Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Parallel Computing 29 (2003) 1723–1743

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Self-adapting software for numerical linear algebra and LAPACK for clusters ☆

Zizhong Chen, Jack Dongarra *, Piotr Luszczek, Kenneth Roche

Computer Science Department, Innovative Computing Laboratory, University of Tennessee, 1122 Volunteer Blvd., Suite 203, Knoxville, TN 37996-3450, USA

Received 15 January 2003; accepted 15 May 2003

Abstract

This article describes the context, design, and recent development of the LAPACK for clusters (LFC) project. It has been developed in the framework of Self-Adapting Numerical Software (SANS) since we believe such an approach can deliver the convenience and ease of use of existing sequential environments bundled with the power and versatility of highly tuned parallel codes that execute on clusters. Accomplishing this task is far from trivial as we argue in the paper by presenting pertinent case studies and possible usage scenarios.
© 2003 Elsevier B.V. All rights reserved.

Keywords: High performance computing; LAPACK; Linear algebra; PBLAS; ScaLAPACK; Numerical parallel libraries; Self-adapting software

1. Introduction

Judging by the evolution and current state of the high performance computing industry, it is rather apparent that a steady growth of performance level is easier to achieve in hardware than in software. The computer hardware industry (and its high performance branch in particular) continues to follow Moore's law [1,2] which on one hand makes the integrated circuits faster but, on the other hand, more complex and harder to use. At the same time, the software creation process remains unchanged [3,4]. As the chip fabrication technologies change, the same gate logic will

☆ This research was sponsored by NSF through the NPACI, and DOE through the LACSI.
* Corresponding author. Fax: +1-865 9748296.
E-mail address: dongarra@cs.utk.edu (J. Dongarra).

0167-8191/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2003.05.014

invariably yield the same performance, regardless of the underlying electronic circuitry, as long as the clock speed is adequate. In contrast, performance of highly tuned software can differ drastically downward upon even slight changes in the hardware. Consequently, a different approach to software design needs to be taken as opposed to the practices from the hardware manufacturing community. Self-Adapting Numerical Software (SANS) systems are intended to meet this significant challenge [5]. In particular, the LAPACK for clusters (LFC) project [6] focuses on issues related to solving linear systems for dense matrices on highly parallel systems.

Driven by the desire of scientists for ever higher levels of detail and accuracy in their simulations, the size and complexity of required computations is growing at least as fast as the improvements in processor technology. Scientific applications need to be tuned to extract near peak performance even as hardware platforms change underneath them. Unfortunately, tuning even the simplest real-world operations for high performance usually requires an intense and sustained effort, stretching over a period of weeks or months, from the most technically advanced programmers, who are inevitably in very scarce supply. While access to necessary computing and information technology has improved dramatically over the past decade, the efficient application of scientific computing techniques still requires levels of specialized knowledge in numerical analysis, mathematical software, computer architectures, and programming languages that many working researchers do not have the time, the energy, or the inclination to acquire. With good reason scientists expect their computing tools to serve them and not the other way around. And unfortunately, the growing desire to tackle highly interdisciplinary problems using more and more realistic simulations on increasingly complex computing platforms will only exacerbate the problem. The challenge for the development of next generation software is the successful management of the complex computing environment while delivering to the scientist the full power of flexible compositions of the available algorithmic alternatives and candidate hardware resources.

With this paper we develop the concept of Self-Adapting Numerical Software for numerical libraries that execute in the cluster computing setting. The central focus is the LFC software which supports a serial, single processor user interface, but delivers the computing power achievable by an expert user working on the same problem who optimally utilizes the resources of a cluster. The basic premise is to design numerical library software that addresses both computational time and space complexity issues on the user's behalf and in a manner as transparent to the user as possible. The software intends to allow users to either link against an archived library of executable routines or benefit from the convenience of pre-built executable programs without the hassle of resolving linker dependencies. The user is assumed to call one of the LFC routines from a serial environment while working on a single processor of the cluster. The software executes the application. If it is possible to finish executing the problem faster by mapping the problem into a parallel environment, then this is the thread of execution taken. Otherwise, the application is executed locally with the best choice of a serial algorithm. The details for parallelizing the user's problem such as resource discovery, selection, and allocation, mapping the data onto (and off of) the working cluster of processors, executing the user's application in parallel, freeing the allocated resources, and returning control to the user's process in the serial environment from which the procedure began are all handled by the software. Whether the application was executed in a parallel or serial environment is presumed not to be of interest to the user but may be explicitly queried. All the user knows is that the application executed successfully and, hopefully, in a timely manner.

Alternatively, the expert user chooses a subset of processors from the cluster well suited to address memory and computational demands, initializes the parallel environment directly, generates the data set locally, in parallel on the working group in a manner effecting any necessary parallel data structures, and then executes the same parallel application.

The time spent in executing the application in parallel is, by design, expected to be the same in both the LFC and expert user cases. One significant difference between the LFC and expert cases, however, is that the self-adaptive method pays the time penalty of having to interface the user and move the user's data on and off the parallel working group of processors. Thus, for LFC, the time saved executing the application in parallel should necessarily be greater than the time lost porting the user's data in and out of the parallel environment. Empirical studies [6] of computing the solutions to linear systems of equations demonstrated the viability of the method finding that (on the clusters tested) there is a problem size that serves as a threshold. For problems greater in size than this threshold, the time saved by the self-adaptive method scales with the parallel application justifying the approach. In other words, the user saves time employing the self-adapting software.

This paper is organized as follows. Section 2 provides a general discussion of self-adaptation and its relation to software and algorithms. Section 3 presents motivation for the general concept of self-adaptation. Section 4 introduces and gives some details on LFC while Sections 5 concludes the paper.

2. Comment on self-adaptation

The hardness of a problem, in practice, may be classified by the ratio of the number of constraints to the number of variables. It is noted that achieving optimized software in the context described here is an NP-hard problem [7–13]. Nonetheless, self-adapting software attempts to tune and approximately optimize a particular procedure or set of procedures according to details about the application and the available means for executing the application. Here an attempt is made to provide a taxonomy of various approaches and a number of limitations that need to be overcome in order to apply the self-adaptation methodology to a broader range of applied numerical analyses.

In reviewing the literature, a number of generalizations emerge. First, possible optimizations may be performed through algorithmic, software or hardware changes. In the context of LFC, this may be illustrated with two important computational kernels: matrix–matrix multiplication and the solution of linear systems of equations. Table 1 shows some of the common trends for those kernels.

Table 1
Common trends in strategies for computational kernels

Computational kernel	Algorithmic choices	Software (implementation)	Hardware
xGEMM	Triple-nested loop, Strassen [14], Winograd [15]	Based on Level 1 BLAS, Level 2 BLAS, or Level 3 BLAS [16]	Vector processor, superscalar RISC, VLIW processor
Solving a linear system of equations	Explicit inverse, decompositional method (e.g. LU, QR, or LL^T)	Left-looking, right-looking, Crout [17], recursive [18,19]	Sequential, SMP, MPP, constellations [2]

Another aspect that differentiates the adaptive approaches is the time when optimization takes place—it may be performed at compilation time (off-line) or dynamically during execution (at runtime). The former category includes feedback directed compilation systems [20–22], while the latter employs two types of techniques: one is to commit to the selected algorithm and the other is to keep monitoring the performance of the selected algorithm and change it when necessary.

Yet another choice to make for an adaptive program is the selection of the search method for the best solution. It is possible to search exhaustively the entire parameter space, to use one of the generic black box optimization techniques, or to keep reducing the search space through domain-specific knowledge.

Lastly, the inclusion of input data and/or previous execution information in the optimization process also provides a differentiating factor.

In the context of the aforementioned classification, LFC makes optimization choices at the software and hardware levels. LFC obtains a best parameter set for the selected algorithm by applying expertise from the literature and empirical investigations of the core kernels on the target system. The algorithm selection depends on the size of the input data (but not the content) and empirical results from previous runs for the particular operation on the cluster. LFC makes these choices at runtime and, in the current version, commits to the decisions made—e.g. does not monitor the progress of the computation for load balancing, rescheduling, or checkpointing. It is conceivable that such a capability is useful and could be built on top of LFC. Specific hardware resources are selected at runtime based upon their ability to meet the requirements for solving the user problem efficiently. The main time constraints considered are the time spent moving the user's data set on and off the selected parallel working group of processors, and the time spent executing the specific application in parallel. The candidate resources have time-evolved information that is deemed relevant in the process of making this selection.

The ability to adapt to various circumstances may be perceived as choosing from a collection of algorithms and parameters to solve a problem. Such a concept has been appearing in the literature [23] and currently is being used in a wide range of numerical software components.

Here are some examples of successful applications and projects. The ATLAS [24] project started as a "`DGEMM()` optimizer" [25] but continues to successfully evolve by including tuning for all levels of Basic Linear Algebra Subroutines (BLAS)

[26–29] and LAPACK [30] as well as by making decisions at compilation and execution time. Similar to ATLAS, but much more limited, functionality was included in the PHiPAC [31] project. Iterative methods and sparse linear algebra operations are the main focus of numerous efforts. Some of them [32,33] target convergence properties of iterative solvers in a parallel setting while others [34–38] optimize the most common numerical kernels or provide intelligent algorithmic choices for the entire problem solving process [39,40]. In the area of parallel computing, researchers are offering automatic tuning of generic collective communication routines [41] or specific collectives as in the HPL project [42]. Automatic optimization of the fast Fourier transform (FFT) kernel has also been under investigation by many scientists [43–45]. In grid computing environments [46], wholistic approaches to software libraries and problem solving environments such as defined in the GrADS project [47] are actively being tested. Proof of concept efforts on the grid employing SANS components exist [48] and have helped in forming the approach followed in LFC.

Despite success in applying self-adapting techniques to many areas of applied numerical analysis, challenges do remain. Iterative methods for systems of linear equations are an example. It is known that an iterative method may fail to converge even if the input matrix is well conditioned. Recently, a number of techniques have been devised that try to make iterative methods faster and more robust through extra storage and work. Those techniques include running multiple iterative methods at a time and selecting results from the best performing one [33]; dynamic estimating parameters for the Chebyshev iteration [49,50]; estimating the forward error of the solution [51]; and reformulations of the conjugate gradient method to make it perform better in parallel settings [52] to name a few.

3. Motivating factors

The LFC project aims at simplifying the use of linear algebra software on contemporary computational resources, be it a single workstation or a collection of heterogeneous processor clusters. As described, it may leverage the power of parallel processing on a cluster to execute the user's problem or may execute in the sequential environment.

In dense linear algebra the Level 3 BLAS such as matrix–matrix multiplication form the basis for many applications of importance. Many modern sequential routines, for instance, exploit this kernel while executing block recursive steps during a factorization. Even more advances are available in the shared memory, multi-threaded variants. An important point in these efforts is that for a problem of size n they perform $O(n^3)$ operations on $O(n^2)$ elements—an ideal recipe for data reuse and latency hiding. The spatial locality of the data due to the proper utilization of the memory hierarchy ensures good performance in general. Indeed, the LAPACK software library has been very successful in leveraging the BLAS to achieve outstanding performance in a sequential setting. One point to be made here is that extreme care went into devising factorizations of the linear algebra to effectively utilize the highly tuned performance of the core kernels. In Fig. 1, the plot on the left shows

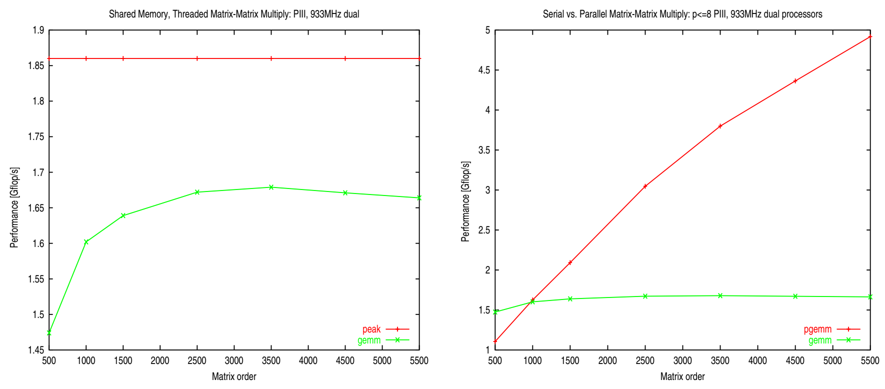


Fig. 1. Typical findings in dense, floating point matrix–matrix multiplication on a single node (left) and ported with no optimization to multiple nodes over Fast Ethernet. Despite excellent out-of-the-box performance attained by the threaded variant, expert users save time by simply executing the kernel on multiple processors in parallel. In each plot, Intel® Pentium III, 933 MHz dual processors are tested.

a typical plot of the performance of matrix–matrix multiplication that has been tuned out-of-the-box with ATLAS on each processor of a sample cluster of eight Intel® Pentium III, 933 MHz dual processors. It performs at over 90% of peak for $n = 3500$, for instance. The plot on the right in Fig. 1 presents the same sequential data against the performance achieved by simply performing the matrix–matrix multiplication in parallel on a subset of ($\leqslant 8$) the processors. No attempt was made to tune the parallel kernel in this particular set of runs. The plots demonstrate the benefits of executing this base kernel in parallel when resources are available. The finding is not surprising and in fact, once parallel versions of the BLAS were formulated (PBLAS [53]), the parallel applications routines followed in the formation of the ScaLAPACK library.

It is noteworthy that matrix–matrix multiplication in the sequential environment serves as a core kernel for the parallel Level 3 BLAS routines utilized by ScaLAPACK. The situation merits a closer look at the kernel in the sequential setting. There are, for instance, cases where the performance is degraded. Figs. 2 and 3 show the performance of this Level 3 BLAS routine on the Intel® Itanium®, Intel® Itanium® SMP and IBM Power 4 processors, respectively. Matrices of dimensions close to 2050 consistently have worse performance than all others due to complex cache effects. Ideally, in the sense of self-adaptation, the BLAS should switch to a different algorithm to circumvent this problem. In the current implementation, however, it does not happen. For users working in a sequential environment, the problem must be handled or otherwise the consequences paid. This makes the assumption that the user knows of the problem and has a remedy.

Another problematic area as far as users' responsibilities are concerned is the linear solver. LAPACK requires a tuning parameter—a block size—which is crucial to attaining high performance. If LAPACK's functionality was embedded in an archived library which was supplied by the vendor then the burden of selecting the

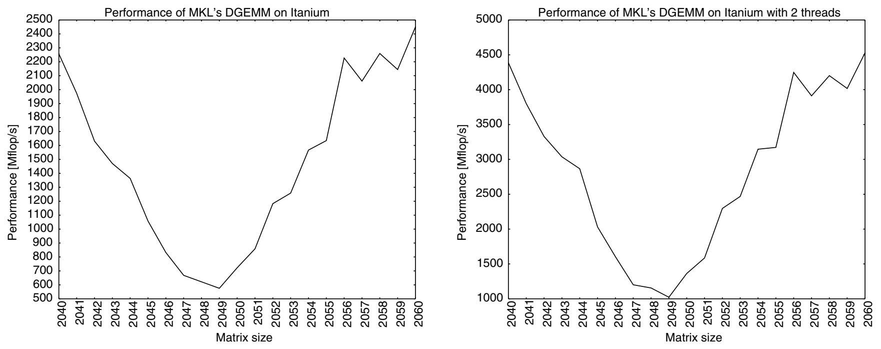


Fig. 2. Performance of Intel® MKL 5.1.1 on Intel® Itanium® 800 MHz with one and two threads.

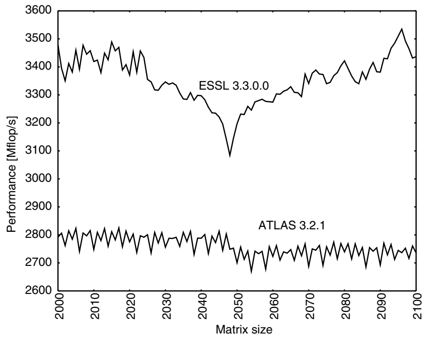


Fig. 3. Performance of Level 3 BLAS `DGEMM()` routine from ATLAS 3.2.1 and ESSL 3.3.0.0 on IBM Power 4 1.3 GHz processor.

block size would have been removed from the user. However, if the vendor supplies only a BLAS library then the block size selection is to be made by the user and there is a possibility of degrading the performance by inappropriate choice. Thus, all the effort that went into tuning the BLAS may be wasted.

It is possible to solve the problem in a sequential environment because of theoretical advances [18,19,54] in the decompositional approach in matrix computations. But in a parallel setting, the procedure is still not mature enough [55] and consequently there is a need for extra effort when selecting parameters that will define the parallel runtime environment for the specific application.

A potential user must, for instance, select the number of processors to accomodate the problem such that a logically rectangular processor grid can be formed, and decompose the data according to another set of parameters onto said processors. Suppose that the data has been handled accurately and with parameters known to preserve a good computation to communication ratio for a *set* number of processors (a non-trivial task in practice). Now, consider Fig. 4. The figure (linear in

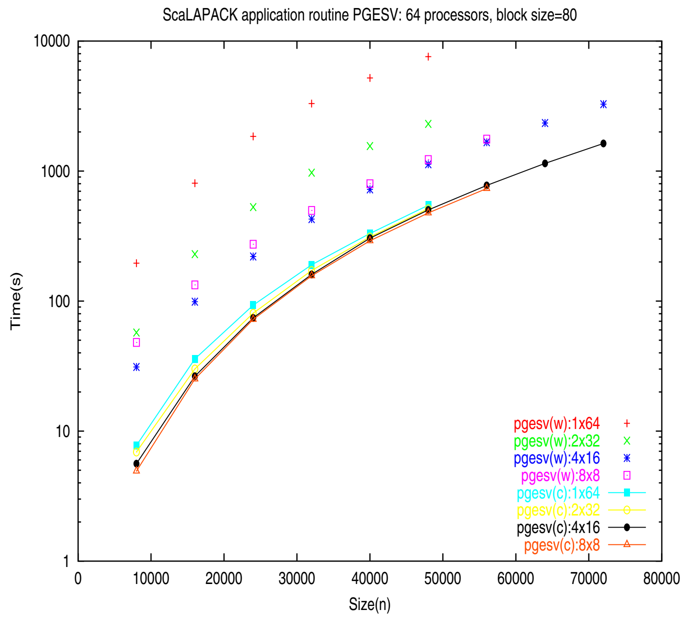


Fig. 4. Wall clock (w) and CPU (c) times are reported for solving a system of linear equations with ScaLAPACK routine `PGESV()` on 32 Intel® XEON(TM) 2.40 GHz dual processors and for a pre-determined block size. Problem sizes up to 80 K are reported. Each grid topology spends effectively the same amount of time in the CPU computing (lines + points) but clearly the wall time (points only) for each topology is dramatically different in the best (4×16) and worst (1×64) cases.

problem size, logarithmic in time) compares problem size to time for the ScaLAPACK application routine that solves systems of linear equations. Both the wall and CPU times are measured while executing the application on a fixed number (32) of nodes (each being a Intel® XEON(TM) CPU 2.40 GHz dual processor connected over Gigabit Ethernet). The possible different grid topologies tested are 1×64, 2×32, 4×16, and 8×8. For each topology tested both wall (points) and CPU (lines + points) times are reported. The wall time is the time the user cares about. The difference in the two times is accounted for by the time spent coordinating the parallel application and moving the data set with message passing—e.g. *communication time*. The CPU times reported are consistent for each grid topology as expected since there is an effectively fixed number of computations to be performed for each problem size. The wall clock times, however, are dramatically different across the different topologies. It is known that different processor topologies impose different communication patterns during execution. Thus, the turnaround time is directly related to the user's selection of grid topology even in the instance that the right number of processors and a judicious block size are given.

Fig. 5 illustrates the fact that the situation is more complicated than just selecting the right grid aspect ratio (e.g. the number of process rows divided by the number of process columns). Sometimes it might be beneficial to use a smaller number of pro-

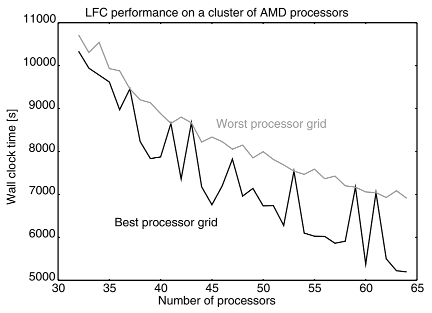


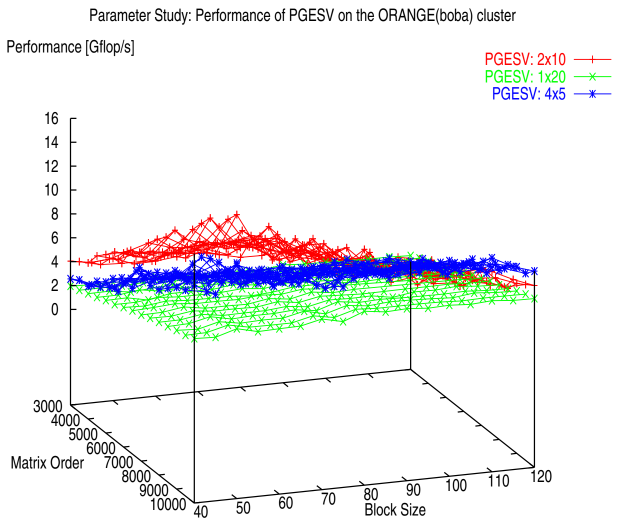
Fig. 5. Timing results for solving a linear system of order 70 K with best and worst possible processor grid shapes reported.

cessors. This is especially true if the number of processors is a prime number which leads to a flat process grid and thus very poor performance on many systems. It is unrealistic to expect that non-expert users will correctly make the right decisions here. It is either a matter of having expertise or experimental data to guide the choice and our experiences suggest that perhaps a combination of both is required to make good decisions consistently. As a side note, with respect to experimental data, it is worth mentioning that the collection of data for Fig. 5 required a number of floating point operations that would compute the LU factorization of a square dense matrix of order almost 300,000. Matrices of that size are usually suitable for supercomputers (the slowest supercomputer on the Top500 [2] list that factored such a matrix was on position 16 in November 2002).

Lastly, the plots in Fig. 6 represent exhaustive search data. Such information comes from a sweeping parameter study in a dedicated environment and is provided here to drive home the point that even experienced users have to carefully initialize the parameters required for a parallel computation using the ScaLAPACK library. Here, the case of the performance of 20 processors on the cluster is compared as a function of the block size, problem size, and grid aspect ratio. We see crossing points in this multi-parameter space which suggests the very real complexity inherent in selecting parameters judiciously. Ideally, given a user's problem, an oracle would be consulted and the appropriate parameters would be assigned. In reality, extreme time and energy go into making such exhaustive studies of applications and the parameter spaces that dictate their execution. In general, such data do not exist a priori on a target system. Furthermore, in open systems such exhaustive searches fail to yield reliably intelligent decisions due to the potentially dynamic state of the available resources.

4. LAPACK for clusters overview

The LFC software addresses the motivating factors from the previous section in a self-adapting fashion. LFC assumes that only a C compiler, an MPI [56–58]



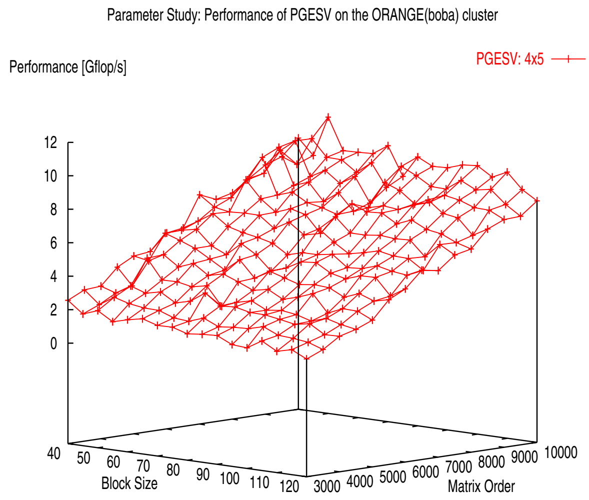


Fig. 6. Oracle data taken from a dedicated system with a set number of processors. The parallel application routine is `PGESV()`, the (LU) linear system solver from ScaLAPACK. For a user working on an open cluster this multi-parameter space is also changing in time.

implementation such as MPICH [59] or LAM MPI [60], and some variant of the BLAS routines, be it ATLAS or a vendor supplied implementation, is installed on the target system. Target systems are intended to be "Beowulf like" and may be depicted as in the diagram of Fig. 7.

There are essentially three components to the software: data collection routines, data movement routines, and application routines.

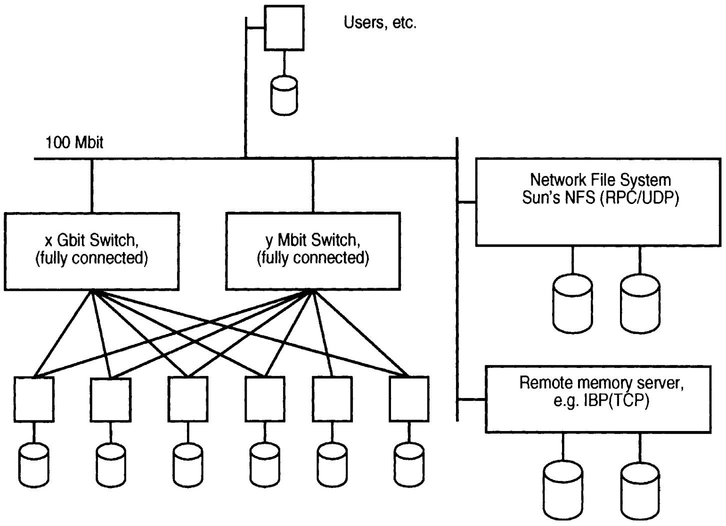

Users, etc.
100 Mbit
x Gbit Switch, (fully connected)
y Mbit Switch, (fully connected)
Network File System Sun's NFS (RPC/UDP)
Remote memory server, e.g. IBP(TCP)

Fig. 7. Here a typical cluster setting is depicted. The cluster is regarded as fully connected locally and sees a network disk that serves users. Users are assumed to be logged into a compute node of the target cluster on invoking the LFC software.

4.1. Data collection

LFC uses discovery and adaptivity to assist the user in problem solving. The LFC routine starts with assessing information that is continually being assembled about the state of the cluster and the states of the components of the cluster. (The service is similar to that provided by the Network Weather Service [61], NWS, sensors in grid computing environments.) The following steps are repeated by the information gathering daemon process: a processor discovery routine is invoked that accounts for the existence of candidate resources, the available physical memory per processor is assessed, the time-averaged CPU load of each processor in a node is assessed, read/write times per processor to/from the local and network disks is assessed, point-to-point and global communications latencies and bandwidths on the cluster are assembled, and the core kernel of matrix–matrix multiplication is studied per processor. In addition to this data gathering cycle, there is an interest in the one-time discovery of the underlying memory hierarchy. Random access loads and stores with uniform and non-uniform stride help with this discovery. Fig. 8 shows an example of cache to memory bandwidth discovery.

One of the difficulties that arises is clock consistency and synchronization. On homogeneous clusters, the problem is difficult. In a grid computing setting, the difficulties are amplified. This is a serious issue which software developers need to address. The main reason is that scheduling tasks based upon empirical analysis conducted on a target system assumes consistency in the resources. LFC uses matrix–matrix multiplication to assess the consistency of internal clocks accorded by several timers on each of the processors available.

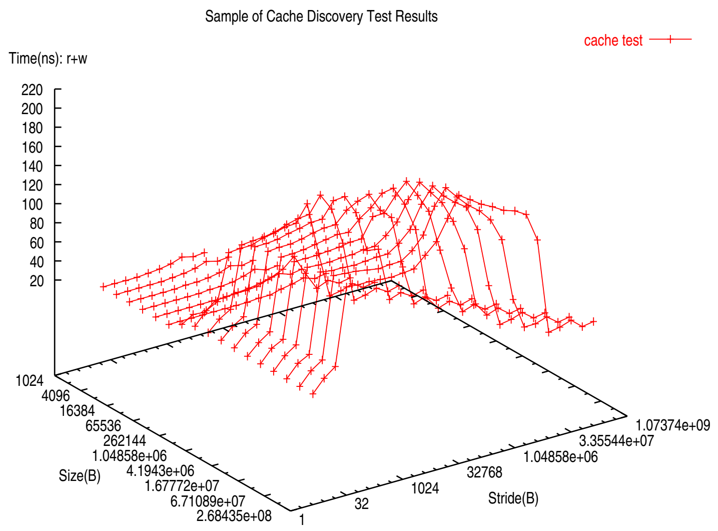


Fig. 8. The plot is of the means of times gathered for loading and storing an element from a block of *size*(B) bytes with a non-uniform stride of size *stride*(B) bytes. Non-uniform memory access studies are being studied as a means for obtaining application specific signatures of memory access patterns [62].

4.2. Data movement

In LFC, regardless of the application, the same data structures are assumed in the case of dense data. The user data is assumed to be in core or on a disk accessible from the cluster in either row or column major storage. If the problem is to be ported to the parallel environment, the two-dimensional block cyclical decomposition is applied in each case. That is, whether the user wants to solve a system of linear equations or an eigenvalue problem, her/his data set is mapped onto the processors allocated as the working group with the same data routines and according to the same rules. The decomposition is known to yield good performance and scalability in (local) distributed computing settings employing message passing [63–66].

Once a decision has been made to solve the user's problem in parallel on the cluster, the user data has to be mapped onto a specific subset of the processors and in a specific (logical) rectangular grid topology. It is noted that the number of processors, which processors, and what topology are not known in advance. As such, the mapping has to be general. After the execution of the parallel application, the data has to be reverse mapped from the parallel environment back to the user's original data structure and location.

The dimensions m, n of the matrix A, the (logical) rectangular processor grid dimensions p, q (process rows and columns, respectively; $p \times q = N_P$ where N_P is the total number of processors involved in the decomposition), and the block dimensions mb, nb where $(1 \leqslant mb \leqslant m)$, $(1 \leqslant nb \leqslant n)$ are the parameters which define the 2D block cyclic mapping. The values m and n are set by the user, however the remaining parameters are initialized based upon an analysis of the cluster and a knowledge of the signatures relevant to the particular application. Once the number of processors, logical process grid dimensions, and the block sizes for the decomposition are decided, the local memory per allocated processor is determined. The number of rows from the global $m \times n$ matrix A that processors in logical process row ip $(0 \leqslant ip < p)$ own is defined by m_{ip}. The number of columns from A that processors in logical process column iq own is defined by n_{iq}. For processor (ip, iq) from the logical process grid, the *local* work array $A_{ip,iq}$ has dimensions m_{ip} rows by n_{iq} columns and $m = \sum_{ip=0}^{p-1} m_{ip}$, $n = \sum_{iq=0}^{q-1} n_{iq}$ where m_{ip}, n_{iq} are calulated as follows:

$$m_{ip} = \begin{cases} \left(\left\lfloor \frac{\lfloor \frac{m}{mb} \rfloor}{p} \right\rfloor + 1\right) mb & \text{if } ((p + ip) \bmod p) < \left(\lfloor \frac{m}{mb} \rfloor \bmod p\right), \\ \left\lfloor \frac{\lfloor \frac{m}{mb} \rfloor}{p} \right\rfloor mb + m \bmod mb & \text{if } ((p + ip) \bmod p) = \left(\lfloor \frac{m}{mb} \rfloor \bmod p\right), \\ \left\lfloor \frac{\lfloor \frac{m}{mb} \rfloor}{p} \right\rfloor mb & \text{if } ((p + ip) \bmod p) > \left(\lfloor \frac{m}{mb} \rfloor \bmod p\right), \end{cases}$$

$$n_{iq} = \begin{cases} \left(\left\lfloor \frac{\lfloor \frac{n}{nb} \rfloor}{q} \right\rfloor + 1\right) nb & \text{if } ((q + iq) \bmod q) < \left(\lfloor \frac{n}{nb} \rfloor \bmod q\right), \\ \left\lfloor \frac{\lfloor \frac{n}{nb} \rfloor}{q} \right\rfloor nb + n \bmod nb & \text{if } ((q + iq) \bmod q) = \left(\lfloor \frac{n}{nb} \rfloor \bmod q\right), \\ \left\lfloor \frac{\lfloor \frac{n}{nb} \rfloor}{q} \right\rfloor nb & \text{if } ((q + iq) \bmod q) > \left(\lfloor \frac{n}{nb} \rfloor \bmod q\right). \end{cases}$$

(It is noted that processor $(ip, iq) = (0, 0)$ owns the element $a_{0,0}$ in the presentation here.)

In global to local 2D block cyclic index mappings in which natural data is transformed into 2D block cyclically mapped data, the values (i, j) from $A(i, j)$ are given. Next, the grid dimensions (ip, iq) of the processor that owns the specific element are identified and initialized as $ip = \lfloor \frac{i}{mb} \rfloor \bmod p$ and $iq = \lfloor \frac{j}{nb} \rfloor \bmod q$. The local indices of the work array on processor (ip, iq) can be labelled (i_{ip}, j_{iq}) where $0 \leqslant i_{ip} < m_{ip}$, $0 \leqslant j_{iq} < n_{iq}$. The assignment is $i_{ip} = \left\lfloor \frac{\lfloor \frac{i}{mb} \rfloor}{p} \right\rfloor \cdot mb + (i \bmod mb)$ and $j_{iq} = \left\lfloor \frac{\lfloor \frac{j}{nb} \rfloor}{q} \right\rfloor \cdot nb + (j \bmod nb)$.

On a particular processor (ip, iq), given the indices (i_{ip}, j_{iq}) of the local work array $A_{ip,iq}$ (where $0 \leqslant i_{ip} < m_{ip}$, $0 \leqslant j_{iq} < n_{iq}$), the block dimensions (mb, nb), and the process grid dimensions (p, q) then the indices (i, j) of the global matrix element $A(i, j)$ are assigned as follows: $i = ip \cdot mb + \lfloor \frac{i_{ip}}{mb} \rfloor \cdot p \cdot mb + (i_{ip} \bmod mb)$ and $j = iq \cdot nb + \lfloor \frac{j_{iq}}{nb} \rfloor \cdot q \cdot nb + (j_{iq} \bmod nb)$. This is the reverse 2D block cyclic map in which the mapped data is transformed into natural data.

In the process of mapping the user's data from the serial environment to the parallel process group selected by the LFC scheduler, direct global to local index mappings are not used. The reason is that, clearly, moving a single matrix element at a time is extremely inefficient. The game is to get the user's data accurately mapped onto the parallel process grid as quickly as possible. Serial reads and writes to local and network based disks, parallel reads from a single file stored on a common, single network disk, parallel reads and writes from/to multiple files on a (common) single network disk, and parallel reads and writes from multiple files on multiple unique network disks are all different possible operations that may be invoked in the process of handling the user's data set. Usually, however, it is found that some combination of these operations is preferred to get the data in place on the cluster in the correct block cyclical mapping. Furthermore, it is noted that the mapping itself may occur at the time of the write which may result in multiple files in *pre-mapped* form or a single file reflecting the mapped structure, at the time of a read (e.g. random access reads into a file, or possibly multiple files, containing the unmapped data), when the data is communicated during message passing on the cluster, or parsed locally in memory on each processor after blocks of unmapped data are read into memory either in parallel or by a subset of lead processors and distributed through message passing in a manner reflective of the 2D block cyclic mapping.

It is noted that space complexity issues may also be addressed on the users behalf by interfacing special utility routines. The idea here is that the user wishes to state and solve a problem that is too large to be addressed on any single node of the cluster. The utility assists in the generation of the large data set on a subset of the cluster presuming there is ample total memory to accommodate the problem on the allocated systems.

Predicting the time to move large data sets in an open network is an active area of research. In some sense, the limitations are well defined by the IP family of protocols and the disk access times. There is no best means for handling the user's data. The main constraint is that the integrity of the user's data set is preserved. The hope is that this stage can be performed in a timely manner so as to reap the benefits of the parallel application. LFC researchers are interested in multiple network disk resources that can be used specifically for assisting in the management of data sets from linear algebra. Such studies are relevant to grid settings as well as local clusters. The work is not discussed further here.

4.3. Applications

In the sequential environment, a stand-alone variant of the relevant LAPACK routines form the backbone of the serial applications in LFC. Achieving high performance in a sequential environment might seem trivial for expert users. Thus, we provide linker hooks to enable such users to use their favorite BLAS library. However, less experienced users could possibly have problems while dealing with linker dependencies. For such users, we provide an executable binary that is correctly built and capable of solving a linear system in a child process with data submitted through a system pipe. Two overheads result from such an approach: the time spent in `fork(2)` and `exec(3)` system calls and copying the data between separate process' address spaces. Intuitively, both overheads will have a lesser impact with increasing dimension of the matrix (the system calls, data copying and linear solver have computational complexities O(1), O(n^2), and O(n^3) respectively). To determine how this theoretical result translates into real world performance, we ran some tests and Fig. 9 shows the results. Matrices of dimension as small as 500 see only 10% of performance drop and the difference decreases to about 1% for dimension 2000. We believe that for many users this is a price worth paying for convenience and certainty.

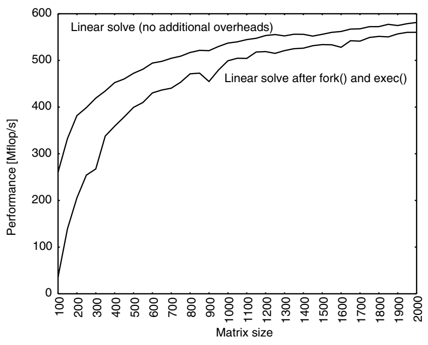


Fig. 9. Performance comparison between the standard LAPACK's linear solve routine and the same routine executed in a separate process created with `fork(2)` and `exec(3)` system calls (matrix data are sent through a system pipe). The tested machine had a Intel® Pentium III 933 MHz processor.

The parallel applications have a stand-alone variant of the relevant ScaLAPACK and BLACS routines. This allows leveraging a large body of expertise as well as software design and engineering. It also allows developers to focus on new issues and address common problems encountered by users.

ScaLAPACK Users' Guide [67] provides the following equation for predicting the total time T spent in one of its linear solvers (LL^T, LU, or QR) [68]:

$$T(n, N_P) = \frac{C_f n^3}{N_P} t_f + \frac{C_v n^2}{\sqrt{N_P}} t_v + \frac{C_m n}{N_B} t_m \tag{1}$$

where

- t_f time per floating-point operation (matrix–matrix multiplication flop rate is a good starting approximation)
- t_m corresponds to latency
- $1/t_v$ corresponds to bandwidth
- C_f corresponds to number of floating-point operations (see Table 2)
- C_v and C_m correspond to communication costs (see Table 2).

In contrast, for a single processor the equation is:

$$T_{seq}(n) = C_f n^3 t_f \tag{2}$$

Eq. (1) yields surprisingly good predictions. The surprise factor comes from the number of simplifications that were made in the model which was used to derive the equation. The hard part in using the equation is measuring system parameters which are related to some of the variables in the equation. The hardship comes from the fact that these variables do not correspond directly to typical hardware specifications and cannot be obtained through simple tests. In a sense, this situation may be regarded as if the equation had some hidden constants which are to be discovered in order to obtain reasonable predictive power. At the moment we are not aware of any

Table 2
Performance parameters of ScaLAPACK. All costs entries correspond to a single right-hand side; LU, LL^T and QR correspond to `PxGESV`, `PxPOSV`, and `PxGELS` routines, respectively

Driver	C_f	C_v	C_m
LU	2/3	$3 + 1/4 \log_2 N_P$	$N_B(6 + \log_2 N_P)$
LL^T	1/3	$2 + 1/2 \log_2 N_P$	$4 + \log_2 N_P$
QR	4/3	$3 + \log_2 N_P$	$2(N_B \log_2 N_P + 1)$

reliable way of acquiring those parameters and thus we rely on parameter fitting approach that uses timing information from previous runs.

4.4. Typical usage scenario

Here the steps involved in a typical LFC run are described.

The user has a problem that can be stated in terms of linear algebra. The problem statement is addressable with one of the LAPACK routines supported in LFC. For instance, suppose that the user has a system of n linear equations with n unknowns, $Ax = b$.

There is a parallel computing environment that has LFC installed. The user is, for now, assumed to have access to at least a single node of said parallel computing environment. This is not a necessary constraint—rather a simplifying one.

The user compiles the application code (that calls LFC routines) linking with the LFC library and executes the application from a sequential environment. The LFC routine executes the application returning an error code denoting success or failure. The user interprets this information and proceeds accordingly.

Again, the details of how LFC handles the user's data and allocates a team of processors to execute the user's problem remain hidden to the user. (If desired a user can ask for details of the actual computation.) From the user's perspective, the entire problem was addressed locally.

A decision is made upon how to solve the user's problem by coupling the cluster state information with a knowledge of the particular application. Specifically, a decision is based upon the scheduler's ability to successfully predict that a particular subset of the available processors on the cluster will enable a reduction of the total time to solution when compared to serial expectations for the specific application and user parameters. The relevant times are the time that is spent handling the user's data before and after the parallel application plus the amount of time required to execute the parallel application.

If the decision is to solve the user's problem locally (sequentially) then the relevant LAPACK routine is executed.

If the decision is to solve the user's problem in parallel then a process is forked that will be responsible for spawning the parallel job and the parent process waits for its return in the sequential environment. The selected processors are allocated (in MPI), the user's data is mapped (block cyclically decomposed) onto the processors (the data may be in memory or on disk), the parallel application is executed (e.g. ScaLAPACK), the data is reverse mapped, the parallel process group is freed, and the solution and control are returned to the user's process.

5. Conclusions and future work

As computing systems become more powerful and complex it becomes a major challenge to tune applications for high performance. We have described a concept and outlined a plan to develop numerical library software for systems of linear equations which adapts to the user's problem and the computational environment in an attempt to extract near optimum performance. This approach has applications beyond solving systems of equations and can be applied to most other areas where users turn to a library of numerical software for their solution.

At runtime our software makes choices at the software and hardware levels for obtaining a best parameter set for the selected algorithm by applying expertise from the literature and empirical investigations of the core kernels on the target system. The algorithm selection depends on the size of the input data and empirical results

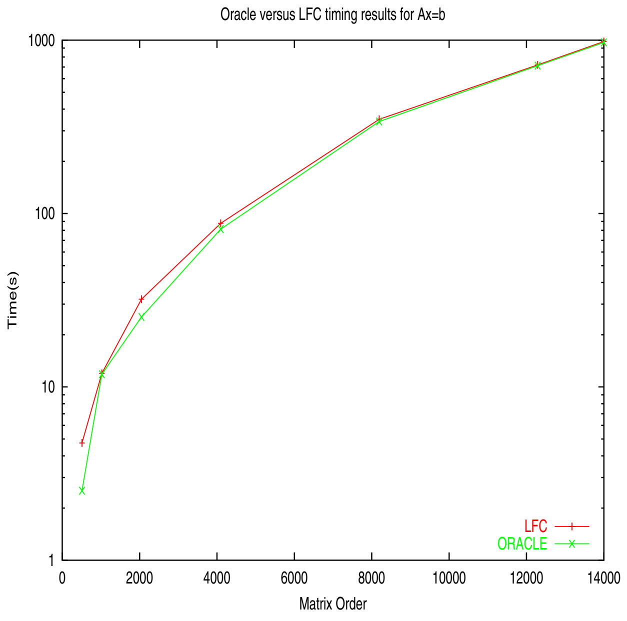


Fig. 10. The plot demonstrates the strength of the self-adapting approach of the LFC software. The problem sizes tested were $N = 512, 1024, 2048, 4096, 8192, 12288, 14000$. LFC chose 2, 3, 6, 8, 12, 16, 16 processes for these problems respectively. The oracle utilized 4, 4, 8, 10, 14, 16, 16 processes respectively. The runs were conducted on a cluster of eight Intel® Pentium III, 933 MHz dual processors, connected with a 100 Mb/s switch. In each run the data was assumed to start on disk and was written back to disk after the factorization. In the parallel environment both the oracle and LFC utilized the I/O routines from ROMIO to load (store) the data in a 2d block cyclical (natural) manner before (after) invoking the ScaLAPACK routine PDGESV.

from previous runs for the particular operation on the cluster. The overheads associated with this dynamic adaptation of the user's problem to the hardware and software systems available can be minimal.

The results presented here show unambiguously that the concepts of self-adaptation can come very close to matching the performance of the best choice in parameters for an application written for a cluster. As Fig. 10 highlights, the overhead to achieve this is minimal and the performance levels are almost indistinguishable. As a result the burden on the user is removed and hidden in the software.

This paper has given a high level overview of the concepts and techniques used in self-adapting numerical software. There are a number of issues that remain to be investigated in the context of this approach [5]. Issues such as adapting to a changing environment during execution, reproducibility of results when solving the same problem on differing numbers of processors, fault tolerance, rescheduling in the presence of additional load, dynamically migrating the computation, etc. all present additional challenges which are ripe for further investigation. In addition, with Grid computing becoming mainstream, these concepts will find added importance [47].

Acknowledgements

We wish to thank the Ohio Supercomputing Center (OSC), the Computational Science and Mathematics Division at Oak Ridge National Laboratory (XTORC cluster), the Center for Computational Sciences at Oak Ridge National Laboratory (Cheetah, Eagle), the Dolphin donation cluster (part of the SinRG program at the University of Tennessee Knoxville), the San Diego Supercomputing Center (SDSC), and the National Energy Research Scientific Computing Center (NERSC) for research conducted on their resources. We also wish to thank NPACI, the National Partnership for the Advancement of Computational Infrastrucure, for including LFC in its NPACkage.

References

[1] G.E. Moore, Cramming more components onto integrated circuits, Electronics 38 (8) (1965).

[2] H.W. Meuer, E. Strohmaier, J.J. Dongarra, H.D. Simon, Top 500 Supercomputer Sites, 20th ed., The report can be downloaded from: <http://www.netlib.org/benchmark/top500.html>, November 2002.

[3] F.P. Brooks Jr., The Mythical Man-Month: Essays on Software Engineering, second ed., Addison Wesley Professional, 1995.

[4] F.P. Brooks Jr., No silver bullet: essence and accidents of software engineering, Information Processing.

[5] J. Dongarra, V. Eijkhout, Self-adapting numerical software for next generation applications, Technical Report, Innovative Computing Laboratory University of Tennessee, Available from: <http://icl.cs.utk.edu/iclprojects/pages/sans.html>, August 2002.

[6] K.J. Roche, J.J. Dongarra, Deploying parallel numerical library routines to cluster computing in a self adapting fashion, in: Joubert, Murli, Peters, Vanneschi (Eds.), Parallel Computing: Advances and Current Issues, Imperial College Press, London, England, 2002.

[7] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, 1979.

[8] A compendium of NP optimization problems, Available from: <http://www.nada.kth.se/theory/problemlist.html>.

[9] D. Hochbaum, D. Shmoys, A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approach, SIAM Journal of Computing 17 (1988) 539–551.

[10] J. Lenstra, D. Shmoys, E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, Mathematical Programming 46 (1990) 259–271.

[11] J. Gergov, Approximation algorithms for dynamic storage allocation, in: Proceedings of the 4th Annual European Symposium on Algorithms, Lecture Notes in Computer Science, 1136, Springer-Verlag, 1996, pp. 52–56.

[12] E. Amaldi, V. Kann, On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems, Theoretical Computer Science 209 (1998) 237–260.

[13] V. Kann, Strong lower bounds on the approximability of some NPO PB—complete maximization problems, in: Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, 969, Springer-Verlag, 1995, pp. 227–236.

[14] V. Strassen, Gaussian elimination is not optimal, Numerical Mathematics 13 (1969) 354–356.

[15] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, Journal of Symbolic Computation 9 (1990) 251–280.

[16] J.W. Demmel, Applied Numerical Linear Algebra, Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[17] J.J. Dongarra, I.S. Duff, D.C. Sorensen, H.A. van der Vorst, Numerical Linear Algebra for High-Performance Computers, Society for Industrial and Applied Mathematics, Philadelphia, 1998.

[18] F.G. Gustavson, Recursion leads to automatic variable blocking for dense linear-algebra algorithms, IBM Journal of Research and Development 41 (6) (1997) 737–755.

[19] S. Toledo, Locality of reference in LU decomposition with partial pivoting, SIAM Journal on Matrix Analysis and Applications 18 (4) (1997) 1065–1081.

[20] G. Ammons, T. Ball, J.R. Larus, Exploiting hardware performance counters with flow and context sensitive profiling, in: Proceedings of ACM SIGPLAN'97 Conference on Programming Language Design and Implementation, Las Vegas, Nevada, USA, 1997.

[21] T. Ball, J.R. Larus, Efficient path profiling, in: Proceedings of MICRO '96, Paris, France, 1996.

[22] P.P. Chang, S.A. Mahlke, W.W. Hwu, Using profile information to assist classic code optimization, Software Practice and Experience 21 (12) (1991) 1301–1321.

[23] J.R. Rice, On the construction of poly-algorithms for automatic numerical analysis, in: M. Klerer, J. Reinfelds (Eds.), Interactive Systems for Experimental Applied Mathematics, Academic Press, 1968, pp. 31–313.

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

[25] J.J. Dongarra, C.R. Whaley, Automatically tuned linear algebra software (ATLAS), in: Proceedings of SC'98 Conference, IEEE, 1998.

[26] J.J. Dongarra, J.D. Croz, I.S. Duff, S. Hammarling, Algorithm 679: a set of level 3 basic linear algebra subprograms, ACM Transactions on Mathematical Software 16 (1990) 1–17.

[27] J.J. Dongarra, J.D. Croz, I.S. Duff, S. Hammarling, A set of level 3 basic linear algebra subprograms, ACM Transactions on Mathematical Software 16 (1990) 18–28.

[28] J.J. Dongarra, J.D. Croz, S. Hammarling, R. Hanson, An extended set of FORTRAN basic linear algebra subprograms, ACM Transactions on Mathematical Software 14 (1988) 1–17.

[29] J.J. Dongarra, J.D. Croz, S. Hammarling, R. Hanson, Algorithm 656: an extended set of FORTRAN basic linear algebra subprograms, ACM Transactions on Mathematical Software 14 (1988) 18–32.

[30] E. Anderson, Z. Bai, C. Bischof, S.L. Blackford, J.W. Demmel, J.J. Dongarra, J.D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, D.C. Sorensen, LAPACK User's Guide, third ed., Society for Industrial and Applied Mathematics, Philadelphia, 1999.

[31] J. Bilmes, K. Asanovic, C. Chin, J. Demmel, Optimizing matrix multiply using PHiPAC: a portable, high-performance, ANSI C coding methodology, in: Proceedings of International Conference on Supercomputing, ACM SIGARC, Vienna, Austria, 1997.
[32] R. Weiss, H. Haefner, W. Schoenauer, LINSOL (LINear SOLver)—Description and User's Guide for the Parallelized Version, University of Karlsruhe Computing Center, 1995.
[33] R. Barrett, M. Berry, J. Dongarra, V. Eijkhout, C. Romine, Algorithmic bombardment for the iterative solution of linear systems: a poly-iterative approach, Journal of Computational and Applied Mathematics 74 (1–2) (1996) 91–109.
[34] R. Agarwal, F. Gustavson, M. Zubair, A high-performance algorithm using preprocessing for the sparse matrix–vector multiplication, in: Proceedings of International Conference on Supercomputing, 1992.
[35] E.-J. Im, K. Yelick, Optimizing sparse matrix-vector multiplication on SMPs, in: Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, Texas, 1999.
[36] E.-J. Im, Automatic optimization of sparse matrix-vector multiplication, Ph.D. Thesis, University of California, Berkeley, California, 2000.
[37] S. Toledo, Improving the memory-system performance of sparse matrix–vector multiplication, IBM Journal of Research and Development 41 (6) (1997).
[38] A. Pinar, M.T. Heath, Improving performance of sparse matrix–vector multiplication, in: Proceedings of SC'99, 1999.
[39] A. Bik, H. Wijshoff, Advanced compiler optimizations for sparse computations, Journal of Parallel and Distributing Computing 31 (1995) 14–24.
[40] J. Ostergaard, OptimQR—a software package to create near-optimal solvers for sparse systems of linear equations, Available from: <http://ostenfeld.dk/jakob/OptimQR/>.
[41] S. Vadhiyar, G. Fagg, J.J. Dongarra, Performance modeling for self adapting collective communications for MPI, In: Los Alamos Computer Science Institute Symposium (LACSI 2001), Sante Fe, New Mexico, 2001.
[42] J.J. Dongarra, P. Luszczek, A. Petitet, The LINPACK benchmark: past, present, and future, Concurrency and Computation: Practice and Experience 15 (2003) 1–18.
[43] M. Frigo, S.G. Johnson, FFTW: an adaptive software architecture for the FFT, in: Proceedings International Conference on Acoustics, Speech, and Signal Processing, Seattle, Washington, USA, 1998.
[44] M. Frigo, A fast Fourier transform compiler, in: Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation, Atlanta, Georgia, USA, 1999.
[45] D. Mirkovic, S.L. Johnsson, Automatic performance tuning in the UHFFT library, in: 2001 International Conference on Computational Science, San Francisco, California, USA, 2001.
[46] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Francisco, 1999.
[47] F. Berman, The GrADS project: software support for high level grid application development, International Journal of High Performance Computing Applications 15 (2001) 327–344.
[48] A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, S. Vadhiyar, Numerical libraries and the grid, International Journal of High Performance Computing Applications 15 (2001) 359–374.
[49] T.A. Manteuffel, Adaptive procedure for estimating parameters for the nonsymmetric Tchebyshev iteration, Numerische Mathematik 31 (1978) 183–208.
[50] T.A. Manteuffel, The Tchebyshev iteration for nonsymmetric linear systems, Numerische Mathematik 28 (1977) 307–327.
[51] E.F. Kaasschieter, A practical termination criterion for the Conjugate Gradient method, BIT 28 (1988) 308–322.
[52] V. Eijkhout, Computational variants of the CGS and BiCGSTAB methods, Technical Report CS-94-241, Computer Science Department, The University of Tennessee Knoxville, August 1994 (Also LAPACK Working Note No. 78).
[53] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, R.C. Whaley, A proposal for a set of parallel basic linear algebra subprograms, Technical Report CS-95-292, University of Tennessee Knoxville, LAPACK Working Note 100, May 1995.

[54] E. Elmroth, F.G. Gustavson, New serial and parallel recursive QR factorization algorithms for SMP systems, in: Proceedings of PARA 1998, 1998.

[55] D. Irony, S. Toledo, Communication-efficient parallel dense LU using a 3-dimensional approach, in: Proceedings of the 10th SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, Virginia, USA, 2001.

[56] M.P.I. Forum, MPI: A Message-Passing Interface Standard, The International Journal of Supercomputer Applications and High Performance Computing 8 (1994).

[57] M.P.I. Forum, MPI: A Message-Passing Interface Standard (version 1.1), 1995, Available from: <http://www.mpi-forum.org/>.

[58] M.P.I. Forum, MPI-2: Extensions to the Message-Passing Interface, 18 July 1997, Available from: <http://www.mpi-forum.org/docs/mpi-20.ps>.

[59] MPICH, Available from: <http://www.mcs.anl.gov/mpi/mpich/>.

[60] LAM/MPI parallel computing, Available from: <http://www.mpi.nd.edu/lam/>.

[61] R. Wolski, N. Spring, H. Hayes, The network weather service: a distributed resource performance forecasting service for metacomputing, Future Generation Computing Systems 14 (1998).

[62] A. Snavely et al., A framework for performance modeling and prediction, in: Proceedings of Supercomputing 2002, IEEE, 2002.

[63] H.P. Forum, High performance fortran language specification, Technical Report CRPC-TR92225, Center for Research on Parallel Computing, Rice University, Houston, TX, May 1993.

[64] A. Petitet, Algorithmic redistribution methods for block cyclic decompositions, Ph.D. Thesis, University of Tennessee, Knoxville, Tennessee, 1996.

[65] O. Beaumont, A. Legrand, F. Rastello, Y. Robert, Dense linear algebra kernels on heterogeneous platforms: redistribution issues, Parallel Computing 28 (2) (2002) 155–185.

[66] W. Lichtenstein, S. Johnson, Block-cyclic dense linear algebra, SIAM Journal on Scientific and Statistical Computing 14 (1993) 1259–1288.

[67] L.S. Blackford, J. Choi, A. Cleary, E.F. D'Azevedo, J.W. Demmel, I.S. Dhillon, J.J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D.W. Walker, R.C. Whaley, ScaLAPACK Users' Guide, Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[68] J. Choi, J.J. Dongarra, S. Ostrouchov, A. Petitet, D.W. Walker, R.C. Whaley, The design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines, Scientific Programming 5 (1996) 173–184.