

Results of the Parallel Solution of a Large Complex Linear System

This document presents results of the parallel solution of a complex linear system provided by ExxonMobil Upstream Research Corporation (URC). The main goal of this work was to consider applying preconditioning techniques to a tested system and numerically compare the efficiency of preconditioned and non-preconditioned parallel, iterative solvers.

Structure of the original matrix

The effort started with a large system to be solved:

$$A x = b \quad (1)$$

where A is a sparse matrix with complex coefficients, b is a right hand side vector and x is a vector of unknowns containing the zero initial guess at the start. The system (1) can be also considered in the form

$$(A_R + iA_I)(x_R + ix_I) = b_R + ib_I \quad (2)$$

where A_R , x_R , b_R are the real parts of the matrix and the vectors and A_I , x_I , b_I are the imaginary parts.

The matrix was a square matrix containing 445,383 rows and 5,686,829 nonzero entries. The matrix had a 3-by-3 nodal structure. The number of nodes along one line was 51, the number of lines in a plane was 41 and the number of planes was 71.

The properties of the matrix made it rather complicated to solve. It was not diagonal dominant, i.e. $|a_{ii}| < \sum_{j \neq i} |a_{ij}|$, and the structure of the connections between lines and planes was complicated as well. Nevertheless, for matrices with this type of structure it is possible to develop a special type of nested preconditioners.

Solution of an equivalent real-valued system

The original complex system (1) can be expressed as an equivalent real-valued system (i.e. with real number coefficients) in several different forms (see [1] below). However, the performance of both serial and parallel real solvers for all these real-valued systems is much worse than that of the complex solver being evaluated. These results have not been included in this report.

Results of the parallel solution of the original complex system

For the parallel solution of the original complex system (1) we used the *LinCoS™ (Linear Complex Solver)* package. *LinCoS* is a complex iterative solver developed by Neurok Techsoft LLC for the solution of Helmholtz and Maxwell equations. The current serial version of *LinCoS* includes a set of iterative methods, an advanced variant of a simple *ILU0* preconditioner and an advanced variant of an *ILUT* preconditioner.

The parallel version of *LinCoS* is based on MPI and includes:

- *Simple* and *Metis* basic partitioners
- *Diagonal* and *Overlap* parallel preconditioners
- Parallel version of iterative methods

Below the parallel algorithms are briefly described:

1. Partitioners

There are two partitioners available in *LinCoS*:

- *Simple partitioner* partition a matrix into a given number of strips with the same number of rows;
- *Metis partitioner* is an external freely available package which implements high-quality multi-level graph partitioning minimizing number of *interface rows* and *edge-cuts* [2].

2. Preconditioners

There are two serial preconditioners implemented in *LinCoS*:

- The *ILU0* preconditioner implements the simplest variant of incomplete LU decomposition without fill-in and has two parameters: ω – the relaxation coefficient, and σ – the shift coefficient [3].
- The *ILUT* preconditioner implements a well-known incomplete LU preconditioner of Y. Saad and has four parameters: ε – the drop tolerance parameter, τ – the maximal number of elements for one preconditioned row in L,U parts, ω – the relaxation coefficient, σ – the shift coefficient [3].

There are also two parallel preconditioners implemented in *LinCoS*:

- *Block Diagonal Preconditioner* applies a serial preconditioner to diagonal block of each strip in parallel;
- *Block Overlap Preconditioner* applies a serial preconditioner to 1-level overlapped diagonal block of each stripe in parallel.

3. Parallel Iterative Methods

LinCoS provide two primary iterative methods:

- *CGB* – Conjugate Gradient Based (similar to BiCGStab) [4];
- *MRB* – Minimal Residual Based (similar to restarted TFQMR) [3].

4. Numerical experiments

All numerical experiments were performed on a cluster of 4 nodes connected through a Dolphin SCI interconnect. Each node included 2 AMD Opteron 246 CPUs and 4 GB memory. The operating system was SLES 9 64x Linux with Scali MPI.

The stopping criterion for an iterative method was

$$|r^n| < 10^{-5} |b|$$

where r^n is an iterative residual on the n^{th} iteration and x^n is an approximate solution. The maximum allowed number of iterations – 2000.

A series of three numerical experiments was performed. In each case a *1-level overlap (BOP(1))* scheme of diagonal block extracting was used. Using a *non-overlapped* scheme of diagonal block extraction showed poor performance.

The results below compare *total time* which consists of *solution time* (time spent finding a solution), *partitioner time* (time spent computing and applying a partition) and *matrix distribution time* (time spent distributing a matrix among the processors). The results also present the number of iterations which shows the influence of the applied algorithms on convergence behavior of an iterative method.

4.1. Solution by non-preconditioned iterative methods

The first series of tests were run to solve the system using non-preconditioned iterative methods and to observe their parallel effectiveness. The results for both *TFQMR* and *BiCGStab(2)* are presented in Table 1. A comparison of *solution time*, *number of iterations* and *speed up* for both methods are shown in Figures 1-3 respectively.

Partitioner	P	Non-preconditioned TFQMR				Non-preconditioned BiCGStab(2)			
		Its	TotTime	SolTime	SpeedUp	Its	TotTime	SolTime	SpeedUp
	1	469	51.24	51.24	1	553	57.98	57.98	0.88
simple	2	469	28.18	27.65	1.85	553	31.64	31.10	1.65
simple	4	469	15.57	14.90	3.44	613	19.07	18.39	2.79
simple	6	469	11.56	10.83	4.73	521	12.26	11.53	4.44
simple	8	469	9.15	8.41	6.09	573	10.51	9.76	5.25
metis	2	469	30.30	28.23	1.82	633	38.16	36.09	1.42
metis	4	469	16.52	14.24	3.60	613	19.92	17.65	2.90
metis	6	469	12.50	10.13	5.06	557	13.93	11.54	4.44
metis	8	469	10.33	7.80	6.57	589	11.79	9.28	5.52

Table 1: Results of Parallel Solution by Non-preconditioned Methods

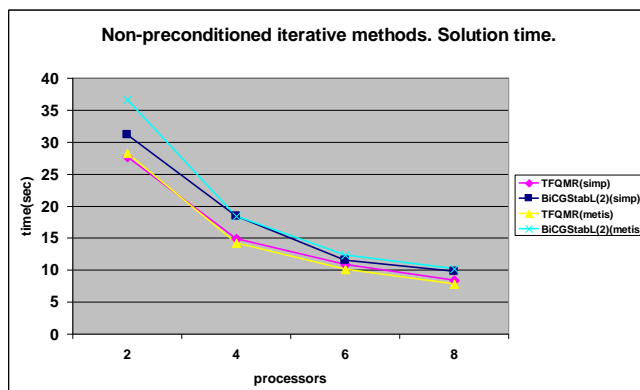


Figure 1: Solution Time

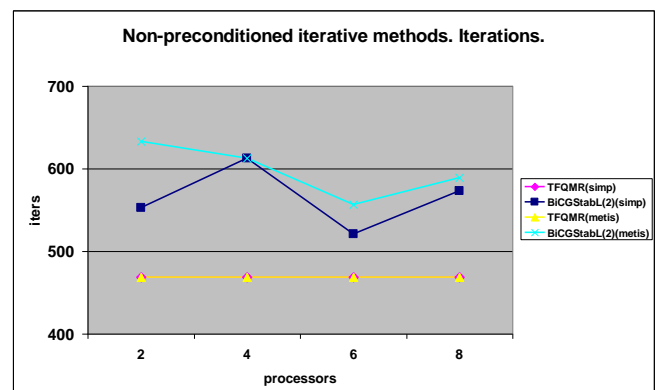


Figure 2: Number of Iterations

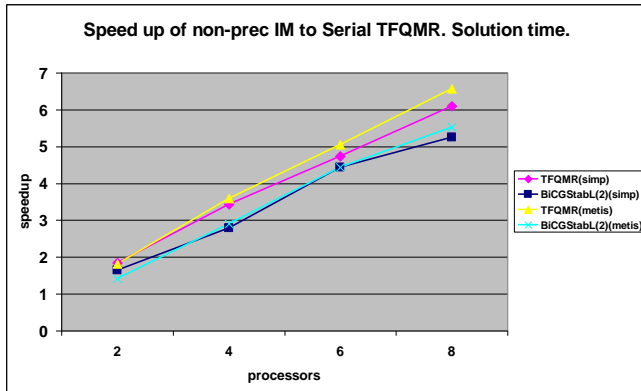


Figure 3: Speed Up of Non-preconditioned Methods

As the results show, the non-preconditioned *TFQMR* method is faster than non-preconditioned *BiCGStab(2)* method. As a result, the time for the serial *TFQMR* method is taken as the base time for estimation of parallel efficiency below.

4.2. Solution by preconditioned iterative methods preconditioned with ILU0

The second series of test were run to solve the system using the same iterative methods, but both methods were preconditioned by *ILU0* and the parameters were varied slightly in order to achieve the best combination. The results for both *TFQMR* and *BiCGStab(2)* are presented in Table 2. Comparison of *solution time*, *number of iterations* and *speed up* for both methods are shown in Figures 4-6 respectively. Note that in these table and figures, the solution time of serial *ILU0+TFQMR* is used as a basis for comparison to show the parallel efficiency related to the serial algorithm of the same type.

Partitioner	P	ILU0+TFQMR				ILU0+BiCGStab(2)			
		Its	TotTime	SolTime	SpeedUp	Its	TotTime	SolTime	SpeedUp
	1	223	40.03	40.03	1	125	21.87	21.87	1.83
simp	2	173	18.68	18.14	2.21	145	15.17	14.63	2.74
simp	4	233	13.61	12.93	3.1	161	9.33	8.66	4.62
simp	6	221	9.99	9.26	4.32	157	7.16	6.43	6.23
simp	8	211	7.66	6.90	5.8	177	6.39	5.64	7.10
metis	2	171	19.45	17.37	2.3	125	14.40	12.32	3.25
metis	4	181	12.16	9.88	4.05	113	8.37	6.07	6.59
metis	6	157	8.75	6.39	6.26	129	7.43	5.07	7.90
metis	8	195	8.67	6.15	6.51	153	7.26	4.76	8.41

Table 2: Results of Parallel Solution Using ILU0 Preconditioner

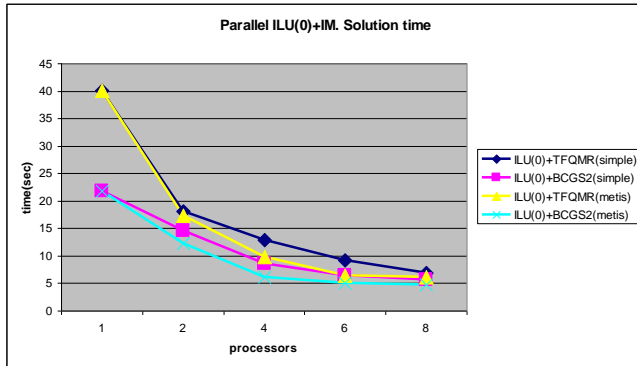


Figure 4: ILU(0) Solution Time

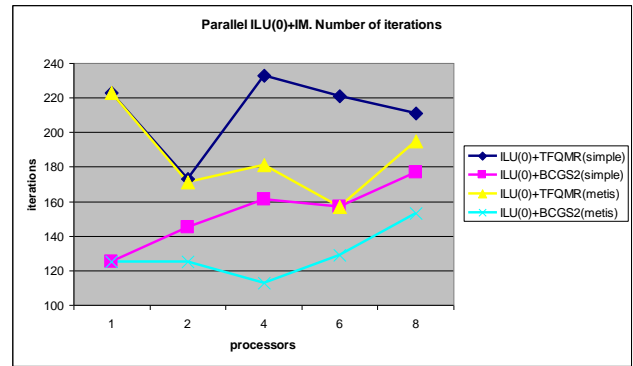


Figure 5: ILU(0) Number of Iterations

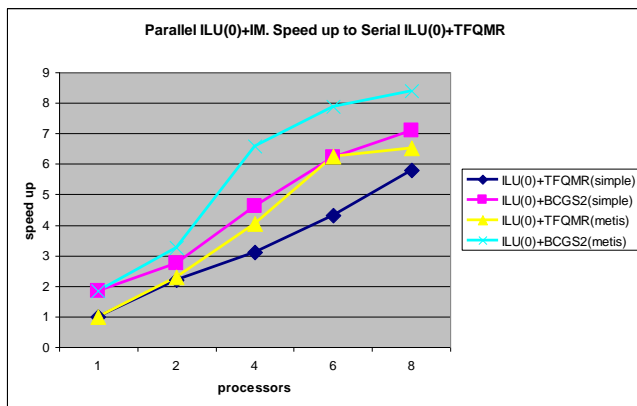


Figure 6: Speed Up to Serial ILU(0)+TFQMR

As can be seen in the table and figures, the combination of *Metis* partitioner, *BOP(1)* diagonal block extraction and *BiCGStab(2)* iterative method shows significantly better parallel performance than other combination. It provided a speed up of up to 8.5 times for 8 processors over the corresponding serial method.

4.3. Solution by iterative methods preconditioned with ILUT

The third series of test were run to solve the system with iterative methods preconditioned by *ILUT*. As before, the parameters were varied slightly to achieve optimum results. The results for both *TFQMR* and *BiCGStab(2)* are presented in Table 3. Comparison of *solution time*, *number of iterations* and *speed up* for both methods are shown in Figures 7-9 respectively. Note that in these table and figures, the solution time of serial *ILUT+TFQMR* is used as a basis for comparison to show the parallel efficiency related to the serial algorithm of the same type.

Partitioner	P	ILUT+TFQMR				ILUT+BiCGStab(2)			
		Its	TotTime	SolTime	SpeedUp	Its	TotTime	SolTime	SpeedUp
	1	163	39.19	39.19	1	101	20.41	20.41	1.92
simp	2	199	23.80	23.26	1.68	133	15.17	14.63	2.68
simp	4	199	12.80	12.12	3.23	157	10.13	9.46	4.14
simp	6	177	8.86	8.14	4.81	149	7.39	6.66	5.88
simp	8	183	7.27	6.53	6.00	117	6.33	5.58	7.02
metis	2	147	19.58	17.52	2.24	117	15.09	13.02	3.01
metis	4	163	12.07	9.78	4.01	129	9.88	7.61	5.15
metis	6	151	9.02	6.66	5.88	117	7.42	5.04	7.78
metis	8	183	8.80	6.28	6.24	97	7.03	4.52	8.67

Table 3: Results of Parallel Solution Using ILUT Preconditioner

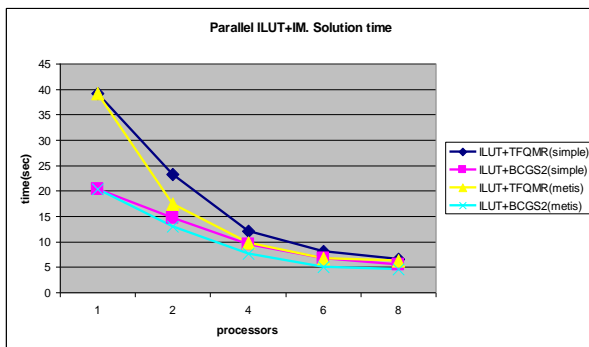


Figure 7: ILUT Solution Time

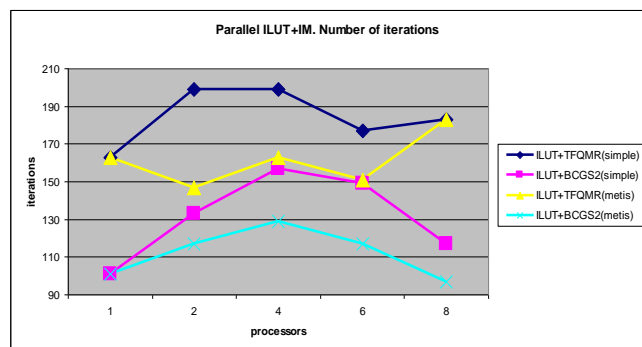


Figure 8: ILUT Number of Iterations

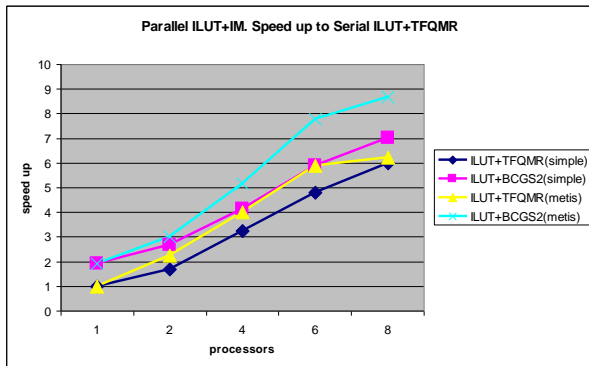


Figure 9: Speed Up to Serial ILUT+TFQMR

Once again, as can be seen in the table and figures, the combination of *Metis* partitioner, *BOP(1)* and *BiCGStab(2)* shows significantly better parallel performance than other combination speeding up the performance of the corresponding serial variant by more than 8.5 times for 8 processors.

4.4. Aggregate comparison

This section compares the performance of preconditioned and non-preconditioned iterative methods. Table 4 provides a comparison of all serial variants for both iterative methods – *TFQMR* and *BiCGStab(2)*:

Preconditioner	TFQMR			BiCGStab(2)		
	Its	SolTime	SpeedUp	Its	SolTime	SpeedUp
	469	51.24	1.00	553	57.98	0.88
ILU(0)	223	40.03	1.28	125	21.87	2.34
ILUT	163	39.19	1.31	101	20.41	2.51

Table 4: Results for Serial Solution

As one can see, preconditioning gives a significant improvement in the solver performance (up to 2.5 times). Note also that non-preconditioned *TFQMR* is faster than non-preconditioned *BiCGStab(2)* while preconditioned *BiCGStab(2)* is much more effective by both number of iterations and total solution time.

Next we present the speed up of all combinations of parallel preconditioned iterative methods to the serial non-preconditioned *TFQMR*.

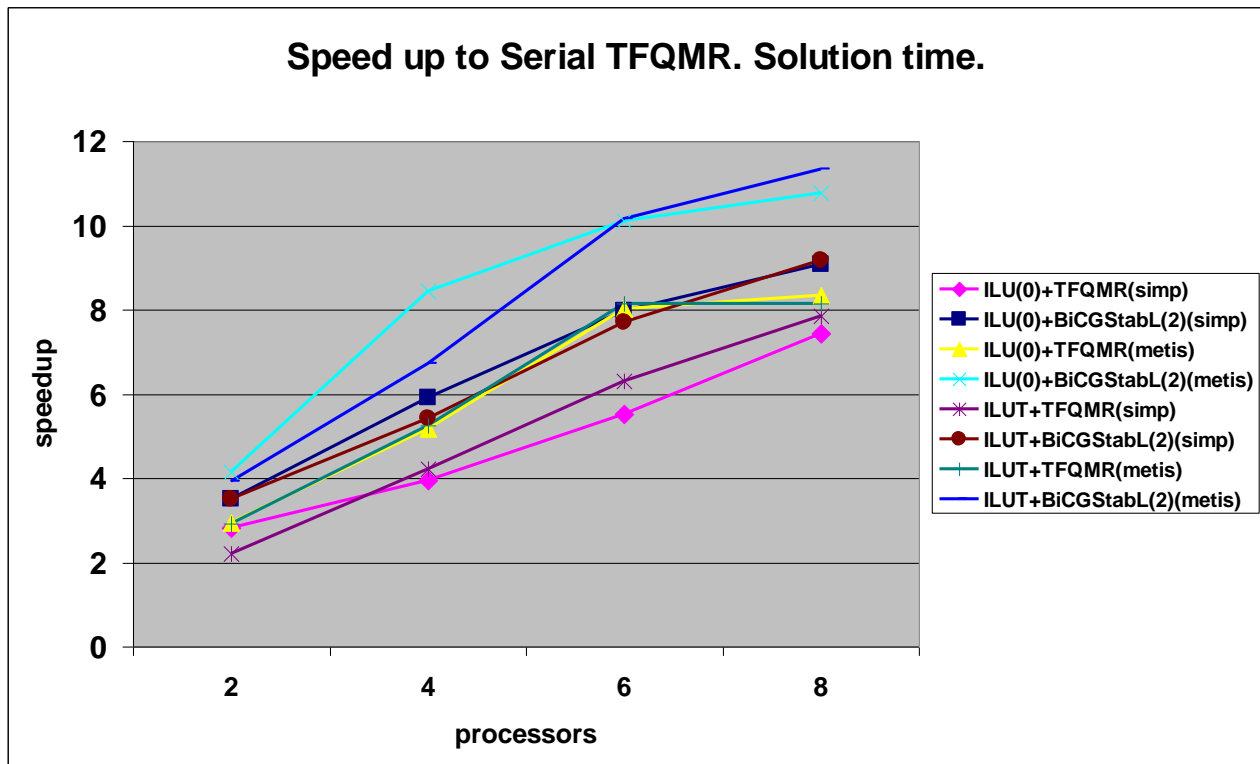


Figure 10: Speed Up of Parallel Preconditioned Iterative Methods

As is seen in Figure 10, the best results were obtained for *Metis* partitioner, *BiCGStab(2)* iterative method and *1-level Overlap* scheme of diagonal block extracting. At the same time, both *ILU(0)* and *ILUT* show similar speed up. The best results obtained are:

Processors	Metis+BCGS2	
	Preconditioner	Speed Up
2	ILU0	4.16
4	ILU0	8.44
6	ILUT	10.17
8	ILUT	11.34

Table 5: Summary of Optimal Performance Improvement

Finally, we show the speed up of all considered methods relative to parallel *TFQMR*. In other words, we compare the performance of preconditioned iterative methods relative to that of non-preconditioned *TFQMR* for 2, 4, 6 and 8 processors.

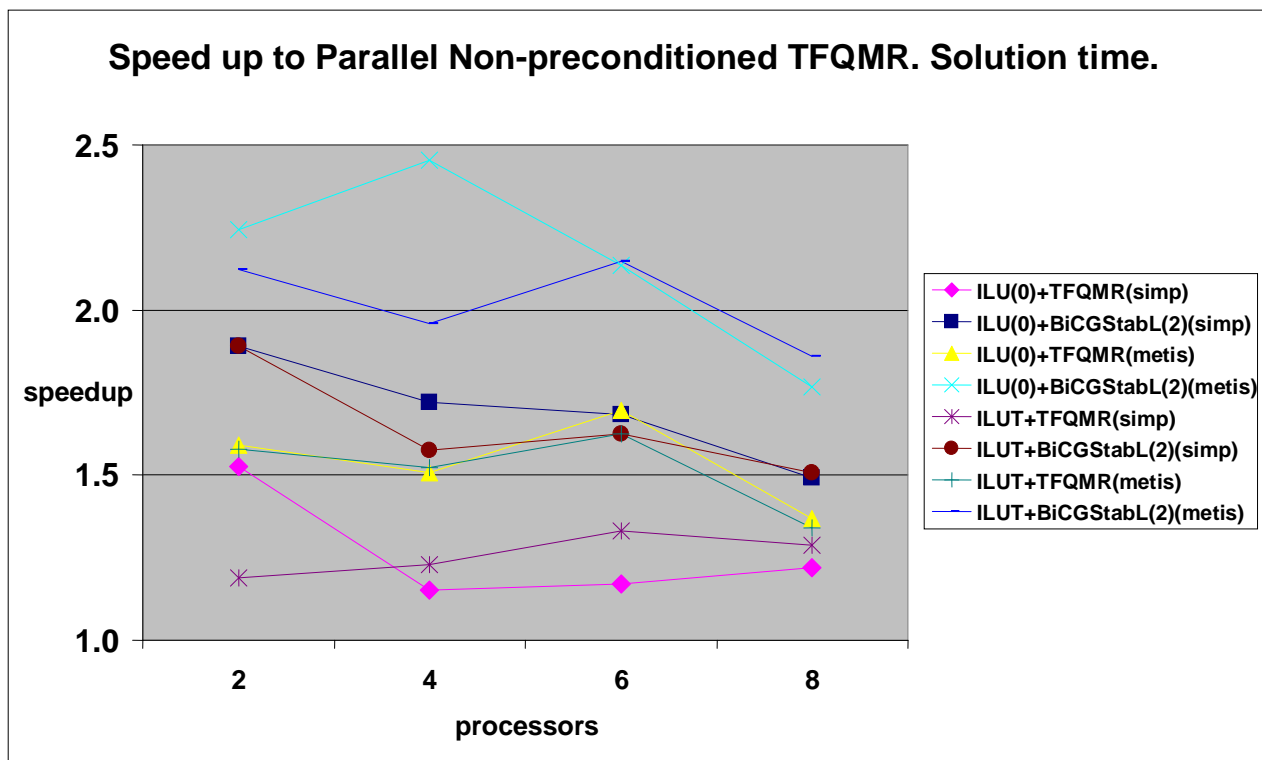


Figure 11: Speed Up of Parallel Preconditioned Iterative Methods over Parallel Non-preconditioned TFQMR

We see in Figure 11 that preconditioning provides a speed up of up to 2.5 times in comparison with non-preconditioned iterative methods.

Conclusion

The results presented in this document show that the original large sparse linear system can be solved efficiently by parallel preconditioned iterative methods on up to eight processors. The parallel preconditioned iterative methods implemented in *LinCoS* give up to 2.5 times increased performance in comparison with the best parallel non-preconditioned iterative method.

Based on these results we can expect that further improvement of the existing algorithms and development of new perspective algorithms, such as Complex Incomplete Cholesky Preconditioner, Complex Nested Factorization and overlap with several levels, would provide a significant speed up for the parallel solution of huge complex systems. It should also be noted that applying existing methods for the solution of the corresponding real-valued system gives significantly worse results.

Bibliography

- [1] Abnita Munankarmy, Michael Heroux. *A comparison of two equivalent real formulations for complex valued linear systems.*
- [2] George Karypis and Vipin Kumar. *Metis: a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices.*
- [3] Yousef Saad. *Iterative Methods for sparse linear systems.*
- [4] Gerard Sleijpen and Diederik Fokkema. *BiCGStab(L) for linear equations involving unsymmetric matrices with complex spectrum.*