Performance Test of Parallel Linear Equation Solvers on Blue Waters – Cray XE6/XK7 system

JaeHyuk Kwack, Gregory H Bauer, and Seid Koric
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
e-mail: {jkwack2, gbauer, koric}@illinois.edu

Abstract— **Parallel linear equation solvers are one of the most important components determining the scalability and efficiency of many supercomputing applications. Several groups and companies are leading the development of linear system solver libraries for HPC applications. In this paper, we present an objective performance test study for the solvers available on a Cray XE6/XK7 supercomputer, named Blue Waters, at National Center for Supercomputing Applications (NCSA). A series of non-symmetric matrices are created through mesh refinements of a CFD problem. PETSc, MUMPS, SuperLU, Cray LibSci, Intel PARDISO, IBM WSMP, ACML, GSL, NVIDIA cuSOLVER and AmgX solver are employed for the performance test. CPU-compatible libraries are tested on XE6 nodes while GPU-compatible libraries are tested on XK7 nodes. We present scalability test results of each library on Blue Waters, and how far and fast the employed libraries can solve the series of matrices.**

Keywords-parallel linear equation solver; dense direct solver; sparse direct solver; sparse iterative solver; CPU-compatible library; GPU-compatible library

I. INTRODUCTION

Solving linear system of equations is responsible for 70%–80% of the total computational time in many problems in computational science and engineering such as continuum and quantum mechanics, multi-physics, geophysics, optimization, linear programing, circuit design etc. Parallel linear equation solvers are therefore one of the most important components determining the scalability and efficiency of many supercomputing applications. Since no one wants to reinvent the wheel, most of HPC simulation programs employ standard libraries in order to solve their system of equations. At the present time, several groups and companies are leading the development of linear system solver libraries, and they have released their own performance test results in public. In this paper, we present an objective performance test results for parallel linear system solvers on Blue Waters.

Blue Waters [1, 2] is one of the most powerful supercomputers in the world. It can complete more than 1 Peta-Flops per second on a sustained basis and more than 13 times that at peak speed. Blue Waters is supported by the National Science Foundation (NSF) and the University of Illinois; the National Center for Supercomputing Applications (NCSA) manages the Blue Waters project and provides expertise to help scientists and engineers take full advantage of the system for their research. The system opened up to the science community at large on March 28, 2013. Blue Waters is a Cray XE6/XK7 system consisting of more than 22,500 XE6 compute nodes augmented by more than 4200 XK7 compute nodes in a single Gemini interconnection fabric. The XE6 nodes are populated with 2 AMD Interlagos model 6276 CPU processors and 64 GB of physical memory. The XK7 nodes are equipped with one Interlagos model 6276 CPU processor with 32 GB system memory and one NVIDIA GK110 "Kepler" accelerator K20X with 6 GB device memory.

A series of non-symmetric matrices is generated from a CFD problem and the matrix size is from 10 thousand equations to 5.5 million equations. We employ dense direct solvers, sparse direct solvers, sparse iterative solvers on CPUs and GPUs: Portable, Extensible Toolkit for Scientific Computation (PETSc) [3], MUltifrontal Massively Parallel sparse direct Solver (MUMPS) [4], Supernodal LU (SuperLU) [5], Cray LibSci, Intel Math Kernel Library Parallel Direct Sparse Solver (MKL PARDISO), IBM Watson Sparse Matrix Package (WSMP) [6], AMD Core Math Library (ACML), GNU Scientific Library (GSL), NVIDIA cuSOLVER, and NVIDIA AmgX library [7]. The performance tests are categorized into two groups: single-node tests for the best SMP performance and multiple-node tests for the best interconnect performance. According to Blue Waters charging policy, we compare the performance of the solvers based on the same number of XE/XK nodes. Since the purpose of this study is to compare the performance of solvers, we exclude the cost of I/O to read matrices and write solutions from the comparison.

II. TEST PROBLEM AND LIBRARIES

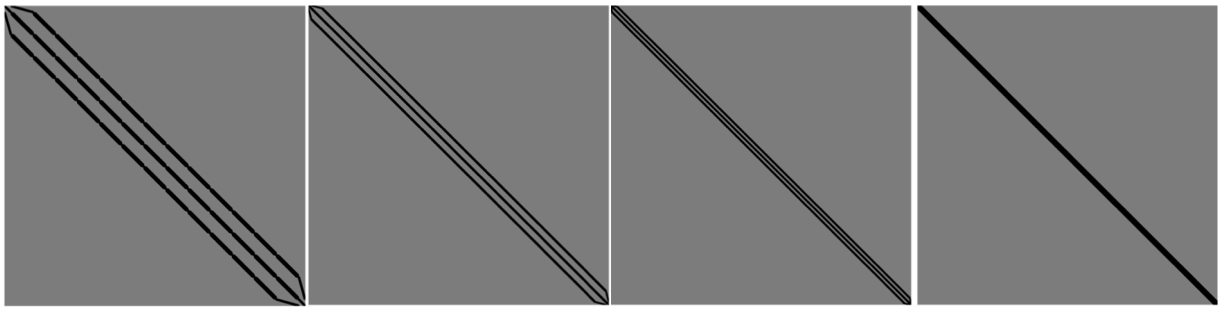
A. Non-symmetric matrices from a CFD simulation

In this study, we employ a variational multi-scale finite element method for computational fluid dynamics [8] in order to generate a series of test matrices. The stabilized mixed finite element code provides a nonlinear, consistent tangent matrix for the given residual vector. Due to the Eulerian specification of the flow field, the convective-time derivative term in the Navier-Stokes equation produces non-

Table 1. Characteristics of non-symmetric matrices

Matrix	Number of finite elements	Number of equations	Number of non-zero components	Sparsity (Density) (%)	Condition number*	File size in CSR format
LHM14	14^3	9,965	891,083	0.8974	1.66E+04	32 MB
LHM20	20^3	29,837	2,835,299	0.3185	6.33E+04	102 MB
LHM24	24^3	52,125	5,066,723	0.18648	1.25E+05	181 MB
LHM28	28^3	83,437	8,239,907	0.11836	2.25E+05	294 MB
LHM30	30^3	102,957	10,231,499	0.09652	2.94E+05	365 MB
LHM32	32^3	125,309	12,520,739	0.07974	3.75E+05	447 MB
LHM36	36^3	179,277	18,075,107	0.05624	5.91E+05	645 MB
LHM46	46^3	377,197	38,621,387	0.02715	1.54E+06	1.4 GB
LHM56	56^3	684,317	70,756,067	0.015109	3.33E+06	2.5 GB
LHM60	60^3	843,117	87,435,299	0.012300	4.38E+06	3.1 GB
LHM62	62^3	930,989	96,677,579	0.011154	4.99E+06	3.4 GB
LHM64	64^3	1,024,756	106,549,283	0.010146	5.65E+06	3.8 GB
LHM66	66^3	1,124,637	117,071,147	0.009256	6.39E+06	4.1 GB
LHM68	68^3	1,230,797	128,263,907	0.008467	7.19E+06	4.5 GB
LHM70	70^3	1,343,437	140,148,299	0.007765	8.07E+06	4.9 GB
LHM80	80^3	2,010,557	210,670,499	0.005212	1.37E+07	7.4 GB
LHM112	112^3	5,545,789	586,210,979	0.0019060	5.27E+07	21 GB

* Condition number is estimated via the error analysis routine with full statics of MUMPS (i.e., ICNTL(11) is set to 1)



(a) LHM14 (b) LHM28 (c) LHM56 (d) LHM112

Figure 1. Graphical representation for non-zero components (non-zeros: black, zeros: gray)

symmetrical entries in the matrices. This non-symmetric aspect requires a totally different approach for linear equation solvers in CFD from computational simulations in solid mechanics or other elliptic problems with symmetric matrices. The test problem is a body force-driven fluid flow in a unit volume of cubic domain. It is filled with 8-node linear hexahedral finite elements. Each node of elements has 4 degrees-of-freedom (DOFs) composed of the velocity vector and the scalar pressure. The body force presented in [8] is prescribed in every Gaussian quadrature point in elements; as a result, the body force is directly injected into the residual vector. The Dirichlet boundary condition is applied for the velocity field on the boundary surface, while the pressure is set to zero only at the centroid of the domain. A series of non-symmetric consistent tangent matrices are generated through the mesh refinement. An equal number of elements are assigned along *x*-, *y*- and *z*-axis. The number of elements along each axis (i.e., *n*) is used as the index of the linear hexahedral mesh (e.g., LHM*n*) that is composed of n^3 hexahedral elements in the domain. The matrices are saved in a compressed row storage format (CSR, CRS or Yale format) in double precision, and then they are converted into coordinate list (COO) format or matrix market (MM) format as required. The sizes of matrices are from 10 thousands to 5.5 millions equations, and the numbers of non-zero components are from 0.9 millions to 0.6 billions respectively. Table 1 presents characteristics of non-symmetric matrices tested in this study. Figure 1 shows the graphical representation for non-zero components. The condition number of non-symmetric matrices grows from 16,600 to 52.7 millions as the number of equations increases as presented in Figure 2.

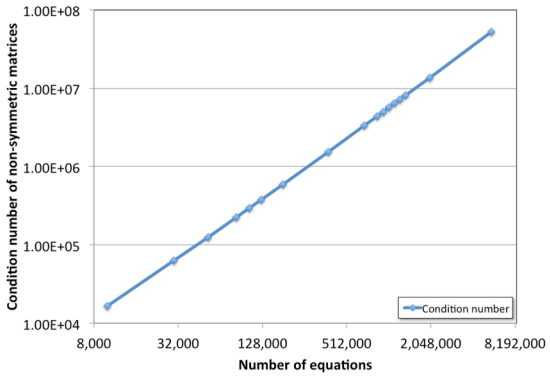


Figure 2. Condition numbers of non-symmetric matrices

B. *Employed Linear Equation Solver Libraries*

We employ PETSc, MUMPS, SuperLU, Cray LibSci, Intel MKL PARDISO, IBM WSMP, ACML, GSL, NVIDIA cuSOLVER, and NVIDIA AmgX library for the performance test. Cray-optimized modules are used for some solver libraries, and we install other libraries on Blue Waters for this study.

The Cray modules tested in this study are as follows:

- Cray LibSci version 13.3.0 for an optimized LAPACK with SMP performance on Cray XE nodes,
- Cray PETSc version 3.6.1.0 for KSP Linear Equations Solvers on Cray XE nodes,
- Cray TPSL version 1.5.2 for MUMPS, SuperLU_DIST and ParMetis on Cray XE nodes,
- Intel Composer XE version 15.0.3.187 for Intel MKL PARDISO solver on Cray XE nodes,
- ACML version 5.1.1 for an optimized LAPACK with SMP performance on Cray XE nodes,
- GSL version 1.16-2015-04 for LAPACK on Cray XE nodes,
- NVIDIA cudatoolkit version 7.0.28-1.0502.10742.5.1 for cuSolver on Cray XK nodes.

The followings are the user-built libraries on Blue Waters:

- SuperLU_DIST version 4.3 for MPI on Cray XE nodes,
- MUMPS version 5.0.0 for MPI on Cray XE nodes,
- IBM WSMP version 16.01.10 on Cray XE nodes,
- NVIDIA AmgX version 1.2.1-build112 and 1.2.0-build108 on Cray XK nodes.

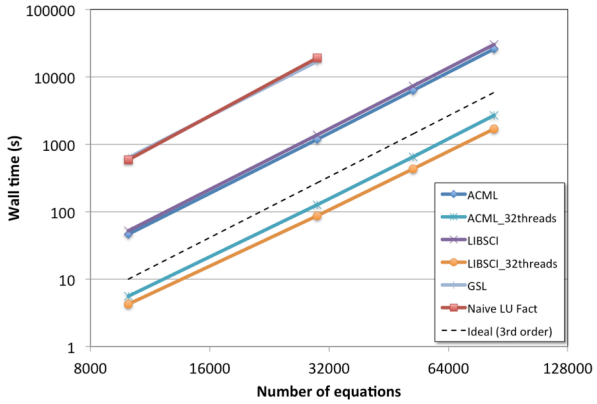


Figure 3. Wall time of dense direct solver on an XE node

III. SINGLE-NODE TEST RESULTS FOR THE BEST SMP PERFORMANCE

In many HPC applications, the optimized single-node performance is one of the most crucial building blocks for the optimization of probabilistic analysis or hybrid MPI-OpenMP/ OpenACC implementation. In this section, one XE6 or XK7 node on Blue Waters is assigned to solve the series of test matrices. Cray LibSci, ACML, GSL, MUMPS, SuperLU, Intel MKL PARDISO, and PETSc are performed on a single XE node. NVIDIA cuSolver and AmgX solvers are executed on one XK node.

A. *Dense matrix direct solvers on an XE node – Cray LibSci vs. ACML vs. GSL vs. Naïve LU factorization*

The LU factorization routine in Cray LibSci, ACML and GSL are employed in this test. As a reference, a naïve LU factorization code is also written and tested. For Cray LibSci and ACML, a simple code using DGESV subroutine is written in Fortran 90, and built with corresponding modules. For GSL, a small code using GSL_LINALG_LU_DECOMP and GLS_LINALG_LU_SOLVE subroutines is written in C and built with the GSL module. Cray LibSci and ACML codes are tested with a single thread as well as 32 threads (i.e., one thread per one integer core), while GSL and naïve codes are run only with a single thread. Since the LU factorization routine solves for a dense matrix, the performance is bounded by the memory capacity. Cray LibSci and ACML complete the solution process for matrices with up to 83,437 equations (i.e., up to LHM28). GSL and the naïve code are tested only for LHM14 and LHM20 due to unacceptably long processing time with a single thread. Speedups of Cray LibSci and ACML with 32 threads are 12.3 – 17.8 and 8.4 – 9.8, respectively as presented in Table 2. All cases follow the third-order time complexity that is ideal for dense matrix direct solvers as shown in Figure 3.

B. *Sparse matrix direct solvers on an XE node – MUMPS vs. SuperLU-DIST vs. Intel MKL PARDISO*

Parallel direct solvers for non-symmetric sparse matrices are tested on a single XE node. MUMPS and SuperLU-DIST libraries from Cray TPSL module are executed with 16 MPI processes (i.e., one MPI-rank per one Bulldozer core) on an XE node, while Intel MKL PARDISO uses 16 threads (i.e., one thread per one Bulldozer core). Since sparse direct solvers are more efficient in memory usages than dense direct solvers, they can solve larger matrices than LHM28 that is the biggest solvable matrices of dense direct solvers on one XE node. MUMPS solves the system of the equations

Table 2. Wall time of dense direct solvers on an XE node

Matrix	Number of equations	Cray LibSci			ACML			GSL	Naïve LU
		1 thread	32 threads*	Speedup	1thread	32 threads	Speedup	1 thread	1 thread
LHM14	9,965	52.13 s	4.23 s	12.3	46.45 s	5.55 s	8.4	631.3 s	582.38 s
LHM20	29,837	1373.7 s	86.27 s	15.9	1198.3 s	125.52 s	9.5	16818.9 s	19078.4 s
LHM24	52,125	7325.4 s	429.7 s	17.0	6348.2 s	649.97 s	9.8	-	-
LHM28	83,437	30017 s	1689.2 s	17.8	25833.7 s	2671.2 s	9.7	-	-

* one thread per one integer core (two threads per one Bulldozer core)

for matrices up to LHM56 with 684,317 equations, and SuperLU-DIST is good enough to solve LHM62 with 930,989 equations. Intel MKL PARDISO accomplishes the most optimized memory usage performance, so it can solve the largest matrix (i.e., LHM68 with 1,230,797 equations) among sparse direct solvers on one XE node. Table 3 shows wall time of each solver on an XE node with 16 Bulldozer cores. For most matrices, they show the second-order time complexity that is ideal for sparse direct solvers, as presented in Figure 4. Intel MKL PARDISO and MUMPS even establish the first-order time complexity for small size of matrices.

Table 3. Wall time of parallel sparse direct solvers on an XE node with 16 Bulldozer cores

Matrix	Number of equations	MUMPS (Cray TPSL)	Intel MKL PARDISO	SuperLU (Cray TPSL)
LHM14	9,965	0.3808 s	0.4317 s	0.4348 s
LHM20	29,837	1.3864 s	1.333 s	1.7960 s
LHM24	52,125	3.027 s	2.860 s	3.776 s
LHM28	83,437	6.170 s	5.889 s	7.790 s
LHM30	102,957	9.485 s	8.051 s	10.689 s
LHM32	125,309	12.650 s	11.142 s	14.564 s
LHM36	179,277	22.85 s	20.79 s	25.79 s
LHM46	377,197	83.91 s	92.52 s	88.41 s
LHM56	684,317	295.2 s	248.3 s	264.9 s
LHM60	843,117	OOM	408.1 s	385.8 s
LHM62	930,989	OOM	477.8 s	455.2 s
LHM64	1,024,756	OOM	601.5 s	OOM
LHM66	1,124,637	OOM	696.9 s	OOM
LHM68	1,230,797	OOM	872.2 s	OOM
OOM: Out of memory during the factorization process				

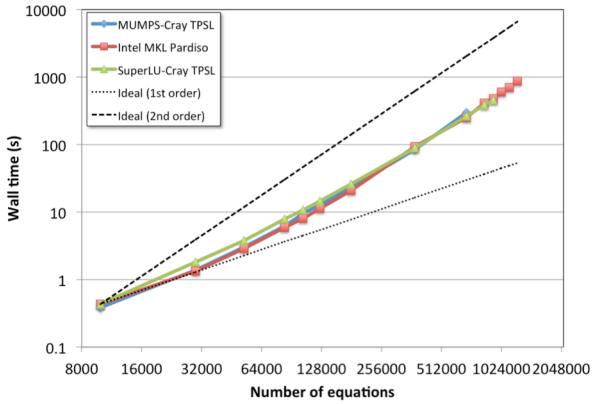


Figure 4. Wall time of sparse direct solver on an XE node

C. *Direct solvers on an XK node – cuSolver_dense_QR vs. cuSolver_sparse_QR*

The cuSolver library is a high-level package based on the cuBLAS and cuSPARSE libraries, and it provides useful LAPACK-like features, such as common matrix factorization and triangular solve routines for dense matrices, a sparse least-squares solver and an eigenvalue solver. In addition, cuSolver provides a new re-factorization library useful for solving sequences of matrices with a shared sparsity pattern. Since CUDATOOLKIT 7.0 does not support LU factorization on GPU yet, we alternatively use QR factorization to solve the systems of linear equations. Table 4 presents simplified steps for dense QR factorization on GPU, sparse QR factorization on GPU and sparse QR factorization on CPU, implemented in this study.

Table 4. Summary of steps for QR factorization on GPU or CPU

cuSolver_dense_QR on GPU
step 1: copy matrices (i.e., A) and residual vectors (i.e., B) to device (i.e., GPU) step 2: solve A*x=B on GPU (call cusolverDnDgeqrf, cusolverDnDormqr, and cublasDtrsm subroutines on device) step 3: copy solution vectors (i.e., x) to host (i.e., CPU)
cusolver_sparse_QR on GPU
step 1: copy matrices (i.e., A) and residual vectors (i.e., B) to device (i.e., GPU) step 2: solve A*x = b on GPU (i.e., call cusolverSpDcsrlsvqr subroutine on device) step 3: copy solution vectors (i.e., x) to host (i.e., CPU)
cusolver_sparse_QR on CPU
step 1: solve A*x = b on CPU (i.e., call cusolveSpDcsrlsvqrHost subroutine on host)

The NVIDIA GK110 Kepler accelerator K20X on one XK node has 6GB of memory on the device; therefore, the performance of cuSolver is bounded by the memory capacity due to high demand for memory of the QR factorization. The dense QR solver on GPU can solve only LHM14 and the sparse QR solver on GPU completes the solution process for LHM14 and LHM20. For larger matrices, the programs return the out of memory (OOM) error. To confirm that it is not from another issue, sparse QR solver on CPU is tested. Since one Interlagos CPU processor on an XK node has 32 GB of system memory that is much larger than 6GB of memory on GPU, sparse QR on CPU can solver up to LHM28, as presented in Table 5. Figure 5 shows wall times of dense/sparse direct solvers on GPU and CPU. Sparse QR solvers on GPU and CPU follow the ideal 2nd order time complexity while direct QR solver on GPU follows the ideal 3rd time complexity.

Table 5. Wall time of direct cuSolvers on an XK node

Matrix	Number of equations	Dense QR on GPU	Sparse QR on GPU	Sparse QR on CPU (1 thread)
LHM14	9,965	7.04 s	6.63 s	76.1 s
LHM20	29,837	OOM	58.5 s	921.2 s
LHM24	52,125	OOM	OOM	3303.2 s
LHM28	83,437	OOM	OOM	10165.6 s

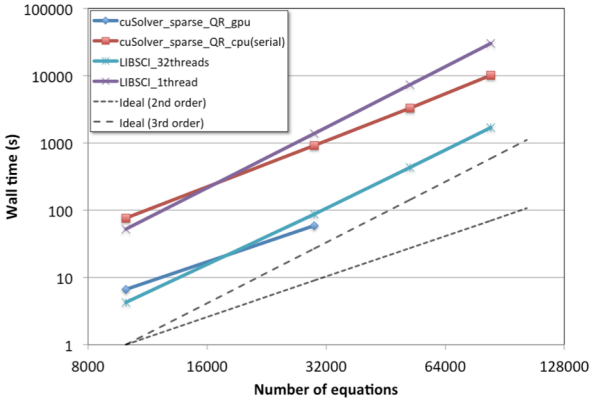


Figure 5. Wall time of direct cuSolvers on an XK node

D. *Sparse iterative solvers on an XE node - PETSc*

Sparse iterative solvers are useful for solving linear and nonlinear problems involving a large number of variables where sparse direct solvers would be prohibitively expensive. In this test, we use PETSc for sparse iterative solvers on an XE node. PETSc provides many popular Krylov subspace (KSP) iterative methods and a variety of preconditioners (PC). BiConjugate Gradient (bicg), Biconjugate gradient stabilized (bcgs), Generalized Minimal Residual (gmres), Flexible Generalized Minimal Residual (fgmres), Deflated Generalized Minimal Residual (dgmres), Generalized Conjugated Residual (gcr), Transpose-Free Quasi-Minimal Residual (tfqmr), and Tony Chan's Transpose-Free Quasi-Minimal Residual (tcqmr) KSP methods are tested with Jacobi, Block Jacobi (bjacobi), and Additive Schwarz (asm) preconditioners. The iterative process requires convergence test based on the L_2-norm of the residual. The convergence test is decided by three quantities: the decrease of the residual norm relative to the norm of the right hand side, *rtol*, the absolute size of the residual norm, *atol*, and the relative increase in the residual, *dtol*. Even without passing the convergence test, the iterative process stops when the number of total iterations reaches to the maximum number of allowable iteration, *maxits*. The default values are set to *rtol*=10^{-5}, *atol*=10^{-50}, *dtol*=10^{5}, and *maxits*=10^{4}. These values are okay with usual cases, but we employ much smaller value, 10^{-16} for *rtol* in order to establish a fair comparison with direct solvers, such as MUMPS, SuperLU, Intel MKL PARDISO, and IBM WSMP solvers.

Most combinations of KSP methods with preconditioners fail to get converged solutions. Table 6 shows a few successful combinations of KSP and PC that return converged solutions for the non-symmetric matrices. BCGS with ASM and DGMRES with ASM show limited numerical stabilities as condition number of matrices increases. BICG with ASM is better than them, but it does not yield a converged solution for LHM112. TFQMR with ASM can solve all of the non-symmetric matrices. Figure 6 shows wall times of the successful combinations of KSP and PC. All of the plots are drawn between 1st order and 2nd order time complexity lines.

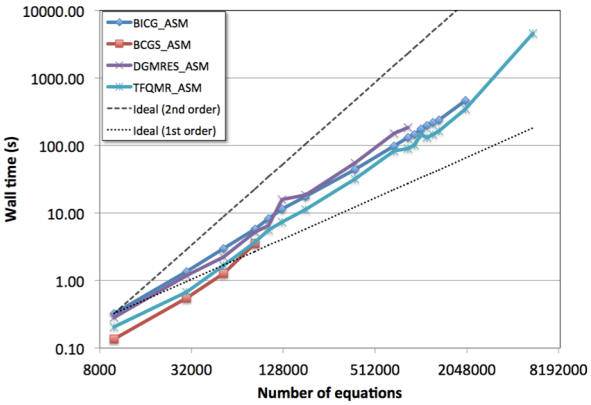


Figure 6. Wall time of iterative solvers on an XE node

E. *Sparse iterative solvers on an XK node – NVIDIA AmgX*

AmgX library provides a simple path to accelerated core solver technology on NVIDIA GPUs. Its flexible configuration allows for nested solvers, smoothers, and preconditioners. It supports Ruge-Steuben algebraic multigrid, un-smoothed aggregation algebraic multigrid, Krylov methods (e.g., PCG, GMRES, and BiCGStab), and smoothers such as Block-Jacobi, Gauss-Seidel, incomplete LU, Polynomial, and dense LU. In this sub-section, we use AmgX on one GPU in an XK node. CSR format matrices are converted to Matrix Market (MM) format. The maximum number of iterations and the relative residual tolerance for the convergence are set to 10,000 and 10^{-12}, respectively. The mode parameter for AmgX is set to dDDI; as a result, the code runs on the device (i.e., GPU) with double precision for the matrix and the vector, and the index type is set to 32-bit integer. The code is built with CUDATOOLKIT version 7.0.28-1.0502.10742.5.1 on Blue Waters. Table 7 shows the configuration used in this test. Table 8 and Figure 7 show

Table 6. Wall time of sparse iterative solvers on an XE node

Matrix	Number of equations	bicg (ksp) with asm (pc)	bcgs (ksp) with asm (pc)	dgmres (ksp) with asm (pc)	tfqmr (ksp) with asm (pc)
LHM14	9,965	0.32 s	0.14 s	0.28 s	0.21 s
LHM20	29,837	1.36 s	0.55 s	1.18 s	0.67 s
LHM24	52,125	2.95 s	1.26 s	2.22 s	1.65 s
LHM28	83,437	5.82 s	3.51 s	5.18 s	3.73 s
LHM30	102,957	8.30 s	NC	6.53 s	5.57 s
LHM32	125,309	11.50 s	NC	15.72 s	7.29 s
LHM36	179,277	17.46 s	NC	18.51 s	11.24 s
LHM46	377,197	44.04 s	NC	54.51 s	31.92 s
LHM56	684,317	98.67 s	NC	149.09 s	82.30 s
LHM60	843,117	130.06 s	NC	182.68 s	89.53 s
LHM62	930,989	144.49 s	NC	NC	100.61 s
LHM64	1,024,756	174.46 s	NC	NC	145.06 s
LHM66	1,124,637	197.02 s	NC	NC	130.24 s
LHM68	1,230,797	217.62 s	NC	NC	145.19 s
LHM70	1,343,437	236.11 s	NC	NC	164.11 s
LHM80	2,010,557	463.37 s	NC	NC	346.59 s
LHM112	5,545,789	NC	NC	NC	4538.61 s
NC: not converged (it means total number of iteration is equal to the maximum allowable iterations).					

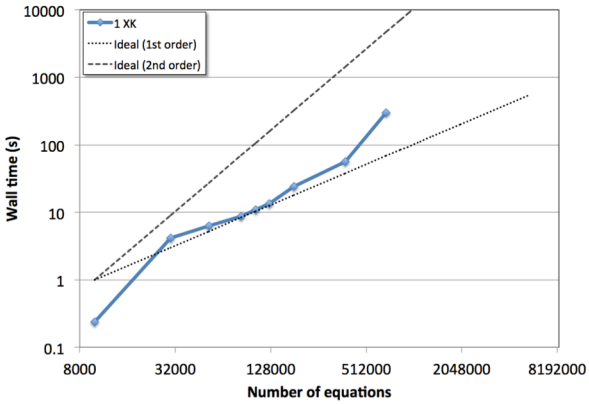
wall time, number of iterations and total reduction in residual for the non-symmetric matrices. AmgX can solve matrices up to LHM56 with satisfying the convergence criteria. For matrices from LHM60 to LHM80, AmgX reaches to the maximum allowable iterations. For LHM112, it returns out-of-memory error message.

Table 7. Configuration for AmgX solver

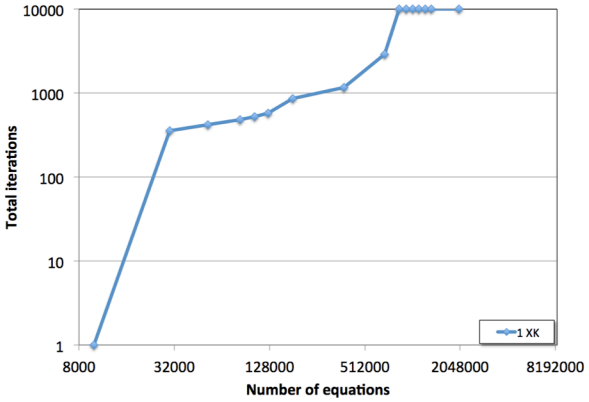
```
{  "config_version": 2,
   "solver": {"print_grid_stats": 1,
              "store_res_history": 1,
              "solver": "FGMRES",
              "print_solve_stats": 1,
              "obtain_timings": 1,
              "preconditioner": {
                  "interpolator": "D2",
                  "print_grid_stats": 1,
                  "aggressive_levels": 1,
                  "solver": "AMG",
                  "smoother": {
                      "relaxation_factor": 1,
                      "scope": "jacobi",
                      "solver": "JACOBI_L1"},
                  "presweeps": 2,
                  "selector": "PMIS",
                  "coarsest_sweeps": 1,
                  "coarse_solver": "DENSE_LU_SOLVER",
                  "max_iters": 1,
                  "max_row_sum": 0.9,
                  "strength_threshold": 0.25,
                  "min_coarse_rows": 2,
                  "scope": "amg_solver",
                  "max_levels": 24,
                  "cycle": "V",
                  "postsweeps": 2},
              "max_iters": 10000,
              "monitor_residual": 1,
              "gmres_n_restart": 500,
              "convergence": "RELATIVE_INI_CORE",
              "tolerance": 1e-12,
              "norm": "L2"}}
```

Table 8. Wall time of sparse iterative solvers on an XK node

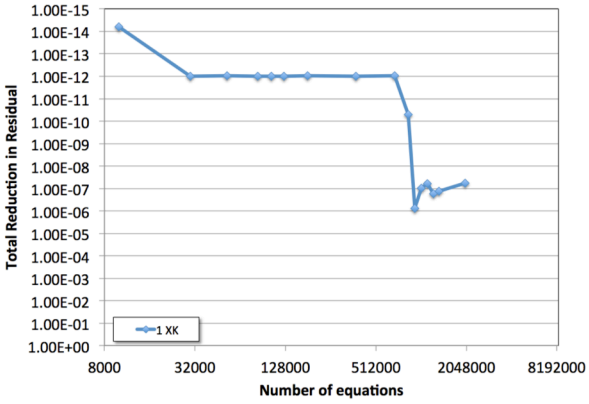
Matrix	Number of equations	Wall time	Number of iterations	Total reduction in residual
LHM14	9,965	0.24 s	1	6.42E-15
LHM20	29,837	4.18 s	357	9.96E-13
LHM24	52,125	6.25 s	420	9.29E-13
LHM28	83,437	8.79 s	483	9.95E-13
LHM30	102,957	10.98 s	529	9.81E-13
LHM32	125,309	13.44 s	581	9.85E-13
LHM36	179,277	24.28 s	867	9.52E-13
LHM46	377,197	55.79 s	1175	9.94E-13
LHM56	684,317	298.25 s	2922	9.70E-13
LHM60	843,117	NC	10000	5.07E-11
LHM62	930,989	NC	10000	7.54E-07
LHM64	1,024,756	NC	10000	9.78E-08
LHM66	1,124,637	NC	10000	6.22E-08
LHM68	1,230,797	NC	10000	1.67E-07
LHM70	1,343,437	NC	10000	1.32E-07
LHM80	2,010,557	NC	10000	5.73E-08
LHM112	5,545,789	OOM		



(a) Wall time



(b) Number of iterations



(c) Total reduction in residual

Figure 7. Performance of iterative solvers on an XK node

IV. MULTIPLE-NODES TEST RESULTS FOR THE BEST INTERCONNECT PERFORMANCE

Large-scale computation for science and engineering projects usually requires a numerical solution of an extensive size of system of equations. For this purpose, MUMPS, IBM WSMP, SuperLU-DIST, PETSc, and NVIDIA AmgX solvers are employed to check their performance in this section.

A. *Sparse direct solvers on XE nodes – MUMPS vs. SuperLU-DIST vs. IBM WSMP*

MUMPS, SuperLU-DIST and IBM WSMP solver libraries are built under PrgEnv-pgi on Blue Waters. MUMPS and SuperLU use Cray-LibSci for BLAS subroutines, and WSMP links against Intel MKL to fully support POSIX threads in WSMP. MUMPS and SuperLU link against METIS for reordering, and WSMP has its own ordering algorithm. For LHM56 and LHM80, WSMP uses 4 MPI-ranks/node with 4 threads while MUMPS and SuperLU use 16 MPI-ranks/node. For LHM112, the performance of sparse direct solvers is bounded by the memory capacity of XE nodes, so number of cores and threads per node need to drop to the half. As a result, WSMP uses 1 MPI-rank/node with 8 threads, and MUMPS and SuperLU use 8 MPI-ranks/node for LHM112

The solution process of sparse direct solvers is composed of three steps: pre-factorization, numerical factorization and forward/backward substitution. Pre-factorization step is for reordering and symbolic factorization required before the numerical factorization. In a typical full Newton-Raphson nonlinear solution scheme, the reordering and symbolic factorization is required only one time, if the sparsity pattern of non-symmetric matrices keeps the same. In usual CFD problems without updating mesh connectivity (e.g., without adaptive meshing technique), the cost of pre-factorization step is negligible in general. Numerical factorization step is required when non-zero components of non-symmetric matrices are updated. Even though the consistent tangent matrix is always updated during non-linear iterations in CFD problems, the factorization step may be executed once a time-step under pseudo-linear conditions (e.g., laminar flows) in so called modified Newton-Raphson scheme. The forward/backward substitution step is required at every iteration-step and time-step regardless if the Newton-Raphson scheme is full or modified. For linear and pseudo-linear problems (e.g., laminar flows), the performance of forward/backward substitution step is the most important component to determine the overall performance, since it is the most repeated step during the simulations. However, in typical nonlinear problems solving the Navier-Stokes equations, both the numerical factorization and forward/backward substitution steps are crucial building blocks to determine the performance of general CFD simulations (e.g., transitional or turbulent flows).

Tables 9, 10 and 11 and Figure 8 present numerical test results of MUMPS, SuperLU, and WSMP solvers for LHM56, LHM80 and LHM112. Wall times are divided into

Table 9. Wall time of sparse direct solvers on XE nodes – LHM56

Nodes	Cores	MUMPS					WSMP					SuperLU_DIST				
		PreFac	Fac	Solve	Total	Tflops	PreFac	Fac	Solve	Total	Tflops	PreFac	Fac	Solve	Total	Tflops
1	16	32	284	9	325	0.085						44	393	1	438	0.06
2	32	33	182	7	222	0.134	74	201	0.6	276	0.123	42	193	0.7	236	0.127
4	64	34	100	5	139	0.244	73	95	0.6	169	0.258	40	96	0.5	137	0.25
8	128	38	75	5	118	0.325	71	70	0.4	141	0.348	39	49	0.4	88	0.49
16	256	42	56	5	103	0.435	70	39	0.4	109	0.622	39	28	0.4	67	0.88
32	512	42	51	5	98	0.478	71	20	0.3	91	1.23	39	18	0.4	57	1.37
64	1024						70	11	0.3	81	2.12	39	14	0.4	53	1.79

Nodes := number of XE nodes
Cores := number of MPI-rank * number of threads. (4 threads/MPI-rank for WSMP, pure MPI for MUMPS and SuperLU_DIST)
PreFac := elapsed time for pre-factorization process including reordering and symbolic factorization
Fac := elapsed time for factorization process
Solve := elapsed time for forward and backward substitution process
Total := PreFac + Fac + Solve
Tflops := Tera Flops for the factorization process

Table 10. Wall time of sparse direct solvers on XE nodes – LHM80

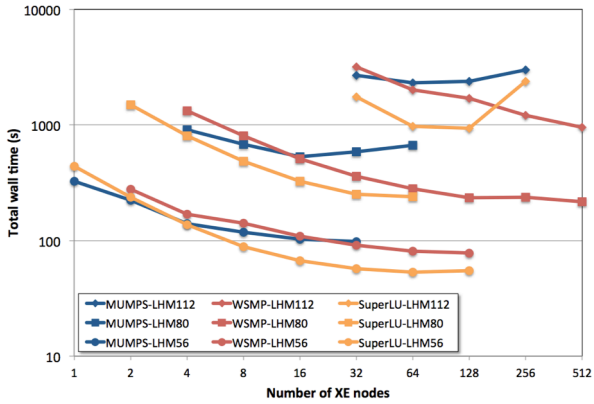
Nodes	Cores	MUMPS					WSMP					SuperLU_DIST				
		PreFac	Fac	Solve	Total	Tflops	PreFac	Fac	Solve	Total	Tflops	PreFac	Fac	Solve	Total	Tflops
2	32											140	1345	2	1487	0.87
4	64	108	780	20	908	0.27	183	1137	1.7	1322	0.19	140	664	2	806	0.33
8	128	122	537	20	679	0.4	183	615	1.5	800	0.35	139	342	2	483	0.61
16	256	138	374	18	530	0.57	184	323	1	508	0.66	140	182	2	324	1.1
32	512	181	386	18	585	0.55	195	163	0.7	359	1.31	138	111	2	251	1.9
64	1024	234	414	18	666	0.51	194	86	0.7	281	2.52	139	98	2	239	2.1
128	2048						190	44	0.7	235	4.93					
512	8192						194	21	0.6	216	10.1					

4 threads/MPI-rank for WSMP; pure MPI for MUMPS and SuperLU_DIST

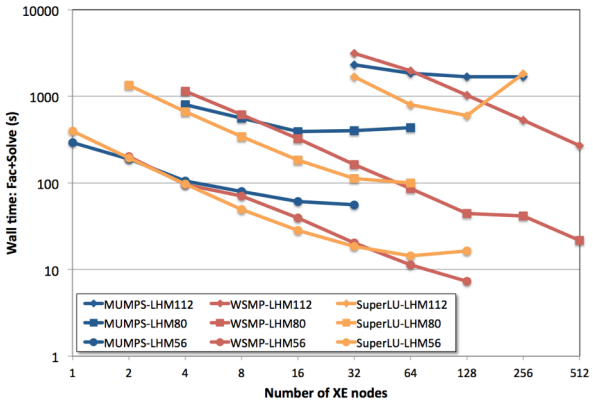
Table 11. Wall time of sparse direct solvers on XE nodes – LHM112

Nodes	Cores	MUMPS					WSMP					SuperLU_DIST				
		PreFac	Fac	Solve	Total	Tflops	PreFac	Fac	Solve	Total	Tflops	PreFac	Fac	Solve	Total	Tflops
32	256	371	2241	60	2672	0.75	676	2501	2.8	3180	0.7	80	1663	11	1754	1.93
64	512	464	1785	60	2309	0.93	683	1326	2.6	2012	1.3	178	788	9	975	3.46
128	1024	714	1611	61	2386	1.05	676	1021	2.5	1697	1.6	334	591	9	934	4.78
256	2048	1317	1608	68	2993	1.04	676	531	1.8	1209	3.2	566	1798	17	2381	1.43
512	4096						685	267	1.7	951	6.4					

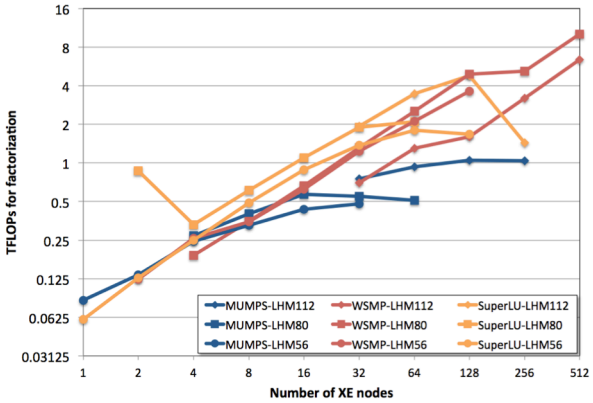
8 threads/MPI-rank for WSMP; pure MPI for MUMPS and SuperLU_DIST



(a) Total wall time



(b) Wall time for Fac+Solve



(c) TFLOPs for factorization process

Figure 8. Test results of sparse direct solvers for LHM112, LHM80 and LHM56 on XE nodes

three steps such as PreFac (i.e., pre-factorization), Fac (i.e., numerical factorization), and Solve (i.e., forward/backward substitution), and Total shows the summation of elapsed times for three steps. With small numbers of XE nodes, MUMPS and SuperLU are faster than WSMP, and WSMP shows better scalability than MUMPS and SuperLU with large numbers of XE nodes. Figure 8 (b) shows wall times for numerical factorization and forward/backward substitution steps and it represents tentative performance of sparse direct solvers for general CFD problems in a typical full NR scheme. In this plot, WSMP achieves the best scalability and fastest times compared to others on large number of nodes.

WSMP use dynamic pivoting as the default, while SuperLU supports only static pivoting. Dynamic pivoting generally provides better stability but it is slower in numerical factorization than static pivoting. Table 12 and Figure 9 show wall time for dynamic and static pivoting of WSMP for LHM112. They show static pivoting is 2-34% faster than dynamic pivoting in numerical factorization of WSMP, while accurate solutions are still obtained from the solution process.

As presented in this sub-section, sparse direct solvers require much larger memory capacity than sparse iterative solvers. However, sparse direct solvers show outstanding stability and robustness for ill-posed problems with very high condition numbers. Koric and his colleagues [9] reported that some symmetric matrices with higher condition number than 10^8, proved difficult for iterative solvers, and some symmetric matrices with 10^9 for the condition number was

Table 12. Wall time for factorization of WSMP for LHM112 (Dynamic vs. Static pivoting)

XE nodes	Dynamic pivoting	Static pivoting	Dynamic/ Static
32	2501 s	2450 s	102 %
64	1326 s	1297 s	102 %
128	1021 s	787 s	130 %
256	531 s	396 s	134 %
512	267 s	224 s	119 %

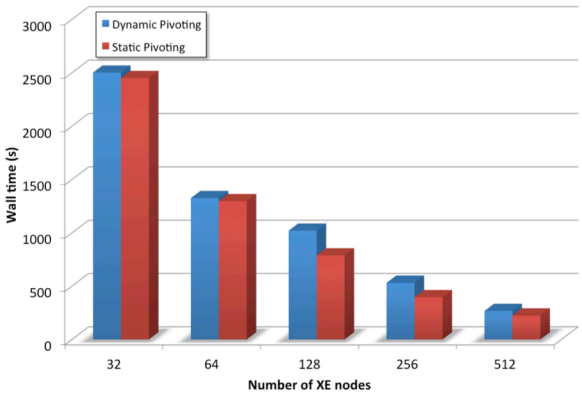


Figure 9. Comparison of wall time for WSMP factorization between dynamic and static pivoting – LHM112

not converged for all combinations of iterative methods and preconditioners in PETSc and HYPRE. As the size, speed, and availability of multi/many core architectures, memory, and interconnect network grows in HPC, the usage of direct spare solver may increase in future HPC systems.

B. *Sparse iterative solvers on XE nodes – PETSc*

Based on the single node test results in the previous section, the combination of TFQMR for iterative solver and ASM for preconditioner is employed to solve LHM112 on multiple XE nodes. This combination works fine with 1, 2 and 4 XE nodes, but it does not yield a converged solution with more than 4 XE nodes, as presented in Table 13 and Figure 10.

Table 13. Wall time and total iterations of TFQMR (ksp) with ASM (pc) for LHM112 on XE nodes

Number of XE nodes	Cores*	Wall time	Number of iterations
1	16	4538.61 s	2469
2	32	2100.78 s	2904
4	64	1653.03 s	4859
8	128	NC	10000
16	256	NC	10000
32	512	NC	10000
64	1024	NC	10000
128	2048	NC	10000

* Used 16 Bulldozer cores/node
NC: not converged

To find an optimal combination of iterative method and preconditioner, an extensive group of combinations are tested as presented in Table 14. After all, it turns out

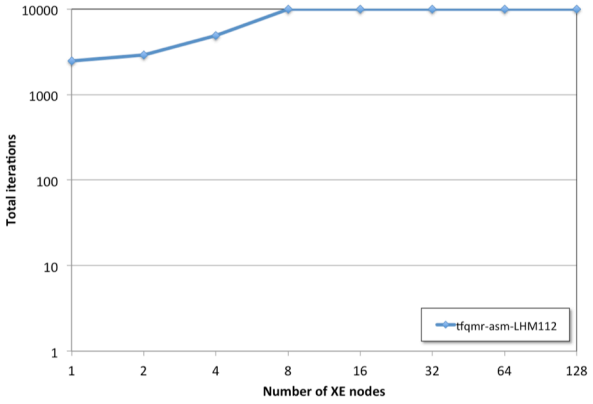


Figure 10. Total iterations of tfqmr (ksp) with asm (pc) for LHM112 on XE nodes

Table 14. Convergence of iterative solvers for LHM112 on 8 XE nodes

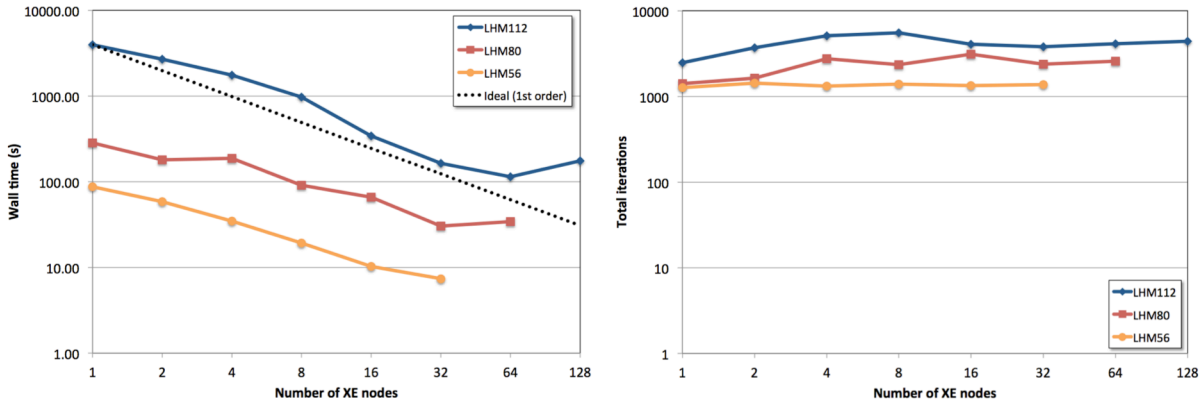
	ksp types										
pc_types	bicg	gmres	fgmres	dgmres	gcr	bcgs	cgs	tfqmr	tcqmr	cr	lsqr
-pc_type jacobi	D	NC	NC	D	NC	D	D	D	NC	NC	NC
-pc_type bjacobi	D	NC	NC	NC	NC	D	D	NC	NC	NC	NC
-pc_type asm	D	NC	NC	NC	NC	D	D	NC	NC	NC	NC
-pc_type gamg -pc_gamg_type agg *	D	NC	NC	NC	NC	D	D	NC	D	D	D
-pc_type gamg -pc_gamg_type classical	D	NC	NC	NC	NC	D	D	NC	NC	D	NC
-pc_type hypre -pc_hypre_set_type ams	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
-pc_type hypre -pc_hypre_set_type ads	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
-pc_type asm -pc_asm_type none	D	NC	NC	NC	NC	D	D	NC	NC	NC	NC
-pc_type asm -pc_asm_type interpolate	D	NC	NC	NC	NC	D	D	NC	NC	NC	NC
-pc_type asm -pc_asm_type basic	D	NC	NC	NC	NC	D	D	C	NC	NC	NC

* with -pc_gamg_agg_nsmooths 0 for non-symmetric matrices
D for 'diverged'/ NC for 'not converged'/ OOM for 'out of memory'/ C for 'converged'

Table 15. Wall time and total iterations of TFQMR (ksp) with ASM_BASIC (pc) for LHM112 on XE nodes

Number of XE nodes	Cores*	LHM112			LHM80			LHM56		
		Wall time	Number of iterations	Speedup	Wall time	Number of iterations	Speedup	Wall time	Number of iterations	Speedup
1	16	3931.61 s	2467	1.0	282.83 s	1406	1.0	87.48 s	1258	1.0
2	32	2690.40 s	3710	1.5	179.68 s	1625	1.6	58.25 s	1431	1.5
4	64	1752.43 s	5107	2.2	185.97 s	2753	1.5	34.60 s	1326	2.5
8	128	966.55 s	5512	4.1	90.13 s	2338	3.1	19.13 s	1397	4.6
16	256	339.41 s	4053	11.6	66.19 s	3120	4.3	10.23 s	1334	8.6
32	512	163.34 s	3772	24.1	30.08 s	2366	9.4	7.37 s	1380	11.9
64	1024	113.73 s	4131	34.6	34.17 s	2563	8.3			
128	2048	174.65 s	4374	22.5						

* Used 16 Bulldozer cores/node



(a) Wall time (b) Number of total iterations

Figure 11. Wall time and total iterations of TFQMR (ksp) with ASM_BASIC (pc) for LHM112, LHM80 and LHM56 on XE nodes

TFQMR for iterative solver gets a stable numerical convergence with ASM_BASIC that uses the full restriction and interpolation operators of the standard additive Schwarz method. This combination (i.e., TFQMR with ASM_BASIC) is tested for LHM56, LHM80 and LHM112 (see Table 15 and Figure 11). It keeps speedup with up to 64 XE nodes (i.e., 1024 MPI ranks) for LHM112, while it speeds up the solution process for LHM80 and LHM56 with up to 32 XE nodes (i.e., 512 MPI ranks). Even though numerical stability of sparse iterative solvers is limited for ill-posed problems, the wall times elapsed by sparse iterative solvers are much shorter than by sparse direct solvers reported in the previous sub-section.

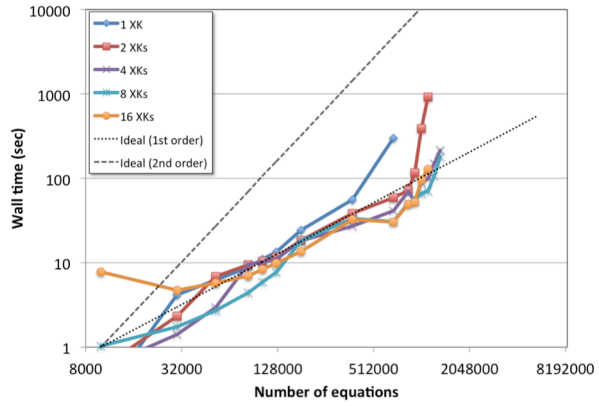
C. *Sparse iterative solvers on XK nodes – NVIDIA AmgX*

Table 16 and Figure 12 show test results of AmgX solver on multiple XK nodes. The configuration in Table 7 is used again with dDDI for the mode parameter. It shows stable results for up to LHM66 with 1, 2, 4, 8 and 16 XK nodes. However, for larger matrices with higher condition numbers than LHM66, AmgX reaches to the maximum allowable iterations. For LHM112, AmgX reports out-of-memory

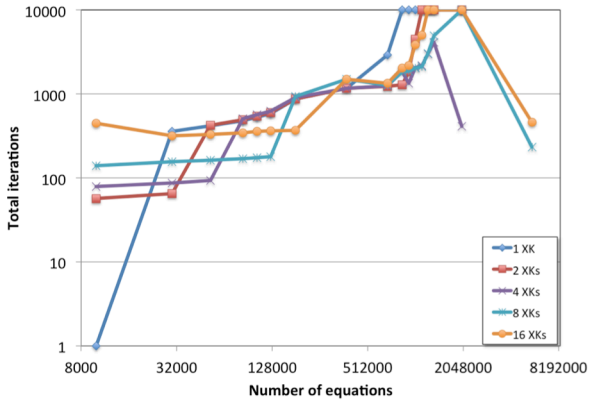
Table 16. Test results of AmgX solver on XK nodes

Matrix	Number of equations	2 XK nodes			4 XE nodes			8 XK nodes			16 XK nodes		
		Wtime	Niter	Reduc	Wtime	Niter	Reduc	Wtime	Niter	Reduc	Wtime	Niter	Reduc
LHM14	9,965	0.56	57	6.89E-13	0.57	79	7.81E-13	1.03	140	9.38E-13	7.83	447	8.66E-13
LHM20	29,837	2.33	65	7.83E-13	1.41	87	9.52E-13	1.75	157	7.98E-13	4.73	318	9.58E-13
LHM24	52,125	6.92	427	9.20E-13	2.94	93	7.70E-13	2.71	163	8.39E-13	5.71	332	8.47E-13
LHM28	83,437	9.49	491	9.74E-13	9.56	506	9.99E-13	4.42	170	7.99E-13	7.02	343	8.51E-13
LHM30	102,957	10.44	545	9.96E-13	10.09	565	9.81E-13	5.90	174	8.71E-13	8.39	357	9.36E-13
LHM32	125,309	11.60	602	9.99E-13	10.97	624	9.79E-13	7.72	179	9.91E-13	9.85	365	9.03E-13
LHM36	179,277	18.42	873	9.33E-13	18.14	914	9.93E-13	18.23	941	9.39E-13	13.55	369	8.83E-13
LHM46	377,197	38.61	1172	9.65E-13	27.36	1172	9.91E-13	33.51	1493	9.93E-13	32.43	1491	9.97E-13
LHM56	684,317	59.14	1239	9.99E-13	41.25	1248	1.00E-12	30.88	1268	9.99E-13	30.49	1351	1.00E-12
LHM60	843,117	71.45	1283	9.80E-13	70.15	1845	9.68E-13	50.46	1841	9.74E-13	49.10	2028	9.97E-13
LHM62	930,989	116.35	1859	9.94E-13	53.31	1337	9.88E-13	55.91	1874	9.79E-13	52.37	2168	9.94E-13
LHM64	1,024,756	390.57	4467	9.75E-13	90.02	2052	9.89E-13	65.84	2033	9.96E-13	95.82	3852	9.99E-13
LHM66	1,124,637	925.48	9973	9.97E-13	99.07	2170	9.96E-13	70.40	2129	9.82E-13	128.03	5000	9.92E-13
LHM68	1,230,797	NC	10000	5.38E-11	150.12	3012	9.98E-13	106.31	3033	9.75E-13	NC	10000	1.82E-12
LHM70	1,343,437	NC	10000	1.12E-07	215.35	4156	1.00E-12	179.33	4946	9.90E-13	NC	10000	7.68E-09
LHM80	2,010,557	NC	10000	8.93E-08	OOM	410	3.60E-06	NC	10000	5.39E-10	NC	10000	6.35E-09
LHM112	5,545,789	OOM			OOM			OOM	234	2.74E-03	OOM	460	1.48E-05

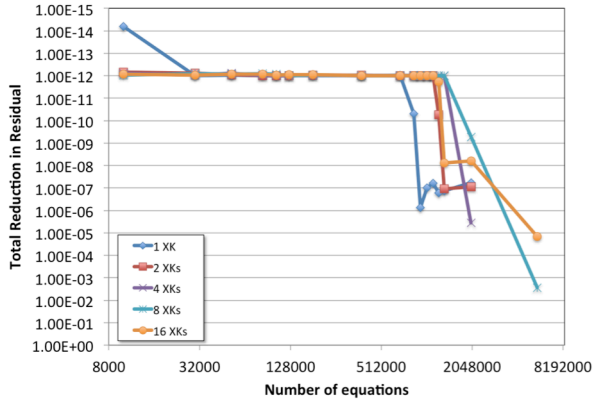
Wtime: elapsed wall time by AmgX; Niter: number of iterations; Reduc: total reduction in residual,
NC for 'not converged'/ OOM for 'out of memory'



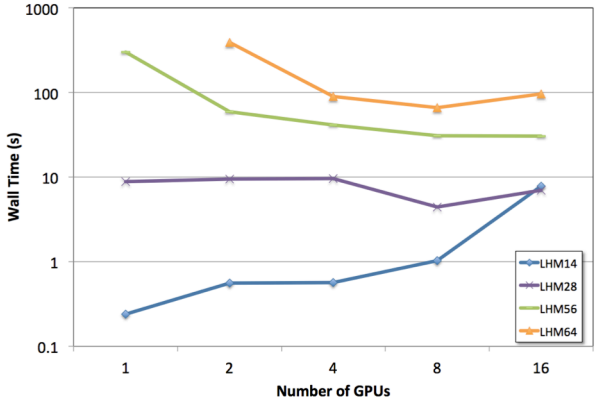
(a) Wall time



(b) Number of total iterations



(c) Total reduction in residual



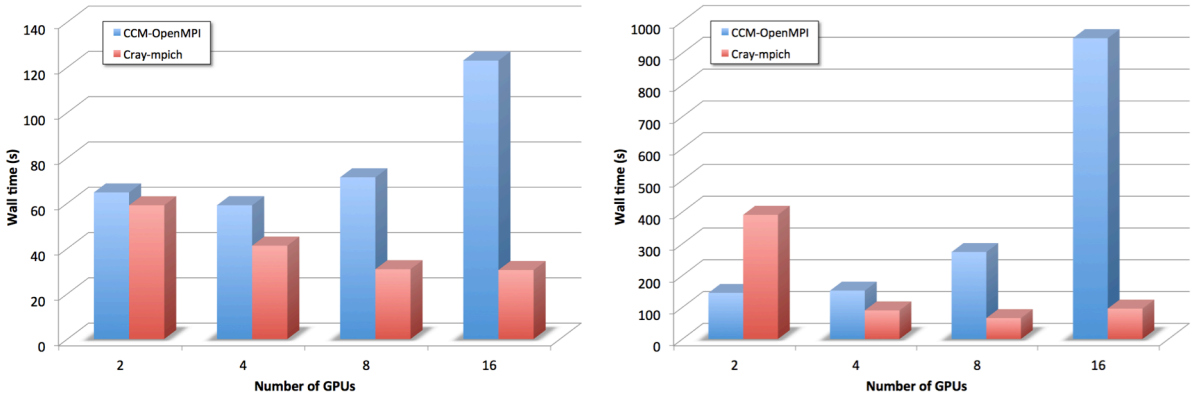
(d) Wall time vs. number of GPUs

Figure 12. Test results of AmgX solver on XK nodes

Table 17. Comparison between AmgX with Cray-mpich and AmgX with OpenMPI under CCM

			2 XK nodes		4 XK nodes		8 XK nodes		16 XK nodes	
Matrix	Number of equations	MPI types	Wtime	Niter	Wtime	Niter	Wtime	Niter	Wtime	Niter
LHM56	684,317	CCM	64.70 s	1239	59.05 s	1248	71.38 s	1258	122.96 s	1351
		Cray-mpich	59.05 s	1239	41.21 s	1248	30.80 s	1268	30.45 s	1351
Wall time ratio, CCM/Cray (%)			110 %		143 %		232 %		404 %	
LHM64	1,024,765	CCM	145.09 s	1959	152.29 s	2052	273.69 s	2033	946.73 s	3852
		Cray-mpich	390.91 s	4467	89.96 s	2052	65.80 s	2033	95.65 s	3852
Wall time ratio, CCM/Cray (%)			37 %*		169 %		416 %		990 %	

* Cray-compatible AmgX got the following warning message, and it restarted from the initial residual at the 500th iteration:
WARNING: Cannot allocate next Krylov vector, out of memory. Falling back to DQGMRES



(a) LHM56 (b) LHM64

Figure 13. Comparison between AmgX with Cray-mpich and AmgX with OpenMPI under CCM

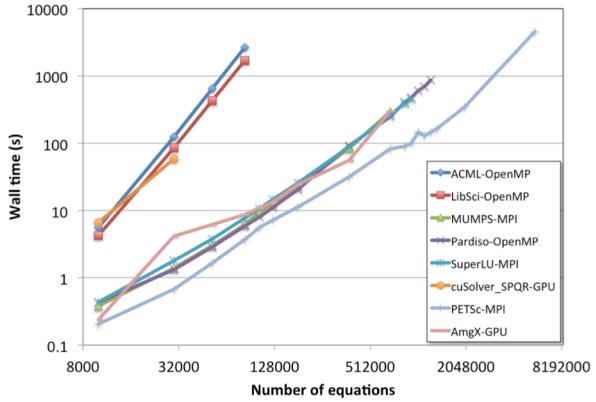
error; it turns out 6GB memory of GPUs is not big enough for the iterative methods to solve the system of linear equations with 5.5 million variables.

AmgX is free for non-commercial use and is available for download from NVIDIA developer's webpage. Since the freely distributed version is compatible only with OpenMPI, it should be executed under Cluster Compatibility Mode (CCM). The OpenFabric port of OpenMPI [10] that runs in Cray native mode was recently installed on Blue Waters but is not used in this work. The OpenMPI compatible AmgX is tested compared to Cray-mpich compatible AmgX that is provided by NVIDIA for this study. OpenMPI version 1.8.4 is installed under PrgEnv-gnu/5.2.82 and CCM version 2.2.1-1.0502.62540.4.51 is used for the test. Table 17 and Figure 13 show wall times of each AmgX for LHM56 and LHM64. As the number of XK nodes increases, the performance degradation of AmgX under CCM becomes more significant. With 2 XK nodes, AmgX under CCM is around 10% slower than Cray-compatible AmgX. With 16 XK nodes, AmgX under CCM is 4 to 10 times slower than Cray-compatible AmgX.

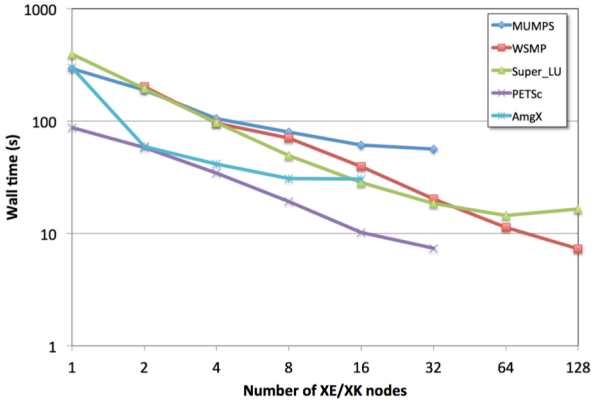
V. Concluding Remarks

We present performance test results of a variety of linear equation solvers available on Blue Waters (i.e., a Cray XE/XK system). A series of non-symmetric matrices from a CFD problem is employed for the test problem. Most solver libraries are very well optimized on the Cray system; consequently, they show the optimal time-complexity depending on the algorithm (i.e., 1st order for iterative methods, 2nd order for sparse direct methods, and 3rd order for dense direct methods).

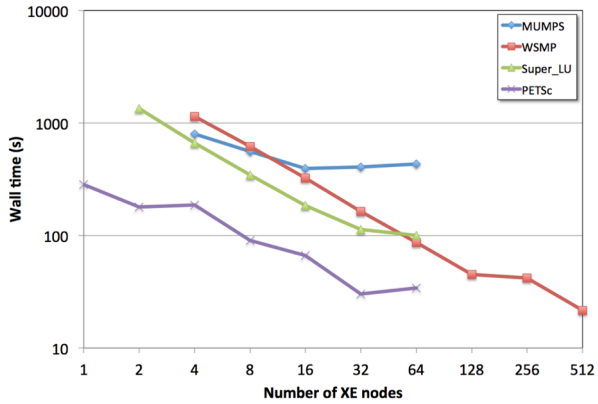
As a summary of our tests, we present Figure 14 to provide a tentative performance chart of parallel solvers for one iteration-step in general CFD simulations. For this reason, the cost of pre-factorization of sparse direct solvers is excluded, since it is almost negligible during a lot of nonlinear iteration-steps and time-steps. Figure 51 (a) shows wall times of parallel solvers on one XE or XK node. As expected, dense direct solvers provide the most convenient interface to users, but they offer the most expensive way to solve linear systems. Sparse direct solvers show better performance in memory usage as well as computational time, and they provide wider envelopes in the size of solvable matrices than dense direct solvers. Sparse iterative solvers are the most cost-effective among employed solvers, but it may be difficult for users to find an optimal combination of an iterative method and a preconditioner to get the convergence for ill-posed problems. Figures 51(b)-(d) show wall time of sparse direct and iterative solvers with multiple XE/XK nodes. For LHM56 with 684K equations, sparse iterative solvers (i.e., PETSc on CPUs and AmgX on GPUs) are more economical than sparse direct solvers such as MUMPS, WSMP and SuperLU_DIST. For LHM80 with 2M equations and LHM112 with 5.5M equations with higher condition numbers than LHM56, PETSc provides the most cost-effective approach compared to sparse direct solvers in this study while the iterative solver, AmgX fails to obtain a converged solution. In summary, sparse direct solvers show outstanding numerical stability for ill-posed problems with very high condition numbers and in some cases, such as WSMP, an excellent scalability. Sparse iterative solvers, however, provide the most cost-effective approaches in solving massive systems of linear equations if they can cope



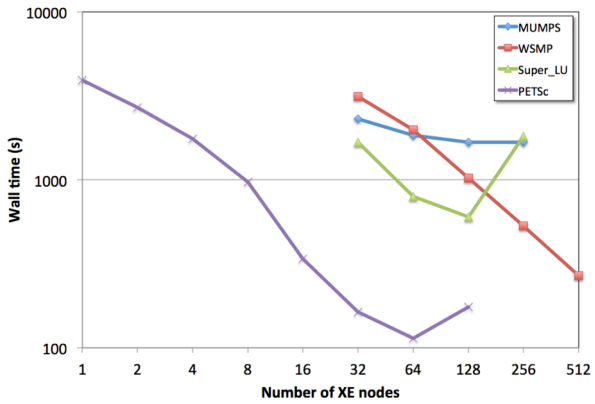
(a) LHM14 to LHM112 on a single XK/XE node



(b) LHM56 (684K equations) on multiple XK/XE nodes



(c) LHM80 (2M equations) on multiple XE nodes



(d) LHM112 (5.5M equations) on multiple XE nodes

Figure 14. Wall times of parallel solvers on Blue Waters (excluding costs of pre-factorization for direct solvers)

with the numerical instabilities in the solution process of ill-posed problems.

ACKNOWLEDGMENT

This study is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

REFERENCES

[1] B. Bode, M. Butler, T. Dunning, W. Gropp, T. Hoe-fler, W. Hwu, and W. Kramer (alphabetical). The Blue Waters Super-System for Super-Science. *Contemporary HPC Architectures*, Jeffery Vetter editor. Sitka Publications, November 2012.Edited by Jeffrey S . Vetter, Chapman and Hall/CRC 2013, Print ISBN: 978-1-4665-6834-1, eBook ISBN: 978-1-4665-6835-8.

[2] W. Kramer, M. Butler, G. Bauer, K. Chadalavada, C. Mendes. Blue Waters Parallel I/O Storage Sub-system, *High Performance Parallel I/O*, Prabhat and Quincey Koziol editors, CRC Publications, Taylor and Francis Group, Boca Raton FL, 2015, Hardback Print ISBN 13:978-1-4665-8234-7.

[3] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc Web page. *https://www.mcs.anl.gov/petsc*, 2016.

[4] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136-156, 2006.

[5] X. S. Li and J. W. Demmel. A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems. *ACM Trans. Mathematical Software*, 29(2):110-140, 2003.

[6] A. Gupta. WSMP: Watson Sparse matrix package (Part-II: direct solution of general systems). *Technical Report RC 21888* (*98472*). Yorkton Heights, NY: IBM T.J. Watson Research Center; 2016.

[7] NVIDIA AmgX webpage. *https://developer.nvidia.com/amgx*, 2016.

[8] J. Kwack and A. Masud. A stabilized mixed finite element method for shear-rate dependent non-Newtonian fluids: 3D benchmark problems and application to blood flow in bifurcating arteries. *Computational Mechanics*, 53:751-776, 2014.

[9] S. Koric, Q. Lu and E. Guleryuz. Evaluation of massively parallel linear sparse solvers on unstructured finite element meshes. *Computers and Structures*, 141:19-25, 2014.

[10] M. G. Venkata, R.L. Grahma, N.T. Hjelm, S.K. Gutierrez. Open MPI for Cray XE/XK systems. *CUG-2012*. https://www.open-mpi.org/papers/cug-2012/cug_2012_open_mpi_for_cray_xe_xk.pdf.