Characterizing the Relationship between ILU-type Preconditioners and the Storage Hierarchy

Diego Rivera†, David Kaeli†and Misha Kilmer‡
†Department of Electrical and Computer Engineering
Northeastern University, Boston, MA 02115
{drivera, kaeli}@ece.neu.edu
‡Department of Mathematics
Tufts University, Medford, MA 02155
misha.kilmer@tufts.edu

EXTENDED ABSTRACT

Many problems in high performance applications involve the solution of large sparse linear systems. A barrier encountered in this class of applications is the computational time required. Optimization techniques that focus on reducing internode communication and improving data partitioning/layout can significantly lower this computational barrier [9], [10], [3], [4], [2]. While parallelization can also be used to improve performance, the sparsity of the data reduces the effectiveness of direct parallel computation.

ILU-type preconditioning techniques are widely recognized as being an extremely effective approach to providing efficient solvers[7]. These techniques have been used to increase the performance and reliability of Krylov subspace methods. However, a drawback of these approaches is that it is difficult to choose appropriate values for the preconditioner tuning parameters[1]. Usually, parameter selection is done through trial-and-error for a few sample matrices from a given application. For instance, Figure 1 shows the percentage of duple (a duple is a set of parameter values) providing convergence in less than 500 iterations using the preconditioner *ILUD*[1] and the Krylov method *GMRES*[2]. Several matrices from different scientific applications were tested[5] and fourteen possible values for each parameter were evaluated, giving us 378 possible combinations to try for each matrix.

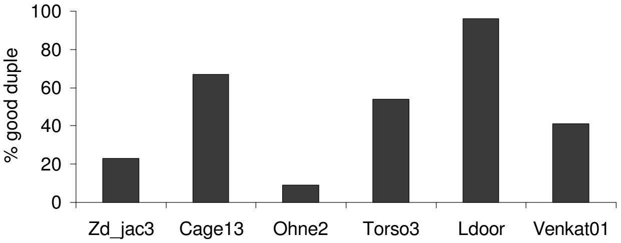


Fig. 1. Percentage of duples providing convergence in less than 500 iterations

In our work we have found that the performance of an application depends upon the relationship between the preconditioner tuning parameters and the memory hierarchy of the machine used to compute the solution. The parameter values used to obtain the fastest execution time and an acceptable final error may be different for different memory hierarchies. This occurs because: 1) the non-zero structure of the new coefficient matrix depends on the tuning parameter values and 2) our ability to exploit the locality present in the new matrix given a particular memory hierarchy.

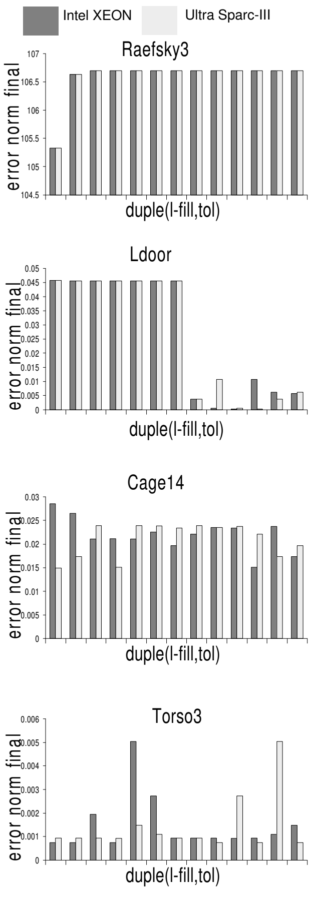
The difference in performance on different memory hierarchies becomes significant when the problem's conditions make it more difficult to solve. These conditions are related to the dropping strategy adopted in the preconditioner algorithm. For example, the relation among the numerical symmetry and the bandwidth (NS/B) of the coefficient matrix allows us to estimate how difficult it can be to solve the problem using the preconditioner *ILUT*[3]. This is because the dropping strategy of the preconditioner *ILUT* is based on dropping elements in the Gaussian Elimination process according to their magnitude. The results shown in Figure 2 support the previous analysis. These graphs show the final error norm obtained by the first thirteen duple, ordered in increasing order by the execution time of ILUT/GMRES. The convergence criterion is based on the residual norm; GMRES stopped iterating when the relative residual norm was below some tolerance.

The experiments were run on a machine based on Sun Ultra Sparc-III 750MHz microprocessor (L1D 64KB 4way, L2 8MB 2way, 1 GB RAM), and on a machine based on Intel XEON 3.06GHz microprocessor (L1D 8KB 4way, L2 512 KB 8way, L3 1 MB 8way, 2 GB RAM). Also, the *PIN* tool was used to capture cache events[6]. LRU and random replacement policies were modeled.

We can see from Table I and Figure 2 that the difference between these machines in terms of performance and parameter values turns out to be significant when the relation between the numerical symmetry and the bandwidth decreases.

We are developing an efficient algorithmic approach to select the best values of the preconditioner parameters for a given memory hierarchy. The algorithm will allow us to balance the total number of iterations until convergence and

[1] Incomplete *LU* factorization preconditioner with diagonal compensating
[2] Generalized Minimal Residual Method
[3] Incomplete *LU* factorization preconditioner with Threshold



Xeon		Ultra	
fill-in	drop tolerance	fill-in	drop tolerance
1.00E+00	5.00E-01	1.00E+00	5.00E-01
1.00E+00	2.50E-01	1.00E+00	2.50E-01
1.00E+00	1.00E-01	1.00E+00	1.00E-01
1.00E+00	5.00E-02	1.00E+00	5.00E-02
1.00E+00	2.50E-02	1.00E+00	2.50E-02
1.00E+00	1.00E-02	1.00E+00	1.00E-02
1.00E+00	1.00E-03	1.00E+00	1.00E-03
1.00E+00	1.00E-04	1.00E+00	1.00E-04
1.00E+00	1.00E-05	1.00E+00	1.00E-05
1.00E+00	1.00E-10	1.00E+00	1.00E-06
1.00E+00	1.00E-20	1.00E+00	1.00E-07
1.00E+00	1.00E-30	1.00E+00	1.00E-20
1.00E+00	1.00E-06	1.00E+00	1.00E-30

Xeon		Ultra	
fill-in	drop tolerance	fill-in	drop tolerance
5.00E+01	1.00E-02	5.00E+01	1.00E-02
5.00E+01	1.00E-03	5.00E+01	1.00E-07
5.00E+01	1.00E-04	5.00E+01	1.00E-06
5.00E+01	1.00E-07	5.00E+01	1.00E-03
5.00E+01	1.00E-06	5.00E+01	1.00E-05
5.00E+01	1.00E-10	5.00E+01	1.00E-10
5.00E+01	1.00E-05	5.00E+01	1.00E-04
5.00E+01	1.00E-01	5.00E+01	1.00E-01
4.00E+01	1.00E-01	5.00E+01	5.00E-02
5.00E+01	5.00E-02	5.00E+01	2.50E-02
5.00E+01	2.50E-02	4.00E+01	1.00E-01
5.00E+01	2.50E-01	4.00E+01	5.00E-02
4.00E+01	5.00E-02	4.00E+01	2.50E-02

Xeon		Ultra	
fill-in	drop tolerance	fill-in	drop tolerance
1.00E+00	5.00E-01	1.50E+01	2.50E-01
4.00E+01	5.00E-01	1.30E+01	2.50E-01
2.00E+00	5.00E-01	4.00E+01	1.00E-01
2.00E+01	5.00E-01	3.00E+01	1.00E-01
1.70E+01	5.00E-01	5.00E+01	1.00E-01
3.00E+00	5.00E-01	2.00E+01	1.00E-01
1.50E+01	5.00E-01	7.00E+00	5.00E-01
5.00E+00	5.00E-01	9.00E+00	5.00E-01
3.00E+01	5.00E-01	1.30E+01	5.00E-01
9.00E+00	5.00E-01	1.10E+01	5.00E-01
5.00E+01	5.00E-01	2.00E+00	5.00E-01
1.10E+01	5.00E-01	1.50E+01	5.00E-01
1.30E+01	5.00E-01	5.00E+00	5.00E-01

Xeon		Ultra	
fill-in	drop tolerance	fill-in	drop tolerance
1.70E+01	5.00E-02	4.00E+01	2.50E-02
1.50E+01	5.00E-02	5.00E+01	2.50E-02
1.10E+01	5.00E-02	2.00E+01	2.50E-02
2.00E+01	5.00E-02	3.00E+01	2.50E-02
3.00E+01	5.00E-02	1.50E+01	2.50E-02
1.30E+01	5.00E-02	1.70E+01	2.50E-02
4.00E+01	5.00E-02	1.70E+01	5.00E-02
5.00E+01	5.00E-02	4.00E+01	5.00E-02
1.70E+01	2.50E-02	5.00E+01	5.00E-02
3.00E+01	2.50E-02	1.50E+01	5.00E-02
1.50E+01	2.50E-02	3.00E+01	5.00E-02
5.00E+01	2.50E-02	1.30E+01	5.00E-02
2.00E+01	2.50E-02	1.10E+01	5.00E-02

Fig. 2. Error norm vs. duples ordered by minimum execution time

Name	Non-zero elements	Rows	NS	NS/B
Raefsky3	1,488,768	21,200	48%	4.3
Ldoor	42,493,817	952,203	100%	1.06
Cage14	27,130,349	1,505,785	21%	0.48
Torso3	4,429,042	259,156	0%	0

TABLE I
DESCRIPTION OF MATRICES EVALUATED

the cost of the preconditioner for a given memory hierarchy. We show that on one memory hierarchy, a greater level of fill-in can be used than on other hierarchies. The general steps of our algorithm are:

- Segmentation of the selection space. The dimension of the subspaces depends on the difficulty of the problem, which can be expressed in terms of the characteristics of the matrix. Some tens of characteristics can be defined[8]. However, only these characteristics that are related to the preconditioner dropping strategy must be considered.
- Find the best subspace based on some previous experiences extrapolating existing results, i.e. use known performance to develop a performance model based on the problem's characteristics considered.
- Predict the execution time dominance given a memory hierarchy, among the phase of factorization and the phase when the matrices L and U are applied. The model to predict has been focused on data locality. We are developing a model of locality for theses phases based on functions of distance among elements of the coefficient matrix. As a result of the prediction, we can decide if it is fitting to use a greater level of fill-in.

The first two steps may help to reduce the time required to re-compute the preconditioner whenever convergence is not satisfactory; meanwhile step three helps to tune the selection of the preconditioner parameters for a given memory hierarchy.

ACKNOWLEDGMENT

This project is supported by the National Science Foundations Computing and Communication Foundations Division, grant number CCF-0342555 and the Institute of Complex Scientific Software.

REFERENCES

[1] Michele Benzi. Preconditioning techniques for large linear systems:a survey. *J. Comput. Phys.*, 182(2):418–477, 2002.
[2] J. Dongarra, G. Bosilca, Z. Chen, V. Eijkhout, G. E. Fagg, E. Fuentes, J. Langou, P. Luszczek, J. Pjesivac-Grbovic, K. Seymour, H. You, and S. S. Vadhiyar. Self-adapting numerical software (sans) effort. *IBM J. Res. Dev.*, 50(2/3):223–238, 2006.
[3] E. M. Garzn and I. Garca. Approaches based on permutations for partitioning sparse matrices on multiprocessors. *J. Supercomput.*, 34(1):41–61, 2005.
[4] Anshul Gupta. Fast and effective algorithms for graph partitioning and sparse-matrix ordering. *IBM J. Res. Dev.*, 41(1-2):171–183, 1997.
[5] Matrix Market. http://math.nist.gov/MatrixMarket.
[6] Pin. http://rogue.colorado.edu/Pin/index.html.
[7] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.
[8] Jun Zhang Shuting Xu. Data mining approach to matrix preconditioning problem. In *Proceedings of the Eighth Workshop on Mining Scientific and Engineering Datasets (MSD05)*, Newport Beach, CA, 2005. ACM Press.
[9] S. Toledo. Improving the memory-system performance of sparse-matrix vector multiplication. *IBM J. Res. Dev.*, 41(6):711–726, 1997.
[10] Manuel Ujaldn, Emilio L. Zapata, Shamik D. Sharma, and Joel Saltz. Parallelization techniques for sparse matrix applications. *J. Parallel Distrib. Comput.*, 38(2):256–266, 1996.