mpC + ScaLAPACK = Efficient Solving Linear Algebra Problems on Heterogeneous Networks

Alexey Kalinov and Alexey Lastovetsky

Institute for System Programming, Russian Academy of Sciences
25, Bolshaya Kommunisticheskaya str., Moscow 109004, Russia
{ka,lastov}@ispras.ru

Abstract. The paper presents experience of using mpC for accelerating ScaLAPACK applications on heterogeneous networks of computers. The mpC is a language, specially designed for parallel programming for heterogeneous networks. It has facilities for distribution of participating processes over processors in accordance with performances of the latters. An mpC application carring out Cholesky factorization on a heterogeneous network of workstations is used to demonstrate that the heterogeneous process distribution has an essential advantage over the traditional homogeneous distribution. The application is implemented using calls to ScaLAPACK routines by means of the interface mpC - ScaLAPACK.

1 Introduction

ScaLAPACK [1] is the most famous library for solving linear algebra problems on distributed-memory, concurrent computers. The main target platforms for ScaLAPACK are distributed-memory supercomputers consisting of identical processors, because present high-performance scientific computations concentrate mostly on them.

On the other hand, progress in network technologies is making networks of computers (in particular, networks of PCs and workstations) more and more attractive for high-performance parallel computing. The main difference between supercomputers and networks is heterogeneity of the latters.

The heterogeneity is displayed at least in two forms. Firstly, in the form of heterogeneity of machine arithmetics of such parallel systems. Related challenges existing in writing reliable numerical library software for heterogeneous computing environments have been analyzed in [2].

Secondly, in the form of heterogeneity of hardware performance of individual processors. ScaLAPACK has been developed and tested with one process per processor running [3]. We will refer to that processes distribution as homogeneous. Let see what happens when a parallel linear algebra application, that provides good distribution of computations and communications when running one process per processor in homogeneous environments, runs with one process per processor on a heterogeneous network of computers. Since volumes of computations executed by different processors are approximately equal to each other, more powerful processors will wait for the slowest one at synchronization points. Therefore, the total time of computations will be determined by the time elapsed on the slowest processor.

We have done an experiment corroborating that statement. We considered two local subnetworks of our local network: homogeneous `abcd` consisting of four SUN workstations `a`, `b`, `c`, and `d`, and heterogeneous `aEFG` consisting of four SUN workstations `a`, `E`, `F`, and `G`. Workstation `a` belongs to the both networks and other workstations of the heterogeneous network are more powerful then workstation `a`. Performances of the workstations were estimated by means of Cholesky factorization of a matrix of the same dimension with a sequential LAPACK [4] Cholesky factorization routine `dpotf2`. The total power of the heterogeneous subnetwork is about 1.9 times greater then that of the homogeneous one. It could be expected that, for example, the parallel ScaLAPACK Cholesky solver [3] would be executed on the heterogeneous subnetwork about 1.9 times faster then on the homogeneous one. But the real situation, shown in figure 1, turned out quite different.

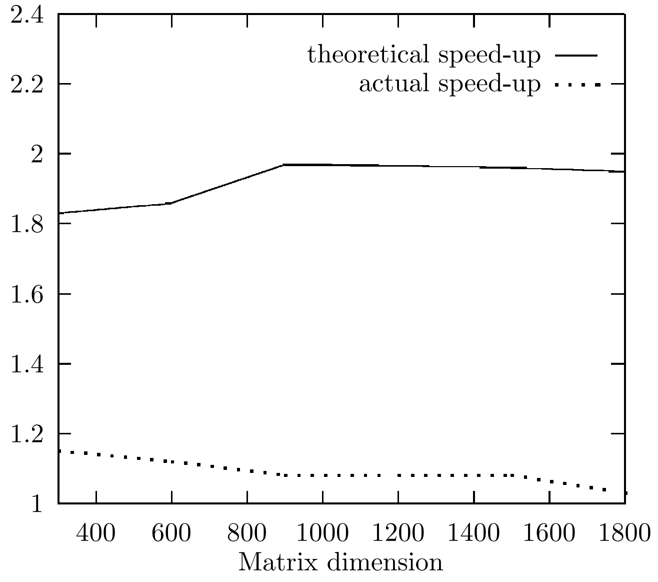

Fig. 1. Speed-up achieved by ScaLAPACK Cholesky solver on the heterogeneous network relative to the homogeneous one. The both networks consist of four workstations. One workstation of the heterogeneous network belongs to the homogeneous one. Other workstations of the heterogeneous network are more powerful. Drawn line represents increase in computing power of the heterogeneous network or theoretical speed-up. Dotted line represents actual speed-up.

A natural solution of this problem is heterogeneous distribution of processes of the parallel program over the processors, taking into account at least differences in performances of processors. The distribution may be done by means of configurational files. But it is a difficult task, and if the application topology is defined at run time (for example, process grid parameters depends on input data), this approach will not work.

An alternative approach is to write such applications that do not need a sophisticated process mapping to start up the applications efficiently. Designed specially to write efficient and portable parallel applications for heterogeneous networks of computers, the mpC language [5] allows to do that. This language is an ANSI C superset allowing to write applications adapting to differences in performances of both processors and communication links of any particular executing network. The basic idea is that an mpC application explicitly builds at run time an abstract heterogeneous computing network and distributes data, computations and communications over the network. The abstract network consists of virtual processors of different performances and different links. The mpC programming system uses this information at run time to map the abstract network to any real executing network of computers in such a way that ensures efficient running of the application on the real network. More about mpC as well as the mpC free software can be found at http://www.ispras.ru/˜mpc.

In this paper, we consider only the heterogeneity of processor performances. We propose a heterogeneous distribution of processes over processors when the number of processes involved in computations on a separate processor depends on its performance. We investigate the distribution using a typical linear algebra problem - the Cholesky factorization of square dense matrices. The heterogeneous distribution of the involved processes is performed by an mpC program, while the latter calls a parallel ScaLAPACK solver to perform the parallel Cholesky factorization proper.

Section 2 shortly introduces the mpC language and describes an implementation of the heterogeneous distribution in mpC with calls to ScaLAPACK functions as well as interface mpC - ScaLAPACK. Section 3 gives experimental results of the Cholesky factorization on a network of heterogeneous workstations using the homogeneous and heterogeneous processes distribution.

2 Implementation of Heterogeneous Distribution of Processes over Processors in mpC

The language mpC [5] is a parallel language that allows an efficiently-portable modular programming heterogeneous networks of computers. It provides facilities for specification of requirements on resources, necessary for efficient execution of parallel application, and the mpC programming system tries to satisfy the requirements taking into account peculiarities of any particular heterogeneous network of computers.

The mpC language is an ANSI C superset that introduces a new kind of managed resource, the *computing space*, defined as a set of virtual processors of difference performances. At run time, the virtual processors are represented by actual processes of the particular running parallel application. The programmer manages the computing space by means of creating and discarding regions of the computing space, named *network objects*, just like he manages storage creating and discarding data objects (regions of storage). At any moment of program execution, just a set of defined network objects represents the abstract computing network.

The following mpC function HeHo implements the heterogeneous distribution of processes involved in computations and calls ScaLAPACK to perform parallel Cholesky factorization properly.

```
/*1 */ #define N 100
/*2 */ nettype Grid(nr,nc) {
/*3 */   coord I=nr, J=nc;
/*4 */ };
/*5 */ int [net Grid(nr,nc) v] mpC2Cblacs_gridinit(int *, char *);
/*6 */ void [*]HeHo(repl int P, repl int Q) {
/*7 */   {
/*8 */    int n=N,info;
/*9 */    double a[N][N];
/*10*/    init(a);
/*11*/    recon dpotf2_("U",&n,a,&n,&info);
/*12*/   }
/*13*/   {
/*14*/     net Grid(P,Q) w;
/*15*/     [w]: {
/*16*/       int ConTxt;
/*17*/       ([(P,Q)w])mpC2Cblacs_gridinit(&ConTxt,"R");
/*18*/       pdlltdriver1_(&ConTxt);
/*19*/       mpC2Cblacs_gridexit(ConTxt);
/*20*/     }
/*21*/   }
/*22*/ }
```

The heterogeneous strategy is implemented in three steps:

1. Performances of real processors for a relevant benchmark, Cholesky factorization by means of the LAPACK routine dpotf2, are determined in lines 7-12 with the help of statement recon. The statement (line 11) updates at run time the information about processor performances of the executing real network by means of execution of the corresponding computations as a benchmark (in our case, it is a call to function dpotf2). It is supposed that performances estimated when matrix dimension is 100 are not essentialy different from that estimated when matrix dimension is different. Our experiments confirm the supposition.
2. The network object w, executing the corresponding computations, is defined in line 14 (see also lines 2-4) as consisting of $P \cdot Q$ virtual processors of the

same performance (by default). Its parent, the virtual host-processor, has coordinates `I=0`, `J=0` (by default). At run time, that definition of `w` leads to such a mapping of its virtual processors into processes of the running parallel program that the number of processes involved in computations on a separate real processor depends on its performance. The algorithm of the mapping is presented in [7].

3. A slightly modified version of the ScaLAPACK test driver for Cholesky factorization is called on the network object `w` (lines 15-20). This driver reads from a file problem parameters (matrix and block sizes), forms a test matrix and performs its Cholesky factorization. The only parameter of the driver is a context which is a ScaLAPACK analog of an mpC network object. The context is created by means of a call to function `mpC2Cblacs_gridinit` in line 17. This call creates the ScaLAPACK context `ConTxt` associated with the process grid in which network object `w` has been mapped. More details of the interface mpC - ScaLAPACK are described below in 2.1. A call to function `mpC2Cblacs_gridexit` in line 19 releases resources, allocated on creation of context `ConTxt`.

In mpC, there exist three kinds of functions: basic, network, and nodal.

Basic functions are called and executed on the entire computing space. Network objects can be defined only in basic functions. Function `HeHo` is a basic function, what is specified with construct `[*]` in line 6 placed just before the name of the function.

A network function is called and executed on a network object. Function `mpC2Cblacs_gridinit`, declared in line 5, is an example of a network function. It has 3 special formal parameters, `v,nr,nc`. Parameter `v` corresponds to a network object on which the function is executed. Parameters `nr,nc` is regarded as integer variables replicated over network object `v`. Network object `v` is of network type `Grid(nr,nc)`. In line 17, this function is called with network object `w` and parameters `P,Q` of its type as arguments corresponding to the above formal parameters.

A nodal function can be executed on any separate virtual processor. In mpC, all C functions are considered nodal.

If declared without any special distribution specifier, a variable, declared in a basic function, is considered distributed over the entire computing space, while a variable, declared in a network function, is considered distributed over the corresponding network object. Distribution specifier `[w]` in line 15 specifies the network object executing the compound statement in lines 15-20.

2.1 Interface mpC - ScaLAPACK

The BLACS (Basic Linear Algebra Communication Subprograms) [6] provide a linear algebra oriented message passing interface that may be implemented efficiently and uniformly across a large range of distributed memory platforms. It is used as the communication layer of ScaLAPACK.

In the BLACS, there are two grid creation routines (`Cblacs_gridinit` and `Cblacs_gridmap`) which create a process grid and its enclosing context. These routines return context handles, which are simple integers. Subsequent BLACS routines will be passed these handles, which allow the BLACS to determine what context/grid a routine is being called from. Releasing contexts is done via the routine `Cblacs_gridexit`.

The mpC programming system allows to call parallel ScaLAPACK routines providing mpC analogs of the above BLACS routines. In particular, mpC provides the following network function

```
int [net Grid(nr,nc) v] mpC2Cblacs_gridinit(int *pConTxt, char
                                    *order);
```

as an analog of the BLACS grid creation routine

```
int Cblacs_gridinit(int *pConTxt, char *order, int nr, int nc);
```

where `pConTxt` is a pointer to the context to be created, `nr` and `nc` are numbers of rows and columns in the process grid associated with the context, and `order` indicates how to map processes to the BLACS grid. The BLACS grid and corresponding context are created by network function `mpC2Cblacs_gridinit` from the processes representing virtual processors of the network object `v`. The created context can be used to call ScaLAPACK routines, for example, routine `pdlltdriver1` in line 18.

The BLACS grid releasing routine `Cblacs_gridexit` has the mpC analog `mpC2Cblacs_gridexit`.

Currently, the mpC programming system uses MPI as a communication platform. So, the above interface works only for the BLACS implementation built on the top of MPI.

3 Experimental Results

We compared two processes distributions:

– The traditional homogeneous processes distribution (one process per processor) implemented in ScaLAPACK.
– The heterogeneous distribution of processes over processors implemented in mpC.

As a factor of the comparison, we used speed-up achieved by the heterogeneous process distribution relative to the homogeneous one when running with the same process grid parameters and block size.

The comparison was performed for the Cholesky factorization on a network of workstations. For our experiments, we used a part of a local network consisting of 8 uniprocessor Sun workstations of different performances interconnected via 10 Mbits Ethernet. MPICH 1.0.13 was used as a particular communication platform. All workstations executed the same copy of code. Performances of the workstations, obtained by means of execution of the LAPACK routine `dpotf2` performing serial Cholesky factorization, is the following: `a` - 200, `b` - 200, `c` - 200, `d` - 200, `e` - 267, `f` - 267, `G` - 801, `h` - 100. Only two workstations `G` (the fastest one) and `h` (the slowest one) have performances essentialy different from others.

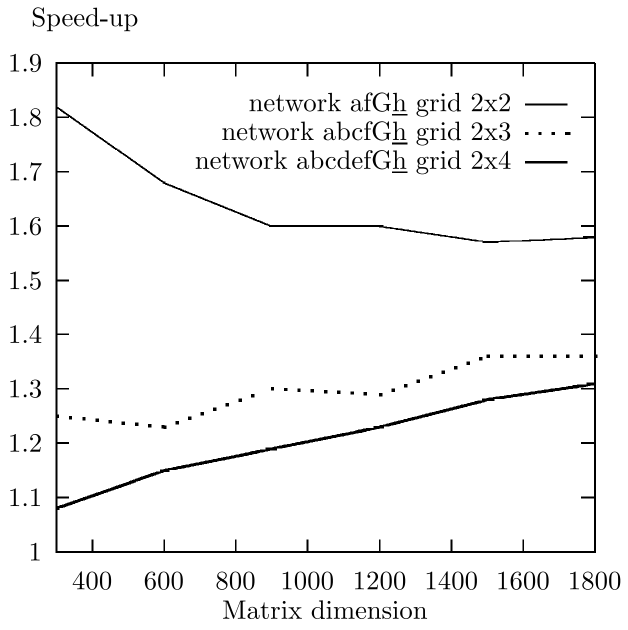

Fig. 2. Speed-up achieved by the heterogeneous processes distribution relative to the homogeneous one on networks `afGh` consisting of 4 workstations `a`, `f`, `G`, and `h`, `abcfGh` consisting of 6 workstations `a`, `b`, `c`, `f`, `G`, `h`, `abcdefGh` consisting of 8 workstations `a`, `b`, `c`, `d`, `e`, `f`, `G`, `h`.

Figure 2 presents the speed-up achieved by the heterogeneous processes distribution when running on the following networks: `afGh` consisting of 4 workstations `a`, `f`, `G`, and `h`) (process grid 2x2), `abcfGh` consisting of 6 workstations `a`, `b`, `c`, `f`, `G`, and `h` (process grid 2x3), and `abcdefGh` consisting of 8 workstations `a`, `b`, `c`, `d`, `e`, `f`, `G`, and `h` (process grid 2x4). It is interesting that in all cases the mpC run-time system distributes the involved processes with two processes running on the most powerful workstation `G` and no processes running on the slowest workstation `h`.

4 Conclusion

Numerical software developed for computations in homogeneous environments does not allow to utilize all performance potential of heterogeneous networks.

It has been demonstrated that in this case a heterogeneous network behaves as a homogeneous network obtained from the heterogeneous one by means of replacing all its processors with the slowest processor.

A natural way to answer this challenge is to develop dedicated numerical software aimed at heterogeneous environments. Such software should at least take into account the heterogeneity of processor performances. The paper presents a way to do that via adapting legacy numerical software .

The mpC parallel language is just aimed at portable and efficient programming for heterogeneous environments. It has facilities for heterogeneous distribution of the processes involved in computations over processors. ScaLAPACK is generally accepted numerical linear algebra package. Call to ScaLAPACK routines from mpC program gives a new facilities for efficiently solving linear algebra problems on heterogeneous environments. The mpC program can perform adaptation to pecularities of the particular network and ScaLAPACK routines can do calculations proper.

5 Acknowledgments

We would like to thank Jack Dongarra who gave us the idea of the research presented in this paper.

References

[1] 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: A Linear Algebra Library for Message-Passing Computers" SIAM Conference on Parallel Processing, March 1997.

[2] L.S.Blackford, A.Cleary, J.Demmel, I.Dhillon, J.Dongarra, S.Hammarling, A.Petitet, H.Ren, K.Stanley, and R.C.Whaley, Practical Experience in the Dangers of Heterogeneous Computing UT, CS-96-330, July 1996.

[3] J.Choi, J.J.Dongarra, S.Ostrouchov, A.P.Petitet, D.W.Walker, and R.C.Whaley "The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines" UT, CS-94-246, September, 1994.

[4] E.Anderson, Z.Bai, C.Bischof, J.Demmel, J.Dongarra, J. Du Croz, A.Greenbaum, S.Hammarling, S.McKenney, S.Octrouchov, and D.Sorensen, "LAPACK Users' Guide, Second Edition", SIAM, Philadelphia, PA, 1995.

[5] A.Lastovetsky, The mpC Programming Language Specification. Technical Report, ISPRAS, Moscow, December 1994.

[6] R. Clint Whaley, "Basic Linear Algebra Communication Subprograms: Analysis and Implementation Across Multiple Parallel Architectures", Tech.Rep. LAPACK Working Note 73, University of Tennesee, TN, 1994.

[7] D.Arapov, A.Kalinov, and A.Lastovetsky, Resource management in the mpC Programming Environment, *in* "Proceedings of the 30th Hawaii International Conference on System Sciences (HICSS'30)", IEEE Computer Society, Maui, HI, January 1997.