Automatic Optimisation of Parallel Linear Algebra Routines in Systems with Variable Load*

Javier Cuenca[1], Domingo Giménez[2], José González[1], Jack Dongarra[3], and Kenneth Roche[3]

[1] Departamento de Ingeniería y Tecnología de Computadores
{javiercm, joseg}@ditec.um.es
[2] Departamento de Informática y Sistemas Informáticos
domingo@dif.um.es
[1,2] Universidad de Murcia
30071 Murcia. Spain
[3] Department of Computer Science
{dongarra, roche}@cs.utk.edu
University of Tennessee
Knoxville, TN 37996, USA

Abstract. In this work an architecture of an automatically tuned linear algebra library proposed in previous works is extended in order to adapt it to platforms where both the CPU load and the network traffic vary. During the installation process in a system, the linear algebra routines will be tuned automatically to the system conditions: hardware characteristics and basic libraries used in the linear algebra routines. At run-time the parameters that define the system characteristics are adjusted to the actual load of the platform. The design methodology is analysed with a block LU factorisation. Variants for a sequential and parallel version of this routine on a logical rectangular mesh of processors are considered. The behaviour of the algorithm is studied with message-passing, using MPI on a cluster of PCs. The experiments show that the configurable parameters of the linear algebra routines can be adjusted during the run-time process despite the variability of the environment.

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) [19] and relies on 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 itself 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 only after a large amount of work by expert programmers. Attempts have been made to develop this type of software in different fields: FFT [9], sparse systems [16] and dense linear algebra [19].

* Partially supported by Comision Interministerial de Ciencia y Tecnología, project TIC2000/1683-C03.
Partially supported by Fundación Séneca, reference number EE 00558 /CV/01 and PI-34/00788/F5/01.
Partially developed using the resources of the ICL, University of Tennessee.

Furthermore, the development of automatically tuned software would help to facilitate the efficient utilisation of the routines by non-expert users, e.g. those normally using linear algebra routines in the solution of large scientific or engineering problems in supercomputers. This has prompted the development of techniques to facilitate the efficient use of this type of routines on homogeneous [6] or distributed [1] systems. Research towards this direction is clearly related to that in automatic tuning, because some of the techniques used to develop automatically tuned software can be used to obtain near optimal executions.

We are investigating the development of dense linear algebra software for message-passing systems. Our approach to tackle the problem has been to identify algorithmic and system parameters and to analyse the algorithms both theoretically and experimentally in order to determine the influence of the value of the system parameters in the algorithmic parameters. In that way, installation routines have been 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 analysed to date, typical system parameters are the cost of arithmetic operations of level 1, 2 or 3 (k_1, k_2, k_3) and communication parameters (start-up, t_s and word-sending time, t_w). The algorithmic parameters are the block size b (in block based algorithms) and parameters defining the logical topology of the process grid or the data distribution. The results are satisfactory with different methods (Jacobi methods for the symmetric eigenvalue problem, LU factorisation and Gauss elimination) and different systems such as distributed and shared memory multiprocessors and clusters of workstations [11].

Since our methodology estimates the parameters' values obtained during installation, it is likely that the system state (CPU load, network traffic) at the moment the routines are to be used will be quite different than at installation time. This may lead to the use of inaccurate parameters and then to execution times far from the optimum. Therefore, the aim of this work 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 installation. This enhanced model constitutes 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 [1], [17]). 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 overheads incurred would be low when using a tool like the Network Weather Service (NWS) [15], and the method could be suitable for both large and small problems.

NWS is a tool (software) that provides measurements and predictions of the particular features of a system at a given time. The current implementation of NWS supports measuring the fraction of CPU available for new processes 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 a GRID environment. In the former, the overheads introduced are almost negligible [17].

The rest of the paper is organised as follows: in section 2 the proposed architecture of an automatically tuned linear algebra library is analysed , in sections 3 and 4 the methodology is applied to sequential and parallel versions of the block LU factorisation, and in section 5 the conclusions are summarised and some possible future research is outlined.

2 Methodology for the design of automatically tuned linear algebra libraries

In this section the architecture of an automatically tuned linear algebra library is analysed. There are three main steps in the construction and use of the library: design process, installation process and run-time process (figure 1). The first two steps were introduced in our previous works [6], whereas the third is the new contribution of this study. The elements used are the following:
LAR: Linear Algebra Routine of the library we want to build.
MODEL: Analytical Model of the execution time for the LAR as a function of the problem size (*n*), the system parameters (SP) and the algorithmic parameters (AP).
SP-Estimators: Estimation Routines of the SP values.
Basic Libraries:
Basic Communication Library: MPI [14], PVM [10], etc.
Basic Linear Algebra Library: reference BLAS [8], machine-specific BLAS, ATLAS [19], etc.
Installation-File: The SP values are obtained using the information (*n* and AP values) of this file.
Static-SP-File: This file contains the estimated values of the SP at installation time, when the SP-Estimators are executed.
Current-SP: Tuned values of the SP at run time.
Optimum-AP: From the MODEL, and with the run time SP values already known, the optimum AP values are obtained.

2.1 Design process

This process is hand-made only once by the LAR-designer. The LAR-designer is in charge of modelling the LAR, obtaining the MODEL:

$$T_{exec} = f(SP, AP, n) \tag{1}$$

In previous works [7], the SP values have been considered as system constants, however we proved that they actually depend not only on the system characteristics but also on the problem size *n* and the AP values:

$$SP = f(AP, n) \tag{2}$$

The LAR-designer also creates the different SP-Estimators. Each SP-Estimator is basically formed by the LAR kernel which constitutes the dominant performance cost regarding each one of the SP. The LAR-designer also decides which aspects are considered relevant for determining the SP values (for example, the data access scheme). When a complete library is being designed each LAR could have a set of SP-Estimators associated with it, but different routines could have common basic kernels, and it may be better to develop an installation process common to several LARs in the library. So far, we have only considered the design of individual installation routines.

2.2 Installation process

The most significant values, *n* and AP, needed to estimate the SP values are written in the Installation-File. Next, the appropriate experiments are performed to obtain the SP values. This means executing the SP-Estimators and generating the Static-SP-File. This file will, therefore, contain the SP values obtained at installation time ($t_{s\text{-}static}$, $t_{w\text{-}static}$, $k_{3\text{-}static}$, $k_{2\text{-}static}$, $k_{1\text{-}static}$) for the *n* and AP values specified in the Installation-File.

This process must be done when the system has a minimum load (close to 100 % of available CPU in all the nodes and minimum traffic in the interconnection network) in order to obtain SP values that reflect the static characteristics of the system.

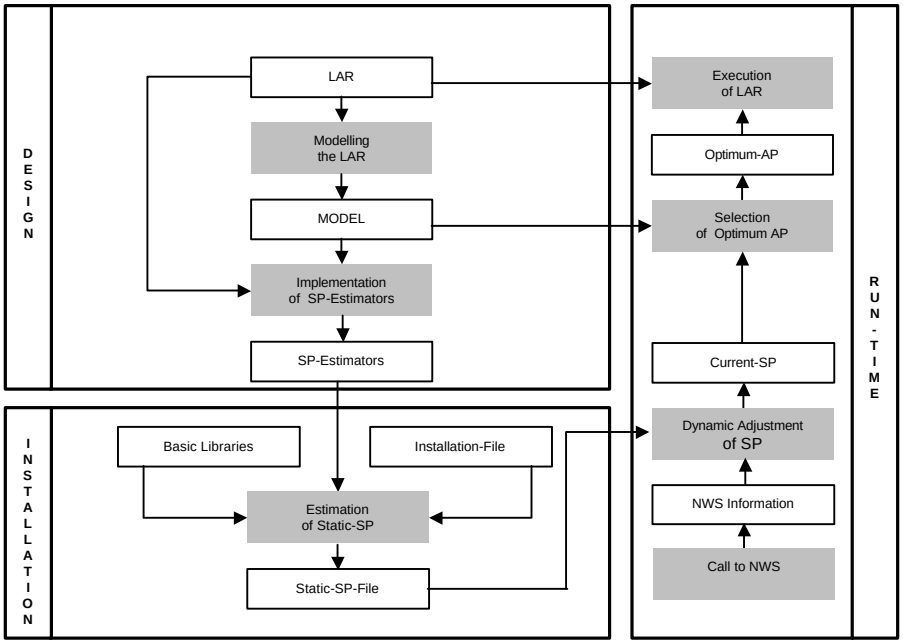

Fig. 1. Methodology for the design of Automatically Tuned Linear Algebra Libraries

2.3 Run-time process

When the user calls the LAR for a specific problem size *n* the following steps are carried out:

1.- The NWS is called and it reports:

- the fraction of available CPU (f_{CPU}) .
- the current word sending time ($t_{w\text{-}current}$) for a specific *n* and AP values (n_0, AP_0).

Then the fraction of available network is calculated:

$$f_{network} = \frac{t_{w-static}(AP_0, n_0)}{t_{w-current}(AP_0, n_0)} \tag{3}$$

2.- The values of the SP are adjusted, according to the current situation:

$$k_{3-current}(AP,n) = \frac{k_{3-static}(AP,n)}{f_{CPU}} \quad (4)$$

$$t_{w-current}(AP,n) = \frac{t_{w-static}(AP,n)}{f_{network}} \quad (5)$$

3.- Next, the optimum AP are automatically calculated by taking the dynamically updated, current SP values and the MODEL.

4.-The LAR automatically obtains the AP values to be used in this execution by taking the values associated with problem sizes close to *n* from the optimum, calculated AP.

The resulting dynamic MODEL used is the union of the basic static MODEL (formulas 1 and 2) and the dynamic adjustment of the system parameters (formulas 3, 4 and 5).

3 Sequential block LU factorisation

In this section, the way in which the values of the parameters of the MODEL are affected by the load of the CPU is analysed. The LAR used is a sequential block LU factorisation following the scheme in [12]. In this case, only the block size (*b*) is considered as an AP, and the costs of the arithmetical (floating point) operations at the different levels (k_1, k_2 and k_3) are the SP. The theoretical cost of the routine, which constitutes the MODEL, is:

$$T_{exec} = \frac{2}{3}k_3 n^3 + bk_3 n^2 \quad (6)$$

The platform considered has been a Pentium III node from the TORC system [18] at the Innovative Computing Laboratory, The 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 this 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. The basic linear algebra library used has been ATLAS [19].

The term n^3 is obtained from matrix-matrix multiplications of dimensions $c_i \times b$ by $b \times c_i$, with *b* the block size and the c_i having values $n - b$, $n - 2b$, ..., b. Thus, the SP-Estimator used for k_3 is a matrix-matrix multiplication for different values of *n* and *b*. Experiments performed on different platforms, with different basic libraries and different values of *n* and *b* show the value of k_3 depends mainly on *b* and not on *n* [6]. Table 1 shows the values of k_3 (in microseconds) for different values of *b*.

Table 1. Estimation of static SP ($k_{3\text{-}static}$) for different block sizes (in μsec)

Block size	**16**	**32**	**64**	**128**
$k_{3\text{-}static}$	0.0038	0.0033	0.0030	0.0027

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 2. Values of the optimum AP (block size) for different problem sizes and CPU loads

	available CPU			
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

Table 2 shows the theoretical optimum *b* for different loads of the CPU, according to the dynamic MODEL proposed.

Table 3 shows the basic case, when the LAR is executed with the same CPU load as when the routine was installed, that is, when the AP values are taken from the first column of table 2 (100 % CPU availability). In this situation, the static MODEL produces a good theoretical estimation of the execution times (SM_the). An accurate choice of the AP is carried out, and experimental execution times (SM_exp) are very close to the optimum ones (opt_exp).

Table 3. 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

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 4. 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), for different values of CPU availability. The results presented were obtained on a single node of the cluster

70% CPU availability							
n	**SM_the**	**DM_the**	**opt_exp**	**SM_exp**	**DM_exp**	**dev SM_exp**	**dev DM_exp**
512	0.36	0.49	0.42	0.58	0.58	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%
30% CPU availability							
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%

Table 4 shows that, when the CPU load increases, the static MODEL produces unrealistic theoretical estimations of the execution time (SM_the), which causes the wrong choice of the

AP values. This leads to an experimental execution time (SM_exp) far from the optimum (exp_opt). On the other hand, the use of the dynamic model leads to more accurate theoretical estimations (DM_the) and a better choice of AP values is made. Thus, experimental execution times (DM_exp) are close to the optimum. The differences in the deviations of SM_exp and DM_exp with respect to the opt_exp have been highlighted. In general, it is more difficult to obtain accurate predictions for small problem sizes when the system load increases because they are more sensitive to the variations in the system load.

In order to carry out all these experiments, for each value of the CPU availability and for each *n*, the LAR has been executed for representative AP values. The opt_exp is the best time, the SM_exp is the time obtained with the AP values of the first column of table 2 and the DM_exp is the time obtained with the AP values of the corresponding column of table 2.

4 Parallel block LU factorisation

In this section, the parallel LAR used is a parallel block LU factorisation. The theoretical arithmetic and communication execution times, which constitute the static MODEL, are:

$$T_{ari} = \frac{2}{3}k_3\frac{n^3}{p} + \frac{r+c}{p}bk_3n^2 + \frac{1}{3}b^2k_2n$$

$$T_{com} = t_s\frac{2nd}{b} + t_w\frac{2n^2d}{p} \qquad (7)$$

where the AP to be estimated are the block size (*b*), the number of processors to be used (*p)* and the dimensions of the logical topology used: a 2D-mesh ($p=r\times c$ and $d=\max(r,c)$). Matrices are distributed in a 2d, block-cyclic fashion (ScaLAPACK style [2]).

4.1 Variable network traffic

This subsection studies how the traffic in the interconnection network affects the parameter values. The arithmetic SP are those obtained in the sequential case, and the communication SP are obtained using communication SP-Estimators with the same communications scheme used in the LAR.

Table 5. Values (in μsec) of t_w, at installation time, for the parallel routine block based LU, for different message sizes. Runs were conducted on4 Pentium III nodes with Fast-Ethernet

Message size (bytes)	**32768**	**262144**	**1048576**	**2097152**
t_{w_static}	0.7000	0.6900	0.6800	0.6750

Experiments have been carried out on four Pentium III nodes of the TORC system, using only a processor per node and a Fast-Ethernet as interconnection network. Table 5 shows the values of t_w (in microseconds) for different message sizes at installation time.

The NWS skill *tcpMessageMonitor* has been used in order to measure the network traffic at execution time. This skill monitors the TCP bandwidth and latency between each pair of a set of machines. 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 6, the theoretical optimum *b* is shown according to the proposed dynamic MODEL for different network traffics, i.e., for different word-sending times.

Table 6. Values of the optimum AP (block size) for different problem sizes and network traffic. Runs were conducted on 4 Pentium III nodes with Fast-Ethernet

	$t_{w\text{-}current}$			
n	**0.7 µsec**	**1.5 µsec**	**4.0 µsec**	**7.0 µsec**
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 7. 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)t_w = 0.7 µsec. Runs were conducted on 4 Pentium III nodes with Fast-Ethernet

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 8. 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 different values of t_w. Runs were conducted on 4 Pentium III nodes with Fast-Ethernet

$t_{w\text{-}current}$ **=4.0 µsec**							
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%
$t_{w\text{-}current}$ **=7.0 µsec**							
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%**

Table 7 shows the basic case, when the execution of the routine is run with similar network traffic to when the routine was installed, i. e., with the AP values of the first column of table 6. In this situation, the static MODEL produces a good theoretical estimation of the execution times (SM_the). An accurate choice of the AP values is made and experimental execution times (SM_exp) are close to the optimum (opt_exp).

As in the sequential case, in Table 8 we can observe that, when the network traffic increases, the static MODEL produces incorrect theoretical estimations of the execution time (SM_the), which causes a wrong choice of the AP values. Thus, an experimental execution time (SM_exp) which is far from the optimum (exp_opt) is observed. On the other hand, with the dynamic

MODEL, the theoretical estimations (DM_the) are more accurate, which leads to a better choice of the AP values.

So far, the viability of the dynamic MODEL has been shown, separately, for variations of CPU availability (previous section) and for variations in the traffic of the interconnection network (this subsection). In the next subsection a combination of both these ideas is shown.

4.2 Variable network traffic and CPU availability

Experiments have been carried out on four and eight Pentium III nodes, using only one processor per node, of the TORC system. The interconnection network used has been 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.

Table 9. Values of the optimum AP (block size) for different problem sizes and platform loads. Runs were conducted on 4 Pentium III nodes with Fast-Ethernet

	% available CPU / $t_{w\text{- current}}$		
n	**100% 0.7µs**	**70% 1.5µs**	**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

Table 10. Values of the optimum AP (block size) for different problem sizes and platform loads. Runs were conducted on 4 Pentium III nodes with Fast-Ethernet

	% available CPU / $t_{w\text{- current}}$		
n	**100% 0.7µs**	**70% 2.0µs**	**60% 5.5 µs**
1024	32	64	64
2048	64	64	128
3072	64	128	128
4096	128	128	128

The theoretical optimum *b* is shown in table 9 for 4 nodes, and in table 10 for 8 nodes. The results follow from the dynamic MODEL proposed for different loads of the parallel platform (different CPU loads and network traffic).

Table 11. 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 µsec and available CPU =100%. Runs were conducted on 8 Pentium III nodes with Fast-Ethernet

n	**SM_the**	**opt_exp**	**SM_exp**	**dev SM_exp**
1024	1.10	0.93	0.99	6%
2048	5.41	4.98	4.98	0%
3072	14.38	13.81	13.81	0%
4096	29.43	27.65	29.31	6%

Table 12. 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 different values of CPU availability and t_w. Runs were conducted on 4 Pentium III nodes with Fast-Ethernet

70% of CPU availability $t_{w\text{-}current}$ =1.5 µsec							
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%
35% of CPU availability $t_{w\text{-}current}$ =7.0 µsec							
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 13. 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 different values of CPU availability and t_w. Runs were conducted on 8 Pentium III nodes with Fast-Ethernet

70% of CPU availability $t_{w\text{-}current}$ =2.0 µsec							
n	**SM_the**	**DM_the**	**opt_exp**	**SM_exp**	**DM_exp**	**dev SM_exp**	**dev DM_exp**
1024	1.10	2.62	3.03	3.10	3.03	**2%**	**0%**
2048	5.41	11.85	12.31	13.04	13.04	6%	6%
3072	14.38	29.56	29.92	29.92	30.63	**0%**	**2%**
4096	29.43	57.34	60.01	60.01	60.01	0%	0%
60% of CPU availability $t_{w\text{-}current}$ =5.5 µsec							
1024	1.10	6.32	9.36	10.17	9.99	**9%**	**7%**
2048	5.41	26.73	25.30	25.34	25.30	0%	0%
3072	14.38	63.11	56.42	58.07	56.42	**3%**	**0%**
4096	29.43	117.48	108.34	112.12	112.12	3%	3%

In table 7, for 4 nodes, and in table 11, for 8 nodes, the basic cases are shown with the minimum load in the platform. A good choice of AP values is made with the static MODEL. The experimental execution times (SM_exp) are close to the optimum (opt_exp).

When the platform load increases, the static MODEL gives worse results, as can be seen in table 12 for 4 nodes, and in the table 13 for 8 nodes. As in the two previous studies, when the load increases, with the dynamic MODEL the theoretical estimations (DM_the) improve with respect to the static MODEL. A better choice of AP values is made and 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. Now, it would be convenient to study cases of heterogeneous load, where some of the system nodes have more load than others at the moment of the execution. An introduction to this study is given in the next subsection.

4.3 Variable and heterogeneous system load

This section looks at the situation when the platform load is not distributed homogeneously, rather the more common case where there are some nodes with heavier loads than others. In this situation, and with a routine with a homogeneous distribution of the work (like the parallel block LU), the execution rate is set by the processors with the worst calculation and communication features. The values of the system load (CPU availability and word sending time) will correspond to the processor with the largest load in order to apply the dynamic adjustment of the model at execution time.

Table 14. 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).One of the nodes has different values for t_w and CPU availability . The rest of the system has t_w = 0.7 μsec and CPU availability =100%

One node: **65% of CPU availability** $t_{w\text{-}current}$ **= 3.0 μsec**							
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%
One node: **30% of CPU availability** $t_{w\text{-}current}$ **= 3.5 μsec**							
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%

In Table 14 the results for 4 nodes are shown (the basic case of the static situation is shown in Table 7). Only one of the nodes has been overloaded by executing several images of an application which is independent of the LU routine. We can observe promising results for the dynamic MODEL, with near optimum experimental execution times.

5 Conclusions and future work

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 software like NWS is suitable for the adjustment of the system parameters' values obtained at installation time. The obtained model is better suited to the state 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 a deeper study of a possible non-linear approach that produces a better adjustment. The heterogeneous load case offers many more possibilities than the one studied. It would be interesting to continue along these lines, for example, considering ideas to develop heterogeneous algorithms [3, 4, 13].

References

[1] 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, and R. Wolski, "The GrADS Project: Software for High-Level Grid Application Development", Rice University, Houston, Texas, 2001.

[2] L. S. Blackford, J. Choi, A. Clearly, E. D'Azavedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, "ScaLAPACK Users' Guide", Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[3] P. Boulet, J. Dongarra, F. Rastello, Y. Robert, F. Vivien, "Algorithmic issues on heterogeneous computing platforms", Parallel Processing Letters, 9(2):197-213, 1999.

[4] V. Boudet, A. Petitet, F. Rastello, Y. Robert, "Data Allocation Strategies for Dense Linear Algebra Kernels on Heterogeneous Two-dimensional Grids",IASTED Parallel and Distributed Computing and Systems, 1999.

[5] J. Cuenca, D. Giménez, and J. González, "Modeling the Behaviour of Linear Algebra Algorithms with Message-Passing", proceedings of the Euromicro Workshop on Parallel and Distributed Processing, Mantova, Italy, February, 2001, pp. 282-289.

[6] J. Cuenca, D. Giménez, and J. González, "Towards the Design of an Automatically tuned Linear Algebra Library", proceedings of the Euromicro Workshop on Parallel and Distributed Processing, Gran Canaria Island, Spain, January, 2002.

[7] K. Dackland, and B. Kågström, "An Hierarchical Approach for Performance Analysis of ScaLAPACK-based Routines Using the Distributed Linear Algebra Machine", in Wasniewski et. al., editor, proceedings of Workshop on Applied Parallel Computing in Industrial Computation and Optimization (PARA96), Lecture Notes in Computer Science, Springer Verlag, Lyngby, Denmark, 1996, pp. 187-195.

[8] J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling, "A set of Level 3 Basic Linear Algebra Subprogram", ACM Trans. Math. Soft, 14, 1988, pp. 1-17.

[9] M. Frigo. "FFTW: An Adaptative Software Architecture for the FFT", proceedings of the ICASSP Conference, volume 3, 1998.

[10] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, Massachusetts, 1994.

[11] D. Giménez, J. Cuenca, and J. González, "Automatic parameterisation of parallel lineal algebra routines", Algèbre Linéaire et Arithmétique: Calcul Numérique, Symbolique et Parallèle, Rabat, Morocco, May, 2001, pp. 63-81.

[12] G. Golub, and C. Van Loan, Matrix Computations, John Hopkins Press, 2nd edition, 1989.

[13] A.Kalinov, A.Lastovetsky, "Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers", Journal of Parallel and Distributed Computing, 61, 4, 2001, pp.520-535.

[14] Message Passing Interface Forum. Web page: www.mpi-forum.org

[15] Network Weather Service. Web page: nws.npaci.edu/NWS.

[16] J. Ostergaard, "OptimQR, A Software-package to create near-optimal solvers for sparse systems of linear equations". Web page: www.ostenfeld.dk/~jakob/OptimQR/

[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] TORC. Web page: icl.cs.utk.edu/projects/torc

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