Some considerations about the automatic optimisation of parallel linear algebra routines*

Javier Cuenca[1] and Domingo Giménez[2]

[1] Departamento de Ingeniería y Tecnología de Computadores.
Universidad de Murcia. Aptdo 4021. 30001 Murcia, Spain.
javiercm@ditec.um.es
[2] Departamento de Informática y Sistemas.
Universidad de Murcia. Aptdo 4021. 30001 Murcia, Spain.
domingo@dif.um.es

Abstract. This report summarizes the work carried out during a stay at the Innovative Computing Laboratory, University of Tennessee, from September 19 to October 31, 2001. The research has been done in the field of the automatic optimisation of parallel linear algebra software, working mainly with two routines: the LU factorisation, which his one of the basic routines used in the study of the behaviour of dense lineal algebra routines, and which has been previously used to analyse the automatic optimisation of linear algebra routines in homogeneous and fixed load parallel systems, and a routine to solve the Least Square Problem of Toeplitz matrices, used in the analysis because it has a lower order ($O\left(n^2\right)$), which produces different problems to those of a higher routine order (such as the LU factorisation).

Topics: Linear algebra, Automatic optimisation of parallel software, Parallel and distributed algorithms.

1 Introduction

In recent years a new technique for the development of efficient software for supercomputers has been developed. The technique is called AEOS (Automatic Empirical Optimisation of Software) [21] and consists in the development of software which is automatically adaptable to new computer architectures in such a way that when a new architecture is developed the library automatically adapts it to the hardware characteristics, thus obtaining highly efficient software for the new architecture. The method is used to alleviate a serious problem in obtaining efficient software, which was traditionally obtained after a large amount of work by a large number of expert programmers.

Attempts have been made to develop this type of software in different fields: FFT [8], sparse systems [15] and dense linear algebra [22].

* Technical Report of the stay at the Innovative Computing Laboratory, funded by Fundación Séneca, reference number EE 00558/CV/01

On the other hand, the development of automatically tuned software would help to facilitate the efficient use of the routines by non-expert users, like those normally using linear algebra routines in the solution of big problems of science or engineering in supercomputers. This has propitiated the development of techniques to facilitate the efficient use of this type of routines in homogeneous [7] or distributed [17] systems. Research in this direction is clearly related to that in automatic tunning, because some of the techniques used to develop automatically tuned software can be used to obtain executions close to the optimum at execution time.

We are researching the development of dense linear algebra software for message-passing systems. Until now we have been working with algorithms of cost $O\left(n^3\right)$: Jacobi methods for the symmetric eigenvalue problem [6], LU factorisation [7] and Gauss elimination [9]. The idea we have been using to tackle the problem has been to identify software and system parameters and to analyse the algorithms both theorically and experimentally in order to determine the influence of the value of the system parameters in the algorithmic parameters. In that way installation routines are developed to enable the installation (or reinstallation) of linear algebra routines in a new (or modified) system. The installation routines estimate the values of the system parameters, and the values of the algorithmic parameters are obtained automatically at execution time. In the routines we have analyzed to date, typical system parameters are: the cost of arithmetic operations of level 1, 2 or 3, and communication parameters (start-up and word-sending time); and algorithmic parameters are: the block size (in algorithms by blocks), and parameters defining the logical topology or the data distribution. The results are satisfactory with these methods, and the execution time obtained with the automatically installed routines is close to the optimum in the different systems [10].

In this work we continue our study of the LU factorisation using the TORC system [25] at the Innovative Computing Laboratory, University of Tennessee. This is a heterogeneous system of 32 nodes (single and dual processors, Pentiums II, III and 4, AMD Athlon and Compaq Alpha, with two communication networks: Fast Ethernet and Myrinet). The system is used by a large number of researchers who share the processors and the communication networks, which causes the load in the processors and/or the communication network to vary greatly. The characteristics of the system are different to those of the systems previously used, and this leads us to extend our research to heterogeneous and/or load variable systems.

We also study another linear algebra routine which has additional difficulties, a routine to solve the Least Square Problem for Toeplitz matrices. The algorithm analyzed is that in [16] and the parallel implementation is in [1]. We are interested in this type of routines because the cost is $O\left(n^2\right)$, and it is more difficult to obtain optimized versions, and even more so, more efficient parallel implementations. This difficulty could provide us some new keys for the development of automatically tuned linear algebra software. In addition, we are working on LS Toeplitz problems for the solution of acoustic virtual problems [20], which means it is necessary to solve the problems in real time, and the obtention of highly efficient routines is specially interesting. Because the routines will be used in different systems (networks of processors and Digital Signal Processors) and by non-expert parallel users, the resulting routine must be capable of being installed adequately in the different systems and must chose the optimum parameters automatically without intervention of those non-expert users.

The rest of the report is divided into three parts: analysis of the Least Square routine, in Section two; the LU factorisation in a variable load system, in Section three; and proposed future work, in Section four.

2 A routine for the solution of the Least Square Problem of Toeplitz matrices

The problem to solve is the Least Square (LS) Problem

$$\max_x ||Tx - b||_2 \tag{1}$$

where $T \in R^{m \times n}$ is Toeplitz ($T_{ij} = t_{i-j}$, $i = 0, 1, \ldots, m-1$, $j = 0, 1, \ldots, n-1$) and $b \in R^m$.

As we have mentioned, we use a parallel implementation [1] of the algorithm proposed in [16].

We begin by analyzing the optimisation of the sequential version of the routine, and after that the parallel version is analyzed.

2.1 Sequential routine

The cost of the algorithm is $O(mn)$ (being $n \leq m$), and to consider the optimisation of the routine we begin by identifying different parts with different basic routines and different costs. Five parts have been identified:

- `TRSM` consists of the solution of triangular systems of equations with multiple right hand side vectors. Three triangular systems of size $n \times n$ are solved using the level 3 BLAS routine `DTRSM`, but because the right hand part consists of three vectors, the cost is $O(n^2)$. In the parallel version the routine `PDTRSM` of PBLAS is used in the solution of the systems. An optimized version of the routine could be used.
- `TRSV` consists of the solution of a triangular system of equations, using the level 2 routine `DTRSV`. The cost is $O(n^2)$, but the execution time is much lower than that of `TRSM`. In the parallel version the routine `PDTRSV` of PBLAS is used in the solution of the system. The reduction in the execution time can be achieved in this case also using an optimized library.
- `LINEAL` consists of the application of level 1 routines (`DDOT`, `DAXPY`, `DSCAL`) and level 2 routines (`DLARFX` and `DORG2R`) on two or three vectors. Thus, the cost is lineal in m or n, and the execution time is very small, and optimisation will not represent an important reduction in the execution time.

- `ROTATIONS` consists of the application of different types of rotations to a matrix, using `DLARFX` and `DROT`. The cost is $O(n^2)$. The use of an efficient library may lead to a reduction in the execution time in this part of the program.
- `GTOMV` performs the matrix-vector operation $y = \alpha T x + \beta y$. It is carried out by using the `DAXPY` routine. Initially it worked on vectors `tr` and `tc` where the first row and column, respectively, were stored. The use of a vector `t` to store `tr` and `tc` allows `DAXPY` to work with bigger vectors. This, and the use of an optimized library, produces a reduction in the execution time. In Table 1 the execution time in this part of the program in the optimized and non-optimized versions is shown, for $m = 4000$ and varying n, in a Pentium III (TORC0) and a Pentium 4 (TORC19), when using ATLAS and reference BLAS. In the table we can see that the reduction in the execution time can be achieved by optimizing the routine and by using an optimized library.

Table 1. Comparison of the execution time (in seconds) in the optimized and non-optimized versions of `GTOMV` when using ATLAS and reference BLAS, on Pentium III and Pentium 4, with $m = 4000$ and n variable.

	TORC0			TORC19		
	ATLAS		BLAS	ATLAS		BLAS
n	non-opt.	opt.	opt.	non-opt.	opt.	opt.
200	0.2304	0.2035	0.2484	0.0853	0.0749	0.0795
400	0.4635	0.3806	0.4326	0.1725	0.1347	0.1565
600	0.7018	0.5780	0.6407	0.2628	0.2042	0.2431
800	0.9558	0.7536	0.8717	0.3562	0.2797	0.3154
1000	1.2320	0.9418	1.1078	0.4530	0.3450	0.4037
1200	1.7784	1.4364	1.6047	0.5470	0.4132	0.4682
1400	2.2598	1.9063	1.9233	0.6509	0.4815	0.5495
1600	2.6316	2.2044	2.2297	0.7530	0.5569	0.6229
1800	2.9793	2.5304	2.6296	0.8519	0.6159	0.7047
2000	3.3388	2.7668	2.7810	0.9574	0.6885	0.7829

Use of different libraries We have seen in Table 1 that the use of an optimized library can produce a reduction in the execution time in some parts of the program. It is possible to use a reference BLAS, a machine specific BLAS, ATLAS installed in the machine, or some of the multiple libraries available free on the web [12]. Experiments have been performed using five libraries: ATLAS, reference BLAS, a BLAS for Pentium II Xeon (BLASII), and two versions of a BLAS for Pentium III (BLASIII). The results obtained with reference BLAS are always worse than those with ATLAS, and the second version of BLASIII always works faster than the first version. Thus, the results obtained with the first version of BLASIII and with reference BLAS are not included.

Tables 2 (Pentium III) and 3 (Pentium 4) show the execution time obtained with the whole routine using different libraries and that which would be obtained using the optimum routine in each part of the program. The optimum library in each part of the program is shown in Tables 4 (Pentium III) and 5 (Pentium 4).

Table 2. Comparison of the execution time (in seconds) of the LS Toeplitz routine, using different basic libraries, and the execution time with the optimum library in each part of the routine. In Pentium III with $m = 4000$ and n variable.

n	ATLAS	BLASII	BLASIII	opt. library
400	0.4394	0.5602	0.5696	0.4341
800	0.9840	1.2409	1.2799	0.9756
1200	1.9612	2.3592	2.4557	1.9488
1600	3.1655	3.6138	3.7845	3.1517
2000	4.2811	4.8081	5.0687	4.2560

Table 3. Comparison of the execution time (in seconds) of the LS Toeplitz routine, using different basic libraries, and the execution time with the optimum library in each part of the routine. In Pentium 4 with $m = 4000$ and n variable.

n	ATLAS	BLASII	BLASIII	opt. library
400	0.1548	0.1041	0.1180	0.1025
800	0.3507	0.2427	0.3046	0.2401
1200	0.5671	0.4056	0.5496	0.4050
1600	0.8341	0.6002	0.8590	0.5999
2000	1.1042	0.8156	1.2431	0.8154

The reduction in the execution time when using the best library in each one of the different parts of the routine is very small in relation with the faster library (normally less than 1%). If we consider there can be small differences in the execution time because the runnings are carried out at different moments, as well as the technical difficulties and the overheads we would incur due to the management of different libraries, then it is better to use a library which is optimized for the machine in the whole program. The cost of the algorithm is small and we have only level 1 and 2 operations, which makes it difficult to obtain a great improvement. With level 3 routines the scene is different, as we will see with the LU factorisation.

To decide the optimum basic library automatically could be interesting, even when only one library is used in the algorithm, because different basic libraries can be used in different algorithms, or in the same algorithm in different machines (in TORC0 the best library is ATLAS, and in TORC19 BLASII).

Table 4. Optimum library in each part of the LS Toeplitz routine. In TORC0 (Pentium III) with $m = 4000$ and n variable.

n	TRSM	LINEAL	GTOMV	ROTATIONS	TRSV
400	ATLAS	BLASII/III	ATLAS	BLASIII	BLASII/III
800	ATLAS	BLASII/III	ATLAS	BLASII/III	BLASII/III
1200	ATLAS	BLASII/III	ATLAS	BLASIII	BLASII/III
1600	ATLAS	BLASII/III	ATLAS	BLASIII	BLASII/III
2000	ATLAS	BLASII/III	ATLAS	BLASIII	BLASII/III

Table 5. Optimum library in each part of the LS Toeplitz routine. In TORC19 (Pentium 4) with $m = 4000$ and n variable.

n	TRSM	LINEAL	GTOMV	ROTATIONS	TRSV
400	ATLAS	BLASII/III	BLASIII	BLASII	BLASII/III
800	ATLAS	BLASII/III	BLASIII	BLASII	BLASII/III
1200	BLASII	BLASII/III	BLASIII	BLASIII	BLASIII
1600	BLASII	BLASII/III	BLASII	BLASII	BLASIII
2000	BLASII	BLASII/III	BLASII	BLASII	BLASII/III

The ATLAS we have used has been installed in TORC0 (Pentium III) and the same ATLAS is used in TORC19 (Pentium 4) because they share the file system. If we install ATLAS in TORC19, it could be the library preferred in this machine, but in the parallel case the routine must be linked with different libraries in both machines (ATLAS installed in each machine).

BLASII is a library which is supposedly optimized for Pentium II and BLASIII for Pentium III, but BLASII usually works better than BLASIII both in TORC0 and TORC19. This means the optimum library for a particular machine is not always clear, and it is also possible not to have a library which is optimized for the machine and then we have to decide between the libraries without any additional information.

For all these reasons, we consider that it is worthwhile developing a methodology to decide automatically the basic library to use in a particular program, or in a particular routine in a program, or even in a particular call to a routine in a program. There are some technical difficulties because the name of the routines are the same in the different libraries, and a technique must be developed to allow the use of different libraries in a program.

2.2 Parallel routine

The parallel version of the routine is analyzed in order to determine if the technique we are normally using with $O\left(n^3\right)$ routines can also be used for routines with lower cost.

Experimental results are shown in [1] in a Pentium II network connected through Myrinet. A speed-up of between 1.95 and 2.40 is obtained when using 8 or 16 processors. The reduction is not very big if we consider the high ratio computation cost/communication cost.

In a system with a smaller ratio the results would be totally different. In Table 6 the execution time of the sequential and the parallel versions are compared when the network used to carry out communications is Fast-Ethernet. In Table 7 the same comparison is made when using Myrinet. In both cases $m = 4000$ and n varies, and only Pentium III have been used, thus the system is homogeneous. The speed-up achieved is also shown. The results shown are not the best obtained: with Fast-Ethernet we show the results using a size 5 block for the distribution of data to the processors, and in Myrinet the block size is approximately $\frac{n}{p}$. The distribution is made with these sizes because they usually give the best results in each case, and this could be the election made using an automatically tuning methodology.

Table 6. Comparison of the sequential and parallel versions of the routine LS Toeplitz, with $m = 4000$ and n variable. In TORC with Fast-Ethernet.

n	seq.	par. 2 pro.		par. 3 pro.	
400	0.4670	1.3471	0.35	2.5547	0.17
800	0.9913	3.1676	0.31	5.3729	0.18
1200	1.9811	5.2374	0.38	7.7288	0.26

Table 7. Comparison of the sequential and parallel versions of the routine LS Toeplitz, with $m = 4000$ and n variable. In TORC with Myrinet.

n	seq.	par. 2 pro.		par. 3 pro.		par. 4 pro.		par. 5 pro.		par. 6 pro.		par. 7 pro.	
400	0.4925	0.5747	0.86	0.5098	0.97	0.5181	0.88	0.6002	0.76	0.6526	0.70	0.6503	0.71
800	1.1418	1.4426	0.79	1.1777	0.97	0.9230	1.06	0.9424	1.07	0.8597	1.14	0.9356	1.05
1200	2.1949	2.6146	0.84	2.2957	0.96	2.1245	0.94	1.6835	1.20	1.4211	1.39	1.5206	1.30

In this algorithm, the algorithmic parameters could be: the number of processors (p) and the block size for the data distribution (b). The system parameters are: the computation cost of operations of level 1 and 2 (k_1 and k_2); and the communication parameters are the start-up and word-sending time (t_s and t_w). Because different level 1 and 2 routines are used, the cost with each one can be different and we have the parameters: $k_{1,dot}$, $k_{1,axpy}$, $k_{1,scal}$, $k_{1,larfx}$, $k_{1,rot}$, $k_{1,others}$, $k_{2,trsm}$ and $k_{2,trsv}$. Thus, we have 10 system parameters. Eight of them are Arithmetic System Parameters (ASP) and two are Communication System Parameters (CSP).

If only the terms of higher order are considered, the execution time of the sequential algorithm is:

$$k_{1,axpy}26nm+k_{1,larfx}8n^2+k_{1,others}3n^2+k_{1,rot}\frac{21}{2}n^2+k_{2,trsm}9n^2+k_{2,trsv}n^2 \quad (2)$$

and the values of these six ASP parameters must be obtained at installation time with the same data access scheme used in the algorithm. If different libraries are considered, the values are obtained for each one of them. In addition, the values can change with the problem size, and the values of the parameters would be obtained for different sizes. In the sequential case the only choice to be made is which library to use in each of the routines and problem sizes.

In parallel it is also possible to use different libraries for communications, but the only library used in the experiments has been MPI [23], `mpich` in Fast-Ethernet and `mpi-gm` in Myrinet. It is also possible to use different values of the communication parameters in the different parts of the program, as we do with the arithmetic parameters formula 2.

In the parallel case, in the Generalized Schur Algorithm the number of communications is reduced from $O\left(\frac{n^2}{b}\right)$ to $O\left(pn\right)$. In Fast-Ethernet this means the cost of communications is $nt_s+\frac{n^2}{b}t_w$, if we consider the communications can start in parallel in the different processors, but they cannot be carried out in parallel in an Ethernet network. What is strange is the fact that experimentally the lowest execution time is obtained with a small value of b. When using the Myrinet network the cost of communictions is $n\left(t_s+t_w\right)$. In the other parts of the program, parallel Givens rotations are applied, and parallel triangular systems are solved. To solve the triangular systems it has been necessary to move the matrix in the system explicitly (it is different in other ScaLAPACK implementations where the movement is possibly done implicitly). More communications are needed when the block distribution is small, and a block size $\frac{n}{p}$ would be preferable, and not a small one, as experimentally obtained in Fast-Ethernet.

Considering only one ASP (k) the parallel time can be approximated by:

$$26\frac{nm}{p}k+\frac{83}{2}\frac{n^2}{p}k+(4np+2n+4)t_s+n\left(n+\frac{5n}{b}+4\right)t_w \quad (3)$$

in Fast-Ethernet, and:

$$26\frac{nm}{p}k+\frac{83}{2}\frac{n^2}{p}k+(4np+2n+4)t_s+n\left(5+\frac{n}{p}+\frac{4n}{p}\right)t_w \quad (4)$$

in Myrinet.

We will use these formulas to try to explain what is happening in the different systems. When considering only one arithmetic parameter, the cost is approximately $k=0.0105$ μsec in TORC (ATLAS has been used). In Pentiums II the value of the parameter can be approximated by $k=0.0434$ μsec. The start-up time is $t_s = 62$ μsec in Fast-Ethernet and $t_s = 125$ μsec in Myrinet. The value of the word-sending time is more difficult to estimate because in the program a lot of small messages of different sizes are sent; a ping-pong has been used to estimate the value for different message sizes.

Using the above values for the parameters and formulas 2, 3 and 4, the best theoretical choice in TORC for the problem sizes of Tables 6 and 7 would normally be to solve the problem in sequential.

To solve the problem in sequential is always the best choice when using Fast-Ethernet (Table 3), but it is not the best choice with Myrinet and $n = 800$ or 1200. Thus, the method does not always predict well, but we must consider the formulas are very simplified and possibly with more detailed formulas and including the overlap of computation and communication (not to include this overlap means that the theoretical choice gives less processors than the experimental results), the election could be better.

In a network of Pentium II with Myrinet the execution time in the sequential case, the optimal experimental time and the execution time with the number of processors provided by the model, are those shown in Table 8 (data are taken from [1]), where the number of processors used and the speed-up are also shown. We can see the model does not provides that the best number of processors, and the difference between the optimum execution time and the execution time with the number of processors provided by the model is about 58%.

Table 8. Comparison of optimum execution time and the execution time with the parameters provided by the model. Routine LS Toeplitz in network of Pentium II with Myrinet.

$m \times n$	seq.	optimum			modeled		
		time	proc.	speed-up	time	proc.	speed-up
1000×500	0.868	0.341	16	2.55	0.868	1	1.00
1000×1000	2.144	0.988	8	2.17	1.715	2	1.25
2000×200	0.600	0.178	16	3.37	0.383	2	1.57
2000×1000	3.630	1.068	16	3.40	1.580	4	2.30
2000×2000	9.225	2.922	16	3.16	4.151	4	2.22

For algorithms with a small cost (as this we are considering) it could be preferable to perform some selected executions at installation time to decide the values of the parameters, and only to use the model for bigger problem sizes [9].

These results lead us to conclude that the use of the technique of parameters election at installation time can be used successfully to provide the users with routines whose execution time is close to the optimum, even in the case of routines with low cost. But to do this, it is necessary to develop a more detailed model of the algorithm.

2.3 Using several libraries in the parallel case

The preferred library in the sequential case could be considered also as preferred to be used in the parallel routine. In that way ATLAS would be used in TORC when using Pentium III (Table 2). In table 9 the execution time when using the different libraries in the parallel routine is compared. The lowest execution time is marked for each matrix and number of processors. We can see the best library is now BLASII, which is the second in the sequential case. It can be due to the fact that in parallel we work with smaller vectors in each processor, and the preferred library with the sizes experimented in the sequential case can not be the preferred for smaller sizes. Anyway, the differences in the execution time are very small and could also have been motivated for other users in the system.

Table 9. Comparison of the execution time (in seconds) of the parallel routine with different basic libraries. Routine LS Toeplitz in network of Pentium III with Myrinet.

processors:	1	2	3	4
ATLAS				
400	**0.4566**	0.5870	0.5280	0.4151
800	**1.0109**	1.4078	1.1537	1.1023
1200	**1.9772**	2.6694	2.3173	1.8172
BLASII				
400	0.6115	**0.5554**	**0.5088**	**0.4073**
800	1.2751	**1.3887**	**1.1412**	**0.9650**
1200	2.4280	2.5838	**2.1818**	**1.7185**
BLASIII				
400	0.6329	0.5850	0.5180	0.4170
800	1.3183	1.4244	1.1862	1.0444
1200	2.5290	**2.5745**	2.2229	2.0067

2.4 Selection of the value of the algorithmic parameters at execution time

In the sequential case, if we can find a formula for the execution time $t(\textit{problem size}, \textit{system parameters}, \textit{algorithmic parameters})$, for a particular *problem size* (*pps*) the values of the algorithmic parameters are obtained from the formula.

For example, in our case, in sequential, where the only algorithmic parameter is the library, when we consider the same library in the whole program, we have three formulas $t_{l_1}(n,m)$, $t_{l_2}(n,m)$ and $t_{l_3}(n,m)$. If the *pps* are n_0 and m_0, $t_{l_1}(n_0,m_0)$, $t_{l_2}(n_0,m_0)$ and $t_{l_3}(n_0,m_0)$ are obtained, and the library which provided the lowest execution time is selected. In fact, the program must be linked with the three libraries, and another routine makes the selection and calls the program.

In some cases, it is imposible to obtain good formulas for the execution time (for example, if the program uses subroutines whose behaviour we do not know exactly). In this case, we have tables like in Table 9, and for a problem size n_0, m_0 the optimum values of the algorithmic parameters are obtained from the tables, using the problem size entry closest to n_0, m_0. With $m = 3000$ and $n = 500$ the selected library would be ATLAS in the sequential case, and BLASII and 4 processors in the parallel case. Tables can be obtained at installation time by running the routine for different problem sizes, number of processors, and libraries. It is not very time consuming with a routine of low cost such as LS Toeplitz, but the time can become too long for routines of higher cost, and in these cases the tables can be partially filled by performing the experiments for the smallest problem size and by using the optimum values obtained for the algorithmic parameters as initial values of a local search for the next problem size: the initial value could be $m = 200$, $n = 200$ and the optimum value of $p = 3$, then, for $m = 400$, $n = 200$, it may be necessary to execute the routine for $p = 3, 4, 5, 6$, to obtain the lowest execution time when $p = 5$. This can be done for numeric parameters (number of processors, block size, number of rows of processors, ...) but not for non-numeric parameters such as the library.

The values of the parameters to experiment with may be a set of significative values: a 2D mesh configuration with no excess of either rows or columns (LU factorisation), block size 16, 32, 64, 96 and 128, for algorithms by blocks, etc. And in some cases the search for the optimum is carried out by performing a guided search: if we obtain the execution time for $b = 16$, 64 and 128 and the lowest time is obtained for 64, the search continues with $b = 40$, 64 and 96, if the lowest execution time is obtained in 96 the search continues with $b = 64$, 96 and 128, etc. Similar techniques were used in [9], and in [19] for the automatic tuning of communications.

2.5 Refining the estimations

The selection could be better if we use all the possible values of the algorithmic parameters. To do this a table of experiments with a selected number of values is not enough. The values of the constants in the formula can be obtained by a minimum square adjustment.

In the sequential case the execution time has the form $t(m,n) = amn + bn^2$. Thus, if we have experiments for problem sizes $(m_1, n_1), \ldots, (m_k, n_k)$, with execution times $t_1, \ldots, t_k$, the function to minimize is $\sum_{i=1}^{k} \left(t_i - am_in_i - bn_i^2\right)$, and the minimum can be calculated by obtaining the partial derivatives with respect to a and b, by equalling zero, and by solving the system:

$$\begin{array}{lcl} a\sum_{i=1}^{k} m_i^2n_i^2 + b\sum_{i=1}^{k} m_in_i^3 & = & \sum_{i=1}^{k} t_im_in_i \\ a\sum_{i=1}^{k} m_in_i^3 + b\sum_{i=1}^{k} n_i^4 & = & \sum_{i=1}^{k} t_in_i^2 \end{array} \quad (5)$$

The values of the parameters are obtained for each libary, which is done at installation time, with the values $t_i\,(m_i, n_i)$ obtained with a reduced number of executions.

When the values of the constants depend on the values of the algorithmic parameters, and are obtained by running estimation routines, the execution time is not obtained by runnings of the routine at installation time, but by substituting the values of the system parameters in the theoretical model.

If different libraries are used in different parts of the program, the problem becomes more complicated. For example, if three parts and two libraries are considered, we would have different values of parameter a: a_{ijk} (with $i, j, k = 1, 2$) indicates the parameter a has been obtained using libraries l_i, l_j and l_k, respectively, in the three parts of the program. After the values a_{ijk} have been obtained, the constants with the different libraries in each part of the program (a_{ij}, with $i = 1, 2, 3$ and $j = 1, 2$, represent the parameter in part i using library l_j) are obtained by solving the equation system formed by the equations $a_{ei} + a_{ej} + a_{ek} = a_{ijk}$. We need to obtain $number\ of\ parts \times number\ of\ libraries$ constants, thus, this number of values of a_{ijk} must be computed.

The same happens in the parallel case, but more parameters are computed. We can consider the formula $t(m, n, p) = a\frac{mn}{p} + b\frac{n^2}{p} + cn + dnp + en^2 + f\frac{n}{p}$, where terms of lower order are considered because they can become important, due to the high cost of communications with respect to computations, or because some terms are divided by p and can become less important when the number of processors increases.

2.6 Systems with variable load

A static installation can be useful in systems with a fixed load, but not in variable load systems. In these systems the installation process we have explained cannot be carried out at execution time. What could be done is to carry out the process at installation time, to obtain (for example, using the Network Weather Service [24]) the load of the system at execution time, and to affect the values obtained at installation time using those reflecting the system load.

Because the CPUs and the communication network can have different load, it is necessary to distinguish between computation and communication time. Let us consider six processors with relative speeds 1, 1, 0.8, 0.6, 0.5 and 0.5 (1 indicates the processor is not being used) and the communications in the network have a speed of 0.7 of the total speed. Executing the program in sequential in one of the processors with relative speed 1 would give us a time $t_{seq}(n)$; the parallel time with the two processors of relative speed 1 is $t_{ari}(n, 2) + 0.7t_{com}(n, 2)$; the execution time using the first three processors can be approximated by $0.8t_{ari}(n, 3) + 0.7t_{com}(n, 3)$; with the first 4 processors, by $0.6t_{ari}(n, 4) + 0.7t_{com}(n, 4)$; with 5 processors by $0.5t_{ari}(n, 5) + 0.7t_{com}(n, 5)$; and with the 6 processors by $0.5t_{ari}(n, 6) + 0.7t_{com}(n, 6)$. The execution would be carried out with the number of processors and the library which provide the lowest execution time.

3 LU factorisation

As we mentioned in the Introduction, during the stay at the Innovative Computing Laboratory we continued our previous studies on the automatic optimisation of the classical parallel LU factorisation [11]. We have worked with a heterogeneous network of processors (TORC) trying to extend our research to variable load systems.

The same aspects of the automatic optimisation of linear algebra software analyzed for the LS Toeplitz routine (use of different libraries and installation routines) will be considered. In addition, the efficient use of the LU factorisation on variable load systems is analyzed, taking into account the possibility of using the information obtained during the installation [7] with information obtained at execution time, through specific routines or some tool such as the Network Weather Service [24].

3.1 Use of different libraries

Sequential case The theoretical cost of the sequential block LU factorisation is:

$$\frac{2}{3}k_3n^3 + bk_3n^2 \tag{6}$$

where the main term is obtained from matrix-matrix multiplications of size $c_i \times b$ by $b \times c_i$, with b the block size and c_i with values $n-b$, $n-2b$, ..., b. Thus, the value of k_3 depends on the values of n and b, and $k_3(n,b)$ could be obtained at installation time, and used at execution time to decide the optimum value of b (the algorithmic parameter) for the problem size in question. To store the optimum value of b for a number of values of n would be enough, and for a problem size n_c, the value of b associated to the n closest to n_c would be used.

If several libraries are used, the value of k_3 also depends on the library used, and $k_3(n,b,l)$ could be obtained at installation time, but by performing executions for different values of n, b and l (the different libraries). l is considered as another algorithmic parameter, and the information generated at installation time consists of different values of n and the optimum block size and library for each value. We have the same problem as with the LS Toeplitz routine: how to manage different libraries with routines of the same names. In the Toeplitz case several level 1 and 2 routines participate in the highest order term $(O\left(n^2\right))$, and different constants could be obtained for the different kernels. In the block LU factorisation, only one routine (the matrix-matrix multiplication) appears in the highest order term; thus, analysing only the matrix-matrix multiplication provides a good estimation [7], and the basic library management is only a small problem because we need to link the routine only with the preferred library.

To determine the influence of the library in the execution time, the execution time of the matrix-matrix multiplication for different libraries and values of b is shown in Tables 10 (Pentium III) and 11 (Pentium 4). Results obtained with reference BLAS and the previous version of BLASIII are not shown because they are always worse than those shown (as in LS Toeplitz). The value of n does not influence the value of k_3. In the tables, the optimum value of k_3 is highlighted for each block size. We can see that the optimum library and block size depend on the machine: BLASIII with $b = 128$ in Pentium III, and ATLAS with $b = 256$ or BLASII with $b = 32$ or 64 in Pentium 4. If we compare with Tables 2 and 3, we can see that in the LS Toeplitz routine the preferred library was ATLAS in Pentium III, and BLASII in Pentium 4. These are different to the preferred libraries for the LU factorisation, and this leads us to think that the automatic selection of the library is a good idea.

Table 10. Estimation of k_3 (in μsec) for different block sizes and libraries. In Pentium III.

block size:	16	32	64	128	256
ATLAS	**0.0036**	0.0030	0.0028	0.0027	0.0025
BLASIII	0.0039	**0.0029**	**0.0026**	**0.0023**	**0.0024**
BLASII	0.0038	**0.0029**	0.0029	0.0030	0.0034

Table 11. Estimation of k_3 (in μsec) for different block sizes and libraries. In Pentium 4.

block size:	16	32	64	128	256
ATLAS	0.0013	0.0011	0.0011	**0.0011**	**0.0010**
BLASIII	0.0017	0.0014	0.0011	**0.0011**	**0.0012**
BLASII	**0.0012**	**0.0010**	**0.0010**	**0.0011**	0.0013

Tables 12 and 13 show the optimum execution time with the libraries considered (and the block size with which the optimum is obtained), the execution time with the parameters provided by the model, and LAPACK linked with the different libraries, Pentium III (Table 12) and Pentium 4 (Table 13).

The optimum values for the different matrix sizes are highlighted. We observe that the parameters (library and block size) provided by the model are normally those with which the optimum is obtained. There is a very small difference in Pentium 4 and matrix size 1024, but this can be due to executions at different moments. In Pentium III and matrix size 512. There is also a difference, in this case bigger. This may be due to the relatively small size of the matrix and to the use of a not very detailed model (Formula 6). The model could be improved, but for such small differences in the execution time, we think it is not worthwhile. The best execution time when using LAPACK is also obtained with the preferred library in each machine; thus, the technique of consider the basic library as an algorithmic parameter can also be useful for existing libraries.

Table 12. Execution time (in seconds) of the LU routine and the corresponding routine of LAPACK, with different basic libraries. In Pentium III.

	ATLAS		BLASII		BLASIII			LAPACK		
	time	blo.	time	blo.	time	blo.	mod.	ATLAS	BLASII	BLASIII
512	0.32	32/64	**0.28**	32	**0.28**	32/64	0.32	0.37	0.35	0.33
1024	2.32	128	2.19	64	**1.96**	128	**1.96**	2.45	2.44	2.23
1536	7.25	128	7.08	32	**6.15**	128	**6.15**	7.83	7.82	7.11
2048	16.67	128	16.67	32	**14.04**	128	**14.04**	18.02	18.14	16.29

Table 13. Execution time (in seconds) of the LU routine and the corresponding routine of LAPACK, with different basic libraries. In Pentium 4.

	ATLAS		BLASII		BLASIII			LAPACK		
	time	blo.	time	blo.	time	blo.	mod.	ATLAS	BLASII	BLASIII
512	0.11	32	**0.10**	32	0.12	64	**0.10**	0.13	0.12	0.11
1024	0.86	32/64	0.84	32	**0.83**	128	0.84	0.93	0.85	0.90
1536	2.83	64	**2.44**	32	2.62	128	**2.44**	2.98	2.79	3.05
2048	6.54	64	**5.75**	32	6.06	128	**5.75**	6.86	6.42	7.14

Parallel case The cost of the parallel block LU factorisation can be modelled by the formulas:

$$\frac{2}{3}k_3\frac{n^3}{p} + \frac{r+c}{p}bk_3n^2 + \frac{1}{3}b^2k_2n \tag{7}$$

for the arithmetic cost, and:

$$t_s\frac{2n\max(r,c)}{b} + t_w\frac{2n^2\max(r,c)}{p} \tag{8}$$

for the communication cost. The matrix is distributed in a logical two-dimensional mesh (in ScaLAPACK style [3]) of $p = r \times c$ processors, and using a cyclic-checkerboard distribution [14] with block size b.

The system parameters are k_3, k_2, t_s and t_w; and the algorithmic parameters b, r and c, but the library (l) can also be considered an algorithmic parameter.

Thus we have $k_3(b,r,c,l)$, and the same with the other system parameters. However, we will consider as variable only k_3 because the variation of the other system parameters with respect to the algorithmic parameters is negligible.

Experiments have been done in TORC using four Pentium III and the Myrinet network, and in processors with low workload; thus, the system can be considered homogeneous and with a fixed load. The values of the communication system parameters are $t_s = 125$ μsec and $t_w = 0.09$ μsec, and the values of k_3 for different libraries are those in Table 10. In Table 14 the theoretical (the.) and optimum (opt.) execution times, the execution time obtained with the parameters provided by the model (mod.), and the associated block size are shown for different libraries and problem sizes. The lowest optimum and modelled execution times for each matrix size are highlighted. There is a difference of approximately 7% between the theoretical and the experimental time. This means the model or the estimation of the system parameters could be improved, for example by taking into account the overlapping of communication and computation. Nevertheless, the model provides good values for the parameters. We can see the best results are obtained with BLASIII and BLASII, as predicted by the model. Also the optimum block size (32) is well predicted by the model (the only difference is in ATLAS and $n = 1536$). The difference between the libraries is less important now than in the sequential case (Table 12) due to the cost of communications, which is common to all the libraries. The preferred library is the same in the sequential and the parallel case. This is not the same in the LS Toeplitz routine. Thus, considering the library as an algorithmic parameter is also useful in the parallel case.

Table 14. Execution time (in seconds) of the parallel LU routine, with different basic libraries. In TORC with 4 Pentium III and Myrinet.

ATLAS						
	the.		opt.		mod.	
	time	blo.	time	blo.	time	blo.
512	0.13	32	0.12	32	0.12	32
1024	0.79	32	0.74	32	0.74	32
1536	2.36	32	2.21	64	2.27	32
BLASII						
	the.		opt.		mod.	
	time	blo.	time	blo.	time	blo.
512	0.13	32	**0.11**	32	**0.11**	32
1024	0.77	32	0.71	32	0.71	32
1536	2.30	32	**2.13**	32	**2.13**	32
BLASIII						
	the.		opt.		mod.	
	time	blo.	time	blo.	time	blo.
512	0.13	32	**0.11**	32	**0.11**	32
1024	0.77	32	**0.70**	32	**0.70**	32
1536	2.30	32	**2.13**	32	**2.13**	32

3.2 Parallel systems with variable load

Although the installation methodology we have developed ([6, 10, 7]) was conceived for low variable load systems, it can be used in other systems, but with less predictable results.

Our aim is to extend our methodology in order to include in the system parameters of the analytical model not only the static characteristics of the system obtained when the routine is installed, but also its state when the routine is executed. In this way, the model would be able to make an accurate theoretical prediction of the execution time, even when the load at execution time is very different from that assumed at instalation time. This enhanced model will constitute a better tool for selecting the optimum values of the system parameters in any situation.

One approach which takes into account the system state at execution time involves obtaining the values of the system parameters at execution time (as is done in GRADS [2]), but the high overhead makes this method suitable only for very large problems [17]. Therefore, we propose to perform a static installation to obtain the optimum values of the algorithmic parameters with some values of the system parameters (those at installation time) and to refine these initial values using information of the system parameters obtained at the execution time. In this way, the overhead incurred would be lower when using a tool like NWS [24], and the method could be suitable for smaller problems.

The Network Weather Service (NWS) is a tool that provides measurements and predictions of the particular features of a system at a given time. The current implementation of NWS includes measuring the fraction of CPU available for new processors and for the current ones, TCP connection time, end-to-end TCP network latency, and end-to-end TCP network bandwidth. NWS can be used in a LAN as well as in the GRID. In the former, the overhead introduced is almost negligible [17].

Sequential case: variable CPU load. Firstly, the arithmetic calculation has been studied, and the way in which the values of the parameters of the model are affected by the load of the CPU has been analysed. The strategy, according to the methodology proposed, consists of three main steps: design, installation and dynamic adjustment at execution time. The first two steps belong to the methodology we proposed in previous works (Formula [6]), whereas the third is the new contribution of this study. The description of the three steps is:

1. Design: The designer of the routine obtains a static analytical model of the routine execution time (Formula 6) as a function of the problem size n, the system parameters (in the case of the sequential LU the only system parameter is k_3) and the algorithmic parameters (in the case of the sequential LU the only algorithmic parameter is the block size b).

2. Installation: The System Manager executes the installation routine that carries out the estimations of the system parameters values. This process must be done when the system has a minimum load (close to 100 % of available CPU), in order to obtain system parameter values that reflect the static characteristics of the system.

3. Dynamic adjustment at execution time: when the user calls the routine for a specific problem size n the next substeps are carried out:

3.1. The NWS is called, and it reports the fraction of available CPU, f.

3.2. The values of k_3 are updated, according to the current state, in this way:

$$k_{3-current} = \frac{k_{3-installation}}{f} \tag{9}$$

3.3. By applying the current values of k_3 and the problem size n chosen by the user to the model (Formula 6), the optimum values of the algorithmic parameters (in this case, only b) are obtained.

3.4. Finally, the routine is executed using the algorithm parameters obtained.

The resulting dynamic model is the union of the basic static model (6) and the dynamic adjustment of the system parameters (9).

Experimental results of the sequential case. Experiments for the sequential LU routine by blocks have been carried out on a Pentium III. Table 15 shows the values obtained for k_3 during the installation process depending on b. The NWS skill *cpuMonitor* has been used to measure the load of the CPU at execution time. This tool monitors the fraction of CPU available for newly-started and existing processes. The different CPU loads have been obtained by executing several images of a sequential application which is independent of the LU routine.

Table 15. Estimation of k_3 (in μsec) for different block sizes. In Pentium III.

block size:	16	32	64	128
$k_{3installation}$	0.0038	0.033	0.0030	0.0027

Table 16, shows the theoretical optimum b for different loads of the CPU according to the dynamic model proposed.

Table 16. Values of the optimum block size for different problem sizes and workloads. In Pentium III.

	NWS currentCPU			
n	100 %	70 %	40 %	30 %
512	32	32	64	128
1024	64	64	128	128
1536	64	128	128	128
2048	64	128	128	128
2560	128	128	128	128
3072	128	128	128	128

In Tables 17, 18, 19 and 20 there are comparisons between:

SM-the: the theoretical execution time predicted by the static model.

DM-the: the theoretical execution time predicted by the dynamic model.

opt-exp: the experimental optimum execution time, obtained by executing the routine for the different values of the algorithmic parameters and taking the lowest.

SM-exp: the experimental execution time, taking the system parameters values predicted as optimum by the static model.

DM-exp: the experimental execution time, taking the system parameters values predicted as optimum by the dynamic model.

Table 17. Comparison of the optimum time predicted with a static model (SM-the), the optimum experimental time (opt-exp) and the experimental time with the parameters provided by the static model (SM-exp), with 100% CPU availability. In Pentium III.

n	SM-the	opt-exp	SM-exp	dev. SM-exp
512	0.36	0.33	0.33	0%
1024	2.62	2.28	2.28	0%
1536	8.30	7.20	7.31	2%
2048	18.68	16.60	16.60	0%
2560	35.23	31.68	31.68	0%
3072	59.43	54.25	54.25	0%

Table 18. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with 70% CPU availability. In Pentium III.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.36	0.82	0.44	0.56	0.56	38%	38%
1024	2.62	3.69	3.83	4.19	3.83	9%	0%
1536	8.30	11.92	10.93	12.09	12.09	11%	11%
2048	18.68	27.63	29.32	32.31	29.32	10%	0%
2560	35.23	52.90	51.17	51.17	51.17	0%	0%
3072	59.43	89.96	87.15	87.15	87.15	0%	0%

In Table 17 we can observe the basic case, when the execution of the routine is made with the same CPU load as when the routine was installed. In this situation, the static model produces a good theoretical estimation of the execution time (SM-the). An accurate choice of the algorithmic parameter values is

Table 19. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with 40% CPU availability. In Pentium III.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.36	0.82	0.44	0.56	0.56	27%	27%
1024	2.62	6.14	6.00	6.71	6.71	12%	12%
1536	8.30	20.16	19.59	23.74	19.59	21%	0%
2048	18.68	46.71	44.30	44.30	44.30	0%	0%
2560	35.23	89.98	84.24	84.24	84.24	0%	0%
3072	59.43	154.04	144.25	144.25	144.25	0%	0%

Table 20. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with 30% CPU availability. In Pentium III.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.36	1.06	0.84	1.32	1.30	57%	57%
1024	2.62	7.95	8.70	8.77	8.70	1%	0%
1536	8.30	25.94	29.71	33.13	31.42	12%	6%
2048	18.68	60.40	64.05	69.95	69.95	9%	0%
2560	35.23	116.72	117.25	117.25	117.25	0%	0%
3072	59.43	200.24	202.48	202.48	202.48	0%	0%

carried out. Therefore, experimental execution times (SM-exp) are very close to the optimum ones (opt-exp).

Tables 18, 19 and 20 show that when the CPU load grows the static model produces unrealistic theoretical estimations of the execution time (SM-the), which causes a wrong choice of the algorithmic parameters, leading to an experimental execution time (SM-exp) far from the optimum (exp-opt). On the other hand, using the dynamic model leads to more accurate theoretical estimations (DM-the) and a better choice of algorithmic parameters values is made. Experimental execution times (DM-exp) are, therefore, close to the optimum ones.

Parallel case I: variable traffic in the communication network. This Section studies how the traffic in the interconnection network affects the parameters values. Following the strategy introduced for the sequential case, there are three steps:

1. Design: The designer of the routine obtains a static model of the routine execution time (7, 8) as a function of the problem size n, the system parameters and the arithmetic parameters. In this case, the system parameters studied are those pertaining to communications (t_s and t_w), and the algorithmic parameters (p, r, c and b).

2. Installation: The System Manager executes the installation routine that carries out the estimations of the system parameter values. This process must be done when the network has minimum traffic, in order to obtain system parameter values that reflect the static characteristics of the system.

3. Dynamic tuning at execution time: when the user calls the routine for a specific problem size n the following substeps are carried out:

3.1. The NWS is called and it reports the bandwidth available, and hence, the word sending time (t_w) for a specific message size, for example, for m=32768.

3.2. The correction factor of t_w is calculated:

$$f = \frac{t_{w-installation}(m = 32768)}{t_{w-current}(m = 32768)} \tag{10}$$

3.3. The values of t_w are updated, according to the current situation:

$$t_{w-current}(m) = \frac{t_{w-installation}(m)}{f} \tag{11}$$

3.4. The optimum values of the algorithmic parameters are obtained by applying the current values of t_w (11) and the problem size n chosen by the user to the model (7, 8).

3.5. Finally, the routine is executed using the algorithmic parameters obtained.

Therefore, the resulting dynamic model used is the union of the basic static model (7, 8) and the dynamic adjustment of the system parameters (10, 11).

Experimental results of parallel case I. Experiments with the parallel LU have been carried out using 4 nodes of TORC (all of them are dual Pentium III, but only one processor per node has been used). The interconnection network used has been Fast-Ethernet. Table 21 shows the values obtained, in the installation step, for t_w depending on m.

In order to measure the traffic of the network at execution time the NWS skill $tcpMessageMonitor$, which monitors the TCP bandwidth and latency between each pair of a set of machinesm, has been used. The variations in the traffic of the network have been obtained by executing different images of a parallel program that basically performs communications between the nodes used in these experiments.

In Table 22, the theoretical optimum b is shown according to the proposed dynamic model, for different network traffics, that is, for different word-sending times.

In Table 23 we can observe the basic case, when the execution of the routine is done with similar network traffic as when the routine was installed. In this situation, the static model produces a good theoretical estimation of the execution time (SM-the). An accurate choice of the algorithmic parameter values is carried out. Thus, experimental execution times (SM-exp) are very close to the optimum ones (opt-exp).

As in the sequential case, in Tables 24, 25 and 26 we can observe that, when the network traffic grows, the static model produces incorrect theoretical estimations of the execution time (SM-the), which, in turn, causes an unsatisfactory choice of the algorithmic parameters values, and thus, an experimental execution time (SM-exp) which is far from the optimum (exp-opt). On the other hand, with the dynamic model, the theoretical estimations (DM-the) are more accurate, leading to a better choice of the algorithmic parameters. Thus, the experimental execution times (DM-exp) are close to the optimum ones.

So far the viability of the dynamic model has been shown separately for variations of CPU availability (in the previous section) and for variations of the traffic of the interconnection network (in this section).

Table 21. Estimation of t_w (in μ sec) for different message sizes. In 4 Pentium III.

message size	32768	262144	1048576	2097152
$t_{winstallation}$	0.7000	0.6900	0.6800	0.6750

Parallel case II: variable network traffic and CPU availability. In this Section, a combination of the ideas presented in the two previous sections is shown. The resulting dynamic model is the union of the static parallel model (7, 8) and the dynamic adjustment of the system parameters (9, 10, 11).

Table 22. Values of the optimum block size for different problem sizes and network workloads. In 4 Pentium III.

	NWS: $t_{w-current}$			
n	0.7 μs	1.5 μs	4.0 μs	7.0 μs
512	32	32	32	32
1024	32	64	64	64
1536	64	64	64	64
2048	64	64	64	128
2560	64	64	128	128
3072	64	128	128	128

Table 23. Comparison of the optimum time predicted with a static model (SM-the), the optimum experimental time (opt-exp) and the experimental time with the parameters provided by the static model (SM-exp), with $t_w = 0.7\mu$s. In 4 Pentium III.

n	SM-the	opt-exp	SM-exp	dev. SM-exp
512	0.30	0.25	0.25	0%
1024	1.47	1.36	1.36	0%
1536	3.86	3.22	3.22	0%
2048	7.85	6.76	6.76	0%
2560	13.81	11.81	11.81	0%
3072	21.90	19.28	19.41	1%

Table 24. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with $t_w = 1.5\mu$s. In 4 Pentium III.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.30	0.37	0.23	0.23	0.23	0%	0%
1024	1.47	1.78	1.30	1.37	1.37	5%	5%
1536	3.86	4.56	3.43	3.43	3.43	0%	0%
2048	7.85	9.09	7.10	7.10	7.10	0%	0%
2560	13.81	15.75	12.78	12.78	12.78	0%	0%
3072	21.90	24.65	21.40	29.25	21.40	37%	0%

Table 25. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with $t_w = 4.0\mu$s. In 4 Pentium III.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.30	1.16	0.43	0.43	0.43	0%	0%
1024	1.47	4.90	3.92	3.92	4.02	0%	3%
1536	3.86	11.55	11.27	11.27	11.27	0%	0%
2048	7.85	21.48	21.40	21.40	21.40	0%	0%
2560	13.81	34.96	36.48	38.81	36.48	6%	0%
3072	21.90	52.20	59.70	62.91	59.70	5%	0%

Table 26. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with $t_w = 7.0\mu$s. In 4 Pentium III.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.30	1.95	0.73	2.22	2.22	204%	204%
1024	1.47	8.02	5.49	6.58	6.59	20%	20%
1536	3.86	18.55	17.01	17.19	17.19	1%	1%
2048	7.85	33.87	38.04	39.30	38.04	3%	0%
2560	13.81	54.15	64.33	66.66	64.33	4%	0%
3072	21.90	79.75	87.98	98.54	87.98	12%	0%

Experimental results of parallel case II. Experiments with the parallel LU for this section have been carried out using eight nodes of the TORC, using only one processor per node. The interconnection network was Fast-Ethernet.

The NWS skills $cpuMonitor$ and $tcpMessageMonitor$ have been used. The different CPU loads and the variations in the network traffic have been obtained by executing different images of a parallel program which performs arithmetic calculations and communications between the nodes used in these experiments.

The theoretical optimum b is shown in Tables 27 (4 nodes) and 28 (8 nodes), according to the dynamic model proposed and for different loads of the parallel platform (different CPU loads and network traffic).

In Tables 23 and 29 the basic cases are shown, with the minimum load in the CPU and in the network. Again, the static model produces good results. However, when the platform load grows, the static model gives worse results, as we can see in Tables 29, 30, 32 and 33.

As in the two previous studies, when the load grows the theoretical estimations (DM-the) with the dynamic model improve with respect to the static model ones. A better choice of algorithmic parameters values is made. Thus, the experimental execution times (DM-exp) are close to the optimum ones (opt-exp).

In this section the viability of the proposed methodology for different parallel platform loads has been shown. It would be convenient to study cases of heterogenous load where some nodes of the system have more load than the others at the moment of the execution of the routine. In the next section an introduction to this study is shown.

Table 27. Values of the optimum block size for different problem sizes and platform workloads. In 4 Pentium III.

	NWS:current CPU / $t_{w-current}$		
n	100% / 0.7 μs	70% / 1.5 μs	35% / 7.0 μs
512	32	32	64
1024	32	64	128
1536	64	64	128
2048	64	128	128
2560	64	128	128
3072	64	128	128

Parallel case III: variable and heterogeneous load in the communication network and in the CPU availability. In this section is studied the situation when the platform load is not distributed homogeneously, but, as it is more common, there are some nodes with bigger load than others. In this situation, and with a routine with a homogeneous distribution of the work, like the parallel by block LU studied, the execution rhythm is beated by the processors with worse calculation and communications features. Therefore, the values

Table 28. Values of the optimum block size for different problem sizes and platform workloads. In 8 Pentium III.

	NWS:current CPU / $t_{w-current}$		
n	100% / 0.7 μs	70% / 2.0 μs	60% / 5.5 μs
512	32	64	64
1024	32	64	64
1536	64	64	64
2048	64	64	128
2560	64	128	128
3072	64	128	128
3584	64	128	128
4096	128	128	128

Table 29. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with $t_w = 1.5\mu$s and with 70% CPU availability. In 4 Pentium III.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.30	0.54	0.46	2.22	2.22	383%	383%
1024	1.47	2.51	4.98	5.35	4.98	7%	0%
1536	3.86	6.45	8.71	8.71	8.71	0%	0%
2048	7.85	12.79	16.70	17.01	16.70	2%	0%
2560	13.81	21.90	24.84	26.30	24.84	6%	0%
3072	21.90	34.32	39.24	39.24	39.24	0%	0%

Table 30. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with $t_w = 7.0\mu$s and with 35% CPU availability. In 4 Pentium III.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.30	2.05	1.70	7.00	6.00	312%	253%
1024	1.47	8.89	10.66	15.49	10.66	45%	0%
1536	3.86	21.36	24.00	27.45	24.00	14%	0%
2048	7.85	40.38	40.47	41.69	40.47	3%	0%
2560	13.81	66.87	64.17	67.36	64.17	5%	0%
3072	21.90	101.73	92.11	92.11	92.11	0%	0%

Table 31. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with $t_w = 0.7\mu$s and with 100% CPU availability. In 8 Pentium III.

n	SM-the	opt-exp	SM-exp	dev. SM-exp
512	0.24	0.25	0.27	8%
1024	1.10	0.93	0.99	6%
1536	2.77	2.51	2.56	2%
2048	5.41	4.98	4.98	0%
2560	9.22	8.42	8.42	0%
3072	14.38	13.81	12.81	0%
3584	21.08	19.55	19.55	0%
4096	29.43	27.65	29.31	6%

Table 32. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with $t_w = 2.0\mu$s and with 70% CPU availability. In 8 Pentium III.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.24	0.61	0.51	1.31	0.81	157%	59%
1024	1.10	2.62	3.03	3.10	3.03	2%	0%
1536	2.77	6.28	6.64	6.64	6.64	0%	0%
2048	5.41	11.85	12.31	13.04	13.04	6%	6%
2560	9.22	19.59	19.77	21.34	19.77	8%	0%
3072	14.38	29.56	29.92	29.92	30.63	0%	2%
3584	21.08	42.07	43.55	44.01	43.55	1%	0%
4096	29.43	57.34	60.01	60.01	60.01	0%	0%

Table 33. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with $t_w = 5.5\mu$s and with 60% CPU availability. In 8 Pentium III.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.24	1.53	2.06	3.12	3.22	51%	56%
1024	1.10	6.32	9.36	10.17	9.99	9%	7%
1536	2.77	14.65	15.17	17.80	17.80	17%	17%
2048	5.41	26.73	25.30	25.34	25.30	0%	0%
2560	9.22	42.80	40.01	42.01	40.01	5%	0%
3072	14.38	63.11	56.42	58.07	56.42	3%	0%
3584	21.08	87.92	78.88	79.76	78.88	1%	0%
4096	29.43	117.48	108.34	112.12	112.12	3%	3%

of the system load (CPU availability and word sending time), in order to do the dynamic adjustment of the model at execution time, will correspond to the processor with more load.

In Tables 34, 35 and 36 the results for 4 nodes are shown (the basic case of the static situation was shown in Table 23). Only one of the nodes has been overload by executing several images of an application, which is independent of the LU routine.

We can observe good results for the dynamic model, obtaining experimental execution time very close to the optimum ones.

Table 34. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with $t_w = 3.0\mu$s and with 65% CPU availability in one processor; and with $t_w = 0.7\mu$s and with 100% CPU availability in three processors.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.30	0.96	0.78	1.02	0.94	31%	21%
1024	1.47	4.13	3.57	4.60	3.57	29%	0%
1536	3.86	10.13	10.08	10.08	10.08	0%	0%
2048	7.85	19.33	22.40	22.40	22.67	0%	1%
2560	13.81	32.25	37.74	37.93	37.74	1%	0%
3072	21.90	49.39	61.71	61.71	61.71	0%	0%

Table 35. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with $t_w = 2.5\mu$s and with 40% CPU availability in one processor; and with $t_w = 0.7\mu$s and with 100% CPU availability in three processors.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.30	0.87	0.27	0.43	0.46	59%	70%
1024	1.47	4.08	5.79	6.52	5.79	13%	0%
1536	3.86	10.40	13.43	13.43	13.70	0%	2%
2048	7.85	20.65	22.31	26.51	22.31	19%	0%
2560	13.81	35.64	38.70	45.19	38.70	17%	0%
3072	21.90	56.18	63.25	63.25	63.25	0%	0%

Table 36. Comparison of the optimum time predicted with a static model (SM-the), the optimum time predicted with a dynamic model (DM-the), the optimum experimental time (opt-exp), the experimental time with the parameters provided by the static model (SM-exp) and the experimental time with the parameters provided by the dynamic model (DM-exp), with $t_w = 3.5\mu$s and with 30% CPU availability in one processor; and with $t_w = 0.7\mu$s and with 100% CPU availability in three processors.

n	SM-the	DM-the	opt-exp	SM-exp	DM-exp	dev. SM-exp	dev. DM-exp
512	0.30	1.15	0.96	2.50	1.81	160%	89%
1024	1.47	5.27	8.02	9.46	9.46	18%	0%
1536	3.86	13.25	19.04	20.52	20.52	8%	0%
2048	7.85	26.01	31.91	37.59	37.59	18%	0%
2560	13.81	44.48	54.14	58.33	58.33	8%	0%
3072	21.90	69.59	58.80	58.80	58.80	0%	0%

4 Future work

In this Section we summarize some possible continuations of the research in different aspects of the automatic optimization of parallel linear algebra software.

4.1 Use of multiple libraries

To use multiple basic libraries in a program it is necessary to develop a technique to rename the libraries and the routines in them. It is also necessary to develop for each basic routine another routine that decides which library to use in a specific call, depending on the problem size and the execution time (theoretical or experimental) obtained at installation time.

It is also possible to develop some techniques to build poly-libraries, in a similar way to that used for poly-algorithms [18]. It would be necessary to determine, for each routine in the library, the significant problem size and data access schemes, to perform experiments when the library is being installed, and to store the information generated. The names of the libraries and routines must be changed, and a poly-library built with a routine for each one of the routines in the libraries. This new routine will be called by the programs, and it must decide which basic library to use for a particular problem size and data access scheme. When a library is included in, or eliminated from the poly-library, each routine in the poly-library is modified.

In a heterogeneous system the library selected for each processor may be different, and a tool to run different programs in the different processors is needed.

We have only considered computational libraries, but it is also possible to use several communication libraries. In fact, different communication networks could also be used (as happends in TORC with Fast-Ethernet and Myrinet) and the best network to use will depend on the size of communications and the workload in the network.

4.2 Toeplitz problems

We have considered only one problem with Toeplitz matrices, but it is interesting to consider other 3D Sound Systems problems. If the problem considered is used in a real 3D Sound System it must be optimized for the particular computational system where it is going to be used in order to have real time response.

4.3 LU factorisation

The use of the proposed methodology is viable in systems where the load is stable or variable. In the case of variable load, the use of a software like NWS is appropriate for adjusting the system parameter values initially obtained at installation time, and thus obtain a model better suited to the circumstances of the system at execution time. How the system load at execution time affects the system parameters has been reflected by a linear approach. Future work will include an in-depth study of a possible non-linear approach.

The heterogeneous load case offers many more possibilities in addition to those studied. It could be interesting to consider the ideas to develop heterogeneous algorithms in [4, 5, 13].

4.4 Distributed systems

For very large problems the solution in large distributed systems may be of interest. Experiments with the LU factorisation in systems of this type could be worthwhile, depending on the results obtained in heterogeneous and variable load systems.

References

1. Pedro Alonso, José M. Badía, Antonio M. Vidal. A Parallel Algorithm for Solving the Toeplitz Least Squares Problem. VECPAR2000, Faculdade de Engenharia da Universidade do Porto, Portugal. June, 21–23, 2000.
2. F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. Rice University, Houston, Texas, 2001.
3. L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. C. Whaley *ScaLAPACK Users' Guide*. (SIAM, 1997).
4. Pierre Boulet, Jack Dongarra, Fabrice Rastello, Yves Robert, Frédéric Vivien. Algorithmic issues on heterogeneous computing platforms. *Parallel Processing Letters*, 9(2):197–213, 1999.
5. Vincent Boudet, Antoine Petitet, Fabrice Rastello, Yves Robert. Data Allocation Strategies for Dense Linear Algebra Kernels on Heterogeneous Two-dimensional Grids. In *IASTED Parallel and Distributed Computing and Systems*, 1999.
6. J. Cuenca, D. Giménez, J. González. Modeling the Behaviour of Linear Algebra Algorithms with Message-Passing. In Proceedings of the Ninth Euromicro Workshop on Parallel and Distributed Processing, pp 282–289. Mantova, Italy, 7–9 February 2001.
7. J. Cuenca, D. Giménez, J. González. Towards the Design of an Automatically Tuned Linear Algebra Library. In Proceedings of the Tenth Euromicro Workshop on Parallel, Distributed and Network-based Processing. Gran Canaria Island, Spain, 9–11 January , 2002.
8. M. Frigo. FFTW: An Adaptive Software Architecture for the FFT. In Proceedings of the ICASSP Conference, volume 3, p 1381. 1998.
9. D. Giménez, G. Carrillo. Installation routines for linear algebra libraries on LANs. VECPAR2000, Faculdade de Engenharia da Universidade do Porto, Portugal. June, 21–23, 2000.
10. D. Giménez, J. Cuenca, J. González. Automatic parameterisation of parallel linear algebra routines. In Algèbre Linéaire et Arithmétique: Calcul Numérique, Symbolique et Paralleèle, pp 63–81. Rabat, Morocco, May 2001.

11. G. Golub, C. Van Loan. *Matrix Computations.* John Hopkins Press, 2nd edition, 1989.
12. Greg Henry's Web page about BLAS. `http://www.cs.utk.edu/~ghenry/distrib/index.htm`
13. A. Kalinov, A. Lastovetsky. Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers. In P. Sloot, M. Bubak, A. Hoekstra and B. Hertzberger, editor, *High Performance Computing and Networking*, number 1593 in Lecture Notes in Computer Science, pages 191–200. Springer, 1999.
14. V. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to Parallel Computing. Design and Analysis of Algorithms* (The Benjamin Cummings Publishing Company, 1994).
15. Jakob Østergaard. OptimQR, A Software-package to create near-optimal solvers for sparse systems of linear equations. `http://www.ostenfeld.dk/~jakob/OptimQR/`
16. Haesun Park, Lars Eldén. Schur-type methods for solving least squares problems with Toeplitz structure. *SIAM J. Sci. Comput.*, 22(2):406–430, 2000.
17. A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, S. Vadhiyar. Numerical Libraries And The Grid. Computer Science Department, University of Tennessee, ut-cs-01-460, 2001.
18. A. Skjellum, A. P. Leung, C. H. Still, S. G. Smith, R. D. Falgout, C. H. Baldwin. The Multicomputer Toolbox – First-Generation Scalable Libraries. In Proceedings of HICSS–27, IEEE Computer Society Press, pp. 644–654. HICSS–27 Minitrack on Tools and Languages for Transportable Parallel Applications. 1994.
19. Sathish S. Vadhiyar, Graham E. Fagg, Jack Dongarra. Automatically Tuned Collective Communications. In Proceedings of the SC'2000 Conference, Dallas, Tx, November 2000.
20. Antonio M. Vidal. Memory of the subproject Highperformance Computation for 3D Sound Systems, in the project Development and Implementation of Signal Processing and Highperformance Computation Algorithms for Virtual Acoustic Environment Emulation. CICYT project TIC2000/1683-C03, 2001–2003.
21. R. Clint Whaley, Antoine Petitet, Jack J. Dongarra. Automated Empirical Optimisation of Software and the ATLAS Project. Computer Science Department, University of Tennessee, ut-cs-00-448. September, 2000.
22. Automatically Tuned Linear Algebra Software (ATLAS) Web page. `http://netlib.org/atlas/index.html`
23. Message Passing Interface Forum Web page. `http://www.mpi-forum.org`
24. Network Weather Service Web page, `http://nws.npaci.edu/NWS`
25. TORC Web page, `http://icl.cs.utk.edu/projets/torc`