Performance Model for Parallel Mathematical Libraries Based on Historical Knowledgebase*

I. Salawdeh, E. César, A. Morajko, T. Margalef, and E. Luque

Departament d'Arquitectura de Computadors i Ssitemes Operatius, Universitat Autònoma de Barcelona, 08193 Bellaterra, Spain
Ihab.Salawdeh@caos.uab.es, {Eduardo.Cesar,Anna.Morajko, Tomas.Margalef,Emilio.Luque}@uab.es

Abstract. Scientific and mathematical parallel libraries offer a high level of abstraction to programmers. However, it is still difficult to select the proper parameters and algorithms to maximize the application performance. This work proposes a performance model for dynamically adjusting applications written with the PETSc library. This model is based on historical performance information and data mining techniques. Finally, we demonstrate the validity of the proposed model through real experimentations.

Keywords: Performance Model, Mathematical Performance, PETSc Performance, Dynamic mathematical model.

1 Introduction

Parallel processing has become attractive with the increase in processor speed and reduction in cost per computation unit. However, writing applications and developing software to solve mathematical problems using the parallel processing programming techniques, such as MPI[1] and PVM[2] is not easy, because of the complexity of the algorithms required to solve such problems, and the need for an elevated level of experience in writing high performance applications.

Mathematical and scientific libraries are helpful tools that provide many pre-implemented algorithms to solve mathematical problems. However, as there is no unique solution for all linear systems, many algorithms are usually provided to solve different problems. However, the performance of these algorithms depends on the nature of the problem to be solved.

Moreover, this performance may vary dynamically during the execution depending on the input data and the parameters of the solvers. For instance, in a PETSc[3] linear solver application the time needed to solve a tri-diagonal 10000x10000 matrix using the Richardson KSP and the Jacobi PC and a sparse memory data structure is more than 14 minutes, while the time needed to solve the same matrix using the Chevychev KSP, the block jacobi preconditioner, and dense memory data structure needs only 15 seconds. Consequently, the need for dynamic models that can help the developer to

* This work was supported by MEC under contracts TIN2004-03388 and TIN2007-64974.

E. Luque, T. Margalef, and D. Benítez (Eds.): Euro-Par 2008, LNCS 5168, pp. 110–119, 2008.

choose between this huge set of library parameters and system configuration is in increase, as obtaining the best performance in such application is a key issue.

This work, aims to build intelligent performance models for parallel mathematical applications, based on a historical knowledgebase of performance behavior information, that can switch between the available mathematical algorithms, such as KSP solvers and preconditioners, and system specific parameters, such as the number of processors, automatically in order increase the application's performance.

To achieve this objective, we have developed a performance model for the PETSc mathematical library, which depending on the problem type, input data, and environmental parameters, chooses the most suitable solving parameter set: solving algorithm, preconditioner, and data structure to represent the problem data.

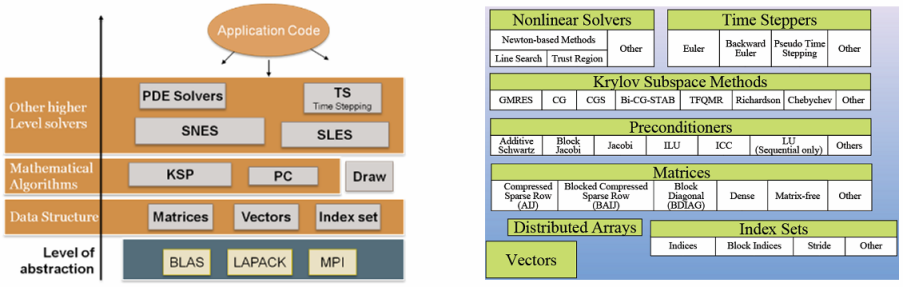
In this paper, we will first analyze PETSc applications in order to highlight the performance problems of this library, then we will discuss the proposed performance model and explain each one of its three components, subsequently we will validate our model by real case experiments followed by the related efforts and conclusions.

2 Performance Analysis for PETSc Applications

Mathematical libraries were developed to provide encapsulated and pre-implemented algorithms for the linear system solvers, which provide high level APIs that let the programmer to reuse these algorithms in the development of scientific applications without concerning neither about the communication policies between the processors nor the inner details of the algorithms.

In particular, PETSc or Portable Extensible Toolkit for Scientific computations is a suite of data structures and routines that provide the building blocks for the implementation of large-scale application codes on parallel (and serial) computers. It uses BLAS[4][5] and LAPACK[6] as a mathematical Kernel and Message Passing Interface (MPI) for the communications between the computation nodes. It is organized hierarchically, as shown in Figure 1-a, and provides an expanded tool for parallel linear equation solvers, nonlinear equation solvers, and time integrators.

PETSc provides many algorithms that could be used to solve parallel problems; for example, it provides a large set of Krylov subspace methods and preconditioners and it has many types of data structures representation, as seen in Figure 1-b, which shows some of PETSc numerical components.



(a) (b)

Fig. 1. (a) PETSc level of abstraction. (b) PETSc Numerical Components.

In order to build a performance model for a parallel mathematical library we studied the behavior of applications written with this library and the effects of both the input data provided by the user and the algorithms' parameters provided by the library. Therefore, we executed all the possible cases for different types of input data, using every possible data representation, and solving each case using different Krylov Subspace Solvers and Preconditioners. The results were surprisingly diverse and varied; that means, the time needed to solve a matrix depends on the nature of the matrix, how it was represented in memory, and on both the KSP and the PC used to solve this problem. For instance, Figure 2 shows the execution time in seconds for a 10000x10000 Around-Diagonal matrix that was executed on 12 KSP and three different Preconditioners.

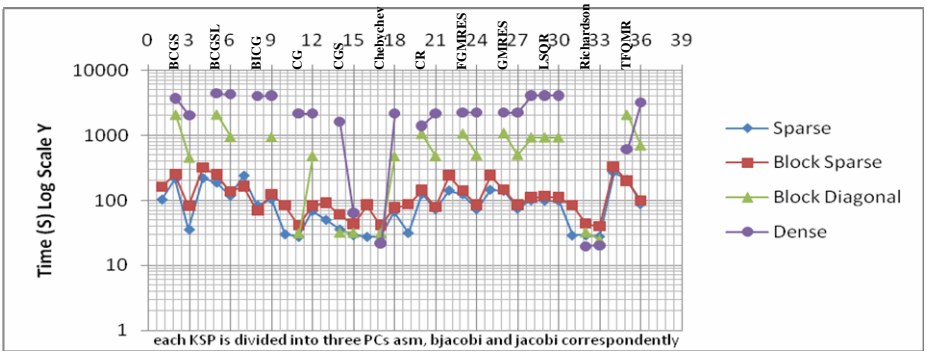


Fig. 2. Around-Diagonal 10000x10000 matrix execution times

From the execution results we can notice that the matrix solving time depends on the way the matrix was stored in memory. It can be seen in the example that the sparse representation had nearly the least execution time, while the dense representation has a very long execution time. Nevertheless, the variance between the KSPs and PCs also affects in a very significant way the problem execution behavior.

3 Performance Model

Implementing and developing a performance model for predicting the performance of an application is not trivial, especially when it considers the application as a black-box and it has to apply tuning actions dynamically for such applications.

To solve an $Ax=b$ problem using PETSc a series of steps should be followed when implementing the system. First, a suitable way should be used to represent the data in memory and to decide how to distribute these data across the processors. Then the solver and preconditioner used for solving the problem should be chosen, taking into consideration that these solver and preconditioner should be compatible with each other as well as with the input data category and representation.

The proposed performance model consists of three main modules or parts which cooperate between each other to achieve our goals:

- The **pattern recognition engine** that helps to summarize and categorize the input data characteristics and information.
- The **knowledgebase** that contains information about the historical executions' input data patterns and performance behavior. The pattern recognition engine is responsible for classifying the input data, and comparing between them and the data patterns in the historical knowledgebase.
- The **data mining engine** that collects all the classifications derived by the pattern recognition engine, the input data, and the system environment parameters and behavior in order to find the performance problems and predict the possible best solving algorithms and its correspondent PETSc parameters.

3.1 The Pattern Recognition Engine

Several matrix types exist, depending on the problem nature, and there is no definite way to classify them or to categorize all matrix types. However, several proposals classify matrices depending on the distribution of the nonzero and the main diagonal. Consequently, we have classified the matrices into six main types:

- Diagonal matrix, which has all the nonzero entries in its main diagonal.
- Tri-diagonal matrix, which has all the nonzero entries in its three main diagonals.
- Around-diagonal matrix, which has all, or the majority, of the nonzero entries in and around the main diagonal of the matrix
- Upper triangular matrix, or Lower triangular matrix which has all, or the majority, of the nonzero entries either above or below the main diagonal correspondently.
- Distributed matrix, which have the nonzero entries distributed randomly in the matrix and not only around the diagonal.
- Zero-diagonal matrix, which is a special case where the matrix main diagonal entries are zeros.

The Pattern Recognition Engine functionality is divided into three stages:

- Pattern Creation Stage
- Density Calculation Stage
- Structural Analysis Stage.

3.1.1 Pattern Creation Stage

The pattern has the format of a 10x10 matrix; each entry represents one of the hundred sub matrices that construct the main matrix and contains a structure of statistics that belongs to the corresponding sub-matrix, such as, the number of nonzero entries in the sub-matrix and the percentage of these entries with respect to its total number of values. In order to create the matrix pattern a number of steps should be followed (Figure 3 illustrates the pattern creation process):

- Divide the matrix into sub matrices and calculate the size and the dimension of each sub-matrix which will be 10% of the total dimension and 1% of the total size
- Divide each sub-matrix into four triangular parts that help to clarify the diagonal data and increase the results accuracy.
- Compute the number of nonzero values in each sub-matrix

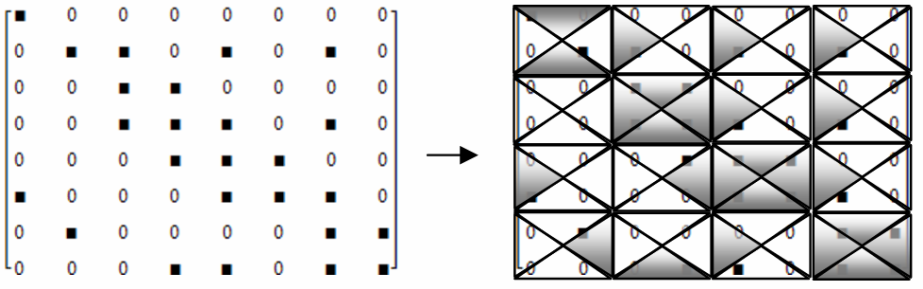


Fig. 3. Pattern creation process for a 4x4 pattern

- Compute the percentage of the nonzero values in each sub-matrix with respect to the sub matrix total size.

The main diagonal is a critical important factor at the time of recognizing the matrix because it has a huge impact on the matrix solving process. Thus, in the pattern creation phase a copy of the main diagonal entries and its related statistics, such as, the number of nonzero entries and their percentage are being saved separately from the matrix Pattern blocks.

3.1.2 Density Calculation Stage

The matrix pattern shows the number and the percentage of the nonzero entries in each of its pattern blocks, these percentages represent the matrix density. The matrix density is a relative characteristic, hence, the computation of such aspect needs techniques which may integrate some image processing and mathematical statistics algorithms. As far as the input matrix and the knowledgebase matrices share the same pattern size of 10x10 blocks each one divided into four triangular parts, the density factor will be the sum of the distances between each triangular part in the input matrix pattern and its equivalent in each knowledgebase pattern.

The distance will be calculated using a special case of the Minkowski Distances[7] of order λ when λ=1 and it is called the City-Block distance or Manhattan distance, expression 1 shows the Manhattan general case.

$$D_{ij} = \sum_{k=1}^{n} |x_{ik} - x_{jk}| \tag{1}$$

The results should be normalized to values between 0 and 1 according to expression 2, where zero means both matrices share the same density, and one means that they have a very different density.

$$Normalized\ Distance = \frac{\text{Distance}}{\text{Max Distance}} \tag{2}$$

This stage helps in identifying the density and the ratio of the nonzero entries in a matrix, but what happens if two matrices have the same density and a very different data distribution?

3.1.3 The Structural Analysis Stage

The main objective of this stage is to characterize the distribution of the data in the matrix without concerning about the data density or size; the only factor that affects this stage is the presence or the absence of a nonzero entry in the pattern blocks.

The structural analysis phase uses a data masking technique when comparing between matrices, it means that when receiving the input matrix pattern from the pattern creation stage it masks each block in the input pattern with the correspondent block in the knowledgebase patterns, converting its value to 1 if both values are equal or both contain a nonzero entry, and to 0 if one of the values is zero and the other contains at least a nonzero entry. Jaccard's Coefficient[7] (expression 3) is used to calculate the Fitness between the knowledgebase patterns and the input matrix pattern.

$$d_{AB} = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{3}$$

Consequently, the expression becomes

$$distance\ (\boldsymbol{Fitness}) = 1 - \frac{\boldsymbol{number\ of\ ones\ in\ the\ masked\ matrix}}{\boldsymbol{400 + 400 - number\ of\ ones\ in\ the\ masked\ matrix}} \tag{4}$$

Matrix similarity value will vary between 0 and 1, zero means both matrices share the same structure while one means they are totally different.

It can be noticed that each of the previous mechanisms classifies the matrix from different perspective, and spotlights different characteristics of the matrix data that cannot be seen by the other. A full image about the data distribution and density is been gotten by combining both pattern recognition strategies

Next, the diagonal density distance is calculated by computing the absolute difference between input matrix diagonal density and the knowledgebase matrices diagonal density.

After combining the three values by an equally likely weight summation where the sum will be a real value between 0 and 3 where zero means both matrices are identical and three represents completely different matrices.

3.2 The Knowledgebase

As the model depends on previous performance knowledge, a historical knowledgebase which contains the performance information about previous executions of linear algebraic problem applications has been built. It contains the behavioral information of a huge number of pre-planned executions of different types of input data with nearly all the possible parameters, providing different data representations, different Krylov subspace solvers, and different Preconditioners, executed on different number of processors.

Then a reference patterns list was built and saved in the historical knowledgebase. These patterns correspond to the matrices which the knowledgebase had been created upon their performance behavior. Table 1 summarizes the input matrices, their sizes and their correspondent patterns and the parameters on which they were executed.

Table 1. The Knowledgebase creating execution parameters

Input Matrix	No. Processors	Matrix Size	Memory Representations	KSP Solvers	Preconditioners
Diagonal Tri-Diagonal Distributed Zero-Diagonal Lower-Triangular Around-Diagonal	1 8 16 32	1000 10000 20000	Sparse Dense Block Sparse Block Diagonal	BiCGStab BiCGStab(L) BiConjugate Gradient Conjugate Gradient Conj. Grad. Squared Chebychev Conjugate Residuals FGMRES GMRES LSQR Richardson TFQMR	Jacobi Bjacobi ASM

3.3 The Data Mining Engine

The pattern recognition engine makes wide steps in finding the most suitable solving parameters by recognizing the data and summarizing its characteristics. After recognizing and classifying the input matrix by the pattern recognition engine, the data mining engine takes the control, and starts searching for the most suitable solving parameter set from the knowledgebase, as the following:

- After receiving the recommended matrix pattern from the pattern recognition engine, the data mining engine starts searching for the solver parameter set from the knowledgebase excluding all the matrix patterns cases, but the recommended one.
- It uses the City-Block (Manhattan) distance algorithms to calculate the distances between the input data factors, and the knowledgebase off-pattern factors such as, the real size of the matrix and the number of processors in the application communication world.
- Then, the solver prediction component chooses the proper configuration according to the distance factors and the least execution time.
- If the solving process did not reach a correct solution within this configuration set, the last step will be repeated until reaching the result excluding the used options.

4 Model Assessment

To examine our model we built a simple linear algebraic solver application based on PETSc library, this application contains only solvers and data related essential PETSc calls. Moreover, we built a tool that controls the application execution, meets the functionality of the model and passes the suitable solver, Preconditioner and data representation to the application by changing the parameters of the PETSc routines calls dynamically.

The experiments were performed on different matrices obtained from Matrix Market[8] web site a component of the NIST project[9], which provides access to a repository of test data for use in comparative studies of algorithms for numerical linear algebra. The SHERMAN5 matrix is an example of the matrices where the model was experimented.

Table 2. SHERMAN5 Pattern Recognition Engine results

Knowledgebase Pattern	City Block (Manhattan) distance	Fitness	Diagonal Density	Overall Distance
Tri-Diagonal	0.14433	0.283262	0	0.427592
Around-Diagonal	0.599593	0.481973	0	1.081566
Lower Triangular	0.35347	0.688525	0.5	1.541994
Distributed	1	0.6733	0	1.6733
Zero-diagonal	0.111314	0.555957	1	1.667271
Diagonal	0.09817	0.4	0	0.49817

The SHERMAN5 is a matrix that represents a fully implicit black oil model from the oil reservoir simulation challenge matrices, it is a 3312x3312 matrix, contains 20793 nonzero entries and 3312 nonzero diagonal entries matrix.

Firstly, we applied the pattern recognition phase which results are shown in Table 2. It can be seen that the most similar matrix in the knowledgebase to the SHERMAN5 matrix was the Tri-Diagonal matrix with a 0.43 overall distance.

At the same time the Data Mining Engine specified the 1000x1000 matrix and the 8 processors execution world as searching criteria for the solving parameter set and the final recommended results were: sparse as memory representation, Jacobi as pre-conditioner, and the chebychev as the recommended Krylov subspace solver.

Upon our proposal both matrices may share similar behavior according to the input parameters. Consequently, to validate these results we made executions of all the possible solution cases for SHAREMAN5 matrix, comparing their solving times to the same cases of the corresponding Tri-diagonal. Accordingly, by looking at the performance behavior for the SHAREMAN5 matrix on 8 processors environment, in Figure 4, it can be found that the model predicted case may not be the most optimal solution, nevertheless, it remains one of the "best" solutions.

At the same time by comparing between SHERMAN5 graph and the Tri-diagonal graph, in Figure 5, it can be noticed that both graphs are not identical; however, most of the Tri-diagonal matrix valid parameters are included in the SHAREMAN5 matrix possible solution, which shows the robustness of the solution predicting process.

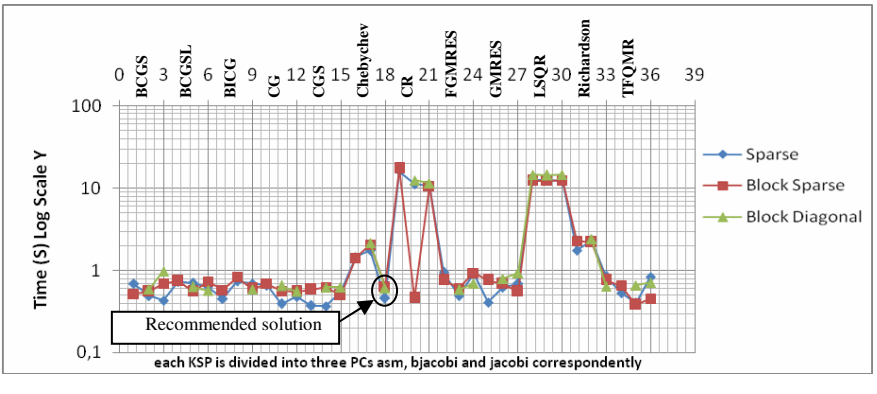


Fig. 4. Execution behavior of SHAREMAN5 3312x3312 matrix on 8 processors

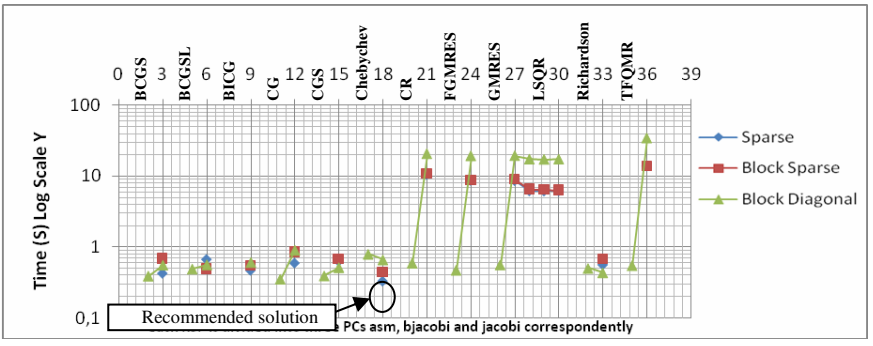


Fig. 5. Execution behavior of Tri-Diagonal 1000x1000 matrix on 8 processors

5 Related Efforts

The importance of parallel programming grew as well as the searching for better performance, a number of investigation lines where lunched to accomplish this objective. ATLAS[10] is an Automatic Tuning Linear Algebra Software project for the automatic generation and optimization of numerical software for processors with deep memory hierarchies and pipelined functional units. It is an implementation of the "Automated Empirical Optimization of Software" AEOS paradigm, that provide many ways of doing a required operation, and uses empirical timing in order to choose the best method for a given operation.

More limited ATLAS like functionality was included in PHiPAC[11], and more dynamic solutions are provided by SANS[12] Self-Adapting Numerical Software and SALSA[13] Self-Adapting Large-scale Solver Architecture. SANS is a collaborative effort between different projects that deals with the optimization of software at different levels in relation to the execution environment and helps to build a common framework on which these projects can possibly coexist. While SALSA aims to assist applications in finding suitable linear and nonlinear system solvers based on analysis of the application-generated data based on a database of performance results that can tune the heuristics over time.

6 Conclusions

From the study of the performance information of the mathematical libraries it was noticed that the performance of the application may vary dynamically according to the input data and the solving environment. In this work we defined a performance model for automatic and dynamic tuning of mathematical applications based on historical performance information.

Thus, we have developed triple component model consisting of: pattern recognition engine that classifies and characterizes the problem, a historical knowledgebase that was filled with plenty of PETSc's library performance information for a wide set of data, and the data mining engine which dives into the knowledgebase in order to get the recommended configuration and tuning points in the application.

Additionally, we planned real case problems in order to validate the model making a full execution for all the possible parameters and the results were optimistic. Moreover, it was noticed that the knowledgebase can be adapted by including more performance information for different types of problems.

References

1. Gropp, W., Lusk, E., Skjellum, A.: Using MPI. In: Portable Parallel Programming with the Message Passing Interface, 2nd edn., Cambridge, London, England. MIT Press in Scientific and Engineering Computation Series (1999)
2. Parallel Virtual Machine, http://www.csm.ornl.gov/pvm/
3. Balay, S., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc Users Manual. ANL-95/11 - Revision 2.1.5, Argonne National Laboratory (2004)
4. Basic Linear Algebra Subprograms, http://www.netlib.org/blas/
5. Dongarra, J.J., Du Croz, J., Duff, I.S., Hammarling, S.: A set of Level 3 Basic Linear Algebra Subprograms. ACM Transactions on Mathematical Software (TOMS) 16, 1–17 (1990)
6. Anderson, E., Bai, Z., Bischof, C., Blackford, L.S., Demmel, J., Dongarra, J.J., Du Croz, J., Hammarling, S., Greenbaum, A., McKenney, A., Sorensen, D.: LAPACK Users' Guide, 3rd edn. Society for Industrial and Applied Mathematics (1999)
7. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search The Metric Space Approach, ch.1, pp. 5–66. Springer, NY (2006)
8. Boisvert, R.F., Pozo, R., Remington, K., Barrett, R., Dongarra, J.J.: The Matrix Market: A web resource for test matrix collections. In: Boisvert, R.F. (ed.) Quality of Numerical Software, Assessment and Enhancement, pp. 125–137. Chapman and Hall, London (1997), http://math.nist.gov/MatrixMarket/
9. Tools for Evaluating Mathematical and Statistical Software, http://math.nist.gov/temss/
10. Whaley, R.C., Petitet, A., Dongarra, J.J.: Automated empirical optimizations of software and the ATLAS project. Parallel Computing 27, 3–35 (2001)
11. Bilmes, J., Asanovic, K., Chin, C., Demmel, J.: Optimizing matrix multiply using PHiPAC: a portable, high-performance, ANSI C coding methodology. In: Proceedings of the 11th International Conference on Supercomputing, pp. 340–347 (1997)
12. Dongarra, J., Bosilca, G., Chen, Z., Eijkhout, V., Fagg, G.E., Fuentes, E., Langou, J., Luszczek, P., Pjesivac-Grbovic, J., Seymour, K., You, H., Vadhiyar, S.S.: Self-adapting numerical software (SANS) effort. IBM J. Res. Dev. 50, 223–238 (2006)
13. Demmel, J., Dongarra, J., Eijkhout, V., Fuentes, E., Petitet, A., Vuduc, R., Whaley, R.C., Yelick, K.: Self Adapting Linear Algebra Algorithms and Software. Proceedings of the IEEE 93(2), 293–312 (2005)