CS554: Parallel Numerical Algorithms
Project Proposal: Parallel Conjugate Gradients

Olek Yardas, Gautham Krishnan, Nicolas Nytko

October 20, 2021

1 Background

The conjugate gradient (CG) method is an iterative method for solving symmetric, sparse linear systems such as the ones arising from finite difference or finite element discretizations of PDEs. These systems, while very sparse, can contain several hundred or thousands of rows and columns and can benefit from methods that take advantage of this and do not factorize the entire matrix. Anecdotally, iterative methods can also be quicker to solve such systems since, e.g., they do not have problems such as introducing fill when solving the problem.

2 Implementation

Our proposed approach is to first implement a serial version of CG to solve the Poisson Equation in 2D. We will discretize the PDE with finite differences, giving us a structured grid and nice analytic properties; in the case of 2D grid-aligned Poisson the eigenvalues and eigenvectors are known. The most computationally expensive steps in the CG method are the Sparse Matrix Vector product and vector inner product steps. Hence, being able to perform these steps in parallel on multiple PEs should ensure performance improvement. However, distributing the matrix over different PEs incur additional communication costs in performing these steps.

1. First, run serial tests of the CG algorithm to have a baseline benchmark for various sizes to compare the parallel implementation later.
2. Next, parallel CG implementation is implemented wherein the algorithm will parallelize expensive computations, till it converges as mentioned above. We expect that our implementation will:
   (a) Employ MPI collective calls like `MPI_Reduce`, Allreduce for gathering matrix multiplication results.
   (b) Will have appropriate synchronization points after each iteration.
   (c) Consider overlapping communication and computation in the algorithm where possible.
   (d) Be scalable to any number of PEs and problem sizes.
   (e) Assumptions - All processes can perform and communicate in same amount of time

The additional benefits/effects of such scheduling on a simple Jacobi (diagonal) preconditioned CG algorithm in contrast to CG will be studied. The convergence can also be compared against another parallel iterative method such as MGCG.

We will base our implementation of parallel CG on work done in Saad et. al (1985) [2], which looks at parallel GMRES instead of vanilla CG.

3 Analysis

One of the fundamental quantities of interest for a problem like this is its scalability. We will analyze our algorithm to obtain a theoretical expression for the number of nodes for strong and weak scaling, as well as run computational experiments to compare actual scaling against the theoretical expressions. We will also test our code's correctness by either

1. direct error analysis against the known analytical solution, or
2. if no analytical expression is available for the solution, performing a convergence study in error for varying input sizes.

We will present the results of our analysis in figures and tables (where applicable).

We will also look at the eigenvalues of our original preconditioned matrices to check that our timing and convergence results agree with the conditioning. Tatebe (1993) [3] performs such analysis on a 2D Poisson problem with Diriclet boundary conditions. We will use their anaylsis as a basis for ours as well as a point of comparison.

4 Extensions

We present the following extension which we may attempt if we have enough time.

4.1 Parallel Geometric Multigrid

We can augment the above solver to use a multigrid scheme at some point: either by using an MG solver for preconditioning the CG, or converting to a full MG cycle with a CG solve for the coarse level grid. We would likely keep this simple and use a geometric multigrid scheme (coarsening by 2, 3, etc. in each dimension); this would allow us to use simple grid-transfer functions, $\mathbf{P}$ and $\mathbf{P}^T$, that could be defined ahead of time.

To parallelize the multigrid scheme, we need some parallel way of forming the Galerkin coarse product $\mathbf{A}_H = \mathbf{P}^T\mathbf{A}_h\mathbf{P}$, with the subscripts h and H denoting values on the fine and coarse grids, respectively. This could naively be done using a sparse mat-mat product, though since each node on the coarse grid is only dependent on a few neighbors on the fine grid there may be more efficient ways of implementing it. We would also need to use a sparse matvec to form the coarse-grid residual $\mathbf{r}_H = \mathbf{P}^T(\mathbf{b} - \mathbf{A}\mathbf{x})$ and interpolate the error $\mathbf{e}_h = \mathbf{P}\mathbf{e}_H$ back to the fine grid, which we would already have from implementing CG.

We would likely use a Jacobi preconditioner for the relaxation scheme, which we would also have implemented from above, with other possible candidates to test and compare against being Incomplete Cholesky and ILU preconditioning [1] . Thus, we could either do:

1. Conjugate gradient solver w/ a geometric multigrid preconditioner.
2. A geometric multigrid solver w/ a conjugate gradient coarse solve.

References

[1] Anshul Gupta, Vipin Kumar, and Ahmed Sameh. Performance and scalability of preconditioned conjugate gradient methods on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 6(5):455–469, 1995.

[2] Youcef Saad and Martin H Schultz. Parallel implementations of preconditioned conjugate gradient methods. Technical report, YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE, 1985.

[3] Osamu Tatebe. Eigenvalue analysis and parallelization of the multi-grid preconditioned conjugate gradient method (fundamental technologies in numerical computation). , 832:64–74, 1993.