECG methods for GPU accelerated systems. We are working on developing further pipelined methods which will provide greater overlap than PIPECG-OATI. We are also developing multi-node multi-GPU version for efficient executions of pipelined CG methods on GPU accelerated architectures to explore very large problem sizes.

References

1. Balay, S., Gropp, W.D., McInnes, L.C., Smith, B.F.: Efficient management of parallelism in object oriented numerical software libraries. In: Modern Software Tools in Scientific Computing. pp. 163–202. Birkhäuser Press (1997)
2. Ghysels, P., Vanroose, W.: Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. Parallel Comput. (2014)
3. Hestenes, M.R., Stiefel, E.: Methods of conjugate gradients for solving linear systems. Journal of research of the National Bureau of Standards **49**, 409–436 (1952)
4. Labs, P.: Paralution v1.1.0 (2020), http://www.paralution.com/
5. Tiwari, M., Vadhiyar, S.: Pipelined preconditioned conjugate gradient methods for distributed memory systems. In: 27th IEEE International Conference on High Performance Computing, Data, and Analytics, 2020
6. Tiwari, M., Vadhiyar, S.: Efficient executions of pipelined conjugate gradient method on heterogeneous architectures. arXiv.org (2021)