STMS 11-34 & 35

Solver Interface & Performance on Cori

PEEKS team

University of Tennessee, Knoxville, Tennessee, U.S.A.
Sandia National Laboratories[1], Albuquerque, New Mexico, U.S.A.

June 28, 2018

[1] Sandia National Labs is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

One of the main objectives of ECP PEEKS project is to improve the parallel scalability of Krylov linear solvers in the Trilinos scientific computing library [5]. For the milestone STMS 11-33, we have developed the following solvers within Trilinos software framework:

1. Single-reduce and pipelined variants [3] of the Conjugate Gradient (CG) method [6] for solving a symmetric positive definite (SPD) linear system of equations, and
2. Single-reduce and pipelined [4], and s-step [7] variants of the Generalized Minimum Residual (GMRES) method [9] for solving a general nonsymmetric linear system of equations.

The single-reduce and s-step variants aim to reduce some of the required communication, while the pipelined variant aims to hide the communication that can be the performance bottleneck on the large-scale supercomputer. Currently, CG and GMRES are used by ECP application projects like EXAALT (molecular dynamics at exascale) and ExaWind (exascale predictive wind plant flow physics). They also have users in non-ECP application codes funded by the US Department of Energy and other US government entities.

This report is for the two milestones STMS 11-34 and STMS 11-35. First, for STMS 11-34, we document the current interfaces to the new PEEKS Krylov solvers (Chapter 2). We have verified that we can use this interface to integrate our new solvers into the Nalu code [1] that is used in the ECP ExaWind project. Then, for STMS 11-35, we report the current solver performance on Cori Supercomputer at NERSC (Chapters 3 and 4). We also list our current efforts (Chapter 5).

CHAPTER 2

Solver Interface

Within Trilinos software, the Krylov solvers are contained in Belos package [2]. To ease the use of our new Krylov solvers for the current Belos users, we designed our solver interfaces such that our new solvers are accessible through `Belos::SolverManager` or `Belos::SolverFactory` class. As shown in Figure 2.1 or 2.2, the user requires a minimum change to the existing codes. To verify if our prototype solvers can be integrated into an ECP application code, we used the `Belos::SolverFactory` interface to integrate our solvers into the Nalu code [1] that is used in the ECP ExaWind project. In Figure 2.3, Lines 5 through 10 show the current required changes to the code. We are improving the `Belos::SolverFactory` interface such that the current `Belos::SolverFactory` users can use the new solver without making any change to their code by simply specifying the new solver name from the command-line. The currently-supported solver names are shown in Figure 2.1. To get the feedbacks from the community, we have presented our new interface at the SIAM conference on parallel processing for scientific computing [8].

type	name
Standard method	"`PeeksGmres`" and "`PeeksCg`"
Pipelined variant	"`PeeksGmresPipeline`" and "`PeeksCgPipeline`"
Single-reduce variant	"`PeeksGmresSingleReduce`" and "`PeeksCgSingleReduce`"
s-step variant	"`PeeksGmresSstep`"

Table 2.1: Names of the new Krylov solvers.

Our solvers are also accessible without using `Belos::SolverManager` or `Belos::SolverFactory` class. Namely, all of our new Krylov solvers inherit the base class called `Krylov`, and hence, an object of a different class can be instantiated at run time as shown in Figure 2.4.

```
1
2  typedef Belos::LinearProblem<SC, mv_type, op_type> belos_problem_type;
3  typedef Belos::SolverManager<SC, mv_type, op_type> solver_manager_type;
4
5  // ////////////////////////////////////////////////
6  // Create KrylovManager
7  solver_manager_type *solverManager;
8  solverManager = new KrylovManager<> (args.solverName);
9
10 // Set parameters
11 solverManager->setParameters (params);
12
13 // Setup Belos::LinearProblem.
14 RCP< belos_problem_type > lp =
15     rcp (new belos_problem_type (rcpFromRef (*A), rcpFromRef (X), rcpFromRef (B)));
16 lp->setProblem ();
17 solverManager->setProblem (lp);
18
19 // Solve the system.
20 Belos::ReturnType belosResult;
21 belosResult = solverManager->solve ();
```

Figure 2.1: Interface to the new Krylov solvers through `Belos::SolverManager` class.

```
1
2  typedef Belos::LinearProblem<SC, mv_type, op_type> belos_problem_type;
3  typedef Belos::CustomSolverFactory<SC, mv_type, op_type> custom_factory_type;
4
5  // ////////////////////////////////////////////////
6  // Create KrylovFactor
7  RCP<custom_factory_type> solverFactory
8      = rcp (static_cast<custom_factory_type*> (new KrylovFactory<> ()));
9
10 // Add KrylovFactory
11 Belos::SolverFactory<SC, mv_type, op_type> factory;
12 factory.addFactory (solverFactory);
13
14 // Create KrylovManager
15 RCP<solver_manager_type> solverManager;
16 solverManager = factory.create (args.solverName, params);
17
18 // Setup Belos::LinearProblem.
19 RCP< belos_problem_type > lp
20     = rcp (new belos_problem_type (rcpFromRef (*A), rcpFromRef (X), rcpFromRef (B)));
21 solverManager->setProblem (lp);
22
23 // Solve the system.
24 belosResult = solverManager->solve ();
```

Figure 2.2: Interface to the new Krylov solvers through `Belos::CustomSolverFactory` class.

```
1
2   // create the solver, e.g., gmres, cg, tfqmr, bicgstab
3   LinSys::SolverFactory sFactory;
4
5   // Create a new factory
6   Teuchos::RCP<custom_factory_type> solverFactory
7     = Teuchos::rcp (static_cast<custom_factory_type*>
8         (new KrylovFactory<SC, LO, GO, NO, MV, OP> ()));
9   // Add the new solver factory
10  sFactory.addFactory (solverFactory);
11
12  // Create a new solver
13  solver_ = sFactory.create(method, params_);
14
15  solver_->setProblem(problem_);
16
17  ...
18
19  problem_->setProblem();
20  solver_->solve();
```

Figure 2.3: Integration of the new Krylov solvers into Nalu.

```
1  OperatorCrsMatrix<SC, LO, GO, NT> *A;
2
3  Tpetra::MultiVector<> X (A->getDomainMap (), 1);
4  Tpetra::MultiVector<> B (A->getDomainMap (), 1);
5
6  // Setup Krylov solver
7  Krylov<> *solver;
8  // Pipeline
9  if (args.solverName == "PeeksGmresPipeline") {
10     solver = new GmresPipeline<> (rcpFromRef (*A));
11 } else if (args.solverName == "PeeksCgPipeline") {
12     solver = new CgPipeline<> (rcpFromRef (*A));
13 // Single-reduce
14 } else if (args.solverName == "PeeksGmresSingleReduce") {
15     solver = new GmresSingleReduce<> (rcpFromRef (*A));
16 } else if (args.solverName == "PeeksCgSingleReduce") {
17     solver = new CgSingleReduce<> (rcpFromRef (*A));
18 // s-step
19 } else if (args.solverName == "PeeksGmresSstep") {
20     solver = new GmresSstep<> (rcpFromRef (*A));
21 // Standard
22 } else if (args.solverName == "PeeksGmres") {
23     solver = new Gmres<> (rcpFromRef (*A));
24 } else {
25     solver = new Cg<> (rcpFromRef (*A));
26 }
27
28 // Set problem
29 RCP<Teuchos::ParameterList> params = rcp(new Teuchos::ParameterList ());
30 solver->setParameters (*params);
31
32 // Solve the system
33 solver->solve (X, B);
```

Figure 2.4: Interface to the new Krylov solvers using the base `Krylov` class.

CHAPTER 3

Experimental Setups

To study the solver performance on a candidate exascale architecture, we conducted our performance studies on the Haswell and Knights Landing (KNL) nodes of the Cori supercomputer at NERSC. Each of the Haswell nodes has two 16-core Intel Xeon E5-2698 v3 Haswell CPUs and 128 GB of main memory, while each of the KNL nodes has 68-core Intel Xeon Phi 7250 KNL CPUs on a single socket, and 16 GB of MCDRAM and 96 GB of DDR4 memories. These nodes are connected through the Cray Aries interconnect with Dragonfly topology.

We compiled Trilinos and our solver using the default Intel compiler (version 18.0.1 20171018) on Cori using the Cray compiler wrapper `CC`. On Haswell nodes, we launched one process per socket, while we launched one process per node on KNL nodes.

For this milestone, we decided to focus on 5-point 2D Laplace problems for which we have using for the previous studies. We are extending the current studies to include the matrices from the ECP applications.

CHAPTER 4

Performance Results

Figures 4.1 and 4.2 compare the performance of various CG solvers on the Haswell and KNL nodes of Cori, respectively. The single-reduce or pipelined variant of CG was able to obtain the speedups of up to $1.3\times$ over the standard CG. To improve the performance of the pipelined variant, we are currently testing different implementations of MPI with different environment variables to ensure the progress of the MPI communication behind the local computation or other MPI communication. We are planning to collaborate with ECP OMPI-X project.

Figure 4.3 shows the performance of different variants of GMRES solvers on the KNL nodes. For s-step GMRES, there are several algorithms for orthogonalizing a set of vectors [7]. For this milestone, we have decided to focus on Cholesky QR (CholQR) [10] that can orthogonalize the vectors with the minimum communication cost. We see that if CholQR is stable and the orthogonalization time is significant in the iteration time, s-step method may obtain a significant speedup over the standard method even on a single process, and its performance benefit can be maintained on multiple processes.

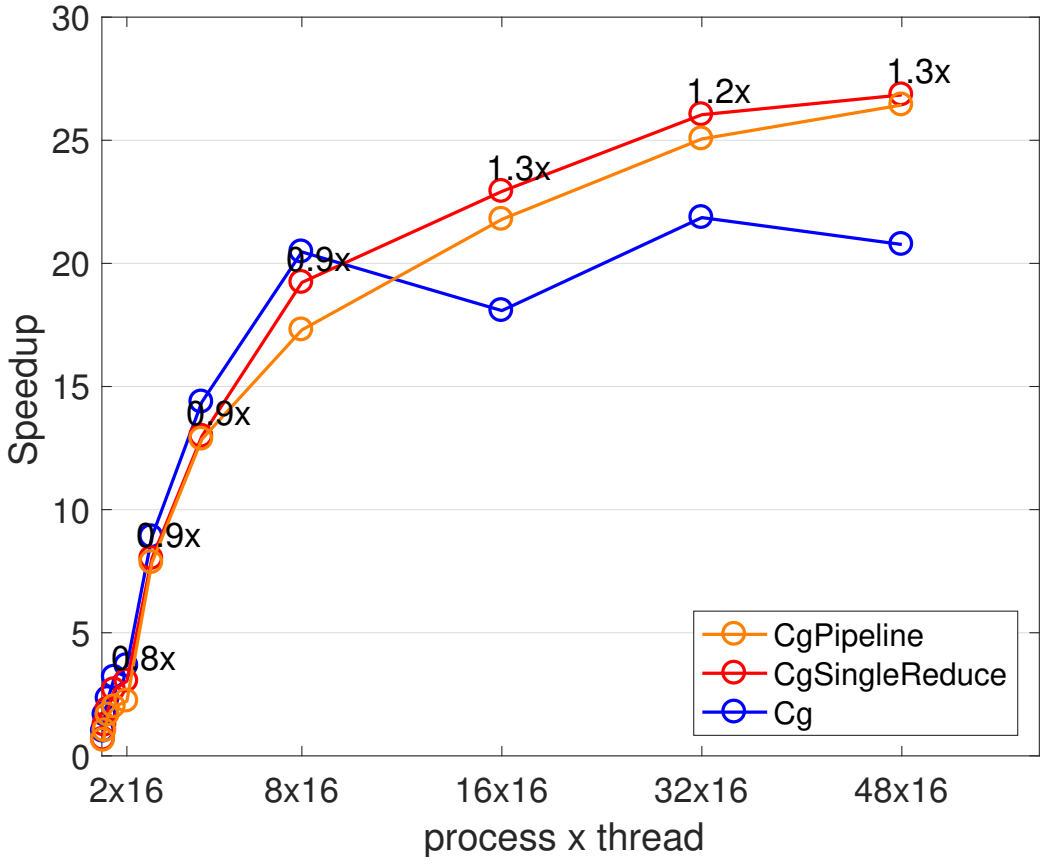

Figure 4.1: Performance of CG on Haswell nodes of Cori.

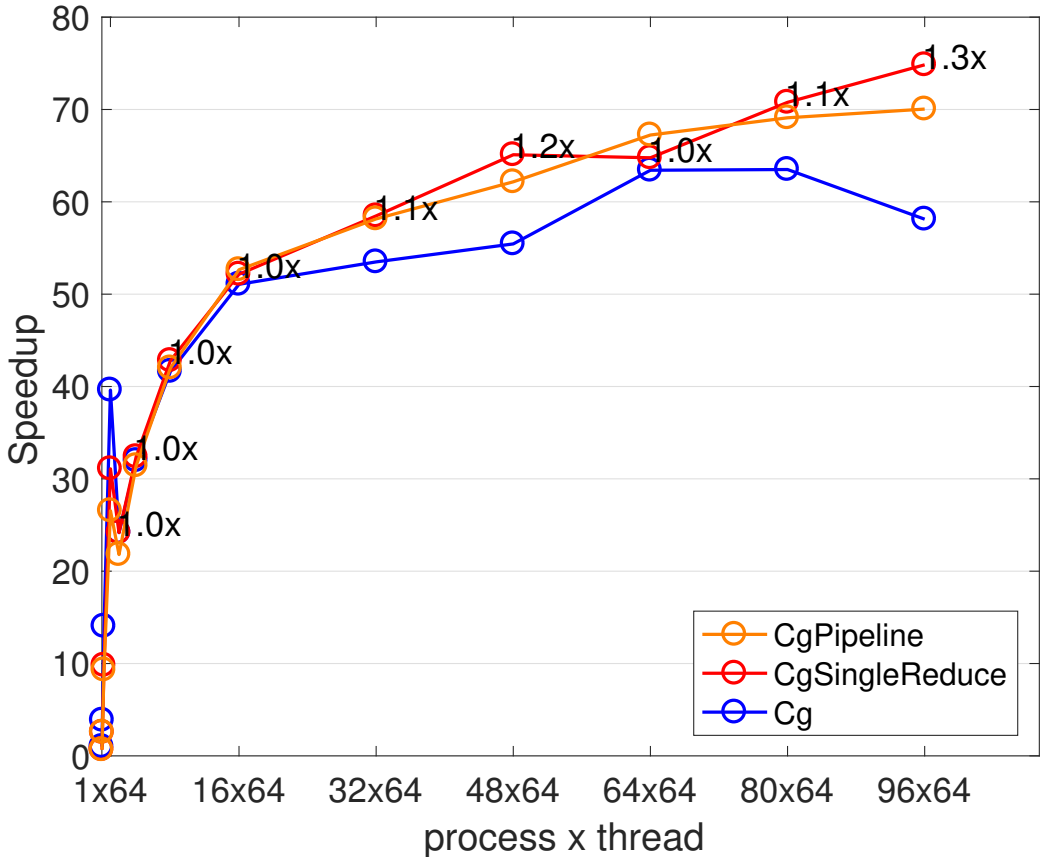

Figure 4.2: Performance of CG on KNL nodes of Cori.

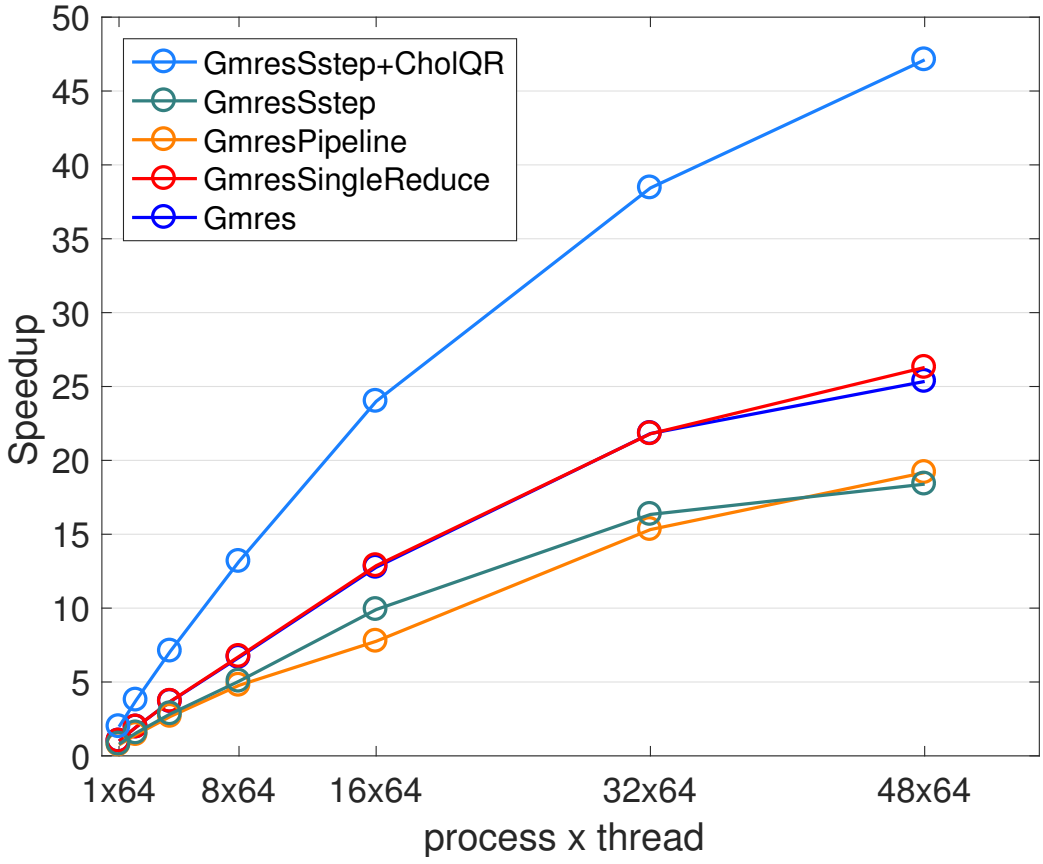

Figure 4.3: Performance of GMRES on KNL nodes of Cori.

CHAPTER 5

Conclusion

For this milestones, we have completed the development, testing, and documentation of the new PEEKS Krylov solvers. We have also reported the performance of the new solvers on the Cori supercomputer at NERSC using 2D model problems. We are currently working to improve the performance of the new solvers. In particular, we have already verified that the new solvers can be integrated into an ECP application and are testing the solver performance using the matrices from the ECP applications.

Bibliography

[1] Nalu: a generalized unstructured massively parallel low mach flow code. URL https://github.com/spdomin/Nalu.

[2] E. Bavier, M. Hoemmen, S. Rajamanickam, and H. Thornquist. Amesos2 and Belos: Direct and iterative solvers for large sparse linear systems. *Scientific Programming*, 20(3):241–255, 2012.

[3] P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Computing*, 40(7):224 – 238, 2014. 7th Workshop on Parallel Matrix Algorithms and Applications.

[4] P. Ghysels, T. J. Ashby, K. Meerbergen, and W. Vanroose. Hiding global communication latency in the GMRES algorithm on massively parallel machines. *SIAM Journal on Scientific Computing*, 35(1):C48–C71, 2013.

[5] M. Heroux, R. Bartlett, V. Howle, R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, A. Williams, and K. Stanley. An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31 (3):397–423, 2005.

[6] Magnus R. Hestenes. The conjugate gradient method for solving linear systems. In *Proc. Symp. Appl. Math VI, American Mathematical Society*, pages 83–102, New York, 1956. McGraw-Hill.

[7] M. Hoemmen. *Communication-avoiding Krylov subspace methods*. PhD thesis, EECS Department, University of California, Berkeley, 2010.

[8] Mark Hoemmen and Ichitaro Yamazaki. Production implementations of pipelined and communication-avoiding iterative linear solvers. SIAM conference on parallel processing for scientific computing, 2018.

[9] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[10] A. Stathopoulos and K. Wu. A block orthogonalization procedure with constant synchronization requirements. *SIAM J. Sci. Comput.*, 23:2165–2182, 2002.