Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 1332 – 1341

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

A Sparse Matrix Library with Automatic Selection of Iterative Solvers and Preconditioners

Takao Sakurai[a]*, Takahiro Katagiri[b], Hisayasu Kuroda[c], Ken Naono[d], Mitsuyoshi Igai[e], and Satoshi Ohshima[b]

[a]*Central Research Laboratory, Hitachi, Ltd., Yokohama, Japan*
[b]*Information Technology Center, The University of Tokyo, Tokyo, Japan*
[c]*Graduate School of Science and Engineering, Ehime University, Matsuyama, Japan*
[d]*R&D Department,Hitachi Asia Malaysia, Kuala Lumpur, Malaysia*
[e]*Hitachi ULSI Systems Co., Ltd., Tokyo, Japan*

Abstract

Many iterative solvers and preconditioners have recently been proposed for linear iterative matrix libraries. Currently, library users have to manually select the solvers and preconditioners to solve their target matrix. However, if they select the wrong combination of the two, they have to spend a lot of time on calculations or they cannot obtain the solution. Therefore, an approach for the automatic selection of solvers and preconditioners is needed. We have developed a function that automatically selects an effective solver/preconditioner combination by referencing the history of relative residuals at run-time to predict whether the solver will converge or stagnate. Numerical evaluation with 50 Florida matrices showed that the proposed function can select effective combinations in all matrices. This suggests that our function can play a significant role in sparse iterative matrix computations.

Keywords: Auto-tuning, linear problem, sparse matrix, iterative solver, preconditioner

1. Introduction

Current computer architectures are generally too complicated for user to manually tune the performance of numerical computations. For example, the increased number of cores in multi-core architectures, the deep hierarchical caches, and the non-uniform memory accesses degrade the performance of main processes for

* Corresponding author. Tel.: +81-45-860-2137.
E-mail: takao.sakurai.ju@hitachi.com.

doi:10.1016/j.procs.2013.05.300

numerical computations. Due to such complicated architectures, the cost for developing high-performance numerical software is increasing. This situation is causing a software crisis in high-performance numerical software.

In order to mitigate this critical situation, auto-tuning (AT) for numerical processing has been investigated in several layers of numerical software. ATLAS [1] is a software package for dense matrix libraries with AT functions in the lower layer, which is the basic numerical linear algebra (BLAS) layer. For sparse matrix libraries, SPARSITY [2] and OSKI [3] offer high performance in the sparse matrix-vector multiplication (SpMV, BLAS2) layer. In the higher layer, which involves numerical algorithms and solvers, FFTW [4] consists of fast Fourier transform routines that are widely used in AT functions. For sparse iterative libraries, ILIB [5] has AT functions of MPI optimization by reducing communication time. For dense matrix libraries, ABCLib [6] provides AT functions of loop unrolling and cash blocking in the middle layer of numerical computation, such as orthogonalization and multiple-loop computations.

However, if library users and developers want to use these AT functions, they first have to implement them in their library, which is laborious. For this reason, common APIs of AT functions for matrix libraries are needed.

In response to this need, we previously proposed a unified API called OpenATLib [7, 8] for the AT framework. OpenATLib is designed to reuse AT functions for matrix libraries. We also used the APIs of OpenATLib to develop a matrix library with a run-time AT function called Xabclib. This unified API has successfully reduced the cost of implementing AT functions in matrix libraries and has enhanced the application of various AT functions in multiple layers.

However, enhancing the solvability of linear matrix libraries is still an open issue. Solvability of a linear matrix problem depends not only on the combination of iterative solvers and preconditioners but also on the matrix characteristic features such as condition number and eigenvalue distribution [9]. Moreover, in many simulation cases, the computation cost of obtaining the characteristic features is more than that of solving the linear problem. In one simulation program [10], for example, there are only a few cases in which the program solves the same matrix enough times to compute the characteristic feature. For this reason, predicting an effective iterative solver/preconditioner combination for the matrix before it runs has thus far been difficult. Therefore, a new AT function of automatically selecting the solvable combination at run-time is needed.

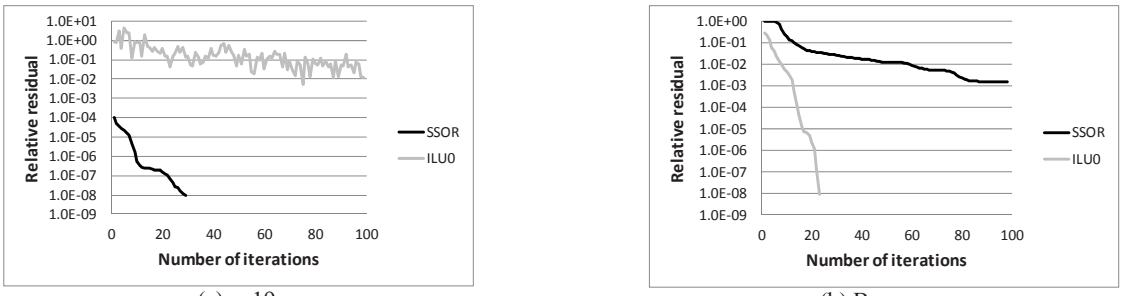
Hence, in the present study, we developed a novel AT function that selects the best combination of iterative solvers and preconditioners for iterative numerical libraries. This function can call two or more iterative solvers and preconditioners and selectively run them in order to satisfy user's requests, such as convergence criterion and iteration number tolerance.

The rest of this paper is organized as follows. In Section 2, a focusing issue of the sparse matrix library is presented. In Section 3, the AT functions for automatic selection of iterative solvers and preconditioners in OpenATLib are explained. Section 4 is a performance evaluation with one node of the HITACHI HA8000, which consists of 16 cores with four sockets of the AMD Opteron. Section 5 describes related research from the viewpoint of automatic selection of iterative solvers and preconditioners. Finally, we conclude the paper in Section 6.

2. Focusing Issue

The objective of the present study is to realize the automatic selection of iterative solvers and preconditioners for sparse matrix libraries by developing novel AT functions.

Recently, many iterative solvers and preconditioners methods have been proposed and implemented. Library users can select which ones they would like to use for solving their matrix. However, if they select the wrong iterative solver/preconditioner combination, they cannot obtain the solution. Figure 1 shows a comparison of the history of relative residuals by a GMRES(m) iterative solver with SSOR and ILU0 preconditioners. The solver with the SSOR preconditioner could solve ex19 but not Baumann, while the solver with the ILU0 preconditioner could solve Baumann but not ex19. Nevertheless, predicting the best iterative solver/preconditioner combination from the characteristic features of the matrix before running is computationally expensive. Therefore, automatic selection at running is an important issue for enhancing the solvability of matrix libraries.



(a) ex19 (b) Baumann

Fig. 1. Comparison of the history of relative residuals by GMRES(m) methods and preconditioners. The matrices are (a) ex19 and (b) Baumann from the University of Florida Sparse Matrix Collection [11].

3. Automatic Solvers and Preconditioners Selection

3.1. The Architecture

We developed two solvers with novel AT functions for automatic selection.

First, we developed an automatic halting function for iterative solvers. This function can halt an iterative solver automatically by predicting the convergence or stagnation of a relative residual.

Second, we developed a meta-solver that runs iterative solvers in a given order. This meta-solver can call multiple iterative solvers and preconditioners, and it determines the effective the iterative solver/preconditioner combination in given order to satisfy iteration tolerance.

Figure 2 shows the architecture of the automatic selection of iterative solvers and preconditioners.

Meta-solver users manually input matrix, right-hand side vector, initial approximate solution, convergence criterion, and iteration tolerance. The combination order is also given by the user or is determined by the meta-solver. The meta-solver calls the first iterative solver/preconditioner combination in the order. If the iterative solver converges, the meta-solver outputs the solution. If the automatic halting function halts the iterative solver, the meta-solver calls the next combination in the order.

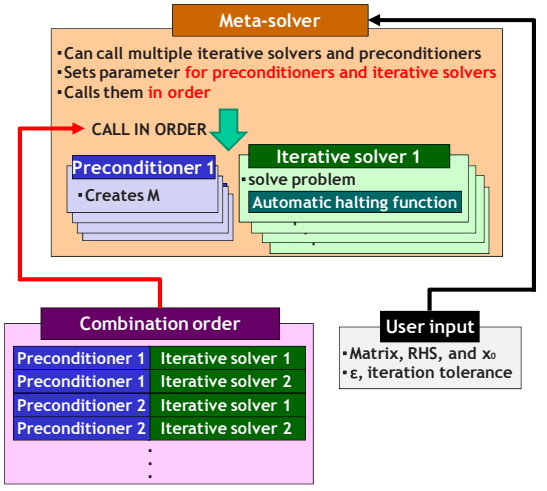


Fig. 2. Architecture of the automatic selection.

3.2. The Automatic Halting Function for the Iterative Solver

The automatic halting function predicts convergence or stagnation from the relative residual history. Figure 3 shows the movements of the relative residual by the BiCGStab and GMRES(m) methods.

In Fig. 3, the vertical axis shows the value of relative residuals, the horizontal axis denotes the number of iterations, and the vertical broken line at 100 denotes the iteration tolerance. The horizontal broken line denotes the convergence criterion as ten to the power of minus eight in this example. The black line shows that the iterative solver can obtain a solution while satisfying iterative tolerance. The red line also denotes that the solver can obtain a solution; however, it does not satisfy iterative tolerance. The blue and purple lines denote that the iterative solver cannot obtain the solution due to stagnation and divergence, respectively.

In this way, the relative residual history shows the various movements by solvers, preconditioners, and matrices. Accordingly, our automatic halting function uses the gradient of the relative residual history for prediction. Figure 4 shows the prediction strategy. The vertical axis shows the value of relative residuals, the horizontal axis denotes the number of iterations, and the vertical broken line denotes the iteration tolerance. The horizontal broken line denotes the convergence criterion. The blue, green, and red solid lines denote the relative residual history, and the blue, green, and red broken lines denote the prediction lines. The automatic halting function calculates the moving average of the relative residual history. Next, from the latest point of history, the automatic halting function draws a prediction line with the calculated moving average to the line of the tolerance. If the intersection of the convergence criterion with the prediction line is less than the tolerant iteration, the automatic halting function estimates that the iterative solver will converge. In contrast, when the intersection point is greater than the criterion, the automatic halting function estimates that the solver will not converge.

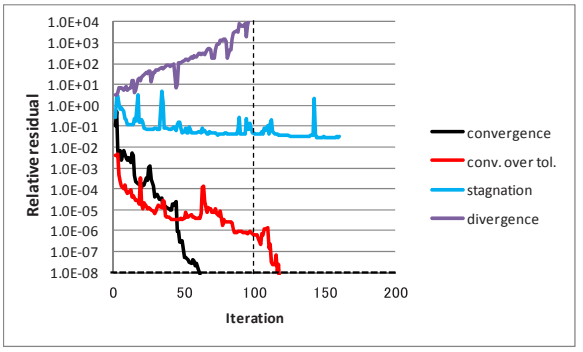


Fig. 3. Movements of relative residual.

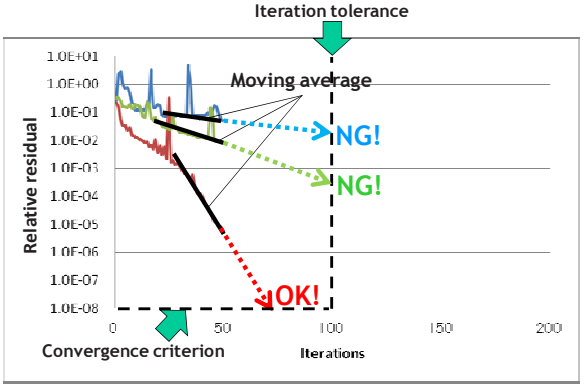


Fig. 4. Prediction strategy.

With the strategy, we implemented the algorithm of automatic halting function predicting stagnation and halting the iterative solver on convergence test. The algorithm, shown in Fig. 5, is as follows.

First, the counter and parameters are set to initialize. Second, the iterative solver runs 1-step iteration. Third, in the convergence test, if the approximate solution satisfies the convergence criterion, the iterative solver outputs it. Otherwise, fourth, the automatic halting function calculates the common logarithm of the latest relative residual. Fifth, the automatic halting function calculates the exponential moving average G_k of the relative residual. Here, when parameter α, called "Exponent", is near 1, the average amount is strongly influenced by the latest relative residual. Sixth, the automatic halting function calculates the predicted common logarithm value of the relative residual at iteration tolerance e_{tol} using G_k. Seventh, if e_{tol} is less than the convergence criterion, the automatic halting function sets counter p as 0; otherwise, it adds p by 1. Finally, if p exceeds a certain threshold value p_{th}, the automatic halting function halts the iterative solver. Otherwise, the iterative solver runs the next iteration.

(1): $p = 0, e_0 = 0, G_0 = 0$
(2): Run 1 iterarion
(3): If $r_k < \varepsilon$ then output "convergence"
Else goto (4)
(4): $e_k = \log(r_k)$
(5): $G_k = \alpha(e_k - e_{k-1}) + (1-\alpha)G_{k-1}$ $(0 < \alpha < 1)$
(6): $e_{tol} = e_k + G_k(I_{tol} - k)$ (I_{tol}: Iteration tolerance)
(7): If $e_{tol} < \log(\varepsilon)$ then $p = 0$
Else $p = p + 1$
(8): If $p > p_{th}$ then output "stagnation"
Else goto (2)

Fig. 5. Algorithm of AT function predicting stagnation and halting the iterative solver.

3.3. Meta-solver with automatic selection of preconditioners and iterative solvers

The meta-solver can call multiple preconditioners and iterative solvers and can automatically select an effective preconditioner/iterative solver combination. Figure 6 shows how the meta-solver operates.

First, the meta-solver runs the first combination of iterative solver and preconditioner. If the iterative solver converges or runs past the iteration tolerance, the meta-solver outputs a solution, and if the iterative solver stagnates, the meta-solver moves on to the next combination. At that time, if the relative residual when the iterative solver halts is the minimum from all that have been performed, the combination is recorded as the "best" combination.

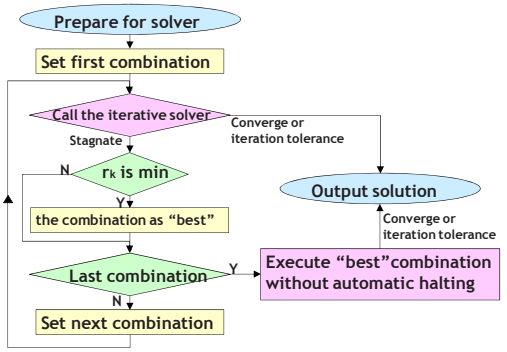


Fig. 6. Operation of meta-solver.

When the iterative solver of the last combination halts, the meta-solver runs the "best" combination without the automatic halting function until it reaches iteration tolerance.

4. Numerical Evaluation

4.1. Machine Settings

We used a HITACHI HA8000 machine for the numerical evaluation. Each node of the machine contains four AMD Opteron 8356 sockets (Quad core, 2.3 GHz). The L1 cache is 64 KB/core, the L2 cache is 512 KB/core, and the L3 cache is 2 MB/4 cores. The memory on each node is 32 GB with 667-MHz DDR2. The theoretical peak is 147.2 GFLOPS/node. We used Intel FORTRAN Compiler Professional Version 11.0 with the option "-O3 -m64 -openmp -mcmodel=medium."

4.2. Test Matrices

We used 50 asymmetric matrices from the University of Florida sparse matrix collection (referred to hereinafter as the UF collection) [11]. Information of the UF collection, including the matrix names and dimension N, is shown in Table 1. The sparse matrix format for OpenATLib is compressed row storage (CRS).

Table 1. Test matrices.

No.	Asymmetric matrix name	N
1	igbt3	10938
2	ex19	12005
3	sme3Da	12504
4	poisson3Da	13514
5	airfoil_2d	14214
6	epb1	14734
7	Memplus	17758
8	nmos3	18588
9	chipcool0	20082
10	epb2	25228
11	wang3	26064
12	wang4	26068
13	3D_28984_Tetra	28984
14	sme3Db	29067
15	viscoplastic2	32769
16	chem_master1	40401
17	sme3Dc	42930
18	xenon1	48600
19	3D_51448_3D	51448
20	ecl32	51993
21	epb3	84617
22	poisson3Db	85623
23	matrix_9	103430
24	Hcircuit	105676
25	Baumann	112211
26	barrier2-1	113076
27	torso2	115967
28	torso1	116158
29	dc2	116835
30	trans4	116835
31	trans5	116835
32	cage12	130228
33	FEM_3D_thermal2	147900
34	para-4	153226
35	xenon2	157464
36	Crashbasis	160000
37	Majorbasis	160000
38	Scircuit	170998
39	Transient	178866
40	hvdc2	189860
41	Stomach	213360
42	torso3	259156
43	Raj1	263743
44	ASIC_320ks	321671
45	ASIC_320k	321821
46	Language	399130
47	cage13	445315
48	rajat29	643994
49	ASIC_680ks	682712
50	ASIC_680k	682862

4.3. Experimental Condition of the Solvers

We compared the proposed meta-solver (including the automatic selection) with eight fixed combinations of iterative solvers and preconditioners and evaluated them using the following conditions. The combination order for this experiment was determined by the amount of time it took to compute the preconditioner.

- Solvers & preconditioners setting
  - Linear equations
    - GMRES(m) method
      - Restart frequency is controlled at run-time by OpenATLib [7]
    - BiCGStab method

- End-user accuracy requirement: 1.0E-08
- Initial RHS vector of x: All elements are set to 1
- Initial guess: All elements are set to 0
- Preconditioners
  - SSOR
  - ILU0
  - ILUT(10,1.0E-08)
  - ILUT(30,1.0E-08)
- Time tolerance: 600 sec
- Iteration tolerance: Meta-solver calculates as follows
  - Time tolerance/computation time of 1 iteration

❖ AT setting for predicting stagnation
- For GMRES(m) method
  - Exponent α: 0.9
  - Threshold value p_{th}: 3
- For BiCGStab method
  - Exponent α: 0.01
  - Threshold value p_{th}: 20

❖ Combination order for meta-solver
1. SSOR and BiCGStab
2. SSOR and GMRES(m)
3. ILU0 and BiCGStab
4. ILU0 and GMRES(m)
5. ILUT(10,1.0E-08) and BiCGStab
6. ILUT(10,1.0E-08) and GMRES(m)
7. ILUT(30,1.0E-08) and BiCGStab
8. ILUT(30,1.0E-08) and GMRES(m)

4.4. Results

Figure 7 shows the number of solved matrices by the proposed automatic selection as well as by the eight fixed combinations of iterative solvers and preconditioners.

None of the fixed iterative solvers/preconditioners combinations could solve all the matrices. In contrast, the proposed automatic selection could solve all of them by selecting the effective combination.

Table 2 shows the last performed combination with solver and preconditioner by automatic selection as well as the computation time by the eight fixed combinations. “Stag” denotes the relative residual stagnates in the solver and “Fail” denotes making preconditioned matrix fails by zero diagonal elements. The best times for “Fixed” are bold.

These results show that the computation time by the meta-solver is equal to the time by fixed SSOR and BiCGStab in over 30 cases. Therefore, it is heuristically recommended that SSOR and BiCGStab be the first combination run by the meta-solver. When the first combination can solve the matrix, the meta-solver runs it to the end of the calculation. Consequently, the computation time by the meta-solver is equal to the time by fixed combination in such cases. In contrast, when the first combination cannot solve the matrix, the meta-solver changes the combination until it obtains a solution. For example, ex19 can be solved only by SSOR and GMRES, and cage13 can be solved only by ILUT(30) and GMRES. In these cases, the meta-solver can select the solvable combination.

However, the proposed prediction function sometimes made a misjudgment. For example, sme3Db can only be solved by SSOR and BiCGStab. Figure 8 shows the relative residual history of sme3Db solved by SSOR and BiCGStab. The relative residual was stagnant from the 500th iteration to the 900th iteration, and over the 1000th

iteration, the solver converged. Consequently, our proposed AT function decided to halt the solver around the 600th iteration. We have to enhance the prediction function to improve the accuracy in cases like this.

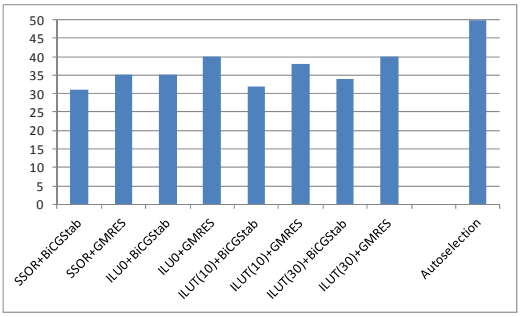


Fig. 7. The number of solved matrices by the eight combinations and by auto selection.

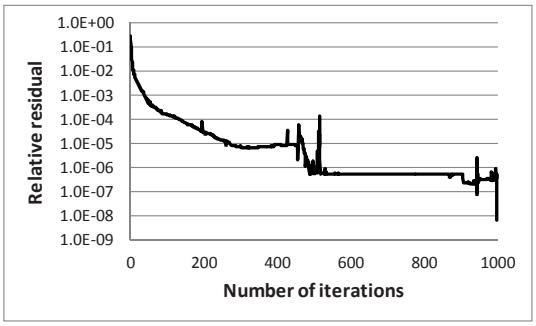


Fig. 8. The relative residual history of sme3Db solved by SSOR and BiCGStab.

5. Related Works

There have been a few previous studies on the AT function of automatic selection.

For example, Bhowmick [12] proposed selecting iterative solvers and preconditioners by a machine learning method for simulation that solves the same linear problem many times. McInnes [13] proposed a selection function by calculating the matrix characteristic features.

Our work differs in that we propose a novel automatic selection. The proposed AT function can automatically select an effective combination of iterative solver and preconditioner at run-time by predicting the convergence or stagnation of the relative residual and by running iterative solvers in a given order. Using our AT functions will enable library users to obtain the solution they require without learning or pre-computation.

6. Conclusion

We proposed a method for automatically selecting preconditioners and iterative solvers. To realize this automatic selection, we developed two novel AT functions.

First, we developed an automatic halting function for iterative solvers. This function can halt an iterative solver automatically by predicting the convergence or stagnation of a relative residual. Second, we developed a meta-solver that runs iterative solvers in a given order. This meta-solver can run multiple iterative solvers and preconditioners. Moreover, it runs the iterative solver/preconditioner combination in a given order to satisfy

iteration tolerance. Numerical experiment with 50 sparse matrices from the University of Florida showed that the proposed automatic selection can select effective preconditioner/solver combinations.

In order to establish a more accurate automatic selection of iterative solvers and preconditioners, we need to enhance the prediction method by increasing the prediction accuracy. Furthermore, we need to develop a parameter selection for SSOR's relaxation and ILUT's fill-in depths. We also need to devise a method of determining the combination order for the meta-solver by matrix characteristic features. These subjects will be considered in future research.

The functions proposed in this paper are implemented in OpenATLib and Xabclib [14]. OpenATLib and Xabclib are available as open-source freeware. The code is provided by the PC Cluster Consortium through an LGPL license. We hope that OpenATLib will provide a reference implementation or a tool for researchers who are investigating a particular numerical algorithm, such as GMRES(m).

Acknowledgements

This study was supported by the Seamless and Highly Productive Parallel Programming Environment for High-Performance Computing program of the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan and by JSPS KAKENHI Grant Number 24300004. Authors appreciate Professors Shoji Itoh and Kengo Nakajima of the University of Tokyo for fruitful discussions on the design and development of OpenATLib and Xabclib.

References

[1] R. C. Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimizations of software and the ATLAS project," Parallel Computing, Vol. 27, Issues 1–2, pp. 3–35, 2001.

[2] E-J. Im, K. Yelick, and R. Vuduc, "SPARSITY: Optimization framework for sparse matrix kernels," International Journal of High Performance Computing Applications (IJHPCA), Vol. 18, No. 1, pp. 135–158, 2004.

[3] R. Vuduc, J. W. Demmel, and K. A. Yelick, "OSKI: A library of automatically tuned sparse matrix kernels," in Proceedings of SciDAC, Journal of Physics: Conference Series, Vol. 16, pp. 521–530, 2005.

[4] M. Frigo and S. G. Johnson, "FFTW: An adaptive software architecture for the FFT," in Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 3, IEEE Press, Los Alamitos, CA, pp. 1381–1384, 1998.

[5] H. Kuroda, T. Katagir, M. Kudoh, and Y. Kanada, "ILIB_GMRES: An auto-tuning parallel iterative solver for linear equations," SC2001, 2001 (a poster).

[6] T. Katagiri, K. Kise, H. Honda, and T. Yuba, "ABCLib_DRSSED: A parallel eigensolver with an auto-tuning facility," Parallel Computing, Vol. 32, Issue 3, pp. 231–250, 2006.

[7] K. Naono, T. Katagiri, T. Sakurai, M. Igai, S. Ohshima, H. Kuroda, S. Itoh, and K. Nakajima, "A Fully Run-time Auto-tuned Sparse Iterative Solver with OpenATLib," The 4th International Conference on Intelligent and Advanced Systems (ICIAS2012), Proceedings of ICIAS2012 (IEEE Xplore database), 2012.

[8] T. Katagiri, T. Sakurai, M. Igai, S. Ohshima, H. Kuroda, K. Naono, and K. Nakajima, "Control Formats for Unsymmetric and Symmetric Sparse Matrix–Vector Multiplications on OpenMP implementations," High Performance Computing for Computational Science - VECPAR 2012, 2012.

[9] S. H. Chan, K. K. Phoon, and F. H. Lee, "A modified Jacobi preconditioner for solving ill-conditioned Biot's consolidation equations using symmetric quasi-minimal residual method," International Journal for Numerical and Analytical Methods in Geomechanics, Vol. 25, No. 10, pp. 1001–1025, 2001.

[10] T. Washio, J. Okada, and T. Hisada, "A parallel multilevel technique for solving the bidomain equation on a human heart with Purkinje fibers and a torso model," SIAM J Sci Comput, Vol. 30(6), pp. 2855–2881, 2008.

[11] The University of Florida sparse matrix collection: http://www.cise.ufl.edu/research/sparse/matrices/.

[12] S. Bhowmick, V. Eijkhout, Y. Freund, E. Fuentes, and D. Keyes, "Application of Alternating Decision Trees in Selecting Sparse Linear Solvers," Software Automatic Tuning: from concepts to state-of-the-art results, Springer-Verlag, pp. 153–174, 2010.

[13] L. McInnes, B. Norris, S. Bhowmick, and P. Raghavan, "Adaptive Sparse Linear Solvers for Implicit CFD Using Newton-Krylov Algorithms," Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics, 2003.

[14] Xabclib Project: http://www.abc-lib.org/Xabclib/index.html.

Table 2. Computation time by meta-solver and fixed combination of solvers and preconditioners.

#	Matrix name	Automatic selection			Fixed							
		Last run		Time	SSOR		ILU0		ILUT(10)		ILUT(30)	
		Precondi tioner	Solver		BiCG Stab	GMR ES(m)	BiCG Stab	GMR ES(m)	BiCG Stab	GMR ES(m)	BiCG Stab	GMR ES(m)
1	igbt3	ILU0	BiCGstab	22.15	Stag.	Stag.	1.59	**1.25**	Stag.	104.48	317.77	109.62
2	ex19	SSOR	GMRES	23.35	Stag.	**6.18**	Stag.	Stag.	Stag.	Stag.	Stag.	Stag.
3	sme3Da	SSOR	GMRES	65.75	Stag.	59.38	**25.71**	44.67	Stag.	186.82	Stag.	Stag.
4	poisson3Da	SSOR	BiCGstab	0.50	0.51	0.80	**0.47**	0.56	1.50	1.65	Stag.	52.61
5	airfoil_2d	SSOR	BiCGstab	0.81	0.79	3.71	0.35	0.88	**0.31**	0.47	0.68	0.84
6	epb1	SSOR	BiCGstab	0.21	0.22	0.23	**0.11**	0.13	0.23	0.27	0.48	0.55
7	Memplus	SSOR	BiCGstab	0.35	0.34	**0.19**	0.58	0.49	1.75	0.29	0.71	0.63
8	nmos3	ILU0	BiCGstab	36.31	Stag.	158.50	1.02	**0.96**	Stag.	16.84	32.83	30.60
9	chipcool0	SSOR	BiCGstab	0.54	0.56	0.86	**0.37**	0.58	0.71	0.72	2.21	2.22
10	epb2	SSOR	BiCGstab	0.18	0.17	0.15	0.10	**0.08**	0.27	0.27	0.70	0.77
11	wang3	SSOR	BiCGstab	0.27	0.23	0.23	**0.17**	0.19	1.16	1.32	3.01	3.32
12	wang4	SSOR	BiCGstab	0.27	0.28	0.32	**0.19**	0.21	0.66	0.72	1.64	1.62
13	3D_28984_Tetra	ILU0	BiCGstab	16.40	Stag.	Stag.	**0.07**	0.11	Stag.	124.81	Stag.	164.71
14	sme3Db	ILU0	GMRES	176.08	**80.71**	222.74	Stag.	172.51	Stag.	Stag.	Stag.	Stag.
15	viscoplastic2	SSOR	BiCGstab	2.10	**2.10**	2.12	5.98	5.09	Stag.	Stag.	Stag.	Stag.
16	chem_master1	SSOR	BiCGstab	0.73	**0.71**	1.54	0.43	0.86	0.77	0.93	1.22	1.20
17	sme3Dc	SSOR	GMRES	414.87	**170.30**	312.05	Stag.	237.91	Stag.	Stag.	Stag.	Stag.
18	xenon1	SSOR	BiCGstab	15.10	15.04	**12.33**	Stag.	470.68	Stag.	Stag.	27.51	28.40
19	3D_51448_3D	ILU0	BiCGstab	25.63	Stag.	Stag.	1.95	**1.54**	Stag.	17.35	Stag.	34.12
20	ecl32	SSOR	BiCGstab	2.39	2.39	3.74	**1.69**	3.72	3.21	4.33	9.12	10.41
21	epb3	SSOR	BiCGstab	2.06	2.10	2.06	**0.98**	1.11	1.70	1.79	1.94	2.41
22	poisson3Db	SSOR	BiCGstab	7.55	**7.43**	10.62	8.42	9.88	28.22	28.33	Stag.	Stag.
23	matrix_9	SSOR	BiCGstab	43.35	43.35	58.32	**3.42**	4.15	12.74	15.11	63.78	63.86
24	Hcircuit	ILUT(10)	BiCGstab	1.57	Fail	Fail	Fail	Fail	**1.56**	1.66	7.73	7.93
25	Baumann	ILU0	BiCGstab	50.12	Stag.	Stag.	23.90	18.03	Stag.	Stag.	**15.19**	20.37
26	barrier2-1	ILUT0	GMRES	162.38	Stag.	Stag.	Stag.	**83.72**	Stag.	Stag.	Stag.	Stag.
27	torso2	SSOR	BiCGstab	0.32	0.32	0.28	**0.20**	0.24	0.64	0.74	1.05	1.38
28	torso1	ILU0	BiCGstab	143.12	Stag.	Stag.	**70.01**	430.39	Stag.	Stag.	Stag.	Stag.
29	dc2	SSOR	BiCGstab	2.19	2.16	**1.31**	31.22	30.84	6.10	5.37	110.25	112.47
30	trans4	SSOR	BiCGstab	1.44	1.45	**1.28**	30.07	29.78	1.39	1.53	2.13	1.42
31	trans5	SSOR	BiCGstab	3.60	3.57	4.30	31.82	32.90	2.34	3.43	2.39	**2.00**
32	cage12	SSOR	BiCGstab	0.36	0.36	**0.33**	0.50	0.47	9.38	9.35	91.52	91.84
33	FEM_3D_thermal2	SSOR	BiCGstab	1.61	1.58	1.66	**0.93**	1.43	2.56	2.84	5.98	6.24
34	para-4	ILUT0	GMRES	242.57	Stag.	Stag.	Stag.	**202.63**	Stag.	Stag.	Stag.	Stag.
35	xenon2	SSOR	BiCGstab	63.80	**63.14**	55.13	Stag.	602.11	Stag.	Stag.	129.36	130.89
36	crashbasis	SSOR	BiCGstab	1.06	1.05	**0.96**	1.24	1.18	3.76	3.65	10.22	10.75
37	majorbasis	SSOR	BiCGstab	0.57	0.58	**0.49**	0.67	0.53	2.52	2.78	7.31	7.28
38	scircuit	ILUT(10)	BiCGstab	19.82	Fail	Fail	Fail	Fail	19.79	19.41	**13.18**	14.46
39	Transient	ILUT(10)	GMRES	68.50	Fail	Fail	Fail	Fail	Stag.	29.57	Stag.	**13.75**
40	hvdc2	ILUT(30)	GMRES	220.86	Fail	Fail	Fail	Fail	Stag.	Stag.	Stag.	**199.68**
41	Stomach	ILU0	BiCGstab	33.83	Stag.	395.81	**0.96**	0.98	2.53	2.48	5.91	6.25
42	torso3	SSOR	BiCGstab	27.92	27.98	28.29	**3.92**	4.06	56.93	30.46	12.94	15.27
43	Raj1	ILUT(10)	BiCGstab	97.79	Fail	Fail	Fail	Fail	97.64	**42.91**	58.77	74.71
44	ASIC_320ks	SSOR	BiCGstab	10.13	10.10	5.31	0.78	0.62	**0.19**	0.29	0.20	0.29
45	ASIC_320k	ILUT(10)	BiCGstab	1.89	Fail	Fail	Fail	Fail	1.89	**1.73**	2.25	2.05
46	Language	SSOR	BiCGstab	0.75	0.75	0.57	0.67	0.55	0.66	**0.32**	0.54	0.35
47	cage13	SSOR	BiCGstab	1.34	1.35	**1.23**	2.03	1.90	53.58	54.48	Stag.	Stag.
48	rajat29	ILUT(30)	GMRES	278.14	Fail	Fail	Fail	Fail	Stag.	Stag.	Stag.	**293.44**
49	ASIC_680ks	SSOR	BiCGstab	0.36	0.36	0.50	**0.28**	0.52	0.28	0.40	0.28	0.40
50	ASIC_680k	ILUT(10)	BiCGstab	12.29	Fail	Fail	Fail	Fail	**12.27**	12.27	13.54	13.64
Solved				50	31	35	35	40	32	38	34	40