Non-blocking Preconditioned Conjugate Gradient Methods for Extreme-scale Computing

Paul R. Eller, William Gropp (Advisor)
University of Illinois at Urbana-Champaign
Urbana, IL 61801

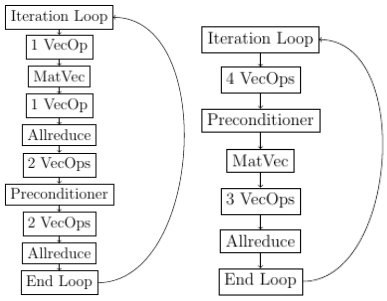

Figure 1: Diagrams for the PCG (left) and L56PCG (right) solver iteration loops.

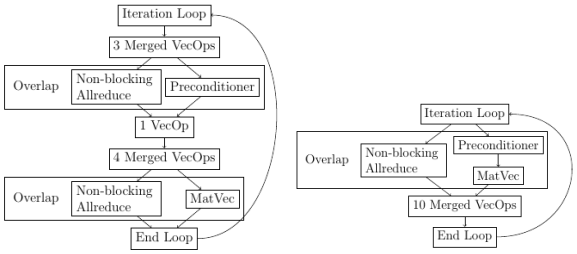

Figure 2: Diagrams for the NBPCG (left) and PIPECG (right) solver iteration loops.

1. INTRODUCTION

To achieve the best performance on extreme-scale systems we need to develop more scalable methods. For the preconditioned conjugate gradient method (PCG), dot products limit scalability because they are a synchronization point. Non-blocking methods have the potential to hide most of the cost of the allreduce and avoid the synchronization cost due to performance variation across cores.

PCG is an iterative algorithm for solving large sparse systems of linear equations that uses preconditioners to accelerate convergence. The key operations within PCG can be rearranged to reduce communication latency by using a single allreduce and to use non-blocking allreduces to overlap communication and computation at the cost of additional overhead.

2. SCALABLE CONJUGATE GRADIENT METHODS

We analyze PCG and three scalable variations of PCG shown in figures 1 and 2. Lapack Working Note 56 PCG (L56PCG) [1] uses a single allreduce, non-blocking PCG (NBPCG) uses two non-blocking allreduces, and pipelined PCG (PIPECG) [2] uses a single non-blocking allreduce. This approach produces methods equivalent to PCG in exact arithmetic. We implement these solvers as custom solvers in PETSc. To obtain the best performance with these methods, we need to effectively overlap communication and computation and minimize the cost of added vector operations.

2.1 Non-blocking Allreduce

There are two approaches to effectively using a non-blocking allreduce. The first approach uses calls to MPI_Test() within the computation to make progress on the non-blocking allreduce. This gives MPI control of the process to complete the next step of communication, but requires code modifications.

A second approach uses progress threads, which dedicates one or more threads per node to communication. This allows a thread to make progress on the non-blocking allreduce at the potential cost of one core per node for computation.

These approaches are not ideal, but in the future we will have machines capable of performing the non-blocking allreduce fully in hardware. This will allow message passing libaries to perform collectives in network hardware, allowing the processor to focus on computation.

2.2 Merged Vector Operations

Both NBPCG and PIPECG produce additional vector operations that can be grouped together so that we can reduce the number of vector reads. Instead of computing vector operations like AXPY's sequentially, we can perform these operations per array element so that we can read each array element from memory once, but use it multiple times. Figure 3 shows this avoids extra reads from memory, but

This work was performed in collaboration with Mark Hoemmen (Sandia National Laboratories).

This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy award DE-FG02-13ER26138/DE-SC0010049 and in part by the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number OCI 07-25070) and the state of Illinois.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. This paper is cross-referenced at Sandia as SAND2015-6373A.

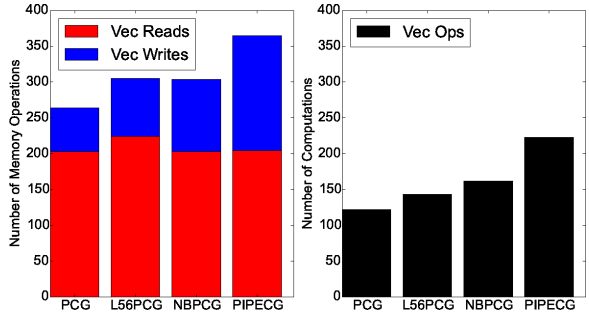

Figure 3: Number of vector reads, writes, and computations for 20 iterations of each method.

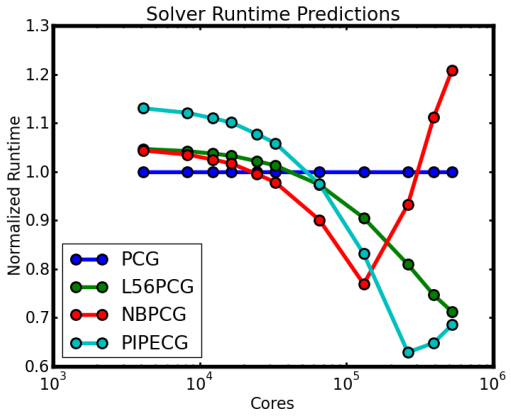

Figure 4: Performance model predicted performance of each solver normalized with respect to PCG.

still requires extra writes to memory and computations, although computations are cheap compared to memory accesses. PCG and L56PCG require mostly separate vector operations, but read some vectors from cache when multiple vectors fit in cache, improving performance.

3. RESULTS AND ANALYSIS

3.1 Performance Modeling

Due to the complicated nature of large-scale systems and non-optimal non-blocking allreduce implementations we can use performance modeling to better understand what non-blocking methods are capable of. We model computation using parameters produced by a modified STREAM benchmark. We use LogGOPS to model communication and Netgauge to compute communication parameters. We analyze performance using strong scaling tests, 5-point Poisson matrices, and a block-Jacobi ILU preconditioner.

Figure 4 shows that in our model non-blocking methods perform better than blocking methods as the vector operations cost decreases and allreduce cost increases. Non-blocking methods perform well while the matrix-vector multiply and/or preconditioner have enough computation to hide the cost of the allreduce. NBPCG initially outperforms PIPECG due to the lower vector operations cost, but PIPECG scales better due to overlapping the allreduce cost with both the matrix-vector multiply and preconditioner.

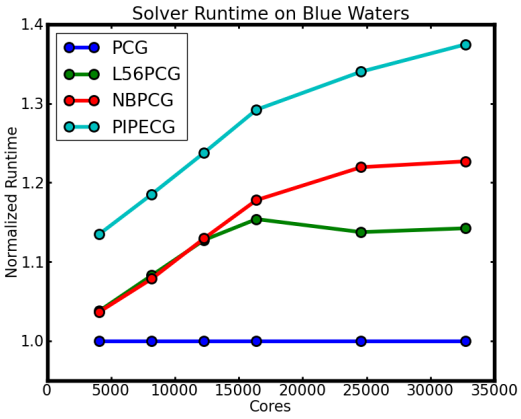

Figure 5: Performance of each solver on Blue Waters normalized with respect to PCG.

3.2 Impact of Noise

Noise throughout PCG methods limits performance by causing all processes to wait for the slowest process at synchronization points. Computational noise sources include operating system processes, error correction, etc. Communication noise sources include contention in the network, varying distances between nodes, the size and number of messages per process, etc. Modifying the performance model to take into account noise predicts 10-20% improved performance for NBPCG and PIPECG, suggesting that the ability to minimize the impact of noise may be one of the key advantages to non-blocking methods.

3.3 Solver Performance Results

We evaluate solver performance by running tests on Blue Waters, a Cray XE-6, on up to 32k cores using 5-point, 1 billion row Poisson matrices. Figure 5 shows that currently non-blocking solvers cannot efficiently enough overlap communication and computation to overcome the increased vector operations cost. Blocking methods obtain improved performance when multiple vectors fit in cache, helping them perform better as we scale. The non-blocking methods produce more consistent runs and allow processes to make progress at different rates. Using merged instead of separate vector operations can reduce the cost by over 30%.

The results from Blue Waters confirm that the performance model is able to accurately model the performance of PCG. The performance model predicts ideal non-blocking allreduce runs more quickly than we see in practice, suggesting we need to develop more detailed non-blocking allreduce models to accurately model non-blocking solvers.

4. REFERENCES

[1] E. D'Azevedo, V. Eijkhout, and C. Romine. Lapack working note 56: Reducing communication costs in the conjugate gradient algorithm on distributed memory multiprocessors. Technical report, University of Tennessee, Knoxville, TN, USA, 1993.

[2] P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Comput.*, 40(7):224–238, July 2014.