RC 24737, January 30, 2009 (Last revised on December 23, 2009)
Computer Science/Mathematics

IBM Research Report

An Empirical Analysis of Iterative Solver Performance for SPD Systems

Thomas George

IBM India Research Laboratory
thomasgeorge@in.ibm.com

Anshul Gupta

Business Analytics and Mathematical Sciences
IBM T. J. Watson Research Center
anshul@watson.ibm.com

Vivek Sarin

Department of Computer Science and Engineering
Texas A&M University
sarin@cse.tamu.edu

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

Research Division
Almaden · Austin · China · Haifa · India · Tokyo · Watson · Zurich

An Empirical Analysis of Iterative Solver Performance for SPD Systems

Thomas George
IBM India Research Laboratory,
4, Block C, Vasant Kunj, New Delhi 110 070, India.
thomasgeorge@in.ibm.com

Anshul Gupta
Business Analytics and Mathematical Sciences,
IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA.
anshul@watson.ibm.com

Vivek Sarin
Department of Computer Science and Engineering,
Texas A&M University, College Station, TX 77843, USA.
sarin@cse.tamu.edu

January 30, 2009 (Last revised on December 23, 2009)

Abstract

Preconditioned iterative solvers have the potential to solve very large sparse linear systems with a fraction of the memory used by direct methods. However, the effectiveness and performance of most preconditioners is not only problem dependent, but also fairly sensitive to the choice of their tunable parameters. As a result, a typical practitioner is faced with an overwhelming number of choices of solvers, preconditioners and their parameters. The diversity of preconditioners makes it difficult to analyze them in a unified theoretical model. Hence, a systematic evaluation of existing preconditioned iterative solvers is necessary to identify the relative advantages of various implementations. We present the results of a comprehensive experimental study of the most popular preconditioner and iterative solver combinations for symmetric positive-definite systems. We introduce a methodology for a rigorous comparative evaluation of various preconditioners, including the use of some relatively simple but powerful metrics. We describe a semi-automated system for the collection, analysis, and visualization of relative performance data. The detailed comparison of various preconditioners preconditioner implementations and a state-of-the-art direct solver gives interesting insights into their relative strengths and weaknesses. We believe that these results would be useful to researchers developing preconditioners and iterative solvers as well as practitioners looking for appropriate sparse solvers for their applications.

1 Introduction

A fundamental step in many scientific and engineering simulations is the solution of a system of linear equations of the form $Ax = b$, where $A \in C^{m \times n}$, typically known as the coefficient matrix, is sparse, $b \in C^m$ is the right hand side vector (RHS), and $x \in C^n$ is the vector of unknowns. Large sparse linear systems involving millions and even billions of equations are becoming increasingly

common in scientific, engineering, and optimization applications. These systems of equations can be solved using either *direct* or *iterative* methods. Direct methods are fast and robust, but are often unsuitable for large three-dimensional problems due to their prohibitive memory requirements. This limitation, combined with the need to solve ever increasing sizes of the linear systems has continued to fuel strong research efforts in iterative solvers and their preconditioning methods.

The effectiveness and performance of most preconditioners are not only problem dependent, but also fairly sensitive to the choice of their tunable parameters. As a result, a typical practitioner has an overwhelming number of choices of solvers, preconditioners and their parameters in several black-box solver packages, such as PETSc [1], Trilinos [20], ILUPACK [4], Hypre [10], and many others [9]. Choosing the appropriate scheme and fine-tuning the parameters for a particular linear system to be solved remains a blend of art and science due to several reasons. The diversity of the preconditioners makes it difficult to analyze them in a unified theoretical model. There is often a significant variability in the different implementations for the same preconditioner, which limits the utility of a purely theoretical analysis. Finally, most preconditioners have a multitude of tunable parameters, and the best set of parameters for the same preconditioner can vary significantly not only between problems, but also from one implementation to another. Survey articles by Benzi [2] and Saad and van der Vorst [25] describe the relative strengths and weaknesses of a wide range of preconditioned iterative schemes from a theoretical perspective. There have been a few empirical studies in the past [3, **?**, 14], typically focused on a single preconditioner. Despite the widespread use of preconditioned iterative solvers, there has not been an extensive empirical evaluation of the readily available implementations of most common classes of preconditioners. This paper is an attempt to fill a part of this void for systems with symmetric positive definite coefficient matrices. To the best of our knowledge, it provides the most extensive practitioner-centric empirical evaluation to date of a number of popular and promising general purpose preconditioners available in black-box solver packages.

We expect our study to provide insights into the relative effectiveness of these preconditioners for problems from a variety of domains. Prospective users could benefit from our experimentation with the important parameters associated with the preconditioners. In addition to the users, the study could also provide guidance to the current and future developers of preconditioner and solver packages on ways and areas of potential improvement to their software. The main contributions of this work are as follows:

Benchmarking methodology: We introduce a methodology for a rigorous comparative evaluation of various preconditioners, including the use of some relatively simple but powerful metrics to facilitate a credible ranking of solver configurations (combinations of solver packages, preconditioners, and preconditioner parameters). Notable among these metrics are memory-time product and area under the curve for performance profiles (Section 3). We also define formal quantitative metrics for comparing different preconditioners, each with several possible parameter configurations.

Performance analysis infrastructure: We developed a semi-automated system for the collection, analysis, and visualization of relative performance data. It generates scripts to run experiments for all combinations generated from a user specified set of linear systems, a set of hardware configurations (number of CPUs and memory limits, etc.), and sets of values of various solver and preconditioner parameters. It also collects performance data (time, memory, error norm, etc.) via a data collection unit composed of both serial and parallel driver programs and associated scripts for some widely used solver packages. Subsequently, the analysis and reporting unit of the system performs various comparative and sensitivity

analyses within and across pre-specified groups of solver configurations using the collected performance data.

Extensive empirical evaluation: Using the above system, we evaluate a suite of preconditioners based on the incomplete factorization, sparse approximate inverse, and the algebraic multigrid schemes available in packages such as PETSc, Trilinos, Hypre, ILUPACK, and WSMP. We compare the robustness, speed, and memory consumption of these preconditioners on a set of benchmark problems and present results that can serve as guidance to practitioners. For packages that provide support for parallel execution, we collect and present performance data on up to 64 processors.

Good default configurations: For each combination of solver package and preconditioner, we identify the best overall choice of solver and preconditioner parameters on a suite of diverse problems. These observations can be used for choosing good *default* configurations for each package-preconditioner combination.

Choice of package-preconditioner combination: We simultaneously project the performance of all package-preconditioner combinations included in this study along three carefully chosen dimensions involving time, memory, and robustness to allow a ready comparison of the relative strengths of various implementations. We perform this comparison for the overall best set of parameters as well as for problem specific best set of parameters for each preconditioner implementation because their relative rankings can be different under the two scenarios.

Fine tuning of parameters: In addition to determining good default configurations for each preconditioner implementation, we also study how sensitive the performance of a certain preconditioner is to parameter choices and which parameters have the greatest impact on performance. This analysis can guide the fine tuning the parameters to a specific problem or class of problems.

The remainder of the paper is organized as follows: Section 2 provides details of our experimental set up, including the test suite, the solver packages and the preconditioners included in the study, and the hardware used. Section 3 gives an overview of the various metrics that are used to rank the performance of individual solver configurations as well as that of package-preconditioner combinations. In section 4, we present the detailed empirical evaluation methodology and results. In Section 5, we analyze the relative importance of the various parameters on the performance of the preconditioners evaluated in this report. We describe our performance analysis framework and a prototype implementation in Section 6. Section 7 contains concluding remarks and possible directions for future work.

2 Experimental Setup

In this section, we present the details of our experimental setup. These include an introduction to the solver packages, preconditioners and their parameters, descriptions of the test matrices and the hardware platform, and the specifics of our experimental approach.

2.1 Software Packages

We included the following packages in our study, which we believe are likely to be of most value to researchers and practitioners. These include well-established packages that include most commonly

used preconditioners, as well as research packages with recently published promising general purpose preconditioners.

PETSc - Release Version 2.3.3-p0: PETSc [1] is a parallel scientific computing package developed at Argonne National Laboratory. The main goal of the PETSc project is to equip a user with the tools necessary for building scalable scientific applications. PETSc provides efficient implementations for a number of commonly used Krylov subspace methods as well as fixed pattern and threshold based incomplete factorization preconditioners. It is possible to access a number of other preconditioning schemes via interfaces to external packages. BlockSolve95 [**?**] is one of such packages that we used in our experiments.

Trilinos - Release Version 8.0.3: Trilinos [20] was developed at Sandia National laboratories and its main focus is to provide parallel solvers and libraries in an object oriented framework. Trilinos is composed of a number of self contained independently developed packages that could be used as a stand-alone application or in conjunction with other packages that support a minimal set of prerequisites for the interfaces. A suite of object oriented preconditioners are available in the Ifpack [26] and ML [12] packages. The AztecOO [19] package, provides an object oriented interface to the popular Aztec solver library which contains implementations of the Krylov subspace methods. Ifpack supports a suite of Jacobi-style and incomplete factorization-based preconditioners whereas the ML package provides variants of algebraic multigrid type of preconditioners based on smoothed aggregation.

Hypre - Release Version 2.0.0: Hypre [10], developed at Lawrence Livermore National Laboratory, is designed primarily for the solution of large, sparse linear systems of equations on massively parallel computers. Hypre provides four different logical interfaces, namely, structured, semi-structured, finite element and linear algebraic. It includes distributed-memory parallel implementations of three classes of preconditioners, namely incomplete factorization (Euclid [21]), sparse approximate inverse (ParaSails [5, 6]), and algebraic multigrid (BoomerAMG [18]).

ILUPACK - Development Version 2.2: ILUPACK [4] was developed at Technische Universit*ä*t Berlin and it contains implementations of an inverse-based multilevel ILU preconditioners that controls the growth of the norm of inverse triangular factors for both real and complex matrices. In addition to the standard static reordering schemes, it also includes the ARMS ordering schemes such as INDSET and ddPQ [24].

WSMP - Development Version 8.7: The Watson Sparse Matrix Package (WSMP) [15, 16], developed at IBM, contains serial and parallel sparse direct solvers as well as CG and GMRES solvers with new incomplete factorization based preconditioners [17].

2.2 Matrix Partitioning and Reordering

Previous research has shown that a suitable reordering of the coefficient matrix can reduce the memory requirement and potentially have a significant impact on the performance of many preconditioners [23]. Hence, wherever applicable, we tried the Reverse Cuthill Mckee ordering (RCM) [**?**] and the Nested Dissection ordering (ND) [**?**, **?**] from the WSMP package, in addition to the matrices' natural ordering. For the ILUPACK package, we used five built-in reordering schemes, namely, Nested dissection (ND), RCM, Approximate Minimum Fill (AMF), Independent set (INDSET), and permutation for diagonal dominance (ddPQ) [24].

When using more than one processor, ParMETIS [**?**] is used to partition the rows of the matrix and distribute them appropriately. We then reorder the local matrix on each processor according to the specified reordering scheme.

Package	Solver	Preconditioner
PETSc	CG	IC(k)
PETSc	CG	BlockSolve95
Trilinos	CG	IC(k)
Trilinos	CG	Algebraic Multigrid (AMG)
ILUPACK	CG	Multilevel-ICT
Hypre	CG	IC(k)
Hypre	CG	Algebraic Multigrid (AMG)
Hypre	CG	Sparse Approximate Inverse (SAI)
WSMP	CG/GMRES	ICT

Table 1: Preconditioners and solvers used in each package for SPD matrices.

2.3 Preconditioners and Parameters

We now describe the preconditioners that we use in our study. Table 1 contains a list of these preconditioners and the corresponding solvers. We study primarily three broad classes of preconditioners, namely incomplete factorization, sparse approximate inverse, and algebraic multigrid.

2.3.1 Level-of-fill based incomplete Cholesky factorization, IC(k):

An important class of preconditioners for SPD systems is based on the incomplete Cholesky factorization of the coefficient matrix. There are several variations of incomplete factorization that differ in the rules for dropping entries while computing the incomplete factors. One strictly positional criterion for dropping is based on what is known as the *level of fill*, which is a measure of "distance" of a fill entry from the original entries in the coefficient matrix (please refer to the book by Saad [23] or the survey by Benzi [2] for a formal definition). In IC(k) factorization, all fill-in entries at levels exceeding k are dropped. An important advantage of a pure IC(k) preconditioner is that the sparsity pattern can be determined a-priori by a symbolic factorization step whose cost is amortized when solving multiple linear systems with the same sparsity pattern. Often, depending on the implementation, when the parameter $k > 0$, it is supplemented with additional parameters to handle fill levels higher than 0. Thus, many implementations of the IC(k) preconditioner use a combination of dropping based on both the position and magnitude of the nonzeros. The following parameters are used in the implementations of IC(k) that we study in this paper.

1. *Highest fill level,* k, is the fundamental parameter in all implementations of the IC(k) preconditioner and denotes the level beyond which all fill-ins are dropped. We experimented with values of 0, 1, and 2 for k for PETSc and 0, 1, 2, 4, 6, and 8 for Hypre and Trilinos.

2. *Fill factor* specifies an upper bound on the amount of memory used by the preconditioner for levels of fill greater than zero. A fill factor f denotes that the preconditioner would not use more that f times the number of nonzeros in the original matrix. This is a user controlled parameter in the IC(k) implementation of PETSc only.

BlockSolve95 [**?**] implements a hybrid strategy that incorporates ideas from both level-based and threshold based incomplete factorization preconditioners. The sparsity of the factors are guaranteed by retaining only the largest elements such that the memory usage is no larger than that required by a ILU(0) preconditioner. BlockSolve95 has the added advantage that no parameters need to be specified.

Careful partitioning and ordering of sub-domains has been shown to be effective in obtaining scalable parallel implementations of the IC(k) preconditioner [21]. One approach for parallelization of IC(k) is to use the sequential IC(k) algorithm within each sub-domain, popularly known as the Block Jacobi based IC(k). A preliminary comparison of the Block Jacobi and parallel versions of IC(k) in Hypre's Euclid indicated that the Block Jacobi version was more scalable. All parallel implementation of the IC(k) preconditioner studied in this paper are those of the Block Jacobi variant.

2.3.2 Threshold based incomplete Cholesky (ICT):

The threshold based incomplete Cholesky or ICT preconditioners control fill-in by means of a dual dropping strategy based on a numerical threshold τ and an upper limit f on the number of fill-ins in each row or column. Typically, any new fill-in whose magnitude is below τ times a chosen metric is dropped. In addition, if the number of nonzeros in a row or column of the factor exceeds f times the number of nonzeros in that row or column in the original coefficient matrix, then the excess entries with the smallest magnitude are also dropped. Incomplete Cholesky factorization may fail due to negative diagonal entries and diagonal perturbation is often used to boost the diagonal dominance of the original coefficient matrix prior to factoring. Typically low values are preferred for diagonal perturbation since high values could introduce additional error in the solution.

Three packages listed in our study have implementations of the ICT preconditioner. Of these, WSMP uses a strategy of self selection and tuning of parameters [17]. However, in order to be consistent in our evaluation, we override automatic selection and experiment with multiple values of drop tolerance, fill factor, and diagonal perturbation. The second implementation of ICT, in ILUPACK, is a variation that uses a multi-level incomplete factorization strategy combined with static pre-ordering and a dropping criterion that attempts to minimize the norms of the inverses of the triangular factors. We found that the preconditioner generation phase of the ICT preconditioner in Trilinos consumes an excessive amount of time for most of the problems in our test suite. Therefore, we report the results of the performance of the ICT implementations in WSMP and ILUPACK only. The main user controlled parameters in WSMP and ILUPACK's implementations of the ICT preconditioner are described below.

1. *Drop tolerance*, used in both WSMP and ILUPACK, specifies the numerical threshold used to drop the fill-in values. Lower values for drop tolerance lead to more accurate preconditioners but result in higher memory consumptions, and vice versa.

2. *Fill factor*, used the ICT implementation of WSMP only, specifies an upper bound on the amount of memory used by the preconditioner for levels of fill greater than zero. A fill factor f denotes that the preconditioner would strive to use no more than f times the number of nonzeros in the original matrix.

3. *Norm of inverse estimate*, used in ILUPACK only, provides an upper bound on the norm of the inverse of the triangular factors.

2.3.3 Algebraic multigrid (AMG):

Algebraic multigrid or multilevel methods are currently enjoying a lot of popularity as black-box solvers. The basic idea of an algebraic multilevel solver is to construct a hierarchy of coarser graphs, where each node in a coarse level represents multiple nodes of the previous finer level. At each coarse level, an iterative solver or a smoother is used to compute an approximate solution to system

corresponding to that level and then project this solution to the next finer level. The entire scheme can be viewed as a preconditioner for an iterative solver at the finest level.

We study two implementations of AMG in this paper, namely BoomerAMG [18], which is a parallel implementation of the classical AMG [22] available in Hypre, and Trilinos ML [12, 29], which includes a parallel implementation of the smoothed aggregation approach [**?**] for AMG. Trilinos provides default sets of parameters for three main preconditioner types for problems arising from specific domains; classical smoothed aggregation for SPD or nearly SPD systems (SA), classical smoothed aggregation based 2-level domain decomposition (DD), and 3-level algebraic domain decomposition (DD-ML).

Implementations of AMG typically have a large number of user tunable parameters. Based on the advice from the authors of BoomerAMG and Trilinos ML, we chose to vary the following parameters, while using the default values for others.

1. *Smoothers* - For Hypre BoomerAMG, hybrid symmetric Gauss-Seidel/Jacobi [10] was the smoother of choice since CG was the default solver used for SPD matrices. For Trilinos ML, we experimented with symmetric Gauss-Seidel, Chebyshev polynomial, and the default IFPACK smoothers.

2. *Coarsening schemes:* We experimented with three different coarsening schemes in BoomerAMG, namely, Falgout (FALG), Parallel Modified Independent Set (PMIS) and Hybrid Modified Independent Set (HMIS). For Trilinos ML, we tried Uncoupled, MIS, hybrid Uncoupled-MIS, and ParMETIS coarsening schemes for classical smooth aggregation (SA).

3. *Number of Smoother Sweeps:* This parameter gives users of Trilinos ML another means of controlling the trade-off between the cost per iteration and the number of iterations required. We tried values of 2 and 3 for symmetric Gauss-Seidel and Chebyshev polynomial smoothing.

4. *Number of levels for aggressive coarsening (AGG):* The value given to this parameter in BoomerAMG sets the number of levels for which aggressive coarsening must be applied starting from the finest level. We experimented with values of 10 and 0 for all the three coarsening schemes.

5. *Interpolation type:* For the Falgout coarsening scheme, we used the recommended *multi-pass interpolation* for use with aggressive coarsening and the *classical interpolation* when not using aggressive coarsening. For the PMIS and HMIS coarsening schemes, we used the *extended classical interpolation* scheme as recommended in the user guide.

6. *Strong threshold:* The value specified for this BoomerAMG parameter serves as a threshold to determine whether two points in a graph are strongly or weakly connected. High values of strong threshold lead to cheaper but less effective preconditioners whereas low values result in expensive preconditioners with better convergence properties. We experimented with values of 0.25, 0.5, 0.7, and 0.9 for this parameter.

2.3.4 Sparse approximate inverse (SAI):

The problems inherent in using incomplete factorization based variants are partially addressed by preconditioners based on sparse approximate inverses [3]. Depending on the algorithms used for finding the sparse inverse, approximate inverse based preconditioners could be fairly robust in practice and easily parallelizable. However, these preconditioners usually incur a high initial setup cost and the efficacy and the cost of applying the preconditioner depends on the choice of the sparsity

Package	Solver	Preconditioner	Orderings	Parameters
PETSc	CG	BlockSolve95	RCM, ND	-
		IC(k)	RCM, ND	*Level of fill:* 0, 1, 2 *Fill factor:* 3, 5, 8, 10
Hypre	CG	IC(k)	RCM, ND	*Level of fill:* 0, 1, 2, 4, 6, 8
		ParaSails	RCM, ND, NONE	*Number of levels:* 0, 1, 2 *Threshold:* 0, 0.01, 0.1, -0.75, -0.9 *Filter:* 0, 0.001, 0.05, -0.9
		BoomerAMG	RCM, ND, NONE	*Maximum number of levels:* 25 *Number of aggressive coarsening levels:* 0, 10 *Coarsening schemes:* Falgout, HMIS, PMIS *Strong threshold:* 0.25, 0.5, 0.8, 0.9
Trilinos	CG	IC(k)	RCM, ND	*Level of fill:* 0, 1, 2, 4, 6, 8
		ML (SA)	RCM, ND, NONE	*Smoothers:* Symmetric Gauss-Seidel Chebyshev Polynomial, Incomplete Factorization *Smoother sweeps:* 1, 2, 3 *Coarsening Schemes:* Uncoupled, MIS Hybrid Uncoupled-MIS, ParMETIS
		ML (DD), ML (DD-ML)	RCM, ND, NONE	- -
ILUPACK	CG	Multilevel ICT	RCM, ND, AMF, INDSET, DDPQ	*Drop Tolerance:* 0.03, 0.01, 0.003, 0.001, 0.0005 *Inverse Norm Estimate:* 10, 25, 50, 75, 100
WSMP	Auto-select CG/GMRES	ICT	RCM, ND	*Drop Tolerance:* 0.01, 0.003, 0.001,0.0003 *Diagonal Perturbation:* OFF, ON (0.001) *Fill factor:* 2.5, 3.3, 4.1, 4.9

Table 2: Parameter values used in the experiments for various preconditioner implementations.

pattern. Hypre ParaSails preconditioner uses *a-priori* knowledge of sparsity patterns and Frobenius norm minimization to generate an approximate inverse. For SPD matrices, a symmetric factored approximate preconditioner is generated. ParaSails uses three main parameters for controlling the accuracy and the cost of the preconditioner, and these are described below.

1. *Threshold* controls the sparsification of the coefficient matrix such that it can be used to generate the *a-priori* sparsity pattern by dropping elements that are below the specified value. The range of values for *threshold* is between 0.0 and 1.0. One can also specify a negative value for *threshold* such that its absolute value dictates the percentage of nonzeros that must be dropped. The exact value for *threshold* is determined automatically in this case.

2. *Number of levels* controls the memory usage of the resulting preconditioner. ParaSails uses *a-priori* sparsity patterns that are powers of sparsified matrices. For example, if a value of 2 is used for the number of levels, then the sparsity pattern corresponds to the power of 3 of the sparsified matrix. Typical values are 0, 1 and 2 with the default value being 1.

3. *Filter* is a numerical threshold used to reduce the cost of applying the preconditioner by further dropping elements from the computed approximate inverse. This parameter works similar to *threshold* and one could also specify a negative value if it has to be determined automatically based on a percentage of sparsity that is desired. For examples, if $filter = -0.9$, then the threshold is calculated such that 90% of the non zeros in the computed preconditioner are dropped.

Table 2 lists the specific preconditioners and the values of their tunable parameters that were experimented with. In all, 470 different combinations of solvers, preconditioners, and parameters

Matrix	N	NNZ	Application
90153	90153	5629095	Sheet metal forming
af_shell7	504855	17588875	Sheet metal forming
algor-big	1073724	84317460	Static stress analysis
audikw_1	943695	77651847	Automotive crankshaft modeling
bmwcra_1	148770	10644002	Automotive crankshaft modeling
ctu-1	1017397	74144859	Structural analysis
ctu-2	384012	28069776	Structural analysis
cfd1	70656	1828364	C.F.D. pressure matrix
cfd2	123440	3087898	C.F.D. pressure matrix
conti20	20341	1854361	Structural analysis
garybig	42459173	238142243	Circuit simulation
G3_circuit	1585478	7660826	Circuit simulation
hood	220542	10768436	Automotive
inline_1	503712	36816342	Structural engineering
kyushu	990692	26268136	Structural engineering
ldoor	952203	46522475	Structural analysis
msdoor	415863	20240935	Structural analysis
mstamp-2c	902289	70925391	Metal stamping
nastran-b	1508088	111614436	Structural analysis
nd24k	72000	28715634	3D mesh problems (ND problem set)
oilpan	73752	3597188	Structural analysis
pbolic_fem	525825	3674625	C.F.D. convection-diffusion
pga-rem-1	5978665	29640547	Power network analysis
pga-rem-2	1480825	7223497	Power network analysis
qa8fk	66127	1660579	F.E.M. stiffness matrix for 3D acoustic problem
qa8fm	66127	1660579	F.E.M. mass matrix for 3D acoustic problem
ship_003	121728	8086034	Structural analysis - ship structure
shipsec5	179860	10113096	Structural analysis - ship section
thermal2	1228045	8580313	Steady state thermal problem
torso	201142	3161120	Human torso modeling

Table 3: SPD test matrices with their order (N), number of non-zeros (NNZ), and the application area of origin.

were tried for the single processor case. The total number of solver configurations including all the serial and parallel cases added up to 2156.

2.4 Test Matrices

This study focuses primarily on general performance trends of different preconditioners. To this effect, we chose the matrices from a range of applications including structural analysis, fluid dynamics, metal forming, circuit simulation, etc. We do not attempt to compare preconditioers in specific application areas, although the results in Tables 12–14 and in the Appendix can give readers an idea of their performance for individual matrices in our test suite. The details of these SPD matrices are shown in Table 3. Most of the matrices are obtained from the University of Florida collection [7] and the remaining ones are obtained from some of the applications that use the WSMP [15] sparse solver package.

2.5 Hardware Specifics

The experiments were conducted on up to 64 processors on an IBM HPC cluster 1600, based on the Power5+ IBM processor running the 64-bit version of AIX (version 5.3). Each of the p5-575 nodes on the cluster has sixteen 1.9 Ghz Power5+ processors. A memory limit of 24 GB per node and a wall time limit of 4 hours was used for each empirical trial involving a single matrix and a solver configuration.

All the packages were compiled using IBM compilers xlf (Fortran), xlc (C) or xlC (C++) in 64-bit mode with the -O3 optimization flag. The Engineering Scientific Subroutine library (ESSL) was linked in to provide BLAS routines. The page size for text and data was set to 64 KB.

2.6 Experimentation Methodology

We now describe our methodology for conducting the experiments and collecting the performance data. In order to make the evaluation as uniform as possible, we adhered to the following rules for all the experiments.

- Diagonal scaling is performed on the linear system before starting the solution process.
- A right hand side vector of all 1's is used and the initial guess of the solution for the iterative process is always the zero vector.
- We use right preconditioning since it is the default for all the packages except PETSc and it allows us to have a uniform convergence criteria based on the true residual for all the experiments. This choice of right preconditioning was also influenced by a similar study conducted on ILU preconditioners [**?**].
- The iterations are stopped when the number of iterations reaches 1000 or when the relative residual norm drops below 10^{-5}.
- A trial is considered to have failed if the total time is above 4 hours, or the memory consumption per node exceeds 16 GB while using up to 8 processors per node and 24 GB when using all 16 CPUs in a node, or the final relative error norm exceeds 0.02. The relative error norm is the ratio of L2 norm of the final error to that of the initial error. For computing the error, we use the solution obtained using the WSMP direct solver.
- A trial is also considered to have failed if its performance on a desired metric is more than one order of magnitude worse than that of the best performing trial. More details on this can be found in Section 3.4.

2.7 Performance Metrics

For each successful trial, we measured and recorded the following performance metrics.

Time taken (in seconds): This is the total time required for creating the preconditioner and solving the linear system. We do not include the time for partitioning and distributing the matrices or other set-up operations that would be performed only once in a typical application with many systems to solve over which the cost of the one-time operations could be amortized.

Memory usage (in bytes): This is the amount of memory allocated on the heap during the preconditioner creation phase. In the case of multiple processor runs, we compute the *cumulative*

memory usage across all MPI processes. If GMRES is used as the iterative solver, we also add the memory required for storing the subspace vectors to the total observed memory.

Memory time product: In order to perform a meaningful comparison of solver configurations, we introduce a simple intuitive performance criterion. Specifically, we use the product of total solution time and the memory required for storing the coefficient matrix and preconditioner as our primary performance criteria. Henceforth, we will refer to this quantity as the *Memory-Time-Product* (MTP). The choice of MTP is motivated by our observation that both computation time and memory use appear to be inadequate measures of the quality of a preconditioner, when considered individually. For most preconditioners, there is a range of parameter choices in which there is a trade-off between solution time and memory consumption, although it is possible to make parameter choices that increase or decrease both time and memory simultaneously. The optimum operating point of a preconditioner for a given problem usually lies in a trade-off zone. As reported in literature [11, 14], which we also confirm by our own experiments, direct solvers often result in the overall fastest time, albeit at the cost of a significantly high memory consumption (Section 4.2). Therefore, a preconditioner could simply emulate the direct solver and emerge as the fastest preconditioner. At the other extreme, preconditioners such as Jacobi, Gauss-Seidel, or IC(0), consume very little memory, but can take an impractically large number of iterations to converge. As a result, judging the quality of preconditioners based solely on their time or memory requirements simply yields winners that are extreme cases and are of little practical interest.

3 Benchmarking Methodology

In this section, we present the benchmarking methodology that we use in the paper to evaluate the preconditioners and the relative performance of their various configurations resulting from different choices of parameters and other user selectable options. Our methodology is based on performance profiles [8], which we augment with some other metrics described later in this section.

3.1 Solver Configurations and Performance Data

A solver configuration is a solver and preconditioner implementation with a given set of values for all parameters and user selectable options. For example, Hypre's CG solver and its ParaSails preconditioner, with RCM ordering, 2 levels of fill, a threshold of 0.01, and a filter value of 0.05 is one solver configuration, PETSc's CG solver and BlockSolve95 preconditioner with ND ordering is another. We denote the set of all solver configurations by $\mathcal{S}$. Let $|\mathcal{S}| = m$. The set $\mathcal{S}$ used in our study was constructed by using all feasible combinations in Table 2, with $m = 470$ for the single processor case. We denote by $\mathcal{P}$ the set of linear systems/problems to be solved with $|\mathcal{P}| = n = 30$ in our study (Table 3). A trial is the application of a solver configuration to a problem. We performed $m \times n = 14100$ trials for the single processor case.

Let μ represent any performance measure that takes a specific value for each evaluation trial. Examples of performance measures include time taken, memory usage, memory-time product, etc. The $n \times m$ trials result in an $n \times m$ matrix μ of performance data for each performance metric, where the element (p, s) corresponds to the performance $\mu_{p,s}$ of solver configuration s with respect to problem p. The performance values $\mu_{p,s}$ may not always be well-defined due to solver configuration failure and other practical limitations. Without loss of generality, we assume that lower values of performance values are desirable and therefore, we represent ill-defined values corresponding to solver configuration failures with a very high value (∞).

The solver configurations are partitioned into configuration groups to facilitate the analysis of the performance data collected through the trials. Each solver configuration belongs to one or more (possibly overlapping) groups. For example, all solver configurations for the Hypre package could be considered to belong to a configuration group, all solver configurations using the BoomerAMG preconditioner could be considered to belong to another configuration group, and all solver configurations using an algebraic multigrid preconditioner (i.e., Hypre BoomerAMG and Trilinos ML) could be considered to belong to a third configuration group.

3.2 Performance Ratios

Given the data for a particular performance measure, it is straightforward to compare the effectiveness of the methods with respect to a single problem. Specifically, we assume that methods with lower performance values are better. However, comparing methods based on their collective performance requires calibration across the problems. A natural way to compare the solver configurations in a configuration group $\mathcal{C}$ would be to consider the normalized performance values $r_{p,s}(\mathcal{C})$, otherwise known as the performance ratios of the methods for each problem:

$$r^{\mu}_{p,s}(\mathcal{C}) = \frac{\mu_{p,s}}{\min\limits_{s' \in \mathcal{C}} \mu_{p,s'}} \ ,$$

which is the ratio of the actual performance value of solver configuration s to the best (least) value over all solver configurations for the problem p. Note that $r_{p,s}(\mathcal{C}) \geq 1$ for all (p, s) and is equal to 1 for at least one solver configuration $s \in \mathcal{C}$ for each problem p, as long as at least one $s \in \mathcal{C}$ is able to solve the problem p.

It seems reasonable that the average performance ratio $\eta_s(\mathcal{C})$ of a configuration $s \in \mathcal{C}$ would be a fair indicator of the effectiveness of the configuration s with respect to that performance metric μ, where

$$\eta^{\mu}_{s}(\mathcal{C}) = \frac{1}{n} \sum_{p=1}^{n} r^{\mu}_{p,s}(\mathcal{C}).$$

In practice, however, η_s is often not very useful since a single failure for a solver configuration s can make its average performance ratio η_s ill-defined, making it difficult to compare the different methods. One simplistic solution for handling this issue is to only consider problems that have well defined performance ratios, but this would not be fair to methods that actually solve the harder problems not solved by all the methods. A more principled approach is to compare the performance of the methods both in terms of the number of problems solved as well as average performance ratio directly using the distribution of the performance ratios. To achieve this, we use the notion of performance profiles, which we now describe.

3.3 Performance Profile

A performance profile [8] is a plot of the cumulative distribution of the performance ratios. Let $\rho^{\mu}_{s}(\tau)$ denote the cumulative distribution of the performance ratios of a solver configuration s with respect to the measure μ:

$$\rho^{\mu}_{s}(\tau) = \frac{1}{n} |r^{\mu}_{p,s}(\mathcal{C}) \leq \tau|.$$

$\rho^{\mu}_{s}(\tau)$, therefore, denotes the fraction of the problems that the configuration $s \in \mathcal{C}$ can solve with performance that is within a factor of τ of the best performance for each problem.

Solver Configurations	**Configuration Groups**	$\mu_{p,s}$			$r^{\mu}_{p,s}(\mathcal{C}_1)$		
		p_1	p_2	p_3	p_1	p_2	p_3
s_1	$\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$	7	2	3	3.5	1	1
s_2	$\mathcal{C}_1$, $\mathcal{C}_2$	3	4	6	1.5	2	2
s_3	$\mathcal{C}_1$, $\mathcal{C}_3$	2	∞	5	1	∞	5/3

Table 4: Hypothetical performance data with three solver configurations (s_1, s_2, s_3), three configuration groups ($\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$), and three problems (p_1, p_2, p_3). Solver configuration failures are represented by ∞.

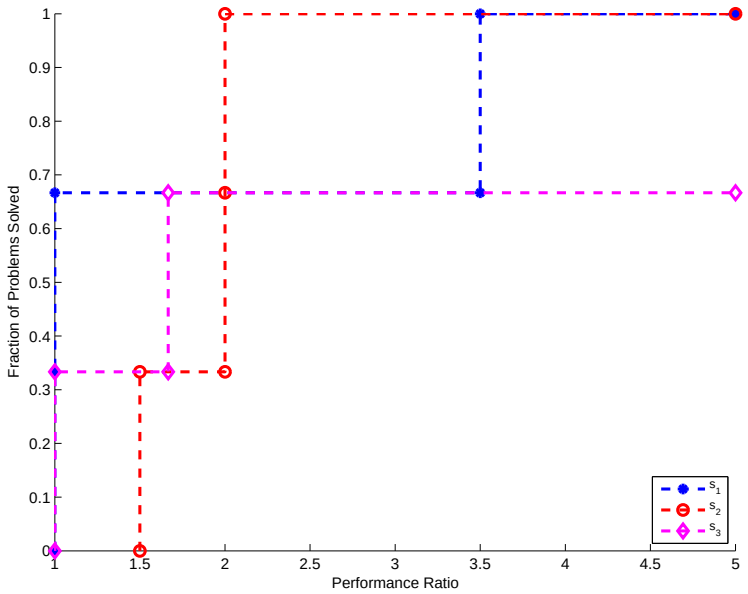


Figure 1: Performance profile curves for solver configurations in group $\mathcal{C}_1$.

Table 4 shows hypothetical performance data (say, run time in seconds) for a set of three solver configurations and three problems. There are three configuration groups: $\mathcal{C}_1 = \{s_1, s_2, s_3\}$, $\mathcal{C}_2 = \{s_1, s_2\}$, and $\mathcal{C}_3 = \{s_1, s_3\}$. Figure 1 shows the performance profiles for the configuration group $\mathcal{C}_1$ shown in Table 1. The performance profile plot readily reveals information that may not be apparent from average performance ratios even when they are well defined. For example, the point (1.67, 0.67) on the plot for configuration s_3 denotes that this configuration was able to solve 67% of the problem within 1.67 times the best running time for any configuration for these problems. Similarly, the point (2.0, 1.0) on the plot for configuration s_2 denotes that this configuration was able to solve all (100%) of the problems consuming at most twice the best running time for each problem. Performance profiles also enable one to easily compare different solver configurations when an additional success threshold θ is specified; i.e., any solver configuration that results in performance value that is θ times greater than that of the best value is considered a failure. In the example in Table 4 and Figure 1, s_1 is clearly the best method for $\theta = 1.3$, followed by s_3 and s_2. On the other hand, for $\theta = 4$, both s_1 and s_2 are equally good since they both solve all the three problems and that too with identical average performance ratio of 11/6.

3.4 Solver Configuration Quality Measure

When comparing a large number of performance profile curves, it may not be possible to visually determine the relative ordering or to pick the best solver configuration. To address this issue and to eliminate human intervention in the case of a large number of solver configurations, we propose to use the area under the performance profile curves for ranking the various solver configurations. The area under the curve (AUC) provides a longitudinal summary of multiple assessments at all possible performance ratios of interest. For a performance ratio threshold of θ, the only τ values of interest are those from 1 to θ. Therefore, the relevant area $AUC_s^\mu(\mathcal{C}, \theta)$ under the performance profile curve for a configuration s with respect to the configuration group $\mathcal{C}$ is the area under the cumulative distribution curve up to θ, which is given by

$$AUC_s^\mu(\mathcal{C}, \theta) = \theta - \frac{1}{n} \sum_{p \in \mathcal{P}} \min(\theta, r_{p,s}^\mu(\mathcal{C})).$$

This formulation, ignores all the performance ratios of solver-configurations outside the range of the threshold θ and effectively considers those as failures. For the example in Figure 1, $AUC_{s_1}^\mu(\mathcal{C}_1, 5.0)$ $= 3.17$, $AUC_{s_2}^\mu(\mathcal{C}_1, 5.0) = 3.17$, and $AUC_{s_3}^\mu(\mathcal{C}_1, 5.0) = 2.44$. We also notice that AUC of s_3 is comparable to that of s_1 and s_2 for smaller values of θ, but becomes progressively worse as the threshold increases.

Note that for the special case where there are no solver configuration failures and all performance ratios are less than or equal to θ, the area under curves is directly related to average performance ratios. The choice of θ is critical in the AUC-based comparison since this can significantly affect the AUC and thereby, the relative ranking of the different methods. In the current study, we choose this threshold to be equal to 10. In other words, we assume that a trial that results in performance that is more than an order of magnitude worse than the best performance for a given problem is effectively a failure. This is in addition to the failure criteria described in Section 2.6.

3.5 Configuration Group Quality Measures

So far in this section, we have discussed metrics for comparing the relative performance of individual solver configurations. It is often desirable to compare different configuration groups. For example, for a given problem set $\mathcal{P}$, it would be interesting to be able to objectively compare the implementations of a preconditioner in different packages. In our study, we measure and compare the following metrics for configuration groups comprised of various configurations of a given package-preconditioner combination.

3.5.1 Default configuration performance

For a configuration group $\mathcal{C}$, we define $DEF^\mu(\mathcal{C})$ as the solver configuration $s' \in \mathcal{C}$ such that $AUC_{s'}^\mu(\mathcal{C}, \theta)$ is the maximum among all $AUC_s^\mu(\mathcal{C}, \theta)$ for all $s \in \mathcal{C}$. In other words, $DEF^\mu(\mathcal{C})$ is the configuration that results in overall best performance with respect to metric μ for a given problem set $\mathcal{P}$. Therefore, if $\mathcal{C}$ represents a package-preconditioner combination, then the values of the various parameters and user selectable options corresponding to $DEF^\mu(\mathcal{C})$ are logical choices for default parameters for $\mathcal{P}$. The performance of the configuration $DEF^\mu(\mathcal{C})$ can be considered representative of the performance of configuration group $\mathcal{C}$, and can be used to compare different groups by using the performance profiles and the AUC metric.

3.5.2 Problem-specific best performance

An alternative to using the performance of $DEF^{\mu}(\mathcal{C})$ to represent the performance of configuration group $\mathcal{C}$ is to represent it by its problem-specific best performance. Formally, we define the problem-specific best performance $\mu_{p,\mathcal{C}}^{PSB}$ of configuration group $\mathcal{C}$ for problem p as the best performance value among all the solver configurations in $\mathcal{C}$; i.e.,

$$\mu_{p,\mathcal{C}}^{PSB} = \min_{s \in \mathcal{C}} \mu_{p,s}.$$

Note that $\mu_{p,\mathcal{C}}^{PSB}$ is derived from the performance values of all the member configurations of $\mathcal{C}$. This is in contrast to $\mu_{p,DEF^{\mu}(\mathcal{C})}$, which corresponds to the performance of a particular (the overall winner) member configuration of $\mathcal{C}$.

3.5.3 Fine-tuning gain

An important issue in the context of problem-specific performance analysis is related to the benefits of fine-tuning. Specifically, one would like to quantitatively measure how sensitive a given preconditioner implementation is to fine-tuning its parameters. The benefits of fine-tuning within a configuration group $\mathcal{C}$ comprised of multiple configurations of a package-preconditioner combination can then be captured in terms of the metric fine-tuning gain or $\nu_{p,\mathcal{C}}$ defined as

$$\nu_{p,\mathcal{C}} = \frac{\mu_{p,\mathcal{C}}^{PSB}}{\mu_{p,DEF^{\mu}(\mathcal{C})}}.$$

Just like the problem specific best performance, the fine-tuning gain is defined for every configuration group and problem and can be considered a meta performance metric associated with the configuration groups. However, it is different from the core performance metric μ in the sense that is not defined for each solver configuration-problem trial. The fine-tuning gain can be viewed as an indicator of both the potential benefits of tuning the parameters of a package-preconditioner combination, as well as its sensitivity to its user controlled parameters.

3.6 Hardware Configurations

The performance metrics $\mu_{p,s}$ obtained for each problem p and solver configuration s is also a function of the hardware configuration. Let $\mathcal{H} = \{h\}$ denote the set of all hardware configurations on which performance data is obtained from. For the current study, $h \in \{1, 2, 4, 8, 16, 32, 64\}$ where each number consists of the number of processors used for each trial. We study the performance with respect to one value of h at a time.

3.7 Parallel Performance

Most solver packages included in this study are designed to solve large sparse systems in highly parallel environments. In the parallel case, users may be interested in additional performance metrics other than those studied in the context of a single processor. For example, a user might be interested in knowing how the relative performance of the solvers observed in a serial environment changes in various parallel settings. We consider each multi-processor run to be part of a different hardware group. The various solvers are evaluated in each of these hardware groups separately and the AUC of performance profiles are used to study the behavior of the solvers under various hardware configurations. An important performance metric in a parallel scenario is the efficiency of the respective parallel implementations. Efficiency is computed as $\epsilon = \frac{1}{np} Time^{sp}/Time^{np}$ where

Parameter Name	Values	Acronyms
Level of fill	0, 1, 2, 4, 6 ,8	LF0, LF1, LF2, LF4, LF6, LF8
IC(k) fill factor	1, 3, 5, 8, 10	F1, F3, F5, F8, F10
Max. nonzeros per row	∞	NzINF
Drop tolerance	0.03, 0.01, 0.003, 0.001, 0.0003, 0.0005	DT3e-2, DT1e-2, DT3e-3, DT1e-3, DT3e-4, DT5e-4
Inverse norm estimate	10, 25, 50, 75, 100	IE10, IE25, IE50, IE75, IE100
Number of ParaSails levels	0, 1, 2	PLev0, PLev1, PLev2
Threshold	0, 0.01, 0.1, -0.75, -0.9	Th0, Th.01, Th.1, Th-.75, Th-.9
Filter	0, 0.001, 0.05, -0.9	Flt0, Flt.001, Flt.05, Flt-.9
BoomerAMG coarsening schemes	Falgout, Hybrid MIS, Parallel MIS	FALG, HMIS, PMIS
Strong threshold	0.25, 0.5, 0.7, 0.9	ST.25, ST.5, ST.7, ST.9
Aggresive coarsening levels	0, 10	AGG0, AGG10
ML preconditioner type	Classical SA, SA based 2-level domain decomposition, 3-level algebraic domain decomposition	SA, DD, DD-ML
Smoothers	Symmetric Gauss-Seidel, Chebyshev, Incomplete factorization	SGS, CBY, IFPACK
Smoother sweeps	1, 2, 3	SS1, SS2, SS3
ML coarsening schemes	Uncoupled, ParMETIS, MIS, Hybrid uncoupled-MIS	UC, PMETIS, MIS, UCMIS
WSMP fill factor	2.5, 3.3, 4.1, 4.9	F2.5, F3.3, F4.1, F4.9
WSMP diagonal shift	-1, 0.001	SHIFT-OFF, SHIFT-ON

Table 5: List of acronyms used to denote various parameter choices. These acronyms are used in the legends of most tables and figures in Section 4.

$Time^{sp}$ is the best sequential time and $Time^{np}$ represents the time observed for np processors. A relatively high efficiency for large processors could either suggest that the solver can be parallelized efficiently or that the serial implementation is not optimal. Similarly, a low value of efficiency could suggest the existence of expensive sequential components or a poor parallel implementation.

4 Results and Discussion

In this section, we present the results of our empirical evaluation. First, in Section 4.1, we analyze the performance of various solver configurations within configuration groups comprised of package-preconditioner combinations. Different configurations in a group are defined by different combinations of user-controlled parameters, whose abbreviated names are shown in Table 5 and appear in the legends of the tables and figures in this section. We report the best configuration for each group over all the problems with respect to time, memory, and memory-time product. In addition, we also discuss the effect of certain important parameters on the memory and time performance. In Section 4.2, we compare the default and problem-specific best performance of the various configuration groups along with the performance of the WSMP direct solver.

4.1 Performance Within Configuration Groups

For the purpose of the analysis in this section, we divided the set of all solver configurations into configuration groups, where each group represents a package-preconditioner combination.

Preconditioner	Memory Winner	Time Winner	MTP Winner
PETSc IC(k)	CG, RCM, LF0, F1 (1 2 8 16 32 64) CG, ND, LF0, F1 (4)	CG, RCM, LF0, F1 (1 2 4 8 16 32 64)	CG, RCM, LF0, F1 (1 2 4 8 16 32 64)
PETSc BlockSolve	CG, RCM (2 4 8 16 32 64) CG, ND (1)	CG, RCM (1 2 16 32 64) CG, ND (4 8)	CG, RCM (1 2 4 16 32 64) CG, ND (8)
Trilinos IC(k)	CG, RCM, LF4 (1 2) CG, RCM, LF6 (4 8 16 32 64)	CG, RCM, LF8 (1 2 4 8 16 32 64)	CG, RCM, LF4 (2) CG, RCM, LF6 (32 64) CG, RCM, LF8 (1 4 8 16)
Hypre IC(k)	CG, RCM, LF1, NzINF (1 64) CG, RCM, LF2, NzINF (2 4 16 32) CG, ND, LF1, NzINF (8)	CG, RCM, LF1, NzINF (1) CG, RCM, LF2, NzINF (2 64) CG, ND, LF1, NzINF (8) CG, ND, LF2, NzINF (4 16) CG, ND, LF6, NzINF (32)	CG, RCM, LF1, NzINF (1 2 16 32 64) CG, ND, LF1, NzINF (4 8)

Table 6: Configurations that resulted in the best overall memory, time, and MTP performance profile area for IC(k) preconditioners in PETSc, BlockSolve95, Hypre, and Trilinos for various numbers of processors (shown in parentheses). Expansions of the parameter acronyms can be found in Table 5.

For each configuration group and hardware configuration, we identified the solver configurations that resulted in the best overall performance with respect to time, memory, and memory-time product (MTP) over all the problems in the test suite. These are shown in Tables 6–9. The best performance was determined using the area (AUC) under the performance profile (PP) curves, as discussed in Section 2.7. We also provide detailed analyses of the effects of various parameters for suitable subsets of the serial configurations by means of the PP curves in Figures 2–19. We chose these figures on a case-by-case basis depending on the interesting performance trends that we found for individual preconditioner implementations.

4.1.1 Level-based incomplete Cholesky, IC(k)

The PETSc, Trilinos, and Hypre packages include implementations of the IC(k) preconditioner. We have also included the BlockSolve95 preconditioner in this section because it can be regarded as a variation of the IC(0) preconditioner. We experimented with a number of configurations of each. In the case of PETSc, experiments were conducted with values of $k = 0$, 1, and 2. Hypre and Trilinos IC(k) implementations consume reasonable time and memory for levels of fill higher than 2; and therefore, we included values of $k = 4$, 6, and 8 in our experiments. For nonzero values of k, we experimented with fill factors of 3, 5, 8, and 10 for PETSc. Trilinos IC(k) does not provide a user controlled parameter for controlling the fill factor and the default setting in Hypre IC(k) is ∞, i.e., no limit. Table 6 shows the overall best configurations with respect to time, memory, and

Preconditioner	Memory Winner	Time Winner	MTP Winner
ILUPACK MLICT	CG, AMF DT1e-2, IE10	CG, AMF DT3e-3, IE75	CG, RCM DT3e-02, IE75
WSMP ICT	AUTO, ND, DT3e-3 F3.3, SHIFT-ON	AUTO, RCM, DT1e-3 F4.9, SHIFT-OFF	AUTO, RCM, DT3e-3 F4.9, SHIFT-ON

Table 7: Configurations that resulted in the best overall memory, time, and MTP performance profile area for the ILUPACK MLICT and WSMP ICT preconditioners. Table 5 contains expansions of the parameter acronyms.

MTP for various hardware configurations (number of processors) for all the IC(k) implementations.

As is often the case with preconditioners, we noticed that the different implementations of even the relatively straightforward IC(k) had very different performance characteristics and responses to values of their parameters.

Hypre IC(k): Our experimentally determined overall best parameters for the Hypre IC(k) preconditioner for different numbers of processors are shown in Table 6. We also investigated the impact of levels of fill on performance. Figure 2 shows the memory and time profile curves for various levels of fill with RCM ordering. As suggested by the authors of the software, we set the parameter for maximum fill per row to infinity. Increasing the number of levels of fill from 0 to 1 results in increased robustness and improved run times. However, the performance drops as the number of levels is increased beyond one because the improvement in quality of the preconditioner cannot compensate for the increased memory and time required for higher levels of fill.

PETSc IC(k): The overall best parameter configurations shown in Table 6 indicate that for PETSc IC(k), level of fill $k = 0$ with RCM ordering resulted in the best overall performance. Figure 3 shows the time and memory profiles for various levels of fill with RCM ordering and fill factor of 3, 5, 8, and 10. The performance showed little variation in response to changing the fill factor; Figure 3 has three clusters of curves corresponding to fill levels 0, 1, and 2. Unlike Hypre IC(k), both the memory and the time performance of PETSc IC(k) deteriorates rapidly with even a modest increase in level of fill beyond 0.

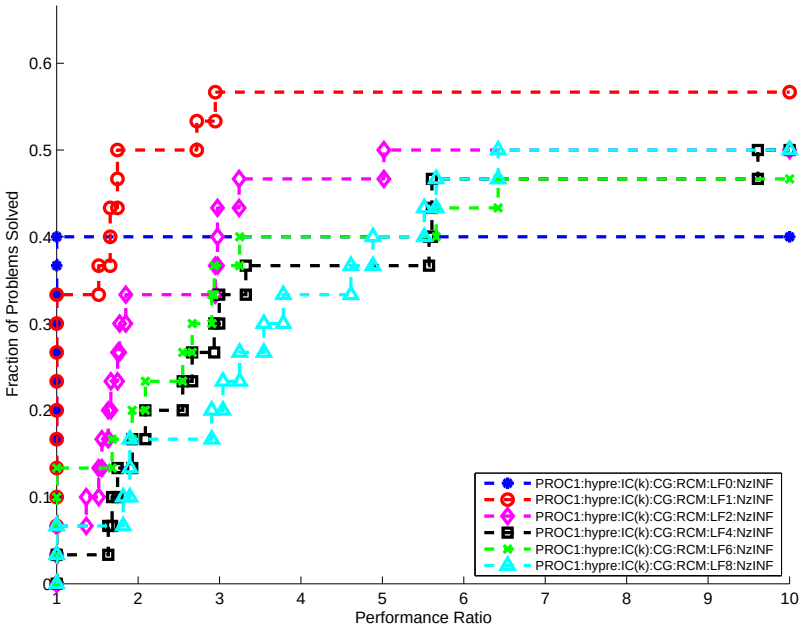
PETSc BlockSolve95: Ordering is the only user controlled parameter in BlockSolve95; however, we observed almost no difference in its performance between RCM and ND orderings.

Trilinos IC(k): Figure 4 shows the time and memory profiles for various levels of fill with RCM ordering for the Trilinos IC(k) preconditioner. The memory profile indicates that higher levels of fill solve more problems at the expense of using slightly more memory. The time profiles reveal that the higher levels of fill result in more effective preconditioners that result in faster overall solution times. This is in stark contrast to the IC(k) implementations of PETSc and Hypre.

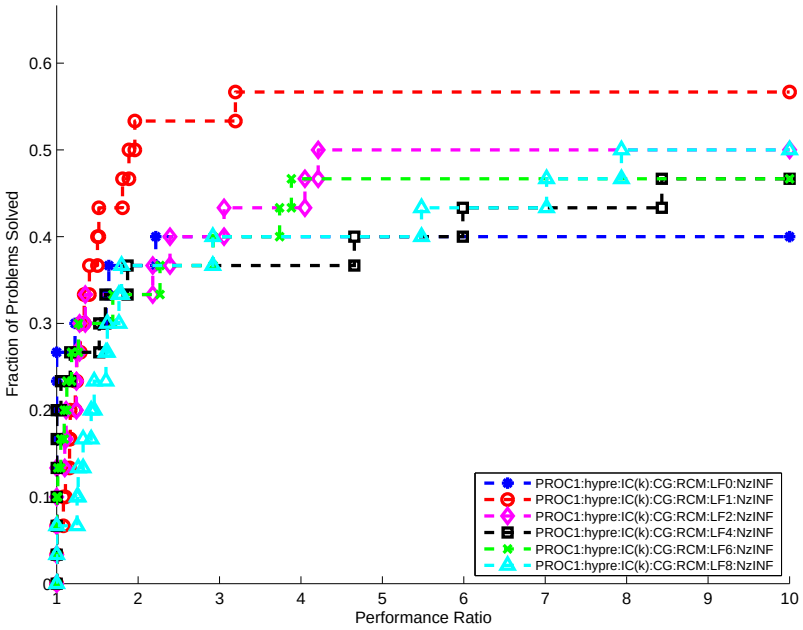
Comparison of Hypre, PETSc, and Trilinos IC(k): Figure 5 shows a comparison of BlockSolve95 and the configurations of Hypre, PETSc, and Trilinos IC(k) that resulted in the best overall MTP performance as measured by the AUC metric. PETSc IC(k) is most memory efficient because its best configuration is for level 0; however, it is less robust than Hypre whose best configuration is for level 1. For the problems that they both can solve, the overall best configurations of PETSc and Hypre are equally fast. However, the best Hypre IC(k) configuration is able to solve more problems than the best PETSc IC(k) configuration.

4.1.2 Threshold-based incomplete Cholesky, ICT

We studied the ICT preconditioners of WSMP and ILUPACK in detail. We do not report the results of ICT preconditioner implementations of other packages due to their serious performance

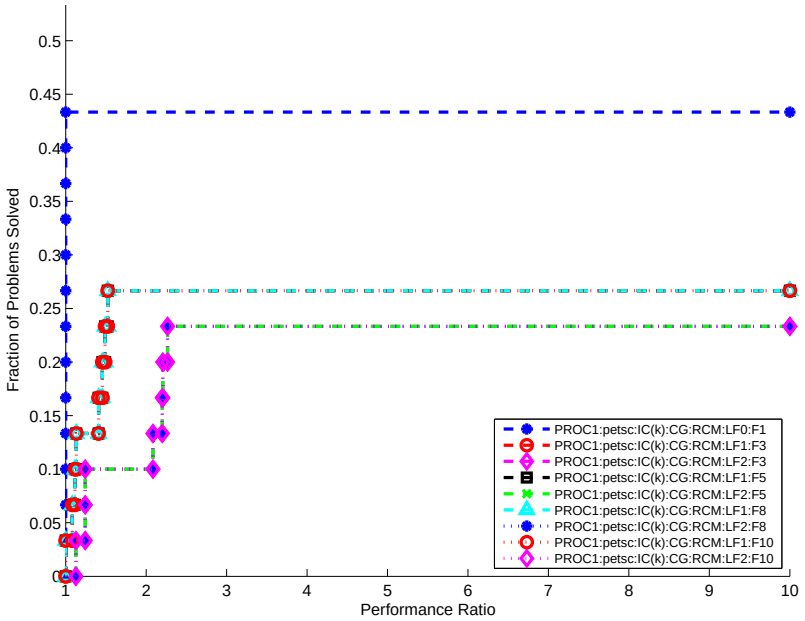


(a) Memory performance profile

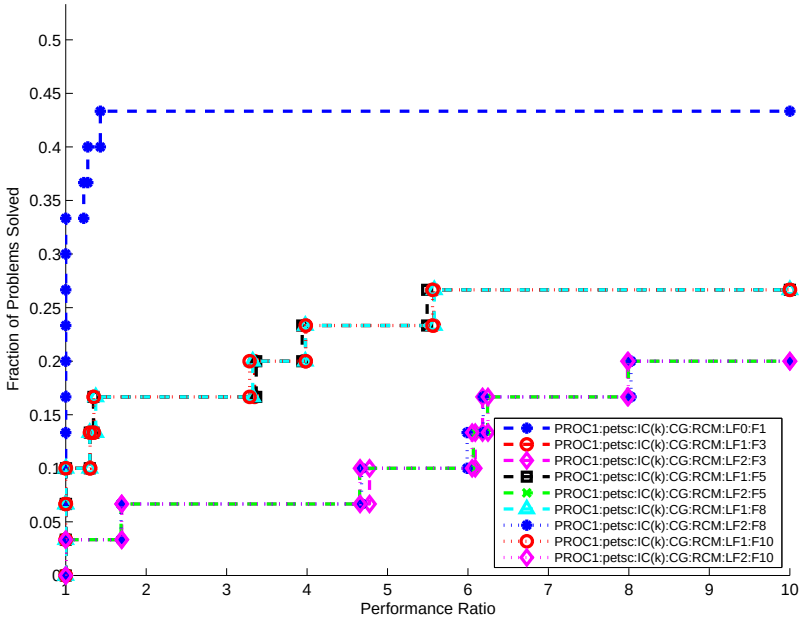


(b) Time performance profile

Figure 2: Serial memory and time profile curves for the Hypre IC(k) solver configurations with RCM ordering and various levels of fill. Expansions of the acronyms in the legends can be found in Table 5.

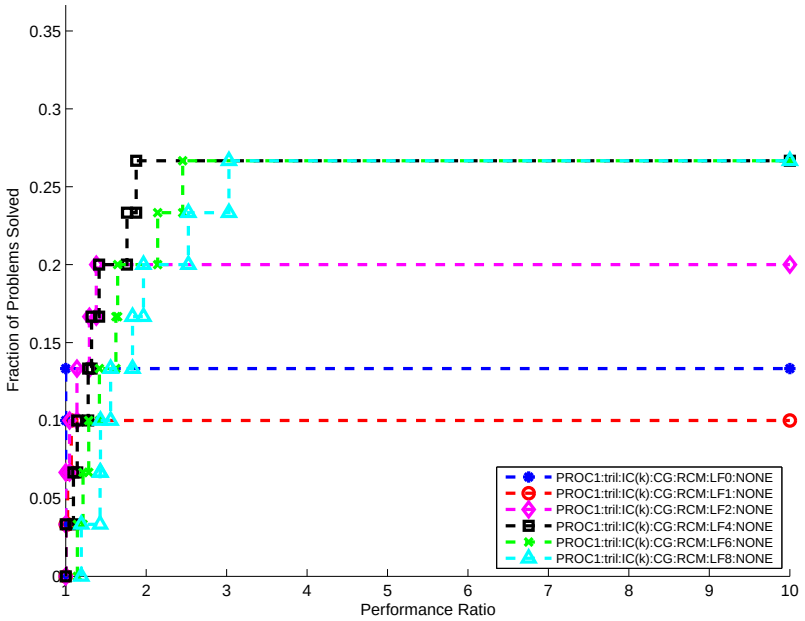


(a) Memory performance profile

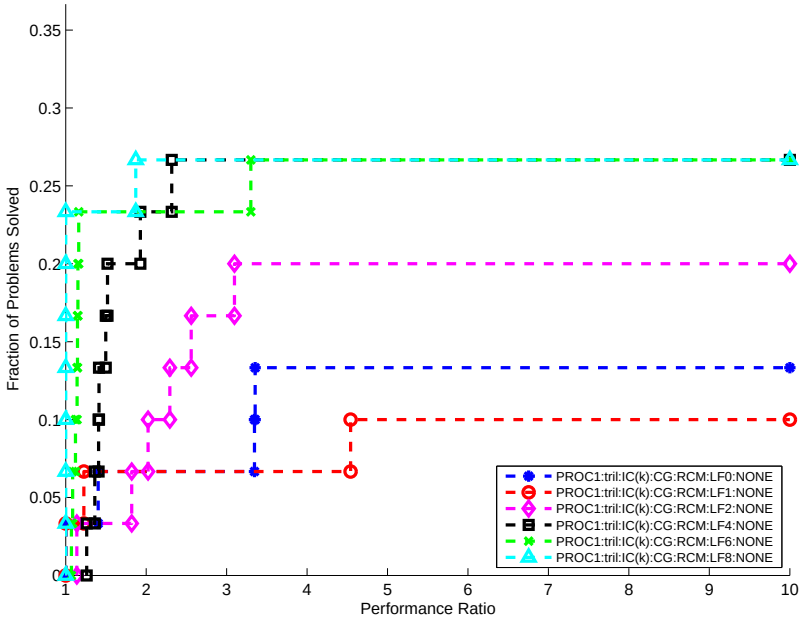


(b) Time performance profile

Figure 3: Serial memory and time profile curves for the PETSc IC(k) solver configurations with RCM ordering and various levels of fill and fill factors. Expansions of the acronyms in the legends can be found in Table 5.

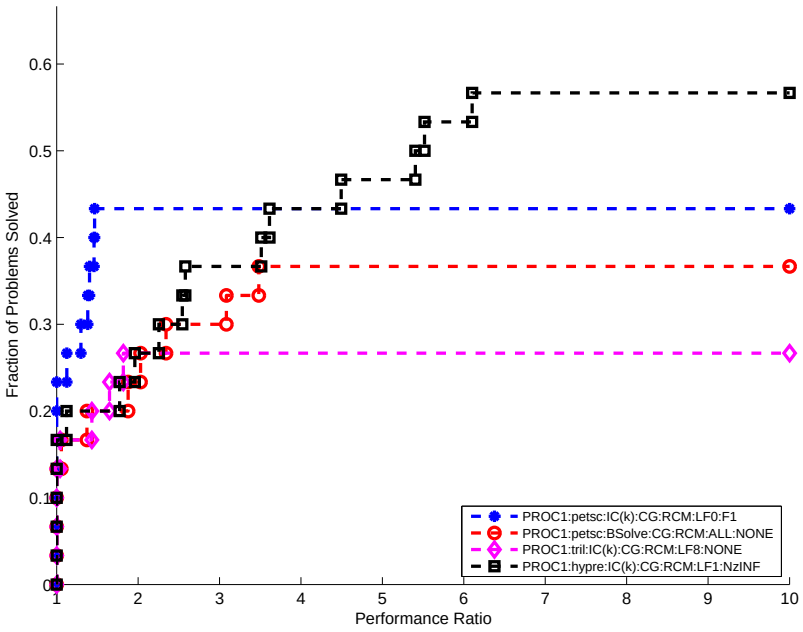


(a) Memory performance profile

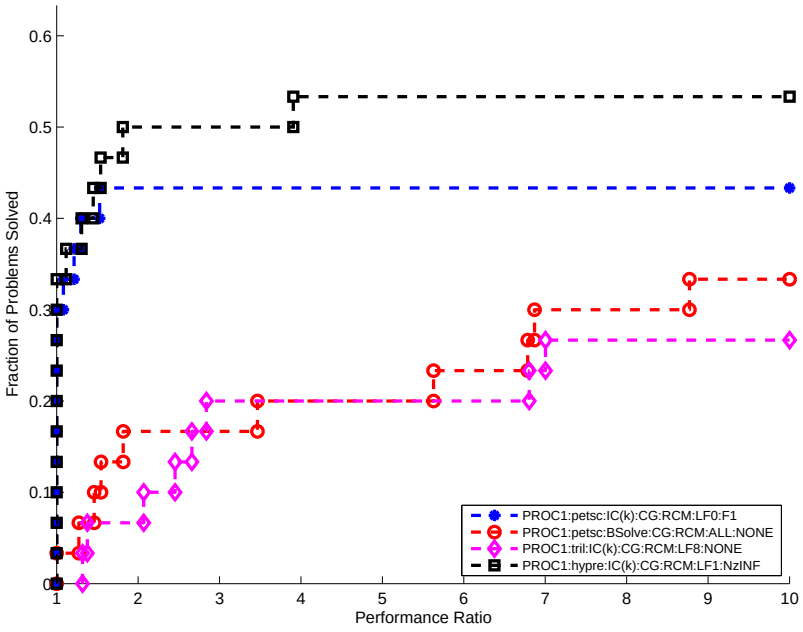


(b) Time performance profile

Figure 4: Serial memory and time profile curves for the Trilinos IC(k) solver configurations with RCM ordering and various levels of fill. Expansions of the acronyms in the legends can be found in Table 5.



(a) Memory performance profile



(b) Time performance profile

Figure 5: Serial memory and time profile curves for the overall best configurations of Trilinos IC(k), PETSc IC(k), PETSc BlockSolve95, and Hypre IC(k). Table 5 contains expansions of the parameter acronyms.

and robustness problems. For ILUPACK MLICT, we tried five different built-in reordering schemes (RCM, AMF, INDSET, PQ, and METISN), five different values for drop tolerance (0.03, 0.01, 0.003, 0.001, 0.0005), and five different values of the inverse norm estimate (10, 25, 50, 75, 100). We tried the WSMP ICT with and without diagonal perturbation with two ordering schemes (RCM, ND), four values of drop tolerance (0.01, 0.003, 0.001, 0.0003), and four values of fill factor (2.5, 3.3, 4.1, 4.9). Table 7 shows the solver configurations that resulted in the best memory, time, and MTP profile areas for ILUPACK MLICT and WSMP ICT.

ILUPACK MLICT: In Figure 6, we study the effect of drop tolerance for RCM ordering and a low inverse norm estimate value of 10. We observe that the configuration corresponding to moderately low drop tolerance value of 0.003 is the most robust and fastest, but requires more memory than higher drop tolerance values. Figure 7 shows that even for a high value of inverse norm estimate such as 100, a drop tolerance of 0.003 is most robust, but considerably more expensive than higher drop tolerance values in terms of both time and memory.

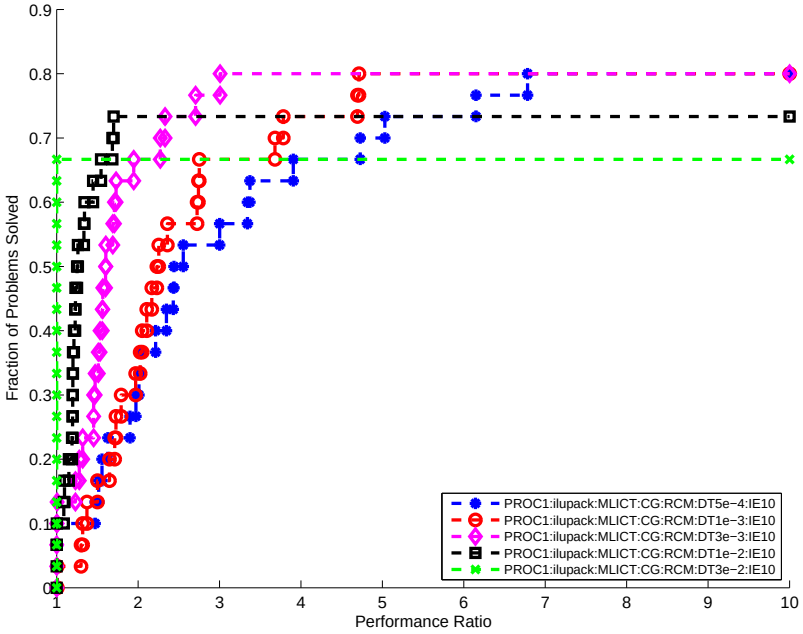
WSMP ICT: Figure 8 shows the performance profile curves corresponding to the various drop tolerance values for RCM ordering, diagonal perturbation of 0.001, and fill factor of 4.9 for WSMP ICT. Drop tolerance values of 0.001 and 0.003 seem to offer the best balance between robustness and memory and time consumption. Figure 9 shows the effect of varying the ordering and diagonal perturbation for the best MTP values of drop tolerance and fill factor (0.003 and 4.9, respectively). The use of diagonal perturbation results in more robust solver configurations. While using diagonal perturbation, ND is slightly more robust than RCM, and without it, RCM is slightly more robust. In both cases, RCM seems to be somewhat faster than ND, though they both use about the same memory.

Comparison of ILUPACK MLICT and WSMP ICT: Figure 10 shows the serial memory and time profiles of the overall best MTP configurations of ILUPACK MLICT and WSMP ICT. The figure shows that although the memory consumption of the two ICT preconditioners appears to be comparable, WSMP ICT is faster and more robust.

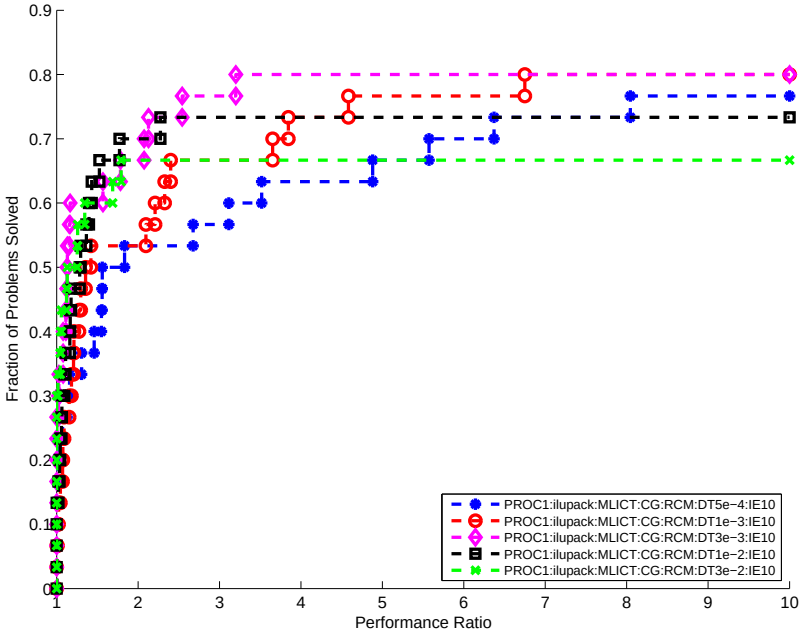
4.1.3 Algebraic multigrid preconditioners

Multigrid preconditioners typically have a large number of parameters that need to be fine tuned. We used the default values suggested in the user manuals [10, 12] for a majority of the parameters, and varied a few key ones, as shown in Table 2, based on the suggestions from the authors of Hypre and Trilinos. Table 8 shows the best configurations for this class of preconditioners.

Hypre BoomerAMG: We experimented with the ordering, coarsening scheme, maximum number of levels for aggressive coarsening, and strong threshold for the BoomerAMG preconditioner. Figures 11 and 12 show how the coarsening scheme and the number of aggressive coarsening levels affect the performance. These figures show the results with the RCM ordering, which resulted in the best MTP performance in the serial case according to Table 8. We found that the performance of the HMIS coarsening scheme to be quite similar to that of the PMIS scheme, so we have included only PMIS and Falgout schemes in these figures. Figure 11 shows the results for a relatively small value of 0.25 for the strong threshold and Figure 12 for a high value of 0.9. Figure 11 shows that Falgout coarsening scheme results in heavy memory and time usage for most problems when used without aggressive coarsening. PMIS appears to be the better coarsening scheme for our test suite with low strong threshold values. This observation is different from what is suggested in the user manual [10], which recommends Falgout coarsening scheme as the default. The results in Table 8 also indicate that the best parameter configurations in most cases include the PMIS coarsening scheme. The memory and time profiles in Figure 12 indicate that the performance difference between the various coarsening schemes is not as significant for a high strong threshold

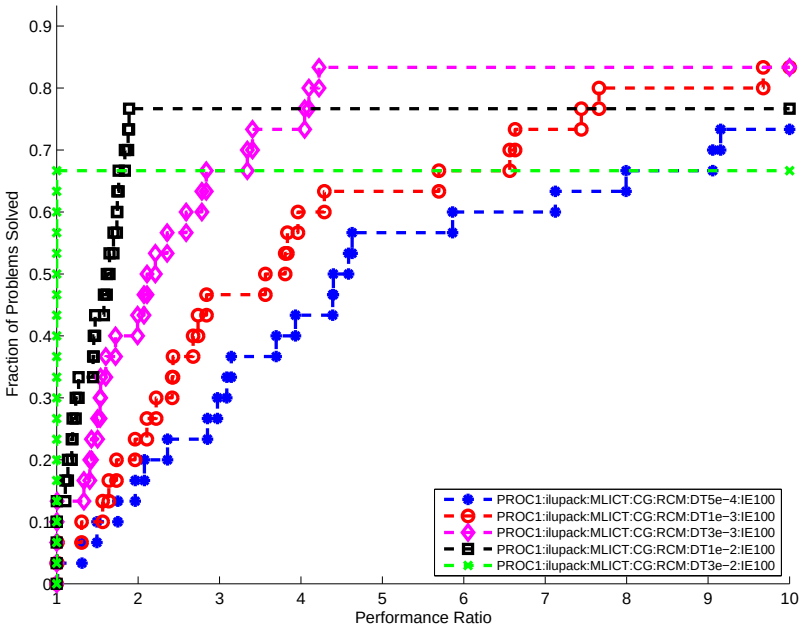


(a) Memory performance profile

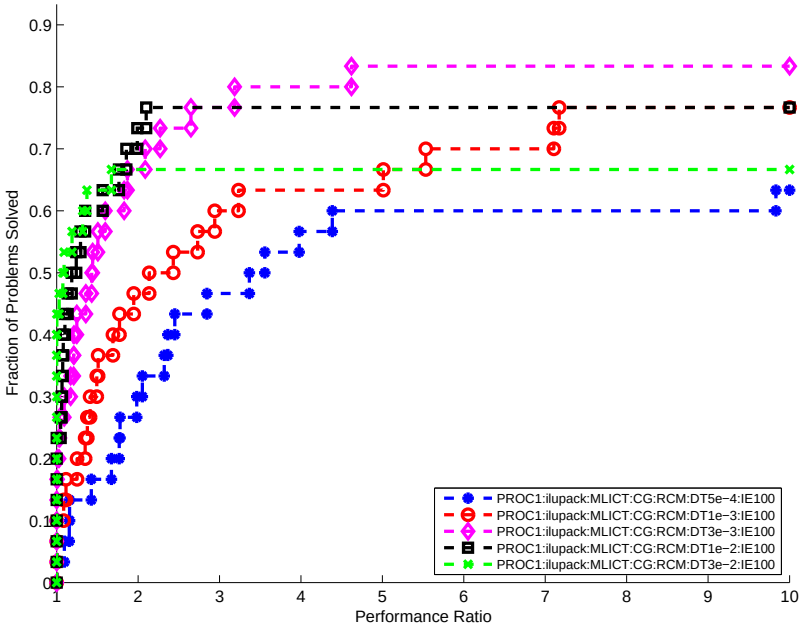


(b) Time performance profile

Figure 6: Memory and time performance profile curves for ILUPACK MLICT with RCM ordering, inverse norm estimate value of 10, and different values of drop tolerance.

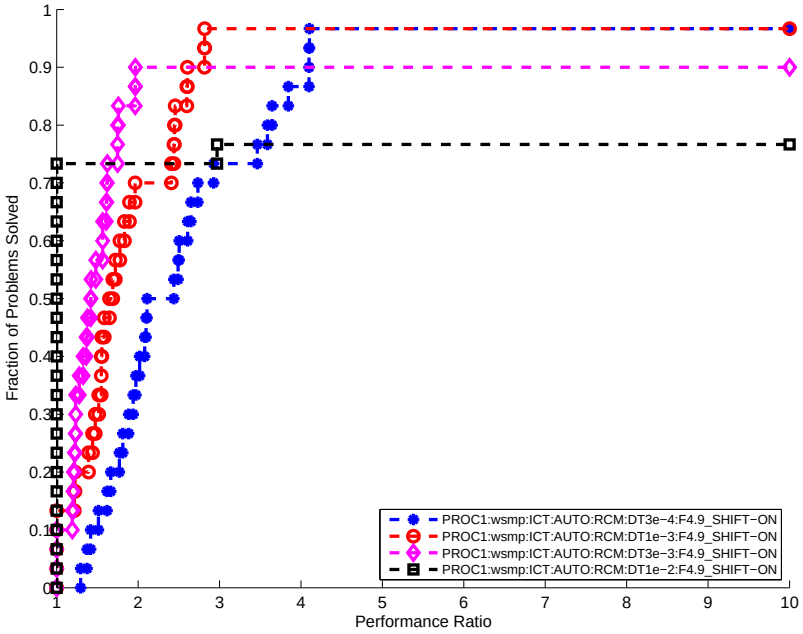


(a) Memory performance profile

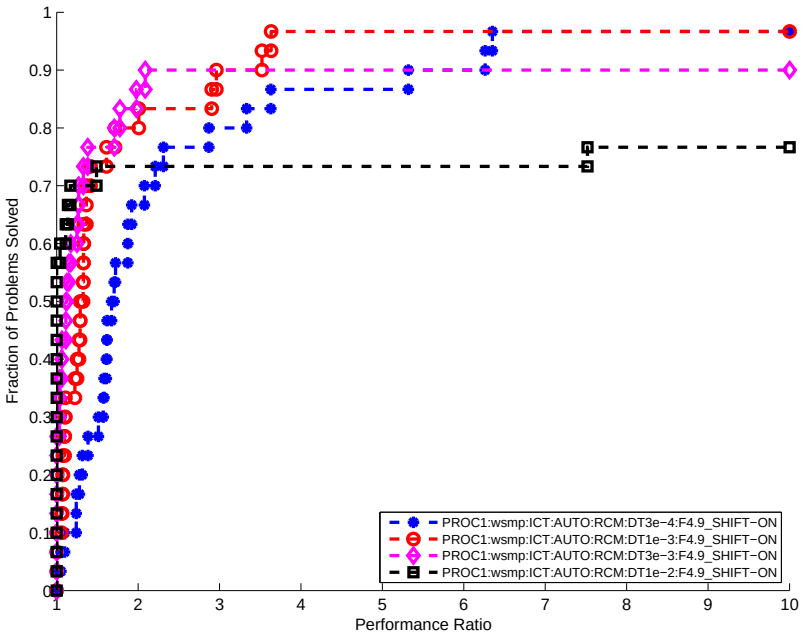


(b) Time performance profile

Figure 7: Memory and time performance profile curves for ILUPACK MLICT with RCM ordering, inverse norm estimate value of 100, and different values of drop tolerance.

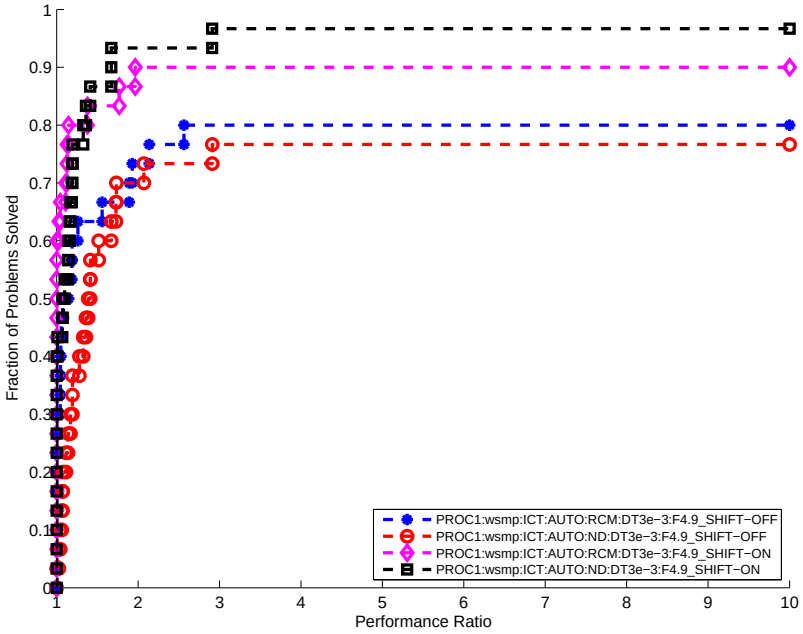


(a) Memory performance profile

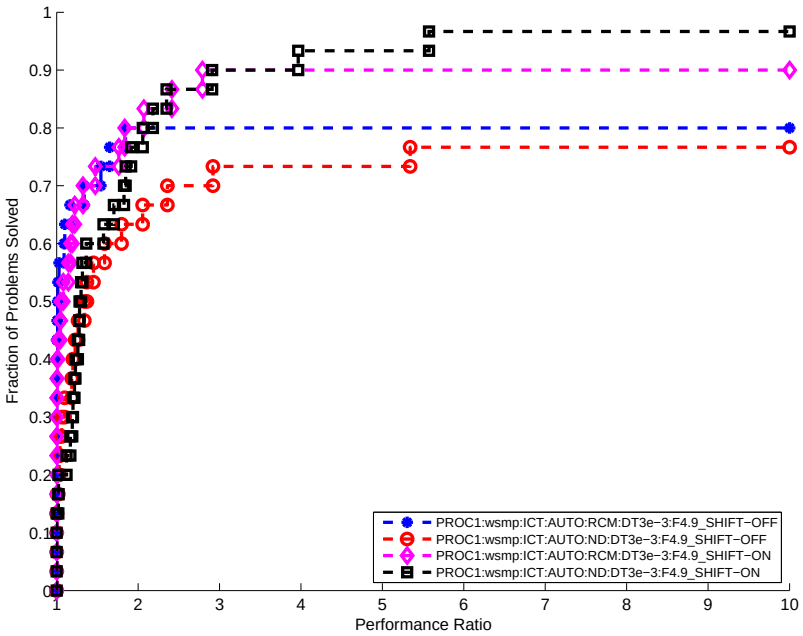


(b) Time performance profile

Figure 8: Memory and time profile curves for WSMP ICT with drop tolerance of 0.003, fill factor of 4.9, and different orderings and diagonal perturbation choices.

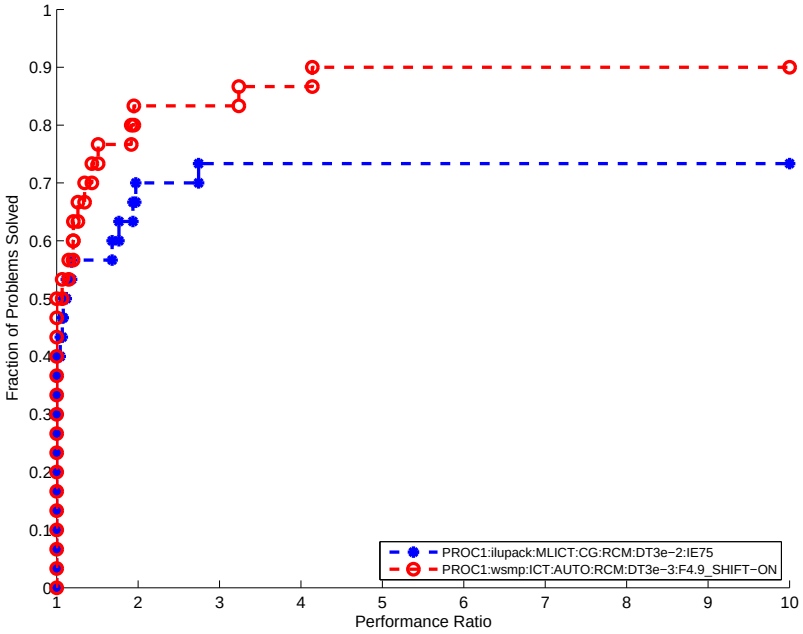


(a) Memory performance profile

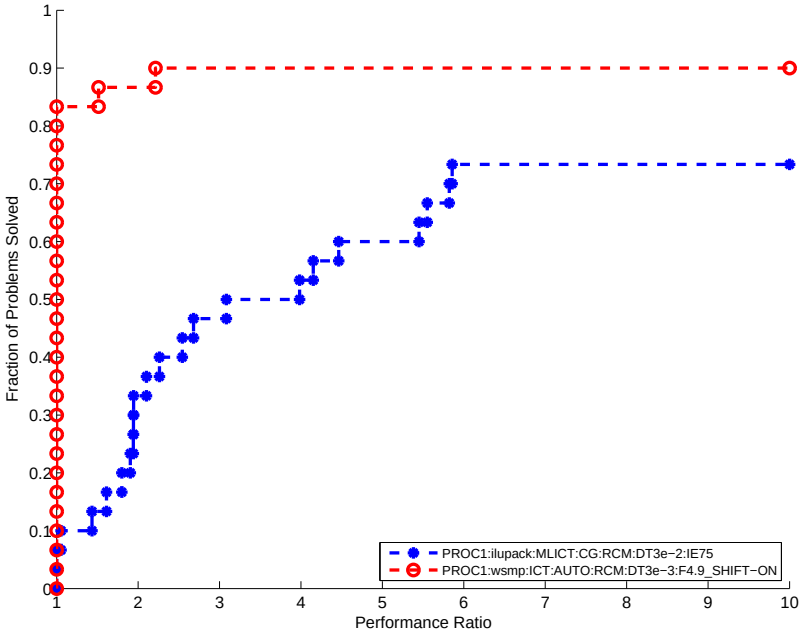


(b) Time performance profile

Figure 9: Memory and time performance profile curves for WSMP ICT with RCM ordering, diagonal perturbation turned on, fill factor of 4.9, and different values of drop tolerance.



(a) Memory performance profile

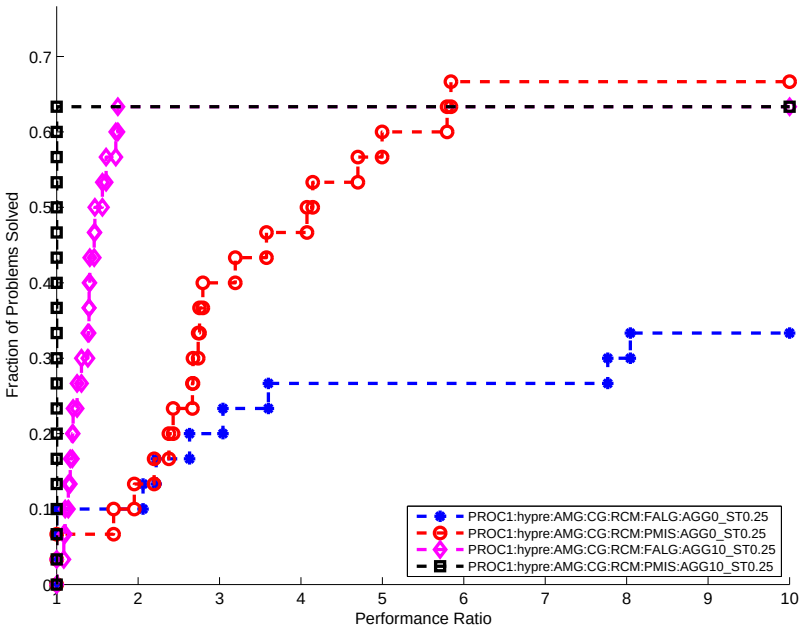


(b) Time performance profile

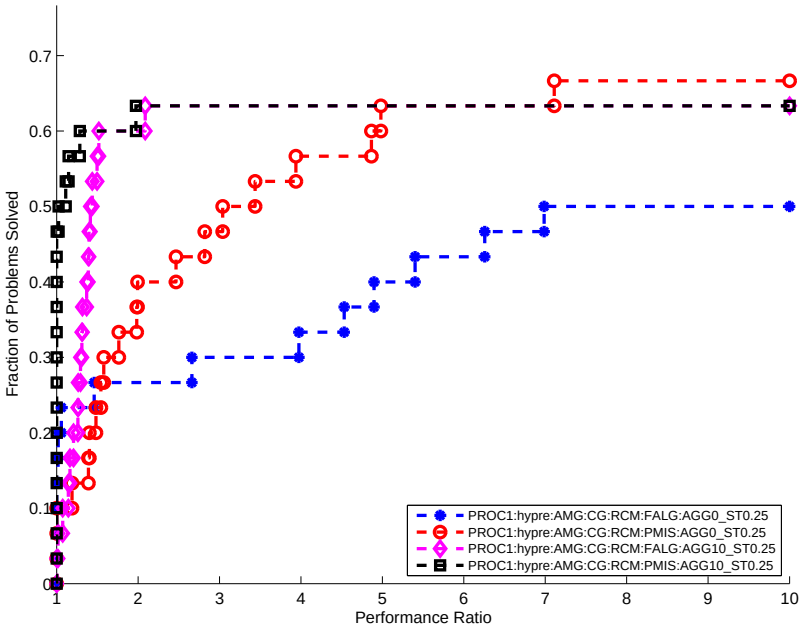
Figure 10: Serial memory and time profile curves for the overall best configurations of ILUPACK MLICT and WSMP ICT. Table 5 contains expansions of the parameter acronyms.

Preconditioner	Memory Winner	Time Winner	MTP Winner
Hypre BoomerAMG	CG, RCM, HMIS AGG10, ST0.9 (32) CG, RCM, PMIS AGG0, ST0.9 (64) CG, RCM, PMIS AGG10, ST0.9 (1 2 4 8) CG, NONE, PMIS AGG10, ST0.9 (16)	CG, RCM, HMIS AGG10, ST0.9 (32) CG, RCM, PMIS AGG0, ST0.7 (4 64) CG, RCM, PMIS AGG10, ST0.9 (1) CG, ND, FALG AGG0, ST0.9 (8) CG, NONE, FALG AGG0, ST0.9 (2) CG, NONE, PMIS AGG0, ST0.7 (16)	CG, RCM, HMIS AGG10, ST0.9 (32) CG, RCM, PMIS AGG10, ST0.9 (1 2 4 8) CG, NONE, PMIS AGG10, ST0.7 (64) CG, NONE, PMIS AGG10, ST0.9 (16)
Trilinos ML	CG, RCM, SA SGS, SS2, UCMIS (8) CG, RCM, SA SGS, SS3, UCMIS (4) CG, NONE, SA SGS, SS2, UCMIS (1 2) CG, NONE, SA SGS, SS2, MIS (32) CG, NONE, SA SGS, SS3, UCMIS (16) CG, NONE, SA SGS, SS3, MIS (64)	CG, RCM, SA SGS, SS2, UC (2 8) CG, RCM, SA SGS, SS2, PMETIS (32) CG, RCM, SA SGS, SS3, UC (4) CG, RCM, SA SGS, SS3, PMETIS (1) CG, NONE, SA SGS, SS2, UC (16) CG, NONE, SA SGS, SS3, UCMIS (64)	CG, RCM, SA SGS, SS2, UCMIS (2) CG, RCM, SA SGS, SS2, UC (8) CG, RCM, SA SGS, SS2, PMETIS (32) CG, RCM, SA SGS, SS3, UC (4) CG, RCM, SA SGS, SS3, PMETIS (1) CG, NONE, SA SGS, SS2, UC (16) CG, NONE, SA, SGS SGS, SS3, UCMIS (64)

Table 8: Configurations that resulted in the best memory, time, and MTP performance profile area for the AMG preconditioners in Hypre and Trilinos. The numbers enclosed in parentheses denote the number of processors. Table 5 contains expansions of the parameter acronyms.

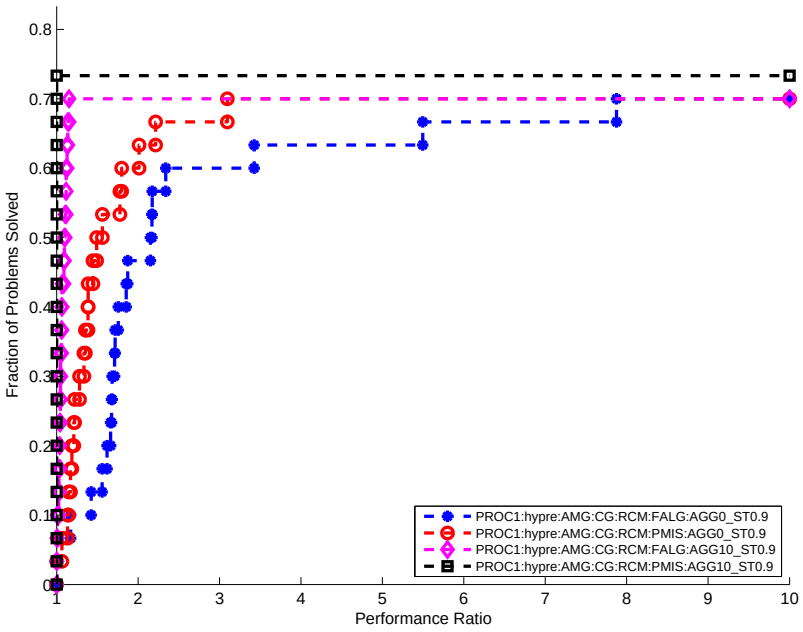


(a) Memory performance profile

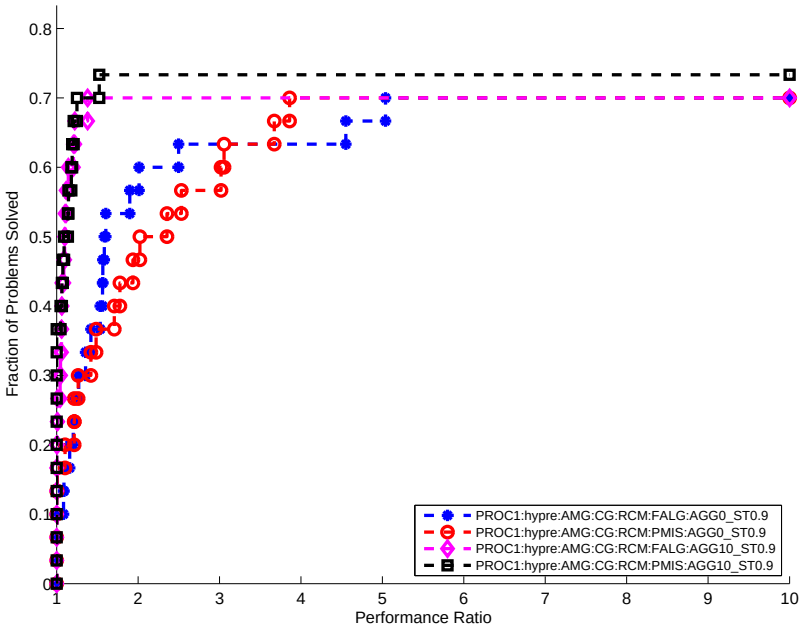


(b) Time performance profile

Figure 11: Serial memory and time profile curves for Hypre BoomerAMG with RCM ordering, a strong threshold value of 0.25, and two different coarsening schemes and aggressive coarsening levels each. Table 5 contains expansions of the parameter acronyms.



(a) Memory performance profile



(b) Time performance profile

Figure 12: Serial memory and time profile curves for Hypre BoomerAMG with RCM ordering, a strong threshold value of 0.9, and two different coarsening schemes and aggressive coarsening levels each. Table 5 contains expansions of the parameter acronyms.

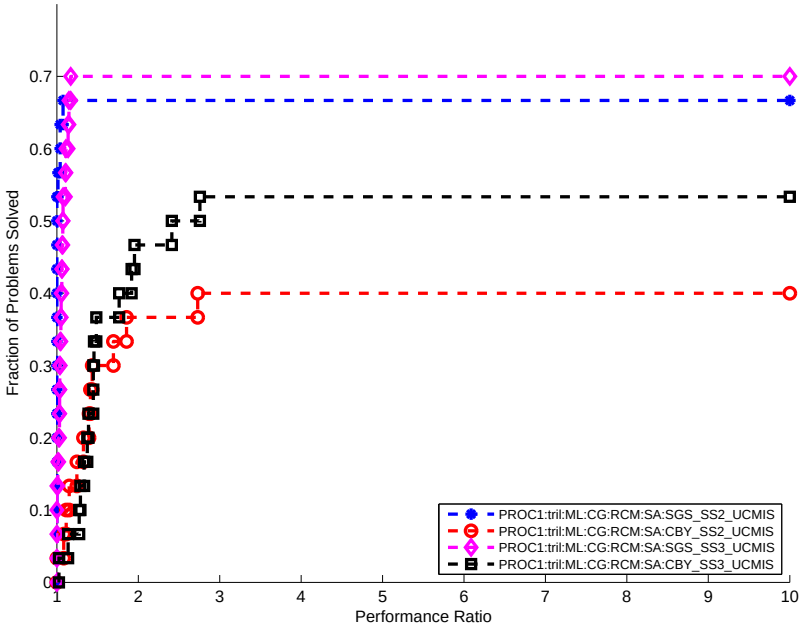


Figure 13: Serial time profile curves for Trilinos ML with RCM ordering, UCMIS coarsening scheme, and two different smoothers and smoother sweep values each. Table 5 contains expansions of the parameter acronyms.

value, especially when aggressive coarsening is used. Note that the authors recommend a high value of strong threshold for 3D problems, which constitute about 50% of our test suite.

Trilinos ML: For the ML preconditioner in Trilinos, we compared the performance of classical smoothed aggregation (SA), two level SA based domain decomposition (DD), and three level algebraic domain decomposition (DD-ML) with their predefined default set of parameters as described in Trilinos ML 5.0 user's guide [12]. In addition, we also experimented with multiple coarsening schemes, smoothers, and the number of smoother sweeps for the SA preconditioner.

Figure 13 shows the time profiles for varying the number of smoother sweeps for the symmetric Gauss-Seidel and Chebyshev smoother. We observe that increasing the number of sweeps form two to three significantly improves the robustness of Chebyshev polynomial smoother. For the symmetric Gauss-Seidel smoother, increasing the smoother sweeps causes only a slight change in the number of problems solved. We plot only the time profiles since the memory usage is not affected by the number of smoother sweeps. Overall, the symmetric Gauss-Seidel smoother is faster and solves more problems.

Figure 14 shows the effect of various coarsening schemes on the performance of Chebyshev and symmetric Gauss-Seidel smoothers. Once again, we do not show the memory profiles since they are very similar for all the coarsening schemes. The time profiles indicate that the performance of all coarsening schemes is nearly identical in Trilinos ML with classical smoothed aggregation with the exception of MIS, which performs slightly worse than others.

Figure 15 shows a comparison of the time and memory usage of the DD, DD-ML configurations, and the overall best SA configuration while using the ParMETIS coarsening scheme. We observe that the smoothed aggregation approach significantly outperforms DD and DD-ML and that the difference is much more pronounced in memory than in time.

Comparison of Hypre BoomerAMG and Trilinos ML: Figure 16 shows a comparison of

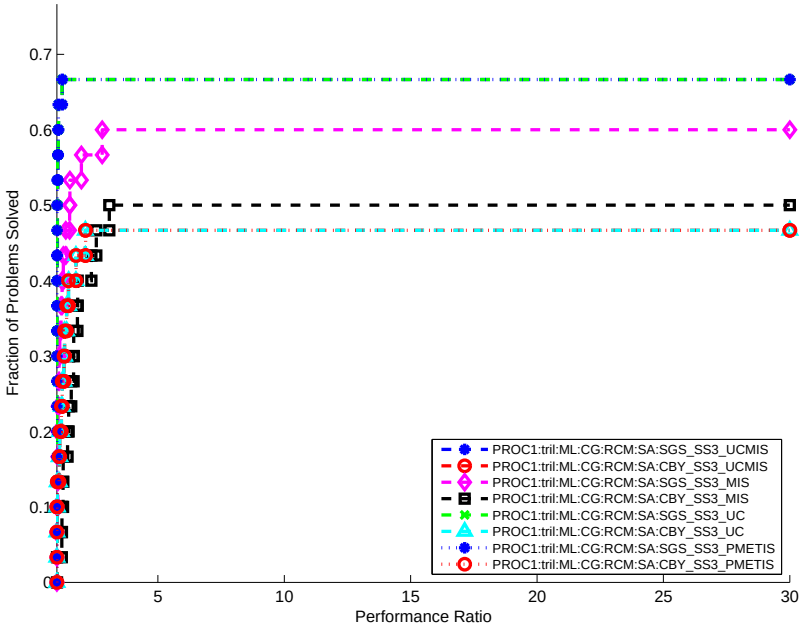


Figure 14: Serial time profile curves for Trilinos ML with RCM ordering, three smoother sweeps, and different smoothers and coarsening schemes. Table 5 contains expansions of the parameter acronyms.

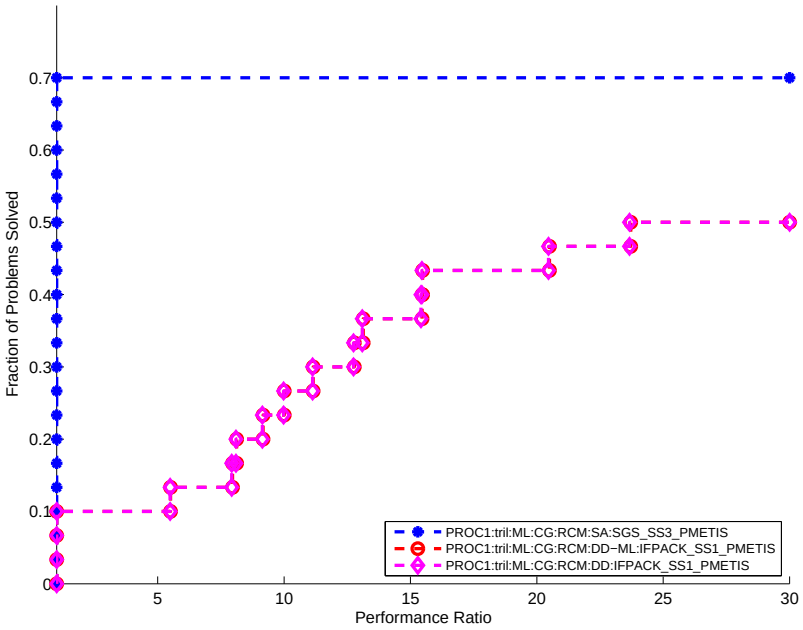
the overall best configurations of Trilinos ML and Hypre BoomerAMG on a single processor. We observe that the memory profiles of Trilinos ML and Hypre BoomerAMG are quite close, although BoomerAMG solves more problems. The time profiles indicate that Hypre BoomerAMG solves a large fraction of problems using much less time than Trilinos ML.

4.1.4 Sparse approximate inverse

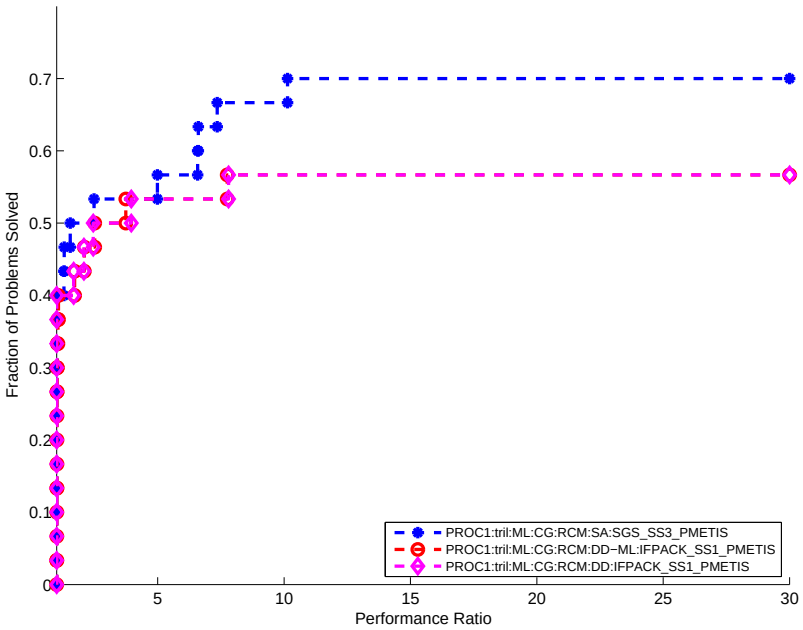
For the ParaSails preconditioner in Hypre, we experimented with multiple threshold values (0, 0.01, 0.1, −0.75, −0.9) and filter values (0, 0.001, 0.05, −0.9) for three different levels as suggested by the user manual [10]. Table 9 summarizes the configurations that resulted in the best performance profiles in our experiments.

Preconditioner	Memory Winner	Time Winner	MTP Winner
Hypre ParaSails	CG, ND, PLev1 Th0, Flt0.05 (2) CG, ND, PLev2 Th0.01, Flt0.001 (32 64) CG, ND, PLev2 Th0.1, Flt0.001 (1 4 8 16)	CG, ND, PLev1 Th0.1, Flt0 (32) CG, ND, PLev2 Th0.1, Flt0.001 (1 2 4 8 16) CG, NONE, PLev2 Th0.1, Flt0 (64)	CG, ND, PLev1 Th0.1, Flt0 (32) CG, ND, PLev1 Th0.1, Flt0.001 (1 2 4 8 16 64)

Table 9: Hypre ParaSails configurations that resulted in the best memory, time, and MTP performance profile area. The numbers enclosed in parentheses denote the number of processors. Table 5 contains expansions of the parameter acronyms.

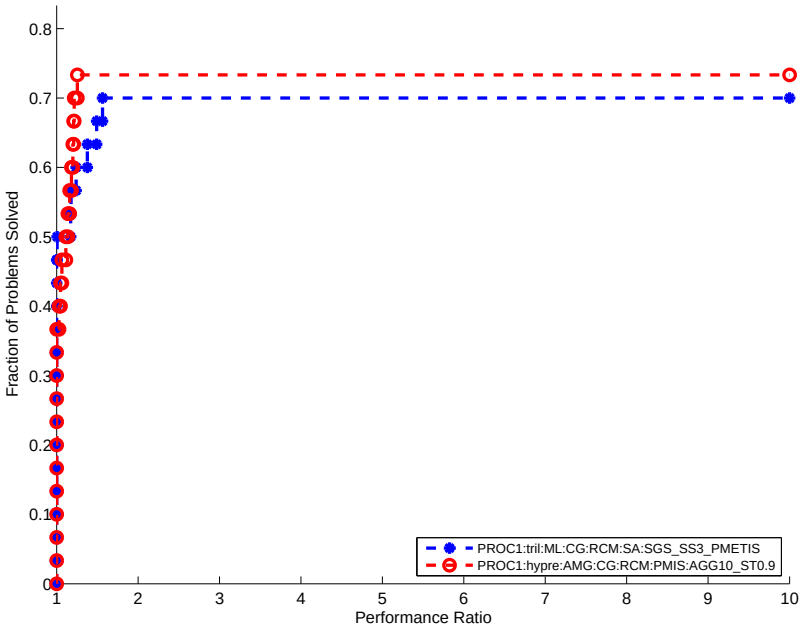


(a) Memory performance profile

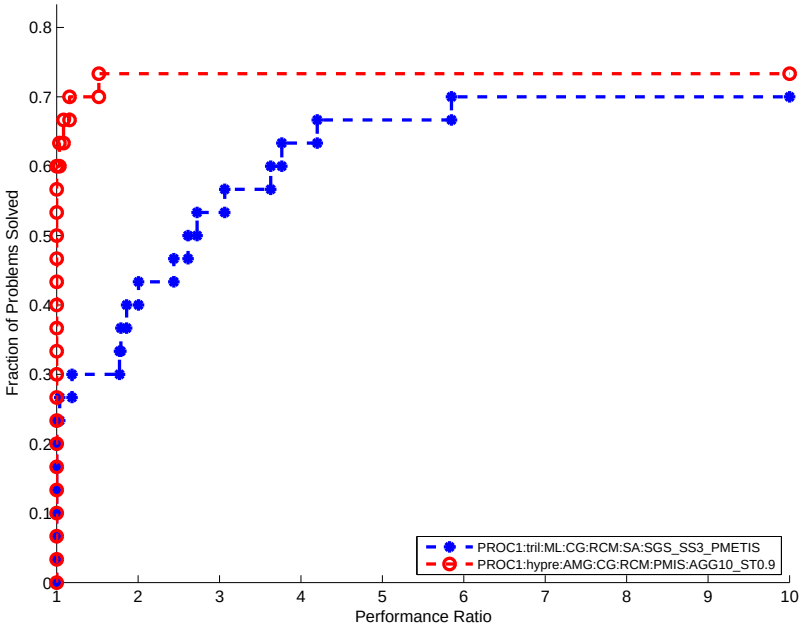


(b) Time performance profile

Figure 15: Serial memory and time profile curves for the best Trilinos ML SA, DD, and DD-ML configurations with RCM ordering. Table 5 contains expansions of the parameter acronyms.

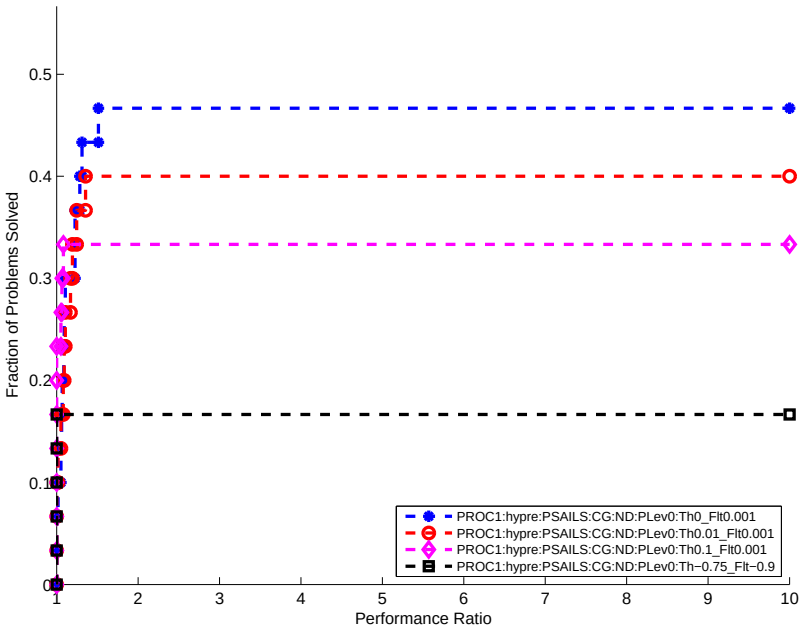


(a) Memory performance profile

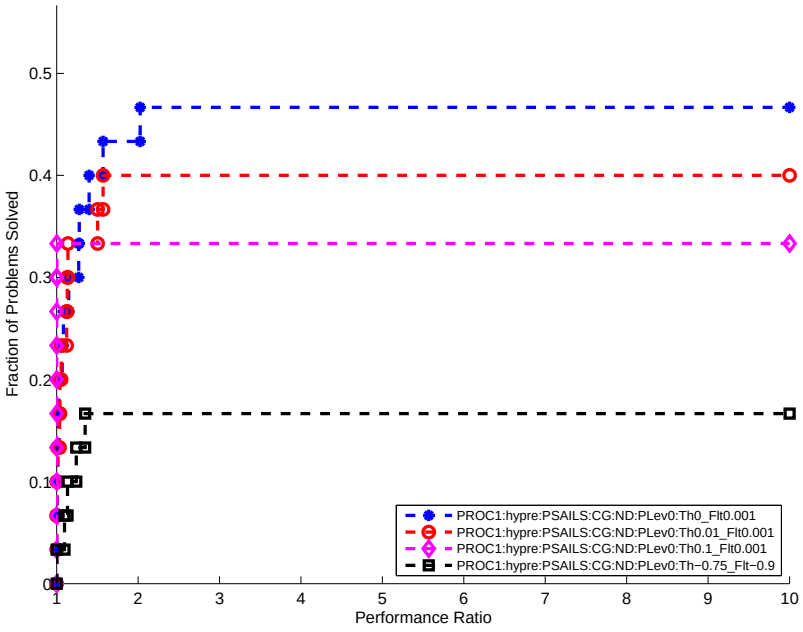


(b) Time performance profile

Figure 16: Serial memory and time profile curves for the overall best Trilinos ML and Hypre BoomerAMG solver configurations. Table 5 contains expansions of the parameter acronyms.

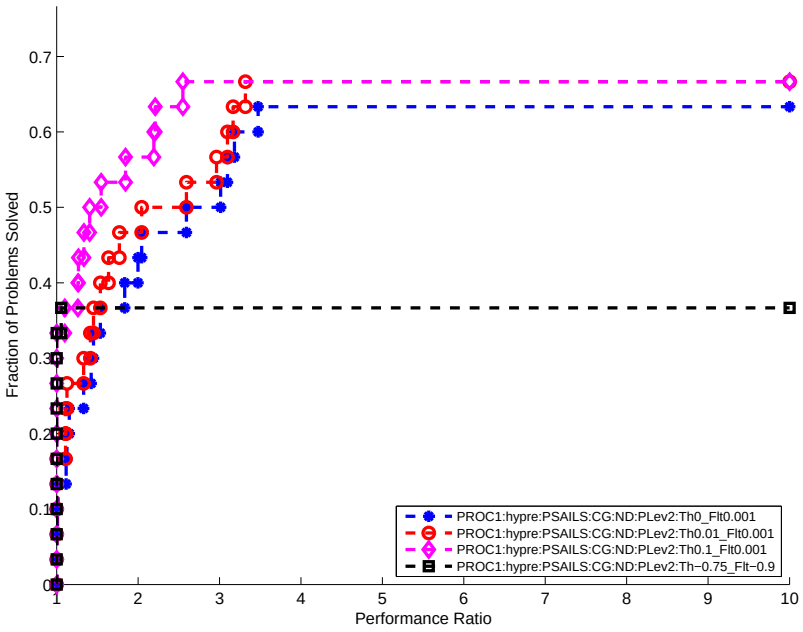


(a) Memory performance profile

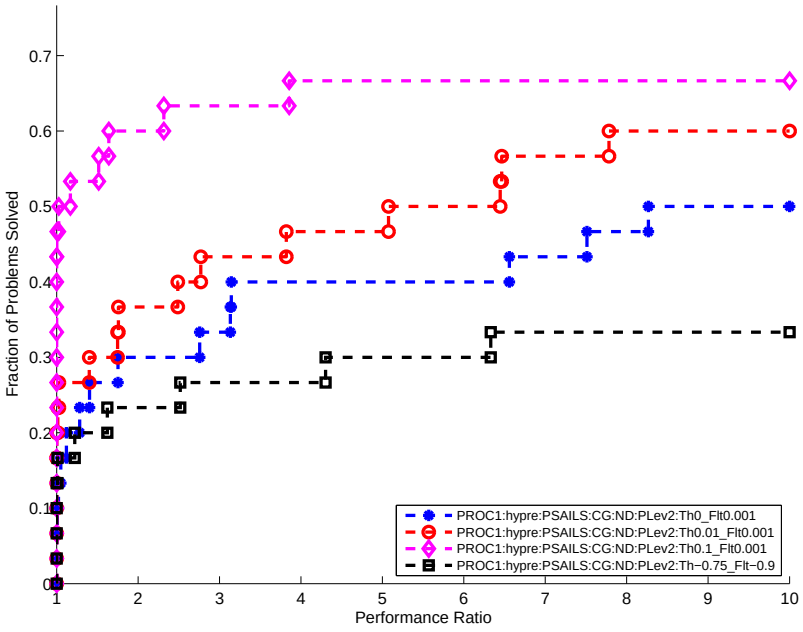


(b) Time performance profile

Figure 17: Serial memory and time performance profile curves for Hypre ParaSails configurations with ND ordering, 0 levels, and various threshold and filter values. Table 5 contains expansions of the parameter acronyms.

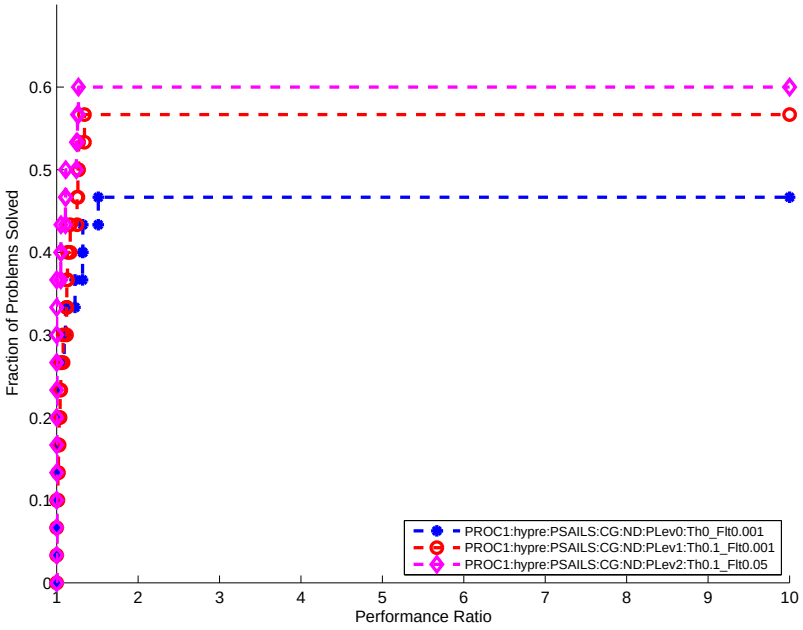


(a) Memory performance profile

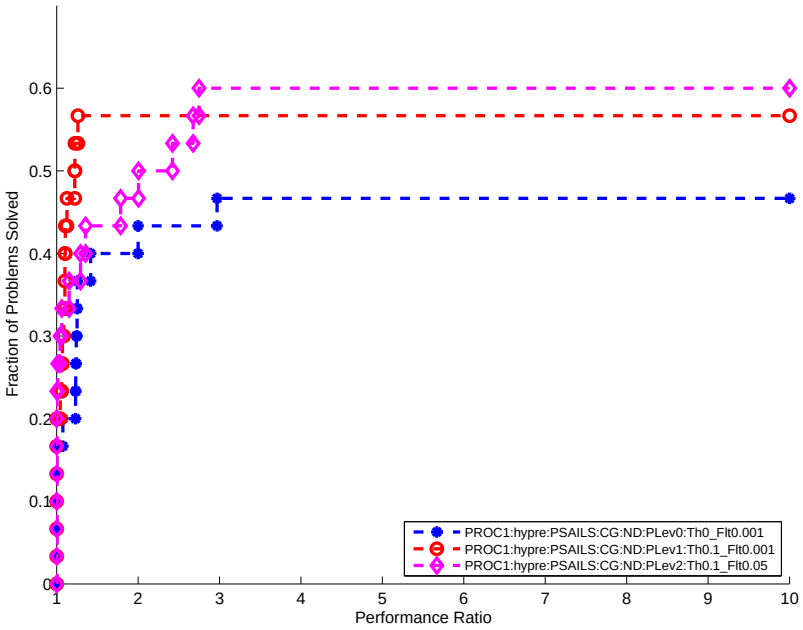


(b) Time performance profile

Figure 18: Serial memory and time performance profile curves for Hypre ParaSails configurations with ND ordering, 2 levels, and various threshold and filter values. Table 5 contains expansions of the parameter acronyms.



(a) Memory performance profile



(b) Time performance profile

Figure 19: Serial memory and time performance profile curves for the best Hypre ParaSails configurations for each level (i.e., 0, 1, and 2). Table 5 contains expansions of the parameter acronyms.

Figures 17 and 18 show the time and memory profiles for different threshold values and the best filter values corresponding to each threshold values with nested dissection ordering for 0 and 2 levels, respectively. When the number of levels is 0, the memory and time requirements of all the configurations are similar. However, their robustness varies a lot and the configuration with threshold 0.0 and filter 0.001 solves the most problems. In the case of 2 levels, the memory and time profile curves for different configurations are well separated and the hreshold value of 0.1 appears to work the best. The negative threshold values suggested by the authors in the user manual, which have a different interpretation from nonnegative values, solved the fewest problems. The performance for the threshold value of -0.9 was much worse than the other threshold values and is not included in Figures 17 and 18.

Figure 19 shows the performance profile curves for the best threshold and filter combination for 0, 1, and 2 levels. We observe that the best configuration for 2 levels solves the maximum number of problems, but is considerably slower than the best configuration for 1 level. An interesting observation is that the memory consumption actually declines with increasing number of levels. This is because the best configurations of higher levels include higher values of threshold and filter to drop more entries.

4.1.5 Variation of overall best configurations with number of processors

From Tables 6–9, we note that the best overall configurations of most preconditioner implementations are different for different number of processes. To determine if these processor-specific overall best configurations are substantially different in their performance, we plotted the MTP performance profile curves for each of these configurations for a fixed number of processors. Between 1 and 64 processors, we observed very small differences between the performance of the 7 sets of best configurations (for 1, 2, 4, 8, 16, 32, and 64 processors). The differences were usually tied to different scalabilities of the preconditioner generation and the solution phases. If there is a considerable difference in the scalability of the two phases, then parameters that shift more computation to the more scalable phase would be favored as the number of processors is increased.

4.2 Relative Performance of Preconditioner Implementations

We now use the MTP metric to compare the performance of all the preconditioner implementations studied in this paper. We compare the various implementation under two scenarios. We first compare the overall best or the experimentally determined default configurations of the preconditioners; i.e., we choose the parameter configuration for each preconditioner that has the best overall performance on our test suite. In the second scenario, we compare the preconditioners based on their PSB (problem specific best) configurations; i.e., we pick the best parameter configuration of each preconditioner for each individual matrix. For both the default and PSB scenarios, we present the results on a single processor and on 64 processors. We also present the results of simultaneously projecting multiple performance metrics, which helps in analyzing the relative memory, time, and robustness of the preconditioners, both in the serial and parallel case.

4.2.1 Overall best or default configurations

The parameter combinations for all package-preconditioner combinations (gathered from Tables 6–9) resulting in the best overall MTP are shown in Tables 10 and 11 for the 1 and 64 processor cases. These solver configurations are good candidates for default values that have a high probability of yielding a small memory-time product for an arbitrary problem.

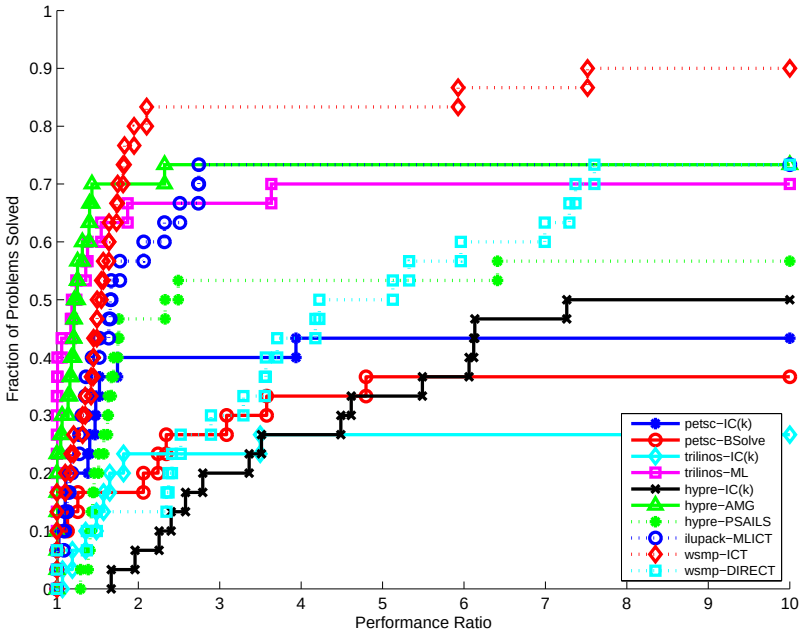


Figure 20: Serial memory profile curves for the direct solver and the overall best MTP configurations of various preconditioner implementations.

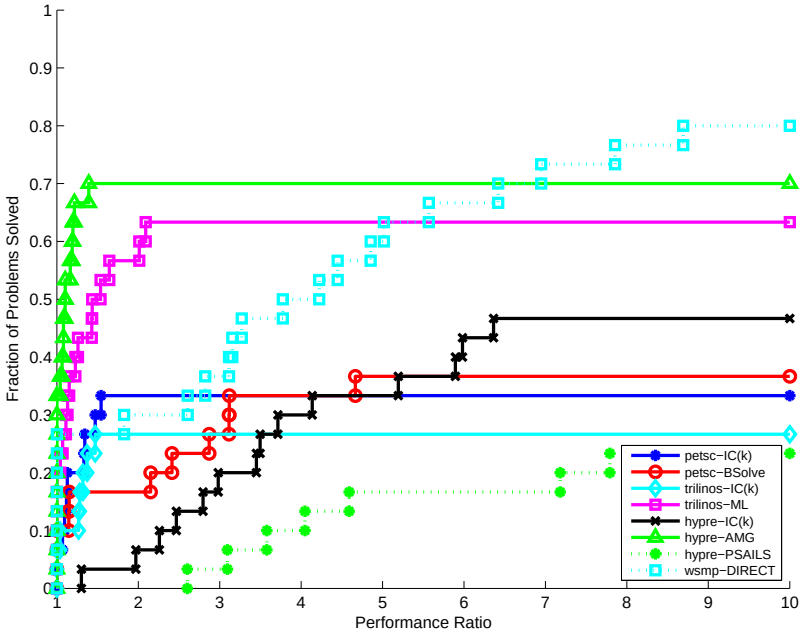


Figure 21: 64-CPU memory profile curves for the direct solver and the overall best MTP configurations of various preconditioner implementations.

Preconditioner	**Ordering**	**Preconditioner Parameters**
Hypre IC(k)	RCM	Level of fill 1
PETSc IC(k)	RCM	Fill factor 1, Level of fill 0
PETSc BlockSolve	RCM	-
Trilinos IC(k)	RCM	Level of fill 8
Ilupack MLICT	RCM	Drop-tolerance 0.03, Inverse norm estimate 75
WSMP ICT	RCM	Drop tolerance 0.003, Fill factor 4.9, SHIFT-ON
Hypre BoomerAMG	RCM	PMIS Coarsening, Aggressive coarsening levels 10, Strong threshold 0.9
Trilinos ML	RCM	Smoothed aggregation, Symmetric Gauss-Seidel smoother, Smoother sweeps 3, ParMETIS Coarsening
Hypre ParaSails	ND	Number of levels 1, Threshold 0.1, Filter 0.001

Table 10: Iterative solver configurations that resulted in the maximum overall memory-time product profile area in the serial case.

Preconditioner	Ordering	**Preconditioner Parameters**
Hypre IC(k)	RCM	Level of fill 1
PETSc IC(k)	RCM	Fill factor 1, Level of fill 0
PETSc BlockSolve	RCM	-
Trilinos IC(k)	RCM	Level of fill (6)
Hypre BoomerAMG	NONE	PMIS Coarsening, Aggresive coarsening levels 10, Strong threshold 0.7
Trilinos ML	NONE	Smoothed aggregation, Symmetric Gauss-Seidel smoother, Smoother sweeps 3, Hybrid Uncoupled-MIS Coarsening
Hypre ParaSails	ND	Number of levels 1, Threshold 0.1 , Filter 0.001

Table 11: Iterative solver configurations that resulted in the maximum overall memory-time product profile area in the 64-processor case.

Figures 20 and 21 show the memory profiles of the configurations shown in Tables 10 and 11, respectively. The memory profile of WSMP direct solver is also included in these figures. The direct solver memory includes the memory for storing the nonzeros in the factors as well as other working memory allocated on the heap during factorization. The iterative solver memory includes the memory needed to store the preconditioner as well as the memory required during the solve phase, including the memory required for restarted GMRES that is sometimes used by the WSMP ICT solver.

For the single processor case in Figure 20, Hypre BoomerAMG, Trilinos ML, WSMP ICT, and ILUPACK MLICT appear to be the most memory efficient and robust. The IC(k) preconditioners are not as robust as the others and their respective curves flatten out fairly early. For the 64 processor case in Figure 21, Hypre BoomerAMG is the most memory efficient followed by Trilinos ML, PETSc IC(k), and Trilinos IC(k). The relative ranking of other preconditioners remains the same except for Hypre ParaSails, which shows a higher memory usage than in the serial case. The high memory usage of Hypre ParaSails is due to the specific implementation choice in which, all the external rows needed by a processor are collected and stored for each processor. While this choice improves the time performance, we observed that its memory consumption increases with the number of processors.

Figures 22 and 23 show the time profiles of the configurations shown in Tables 10 and 11, along with that of the WSMP direct solver. In Figure 22, the direct solver turns out to be the fastest solver for about 60% of the problems in the serial case. This is followed by WSMP ICT, which is

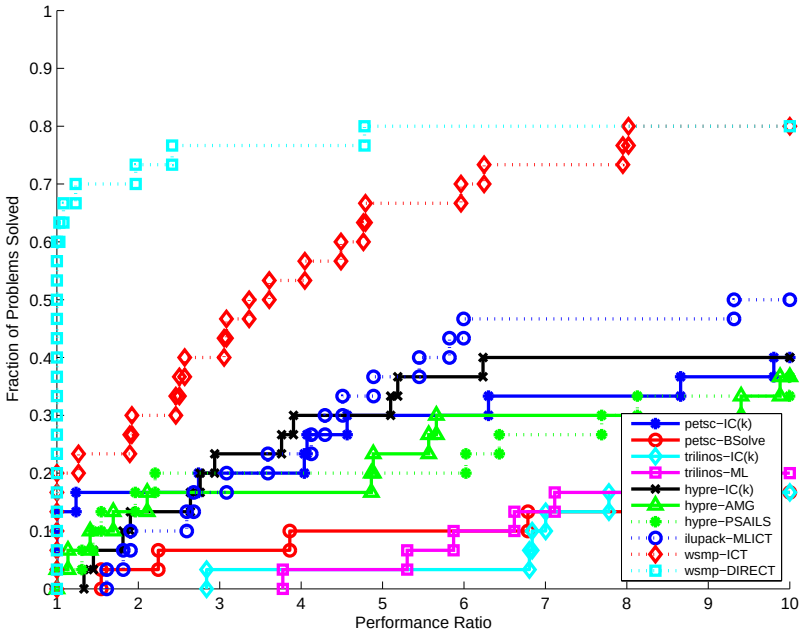


Figure 22: Serial time profile curves for the direct solver and the overall best MTP configurations of various preconditioner implementations.

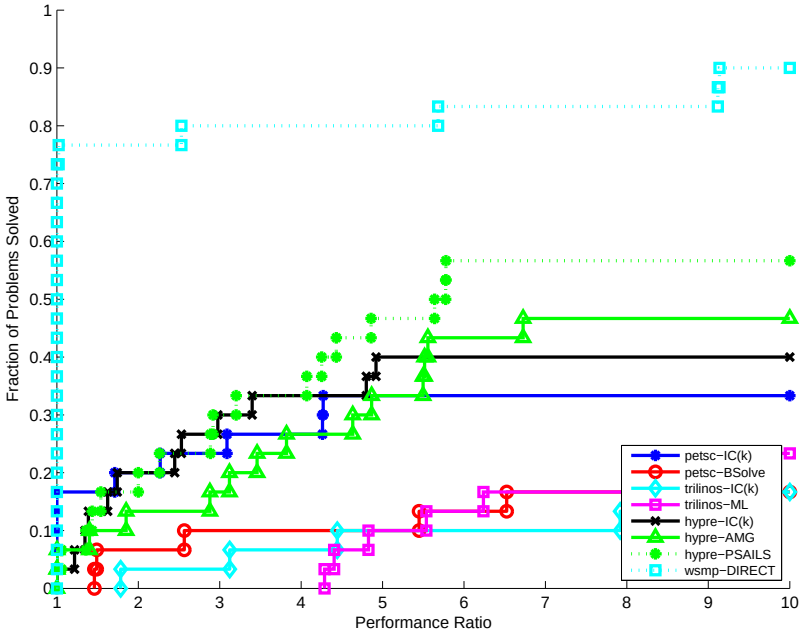


Figure 23: 64-CPU time profile curves for the direct solver and the overall best MTP configurations of various preconditioner implementations.

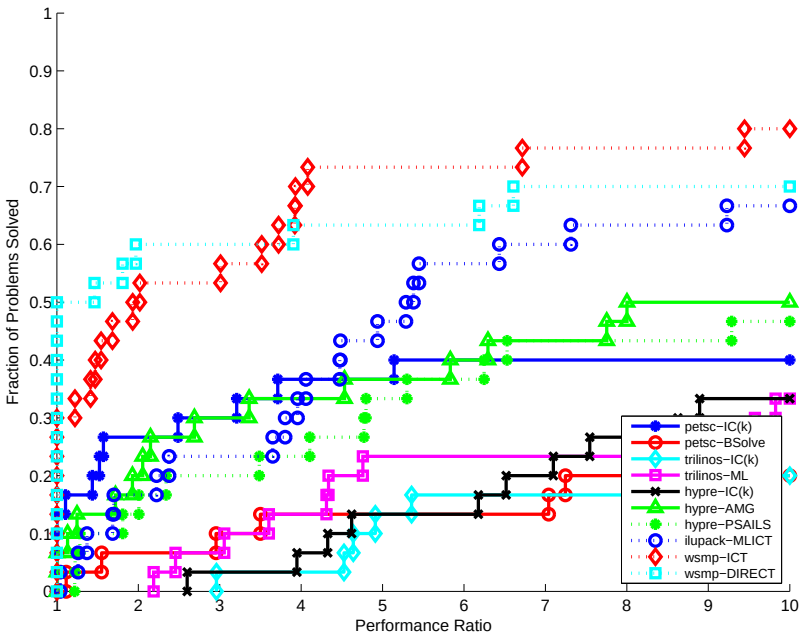


Figure 24: The overall best memory-time profile curves for the direct solver and the various preconditioner implementations on a single processor.

the fastest in about 20% of the cases and PETSc IC(k), which is the fastest for about 13% of the problems. However, understandably, PETSc IC(k) does not do as well for more difficult problems and its time profile curve is soon surpassed by that of other solvers such as ILUPACK and Hypre IC(k). Hypre BoomerAMG and Trilinos ML, which are highly memory efficient, appear to be slower than many other solvers. In the 64 processor case shown in Figure 23, the direct solver is the fastest for about 78% of the problems. Although, many of the iterative solvers seem to show better strong scaling than the direct solver, it is the fastest solver for more matrices on 64 processors than on a single processor because WSMP ICT and ILUPACK and not included in the 64-processor results. The time profile of the direct solver is followed by that of Hypre ParaSails, Hypre IC(k), PETSc IC(k), and Hypre BoomerAMG.

Figures 24 and 25 show the memory-time product profiles of the configurations shown in Tables 10 and 11 along with that of the WSMP direct solver for 1 and 64 processors, respectively. On a single processors, WSMP ICT narrowly outperforms the direct solver for the MTP metric. The relative position of most solvers is about the same on both 1 and 64 processors. In the 64 processor case, the curve for Hypre BoomerAMG moves up because of its excellent memory efficiency and that for Hypre ParaSails curve moves down due to its increasing memory usage in the number of processors.

A comparison of the memory and time performance of the iterative solvers relative to WSMP's direct solver confirms the conventional wisdom that direct solvers are generally fast and robust, but require more memory resources. Conventional wisdom also holds that the preconditioned iterative solvers should outperform the direct solver on larger matrices, particularly those arising in 3-D problems. Our results indicate that, although half of the problems in our test suite have more than half a million unknowns, the average problem size is still too small for most iterative solvers to outperform the direct solver in terms of solution time. Our results also show that among the iterative solvers, WSMP ICT and ILUPACK are among the best performers on a single processor

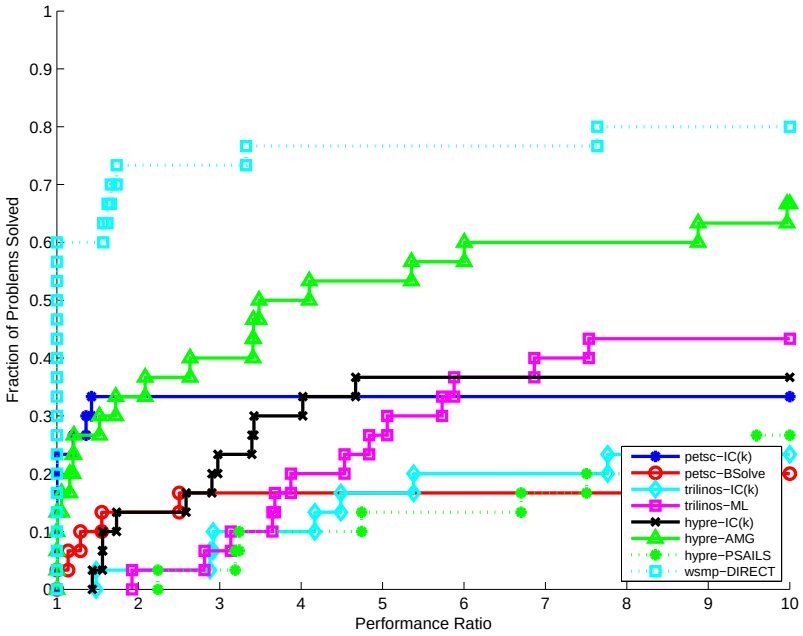


Figure 25: The overall best memory-time profile curves for the direct solver and the various preconditioner implementations on 64 processors.

and Hypre BoomerAMG on 64 processors.

4.2.2 Problem specific best configurations

While the overall best or the default configuration of a preconditioner offers a good choice of parameter settings for an arbitrary problem, users may be able to improve the performance of their applications by tuning the parameters for the matrices arising in their applications. In this section, we discuss the relative performance of various preconditioner implementations when the best parameter configuration is chosen individually for each problem from a reasonably comprehensive set of configurations. This analysis can give a good indication of the best possible performance that a preconditioner is capable of delivering for each problem. While it may not be practical to fine-tune the parameters for each individual problem, fine-tuning can be useful when all matrices arising in a particular application have similar properties.

Figures 26 and 27 show the memory profiles (for 1 and 64 processors, respectively) when the parameter configuration for each problem was chosen individually to minimize its memory-time product. These figures show that problem specific fine-tuning results in significant improvements in memory use for most preconditioners, when compared with the best overall parameter configuration. Compared to Figures 20 and 21, all iterative solver curves move upwards with respect to the direct solver curve in Figures 26 and 27. Besides consuming less memory, most preconditioners are able to solve more problems successfully with problem-specific parameter tuning. The most remarkable improvement with respect to memory occurs for Hypre IC(k), in a large part because of an increase in robustness.

Figures 28 and 29 show the time profiles of all the preconditioners when the parameter configuration for each problem was chosen individually to minimize its memory-time product. Just like the memory profiles, the time profiles of the preconditioners improve significantly when compared to those for the overall best parameter configuration. The most notable improvements in the serial

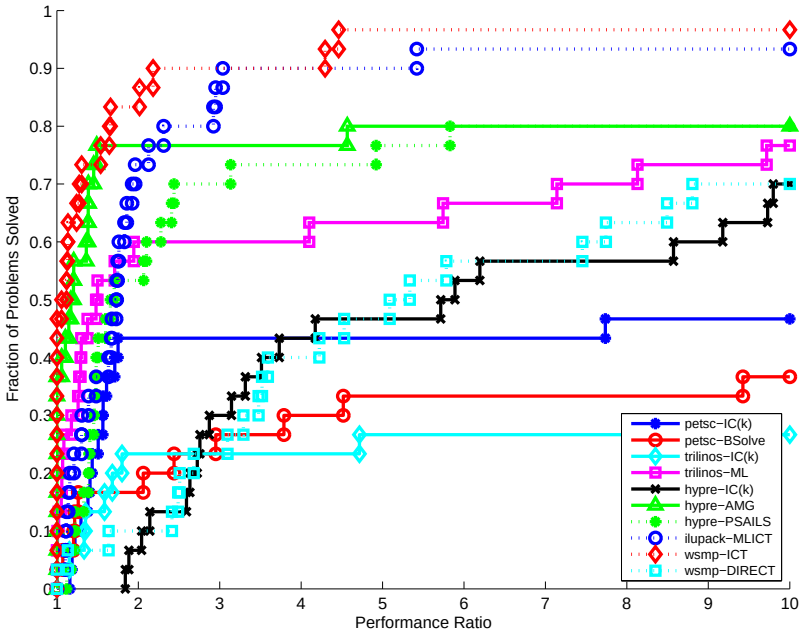


Figure 26: Serial memory profiles for the direct solver and the problem-specific best MTP configurations of the various preconditioner implementations.

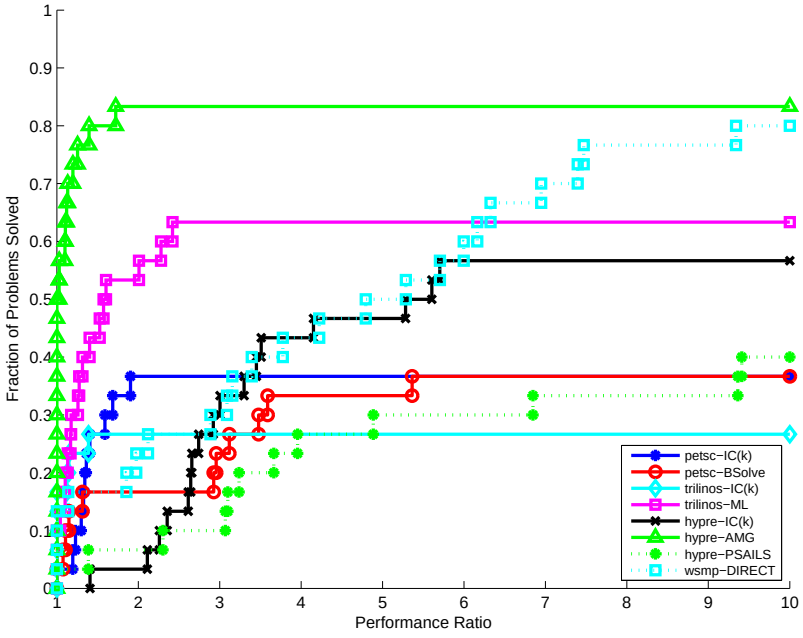


Figure 27: 64-processor memory profiles for the direct solver and the problem-specific best MTP configurations of the various preconditioner implementations.

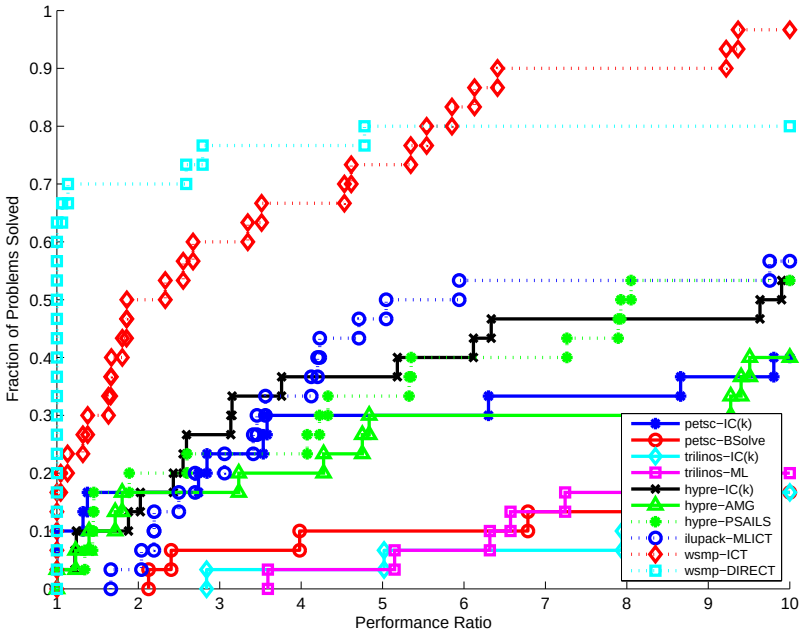


Figure 28: Serial time profiles for the direct solver and the problem-specific best MTP configurations of the various preconditioner implementations.

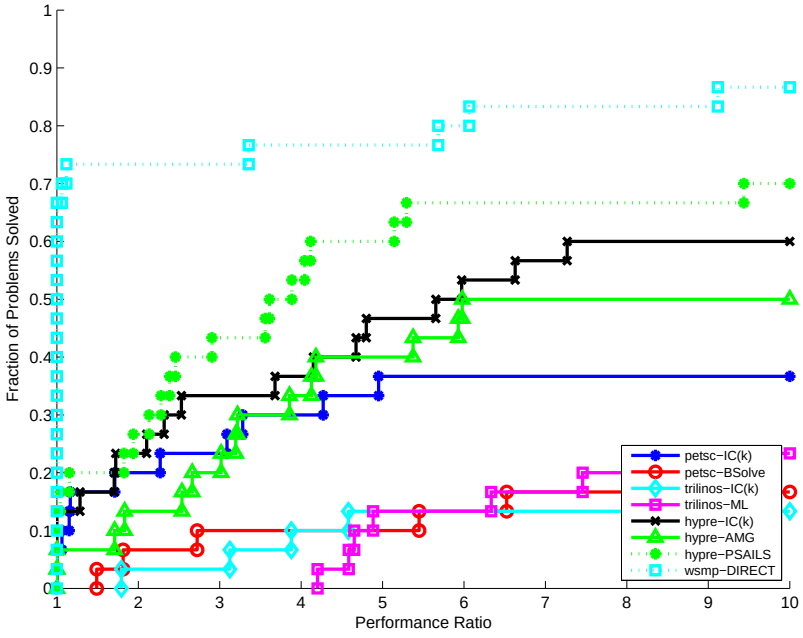


Figure 29: 64-processor time profiles for the direct solver and the problem-specific best MTP configurations of the various preconditioner implementations.

Matrix	Iterative solver configurations with best memory-time product	Iter. Mem	Iter. Time	Dir. Mem	Dir. Time
90153	wsmp, RCM, DT1e-2, F2.5, SHIFT-ON	**55**	**6.61**	194	2.47
af_shell7	wsmp, RCM, DT1e-2, F2.5, SHIFT-OFF	223	62.8	**830**	**11.8**
algor-big	wsmp, RCM, DT1e-2, F4.9, SHIFT-OFF	**788**	**306.**	-	-
audikw_1	wsmp, RCM, DT1e-2, F2.5, SHIFT-ON	**959**	**336.**	9500	870.
bmwcra_1	wsmp, RCM, DT1e-2, F2.5, SHIFT-ON	**110**	**51.1**	568	11.3
ctu-1	wsmp, ND, DT3e-4, F2.5, SHIFT-OFF	2830	532.	**3210**	**86.8**
ctu-2	wsmp, ND, DT3e-3, F4.1, SHIFT-ON	936	463.	**2350**	**100.**
cfd1	petsc IC(k), RCM, LF0, F1	**29**	**6.59**	157	2.41
cfd2	ParaSails, NONE, PLev1, Th.1, Flt.05	80	26.8	**310**	**6.35**
conti20	wsmp, RCM, DT3e-3, F3.3, SHIFT-ON	36	5.63	**64**	**0.96**
garybig	BoomerAMG, RCM, FALG, AGG10, ST.7	**6660**	**1.2e4**	-	-
G3_circuit	wsmp, RCM, DT1e-2, F2.5, SHIFT-OFF	**179**	**37.4**	956	20.1
hood	wsmp, ND, DT3e-3, F4.1, SHIFT-OFF	170	4.10	**257**	**2.51**
inline_1	wsmp, RCM, DT3e-4, F2.5, SHIFT-OFF	1910	91.8	**1500**	**27.4**
kyushu	petsc IC(k), RCM, LF0, F1	**417**	**32.7**	9220	1.3e3
ldoor	wsmp, ND, DT3e-4, F4.9, SHIFT-OFF	965	40.5	**1370**	**22.5**
msdoor	wsmp, ND, DT3e-4, F4.9, SHIFT-OFF	499	28.4	**492**	**5.12**
mstamp-2c	ParaSails, RCM, PLev0, Th.1, Flt0	**726**	**76.4**	-	-
nastran-b	wsmp, RCM, DT1e-2, F4.1, SHIFT-OFF	**1160**	**474.**	8620	539.
nd24k	wsmp, RCM, DT1e-2, F2.5, SHIFT-ON	**88**	**23.6**	2750	412.
oilpan	wsmp, ND, DT3e-4, F2.5, SHIFT-OFF	60	2.38	**95**	**1.02**
pbolic_fem	wsmp, RCM, DT1e-2, F2.5, SHIFT-ON	**71**	**7.39**	233	2.90
pga-rem1	wsmp, RCM, DT1e-2, F4.9, SHIFT-OFF	277	39.0	**742**	**11.1**
pga-rem2	wsmp, RCM, DT1e-2, F2.5, SHIFT-OFF	**437**	**74.2**	1980	44.6
qa8fk	BoomerAMG, RCM, PMIS, AGG10, ST.25	**18**	**2.03**	193	4.60
qa8fm	wsmp, RCM, DT1e-2, F2.5, SHIFT-OFF	**3**	**0.16**	187	4.21
ship_003	wsmp, RCM, DT1e-2, F2.5, SHIFT-ON	**62**	**17.7**	529	16.9
shipsec5	wsmp, RCM, DT1e-2, F3.3, SHIFT-ON	**88**	**12.2**	448	12.9
thermal2	wsmp, RCM, DT1e-2, F2.5, SHIFT-OFF	139	59.4	**484**	**6.44**
torso	wsmp, RCM, DT1e-2, F2.5, SHIFT-ON	**48**	**5.79**	677	24.4

Table 12: Serial time (seconds) and memory (Megabytes) of the direct solver and the problem-specific best MTP configurations of the various preconditioners. The bold values indicate the solver with the smallest MTP. Table 5 contains the expansions of the parameter acronyms.

case are for ILUPACK MLICT, Hypre IC(k) and Hypre ParaSails. While using 64 processors, a comparison of Figures 23 and 29 shows that Hypre's BoomerAMG, IC(k), and ParaSails reduce the time performance gap with the WSMP direct solver.

Table 12 shows the best iterative solver configuration (in terms of MTP) and its time and memory consumption for each problem for each problem in the serial case. The time and memory of the direct solver are also included for comparison. The values corresponding to the best MTP are in bold font. The direct solver has the best MTP for 12 of the 30 matrices. As expected, the iterative solvers do much better for large 3-D problems. Among the iterative solvers, WSMP ICT does best for 13 problems, PETSc IC(k) and Hypre BoomerAMG for 2 problems each, and Hypre ParaSails for one problem. In Table 13, we show the same data excluding WSMP ICT and ILUPACK for the single CPU case. At the time of preparing this report, WSMP ICT and ILUPACK did not have scalable parallel implementations for comparison with the 64-processor case shown in Table 14. From Tables 13 and 14, we see that the direct solver does best for 17 problems

Matrix	Iterative solver configurations with best memory-time product	Iter. Mem	Iter. Time	Dir. Mem	Dir. Time
90153	ParaSails, RCM, PLev1, Th.1, Flt.05	80	19.6	**194**	**2.47**
af_shell7	BoomerAMG, ND, PMIS, AGG10, ST.9	237	111.	**830**	**11.8**
algor-big	BoomerAMG, NONE, HMIS, AGG10, ST.9	**150**	**427.**	-	-
audikw_1	trilinos ML, NONE, SA, SGS, SS2, UCMIS	**772**	**5.9e3**	9500	870.
bmwcra_1	BoomerAMG, ND, PMIS, AGG10, ST.25	98	292.	**568**	**11.2**
ctu-1	-	-	-	**3210**	**86.8**
ctu-2	trilinos ML, RCM, SA, IFPACK, SS1, UC	13400	5.8e3	**2350**	**100.**
cfd1	petsc IC(k), ND, LF0, F1	**29**	**6.59**	157	2.41
cfd2	ParaSails, NONE, PLev1, Th.1, Flt.05	80	26.8	**310**	**6.35**
conti20	trilinos ML, NONE, SA, SGS, SS2, UCMIS	18	126.	**64**	**0.96**
garybig	BoomerAMG, ND, FALG, AGG10, ST.7	**6660**	**1.2e4**	-	-
G3_circuit	petsc IC(k), ND, LF1, F10	**208**	**57.2**	956	20.1
hood	ParaSails, ND, PLev0, Th.1, Flt0	143	18.2	**257**	**2.51**
inline_1	trilinos ML, NONE, SA, IFPACK, SS1, UC	6160	757.	**1500**	**27.4**
kyushu	petsc IC(k), ND, LF0, F1	**417**	**32.7**	9220	1.3e3
ldoor	ParaSails, RCM, PLev0, Th.1, Flt.05	538	181.	**1370**	**22.5**
msdoor	ParaSails, NONE, PLev1, Th0, Flt.05	353	215.	**492**	**5.12**
mstamp-2c	ParaSails, ND, PLev0, Th.1, Flt0	**726**	**76.4**	-	-
nastran-b	BoomerAMG, ND, HMIS, AGG10, ST.7	1280	4.5e3	**8620**	**539.**
nd24k	ParaSails, NONE, PLev2, Th.01, Flt.05	**277**	**61.3**	2750	412.
oilpan	ParaSails, NONE, PLev0, Th0, Flt.05	59	16.8	**95**	**1.02**
pbolic_fem	BoomerAMG, ND, HMIS, AGG10, ST.7	75	9.36	**233**	**2.90**
pga-rem1	hypre IC(k), ND, LF1, NzINF	521	136.	**742**	**11.1**
pga-rem2	petsc IC(k), ND, LF0, F1	535	281.	**1980**	**44.6**
qa8fk	BoomerAMG, ND, PMIS, AGG10, ST.25	**18**	**2.03**	193	4.60
qa8fm	petsc IC(k), ND, LF0, F1	**27**	**0.12**	187	4.21
ship_003	ParaSails, RCM, PLev2, Th.1, Flt.05	**62**	**17.7**	529	16.9
shipsec5	ParaSails, RCM, PLev1, Th.1, Flt.05	**130**	**17.6**	448	12.9
thermal2	BoomerAMG, ND, HMIS, AGG10, ST.25	160	59.7	**484**	**6.44**
torso	petsc IC(k), ND, LF0, F1	**54**	**5.11**	677	24.4

Table 13: Serial time (seconds) and memory (Megabytes) of the direct solver and the problem-specific best MTP configurations of the various preconditioners excluding ILUPACK MLICT and WSMP ICT. The bold values indicate the solver with the smallest MTP. Table 5 contains the expansions of the parameter acronyms.

Matrix	Iterative solver configurations with best memory-time product	Iter. Mem	Iter. Time	Dir. Mem	Dir. Time
90153	hypre IC(k), ND, LF4, NzINF	352	0.84	**195**	**0.13**
af_shell7	BoomerAMG, RCM, PMIS, AGG10, ST.9	240	1.88	**848**	**0.45**
algor-big	BoomerAMG, NONE, PMIS, AGG10, ST.7	**987**	**9.95**	32200	90.4
audikw_1	BoomerAMG, RCM, FALG, AGG0, ST.9	**1540**	**86.6**	9750	26.9
bmwcra_1	BoomerAMG, RCM, PMIS, AGG10, ST.25	100	6.70	**572**	**0.42**
ctu-1	-	-	-	**3460**	**4.64**
ctu-2	-	-	-	**2320**	**2.51**
cfd1	petsc IC(k), RCM, LF0, F1	**29**	**0.28**	164	0.16
cfd2	BoomerAMG, NONE, PMIS, AGG10, ST.7	44	5.30	**306**	**0.29**
conti20	-	-	-	**67**	**0.08**
garybig	BoomerAMG, ND, FALG, AGG10, ST.7	**6680**	**264.**	-	-
G3_circuit	petsc IC(k), RCM, LF0, F1	**187**	**2.12**	939	0.69
hood	Blocksolve95, RCM, ALL, NONE	116	0.95	**303**	**0.15**
inline_1	-	-	-	**1530**	**1.33**
kyushu	petsc IC(k), RCM, LF0, F1	**409**	**2.68**	9220	34.5
ldoor	ParaSails, ND, PLev0, Th0, Flt.05	1760	2.99	**1510**	**0.83**
msdoor	ParaSails, ND, PLev1, Th0, Flt.05	1520	4.08	**558**	**0.27**
mstamp-2c	petsc IC(k), ND, LF0, F1	**1000**	**2.01**	15900	62.8
nastran-b	BoomerAMG, RCM, PMIS, AGG10, ST.9	**1350**	**79.3**	9130	19.2
nd24k	ParaSails, NONE, PLev2, Th.1, Flt.05	**1450**	**1.76**	2680	10.7
oilpan	BoomerAMG, RCM, HMIS, AGG0, ST.9	49	6.09	**103**	**0.06**
pbolic_fem	BoomerAMG, NONE, HMIS, AGG10, ST.25	**76.6**	**0.41**	235	0.16
pga-rem1	hypre IC(k), RCM, LF1, NzINF	525	2.36	**736**	**0.49**
pga-rem2	petsc IC(k), RCM, LF0, F1	**535**	**5.09**	2020	2.24
qa8fk	BoomerAMG, NONE, HMIS, AGG10, ST.25	**19.9**	**0.16**	186	0.20
qa8fm	BoomerAMG, RCM, PMIS, AGG0, ST.25	**17**	**0.03**	185	0.20
ship_003	BoomerAMG, RCM, HMIS, AGG10, ST.25	91	7.91	**560**	**0.76**
shipsec5	Blocksolve95, RCM, ALL, NONE	**113**	**1.31**	505	0.51
thermal2	BoomerAMG, ND, PMIS, AGG10, ST.25	159	1.37	**492**	**0.36**
torso	petsc IC(k), RCM, LF0, F1	**54**	**0.16**	685	0.91

Table 14: 64-CPU time (seconds) and memory (Megabytes) of the direct solver and the problem-specific best MTP configurations of the various preconditioners excluding ILUPACK MLICT and WSMP ICT. The bold values indicate the solver with the smallest MTP. Table 5 contains the expansions of the parameter acronyms.

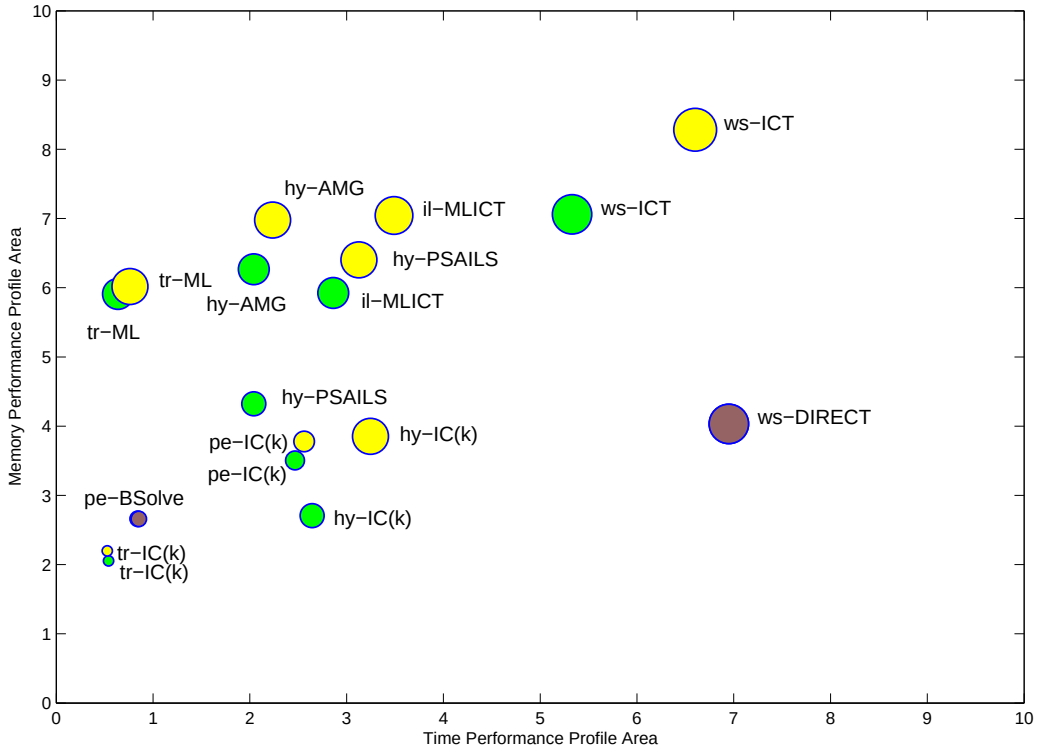


Figure 30: Plot of the time profile area versus the memory profile area for the direct solver and various preconditioner implementations on a single processor. Each circle represents a preconditioner whose name consists of the first two letters of the name of the package followed by the type of preconditioner. The size of a circle is proportional to the number of problems solved. The green (dark) circles correspond to profile areas for the default/overall best parameter configuration and the yellow (light) ones correspond to profile areas for problem-specific best parameters. Overlapping yellow and green circles are shown as a brown circle.

in the serial case and for 15 problems in the 64-processor case. Among the iterative solvers, Hypre BoomerAMG does best for 3 problems in the serial case and for 7 problems on 64 processors, Hypre ParaSails does best for 4 problems in the serial case and for 1 problem on 64 processors, PETSc IC(k) has the smallest MTP for 5 problems in the serial case and for 6 problems on 64 processors, Trilinos ML does best for 1 problem on a single processor and for none on 64 processors, and PETSc BlockSolve has the smallest MTP for 1 problem on 64 processors and for none in the serial case.

4.2.3 Relative performance and sensitivity to parameter tuning

We have observed that different preconditioners and solvers have different strengths and weaknesses. Some are more memory efficient than others, while some are faster than others. They also have different degrees of robustness. In Section 4.2.2, we also saw that, as expected, most preconditioners performed significantly better when their parameters were permitted to be tuned to each coefficient matrix. However, different preconditioners displayed different degrees of improvement. Figure 30 displays all this relative information about the performance of various preconditioners by means of

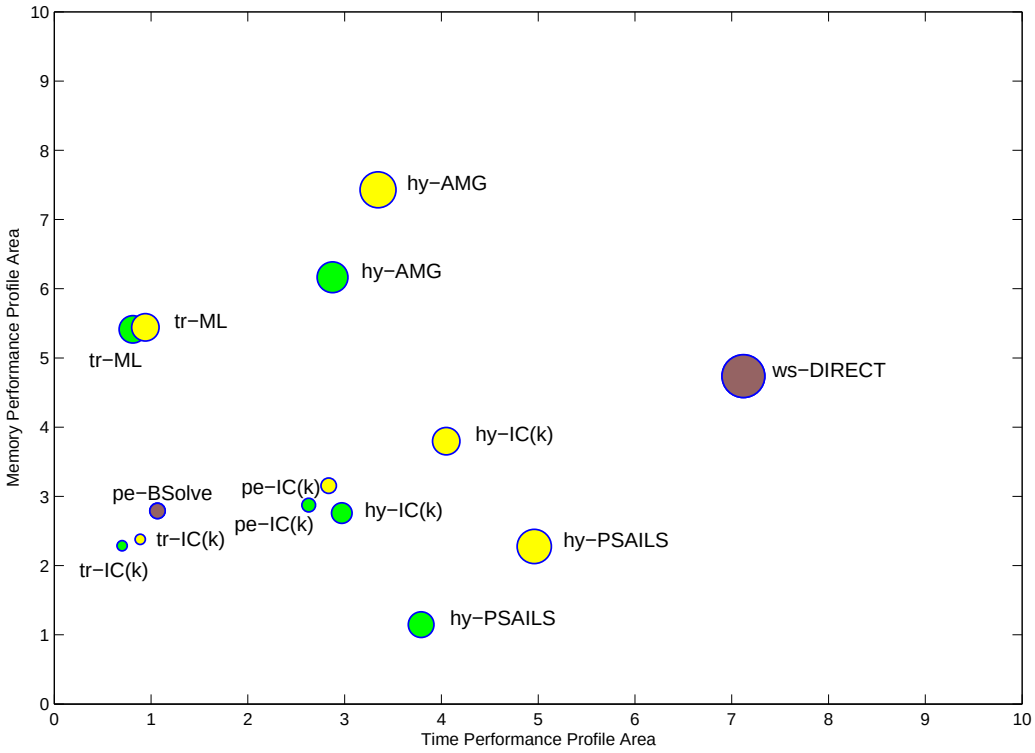


Figure 31: Plot of the time profile area versus the memory profile area for the direct solver and various preconditioner implementations on a single processor. Each circle represents a preconditioner whose name consists of the first two letters of the name of the package followed by the type of preconditioner. The size of a circle is proportional to the number of problems solved. The green (dark) circles correspond to profile areas for the default/overall best parameter configuration and the yellow (light) ones correspond to profile areas for problem-specific best parameters. Overlapping yellow and green circles are shown as a brown circle.

a single information-rich graphic for the uniprocessor case. The figure has two sets of circles for each preconditioner. The green (dark) circles correspond to the default parameter configurations. The yellow (light) circles correspond to the corresponding problem-specific best parameters. The x- and y-coordinates of each circle are the areas under the time and memory profile curves of the corresponding preconditioner derived from plotting the default and problem specific performance profiles in a single plot. The size of each circle is proportional to the number of problems solved.

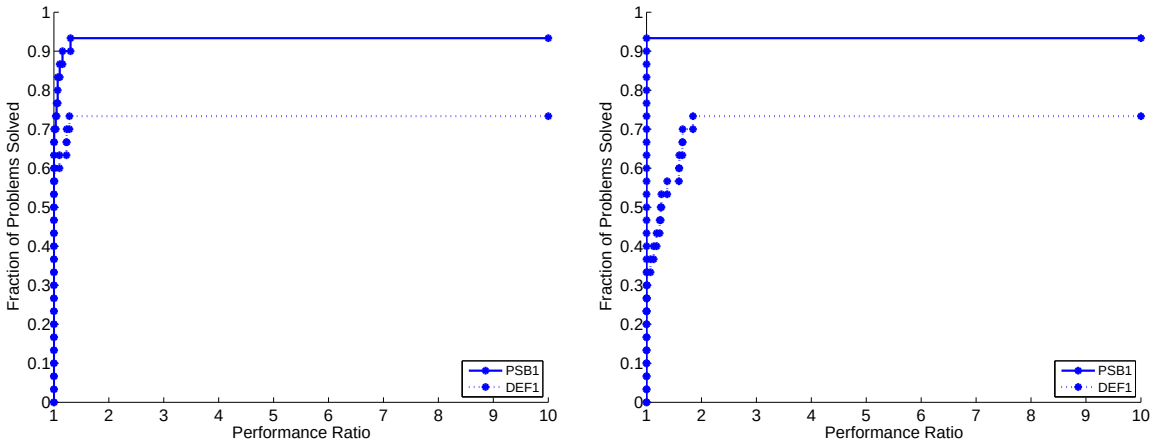
The height of a circle in Figure 30 is indicative of the memory efficiency of the corresponding preconditioner. Similarly, the distance from the y-axis towards the right is indicative of its speed. The figure shows at a glance which solvers and preconditioners are most memory efficient and which ones are most time efficient. For example, Figure 30 shows that the direct solver is very fast and robust, but is less memory efficient than many of the preconditioners. On the other hand, Hypre BoomerAMG is very robust and memory efficient, but is relatively slow. WSMP ICT with default parameters is fairly memory efficient and significantly faster. Some preconditioners benefit a great deal from parameter tuning. This is evident from the fact the yellow (light) circles corresponding to most preconditioners lie above and to the right of their dark counterparts. Hypre IC(k) and ParaSails, ILUPACK MLICT, and WSMP ICT show remarkable improvements in both time and memory. On the other hand, the performance of the default and tuned configurations of Trilinos ML appear to be quite close. Another interesting observation from Figure 30 is that the time, memory, and robustness of different implementations of the same underlying preconditioning method can be very different. Whether with default or with fine-tuned parameters, the best preconditioner implementations lie on the periphery of the plot. When looking for candidates for the most suitable package-preconditioner combinations for an application, a user is likely to fare best by picking one of these depending on the desired balance between computation time and memory.

In Figure 31, we compare the relative performance of the preconditioners in the 64 processor case. Just like the serial case, the direct solver is very fast but memory intensive in comparison to other preconditioners. Hypre BoomerAMG is relatively more efficient with respect to both time and memory and the benefits of fine-tuning are also somewhat more pronounced than in 30. Hypre IC(k) outperforms PETSc IC(k) and Trilinos IC(k) with respect to both memory and time. The relative time efficiency of Hypre ParaSails increases, but it comes at the expense of increased memory usage. To summarize, on 64 processors, Hypre BoomerAMG is highly memory efficient, WSMP direct and Hypre ParaSails are relatively fast whereas Hypre IC(k) seems to balance both time and memory.

We now compare the performance of the overall best or default (DEF) configuration of each preconditioner-package combination with that of its problem-specific best (PSB) configuration. These two sets of performance values are compared for 1, 2, 4, 8, 16, 32, and 64 processors for the solvers whose parallel implementations are available. The difference between the DEF and PSB profiles and their areas are indicative of the degree to which a particular solver benefits from problem specific fine tuning.

Figure 32 shows the memory and time performance profile curves for the overall best and problem-specific best configurations for WSMP ICT and ILUPACK MLICT. These preconditioners currently have only a serial implementation available, so only single processor results are shown. Figure 32 shows that both solvers show considerable improvement in performance due to problem-specific best parameter selection. ILUPACK MLICT shows more improvement in robustness due to fine-tuning, which WSMP ICT shows more improvement in the time and memory used.

Figures 33–37 show the performance variation between the PSB and overall best for all the preconditioners with parallel implementations for 1, 2, 4, 8, 16, 32, and 64 processors. Instead of showing separate performance profile curves for memory and time separately for each processor setting, we combine the information contained in separate memory and time plots in a single figure



(a) ILUPACK MLICT Memory Profiles

(b) ILUPACK MLICT Time Profiles

(c) WSMP ICT Memory Profiles

(d) WSMP ICT Time Profiles

Figure 32: Serial memory and time profile curves for the overall best (DEF) and the problem-specific best (PSB) configurations of ILUPACK MLICT and WSMP ICT.

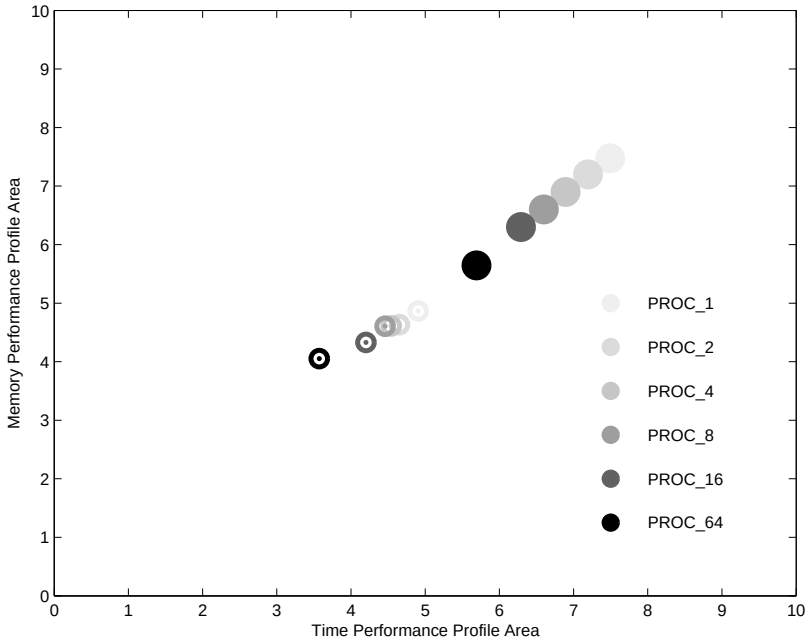


Figure 33: Memory and time profile areas for the overall best (empty circles) and the problem-specific best configurations (filled circles) of Hypre IC(k) preconditioner for multiple processors.

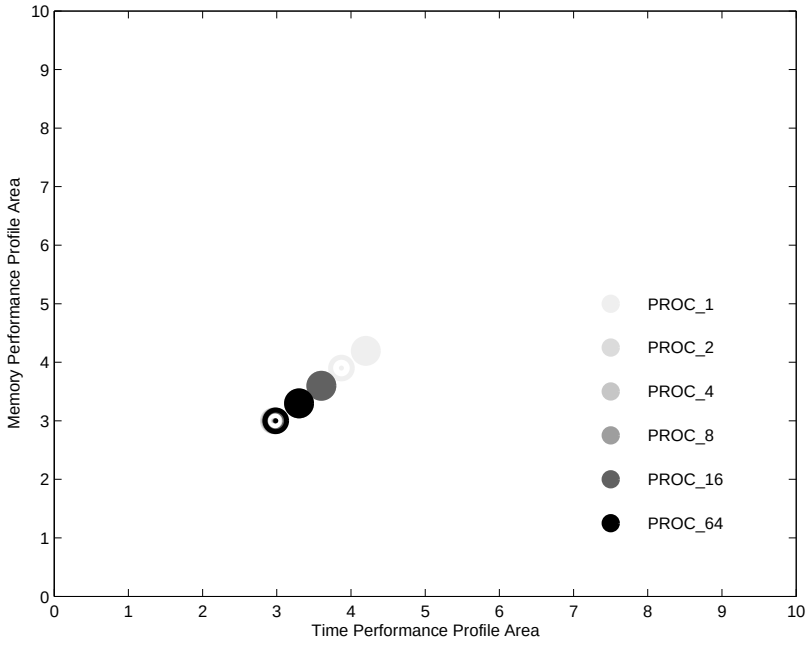


Figure 34: Memory and time profile areas for the overall best (empty circles) and the problem-specific best (filled circles) configurations of PETSc IC(k) for multiple processors.

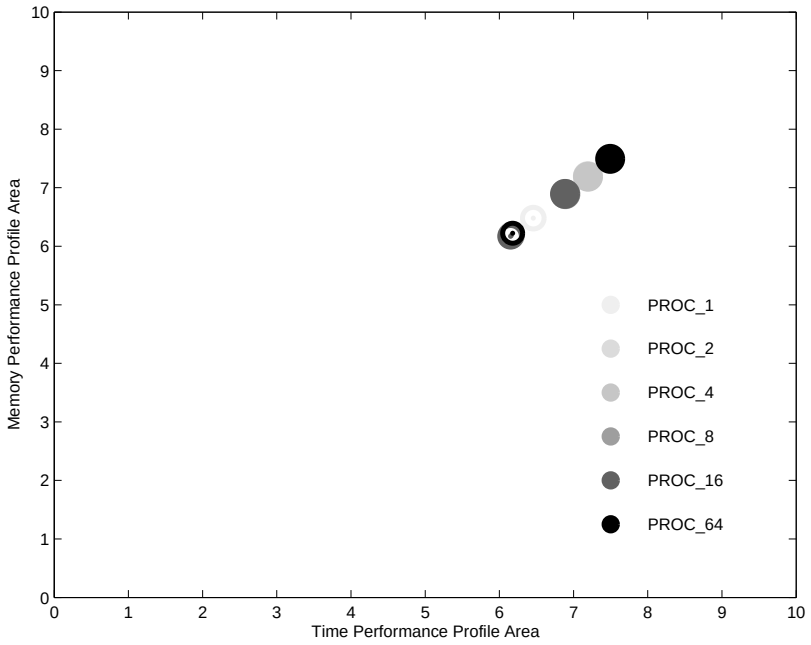


Figure 35: Memory and time profile areas for the overall best (empty circles) and the problem-specific best configurations (filled circles) of Hypre BoomerAMG preconditioner for multiple processors.

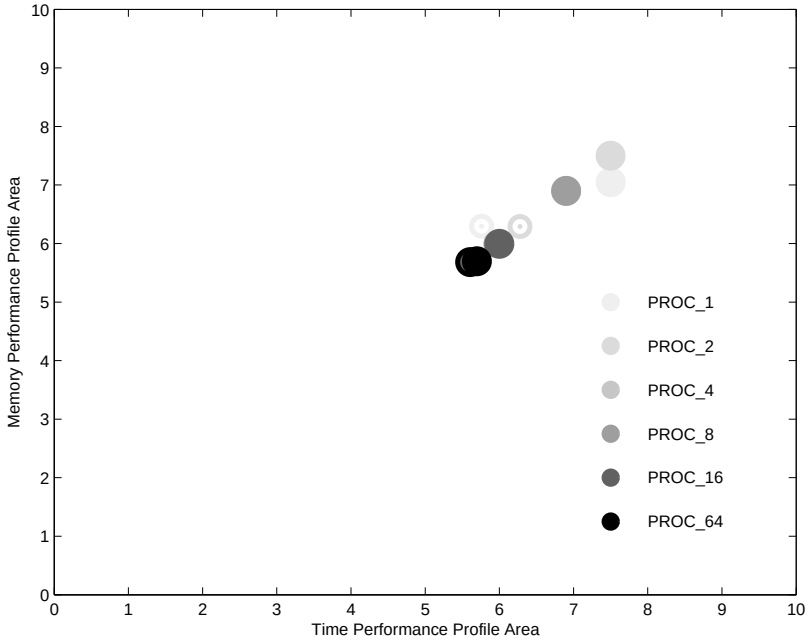


Figure 36: Memory and time profile areas for the overall best (empty circles) and the problem-specific best (filled circles) configurations of Trilinos ML for multiple processors.

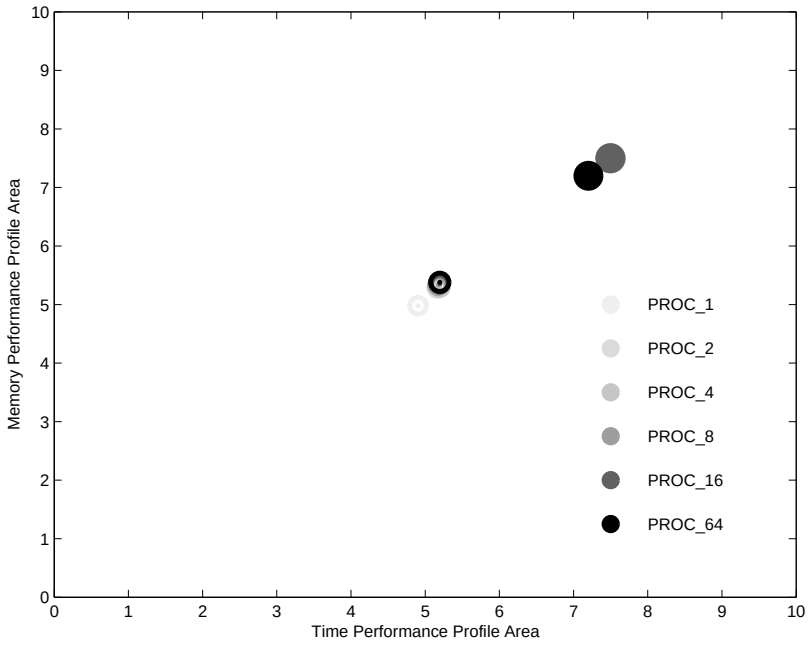


Figure 37: Memory and time profile areas for the overall best (empty circles) and the problem-specific best (filled circles) configurations of Hypre ParaSails for multiple processors.

plotting the AUC values (area under the PP curve) for time and memory. Each figure has two circles for each processor configuration. The empty circles correspond to the overall best parameter configurations and the filled circles correspond to the problem-specific best performance. The x- and y-coordinates of each circle are respectively the areas of time and memory profiles obtained by considering the overall best and PSB performance of a single configuration group together. The size of each circle is proportional to the number of problems solved. These figures also show the AUC trends for the different package-preconditioner combinations as the number of processors is increased from 1 to 64.

In our experiments, the PSB and overall best profile areas for PETSc BlockSolve95 and Trilinos IC(k) were nearly identical and are, therefore, not shown. However, in the case of other preconditioner implementations, we observe that varying degrees of performance benefit can be obtained from fine-tuning, as indicated by the separation between the overall best and PSB circles. Figure 33 indicates that for Hypre IC(k), problem specific tuning results in considerable performance improvement. The figure also shows that the performance of Hypre IC(k) relative to other parallel preconditioner implementations declines as the number of processors is increased. Figure 34 shows that the gap between the empty and filled circles is much smaller for PETSc IC(k), although its performance relative to other preconditioners also declines as the number of processors are increased. Figure 35 shows a generally increasing trend for both the improvement from fine tuning and the performance relative to other preconditioners as the number of processors are increased. Figure 36 shows that for Trilinos ML, the gap between the default and the PSB performance shrinks as the number of processors are increased, but the overall performance of the preconditioner relative to oter solvers declines. Figure 37 shows that Hypre ParaSails benefits considerably from fine tuning and its relative performance with both the default and PSB parameters is mostly unchanged in response to a change in the number of processors.

5 Influence of Parameters on Solver Performance

In this section, we analyze the relative importance of the various preconditioner parameters on the time and memory performance of each preconditioner-package combination. In Section 4.2.2, we observed the impact of collective fine-tuning of the parameters on performance. Here we seek to study the effect of varying individual parameters that we experimented with. By means of the analysis in this section, we are able to determine the parameters that have the most and the least impact on a solver's performance. These results can help practitioners perform more selective fine-tuning of the parameters to their applications and enable them to extract good performance with less experimentation.

There are two main approaches described in global sensitivity analysis literature [27] that can potentially be used to capture the relative importance of parameters. These are based on linear regression parameters and conditional variance. It is possible to perform a linear regression with the performance values as the target variables and the various parameters as the dependent variables. The categorical parameters could be converted to multiple binary variables. One could choose the performance ratios (normalized performance with respect to the overall best configuration $DEF(\mathcal{C})$) as the target variable in order to adjust for effects of the individual matrices. The coefficients of the regression model would indicate the degree and direction of change in the performance ratio for each unit change in a parameter and could be interpreted as measures of sensitivity of the performance with respect to the influencing parameters. However, this approach is often not suitable in case of non-linear behavior or large number of outliers, as is the case in our experimental data. Furthermore, this approach does not adequately address the fact that the parameters tend to have different distributions. These drawbacks limit the applicability of this regression based approach for our domain, and we did not pursue it. Instead, we use an approach based on conditional variance in Section 5.1, and propose a new fine-tuning score based on variance conditioned on complementary factors in Section 5.2.

5.1 Conditional Variance-Based Sensitivity

It is possible to capture the relative importance of a given parameter in terms of the reduction in variance of the performance metric conditioned on the parameter's value [28]. Let F_i be the i^{th} parameter and μ be the performance metric of interest normalized for each matrix to reduce variance. Let $V(\mu)$ be the global variance of the performance metric, and let $V_{F_{-i}}(\mu|F_i = f_i)$ denote the conditional variance of μ when the parameter F_i takes the value f_i and the variation is over F_{-i}(i.e., all factors except F_i). The key idea is that freezing one potential source of variation results in a conditional variance $V_{F_{-i}}(\mu|F_i = f_i)$ that is lower than the unconditional global variance $V(\mu)$ and is determined only by parameters other than F_i. Since we desire a sensitivity measure independent of the parameter values f_i, we consider the expectation of the conditional variance over all possible values of the factor F_i, i.e., $E_{F_i}(V_{F_{-i}}(\mu|F_i))$. This sensitivity measure is always lower or equal to the global variance $V(\mu)$ and it can be shown that

$$E_{F_i}(V_{F_{-i}}(\mu|F_i)) + V_{F_i}(E_{F_{-i}}(\mu|F_i)) = V(\mu),$$

where the second term denotes the variance in the expected performance metric conditioned on F_i, which is always non-negative. A small value of $E_{F_i}(V_{F_{-i}}(\mu|F_i))$ or, in other words, a large value of $V_{F_i}(E_{F_{-i}}(\mu|F_i))$ implies that most of the variance in μ can be explained by the parameter F_i indicating that it is an important factor. The conditional variance $V_{F_i}(E_{F_{-i}}(\mu|F_i))$ is typically

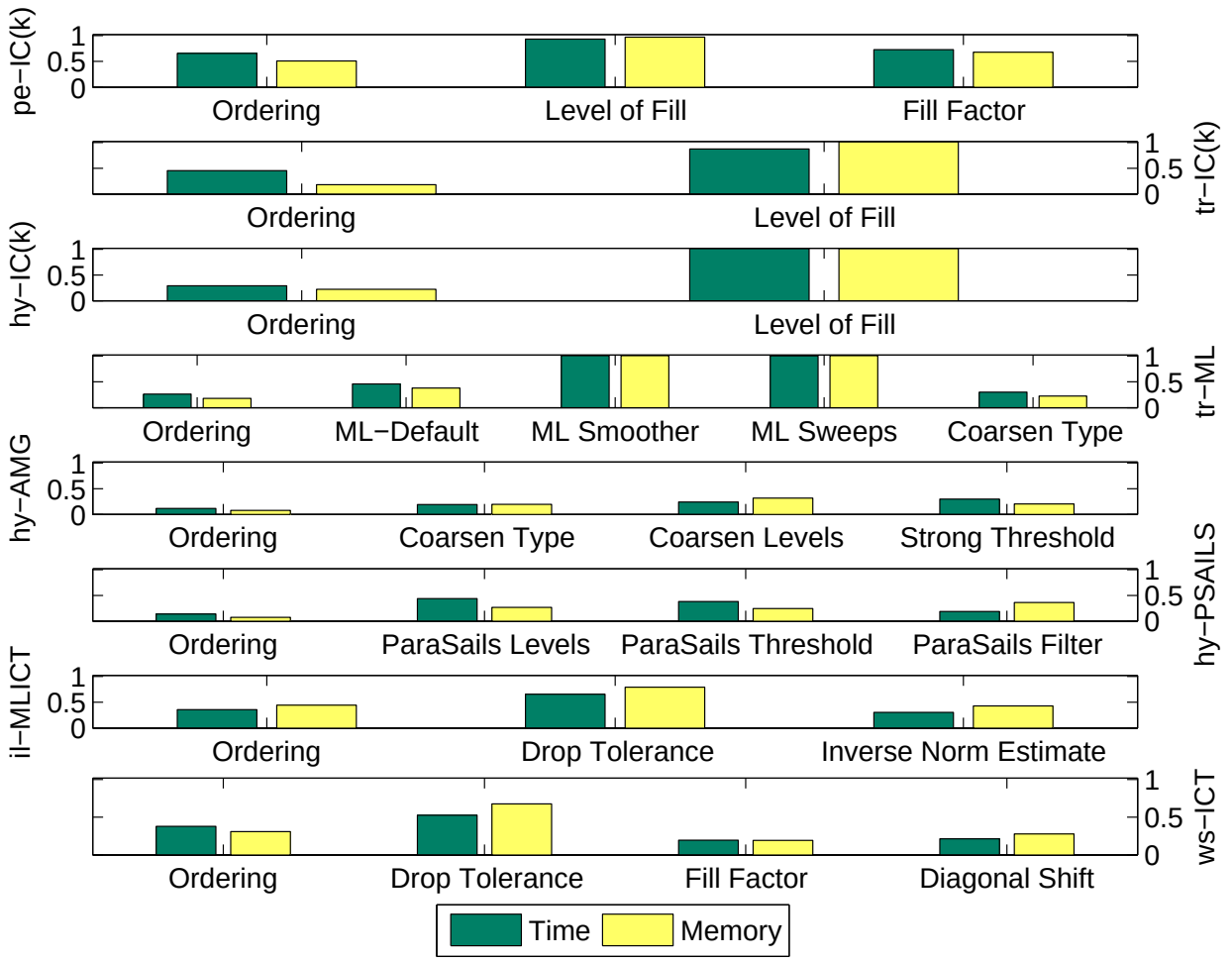


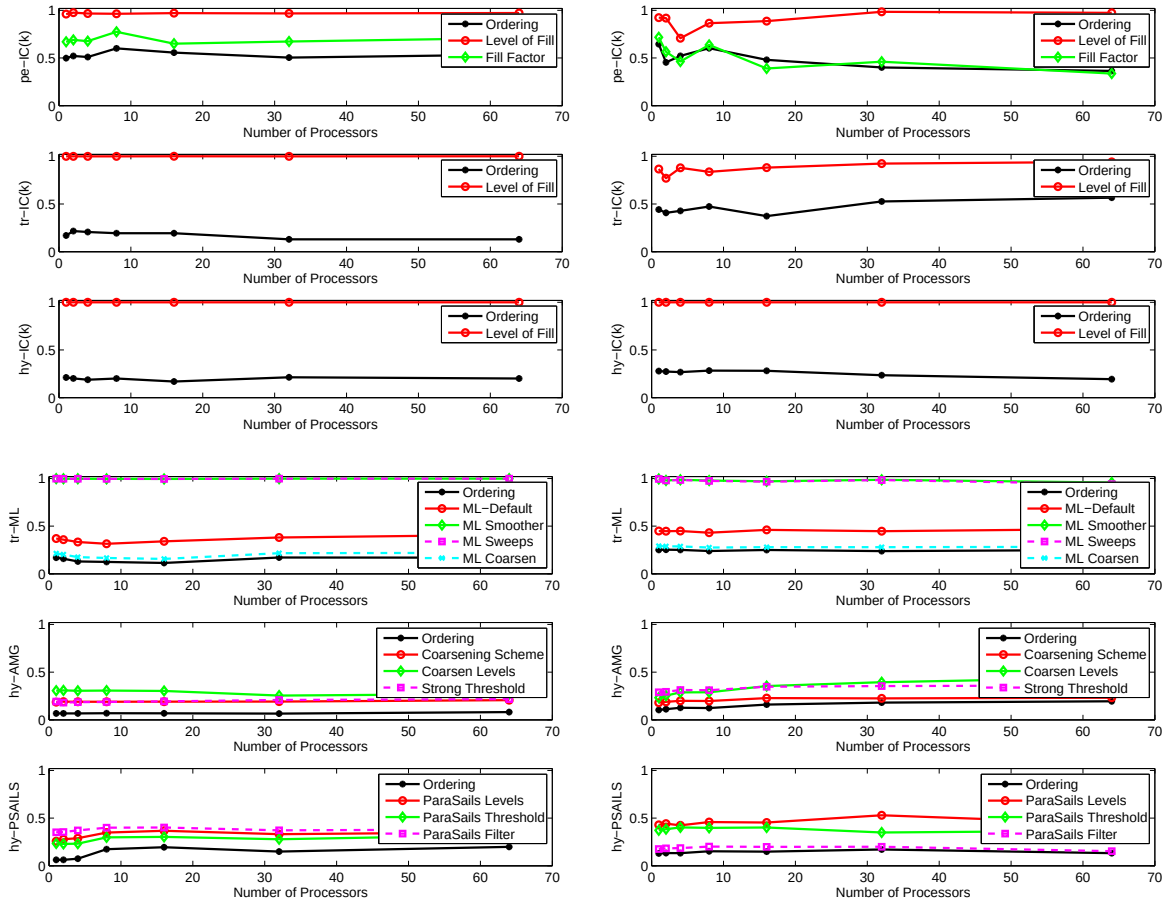
Figure 38: Conditional variance based sensitivity scores of the parameters in various preconditioners with respect to time and memory in the serial case.

normalized by the global variance $V(\mu)$ to give the importance measure

$$S_i = \frac{V_{F_i}(E_{F_{-i}}(\mu|F_i))}{V(\mu)}.$$

Since the behavior tends to vary across matrices, we also aggregate both the conditional and global variance across the matrices.

Figure 38 shows the relative importance of the various factors for different preconditioners in the serial case. The height of the bars corresponding to each parameter indicates the reduction in variance attained with respect to time and memory due to fixing the parameter to a particular value. A high value of reduction indicates that there is very little variance that is not explained by this parameter. For example, from the top three bar graphs in the figure, we observe that the level of fill parameter is the most important parameter whereas ordering is the least important one for all the IC(k) preconditioner implementations. For Trilinos ML, the smoother and the number of smoother sweeps seem to have the maximum influence. However, this is an artifact of the correlation between the parameters used in this study. We analyze this case in more detail later in Section 5.3. In the case of Hypre BoomerAMG, the number of aggressive coarsening levels impacts memory the most,



Relative importance of parameters with respect to memory.

Relative importance of parameters with respect to time.

Figure 39: Conditional variance based sensitivity scores of the various parameters for the different preconditioners in the parallel case. Each curve in the subplots corresponds to a parameter that is varied in our study.

whereas the strong threshold is the most important factor affecting time. For Hypre ParaSails, the memory usage is much more sensitive to the filter parameter than the threshold and the number of levels, but the opposite is true for time. For ILUPACK MLICT and WSMP ICT, drop tolerance is the most important parameter with respect to both time and memory.

Figure 39 shows the relative importance of the various parameters with respect to memory and time as the number of processors is increased from 1 to 64. We see that the relative importance of the parameters changes little in response to the change in the number of processors in the 1–64 range.

5.2 Variance-Based Fine-Tuning Score

The conditional variance based sensitivity score discussed in Section 5.1 is used commonly in statistical literature for retrospective analysis since a high value of sensitivity with respect to a parameter indicates that the response variable can be confined to a small interval (i.e., has low variance), which correlates to the explanatory and predictive power of the parameter. From the variance decomposition relation in Section 5.1, we note that the conditional variance $V_{F_i}(E_{F_{-i}}(\mu|F_i))$ indicates the spread between the average performance values obtained for different fixed values of a parameter. However, this measure does not capture the effect of modifying a single parameter *while keeping the rest fixed* as is common in a practical fine-tuning scenario. Hence, we consider an alternate fine-tuning sensitivity measure for a parameter F_i defined as the expectation of variance of μ for different values of F_i for fixed values on the rest of the parameters, i.e., $E_{F_{-i}}(V_{F_i}(\mu|F_{-i}))$. This variance is a better indication of the change in the performance one can expect by fine-tuning a single parameter keeping others fixed and we define this value normalized by the global variance as the *fine-tuning sensitivity*. Note that the two measures $V_{F_i}(E_{F_{-i}}(\mu|F_i))$ and $E_{F_{-i}}(V_{F_i}(\mu|F_{-i}))$ are closely related to each other and involve applying the aggregation and variance in different orders. When the parameters are all uncorrelated and μ exhibits linear dependence on the parameters, the two measures are identical.

Figure 40 shows the average normalized variation with respect to both time and memory for each of the fine-tunable parameters for different preconditioners in the serial case. The height of the bars corresponding to each parameter corresponds to the normalized variance of the change in performance one can expect by fine-tuning that parameter keeping others fixed. The relative heights of the bars is important in this case since it gives an indication of the variance on performance due to each parameter. This is more evident when the plots for PETSc IC(k) in Figures 38 and 40 are compared. In Figure 38, we saw that the ordering and fill factor parameters have only slightly lower variance-based scores compared to the level of fill parameter. However, Figure 40 shows that the variation in performance due to level of fill is much higher in comparison to that due to ordering and fill factor. Another key difference between Figures 38 and 40 is in the bar graph for Trilinos ML. This is due to correlation between parameters, which we discuss in detail in Section 5.3.

Figure 41 shows the effect of fine-tuning the various parameters on memory and time as the number of processors is increased. The level of fill has the largest influence on memory and time for the three IC(k) implementations in PETSc, Trilinos, and Hypre. The impact of level of fill on the solution time increases with the number of processors in case of PETSc IC(k). For Hypre BoomerAMG, the ordering scheme has the least effect on both time and memory, while the effect of the other parameters (coarsening scheme/levels and strong threshold) is substantial, but nearly flat as the number of processors varies. In the case of Hypre ParaSails, the number of levels has the maximum impact on both time and memory. Although the filter parameter has a significant impact on memory, the variation is negligible with respect to time.

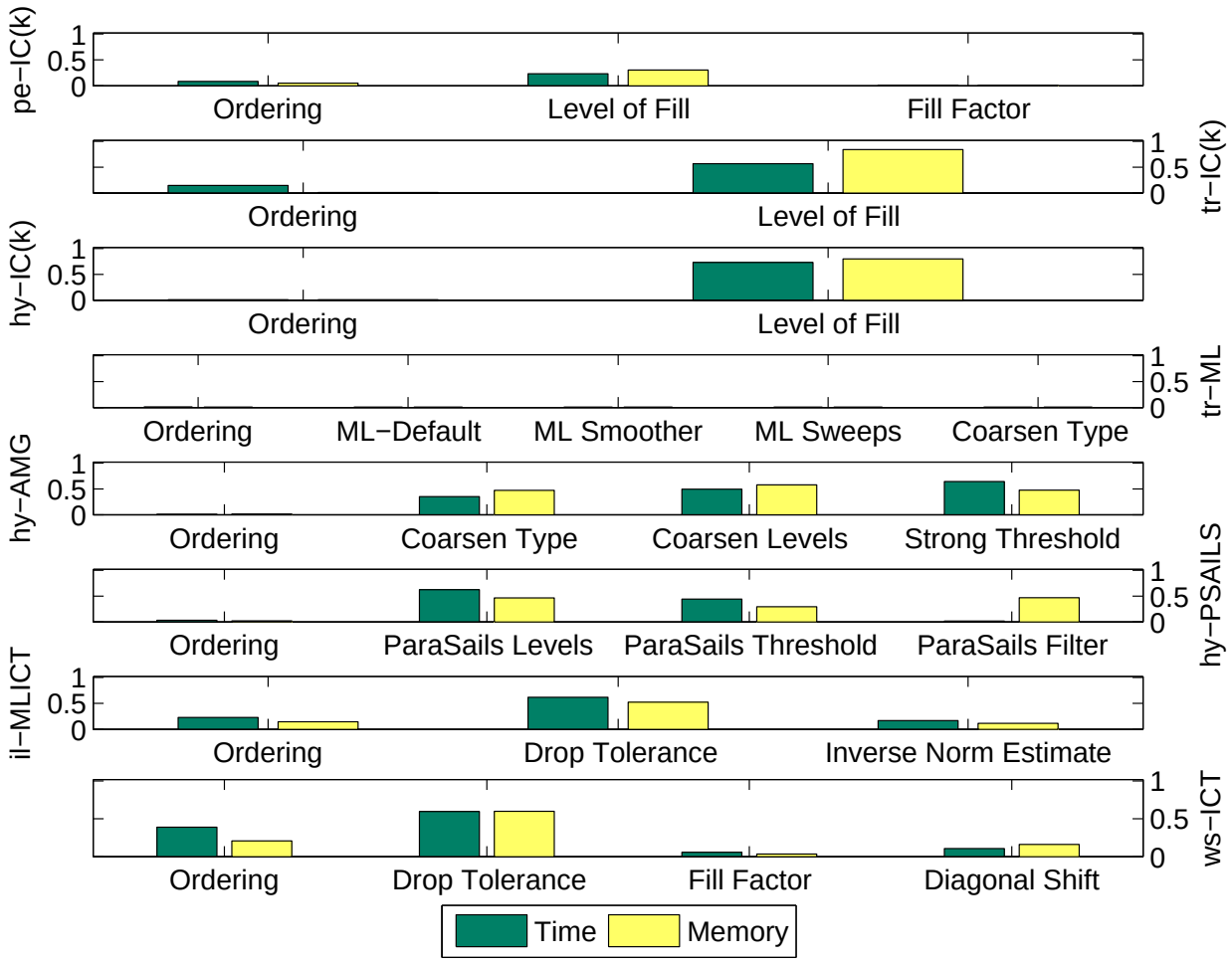
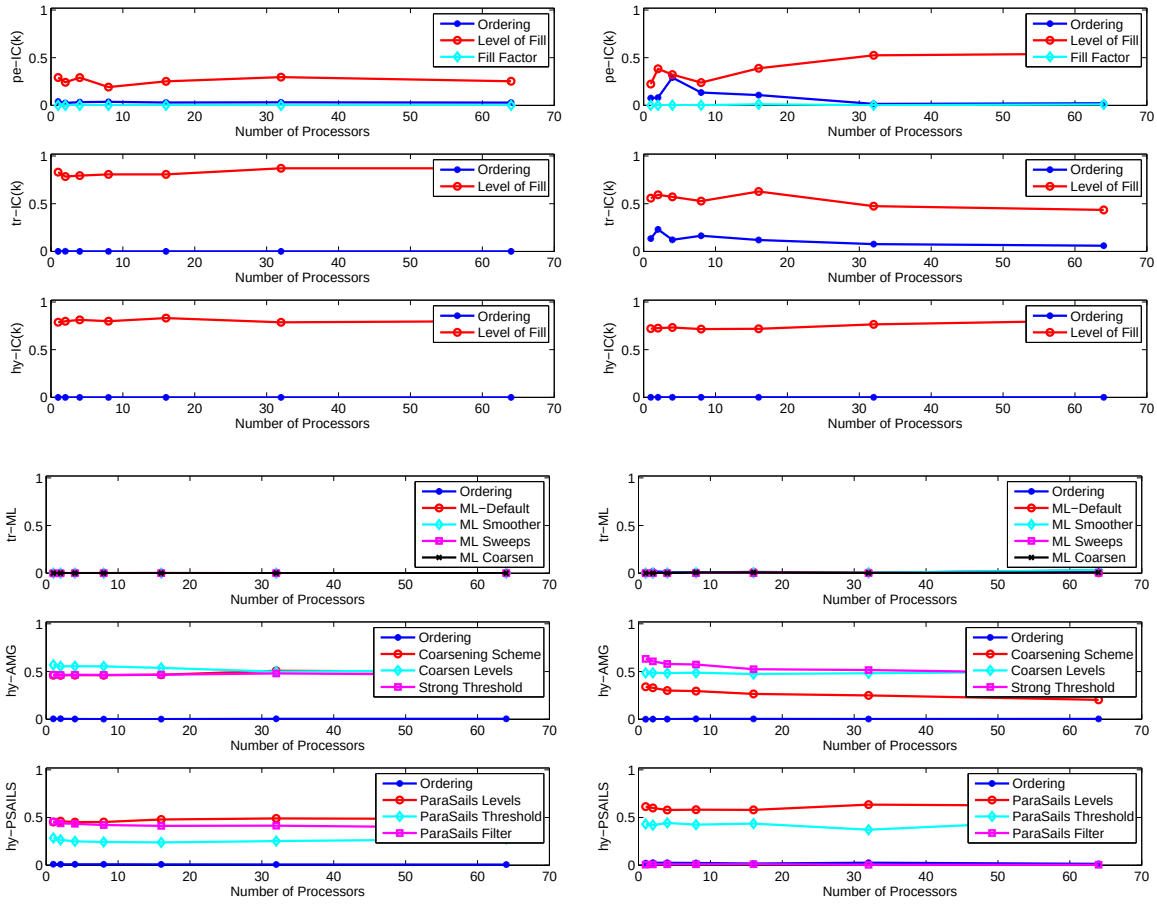
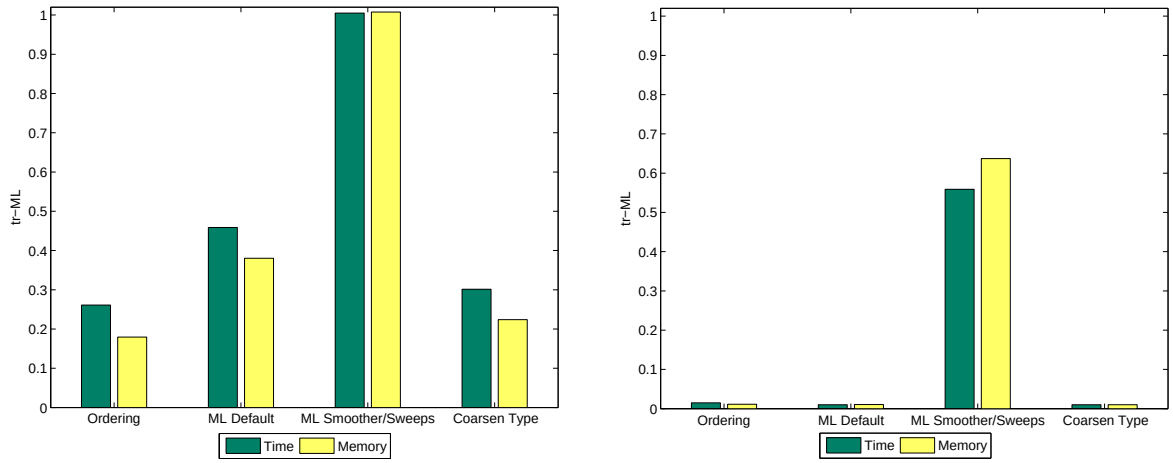


Figure 40: Average normalized variation in memory and time for the tunable parameters of various preconditioners in the serial case.



Variation in memory Variation in time

Figure 41: Average normalized variation in memory and time for the tunable parameters of various preconditioners in the parallel case. Each curve in the subplots corresponds to a parameter that is varied in our study.



(a) Conditional variance based sensitivity scores.

(b) Average normalized variation scores.

Figure 42: Conditional variance based sensitivity scores and average normalized variation scores for memory and time for the tunable parameters of Trilinos ML preconditioner in the serial case. The scales for the time and memory bars are different and are shown on the left and right, respectively.

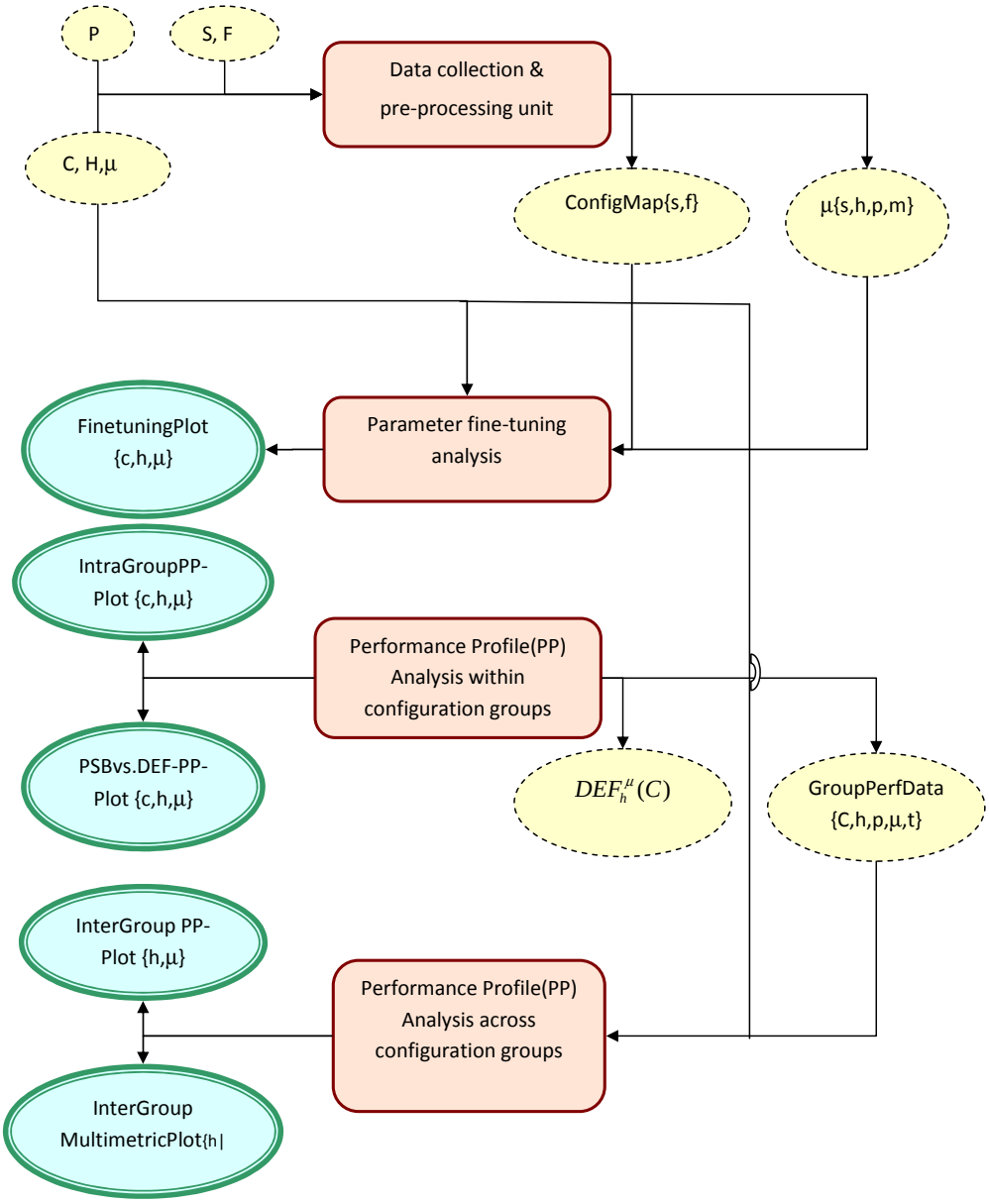
5.3 Correlated Parameters

The fine-tuning measures discussed above are suitable for the cases where each choice of a parameter can be made with all other choices of the remaining parameters to form valid configurations. In such a case, one can order the parameters and fine-tune them one at a time. However, in certain situations, there tend to exist groups of highly (positively or negatively) correlated parameters. An example of this is the smoother and number of smoother sweeps in Trilinos ML, which have a correlation coefficient $<$ -0.7. When there is a strong dependence among the parameters, then freezing all the other parameters has an implicit effect of limiting the range of possible values for the parameter in question resulting in a misleading value for the fine-tuning measure. For such cases, it is more appropriate to form groups of highly correlated parameters and fine-tune them simultaneously. Figure 42 shows the fine-tuning measures for Trilinos-ML with grouping of the parameters smoother and number of sweeps in the serial case. Unlike Figures 38 and 40, we observe that the combined parameter ML Smoother/Sweeps is the most important parameter in Figure 42. The scales indicate that the associated variances are significant.

In our experiments, we observed that across all the solver configuration groups, the parameters smoother and number of smoother sweeps for Trilinos ML are the only ones with absolute correlation ≥ 0.5. Hence, the fine-tuning scores shown in Figures 38–41 are realistic for the rest of the parameters.

6 Performance Analysis Infrastructure

In this section, we describe the software infrastructure that we used for the semiautomated collection, analysis, and visualization of performance data for the iterative solvers. Most results presented in Section 4 were generated using this framework. Figure 43 shows the various components of the framework, which is composed of a performance data collection unit and multiple analysis and visualization units. Although the framework is implemented for studying preconditioned iterative solvers, it is readily extensible to other domains where it may be useful to perform a comparative



P = {p}	:	Linear Systems	S = {s}	:	Solver Configurations
H = {h}	:	Hardware Configurations	F = {f}	:	Fine-tunable Factors
T = {t}	:	Type (DEF, PSB, DEF vs. PSB)	C	:	Solver Configuration Group
μ	:	Performance Metric (time, memory, MTP)	PP	:	Performance Profile

Figure 43: Overview of the performance analysis infrastructure. Boxes represent the processing units, dotted ellipses represent the input and output data, and the solid ellipses represent the plots generated for visualization.

evaluation of several configuration groups with a large number of configurations with respect to their performance on multiple metrics. We now describe each component in more detail.

6.1 Data Collection and Preprocessing Unit

This consists of serial and parallel driver programs for the solver packages of interest. The input to this component includes the set of linear systems $\mathcal{P}$, a set of hardware configurations (i.e., number of processors) $\mathcal{H}$, the set of supported solver configurations $\mathcal{S}$ with the details of the solvers, preconditioners, and related options and parameters, and a set of performance metrics μ. In addition, the user also specifies a grouping (C) of the solver configurations which can be a combination of any of the solver configuration components such as package, preconditioner, solver, ordering etc. For example, Figure 7 shows the performance profiles for groups created with the following specification: package:ILUPACK, preconditioner:MLICT, solver:CG, inverse norm estimate:100, and ordering:RCM.

Other choices of interest include driver specific information such as the right hand side (RHS), initial solution, exact solution, and stopping criterion, each of which has a default value and can also be modified by the user. The system performs empirical trials for each possible setting, collects the specified metrics, and pre-processes them appropriately to generate the performance data (denoted by μ). It also generates a mapping (*ConfigMap*) from the solver configurations S to various key attributes such as the package-name, preconditioner-name, solver and the associated parameters. This parameter mapping is required to partition the configurations into sub-groups that differ along a single parameter. For instance, the configuration map for PETSc IC(k) would include the ordering scheme, level of fill, and fill factors as parameters. A *ConfigMap* entry for PETSc IC(k) will have an integer value that corresponds to each parameter. For example, there are two integer values associated with ordering, 1 for RCM and 2 for ND. Similarly, we assign integer identifiers to other parameters. Thus, each solver configuration is mapped to an integer vector of parameter value identifiers.

6.2 Parameter Fine-tuning Analysis Unit

This unit computes the variability in the performance due to a single parameter while keeping all others fixed. This analysis is especially relevant for solver configurations within each group (Section 5). It requires as input the performance data μ as well as the solver configuration to parameter map *ConfigMap* generated by the data collection and pre-processing unit. In order to study the effect of a single parameter, for example ordering scheme, we first find groups of parameter vectors that vary only in the ordering scheme value. For example in case of ILUPACK MLICT, there are 25 groups corresponding to each combination of drop tolerance and inverse norm estimate (see Table 2). Each group has 5 solver configurations corresponding to the 5 different ordering schemes. For each such group, we calculate the average standard deviation of the percentage change in normalized memory and time performance for the *solved* problems in response to a change in ordering. Each of these average standard deviations are further averaged across the 25 groups to create the normalized plots, such as those in Figures 38–42.

6.3 Intra-Group Analysis Unit

The intra-group analysis unit computes good default solver configurations ($DEF_h^{\mu}(\mathcal{C})$) that result in the maximum AUC among all the solver configurations within each user specified group $\mathcal{C}$. Such groups typically represent package preconditioner combinations. For example, the data in Tables 6–9 is generated by this unit. The intra-group analysis unit also computes the problem specific best

configuration ($PSB_h^\mu(\mathcal{C})$) amongs all configurations in group $\mathcal{C}$ for each testcase in the problem set $\mathcal{P}$. Although, we use MTP as the metric (μ) of choice, a user can specify other metrics such as memory, time, or a weighted product of memory and time. For a given hardware configuration and performance metric, a series of performance profile (PP) plots, such as those shown in Figures 2–19, are created for each configuration for analysis at a fine-grained resolution. In addition, the effect of fine-tuning is captured by comparing the performance of the best default configuration and the problem specific best performance, as in Figures 32–35. The performance results or the μ values corresponding to both $DEF_h^\mu(\mathcal{C})$, $PSB_h^\mu(\mathcal{C})$ are compiled for each group $\mathcal{C}$, problem p, and hardware configuration h to generate group performance data (*GroupPerfData*) for further analysis.

6.4 Inter-Group Analysis Unit

The inter-group analysis unit provides a coarse grain comparison of the different solver configuration groups based on group performance data *GroupPerfData*. This is done both for the default configurations ($DEF_h^\mu(\mathcal{C})$) and for the problem-specific best ($PSB_h^\mu(\mathcal{C})$) configurations. Performance profile plots are generated for each hardware configuration for both DEF_h^μ and PSB_h^μ values, as in Figures 22–29). In addition, we also generate a multi-metric plot that simultaneously captures the trends for up to three metrics (e.g., memory, time and robustness) for both *PSB* and *DEF* performance, as in Figures 30 and 31). This provides a snapshot of the relative strengths and weaknesses of the various package-preconditioner combinations with respect to the performance metrics under consideration.

7 Concluding Remarks and Future Work

We performed an extensive empirical evaluation of some commonly used preconditioned iterative methods available in black box solver packages on a collection of matrices drawn from a wide range of scientific applications. For each package and preconditioner combination, we identify the best parameter choices using a novel performance profile based criterion that takes into consideration the number of problems solved along with the time and memory usage across all the problems in the collection. Our experiments reveal parameter configurations that are good candidates for default configurations. For each preconditioner, we quantify the benefits of parameter fine-tuning by comparing the best performance for each problem with the performance of our experimentally determined default parameters. Different preconditioners show varying levels of tunability and optimizing individual parameters impacts the performance to different degrees. We provide a comparison of the performance of various iterative solver configurations relative to the direct solver, which illustrates the successes and challenges in developing preconditioners for iterative solvers. The results also provide insight into the relative strengths and weaknesses of the various black box preconditioned iterative solver packages. We observed that different implementations of the same preconditioning method can vary widely in performance.

This study, admittedly, has its limitations. The results used for analysis are derived from a test suite of only 30 problems. Although our test suite includes problems from multiple applications, we kept its size modest due to the sheer number of trials (2156) for each matrix. Therefore, the results may not be generalizable to specific application domains. However, the performance collection and reporting infrastructure we have developed is independent of the test suite and can be used on any set of test matrices from any domain. Application scientists can provide us with specific matrices in their domain and the results specific to those matrices can be generated with little effort.

The methodology described in this paper can be helpful to researchers in evaluating different aspects of new solver techniques in a systematic fashion. As part of our future work on this topic,

we plan to set up an anonymous FTP site where users can submit their matrices and obtain a report on the relative performance of each solver configuration group. We also plan to use clustering and other data-mining techniques to explore if a good choice of preconditioner and parameters can be predicted from the characteristics of the coefficient matrix and the underlying physical problem that generates the matrix. Preliminary results [13] have been quite encouraging.

Acknowledgements. The authors wish to thank Allison Baker, Matthias Bollhöfer, Edmond Chow, Robert Falgout, Michael Heroux, Jonathan Hu, Marzio Sala, Heidi Thornquist, Raymond Tuminaro, and Ulrike Yang for their assistance in installing and using their packages, answering queries, and suggesting the parameters to use and tune for the experiments. We also thank Felix Kwok for the initial installation of some of the packages and their driver programs.

This publication is partially based on work supported by Award No. KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST).

References

[1] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc: portable, extensible toolkit for scientific computation, 2001. *http://www.mcs.anl.gov/petsc*.

[2] Michele Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418–477, 2002.

[3] Michele Benzi and Miroslav Tuma. A comparative study of sparse approximate inverse preconditioners. *Applied Numerical Mathematics*, 30(2):305–340, 1999.

[4] Matthias Bollhöfer, Yousef Saad, and Olaf Schenk. ILUPACK V2.2. Available online at *http://www.math.tu-berlin.de/ilupack*, January 2006.

[5] Edmond Chow. Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns. *International Journal of High Performance Computing Applications*, 15(1):56–74, 2001.

[6] Edmond Chow and Yousef Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, 19(3):995–1023, 1998.

[7] Timothy A. Davis. The university of Florida sparse matrix collection. Technical report, Department of Computer Science, University of Florida, Jan 2007.

[8] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[9] Jack Dongarra and Alfredo Buttari. Freely available software for linear algebra on the web, 2006. *http://www.netlib.org/utk/people/JackDongarra/la-sw.html*.

[10] Robert D. Falgout and Ulrike Meier Yang. *hypre*, High Performance Preconditioners: Users manual. Technical report, Lawrence Livermore National Laboratory, 2006. Paper appears in ICCS 2002.

[11] John Fettig, Seid Koric, and Nahil Sobh. A comparison of direct and iterative linear sparse solvers in computational solid mechanics. In *International Conference on Preconditioning Techniques for Large Sparse Matrix Problems*, Napa, CA, 2003.

[12] Michael W. Gee, Chris M. Siefert, Jonathan J. Hu, Raymond S. Tuminaro, and Marzio G. Sala. ML 5.0 smoothed aggregation user's guide. Technical Report SAND2006-2649, Sandia National Laboratories, 2006.

[13] Thomas George, Anshul Gupta, and Vivek Sarin. A recommendation system for preconditioned iterative solvers. Technical Report RC 24600, IBM T. J. Watson Research Center, Yorktown Heights, NY, July 2008. A short version appears in *Proceedings of the IEEE International Conference on Data Mining*, 2008.

[14] John R. Gilbert and Sivan Toledo. An assessment of incomplete-LU preconditioners for non-symmetric linear systems. *Informatica*, 24(3):409–425, 2000.

[15] Anshul Gupta. WSMP: Watson sparse matrix package (Part-I: Direct solution of symmetric sparse systems). Technical Report RC 21886, IBM T. J. Watson Research Center, Yorktown Heights, NY, November 2000. *http://www.cs.umn.edu/~agupta/wsmp.*

[16] Anshul Gupta. WSMP: Watson sparse matrix package (Part-III: Iterative solution of sparse systems). Technical Report RC 24398, IBM T. J. Watson Research Center, Yorktown Heights, NY, November 2007. *http://www.cs.umn.edu/~agupta/wsmp.*

[17] Anshul Gupta and Thomas George. Adaptive techniques for improving the performance of incomplete factorization preconditioning. Technical Report RC 24598, IBM T. J. Watson Research Center, Yorktown Heights, NY, July 2008. To appear in *SIAM Journal on Scientific Computing.*

[18] Van Emden Henson and Ulrike Meier Yang. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics: Transactions of IMACS*, 41(1):155–177, 2002.

[19] Michael A. Heroux. AztecOO users guide. Technical Report SAND2004-3796, Sandia National Laboratories, 2004.

[20] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Raymond S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. An overview of the Trilinos project. *ACM Transactions on Mathematical Software*, 31(3):397–423, 2005.

[21] David Hysom and Alex Pothen. A scalable parallel algorithm for incomplete factor preconditioning. *SIAM Journal on Scientific Computing*, 22(6):2194–2215, 2000.

[22] J. W. Ruge and K. Stüben. Multigrid methods. In Stephen F. McCormick, editor, *Frontiers in Applied Mathematics*, volume 3, pages 73–130. SIAM, Philadelphia, PA, 1987.

[23] Yousef Saad. *Iterative Methods for Sparse Linear Systems, 2nd edition.* SIAM, 2003.

[24] Yousef Saad and Brian Suchomel. ARMS: An algebraic recursive multilevel solver for general sparse linear systems. *Numerical Linear Algebra with Applications*, 9(5):359–378, 2002.

[25] Yousef Saad and Henk A. van der Vorst. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1-2):1–33, 2000.

[26] Marzio G. Sala and Michael A. Heroux. Robust algebraic preconditioners with IFPACK 3.0. Technical Report SAND-0662, Sandia National Laboratories, 2005.

[27] Andrea Saltelli, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. *Global Sensitivity Analysis: The Primer*. WileyBlackwell, 2008.

[28] Andrea Saltelli and Stefano Tarantola. On the relative importance of input factors in mathematical models: Safety assessment for nuclear waste disposal. *Journal of the American Statistical Association*, 97(459):702–709, 2002.

[29] Raymond S. Tuminaro. Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing (CDROM)*, 2000.

Appendix

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	3.61e+07	0.00	21.40	499	9.54e-06	3.3e-07
af_shell7	1.19e+08	0.00	115.18	751	9.87e-06	1.48e-07
algor-big	5.34e+08	0.00	404.14	477	8.39e-06	3.27e-10
audikw_1	4.9e+08	0.00	727.72	1000	3.4	0.299
bmwcra_1	6.77e+07	0.00	88.93	1000	30.2	0.0471
ctu-1	4.71e+08	0.00	738.68	1000	-1	0.986
ctu-2	1.78e+08	0.01	261.02	1000	-1	0.971
cfd1	1.28e+07	0.00	6.59	426	9.63e-06	1.01e-08
cfd2	2.17e+07	0.00	26.34	1000	0.00169	0.000123
conti20	1.17e+07	0.00	11.63	1000	4.05	0.934
garybig	2.53e+09	0.20	438.99	113	8.12	41.9
G3_circuit	8.72e+07	0.01	81.89	647	9.2e-06	1.6e-06
hood	7.04e+07	0.00	54.69	592	9.88e-06	6.99e-06
inline_1	2.34e+08	0.00	354.99	1000	24.8	0.497
kyushu	1.83e+08	0.01	32.65	141	9.88e-06	2.2e-06
ldoor	3.04e+08	0.00	423.43	1000	9.9e-05	0.000639
msdoor	1.32e+08	0.00	181.54	1000	0.0433	0.89
mstamp-2c	4.49e+08	0.00	104.99	140	9.1e-06	2.03e-06
nastran-b	7.09e+08	0.00	1035.06	1000	0.000427	6.78e-06
nd24k	1.74e+08	0.00	220.21	1000	0.342	0.0129
oilpan	2.35e+07	0.00	29.14	1000	0.0362	0.7
parabolic_fem	3.57e+07	0.00	13.22	283	9.98e-06	6.18e-08
pga-rem1	8.18e+07	0.00	119.43	1000	0.047	0.00057
pga-rem2	3.33e+08	0.03	281.16	553	9.99e-06	6.46e-07
qa8fk	1.17e+07	0.00	0.32	19	2.84e+04	6.83
qa8fm	1.17e+07	0.00	0.12	4	1.55e-06	4.63e-06
ship_003	5.17e+07	0.00	64.98	1000	0.0044	0.0117
shipsec5	6.54e+07	0.00	50.60	609	9.75e-06	4.93e-07
thermal2	8.34e+07	0.00	111.58	1000	0.000566	6.9e-07
torso	2.42e+07	0.00	5.11	164	9.63e-06	9.86e-08

Table 15: Table showing the raw data for PROC1:petsc:IC(k):CG:RCM:LF0:F1

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	5.38e+07	1.30	36.00	1000	2.04e-05	1.23e-06
af_shell7	1.96e+08	5.67	140.97	910	9.87e-06	4.36e-07
algor-big	9.51e+08	56.13	678.64	814	9.52e-06	6.07e-10
audikw_1	1.23e+09	129.80	735.18	1000	3.28	0.273
bmwcra_1	1.24e+08	7.06	84.34	1000	48.4	0.476
ctu-1	1.28e+09	107.97	752.62	1000	-1	0.983
ctu-2	4.21e+08	32.79	249.01	1000	-1	0.953
cfd1	5.92e+07	4.29	18.55	770	9.61e-06	5.57e-08
cfd2	1.1e+08	7.64	43.03	1000	0.00999	0.000585
conti20	2.52e+07	2.42	8.70	1000	6.72	0.459
garybig	†	†	†	†	†	†
G3_circuit	4.37e+08	33.46	392.10	809	9.67e-06	2.06e-06
hood	1.11e+08	3.02	39.13	477	9.99e-06	6.17e-06
inline_1	4.08e+08	24.88	329.60	1000	28.7	0.853
kyushu	8.39e+08	69.45	480.97	1000	3.87e-05	3.54e-06
ldoor	4.96e+08	17.70	437.68	1000	0.000112	0.000289
msdoor	2.17e+08	7.64	175.65	1000	0.0328	0.439
mstamp-2c	7.49e+08	38.40	123.92	172	9.72e-06	1.12e-06
nastran-b	1.23e+09	68.46	1097.00	1000	0.185	0.00767
nd24k	4.09e+08	281.89	224.74	1000	2.94	0.0152
oilpan	3.72e+07	1.05	23.87	1000	0.0122	0.279
parabolic_fem	1.46e+08	11.24	104.67	794	9.84e-06	1.18e-07
pga-rem1	4.13e+08	32.86	477.37	1000	0.212	0.00159
pga-rem2	1.65e+09	136.91	1452.73	723	9.97e-06	7.07e-07
qa8fk	3.24e+07	2.42	0.50	22	1.11e+04	2.22
qa8fm	3.24e+07	2.41	0.21	9	4.06e-06	1.49e-05
ship_003	8.74e+07	3.51	54.76	1000	0.00193	0.00236
shipsec5	1.07e+08	3.89	44.49	576	9.88e-06	4.8e-07
thermal2	3.51e+08	27.74	359.50	1000	0.00369	3.82e-05
torso	1.89e+08	12.29	22.41	301	8.93e-06	7.75e-08

Table 16: Table showing the raw data for PROC1:petsc:BSolve:CG:RCM:ALL:NONE

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.41e+07	1.49	114.03	1000	1.14	0.102
af_shell7	7.88e+07	5.90	459.54	1000	0.793	0.0207
algor-big	1.68e+08	22.61	1689.01	1000	0.513	0.00047
audikw_1	1.47e+08	19.65	1493.09	1000	6.99	0.0464
bmwcra_1	2.32e+07	2.79	206.92	1000	9.73	0.0217
ctu-1	1.59e+08	19.42	1488.32	1000	5.31	0.000125
ctu-2	5.99e+07	7.12	543.26	1000	7.51	3.98e-05
cfd1	1.1e+07	0.65	53.82	1000	0.000383	2.57e-07
cfd2	1.93e+07	1.15	92.61	1000	0.00851	4.77e-06
conti20	3.18e+06	0.42	33.72	1000	3.8	0.00567
garybig	†	†	†	†	†	†
G3_circuit	2.46e+08	9.91	146.53	198	9.98e-06	1.27e-06
hood	3.44e+07	3.08	239.66	1000	0.000439	0.00232
inline_1	7.86e+07	9.43	732.56	1000	1.6	0.0062
kyushu	1.55e+08	9.73	791.46	1000	1.59	4.31e-07
ldoor	1.49e+08	13.43	1057.66	1000	0.00308	0.0098
msdoor	6.49e+07	5.82	455.75	1000	0.0568	0.0404
mstamp-2c	1.41e+08	19.00	695.18	515	9.64e-06	0.000112
nastran-b	2.35e+08	28.66	2236.05	1000	2.51	0.64
nd24k	1.12e+07	6.23	430.99	1000	1.33	0.000776
oilpan	1.15e+07	0.99	79.21	1000	0.0346	0.0271
parabolic_fem	8.2e+07	3.27	29.17	109	9.65e-06	4.31e-08
pga-rem1	2.28e+08	9.22	178.35	266	9.84e-06	5.64e-08
pga-rem2	9.14e+08	36.59	268.56	98	8.45e-06	3.97e-07
qa8fk	1.03e+07	0.59	2.13	42	8.88e+04	6.03e-06
qa8fm	1.03e+07	0.57	0.27	4	3.19e-06	3.9e-06
ship_003	1.9e+07	2.10	160.60	1000	0.0469	1.14
shipsec5	2.81e+07	2.76	212.16	1000	0.0147	0.000224
thermal2	1.92e+08	8.27	266.95	422	9.71e-06	5.28e-09
torso	3.13e+07	1.60	12.92	101	9.1e-06	9.97e-09

Table 17: Table showing the raw data for PROC1:tril:IC(k):CG:RCM:LF8:NONE

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	9.66e+06	1.94	394.20	671	9.85e-06	6.44e-05
af_shell7	5.7e+07	6.94	639.04	288	9.38e-06	5.24e-05
algor-big	1.66e+08	32.46	2295.10	216	8.74e-06	3.91e-07
audikw_1	2.07e+08	39.16	6215.41	671	9.43e-06	6.5e-06
bmwcra_1	2.15e+07	4.17	555.95	447	7.76e-06	3.07e-07
ctu-1	†	†	†	†	†	†
ctu-2	5.53e+07	10.87	3370.60	1000	3.61	2.48e-05
cfd1	1.18e+07	1.00	14.92	82	9.28e-06	7.46e-08
cfd2	1.96e+07	1.64	133.94	434	9.97e-06	1.71e-07
conti20	3.11e+06	0.75	143.41	864	9.76e-06	7.22e-07
garybig	†	†	†	†	†	†
G3_circuit	2.87e+08	12.88	286.71	163	9.57e-06	1.85e-06
hood	2.33e+07	3.95	274.04	206	9.74e-06	0.000331
inline_1	8.91e+07	16.50	4290.27	1000	17.5	0.00285
kyushu	1.67e+08	14.68	460.71	156	9.67e-06	4.65e-10
ldoor	1.01e+08	16.87	3085.47	526	9.91e-06	0.000331
msdoor	4.45e+07	7.38	2500.23	1000	0.0111	0.0137
mstamp-2c	1.38e+08	27.06	529.72	57	9.64e-06	0.000152
nastran-b	2.89e+08	53.63	8626.90	634	9.89e-06	0.000843
nd24k	†	†	†	†	†	†
oilpan	7.12e+06	1.19	371.61	1000	0.00392	0.00741
parabolic_fem	6.81e+07	3.61	13.39	21	6.73e-06	1.49e-08
pga-rem1	2.46e+08	11.22	1327.51	849	9.96e-06	4.28e-08
pga-rem2	†	†	†	†	†	†
qa8fk	1.12e+07	0.92	0.87	5	6.86e+03	2.01e-06
qa8fm	1.12e+07	0.92	0.51	2	5.75e-06	6.74e-06
ship_003	3.25e+07	3.72	628.14	681	9e-09	0.00169
shipsec5	3.27e+07	3.91	293.97	252	9.76e-06	4.14e-06
thermal2	1.69e+08	8.69	53.66	37	7.79e-06	3.19e-09
torso	3.68e+07	2.43	16.87	45	8.93e-06	7.86e-09

Table 18: Table showing the raw data for PROC1:tril:ML:CG:RCM:SA:SGS_SS3_PMETIS

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.9e+08	3.53	78.22	1000	1.95	1
af_shell7	5.49e+08	9.16	141.31	490	9.65e-06	5.07e-07
algor-big	4.47e+09	111.91	510.55	319	9.42e-06	4.26e-10
audikw_1	3.93e+09	113.78	944.64	641	9.29e-06	6.43e-09
bmwcra_1	6.19e+08	14.38	197.76	1000	3.32	0.112
ctu-1	†	†	†	†	†	†
ctu-2	†	†	†	†	†	†
cfd1	7.69e+07	1.54	34.68	1000	25.1	0.609
cfd2	1.34e+08	2.47	69.59	1000	1	1
conti20	8.47e+07	3.42	24.34	1000	1.67	1
garybig	†	†	†	†	†	†
G3_circuit	3.04e+08	11.88	63.73	407	9.44e-06	1.73e-06
hood	4.66e+08	6.22	162.25	1000	0.026	0.00976
inline_1	†	†	†	†	†	†
kyushu	2.09e+09	23.74	103.84	131	9.73e-06	1.85e-06
ldoor	1.04e+09	25.72	741.44	1000	0.028	0.068
msdoor	8.78e+08	11.70	324.50	1000	1	1
mstamp-2c	3.75e+09	91.64	98.64	75	9.84e-06	2.35e-05
nastran-b	6.27e+09	141.78	1067.40	520	9.7e-06	1.4e-07
nd24k	2.36e+09	321.35	26.69	79	8.61e-06	3.53e-09
oilpan	8.03e+07	1.40	29.91	1000	0.0163	0.992
parabolic_fem	1.68e+08	4.03	10.73	202	9.4e-06	1.04e-07
pga-rem1	2.84e+08	11.16	125.06	928	9.66e-06	4.57e-08
pga-rem2	1.15e+09	44.66	186.80	333	9.65e-06	6e-07
qa8fk	7.2e+07	1.34	1.78	56	4.56e-06	3.74e-08
qa8fm	7.2e+07	1.33	0.16	4	2.45e-09	4.88e-09
ship_003	2.57e+08	4.09	93.07	994	9.9e-06	9.64e-07
shipsec5	1.96e+08	4.12	28.97	234	6.83e-06	6.53e-07
thermal2	3.93e+08	9.60	93.81	775	9.86e-06	1.47e-08
torso	2.19e+08	2.70	4.72	77	9e-06	1.01e-07

Table 19: Table showing the raw data for PROC1:hypre:IC(k):CG:RCM:LF1:NzINF

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.72e+07	0.53	219.41	1000	0.000724	6.28e-05
af_shell7	9.65e+07	2.55	107.92	105	8.72e-06	9.46e-08
algor-big	3.41e+08	10.45	543.65	147	8.31e-06	2.01e-10
audikw_1	†	†	†	†	†	†
bmwcra_1	4.87e+07	1.24	299.95	618	9.01e-06	2e-10
ctu-1	†	†	†	†	†	†
ctu-2	9.39e+07	2.80	1312.54	1000	-1	0.657
cfd1	1.3e+07	0.28	13.11	165	9.69e-06	3.07e-08
cfd2	2.24e+07	0.49	139.65	903	9.85e-06	2.82e-07
conti20	5.75e+06	0.16	54.55	1000	0.0324	0.000313
garybig	4.83e+09	81.61	11804.23	467	9.88e-06	3.69e-05
G3_circuit	1.73e+08	3.39	94.41	120	8.39e-06	2.6e-07
hood	4.7e+07	1.27	155.62	289	1.97e-06	9.23e-07
inline_1	1.44e+08	3.92	1738.58	1000	25	0.249
kyushu	1.65e+08	3.96	547.21	341	9.72e-06	7.56e-07
ldoor	1.77e+08	5.18	1181.44	469	9.91e-06	1.33e-05
msdoor	8.55e+07	2.29	1126.15	1000	0.00166	0.00363
mstamp-2c	2.67e+08	8.44	139.43	39	7.1e-06	6.71e-07
nastran-b	4.52e+08	12.52	4315.45	760	9.86e-06	1.91e-07
nd24k	6.6e+07	2.29	911.06	1000	0.117	0.00112
oilpan	1.04e+07	0.33	128.97	968	9.86e-06	0.000802
parabolic_fem	4.14e+07	1.19	15.20	59	8.94e-06	4.33e-08
pga-rem1	1.73e+08	2.64	366.36	485	9.92e-06	5.55e-08
pga-rem2	7.07e+08	13.06	619.01	177	9.36e-06	6.19e-07
qa8fk	1.21e+07	0.40	2.26	32	8.99e-06	1.16e-08
qa8fm	2.4e+06	0.04	0.55	26	2.2e-06	9.99e-06
ship_003	3.54e+07	0.85	230.98	790	9.84e-06	0.000552
shipsec5	4.55e+07	1.16	321.86	648	2.31e-06	8.89e-08
thermal2	1.23e+08	2.54	92.16	134	9.76e-06	4.72e-09
torso	2.76e+07	0.59	10.20	55	9.36e-06	3.97e-08

Table 20: Table showing the raw data for PROC1:hypre:AMG:CG:RCM:PMIS:AGG10_ST0.9

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	4.77e+07	4.25	20.46	673	9.78e-06	3.54e-07
af_shell7	1.69e+08	15.72	135.27	980	9.75e-06	3.15e-07
algor-big	4.9e+08	79.47	392.78	763	9.13e-06	3.15e-10
audikw_1	6.25e+08	170.77	585.35	1000	0.0178	9.87e-05
bmwcra_1	8.36e+07	10.78	65.33	1000	11.8	0.262
ctu-1	†	†	†	†	†	†
ctu-2	3.76e+08	137.67	296.52	1000	-1	0.856
cfd1	5.11e+07	4.09	11.39	428	9.53e-06	5.32e-08
cfd2	7.84e+07	7.58	30.63	637	9.81e-06	3.24e-07
conti20	3.45e+07	5.39	13.78	1000	0.0827	0.000558
garybig	†	†	†	†	†	†
G3_circuit	2.21e+08	12.95	141.70	716	9.98e-06	1.78e-06
hood	†	†	†	†	†	†
inline_1	2.67e+08	41.08	243.01	1000	43.2	0.542
kyushu	3.24e+08	32.57	232.89	777	9.96e-06	8.87e-07
ldoor	4.62e+08	72.62	168.07	420	9.81e-06	1.14e-05
msdoor	2.1e+08	31.39	182.73	1000	0.000745	0.000689
mstamp-2c	4.04e+08	64.58	72.75	173	9.87e-06	2.21e-06
nastran-b	8.06e+08	135.30	766.71	1000	0.00458	8.37e-05
nd24k	3.02e+07	4.53	60.78	1000	0.0126	0.000178
oilpan	3.66e+07	2.53	18.48	925	9.67e-06	4.57e-07
parabolic_fem	9.45e+07	4.95	45.60	632	1e-05	2.13e-07
pga-rem1	1.85e+08	10.71	172.80	1000	0.00183	4.19e-05
pga-rem2	6.94e+08	43.64	457.99	639	9.93e-06	5.98e-07
qa8fk	2.37e+07	0.45	1.89	239	9.32e-06	5.31e-07
qa8fm	3e+07	1.13	0.10	7	9.73e-08	2.85e-07
ship_003	3.85e+07	2.67	27.93	846	9.62e-06	2.36e-06
shipsec5	5.76e+07	4.57	14.78	321	1.28e-06	9.15e-08
thermal2	2.06e+08	12.31	178.26	1000	0.000796	9.36e-06
torso	6.06e+07	3.14	8.15	203	9.97e-06	1.05e-07

Table 21: Table showing the raw data for PROC1:hypre:PSAILS:CG:ND:PLev1:Th0.1_Flt0.001

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	3.88e+07	2.28	39.64	697	9.87e-10	1.18e-05
af_shell7	2.55e+08	11.79	105.98	280	6.84e-10	7.82e-06
algor-big	5.71e+08	45.64	577.61	510	2.47e-09	0.000113
audikw_1	8.04e+08	93.16	1870.09	1000	5.63e-08	0.001
bmwcra_1	9.32e+07	7.10	203.05	907	6.14e-09	2.38e-08
ctu-1	8.98e+08	75.19	2366.50	1000	-1	0.736
ctu-2	2.85e+08	24.20	460.05	1000	-1	0.547
cfd1	3.02e+07	1.85	6.79	237	9.88e-07	3.17e-07
cfd2	8.99e+07	5.96	20.21	198	8.98e-07	2.45e-05
conti20	1.22e+07	0.93	21.92	1000	6.5e-06	0.000109
garybig	†	†	†	†	†	†
G3_circuit	4.32e+08	17.43	68.89	138	1.12e-08	4.73e-06
hood	1.23e+08	7.16	52.32	250	7.28e-10	1.42e-06
inline_1	3.47e+08	25.95	871.31	1000	2.94e-05	0.428
kyushu	3.75e+08	24.53	87.77	170	5.09e-05	1.38e-06
ldoor	5.95e+08	33.03	328.64	348	3.54e-10	6.78e-06
msdoor	2.63e+08	15.33	412.94	1000	2.85e-06	0.0989
mstamp-2c	3.08e+08	26.56	164.15	259	6.45e-10	7.4e-07
nastran-b	1.15e+09	93.55	2456.18	788	1.97e-11	1.29e-05
nd24k	2.32e+07	7.14	94.11	584	3.5e-07	1.66e-06
oilpan	2.82e+07	1.58	34.98	1000	7.89e-06	0.0682
parabolic_fem	7.57e+07	1.81	15.55	176	9.16e-07	1.55e-05
pga-rem1	3.7e+08	16.05	87.38	133	7.99e-08	5.13e-06
pga-rem2	1.43e+09	66.19	151.97	56	4.25e-09	2.09e-06
qa8fk	2.19e+07	1.01	0.70	34	153	3.18
qa8fm	1.16e+07	0.41	0.13	9	4.75e-12	7.66e-11
ship_003	3.89e+07	3.36	38.43	425	1.33e-11	0.00177
shipsec5	8.24e+07	5.99	14.21	75	1.68e-09	1.35e-07
thermal2	3.58e+08	11.87	88.42	221	3.39e-07	8.75e-07
torso	8.04e+07	4.64	8.64	60	2.07e-06	6.27e-07

Table 22: Table showing the raw data for PROC1:ilupack:MLICT:CG:RCM:DT3e-2:IE75

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	5.49e+07	0.74	6.81	102	9.98e-06	2.16e-08
af_shell7	2.21e+08	3.07	52.92	161	8.96e-06	7.83e-07
algor-big	1.04e+09	64.33	262.84	221	5.24e-06	5e-10
audikw_1	1.19e+09	152.10	208.07	176	7.62e-07	2.43e-10
bmwcra_1	1.52e+08	9.80	40.82	265	8.2e-06	5.35e-10
ctu-1	1.85e+09	147.90	1572.19	1000	4.68e-06	1.03e-06
ctu-2	2.93e+09	120.43	1126.53	1000	0.332	0.869
cfd1	1.49e+08	9.51	9.61	95	1.56e-07	1.2e-09
cfd2	3.2e+08	16.79	22.84	115	1.57e-07	4.45e-08
conti20	3.02e+07	3.18	2.55	144	9.85e-06	1.82e-09
garybig	†	†	†	†	†	†
G3_circuit	2.42e+08	12.45	25.70	115	8.83e-06	1.59e-07
hood	1.47e+08	2.67	7.48	46	8.07e-06	3.59e-08
inline_1	4.16e+08	23.40	502.61	1000	8.56e-05	9.02e-09
kyushu	3.98e+08	63.13	46.66	73	9.51e-06	1.29e-07
ldoor	5.43e+08	9.59	71.43	104	9.73e-06	8.39e-08
msdoor	2.77e+08	5.01	226.49	734	9.21e-06	1.53e-07
mstamp-2c	8.72e+08	56.86	76.04	76	9.14e-06	1.66e-08
nastran-b	1.26e+09	88.14	349.85	226	9.83e-06	1.32e-07
nd24k	1.72e+06	13.79	19.04	151	8.21e-06	6.3e-09
oilpan	4.2e+07	0.69	15.37	394	1.22e-06	6.29e-10
parabolic_fem	9.44e+07	3.21	5.71	72	9.38e-06	3.98e-07
pga-rem1	1.7e+08	7.82	45.36	238	9.05e-06	8.64e-09
pga-rem2	6.69e+08	29.09	56.70	71	8.86e-06	6.4e-08
qa8fk	3.71e+07	4.68	1.32	36	3.8e-06	1.36e-08
qa8fm	-1.58e+05	0.27	0.03	3	4.38e-10	1.83e-09
ship_003	5.37e+07	2.24	13.34	165	6.01e-06	8.29e-12
shipsec5	7.71e+07	2.36	10.17	92	9.22e-06	9.91e-12
thermal2	1.86e+08	7.83	43.81	250	8.02e-06	1.59e-08
torso	7.33e+07	9.39	3.19	40	7.49e-06	7.67e-08

Table 23: Table showing the raw data for PROC1:wsmp:ICT:AUTO:RCM:DT3e-3:F4.9_SHIFT-ON

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.71e+08	2.31	0.17	1	1e-09	2.05e-11
af_shell7	7.57e+08	10.90	0.84	1	1e-09	1.9e-11
algor-big	†	†	†	†	†	†
audikw_1	9.19e+09	864.37	5.63	1	1e-09	8.14e-08
bmwcra_1	5.25e+08	10.85	0.42	1	1e-09	2.92e-08
ctu-1	2.91e+09	84.32	2.47	1	1e-09	3.52e-07
ctu-2	2.23e+09	98.77	1.53	1	1e-09	1.33e-07
cfd1	1.5e+08	2.26	0.14	1	1e-09	4.68e-11
cfd2	2.97e+08	6.08	0.26	1	1e-09	1.15e-10
conti20	5.65e+07	0.91	0.05	1	1e-09	4.1e-08
garybig	†	†	†	†	†	†
G3_circuit	9.19e+08	18.44	1.67	1	1e-09	3.98e-11
hood	2.13e+08	2.19	0.32	1	1e-09	1.7e-11
inline_1	1.35e+09	26.24	1.19	1	1e-09	3.61e-08
kyushu	9.11e+09	1331.21	5.27	1	1e-09	5.42e-11
ldoor	1.18e+09	20.91	1.54	1	1e-09	3.69e-10
msdoor	4.09e+08	4.55	0.58	1	1e-09	1.67e-08
mstamp-2c	†	†	†	†	†	†
nastran-b	8.17e+09	533.37	5.52	1	1e-09	3.72e-11
nd24k	2.64e+09	410.81	1.37	1	1e-09	6.8e-09
oilpan	7.98e+07	0.91	0.11	1	1e-09	4.24e-09
parabolic_fem	2.16e+08	2.40	0.50	1	1e-09	4.61e-12
pga-rem1	5.99e+08	9.66	1.45	1	1e-09	3.4e-08
pga-rem2	1.95e+09	39.28	5.34	1	1e-09	9.43e-11
qa8fk	1.86e+08	4.44	0.16	1	1e-09	6.05e-06
qa8fm	1.8e+08	4.06	0.15	1	1e-09	1.53e-15
ship_003	4.96e+08	16.47	0.38	1	1e-09	1.07e-07
shipsec5	4.06e+08	12.53	0.39	1	1e-09	1.41e-08
thermal2	4.45e+08	5.31	1.13	1	1e-09	5.02e-11
torso	6.63e+08	23.91	0.52	1	1e-09	1.94e-11

Table 24: Table showing the raw data for PROC1:wsmp:NONE:DIRECT:ND:ALL:NONE

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	3.5e+07	0.01	2.35	1000	1.5	0.406
af_shell7	1.18e+08	0.00	8.51	829	9.72e-06	2.69e-07
algor-big	5.1e+08	0.01	48.84	654	9.17e-06	3.88e-10
audikw_1	4.74e+08	0.00	63.61	1000	6.7	0.872
bmwcra_1	6.44e+07	0.00	4.54	1000	3.63	0.265
ctu-1	4.65e+08	0.00	65.42	1000	-1	0.987
ctu-2	1.73e+08	0.00	20.68	1000	-1	0.984
cfd1	1.23e+07	0.00	0.56	468	9.66e-06	7.29e-08
cfd2	2.1e+07	0.00	1.90	1000	0.00121	5.62e-05
conti20	9.79e+06	0.00	0.84	1000	2.46	0.948
garybig	†	†	†	†	†	†
G3_circuit	8.71e+07	0.00	7.93	904	9.97e-06	1.22e-06
hood	6.95e+07	0.01	4.65	1000	0.000887	0.00257
inline_1	2.3e+08	0.00	28.62	1000	37.1	0.49
kyushu	1.78e+08	0.01	10.40	521	9.93e-06	7.36e-07
ldoor	3.02e+08	0.00	36.20	1000	0.000741	0.00328
msdoor	1.31e+08	0.00	12.55	1000	0.0355	0.817
mstamp-2c	4.28e+08	0.01	11.28	185	9.45e-06	1.22e-06
nastran-b	6.97e+08	0.00	100.05	1000	0.00164	2.42e-05
nd24k	1.4e+08	0.01	18.81	1000	18.1	0.754
oilpan	2.28e+07	0.01	1.58	1000	0.0185	0.912
parabolic_fem	3.57e+07	0.00	2.43	742	9.99e-06	1.84e-07
pga-rem1	8.18e+07	0.01	8.15	1000	0.0451	0.000583
pga-rem2	3.33e+08	0.01	22.73	565	9.99e-06	6.54e-07
qa8fk	1.11e+07	0.00	0.05	26	1.05e+04	2.24
qa8fm	1.11e+07	0.01	0.01	9	9.16e-06	3.19e-05
ship_003	4.87e+07	0.00	3.59	1000	0.00399	0.00516
shipsec5	6.31e+07	0.00	4.39	1000	6.3e-05	1.64e-06
thermal2	8.33e+07	0.00	7.54	1000	0.00203	4.3e-06
torso	2.37e+07	0.01	0.44	177	9.87e-06	8.95e-08

Table 25: Table showing the raw data for PROC16:petsc:IC(k):CG:RCM:LF0:F1

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	5.45e+07	0.10	2.42	1000	3.55e-05	2.61e-06
af_shell7	1.98e+08	0.36	7.34	912	9.77e-06	3.08e-07
algor-big	9.91e+08	4.81	64.85	827	9.92e-06	4.92e-10
audikw_1	1.35e+09	11.44	60.60	1000	3.85	0.241
bmwcra_1	1.26e+08	0.45	5.39	1000	24	0.735
ctu-1	1.31e+09	7.43	59.53	1000	-1	0.983
ctu-2	4.4e+08	2.20	13.86	1000	-1	0.957
cfd1	6.81e+07	0.37	2.71	738	9.65e-06	6.12e-08
cfd2	1.23e+08	0.57	4.10	1000	0.00676	0.000361
conti20	3.06e+07	0.20	2.26	1000	7.08	0.462
garybig	1.11e+10	359.48	587.41	483	572	5.58e+03
G3_circuit	4.44e+08	2.05	31.45	918	9.78e-06	8.24e-07
hood	1.12e+08	0.21	1.70	378	9.85e-06	5.98e-06
inline_1	4.02e+08	1.34	21.24	1000	30	0.919
kyushu	9.97e+08	6.06	39.60	1000	2.33e-05	2.37e-06
ldoor	4.98e+08	1.21	27.59	1000	0.0001	0.000275
msdoor	2.18e+08	0.49	8.40	1000	0.0353	0.435
mstamp-2c	7.78e+08	2.84	10.12	178	8.95e-06	5.72e-06
nastran-b	1.29e+09	6.10	95.32	1000	0.162	0.00656
nd24k	4.99e+08	18.43	29.42	1000	2.49	0.0257
oilpan	3.8e+07	0.08	1.74	1000	0.0112	0.264
parabolic_fem	1.5e+08	0.67	7.23	828	9.94e-06	1.58e-07
pga-rem1	4.18e+08	1.95	31.57	1000	0.131	0.0012
pga-rem2	1.66e+09	8.50	116.66	719	9.86e-06	6.86e-07
qa8fk	3.93e+07	0.23	0.11	22	1.03e+04	2.17
qa8fm	3.93e+07	0.24	0.04	9	8.16e-06	2.95e-05
ship_003	8.48e+07	0.20	3.77	1000	0.00105	0.00828
shipsec5	1.1e+08	0.27	2.65	577	9.56e-06	5.48e-07
thermal2	3.55e+08	1.61	23.35	1000	0.00462	4.79e-05
torso	1.95e+08	0.82	2.32	324	9.73e-06	1.09e-07

Table 26: Table showing the raw data for PROC16:petsc:BSolve:CG:RCM:ALL:NONE

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.42e+07	0.10	7.66	1000	1.21	0.102
af_shell7	7.88e+07	0.38	30.02	1000	0.676	0.0142
algor-big	1.68e+08	1.45	109.01	1000	0.0342	0.000732
audikw_1	1.47e+08	1.27	98.75	1000	7.19	0.0466
bmwcra_1	2.33e+07	0.18	13.95	1000	8.48	0.0217
ctu-1	1.59e+08	1.28	96.83	1000	5.46	0.000125
ctu-2	6e+07	0.46	36.74	1000	7.83	3.98e-05
cfd1	1.11e+07	0.04	3.74	1000	1.01	0.00029
cfd2	1.93e+07	0.08	6.32	1000	1	0.000207
conti20	3.28e+06	0.03	2.48	1000	1.03	0.00596
garybig	†	†	†	†	†	†
G3_circuit	2.46e+08	0.65	23.49	488	9.82e-06	1.18e-06
hood	3.45e+07	0.20	15.71	1000	0.000429	0.00229
inline_1	7.87e+07	0.63	49.72	1000	1.18	0.0062
kyushu	1.55e+08	0.62	51.47	1000	1.66	4.29e-07
ldoor	1.49e+08	0.89	68.66	1000	0.00298	0.00982
msdoor	6.5e+07	0.38	30.01	1000	0.0563	0.0404
mstamp-2c	1.41e+08	1.22	43.66	478	9.98e-06	0.000102
nastran-b	2.35e+08	1.92	145.94	1000	2.44	0.64
nd24k	1.14e+07	0.35	31.46	1000	2.96	0.000776
oilpan	1.16e+07	0.06	5.36	1000	0.0405	0.031
parabolic_fem	8.2e+07	0.22	5.62	321	9.75e-06	2.55e-08
pga-rem1	2.28e+08	0.60	21.46	486	9.38e-06	2.7e-08
pga-rem2	9.14e+08	2.43	33.84	184	9.6e-06	2.75e-07
qa8fk	1.04e+07	0.04	0.11	32	7.17e+04	4.69e-06
qa8fm	1.04e+07	0.04	0.04	9	8.7e-06	6.77e-06
ship_003	1.91e+07	0.14	11.34	1000	0.0342	1.03
shipsec5	2.82e+07	0.18	14.40	1000	0.00672	0.000105
thermal2	1.91e+08	0.52	25.26	636	9.93e-06	3.55e-09
torso	3.14e+07	0.11	1.18	139	8.39e-06	7.29e-09

Table 27: Table showing the raw data for PROC16:tril:IC(k):CG:RCM:LF8:NONE

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.23e+07	0.15	24.80	828	9.84e-06	7e-05
af_shell7	6.32e+07	0.49	35.42	353	9.74e-06	5.1e-05
algor-big	1.98e+08	2.48	169.71	281	8.72e-06	3.62e-07
audikw_1	1.72e+08	2.40	521.97	1000	6.6	0.0429
bmwcra_1	2.53e+07	0.34	33.94	606	8.41e-06	2.9e-07
ctu-1	†	†	†	†	†	†
ctu-2	6.46e+07	0.81	162.62	1000	9.86	3.91e-05
cfd1	1.52e+07	0.10	1.25	105	8.75e-06	6.15e-08
cfd2	2.43e+07	0.15	10.08	527	9.83e-06	1.76e-07
conti20	5.48e+06	0.08	10.70	1000	14.9	0.00298
garybig	†	†	†	†	†	†
G3_circuit	3.19e+08	0.96	19.65	201	9.25e-06	1.05e-06
hood	2.56e+07	0.29	18.98	321	9.61e-06	0.000292
inline_1	8.2e+07	1.17	236.03	1000	14	0.00408
kyushu	1.88e+08	1.13	35.44	220	9.75e-06	4.88e-10
ldoor	1.06e+08	1.17	210.45	673	9.99e-06	0.000319
msdoor	4.82e+07	0.53	114.40	1000	0.0217	0.0241
mstamp-2c	1.65e+08	2.09	37.53	75	9.07e-06	0.000142
nastran-b	2.55e+08	3.44	695.63	853	9.78e-06	0.000823
nd24k	†	†	†	†	†	†
oilpan	9.67e+06	0.10	19.78	1000	0.00711	0.00911
parabolic_fem	8.35e+07	0.28	0.91	24	6.37e-06	1.23e-08
pga-rem1	2.5e+08	0.78	86.08	948	9.44e-06	2.63e-08
pga-rem2	†	†	†	†	†	†
qa8fk	1.54e+07	0.09	0.06	5	2.06e+04	3.27e-06
qa8fm	1.54e+07	0.10	0.06	4	1.51e-06	2.61e-06
ship_003	3.67e+07	0.34	39.01	850	9.81e-09	0.0013
shipsec5	3.88e+07	0.32	17.78	315	9.84e-06	4.25e-06
thermal2	1.75e+08	0.59	3.30	39	7.03e-06	2.3e-09
torso	4.39e+07	0.22	1.49	63	9.41e-06	2.99e-09

Table 28: Table showing the raw data for PROC16:tril:ML:CG:NONE:SA:SGS_SS2_UC

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.91e+08	0.19	3.15	1000	3.76	0.995
af_shell7	5.5e+08	0.45	12.60	616	9.57e-06	1.03e-07
algor-big	4.58e+09	2030.11	120.18	391	9.14e-06	3.66e-10
audikw_1	3.93e+09	6.55	112.09	717	9.54e-06	2.47e-09
bmwcra_1	6.01e+08	0.93	12.52	1000	1.31	0.272
ctu-1	†	†	†	†	†	†
ctu-2	1.6e+09	2.20	113.62	1000	-1	1
cfd1	7.76e+07	0.08	0.70	395	9.68e-06	3.26e-08
cfd2	1.35e+08	0.12	2.90	1000	10.7	0.191
conti20	7.03e+07	0.12	1.19	1000	6.3	0.985
garybig	†	†	†	†	†	†
G3_circuit	3.05e+08	0.40	6.94	670	9.85e-06	5.01e-07
hood	3.54e+08	0.35	88.69	1000	-1	0.0093
inline_1	1.97e+09	3.03	110.81	1000	-1	0.25
kyushu	1.08e+09	1.19	45.17	600	-1	5.92e-07
ldoor	1.04e+09	1.40	116.54	1000	-1	0.0679
msdoor	6.42e+08	0.67	25.02	1000	0.279	0.983
mstamp-2c	3.75e+09	5.19	21.48	147	9.33e-06	9.34e-07
nastran-b	6.27e+09	8.93	173.61	756	9.82e-06	9.27e-08
nd24k	1.19e+09	11.32	41.77	1000	0.162	0.000422
oilpan	8.1e+07	0.10	1.96	1000	0.0115	0.992
parabolic_fem	1.69e+08	0.14	1.79	532	9.78e-06	9.22e-08
pga-rem1	2.85e+08	0.36	8.76	941	9.99e-06	4.36e-08
pga-rem2	1.15e+09	1.50	22.83	347	9.76e-06	5.66e-07
qa8fk	7.27e+07	0.07	0.17	111	8.15e-06	4.09e-08
qa8fm	7.27e+07	0.06	0.02	12	5.09e-07	1.56e-06
ship_003	2.69e+08	58.42	7.62	854	8.94e-06	1.32e-06
shipsec5	1.96e+08	0.24	3.46	607	9.83e-06	4.59e-07
thermal2	3.94e+08	0.33	8.79	1000	1.14e-05	1.01e-08
torso	1.75e+08	0.12	0.38	124	9.63e-06	4.02e-08

Table 29: Table showing the raw data for PROC16:hypre:IC(k):CG:RCM:LF1:NzINF

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.78e+07	0.06	14.91	1000	0.000538	4.71e-05
af_shell7	9.75e+07	0.20	5.96	110	9.88e-06	8.7e-08
algor-big	3.44e+08	0.99	58.27	152	9.94e-06	3.46e-10
audikw_1	1.97e+08	0.71	318.81	1000	0.0121	0.000194
bmwcra_1	4.96e+07	0.12	17.15	624	8.91e-06	1.32e-10
ctu-1	3.37e+08	0.71	352.66	1000	-1	0.905
ctu-2	9.47e+07	0.26	93.27	1000	-1	0.86
cfd1	1.36e+07	0.06	1.25	163	9.68e-06	2.8e-08
cfd2	2.31e+07	0.08	11.25	946	9.99e-06	3.79e-07
conti20	6.34e+06	0.04	6.31	1000	0.0432	0.000354
garybig	†	†	†	†	†	†
G3_circuit	1.73e+08	0.28	6.32	119	8.17e-06	2.3e-07
hood	4.76e+07	0.12	9.32	309	2.08e-06	9.67e-07
inline_1	1.45e+08	0.36	144.63	1000	23.6	0.253
kyushu	1.68e+08	0.56	37.70	336	9.33e-06	7.43e-07
ldoor	1.78e+08	0.45	101.25	478	9.85e-06	1.69e-05
msdoor	8.62e+07	0.20	62.25	1000	0.00261	0.00545
mstamp-2c	2.69e+08	0.86	13.46	40	9.79e-06	8.05e-07
nastran-b	4.54e+08	1.26	433.60	770	9.91e-06	1.88e-07
nd24k	6.73e+07	0.24	77.23	1000	0.371	0.00219
oilpan	1.09e+07	0.06	11.43	975	9.93e-06	0.000446
parabolic_fem	4.2e+07	0.13	1.16	60	8.49e-06	5.69e-08
pga-rem1	1.74e+08	0.25	25.79	447	1e-05	3.82e-08
pga-rem2	7.08e+08	1.38	49.87	181	9.7e-06	5.95e-07
qa8fk	1.27e+07	0.06	0.24	33	6.66e-06	1.19e-08
qa8fm	2.76e+06	0.02	0.05	26	2.2e-06	9.99e-06
ship_003	3.64e+07	0.09	14.52	774	9.85e-06	0.000591
shipsec5	4.65e+07	0.12	17.81	622	2.1e-06	8.86e-08
thermal2	1.24e+08	0.22	6.83	136	9.72e-06	5.27e-09
torso	2.84e+07	0.09	0.84	55	9.55e-06	6.18e-08

Table 30: Table showing the raw data for PROC16:hypre:AMG:CG:NONE:PMIS:AGG10_ST0.9

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	3.09e+08	0.36	1.54	672	9.87e-06	3.51e-07
af_shell7	4.41e+08	1.03	12.07	899	9.99e-06	4.15e-07
algor-big	7.65e+08	6.13	57.10	763	9.34e-06	2.73e-10
audikw_1	8.95e+08	12.41	84.21	1000	0.0206	0.00023
bmwcra_1	3.43e+08	0.80	4.29	1000	10.7	0.263
ctu-1	†	†	†	†	†	†
ctu-2	6.49e+08	10.99	36.18	1000	-1	0.851
cfd1	3.11e+08	0.35	0.88	433	9.96e-06	5.75e-08
cfd2	3.42e+08	0.72	2.10	651	9.82e-06	1.92e-07
conti20	3.04e+08	0.85	3.66	1000	0.0672	0.000471
garybig	†	†	†	†	†	†
G3_circuit	5.01e+08	0.74	9.52	711	9.74e-06	1.47e-06
hood	3.8e+08	1.18	0.62	103	1.94e-06	5.77e-07
inline_1	5.38e+08	2.78	29.99	1000	41.7	0.534
kyushu	5.95e+08	2.69	25.82	776	9.93e-06	9.28e-07
ldoor	7.28e+08	5.29	20.63	414	9.75e-06	1.2e-05
msdoor	4.8e+08	2.48	19.10	1000	0.000837	0.000928
mstamp-2c	6.8e+08	4.99	10.33	170	9.53e-06	8.35e-06
nastran-b	1.08e+09	9.43	116.82	1000	0.00445	8e-05
nd24k	3.07e+08	0.44	9.67	1000	0.0115	0.000219
oilpan	3.06e+08	0.25	1.53	933	9.42e-06	6.15e-07
parabolic_fem	3.58e+08	0.30	2.70	558	1e-05	4.96e-07
pga-rem1	4.64e+08	0.59	11.30	1000	0.00184	3.96e-05
pga-rem2	9.72e+08	2.83	45.01	640	9.92e-06	5.87e-07
qa8fk	3.06e+08	0.06	0.28	408	9.55e-06	1.72e-08
qa8fm	3.06e+08	0.09	0.01	7	1.43e-07	4.6e-07
ship_003	3.09e+08	0.23	1.99	875	9.02e-06	2.4e-06
shipsec5	3.15e+08	0.34	1.04	308	1.71e-06	1.23e-07
thermal2	4.87e+08	0.72	12.14	1000	0.00062	5.13e-06
torso	3.31e+08	0.24	0.60	203	9.56e-06	9.98e-08

Table 31: Table showing the raw data for PROC16:hypre:PSAILS:CG:ND:PLev1:Th0.1_Flt0.001

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.71e+08	0.25	0.02	1	1e-09	1.65e-11
af_shell7	7.77e+08	1.30	0.13	1	1e-09	2.75e-11
algor-big	†	†	†	†	†	†
audikw_1	†	†	†	†	†	†
bmwcra_1	5.28e+08	1.10	0.07	1	1e-09	2.4e-08
ctu-1	3.07e+09	10.86	0.62	1	1e-09	2.93e-07
ctu-2	2.2e+09	7.10	0.21	1	1e-09	1.02e-07
cfd1	1.63e+08	0.34	0.03	1	1e-09	5.92e-11
cfd2	3.01e+08	0.70	0.05	1	1e-09	1.15e-10
conti20	5.95e+07	0.13	0.01	1	1e-09	3.54e-08
garybig	†	†	†	†	†	†
G3_circuit	8.94e+08	3.43	0.23	1	1e-09	0
hood	2.59e+08	0.30	0.04	1	1e-09	2.13e-11
inline_1	1.42e+09	3.42	0.23	1	1e-09	2.68e-08
kyushu	†	†	†	†	†	†
ldoor	1.32e+09	2.09	0.21	1	1e-09	6.04e-10
msdoor	4.77e+08	0.64	0.09	1	1e-09	1.92e-08
mstamp-2c	†	†	†	†	†	†
nastran-b	8.8e+09	66.12	0.83	1	1e-09	4.84e-11
nd24k	2.58e+09	35.32	0.25	1	1e-09	9.15e-09
oilpan	8.77e+07	0.12	0.02	1	1e-09	3.85e-09
parabolic_fem	2.21e+08	0.32	0.08	1	1e-09	4.44e-12
pga-rem1	5.91e+08	1.15	0.25	1	1e-09	2e-08
pga-rem2	1.97e+09	4.70	0.83	1	1e-09	1.2e-10
qa8fk	1.82e+08	0.40	0.02	1	1e-09	5.86e-06
qa8fm	1.76e+08	0.38	0.03	1	1e-09	1.58e-15
ship_003	5.11e+08	1.48	0.06	1	1e-09	6.76e-08
shipsec5	4.63e+08	1.12	0.06	1	1e-09	1.18e-08
thermal2	4.53e+08	0.72	0.17	1	1e-09	5.39e-11
torso	6.67e+08	2.11	0.09	1	1e-09	4.1e-11

Table 32: Table showing the raw data for PROC16:wsmp:NONE:DIRECT:ND:ALL:NONE

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	3.36e+07	0.01	0.75	1000	0.0925	0.00841
af_shell7	1.16e+08	0.01	1.91	915	9.81e-06	1.74e-07
algor-big	4.9e+08	0.01	9.91	710	9.46e-06	2.48e-10
audikw_1	4.56e+08	0.01	11.85	1000	6.43	0.961
bmwcra_1	6.02e+07	0.01	1.24	1000	23.1	0.224
ctu-1	4.57e+08	0.01	11.18	1000	-1	0.99
ctu-2	1.65e+08	0.01	3.07	1000	-1	0.988
cfd1	1.17e+07	0.00	0.28	512	9.63e-06	5.84e-08
cfd2	2.03e+07	0.01	0.71	1000	0.000808	1.69e-05
conti20	8.44e+06	0.01	0.35	1000	3.43	0.908
garybig	2.53e+09	0.02	10.97	124	45.6	244
G3_circuit	8.71e+07	0.01	2.11	921	9.96e-06	7.12e-07
hood	6.78e+07	0.01	1.26	1000	0.00138	0.00404
inline_1	2.24e+08	0.01	4.16	1000	38	0.534
kyushu	1.74e+08	0.01	2.67	671	9.91e-06	5.52e-07
ldoor	2.99e+08	0.01	5.18	1000	0.00347	0.0218
msdoor	1.29e+08	0.01	2.25	1000	0.0251	0.945
mstamp-2c	4.1e+08	0.01	2.14	205	9.91e-06	9.83e-07
nastran-b	6.82e+08	0.01	21.41	1000	0.00419	5.98e-05
nd24k	1.11e+08	0.01	3.20	1000	21	0.562
oilpan	2.19e+07	0.01	0.51	1000	0.0279	0.952
parabolic_fem	3.58e+07	0.01	0.68	704	9.74e-06	2.12e-07
pga-rem1	8.19e+07	0.01	2.15	1000	0.058	0.000732
pga-rem2	3.33e+08	0.01	5.08	579	9.79e-06	5.95e-07
qa8fk	1.06e+07	0.01	0.02	29	1.17e+04	2.28
qa8fm	1.06e+07	0.01	0.01	11	3.66e-06	1.25e-05
ship_003	4.59e+07	0.01	1.14	1000	0.00165	0.00224
shipsec5	6.06e+07	0.01	1.22	1000	0.000409	6.96e-06
thermal2	8.34e+07	0.00	2.03	1000	0.00188	4.69e-06
torso	2.32e+07	0.00	0.16	203	8.87e-06	7.55e-08

Table 33: Table showing the raw data for PROC64:petsc:IC(k):CG:RCM:LF0:F1

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	5.6e+07	0.16	1.09	1000	9.94e-05	5.09e-06
af_shell7	2.01e+08	0.12	2.33	909	9.98e-06	3.58e-07
algor-big	1.03e+09	1.65	13.13	832	9.95e-06	5.23e-10
audikw_1	1.41e+09	2.82	12.87	1000	3.12	0.276
bmwcra_1	1.41e+08	0.27	3.68	1000	30.5	0.547
ctu-1	1.36e+09	1.98	11.75	1000	-1	0.984
ctu-2	4.71e+08	0.64	4.51	1000	-1	0.958
cfd1	7.94e+07	0.22	3.81	974	9.7e-06	3.53e-08
cfd2	1.54e+08	0.33	3.55	1000	0.0017	0.000153
conti20	3.83e+07	0.26	2.26	1000	7	0.519
garybig	1.11e+10	171.25	143.25	485	7.64	73.3
G3_circuit	4.52e+08	0.53	7.34	922	9.96e-06	6.3e-07
hood	1.16e+08	0.11	0.84	474	9.83e-06	6.67e-06
inline_1	4.12e+08	0.43	6.02	1000	26	0.903
kyushu	1e+09	1.45	12.19	1000	4.7e-05	4.6e-06
ldoor	5.02e+08	0.32	5.21	1000	0.000108	0.000296
msdoor	2.21e+08	0.17	2.82	1000	0.0431	0.472
mstamp-2c	7.8e+08	0.70	2.44	202	9.98e-06	3.25e-06
nastran-b	1.28e+09	1.42	17.19	1000	0.293	0.0112
nd24k	6.38e+08	9.65	15.67	1000	2.97	0.0167
oilpan	4.03e+07	0.07	0.91	1000	0.0149	0.251
parabolic_fem	1.55e+08	0.21	2.47	817	9.88e-06	1.46e-07
pga-rem1	4.24e+08	0.48	7.39	1000	0.0877	0.000964
pga-rem2	1.67e+09	2.09	23.94	723	9.92e-06	6.97e-07
qa8fk	4.89e+07	0.20	0.09	23	5.68e+04	8.94
qa8fm	4.89e+07	0.19	0.04	9	5.21e-06	1.96e-05
ship_003	9.8e+07	0.42	2.07	1000	0.00176	0.00217
shipsec5	1.13e+08	0.16	1.15	597	9.92e-06	5.24e-07
thermal2	3.61e+08	0.43	5.96	1000	0.00205	2.45e-05
torso	2.36e+08	0.56	2.12	303	9.92e-06	1.22e-07

Table 34: Table showing the raw data for PROC64:petsc:BSolve:CG:RCM:ALL:NONE

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.15e+07	0.03	2.14	1000	1.82	0.102
af_shell7	6.29e+07	0.09	7.28	1000	19.6	0.0629
algor-big	1.34e+08	0.34	26.99	1000	0.237	0.00123
audikw_1	1.18e+08	0.29	24.81	1000	1.02	0.0805
bmwcra_1	1.89e+07	0.05	3.85	1000	2.81	0.0217
ctu-1	1.27e+08	0.30	24.17	1000	5.29	0.000125
ctu-2	4.8e+07	0.11	9.26	1000	8.17	3.98e-05
cfd1	9.09e+06	0.02	1.18	1000	1.13	0.00029
cfd2	1.56e+07	0.02	1.76	1000	1	0.000207
conti20	2.94e+06	0.01	0.95	1000	1.18	0.00596
garybig	4.86e+09	3.83	16.48	62	2.47	0.0357
G3_circuit	1.96e+08	0.15	6.28	561	9.73e-06	9.45e-07
hood	2.77e+07	0.05	4.11	1000	1	0.1
inline_1	6.3e+07	0.15	12.29	1000	1.86	0.0062
kyushu	1.23e+08	0.14	12.71	1000	1	5.06e-07
ldoor	1.18e+08	0.20	16.71	1000	1	0.0695
msdoor	5.19e+07	0.09	7.53	1000	1	0.0417
mstamp-2c	1.12e+08	0.28	16.72	730	9.96e-06	0.000152
nastran-b	1.88e+08	0.45	36.91	1000	1.01	1.94
nd24k	9.61e+06	0.08	8.71	1000	2.59	0.000776
oilpan	9.5e+06	0.02	1.55	1000	0.0211	0.031
parabolic_fem	6.54e+07	0.06	1.74	405	9.68e-06	1.84e-08
pga-rem1	1.82e+08	0.14	6.56	632	9.77e-06	3.4e-08
pga-rem2	7.3e+08	0.56	9.41	222	9.65e-06	3.08e-07
qa8fk	8.53e+06	0.02	0.04	36	8.58e+04	6.34e-06
qa8fm	8.53e+06	0.02	0.02	11	4.41e-06	4.78e-06
ship_003	1.55e+07	0.04	3.08	1000	0.0133	1.13
shipsec5	2.27e+07	0.04	3.86	1000	0.0149	0.000277
thermal2	1.52e+08	0.13	7.83	816	9.9e-06	3.48e-09
torso	2.52e+07	0.03	0.47	189	8.24e-06	9.04e-09

Table 35: Table showing the raw data for PROC64:tril:IC(k):CG:RCM:LF6:NONE

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.67e+07	0.34	8.44	743	9.94e-06	7.11e-05
af_shell7	6.87e+07	0.49	8.71	262	9.85e-06	7.23e-05
algor-big	2.2e+08	1.59	40.94	263	7.74e-09	4.72e-08
audikw_1	1.87e+08	1.66	134.27	1000	27	0.0786
bmwcra_1	3.26e+07	0.48	11.30	526	9.76e-06	2.86e-07
ctu-1	†	†	†	†	†	†
ctu-2	7.48e+07	0.70	48.66	1000	18.1	4.04e-05
cfd1	2.18e+07	0.53	0.99	127	9.61e-06	4.39e-08
cfd2	3.09e+07	0.84	6.88	700	9.09e-06	1.4e-07
conti20	1.12e+07	0.98	8.26	1000	6.51	0.00582
garybig	†	†	†	†	†	†
G3_circuit	3.3e+08	1.00	6.63	184	9.97e-06	1.06e-06
hood	3.05e+07	0.67	7.83	356	9.21e-06	0.000286
inline_1	8.97e+07	1.35	64.98	1000	17.2	0.00355
kyushu	2.1e+08	0.75	11.04	195	9.28e-06	4.61e-10
ldoor	1.12e+08	0.97	73.77	902	9.94e-06	0.000444
msdoor	5.34e+07	0.72	38.40	1000	0.101	0.0255
mstamp-2c	1.88e+08	1.52	8.85	67	9.52e-06	0.000121
nastran-b	2.7e+08	2.15	182.94	816	9.76e-06	0.00091
nd24k	†	†	†	†	†	†
oilpan	1.39e+07	0.98	8.72	1000	0.00351	0.00785
parabolic_fem	8.87e+07	0.60	0.41	23	7.61e-06	1.51e-08
pga-rem1	2.57e+08	1.05	26.11	798	9.9e-06	4.18e-08
pga-rem2	†	†	†	†	†	†
qa8fk	2.09e+07	0.74	0.03	4	3.56e+03	1.64e-06
qa8fm	2.1e+07	0.71	0.04	4	2.06e-06	2.11e-06
ship_003	4.31e+07	0.84	13.67	802	8.02e-09	0.00105
shipsec5	4.75e+07	0.93	5.86	256	9.97e-06	3.88e-06
thermal2	1.81e+08	0.80	1.17	39	5.4e-06	2.01e-09
torso	5.23e+07	0.68	1.82	92	7.79e-06	6.75e-09

Table 36: Table showing the raw data for PROC64:tril:ML:CG:NONE:SA:SGS_SS3_UCMIS

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.94e+08	0.05	0.84	1000	4.53	0.957
af_shell7	5.52e+08	0.11	2.10	716	9.9e-06	1.55e-07
algor-big	4.67e+09	1316.77	42.55	410	8.66e-06	3.55e-10
audikw_1	3.93e+09	1.52	32.65	1000	0.00666	2.31e-05
bmwcra_1	4.11e+08	0.22	2.15	1000	6.63	0.171
ctu-1	4.24e+09	1.91	33.23	1000	-1	1
ctu-2	1.5e+09	0.50	5.77	1000	-1	1
cfd1	8e+07	0.02	0.20	395	9.48e-06	3.45e-08
cfd2	1.37e+08	0.03	0.80	1000	11.5	0.22
conti20	4.62e+07	0.02	0.34	1000	5.74	0.534
garybig	8.16e+09	2.76	130.18	1000	0.000552	0.00479
G3_circuit	3.08e+08	0.10	1.58	629	9.74e-06	1.22e-06
hood	2.43e+08	0.08	1.70	1000	0.0102	0.00719
inline_1	1.82e+09	0.73	8.95	1000	30.6	0.379
kyushu	1.08e+09	0.28	7.22	1000	0.000146	9.46e-06
ldoor	1.04e+09	0.35	9.96	1000	0.0178	0.0679
msdoor	5.36e+08	0.16	3.26	1000	0.192	0.983
mstamp-2c	3.76e+09	1.19	5.20	174	8.8e-06	7.31e-07
nastran-b	5.96e+09	2.19	46.42	891	9.46e-06	7.85e-08
nd24k	1.04e+09	1.42	3.62	1000	1.03	0.00358
oilpan	6.18e+07	0.03	0.58	1000	0.0303	0.984
parabolic_fem	1.71e+08	0.04	0.51	535	9.63e-06	9.43e-08
pga-rem1	2.87e+08	0.10	2.26	990	9.96e-06	4.24e-08
pga-rem2	1.15e+09	0.38	3.53	356	9.95e-06	5.36e-07
qa8fk	7.51e+07	0.02	0.06	120	7.75e-06	2e-07
qa8fm	7.51e+07	0.02	0.01	13	6.7e-07	2.21e-06
ship_003	2.87e+08	46.50	4.23	957	9.84e-06	9.83e-07
shipsec5	1.99e+08	0.06	0.77	548	5.64e-06	2.79e-07
thermal2	3.96e+08	0.09	2.09	1000	1.83e-05	1.7e-08
torso	1.25e+08	0.03	0.16	163	9.53e-06	5.21e-08

Table 37: Table showing the raw data for PROC64:hypre:IC(k):CG:RCM:LF1:NzINF

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.65e+07	0.05	5.83	1000	0.000639	7.25e-05
af_shell7	9.35e+07	0.10	2.38	141	9.81e-06	1.16e-07
algor-big	3.12e+08	0.37	9.58	156	8.26e-06	2.76e-10
audikw_1	1.33e+08	0.28	50.93	1000	22.9	0.927
bmwcra_1	4.23e+07	0.09	7.69	718	9.94e-06	1.71e-10
ctu-1	2.14e+08	0.28	51.71	1000	-1	0.993
ctu-2	7.92e+07	0.14	21.81	1000	-1	0.991
cfd1	1.09e+07	0.06	1.04	186	9.67e-06	3.36e-08
cfd2	1.94e+07	0.05	5.25	935	9.97e-06	5.63e-07
conti20	7.28e+06	0.08	8.07	1000	6.65	0.91
garybig	4.71e+09	2.38	382.63	590	9.87e-06	1.67e-05
G3_circuit	1.63e+08	0.12	2.02	121	9.35e-06	2.56e-07
hood	5.22e+07	0.08	3.40	329	2.34e-06	1.36e-06
inline_1	1.17e+08	0.13	24.78	1000	27.1	0.37
kyushu	1.55e+08	0.24	9.99	322	9.86e-06	6.89e-07
ldoor	1.61e+08	0.17	20.60	584	9.91e-06	1.76e-05
msdoor	8.07e+07	0.11	18.68	1000	0.00609	0.0123
mstamp-2c	2.59e+08	0.35	2.66	44	9.29e-06	1.07e-06
nastran-b	3.64e+08	0.41	88.61	899	9.99e-06	1.53e-07
nd24k	5.14e+07	0.17	26.69	1000	14.4	0.0392
oilpan	1.27e+07	0.05	5.62	1000	3.46e-05	0.00277
parabolic_fem	4.24e+07	0.06	0.50	61	8.41e-06	3.71e-08
pga-rem1	1.66e+08	0.11	8.83	512	9.86e-06	7.8e-08
pga-rem2	6.87e+08	0.42	12.05	185	9.52e-06	4.9e-07
qa8fk	8.09e+06	0.05	0.18	34	9.91e-06	2.52e-08
qa8fm	3.74e+06	0.01	0.03	26	2.2e-06	9.99e-06
ship_003	3.59e+07	0.10	8.72	819	9.89e-06	0.00047
shipsec5	3.98e+07	0.11	7.27	543	2.6e-06	1.88e-07
thermal2	1.06e+08	0.09	1.64	120	9.83e-06	5.53e-09
torso	2.73e+07	0.10	0.78	67	9.4e-06	2.95e-08

Table 38: Table showing the raw data for PROC64:hypre:AMG:CG:NONE:PMIS:AGG10_ST0.7

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.22e+09	0.12	0.49	660	9.79e-06	3.25e-07
af_shell7	1.3e+09	0.31	2.23	894	9.91e-06	5.83e-07
algor-big	1.65e+09	1.72	11.74	750	9.91e-06	5.02e-10
audikw_1	1.78e+09	3.69	18.39	1000	0.0267	0.0003
bmwcra_1	1.22e+09	0.27	1.45	1000	6.49	0.261
ctu-1	2.35e+09	12.39	31.54	1000	-1	0.947
ctu-2	1.51e+09	5.21	6.49	1000	-1	0.856
cfd1	1.22e+09	0.13	0.34	425	9.74e-06	8.93e-08
cfd2	1.22e+09	0.21	0.72	663	9.68e-06	2.48e-07
conti20	1.22e+09	0.22	0.59	1000	0.139	0.00113
garybig	5.46e+09	5.72	120.57	1000	0.0131	0.186
G3_circuit	1.4e+09	0.28	2.51	708	9.87e-06	1.22e-06
hood	1.22e+09	0.46	0.19	104	2.16e-06	6.47e-07
inline_1	1.37e+09	0.78	4.15	1000	43	0.538
kyushu	1.47e+09	0.77	4.59	778	9.95e-06	9.77e-07
ldoor	1.59e+09	1.68	3.10	415	9.95e-06	1.19e-05
msdoor	1.36e+09	0.75	3.13	1000	0.0008	0.000936
mstamp-2c	1.54e+09	1.42	1.89	168	9.34e-06	7.83e-06
nastran-b	1.94e+09	2.64	27.27	1000	0.00457	8.2e-05
nd24k	1.22e+09	0.21	1.59	1000	0.0137	0.000356
oilpan	1.22e+09	0.09	0.56	942	9.51e-06	5.35e-07
parabolic_fem	1.23e+09	0.12	0.81	576	9.84e-06	1.54e-07
pga-rem1	1.27e+09	0.25	2.96	1000	0.00206	4.54e-05
pga-rem2	1.86e+09	0.93	8.61	641	9.97e-06	5.81e-07
qa8fk	1.22e+09	0.03	0.08	238	9.55e-06	1.1e-06
qa8fm	1.22e+09	0.05	0.01	7	1.52e-07	4.87e-07
ship_003	1.22e+09	0.09	0.65	845	9.52e-06	2.37e-06
shipsec5	1.22e+09	0.17	0.37	342	2.2e-06	1.72e-07
thermal2	1.38e+09	0.26	3.20	1000	0.00688	2.73e-05
torso	1.22e+09	0.10	0.26	203	9.77e-06	1.06e-07

Table 39: Table showing the raw data for PROC64:hypre:PSAILS:CG:ND:PLev1:Th0.1_Flt0.001

Matrix	FMem	FTime	STime	nIter	rNorm	eNorm
90153	1.72e+08	0.12	0.01	1	1e-09	1.8e-11
af_shell7	7.76e+08	0.41	0.04	1	1e-09	1.83e-11
algor-big	3.18e+10	89.89	0.51	1	1e-09	1.99e-10
audikw_1	9.43e+09	26.62	0.29	1	1e-09	7.85e-08
bmwcra_1	5.28e+08	0.40	0.03	1	1e-09	2.81e-08
ctu-1	3.16e+09	4.47	0.17	1	1e-09	2.41e-07
ctu-2	2.2e+09	2.43	0.08	1	1e-09	9.15e-08
cfd1	1.56e+08	0.15	0.01	1	1e-09	4.99e-11
cfd2	2.94e+08	0.27	0.02	1	1e-09	1.13e-10
conti20	5.97e+07	0.08	0.01	1	1e-09	4.82e-08
garybig	†	†	†	†	†	†
G3_circuit	9.02e+08	0.59	0.10	1	1e-09	3.57e-11
hood	2.59e+08	0.13	0.02	1	1e-09	2.05e-11
inline_1	1.38e+09	1.25	0.08	1	1e-09	2.68e-08
kyushu	9.11e+09	34.20	0.29	1	1e-09	4.46e-11
ldoor	1.32e+09	0.75	0.08	1	1e-09	3.59e-10
msdoor	4.75e+08	0.24	0.03	1	1e-09	2.32e-08
mstamp-2c	1.56e+10	62.36	0.43	1	1e-09	1.66e-12
nastran-b	8.68e+09	18.91	0.30	1	1e-09	5.35e-11
nd24k	2.57e+09	10.57	0.10	1	1e-09	7.49e-09
oilpan	8.81e+07	0.05	0.01	1	1e-09	4.54e-09
parabolic_fem	2.18e+08	0.13	0.03	1	1e-09	4.97e-12
pga-rem1	5.94e+08	0.41	0.08	1	1e-09	3.91e-08
pga-rem2	1.99e+09	1.91	0.33	1	1e-09	1.98e-10
qa8fk	1.79e+08	0.19	0.01	1	1e-09	6.59e-06
qa8fm	1.78e+08	0.19	0.01	1	1e-09	1.56e-15
ship_003	5.27e+08	0.73	0.03	1	1e-09	1.1e-07
shipsec5	4.63e+08	0.49	0.02	1	1e-09	1.05e-08
thermal2	4.53e+08	0.29	0.07	1	1e-09	5.11e-11
torso	6.72e+08	0.87	0.04	1	1e-09	3.05e-11

Table 40: Table showing the raw data for PROC64:wsmp:NONE:DIRECT:ND:ALL:NONE