Designing Polylibraries to Speed Up Parallel Computations*

Pedro Alberti[1], Pedro Alonso[1], Antonio Vidal[1], Javier Cuenca[2], Domingo Giménez[3]

***Abstract*— In this paper we analyze the design of polylibraries, where the programs call routines from different libraries according to the characteristics of the problem and of the system used to solve it. An architecture for this type of libraries is proposed. Our aim is to develop a methodology which can be used in the design of parallel libraries. To evaluate the viability of the proposed method, the typical linear algebra libraries hierarchy has been considered. Experiments have been performed in different systems and with linear algebra routines from different levels of the hierarchy. The results confirm the design of polylibraries as a good technique for speeding up computations.**

Keywords—**Linear Algebra, Automatic Tuning.**

I. INTRODUCTION

Linear algebra functions are widely used in the solution of scientific and engineering problems, which means that there is much interest in the development of highly efficient linear algebra libraries, and these libraries are used by non-experts in parallel programming. In recent years several groups have been working on the design of highly efficient parallel linear algebra libraries. These groups work in different ways: optimizing the library in the installation process for shared memory machines [1] or for message-passing systems [2,3], analyzing the adaptation of the routines to the conditions of the system at a particular moment [4,5,6], or using polyalgorithms [7,8]. Polylibraries can be used in combination with other techniques to speed up linear algebra computations, in such a way that a user will not need to decide the best library to use, and the polylibrary is responsible for obtaining a near optimum execution.

In Section 2, an architecture for a polylibrary is proposed, and the ensuing variations in the library hierarchy are considered. Section 3 shows some of the experimental results obtained. Finally, in Section 4, some conclusions are given.

A. *The design of polylibreries*

When solving linear algebra problems, basic linear algebra routines are called to solved different parts of the problem. Typically some version of BLAS [9] or LAPACK [10] is used in sequential or shared memory systems, and ScaLAPACK [11] can be used in message-passing systems. But there may be different versions of the libraries. It is possible to have libraries optimized for the machine (either provided by the vendor or freely available [12,13]), or automatic tuning libraries like ATLAS [1] can be installed, or again alternative libraries can be used [14].

The hierarchy of linear algebra libraries is shown in figure 1. At the lowest level of the hierarchy is the basic linear algebra library (BLAS), which may be the reference BLAS, a BLAS optimized for the machine, or an automatic tuning library. To perform the communications in message passing programs a library optimized for the machine can be used, as could a free version of MPI [15] or PVM [16]. At higher levels of the hierarchy there are also different libraries with the same or similar functionality.

For example, it is possible to solve an eigenvalue problem using ScaLAPACK, LAPACK, ATLAS and machine MPI; alternatively, it could be solved using PLAPACK, machine BLAS and MPICH. Likewise, the same problem, with different sizes, may require distinct libraries.

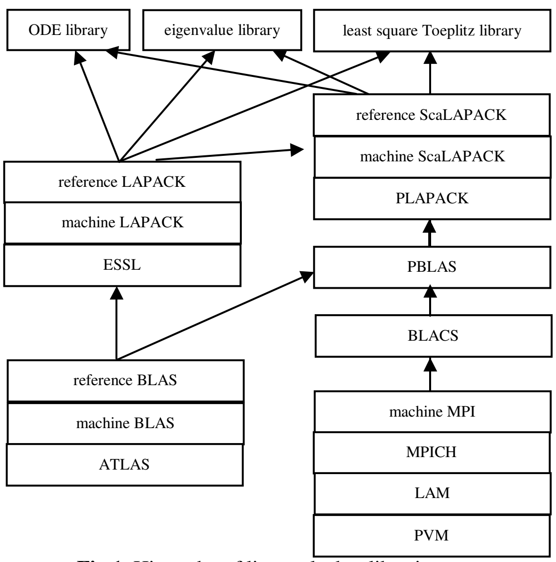

Fig 1. Hierarchy of linear algebra libraries

* Partially supported by Comision Interministerial de Ciencia y Tecnología, project TIC2000/1683-C03. Partially supported by Fundación Séneca, reference number PI-34/00788/FS/01. This work has been developed using the systems of the CEPBA European enter for Parallelism of Barcelona), the CESCA (Centre de Supercomputació de Catalunya) and the Innovative Computing Laboratory at the University of Tennessee.

[1] Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia

[2] Departamento de Ingeniería y Tecnología de Computadores. Universidad de Murcia.

[3] Departamento de Informática y Sistemas Informáticos.Universidad de Murcia.

B. The Advantage of Polylibreries

Traditionally the user decides to use a particular set of libraries in the solution of all the problems. There are some considerations which make it advisable to use different libraries for different problems. Automatic selection is therefore interesting for a number of reasons:

- A library optimized for the system might not be available. The use of an automatic tuning library would be satisfactory, although with time a better library may become available. Thus, it is necessary to have a common method to evaluate libraries or routines in the libraries, so that when a library is installed, the preferred library is changed.
- When the characteristics of the system change, the preferred library can also change. This makes it necessary to reinstall the libraries.
- Which library is the best may not be clear since library preference will vary according to the routines and the systems in question. This makes it necessary to develop the concept of polylibrary. The information generated in the installation of each library is used to construct the polylibrary.
- Even for different problem sizes or different data access schemes the preferred library can change. This makes it necessary to design polylibraries in such a way that a routine in the polylibrary calls to one library or another not only according to the routine, but also to other factors such as the problem size or the data access scheme.
- We may be working in a parallel system with the file system shared by processors of different types, which may or may not share the libraries. In the case of their sharing a basic library, this may be optimized for one type of processors and not for the others. In any case, the resulting library will be different for the different types of processors.

In this paper, we propose the architecture of a polylibrary, and show with experiments with some linear algebra routines how the use of polylibraries helps in the decision of the libraries to be used to reduce the execution time when solving a problem.

C. Architecture of a Polylibrary

A possible architecture for a polylibrary is shown in figure 2. For each of the basic libraries an installation file (LIF) will be generated, containing information about the performance of the different routines in the library for different variable factors. The format of the different LIFs must be the same, and the generation of the LIFs must be carried out in the polylibrary installation or updating process.

This generation can be made by using some model of the routines to predict the execution time or by performing some executions [17,2]. When the generation is made by performing some executions, the process must be guided in order to avoid an overlong installation [18]. For each routine in the polylibrary, an interface routine is developed.

The polylibrary will use the information generated in the lower level libraries installation (LIFs) to generate the code of the interface routine. To do this, the format of the different LIFs would be the same, and what we propose is to develop each library together with an installation engine which generates the LIF in the format required. This format must be decided by the linear algebra community in order to facilitate the use of the information to design polylibraries or libraries of higher level in the libraries hierarchy. The information in the LIF could be the cost of an arithmetic operation using some routines. In some cases this could be obtained for different problem sizes, because the use of different costs for different sizes can produce a better prediction and consequently a reduction in the execution time [19].

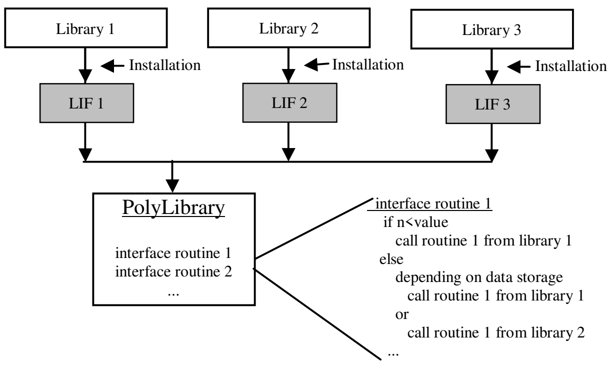

Fig 2. Architecture of a polylibrary

For some routines more than one parameter can be used, as for example in matrix-matrix multiplication, where it could be interesting to store the cost with square or rectangular matrices. Also the different access schemes can produce different costs, for example, the BLAS routine *drot* can have different costs when the data in the vectors are stored in adjacent positions of memory than when they are stored in non-adjacent positions. The format of the LIF could be as follows:

drot	*ld* →	1	100	200
	100			
n	200			
	400			

dgemm	*column* →	20	40	80
	20			
row	40			
	80			

The development of installation engines must be included in the basic libraries (BLAS, ATLAS, MPI, PVM,...), but also in libraries of higher level (LAPACK, PLAPACK, ScaLAPACK,...), which allows the development of higher level libraries, which could include an installation engine to generate their own LIFs.

D. Combining Polylibraries with other Optimisation Techniques

There are different techniques for obtaining optimized libraries, and better libraries can be obtained by combining different techniques.

In polyalgorithms we have different algorithms to solve the same problem. The algorithms can be implemented in different libraries, and the use of different algorithms does not differ from the use of different libraries. If one of the basic libraries used is an automatically tuned library a large amount of time may be necessary to install the library in the system [1,2], which means a large amount of time is necessary to update the polylibrary when the conditions on the system change. The same is true when installing or reinstalling the polylibrary, and in a parallel system multiple installations will be necessary (possibly in parallel). Thus the installation process must be guided by the system manager [3], which decides when it is convenient to reinstall the polylibrary or include a new library, or a new type of processor, and which factors and range of values to consider.

In some cases the value of some algorithmic parameters are obtained at execution time [19]. These values can be the block size in algorithms by blocks; the number of processors or some parameters determining the logical topology in a parallel system, ..This means different values of these parameters must be considered in the polylibrary installation.

To adapt the method to systems where the load varies dynamically some heuristic can be used [6]. The values obtained at installation time can be influenced by using some measures of the system load obtained at running time.

II. Experimental Results

Experiments have been carried out with routines of different levels in the hierarchy:

- In the lowest level of the hierarchy, the routines are the typical BLAS ones. These routines must be installed by performing selected computations, and the information is stored in the LIF of the library. The matrix-matrix multiplication has been used because it is a representative operation of the lowest level libraries, it is used in algorithms by blocks and is a good example to show the methodology.
- Matrix factorizations by blocks [21] have been analyzed (LU and QR) because they are basic operations at a second level of libraries, and use routines of a lower level, as for example the matrix-matrix multiplication.
- As examples of the highest level of libraries a lift-and-project algorithm to solve the inverse additive eigenvalue problem [22] and an algorithm to solve the Toeplitz least square problem [23,24] have been considered. In both cases they use routines of lower levels, and those routines are used in a blackbox way.

The experiments have been carried out over different systems because the goal is to obtain a methodology valid for a wide range of systems. The platforms used have been: a shared memory space system (Origin 2000 with 32 processors), and some networks of processors. Some of the experimental results obtained are reported in this section. More experimental results can be found in [25].

A. *Matrix-matrix Multiplication*

The matrix-matrix multiplication is an operation from the lowest level of the hierarchy, but higher level implementations have been considered. For polyalgorithms we use a direct multiplication, a multiplication by blocks, and a Strassen multiplication. The use of these algorithms allows us to combine the design of polylibraries with the automatic choice of algorithmic parameters: in the algorithm by blocks the block size is an algorithmic parameter, and in the Strassen multiplication the algorithmic parameter is the size of the matrix to stop the recurrence.

For routines in the lowest level of the hierarchy only the sequential case is analyzed, because these routines would be used by parallel routines which will call them in each processor.

In a network of six SUNs with two different types of processors (five SUN Ultra 1 and one SUN Ultra 5) the interface of the routine would have one part for each type of processor. The libraries used in the experiments have been: a freely available reference BLAS (BLASref), the scientific library of SUN (BLASmac), and three versions of ATLAS [26]: a version of ATLAS installed in SUN Ultra 1 (ATLAS1), and another two prebuilt versions of ATLAS 3.2.0 for Ultra 2 (ATLAS2) and Ultra 5 (ATLAS5). The same libraries are used by the SUN Ultra 1 and the SUN Ultra 5 because they share the file system.

The values in the LIF can be obtained by performing executions for the different algorithms, problem sizes and values of the parameters provided by the library manager. This may suppose a large amount of time, and heuristics can be used to reduce that time. For example, in the algorithm by blocks, experiments can be performed for different blocks for the smallest problem size. In the Strassen algorithm, for the smallest problem size, the problem is solved with base size half the problem size, then a quarter of the problem size, and so on until the execution time increases. The optimum base size is taken as the initial for the local search for the next problem size.

If the installation is made with matrix sizes 400, 800 and 1200, the interface routine could be:

```
if processor is SUN Ultra 5
 if problem-size<600
        solve using ATLAS5 and Strassen method with
        base size half of problem size
 else if problem-size<1000
        solve using ATLAS5 and block method with
        block size 400
 else
        solve using ATLAS5 and Strassen method with
        base size half of problem size
endif
else if processor is SUN Ultra 1
 if problem-size<600
        solve using ATLAS5 and direct method
 else if problem-size<1000
        solve using ATLAS5 and Strassen method with
        base size half of problem size
 else
        solve using ATLAS5 and direct method
 endif
endif
```

The lowest execution time (low.) is compared with the execution time with the library, method and parameter provided by the model (mod.), and that obtained using the direct method with ATLAS5. The comparison is shown in table 1. The execution time using the model is close to the lowest, and the same happens when the direct method and ATLAS5 are used. The conclusion could be that it is better to determine the best library (ATLAS5) and method (direct). Possibly this is a good option in some cases, but we observe that the optimum library is not always apparent. The preferred library should be the scientific library of SUN, but it is clearly surpassed by all the versions of ATLAS. In SUN Ultra 1, one would expect the preferred ATLAS to be that installed in the system by the manager (ATLAS1), but in fact the best is ATLAS5.

The selection of the library, method and parameters is satisfactory in all the systems considered, and this produces small differences between the lowest execution time and that obtained with the values provided by the polylibrary.

The use of different algorithms and libraries is not a great improvement with respect to the use of the best library, but the proposed method can also be used just to obtain or update which is the best library among those available.

The method can also be used to decide which algorithm to choose from a set of available algorithms, because each algorithm can be considered as the routine of a different library.

The selection of the parameters which produce lowest execution times can be combined with the selection of the library or algorithm.

TABLE I. COMPARISON OF THE LOWEST EXECUTION TIME, THE EXECUTION TIME OBTAINED USING THE MODEL, AND THE EXECUTION TIME USING THE DIRECT METHOD AND ATLAS5. IN SUN ULTRA 5.

		200	600	1000	1400	1600
low	time	0.038	1.063	4.678	12.532	20.031
	library	ATL5	ATL5	ATL5	ATL5	ATL5
	method	direct	direct	direct	direct	direct
	parameter			2	2	400
mod	time	0.041	1.112	4.678	12.575	26.569
	library	ATL5	ATL5	ATL5	ATL5	ATL5
	method	Strassen	Strassen	Strassen	Strassen	Strassen
	parameter	2	400	2	2	2
ATLAS5 direct	time	0.038	1.063	4.833	13.501	31.023

B. *Matrix factorizations*

Automatic tuning medium level routines can be developed by obtaining a theoretical model of the routine where the system parameters are identified, and the values of the system parameters are obtained from those stored in the LIFs of lower level libraries. After the substitution of the values some decisions can be taken, e. g. the best library and the optimum values of some algorithmic parameters. We show how the method works with the LU and QR factorizations. The cost of the parallel block LU factorization can be modelled by the formulas:

$$\frac{2}{3}k_3\frac{n^3}{p}+\frac{r+c}{p}bk_3n^2+\frac{1}{3}b^2k_2n \qquad (1)$$

for the arithmetic cost, and:

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

for the communication cost.

For the QR factorizations the costs are modelled by:

$$\frac{\frac{4}{3}n^3k_3}{p}+\frac{\frac{1}{4}n^2bk_3}{c}+\frac{n^2bk_2}{r}+\frac{\frac{1}{2}n^2bk_2}{r} \qquad (3)$$

and:

$$t_s\left[\frac{n}{b}\left((2+3b)\log r+2\log c\right)\right]+t_w\left[\frac{n^2}{2}\left(\frac{2(r-1)}{r^2}+\frac{\log r}{c}+\frac{\log r}{r}\right)+nb\log p\right] \qquad (4)$$

The matrix is distributed in a logical two-dimensional mesh (in ScaLAPACK style [11]) of $p=r\times c$ processors, with block size b and d=max(r,c). Typical system parameters are the cost of arithmetic operations using BLAS (or similar one) kernels of levels 1, 2 or 3 (k_1, k_2, k_3) and communication parameters (start-up, t_s, and word-sending time, t_w). Experiments have been carried out in a network of Pentium III with Myrinet and Fast-Ethernet. Five libraries have been used: 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.

In table 2 the experimental results with the LU factorization are summarized. The theoretical (the.) and lowest (low.) 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. The model provides good values for the parameters. The best results are obtained with BLASIII and BLASII, as predicted by the model. Also the optimum block size is well predicted. The difference between the libraries is less important in the parallel case than in the sequential case due to the cost of communications, which is common to all the libraries.

TABLE II. EXECUTION TIME (IN SECONDS) OF THE PARALLEL LU ROUTINE, WITH DIFFERENT BASIC LIBRARIES. IN A NETWORK OF 4 PENTIUM III WITH MYRINET.

	the.		**low.**		**mod.**	
	time	blo.	time	blo.	time	blo.
ATLAS						
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
BLAS -II						
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
BLAS –III						
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

In table 3 results obtained with the QR factorization are shown. The execution time with the parameters provided by the model (mod.) and the lowest execution time (low.) are shown together with their parameters. The times with the model are always close to the lowest times, because the model makes a good selection of the parameters: number of processors, dimensions of the mesh, block size and library.

TABLE III. COMPARISON OF THE EXECUTION TIME OF THE QR FACTORIZATION. IN A NETWORK OF 8 PENTIUM III WITH FAST-ETHERNET.

	size:	128	256	384	1024	2048	3072
mod.	time	0.025	0.087	0.178	1.92	9.90	28.00
	proc.	1×2	1×8	1×8	1×8	1×8	1×8
	block	size	8/16	8	8	16	32
	library	BLASII	BLASII	BLASII	BLASII	BLASII	BLASII
low.	time	0.025	0.086	0.176	1.92	9.40	25.11
	proc.	1×2	1×8	1x8	1×8	2×4	2×4
	block	size	16	8	8	32	32
	library	BLASII	BLASIII	BLASIII	BLASII	BLASII	BLASII

C. Experiments with a Toeplitz Least Square program

The problem to solve is the Least Square (LS) Problem: $\max_x \|Tx - b\|_2$, where $T \in R^{m\times n}$ is Toeplitz ($T_{ij} = t_{i-j}$; $i= 0,1,...,m - 1$; $j=0,1,...,n - 1$) and $b \in R^m$.

Experiments have been carried out in two networks of Pentiums with Myrinet. In a Pentium II network with Myrinet the execution time in the sequential case (seq.), the lowest experimental time (low.) and the execution time with the number of processors provided by the model (mod.), are those shown in table 4, along with the number of processors used and the speed-up. We can see the model does not provide 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 on average about 58%. For algorithms with a small cost it could be preferable to perform some selected executions at installation time to decide the values of the parameters, and to use the model only for bigger problem sizes [17].

The library preferred in the sequential case could also be considered as that preferred for the parallel routine. In that way ATLAS would be used in the Pentium III network. In table 5 the execution times are compared when using the different libraries. The lowest execution time is highlighted for each matrix and number of processors. We can see the best library is now BLASII, which is the second best in the sequential case. This may 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 may not be the one preferred for smaller sizes.

D. Lift-and-project Method for the Inverse Additive Eigenvalue Problem

A lift-and-project method for the solution of the inverse additive eigenvalue problem has been used as an example of high level algorithm of high cost. The order of the algorithm is $O(n^4)$, and this high order makes it easier to obtain optimizations than in the Toeplitz Least Square algorithm considered, which has an order $O(n^2)$.

The problem is: given a set of $L + 1$ real symmetric matrices $A_0, A_1, ..., A_L$, of sizes $n \times n$, and a set of real numbers $\lambda^*_1 \leq \lambda^*_2 \leq \lambda^*_m$ ($m \leq n$), obtain $d \in R^L$ and a permutation $\sigma = \sigma_1, ..., \sigma_m$, with $1 \leq \sigma_1 < ... < \sigma_m \leq n$, which minimizes the function:

$$F(d, \sigma) = \tfrac{1}{2} \Sigma^m_{i=1} (\lambda_{\sigma i} (d) - \lambda^*_i)^2$$

with $\lambda_i(d)$, $i=1,2,...,n$; the eigenvalues of matrix

$$A(d) = A_0\ \Sigma^L_{i=1}(d_iA_i).$$

TABLE IV. 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.	low.			mod.		
		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

TABLE V. 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

The lift-and-project method used to solve the Least Square problem has been that described in [22]. We analyze the different parts (TRACE, ADK, EIGEN, MATEIG, MATMAT and ZKA0A) to identify the lower level routines used in the solution of the problem, and the cost of the different parts of the algorithm. The theoretical model obtained is:

$$\begin{aligned} & iter\left(\frac{22}{3}k_{syev} + 2k_{3,gemm} + k_{3,diaggemm}\right)n^3 + \\ & iter\left(2k_{1,dot} + k_{1,scal} + k_{1,axpy}\right)n^2 L + 2k_{1,dot}n^2L^2 + k_{sum}nL^2 \end{aligned} \tag{5}$$

In the formula there appear five parameters of routines from the lowest level of the hierarchy (BLAS), and one from the middle level (LAPACK). Three of the parameters of BLAS correspond to operations of BLAS 1, and two are from BLAS 3, but of the same operation (*gemm*) with two different types of matrices.

Table 6 shows the time obtained for a 250 × 250 matrix and with $L = 25$. The total time and the time in each part of the algorithm appear in the table. The combinations of libraries used are: LAPACK and BLAS installed in the system and supposedly optimized for the machine (LaIn+BlIn), reference LAPACK and a freely available BLAS for Pentium III (LaRe+BlII), reference LAPACK and a freely available BLAS for Pentium II (LaRe+BlII), reference LAPACK and the installed BLAS (LaRe+BlIn), LAPACK and BLAS installed for the use of threads (LaInTh+BlInTh), reference LAPACK and a freely available BLAS for Pentium II using threads (LaRe+BlIITh), and reference LAPACK and the

BLAS installed which uses threads (LaRe+BlInTh). The lowest execution time when not using threads (lnt.) and when using threads (lwt.), and the lowest execution time (low.) are shown for each part of the algorithm and the total time. We can see the lowest time represents an important reduction with respect to the execution time obtained with the best combination (LaRe+BIII), which is not the one installed in the system. The versions that use threads are not better than those that do not.

TABLE VI. COMPARISON OF THE EXECUTION TIME (IN SECONDS) IN THE DIFFERENT PARTS OF THE ALGORITHM AND WITH DIFFERENT BASIC LIBRARIES. IN DUAL PENTIUM III. $N=M=250$, $L=25$.

libraries	TRACE	ADK	EIGEN	MATEIG	MATMAT	ZKA0A	TOTAL
LaIn+BlIn	1.69	**12.86**	**165.81**	0.94	98.79	14.22	294.32
LaRe+BIII	1.16	14.87	210.85	0.83	26.70	10.46	**264.89**
LaRe+BlII	1.16	15.65	255.20	0.86	10.52	10.44	293.85
LaRe+BlIn	1.69	16.41	336.49	1.21	123.73	18.03	497.59
lnt.	1.16	12.86	165.81	0.83	10.52	10.44	201.64
LaInTh+BlInTh	**1.10**	13.92	266.63	0.66	14.13	12.34	308.80
LaRe+BlIITh	1.16	15.68	254.34	0.79	**6.66**	**9.99**	288.66
LaRe+BlInTh	**1.10**	13.71	249.59	**0.62**	13.74	11.90	290.68
lwt.	1.10	13.71	249.59	0.62	6.66	9.99	281.70
low.	1.10	12.86	165.81	0.62	6.66	9.99	197.06

If the information for matrix sizes 150 and 250 is stored in the LIFs (and for matrix size 100 the information associated to 150 is used, and for sizes 200 and 300 that associated to 250), then the execution time obtained with the model (mod.) is compared with the execution time of the best library (lib.), with the time using the best library in each part of the algorithm (low.), and with the mean of the execution time of all the libraries considered (mea.). The comparison is shown in table 7. The model gives a good approximation to the lowest achievable execution time, and the time with the model is clearly better than that with the best library, and much better than the mean time.

III. CONCLUSIONS AND FUTURE WORK

The architecture of a polylibrary has been described, and the advantage of using polylibraries has been shown with experiments with routines of different levels in the libraries hierarchy, and in different systems. The method can be applied to sequential and parallel algorithms, and it can be combined with other methods of computation speed up.

The LIF of each library will contain the cost of an operation for each one of the routines in the library. These costs may be different for different data sizes or access schemes. The main issue for the design of polylibraries is the decision of the format of the LIFs, which must be taken by the linear algebra community. This would allow the development of a hierarchy of automatic tuning libraries, with associated LIFs.

The proposed method could be applied to help in the development of efficient parallel libraries in other fields. At the moment we are researching the application to Divide and Conquer and Dynamic Programming algorithms.

TABLE VII. COMPARISON OF THE EXECUTION TIME USING THE MODEL (MOD.) WITH THE MEAN EXECUTION TIME (MEA.), THE TIME WITH THE BEST LIBRARY (LIB.) AND WITH THE BEST LIBRARY IN EACH PART OF THE ALGORITHM (BLI.). IN DUAL PENTIUM III. SQUARE MATRICES AND $L = N/10$.

size:	100	200	300
mea.	3.57	91.30	601.95
lib.	3.06	70.34	417.10
mod.	3.10	68.75	408.99
low.	3.02	66.54	406.11

IV. REFERENCES

[1] R. Whaley, A. Petitet and J. Dongarra, Automated Empirical Optimization of Software and the ATLAS Project, *Parallel Computing* 27 (2001).

[2] K. Takahiro, A Study on Large Scale Eigensolvers for Distributed Memory Parallel Machines, Ph. D. Thesis, University of Tokyo, 2001.

[3] J. Cuenca, D. Giménez and J. González, Towards the Design of an Automatically Tuned Linear Algebra Library, in 10th EUROMICRO Workshop on Parallel, Distributed and Networked Processing PDP 2002, (IEEE, 2002) 201-208.

[4] A. Kalinov and A. Lastovetsky, Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers, in High Performance Computing and Networking, eds. P. Sloot, M. Bubak, A. Hoekstra and B. Hetzberger, Lecture Notes in Computer Science, no. 1593, (Springer, 1999) 191-200.

[5] A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche and S. Vadhiyar, Numerical Libraries and The Grid, International Journal of High Performance Applications and Supercomputing 15 (2001) 359-374.

[6] J. Cuenca, D. Giménez, J. González, J. Dongarra and K. Roche, Automatic Optimisation of Parallel Linear Algebra Routines in Systems with Variable Load, in 11th EUROMICRO Workshop on Parallel, Distributed and Networked Processing PDP 2003, (IEEE, 2003) 409-416.

[7] E. A. Brewer, Portable High-Performance Supercomputing: High-Level Platform Dependent Optimization, Ph. D. Thesis, Massachusetts Institute of Technology, 1994.

[8] A. Skjellum and P. V. Bangalore, Driving Issues in Scalable Libraries: Polyalgorithms, Data Distribution Independence, Redistribution, Local Storage Schemes, in 7th SIAM Conference on Parallel Processing for Scientific Computing, (SIAM, 1995) 734-737.

[9] J. J. Dongarra, J. Du Croz, I. S. Duff and S.Hammarling, A set of Level 3 Basic Linear Algebra Subprograms, ACM Trans. Math. Soft. 16 (1990) 1-17.

[10] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, LAPACK Users' Guide (SIAM, 1999).

[11] 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 and R. C. Whaley, ScaLAPACK Users' Guide, (SIAM, 1997).

[12] Greg Henry web page, www.cs.utk.edu/~ghenry.

[13] Kazushige Goto web page, www.cs.utexas.edu/users/flame/goto.

[14] R. A. van de Geijn, Using PLAPACK: Parallel Linear Algebra Package, (The MIT Press, 1997).

[15] Message Passing Interface Forum web page, www.mpi-forum.org.

[16] PVM web page, www.epm.ornl.org.gov/pvm/pvm home.html.

[17] D. Giménez and G. Carrillo, Installation routines for linear algebra libraries on LANs, in 4th International Meeting VECPAR2000, (Faculdade de Engenharia da Universidade do Porto, 2000) 393-398.

[18] S. S. Vadhiyar, G. E. Fagg and J. Dongarra, Automatically Tuned Collective Communications, in Proceedings of the SC'2000 Conference, Dallas, Tx, Nov. 2000.

[19] J. Cuenca, D. Giménez and J. González, Modelling the Behaviour of Linear Algebra Algorithms with Message Passing, in 9th EUROMICRO Workshop on Parallel, Distributed and Networked Processing PDP 2001, (IEEE, 2001) 282-289.

[20] T. H. Cormen, C. E. Leiserson and R. L. Rivest, Introduction to Algorithms, (The MIT Press, 1990).

[21] G. H. Golub and C. F. Van Loan, Matrix Computations, (The Johns Hopkins University Press, 1989).

[22] P. Alberti, Solución Paralela de Mínimos Cuadrados para el Problema Inverso Aditivo de Valores Propios, Master's Thesis, Universidad Politécnica de Valencia, 2002.

[23] H. Park and L. Eldén, Schur-type methods for solving least squares problems with Toeplitz structure, SIAM J. Sci. Comput. 22 (2000) 406-430.

[24] P. Alonso, J. M. Badía and A. M. Vidal, A Parallel Algorithm for Solving the Toeplitz Least Squares Problem, in Vector and Parallel Processing-VECPAR 2000 eds. J. M. L. M. Palma, J. Dongarra V. Hernández, Lecture Notes in Computer Science, no. 1981, (Springer, 2001).

[25] A. Alberti, A. Alonso, A. Vidal, J. Cuenca, L. P. García and D. Giménez, Designing polylibraries to speed up linear algebra computations, TR LSI 1-2003, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Murcia, 2003. dis.um.es/~domingo/investigacion.html.

[26] ATLAS web page: www.netlib.org/atlas/index.html.