Scalable Computing: Practice and Experience
Volume 10, Number 2, pp. 201–216. http://www.scpe.org
ISSN 1895-1767

HETEROPBLAS: A SET OF PARALLEL BASIC LINEAR ALGEBRA SUBPROGRAMS OPTIMIZED FOR HETEROGENEOUS COMPUTATIONAL CLUSTERS*

RAVI REDDY† ALEXEY LASTOVETSKY†, AND PEDRO ALONSO‡

Abstract. This paper presents a software library, called Heterogeneous PBLAS (HeteroPBLAS), which provides optimized parallel basic linear algebra subprograms for Heterogeneous Computational Clusters. This library is written on the top of HeteroMPI and PBLAS whose building blocks, the de facto standard kernels for matrix and vector operations (BLAS) and message passing communication (BLACS), are optimized for heterogeneous computational clusters. This is the first step towards the development of a parallel linear algebra package for Heterogeneous Computational Clusters.

We show that the efficiency of the parallel routines is due to the most important feature of the library, which is the automation of the difficult optimization tasks of parallel programming on heterogeneous computing clusters. They are the determination of the accurate values of the platform parameters such as the speeds of the processors and the latencies and bandwidths of the communication links connecting different pairs of processors, the optimal values of the algorithmic parameters such as the data distribution blocking factor, the total number of processes, the 2D process grid arrangement, and the efficient mapping of the processes executing the parallel algorithm to the executing nodes of the heterogeneous computing cluster.

We present the user interface and the software hierarchy of the first research implementation of HeteroPBLAS. We demonstrate the efficiency of the HeteroPBLAS programs on a homogeneous computing cluster and a heterogeneous computing cluster.

Key words: parallel linear algebra, ScaLAPACK, HeteroMPI, heterogeneous platforms

1. Introduction. This paper presents a software library, called Heterogeneous PBLAS (HeteroPBLAS), which provides optimized parallel basic linear algebra subprograms (PBLAS) for Heterogeneous Computational Clusters. This library is written on top of HeteroMPI [1], and PBLAS [2] whose building blocks, the de facto standard kernels for matrix and vector operations (BLAS [3]) and message passing communication (BLACS [4]), are optimized for heterogeneous computational clusters.

We start with presentation of related works, which have produced solely heterogeneous parallel algorithms/strategies. There are two strategies of distribution of computations that can be used to implement PBLAS for Heterogeneous Computing Clusters (HCCs) [5]:

1. HeHo: heterogeneous distribution of processes over processors and homogeneous block distribution of data over the processes. This was implemented in mpC language [6, 7] with calls to ScaLAPACK [8];
2. HoHe: homogeneous distribution of processes over processors with each process running on a separate processor and heterogeneous block cyclic distribution of data over the processes. This was implemented in mpC language with calls to BLAS and LAPACK [9].

The HeHo strategy is a multiprocessing approach that is commonly used to accelerate ScaLAPACK programs on HCCs. The strategies are compared using Cholesky factorization on a HCC. The authors [5] show that for HCCs, the strategies HeHo and HoHe are more efficient than the traditional homogeneous strategy HoHo (homogeneous distribution of processes over processors with each process running on a separate processor and homogeneous block cyclic distribution of data over the processes implemented in ScaLAPACK). The main disadvantage of the HoHe strategy is non Cartesian nature of the data distribution, which is the distribution where each processor has to communicate with more than four direct neighbors. This leads to additional communications that can be crippling in the case of large networks. Moreover, the non Cartesian nature leads to poor scalability of the linear algebra algorithms that are proven scalable in the case of Cartesian data distribution and networks allowing parallel communications.

Beaumont et al. [10] present a proposal motivating provision of PBLAS in the form of a library for HCCs. The authors discuss HoHe strategies to implement matrix products and dense linear system solvers for HCCs and present them as a basis for a successful extension of the ScaLAPACK library to HCCs. They show that extending the standard ScaLAPACK block cyclic distribution to heterogeneous 2D grids is a difficult task. However, providing this extension is only the first step. The ScaLAPACK software must then be completely redesigned and rewritten, which is not a trivial effort.

*The research was supported by Science Foundation of Ireland (SFI).

†School of Computer Science and Informatics, University College Dublin ({manumachu.reddy, alexey.lastovetsky}@ucd.ie).

‡Department of Information Systems and Computation, Polytechnic University of Valencia (palonso@dsic.upv.es). Partially supported by Vicerrectorado de Investigación, Desarrollo e Innovación de la Universidad Politécnica de Valencia, and Generalitat Valenciana.

The multiprocessing strategy (HeHo), however, is easier to accomplish. It allows the complete reuse of high quality software such as ScaLAPACK on HCCs with no redesign efforts and provides good speedups. It can be summarized as follows:

1. The whole computation is partitioned into a large number of equal chunks;
2. Each chunk is performed by a separate process;
3. The number of processes run by each processor is proportional to its speed.

Thus, while distributed evenly across parallel processes, data and computations are distributed unevenly over processors of the HCC so that each processor performs the volume of computations proportional to its speed.

There have been several research contributions illuminating the merits of this strategy. Kishimoto and Ichikawa [11] use it to estimate the best processing element (PE) configuration and process allocation based on an execution time model of the application. The execution time is modeled from the measurement results of various configurations. Then, a derived model is used to estimate the optimal PE configuration and process allocation. Kalinov and Klimov [12] investigate the HeHo strategy where the performance of the processor is given as a function of the number of processes running on the processor and the amount of data distributed to the processor. They present an algorithm that computes optimal number of processes and their distribution over processors minimizing the execution time of the application. Cuenca et al. [13] analyze automatic optimization techniques in the design of parallel dynamic programming algorithms on heterogeneous platforms. The main idea is to determine automatically the optimal values of a number of algorithmic parameters such as number of processes, number of processors, and number of processes per processor.

Therefore, there has been a proliferation of research efforts presenting approaches to implement parallel solvers for HCCs but scientific software based on these approaches is virtually nonexistent. However, one can conclude from these research efforts that any software aspiring to provide automatically tuned parallel linear algebra programs for HCCs must automate the following essential and complicated tasks, which are also described in [14]:

1. Determination of the accurate values of platform parameters such as speeds of the processors, latencies and bandwidths of the communication links connecting different pairs of processors. These parameters compose the performance model of the executing heterogeneous platform;
2. Determination of the optimal values of algorithmic parameters such as the data distribution blocking factor, the total number of processes, and the 2D process grid arrangement to be used during the execution of the linear algebra kernel, and
3. Efficient mapping of the processes executing the parallel algorithm to the executing nodes of the HCC.

In this paper, we present a software library, called Heterogeneous PBLAS (HeteroPBLAS), which provides optimized PBLAS for HCCs. The design of the software library adopts the multiprocessing approach and thus reuses the PBLAS software (a component of ScaLAPACK) completely. The library also performs the automations of the aforementioned tedious and error prone tasks. This can be seen as the first step towards the development of a parallel linear algebra package for HCCs.

The rest of the paper is organized as follows. We start with an overview of the software in the HeteroPBLAS package. We describe the different software components and building blocks of the first research implementation as well as the user interface. In §3, we present the performance model of one of the PBLAS routines. The writing of the performance models formed the core development activity of the project. In §4, we explain how the optimal values of the algorithmic parameters are determined. In §5, we present the experimental results of execution of HeteroPBLAS programs on a homogeneous computing cluster and a heterogeneous computing cluster. We conclude the paper by outlining our future research goals.

2. Overview of Heterogeneous PBLAS. The flowchart of the main routines of HeteroPBLAS software library is shown in Fig. 2.1. The high level building blocks of HeteroPBLAS are HeteroMPI and PBLAS. The Parallel BLAS (PBLAS) is a parallel set of BLAS, which perform message passing and whose interface is as similar to BLAS as possible. The design goal of PBLAS was to provide specifications of distributed kernels, which would simplify and encourage the development of high performance and portable parallel numerical software, as well as providing manufacturers with a small set of routines to be optimized. These subprograms were used to develop parallel libraries such as ScaLAPACK, which is a well known standard package providing high performance linear algebra routines for distributed memory message passing MIMD computers supporting PVM and/or MPI.

ScaLAPACK uses block partitioned algorithms to ensure good performance on MIMD distributed memory concurrent computers, by minimizing the frequency of data movement between different levels of the memory hierarchy. The most fundamental building blocks of ScaLAPACK are the sequential BLAS, in particular the Level 2 and 3 BLAS, and the BLACS, which perform common matrix oriented communication tasks. The PBLAS are used as the highest level building blocks for implementing the ScaLAPACK library and provide the same ease of use and portability for ScaLAPACK that the BLAS provide for LAPACK.

HeteroMPI is an extension of MPI for programming high performance computations on HCCs. The main idea of HeteroMPI is to automate the process of selection of a group of processes, which would execute the parallel algorithm faster than any other group.

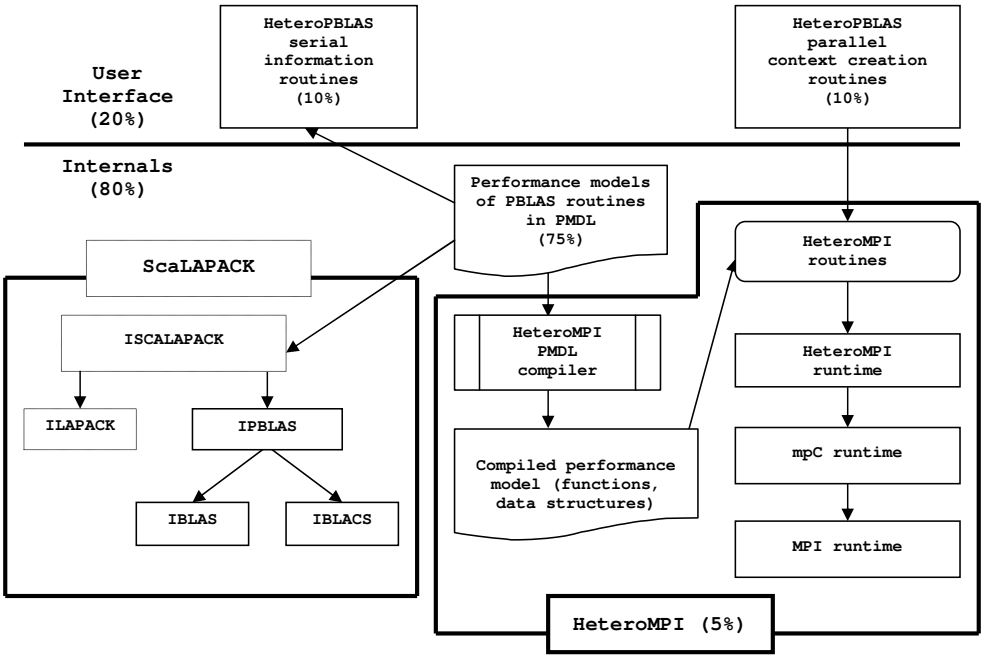


FIG. 2.1. *Flow of the HeteroPBLAS context creation routine call. The percentages give the breakup of HeteroPBLAS development efforts.*

The first step in this process of automation is the writing/specification of the performance model of the parallel algorithm. Performance model is a tool supplied to the programmer to specify his high level knowledge of the application that can assist in finding the most efficient implementation on HCCs. This model allows description of all the main features of the underlying parallel algorithm that affect the execution performance of parallel applications on HCCs. These features are:

1. The total number of processes executing the algorithm;
2. The total volume of computations to be performed by each of the processes in the group during the execution of the algorithm;
3. The total volume of data to be transferred between each pair of processes in the group during the execution of the algorithm;
4. The order of execution of the computations and communications by the parallel processes in the group, that is, how exactly the processes interact during the execution of the algorithm.

HeteroMPI provides a dedicated performance model definition language (PMDL) for writing this performance model. The model and the PMDL are borrowed from the mpC programming language. The PMDL compiler compiles the performance model written in PMDL to generate a set of functions, which make up the algorithm specific part of the HeteroMPI runtime system. These functions are called by the mapping algorithms of HeteroMPI runtime system to estimate the execution time of the parallel algorithm.

A typical HeteroMPI application contains HeteroMPI group management function calls to create and destroy a HeteroMPI group, and the execution of the computations and communications involved in the execution of the parallel algorithm employed in the application by the members of the group. The principal group constructor

routine `HMPI_Group_auto_create` determines the optimal number of processes and the optimal process grid arrangement, thereby relieving the application programmers from having to specify these parameters during the execution of the parallel application.

During the creation of a group of processes, the HeteroMPI runtime system solves the problem of selection of the optimal set of processes running on different computers of the HCC. It should be noted that this problem of mapping, in general, is NP-complete. The mapping algorithms used to solve the problem of selection of processes are explained in [1, 7]. The solution is based on the following:

1. The performance model of the parallel algorithm in the form of the set of functions generated by the compiler from the compilation of the performance model written in PMDL;
2. The performance model of the executing network of computers, which reflects the state of this network just before the execution of the parallel algorithm. This model considers the executing heterogeneous network as a multilevel hierarchy of interconnected sets of heterogeneous multiprocessors [7]. This model takes into account the material nature of communication links and their heterogeneity.

The principal routines in HeteroPBLAS software library are the context creation functions for the PBLAS routines. There is a context creation function for each PBLAS routine. It provides a context for the execution of the PBLAS routine but most importantly, performs the critical work of automating the difficult optimization tasks.

All the context creation routines have names of the form `hscal_pxyyzz_ctxt`. The second letter, `x`, indicates the data type. For example, `d` would mean double precision real data. The next two letters, `yy`, indicate the type of matrix (or of the most significant matrix). For example, `ge` would represent a general matrix. The last three letters `zz` indicate the computation performed. For example, the context creation function for the PDGEMM routine has an interface, which is shown below:

```
int hscal_pdgemm_ctxt(char* transa, char* transb,
    int * m, int * n, int * k, double * alpha, int * ia, int * ja,
    int * desca, int * ib, int * jb, int * descb, double * beta,
    int * ic, int * jc, int * descc, int * ictxt)
```

This function call returns a handle to a HeteroMPI group of MPI processes in `ictxt` and a return value of `HSCAL_SUCCESS` on successful execution. It differs from the PBLAS PDGEMM call in the following ways:

1. It returns a context but does not actually execute the PDGEMM routine;
2. The input arguments are the same as for the PDGEMM call except

(i) The matrices A, B and C containing the data are not passed as arguments;

3. The output arguments differ as follows:

(i) An extra return argument, `ictxt`, which contains the handle to a group of MPI processes that is subsequently used in the actual execution of the PDGEMM routine;

(ii) A return value of `HSCAL_SUCCESS` indicating successful execution or otherwise an appropriate error code.

The function call is a collective operation and must be called by all the processes running in the HeteroPBLAS program. The context contains a handle to a HeteroMPI group of MPI processes, which tries to execute the PBLAS routine faster than any other group of processes. This context can be reused in multiple calls of the same routine or any routine that uses similar parallel algorithm as PDGEMM. The reader is referred to the HeteroPBLAS programmer's manual for more details of the user interface.

The HeteroPBLAS context creation/destruction routines call interface functions of HeteroMPI runtime system (the main routines being `HMPI_Recon`, `HMPI_Timeof`, `HMPI_Group_auto_create`). The HeteroPBLAS information functions calculate the total number of computations (arithmetical operations) performed by each process and the total number of communications in bytes between a pair of processes involved in the execution of the PBLAS routine. These routines are serial and can be called by any process. They do not actually execute the corresponding PBLAS routine but just calculate the total number of computations and communications involved. The block IPBLAS ('I' standing for instrumented) represents the instrumented code of PBLAS, which reuses the existing code base of PBLAS completely. The instrumentations made to it are (a) Wrapping of the parallel regions of the code in mpC par loops recognized by the PMDL compiler and (b) Replacement of the serial BLAS computation routines and the BLACS communication routines by calls to information functions, which return the number of arithmetical operations performed by each process and number of communications in bytes between different pairs of processes respectively.

The first step in the implementation of the context creation routine for a HeteroPBLAS routine is the writing of its performance model using the PMDL. The performance model definitions contain the instrumented code components. The HeteroMPI compiler compiles this performance model to generate a set of functions. During the creation of the context, the mapping algorithms of HeteroMPI runtime system call these functions to estimate the execution time of the PBLAS routine.

The writing of performance models of all the PBLAS routines has been the most laborious and intricate effort of this project. The key design issues were (a) *accuracy*, to facilitate accurate prediction of the execution time of the PBLAS routine, (b) *efficiency*, to execute the performance model in reasonable execution time, (c) *reusability*, as these performance models are to be used as building blocks for the performance models of ScaLAPACK routines, and (d) *preservability*, to preserve the key design features of underlying ScaLAPACK package. In the next section, we present one such performance model that demonstrates the tedium and accuracy of the writing process.

3. Performance model of PBLAS PDGEMM. The performance model definition of PDGEMM routine shown in Fig. 3.1 is used to demonstrate the complexity of the effort of writing a performance model. It describes the simplest case of parallel matrix–matrix multiplication of two dense square matrices A and B of size `n*n`. The reader is referred to [1, 7] for more details of the main constructs, namely `coord`, `parent`, `node`, `link`, and `scheme`, used in a description of a performance model. This definition is an extensively stripped down version of the actual definition, which can be studied from the package. The data distribution blocking factor `b` is assumed equal to the algorithmic blocking factor. Another simplification is that the matrices are divided such that `(n%(b*p))` and `(n%(b*q))` (see explanation of variables below) are both equal to zero.

```
/* 1 */ algorithm pdgemm(int n, int b, int bp, int bq, int p, int q) {
/* 2 */   coord I=p, J=q;
/* 3 */   node {I>=0 && J>=0:
/* 4 */         bench*((n/b)*((n/p)*(n/q)*(b))/((n/bp)*(n/bq)*(b)));};
/* 5 */   link (K=p, L=q)
/* 6 */   {
/* 7 */      I>=0 && J>=0 && J!=L:
/* 8 */        length*((n/p)*(n/q)*sizeof(double))
/* 9 */              [I, J]->[I, L];
/* 10 */     I>=0 && J>=0 && I!=K:
/* 11 */       length*((n/p)*(n/q)*sizeof(double))
/* 12 */             [I, J]->[K, J];
/* 13 */   };
/* 14 */   parent[0,0];
/* 15 */   scheme
/* 16 */   {
/* 17 */     int i, j, k;
/* 18 */     for(k = 0; k < n; k+=b)
/* 19 */     {
/* 20 */       par(i = 0; i < p; i++)
/* 21 */          par(j = 0; j < q; j++)
/* 22 */             if (j != ((k/b)%q))
/* 23 */               (100.0/(n/q)) %% [i,((k/b)%q)]->[i,j];
/* 24 */       par(i = 0; i < p; i++)
/* 25 */          par(j = 0; j < q; j++)
/* 26 */             if (i != ((k/b)%p))
/* 27 */               (100.0/(n/p)) %% [((k/b)%p),j]->[i,j];
/* 28 */       par(i = 0; i < p; i++)
/* 29 */         par(j = 0; j < q; j++)
/* 30 */           ((100.*(b/n)) %% [i,j];
/* 31 */     }
/* 32 */   };
/* 33 */ };
```

FIG. 3.1. *The performance model of the PDGEMM routine written in the performance model definition language.*

Line 1 is a header of the performance model declaration. It introduces the name of the performance model `pdgemm` parameterized with the scalar integer parameters `n`, `b`, `bp`, `bq`, `p`, and `q`. Parameter `n` is the size of square matrices A, B, and C. Parameter `b` is the size of the data distribution blocking factor. Parameters `bp` and `bq` are used in the execution of the benchmark code, which performs a matrix update of a matrix C_b of size `(n/bp)*(n/bq)` using A_b of size `(n/bp)*(b)` and B_b of size `(b)*(n/bq)`. The value of the parameter `bp` is the square root of the total number of processes available for computation. The value of parameter `bq` is equal to total number of processes divided by `bp`. Parameters `p` and `q` represent the number of processes along the row and the column in the process grid arrangement.

Line 3 is a *coordinate declaration* declaring the 2D coordinate system to which the processor nodes of the network are related. Line 4 is a *node declaration*. It associates the processes with this coordinate system to form a `p\times q` grid. It specifies the (absolute) volume of computations performed by each of the processes. The statement `bench` just specifies that as a unit of measurement, the volume of computation performed by some benchmark code be used. The benchmark code solves the core computational task, which is the matrix update representing the largest share of the computations performed by the processes at each step of the PBLAS routine. In other words, it is the costliest task performed by a process at each step of the PBLAS routine. In this case, the benchmark code used for estimation of speeds of processes performs a local DGEMM update of two dense `(n/bp)*b` and `b*(n/bq)` matrices where `b` is the optimal data distribution factor determined by the HeteroPBLAS runtime system (to follow in the next section). The total volume of computations performed by each process in the benchmark code is `(n/bp)*(n/bq)*b`. The total volume of computation performed by each process P_{IJ} at each step of the parallel algorithm is `(n/p)*(n/q)*b`. Since there are `n/b` steps in the parallel algorithm, the total volume would be `((n/b)*(n/p)*(n/q)*(b))`. The line 4 of node declaration describes the ratio of this total volume to the total volume of computations involved in the execution of the benchmark code. Lines 5–13 represent a link declaration. This specifies the links between the processes, the pattern of communication among the processes, and the total volume of data to be transferred between each pair of processes during the execution of the algorithm. Lines 7–9 of the link declaration describe horizontal communications related to matrix A. Obviously, processes from the same column of the process grid do not send each other elements of matrix A. Only processes from the same row of the process grid send each other elements of matrix A. Process P_{IJ} will send `(n/p)*(n/q)` number of elements of matrix A to process P_{IL}. So the total volume of data transferred (in bytes) from process P_{IJ} to process P_{IL} will be given by `(n/p)*(n/q)*sizeof(double)`.

Lines 10–13 of the link declaration describe vertical communications related to matrix B. Obviously, only processes from the same column of the process grid send each other elements of matrix B. In particular, process P_{IJ} will send all its elements of matrix B to all other processes from column J of the processor grid. That is, process P_{IJ} will send `(n/p)*(n/q)` number of elements of matrix B to process P_{KJ}. So the total volume of data transferred (in bytes) from process P_{IJ} to processor P_{KJ} will be given by `(n/p)*(n/q)*sizeof(double)`.

Line 15 introduces the *scheme declaration*. The `scheme` block describes how exactly processes interact during the execution of the algorithm. The scheme block is composed mainly of two types of units. They are computation and communication units. Each computation unit is of the form $e\%\%[i]$ specifying that e percent of the total volume of computations is performed by the process with the coordinates (i). Each communication unit is of the form $e\%\%[i[\rightarrow [j]$ specifying transfer of data from process with coordinates i to the process with coordinates j. There are two types of algorithmic patterns in the scheme declaration, which are sequential and parallel. The parallel algorithmic patterns are specified by the keyword `par` and they describe parallel execution of some actions (mixtures of computations and communications). The scheme declaration describes `(n/b)` successive steps of the algorithm. At each step `k`,

1. Lines 20–23 describe horizontal communications related to matrix A. A column of `b*b` blocks of matrix A is communicated horizontally. The number of elements transferred from P_{IJ} to P_{IL} at this step will be `(n/p)`. The total number of elements of matrix A, which process P_{IJ} sends to process P_{IL}, during the execution of the algorithm is `(n/p)*(n/q)`. Therefore, `(n/p)/((n/p)*(n/q))*100=(100./(n/q))` percent of all data that should be sent from process P_{IJ} to process P_{IL} will be sent at the step. The nested `par` statement in the main for loop of the `scheme` declaration specifies this fact. Again, we use the `par` algorithmic patterns in this specification to stress that during the execution of this communication, data transfer between different pairs of processes is carried out in parallel;
2. Lines 24–27 describe vertical communications related to matrix B. A row of `b*b` blocks of matrix B is communicated vertically. The number of elements transferred from P_{IJ} to P_{KJ} at this step will be `(n/q)`.

The total number of elements of matrix B, which process P_{IJ} sends to process P_{KJ}, during the execution of the algorithm is `(n/p)*(n/q)`. Therefore, `(n/q)/((n/p)*(n/q))*100=(100./(n/p))` percent of all data that should be sent from process P_{IJ} to process P_{KJ} will be sent at the step. The nested `par` statement in the main `for` loop of the `scheme` declaration specifies this fact. Again, we use the `par` algorithmic patterns in this specification to stress that during the execution of this communication, data transfer between different pairs of processes is carried out in parallel;

3. Lines 28–30 describe computations. Each process updates each of its `b*b` block of matrix C with one block from the pivot column and one block from the pivot row. At each of `(n/b)` steps of the algorithm, each process will perform `(100.*b/n)` percent of the volume of computations it performs during the execution of the algorithm. The third nested par statement in the main for loop of the scheme declaration just specifies this fact. The par algorithmic patterns are used here to specify that all processes perform their computations in parallel.

The simplest case of PDGEMM PBLAS routine just described demonstrates the complexity of the task of writing a performance model. There are altogether 123 such performance model definitions covering all the PBLAS routines. They can be found in the HeteroPBLAS package in the directory `/PBLAS/SRC` available at the URL [15]. The performance model files start with prefix `pm_` followed by the name of the PBLAS routine and have a file extension `mpc`.

```
int main(int argc, char **argv) {
   int nprow, npcol, pdgemmctxt, myrow, mycol, c__0 = 0;
/* Problem parameters */
   char *TRANSA, *TRANSB;
   int  *M, *N, *K, *IA, *JA, *DESCA, *IB, *JB, *DESCB, *IC, *JC,
        *DESCC;
   double *ALPHA, *A, *B, *BETA, *C;
/* Initialize the heterogeneous PBLAS runtime */
   hscal_init(&argc, &argv);
/* Get the heterogeneous PDGEMM context */
   hscal_pdgemm_ctxt(TRANSA, TRANSB, M, N, K, ALPHA, IA, JA, DESCA,
                     IB, JB, DESCB, BETA, IC, JC, DESCC, &pdgemmctxt);
   if (!hscal_in_ctxt(&pdgemmctxt))
      hscal_finalize(c__0);
/* Retrieve the process grid information */
   Cblacs_gridinfo(pdgemmctxt, &nprow, &npcol, &myrow, &mycol);
/* Initialize the array descriptors for the matrices A, B and C */
   descinit_(DESCA, \dots , &pdgemmctxt);  /* for Matrix A */
   descinit_(DESCB, \dots , &pdgemmctxt);  /* for Matrix B */
   descinit_(DESCC, \dots , &pdgemmctxt);  /* for Matrix C */
/* Distribute matrices on the process grid using user-defined pdmatgen */
   pdmatgen_(&pdgemmctxt, \dots );         /* for Matrix A */
   pdmatgen_(&pdgemmctxt, \dots );         /* for Matrix B */
   pdmatgen_(&pdgemmctxt, \dots );         /* for Matrix C */
/* Call the PBLAS pdgemm routine */
   pdgemm_(TRANSA, TRANSB, M, N, K, ALPHA, A, IA, JA, DESCA, B, IB,
           JB, DESCB, BETA, C, IC, JC, DESCC);
/* Release the heterogeneous PDGEMM context */
   hscal_free_ctxt(&pdgemmctxt);
/* Finalize the Heterogeneous PBLAS runtime */
   hscal_finalize(c__0);
}
```

FIG. 4.1. *HeteroPBLAS program employing PBLAS PDGEMM.*

4. Determination of the optimal algorithmic parameters. Fig. 4.1 shows the essential steps involved in calling the PBLAS PDGEMM routine in a HeteroPBLAS program. The HeteroPBLAS runtime is initialized using the operation `hscal- _init`. The heterogeneous PDGEMM context is obtained using the context constructor routine `hscal_pdgemm_ctxt`. The function call `hscal_in_ctxt` returns a value of 1 for the processes

chosen to execute the PDGEMM routine, otherwise 0. Then the homogeneous PBLAS PDGEMM routine is executed. The heterogeneous PDGEMM context is freed using the context destructor operation `hscal_free_ctxt`. When all the computations are completed, the program is exited with a call to `hscal_finalize`, which finalizes the HeteroPBLAS runtime.

The PDGEMM context constructor routine `hscal_pdgemm_ctxt` is the main function automating the difficult optimization tasks of parallel programming on HCCs. They are the determination of the accurate values of the platform parameters such as the speeds of the processors and the latencies and bandwidths of the communication links connecting different pairs of processors, the optimal values of the algorithmic parameters such as the data distribution blocking factor, the total number of processes, the 2D process grid arrangement, and efficient mapping of the processes executing the parallel algorithm to the executing nodes of the HCC.

4.1. Data distribution blocking factor. The context constructor routine `hscal_pdgemm_ctxt` uses the HeteroMPI function `HMPI_Timeof`, as shown below, to determine the optimal value of the data distribution blocking factor.

```
double time, min_time = DBL_MAX; void *model_params;
for (b = 1; b <= n; b++) {
   // Fill the parameters to the performance model
   model_params[0] = n;  model_params[1] = b;
   model_params[2] = bp; model_params[3] = bq;
   model_params[4] = p;  model_params[5] = q;
   // Refresh the speeds of the processors
   HMPI_Recon(&hscal_dgemm_benchmark, model_params);
   if (HMPI_Is_host()) {
      time = HMPI_Timeof(&hscal_model_pdgemm, model_params);
      if (time < min_time)
         opt_b = b; min_time = time;
   }
}
```

The function `HMPI_Timeof` is used to estimate the execution time of the algorithm on the underlying hardware without its real execution. This local operation can be called by any process, which is a member of the group associated with the predefined communication universe of HeteroMPI. The parameters to this function are the handle, `hscal_model_pdgemm`, generated from the compilation of the performance model of the PDGEMM routine, and the parameters to this performance model. As one can see from the code snippet, this estimation is performed for each possible set of values to the parameters to the performance model. Using the execution time predicted for each set, the optimal value can be determined, which would be the one resulting in minimum estimated execution time.

The estimation is based on the performance model of the PDGEMM routine and the performance model of the executing network of computers, which reflects the state of this network just before the execution of the PDGEMM routine. The function `HMPI_Recon` is used to update dynamically the estimation of processor speeds at runtime. This is a collective operation and must be called by all the processes running in the HeteroPBLAS application. The performance model of the executing network of computers is summarized as follows:

1. The performance of each processor is characterized by the execution time of the same serial code (takes place during the execution of `HMPI_Recon`)

(i) The serial code is provided by the application programmer;

(ii) It is supposed that the code is representative for the computations performed during the execution of the application;

(iii) The code is performed at runtime in the points of the application specified by the application programmer. Thus, the performance model of the processors provides current estimation of their speed demonstrated on the code representative for the particular application.

In this case, the serial code, `hscal_dgemm_benchmark`, performs a local DGEMM update of `(N/bp)`×`b` and `b`×`(N/bq)` matrices where `b` is the data distribution blocking factor;

2. The communication model [7] is seen as a hierarchy of homogeneous communication layers. Each is characterized by the latency and bandwidth. Unlike the performance model of processors, the communication

model is static, a shortcoming that would be addressed in our future work. Its parameters are obtained during the initialization of the Heterogeneous ScaLAPACK runtime and are not refreshed later.

The estimation procedure is explained in detail in [7] and is summarized here. The time of execution for each mapping, $\mu : I \rightarrow C$, where I is a set of the processes of the group, and $C = \{c_0, c_1, \ldots, c_{M-1}\}$ is a set of computers of the executing network, is estimated. The estimation time for the optimal mapping, which would ensure the fastest execution of the parallel algorithm, is returned. In general, for accurate solution of this problem as many as M^K possible mappings have to be probated to find the best one (here, K is the number of processes of the group). Obviously, that computational complexity is not acceptable for a practical algorithm that should be performed at runtime. Therefore, the HeteroMPI runtime system searches for some approximate solution that can be found in some reasonable interval of time, namely, after probation of $M \times K$ possible mappings instead of M^K.

Each computation unit in the `scheme` declaration of the form $e\%\%[i]$ is estimated. Each communication unit of the form $e\%\%[i] \rightarrow [j]$ specifying transfer of data from virtual processor with coordinates i to the virtual processor with coordinates j is estimated. Simple calculation rules are used to estimate the sequential algorithmic patterns in the `scheme` declaration. For example, the estimation of the pattern

```
for (e1; e2; e3) a
```

is calculated as follows:

```
for (T=0, e1; e2; e3)
   T += time taken to execute action a
```

The rules just reflect semantics of the corresponding serial algorithmic patterns.

The rule to estimate time for a parallel algorithmic pattern

```
par (e1; e2; e3) a
```

is more complicated and is explained in detail in [7]. This is the core of the entire mapping algorithm determining its accuracy and efficiency. It takes into account material nature and heterogeneity of both processors and network equipment. It relies on fairly allocating processes to processors in a shared memory multiprocessor normally implemented by operating systems for processes of the same priority (HeteroMPI processes are just the case). At the same time, it proceeds from the pessimistic point of view when estimating workload of different processors of that multiprocessor. Estimation of communication cost by the rule is sensitive to scalability of the underlying network technology. It treats differently communication layers serializing data packages and supporting their parallel transfer. The most typical and widely used collective communication operations are treated specifically to provide better accuracy of the estimation of their execution time. An important advantage of the rule is its relative simplicity and effectiveness. The effectiveness is critical because the algorithm is supposed to be multiply executed at runtime.

This procedure to determine the optimal value of the data distribution blocking factor is now performed during the installation of the HeteroPBLAS software rather than for every call of the context constructor routine for performance reasons.

4.2. Two dimensional process grid arrangement. The optimal value of this algorithmic parameter represents all of the following: the optimal values for the total number of processes, the number of process rows, the number of process columns, and efficient mapping of the processes to the executing computers of the HCC. The context constructor routine `hscal_pdgemm_ctxt` calls the HeteroMPI function `HMPI_Group_pauto_create` to determine the optimal values. Its operation employing HeteroMPI calls is shown below:

```
int i, k, pa, p, opt_p, q, opt_q, *a, terminate = 0;
double t, mint=DBL_MAX; void *model_params = NULL;
// The host process
if (HMPI_Is_host()) {
   // The total number of processes available for computation
   int np = HMPI_Group_size(HMPI_COMM_WORLD_GROUP);
   for (k=np; k>=1; k--) {
      // The possible two dimensional process arrangements (p,q)
      HMPI_Get_2D_process_arrangements(k, &pa, &a);
      // For each two dimensional process arrangement (p,q)
      for (i=0; i<pa; i++) {
         // Fill the parameters to the performance model
         // Estimate the execution time for each process arrangement (p,q)
```

```
            t = HMPI_Timeof(&hscal_model_pdgemm, model_params);
            if (t < mint)
               // A better process arrangement found, continue the algorithm
               terminate = 0; mint = t; opt_p = p; opt_q = q;
         }
         if (terminate) { break; }
         terminate=1;
      }
      // Create a HeteroMPI group of processes with the optimal values of
      // algorithmic parameters
      model_params[0] = n;  model_params[1] = opt_b;
      model_params[2] = bp; model_params[3] = bq;
      model_params[4] = opt_p;  model_params[5] = opt_q;
   }
   // Create a HeteroMPI group of processes
   HMPI_Group_create(gid, model_params);
   return HMPI_SUCCESS;
```

The algorithm can be summarized as follows: The number of steps of the algorithm is represented by the variable `k`. For `k=1`, the total number of processes available for computation, `np`, is determined. All the possible two dimensional process arrangements, (`p`,`q`), whose product is `np`, are obtained using the function, `HMPI_Get_2D_pro- cess_arrangements`. For example, if the total number of processes available is 25, then the possible two dimensional process arrangements whose product of the coordinates is 25 are {(1,25), (5,5), (25,1)}.

Each such process arrangement, (`p`,`q`), is filled into the array of parameters to the performance model of the PDGEMM routine. The function call `HMPI_Timeof` is then invoked to estimate the execution time of the algorithm. One of the inputs to this function call is the handle, `hscal_model_pdgemm`, which encapsulates all the features of the performance model in the form of a set of functions generated by the compiler from the description of the performance model of the PDGEMM routine. The function call `HMPI_Timeof` invokes the mapping algorithms of the HeteroPBLAS runtime system to select such a mapping that is estimated to ensure the fastest execution of the parallel algorithm for that process arrangement. The selection process is described in detail in [1, 7]. It is based on the performance model of the PDGEMM routine and the performance model of the executing network of computers, which reflects the state of the network just before the execution of the PDGEMM routine. During the selection process, the HeteroPBLAS runtime system searches for some approximate solution that can be found in some reasonable interval of time by probation of a subset of all possible mappings. From the execution times predicted for all the possible process arrangements, the process arrangement, (`opt_p`, `opt_q`), that results in the least estimated time of execution of the algorithm is determined.

For the next step, the total number of processes is decremented by one. The possible two dimensional process grid arrangements are again obtained and evaluated using the function `HMPI_Timeof`. The algorithm continues if a process arrangement is found for which the estimated execution time is less than the estimated execution time of the process arrangement (`opt_p`, `opt_q`) determined in the previous step. Otherwise, the algorithm terminates.

The optimal values of the blocking factor and the 2D process grid arrangement (`opt_p`, `opt_q`) are then passed as performance model parameters to the function call, `HMPI_Group_create`, which creates a HeteroMPI group of MPI processes that participate in the execution of the parallel application. These processes become members of the heterogeneous PDGEMM context. This function call is a collective operation and must be called by all the processes available for computation in the predefined communication universe of HeteroMPI.

Heuristics are used to reduce the number of process arrangements evaluated. For example, one dimensional arrangements are not evaluated in the case of matrix–matrix multiplication on Ethernet where it can be proved using simple formulas [16] that they perform poorly compared to two dimensional arrangements and do not minimize the objective function for networks with either sequential communications or parallel communications.

5. Experimental results. We present four sets of experiments. The first set of experiments is run on a homogeneous computing cluster 'Grig' (`https://www.cs.utk\-.edu/help/doku.php?id=clusters:grig`) consisting of 32 Linux nodes with 2 processors per node with Myrinet interconnect. The processor type is Intel EM64T. The software used is heterompi-1.2.0, MPICH-1.2.7, ScaLAPACK-1.8.0 and ATLAS, which is an optimized BLAS library.

We chose two level3 routines, PDGEMM and PDTRSM, for demonstration because they exhibit two different algorithmic patterns. In the case of PDGEMM, the size of the problem solved at each step of its execution, that is number of updates of the resulting matrix, is constant whereas in the execution of PDTRSM, the size of the problem (number of updates of the trailing submatrix) decreases with each step.

The speedup, which is shown in the figures, is calculated as the ratio of the execution time of the homogeneous PBLAS program and the execution time of the HeteroPBLAS program. Dense matrices of size $N \times N$ and vectors of size N were used in the experiments. The homogeneous PBLAS program uses the default parameters suggested by the recommendations from the ScaLAPACK user's guide, which are to (a) use the best BLAS and BLACS libraries available, (b) use a data distribution block size of 64, (c) use a square processor grid and (d) execute no more than one process per processor.

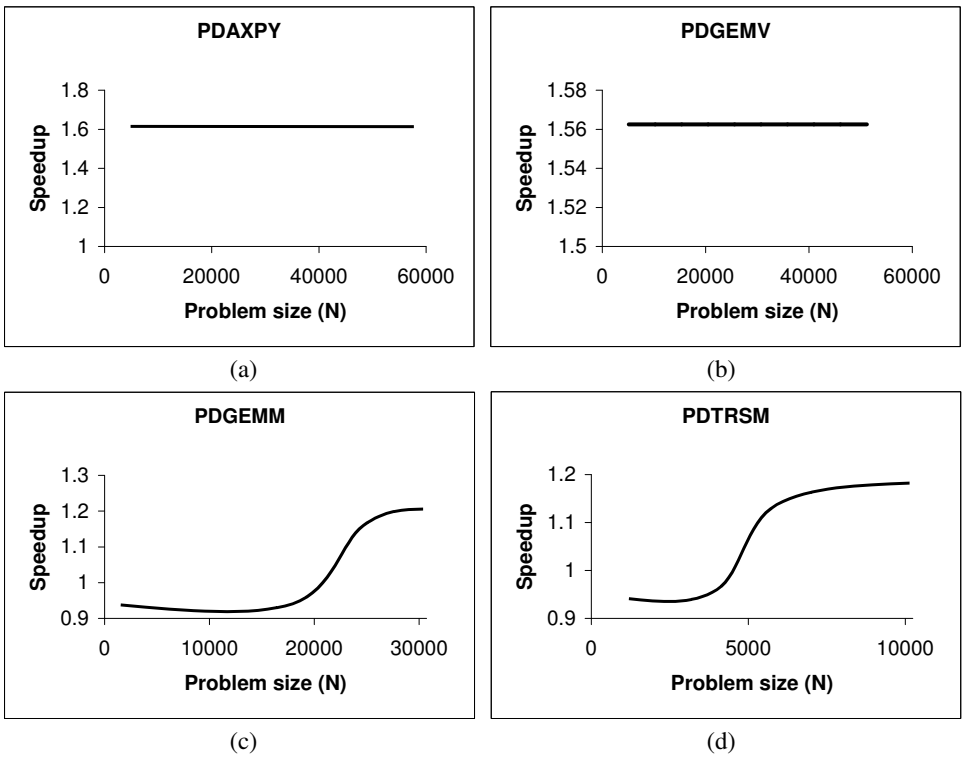


FIG. 5.1. *Experimental results on the homogeneous Grig cluster. N is the size of the vector/matrix. (a) Results of PDAXPY. (b) Results of PDGEMV. (c) Speedup of PDGEMM. (d) Speedup of PDTRSM.*

The first set of experiments is composed of two parts. Fig. 5.1 shows the experimental results of the first part. Fig. 5.1(a) and Fig. 5.1(b) show the experimental results from the execution of the PBLAS level1 routine PDAXPY and level2 routine PDGEMV on the homogeneous cluster. The homogeneous PBLAS programs use a 1×64 grid of processes (using one process per processor configuration). Fig. 5.1(c) and Fig. 5.1(d) shows the experimental results from the execution of the PBLAS level3 routines PDGEMM and PDTRSM, respectively. The homogeneous PBLAS programs use an 8×8 grid of processes (using one process per processor configuration). In the second part, we used the optimal data distribution blocking factor and the optimal process grid arrangement, determined by the HeteroPBLAS program, in the execution of the corresponding homogeneous PBLAS program. From both the parts, it was observed that there is no discernible overhead during the execution of HeteroPBLAS programs. The maximum overhead of about 7% incurred in the case of level3 routines occurs during the creation of the context. The execution times of HeteroPBLAS programs for level1 and level2 routines are the same if one process is executed per computer/node and not per processor. In the case of first part, one can notice that the HeteroPBLAS programs perform better than the homogeneous PBLAS programs. This is because the homogeneous PBLAS programs use the default parameters (recommendations from the user's guide) but not the optimized parameters whereas the HeteroPBLAS programs use the optimal algorithmic parameters such as the optimal blocking factor and the optimal process arrangement.

For equitable comparison, the PBLAS programs must be optimized for the HCC. However one of the goals of this paper is to show how the HeteroPBLAS software automates this procedure of optimization and therefore we use pure/unoptimized ScaLAPACK available online. However, as one must have observed by now, the process of optimization is not trivial.

The second set of experiments is run on a small heterogeneous local network of sixteen different Linux workstations (hcl01–hcl16) whose specifications can be read at the URL `http://hcl.ucd.ie/Hardware/Cluster+Specifications`. The network is based on 2 Gbit Ethernet with a switch enabling parallel communications between the computers. The software used is MPICH-1.2.5, ScaLAPACK–1.8.0, and ATLAS.

The HeteroPBLAS program uses the multiprocessing approach, where more than one process is run on each processor. The number of processes to run on each processor during the program startup is determined automatically by the HeteroPBLAS command line interface tools. A simple heuristic used is the heterogeneity of the network. The heterogeneity of the network due to the heterogeneity of the processors is calculated as the ratio of the absolute speed of the fastest processor to the absolute speed of the slowest processor. For example, consider the benchmark code of a local DGEMM update of two matrices 2560×128 and 128×2560, the absolute speeds of the processors hcl01–hcl16 in million flop/s performing this update are {10829, 9729, 11054, 11151, 11211, 11295, 10958, 10949, 7425, 7047, 10827, 11200, 9056, 11626, 9815, 12250}. As one can see, hcl16 is the fastest processor and hcl10 is the slowest processor. The heterogeneity in this case ≈ 2. The command line tools, therefore, would run two processes on each processor during the program startup.

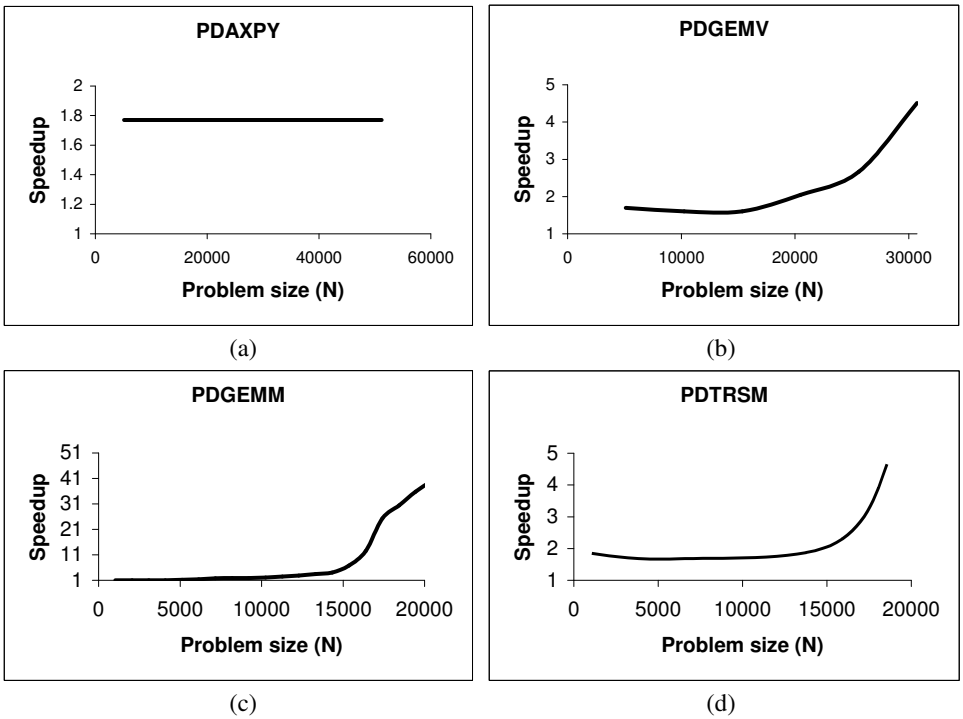


FIG. 5.2. *Experimental results on the heterogeneous cluster. N is the size of the vector/matrix. (a) Speedup of PDAXPY. (b) Speedup of PDGEMV. (c) Speedup of PDGEMM. (d) Speedup of PDTRSM.*

Fig. 5.2(a) and Fig. 5.2(b) show the experimental results from the execution of the PBLAS level1 routine PDAXPY and level2 routine PDGEMV. The homogeneous PBLAS programs use a 1×25 grid of processes (using one process per processor configuration). Fig. 5.2(c) and Fig. 5.2(d) show the experimental results from the execution of the PBLAS level3 routines PDGEMM and PDTRSM respectively. The homogeneous PBLAS program uses a 5×5 grid of processes (using one process per processor configuration).

There are a few reasons behind the superlinear speedups achieved in the case of PDGEMM and eventually for very large problem sizes in the case of PDTRSM not shown in the figure.

The first reason is the better load balance achieved through proper allocation of processes involved in the execution of the algorithm to the processors. During the creation of a HeteroMPI group of processes in

the context creation routine, the mapping of the parallel processes in the group is performed such that the number of processes running on each processor is as proportional to its speed as possible. In other words, while distributed evenly across parallel processes, data and computations are distributed unevenly over processors of the heterogeneous network, and this way each processor performs the volume of computations as proportional to its speed as possible. In the case of execution of PDGEMM on HCCs, it can be seen that for problem sizes larger than 5120, more than 25 processes must be involved in the execution to achieve good load balance. Since only 25 processes are involved in the execution of the homogeneous PBLAS program, good load balance is not achieved. However just running more than 25 processes in the execution of the program would not resolve the problem. This is because in such a case, the optimal process arrangement and the efficient mapping of the processes to the executing computers of the underlying network must also be determined. This is a complex task automated by HeteroMPI. Secondly, the optimal values of the algorithmic parameters that are automatically determined by the HeteroPBLAS library. One of the key algorithmic parameters is the data distribution blocking factor. There is a tradeoff between load imbalance and communication startup cost, which can be controlled by varying the blocking factor, b. The optimal values of the blocking factor determined by the HeteroPBLAS library are in the range ($128 \leq b \leq 256$) as shown in Fig. 5.3. It uses the value of 128. The procedure to determine the optimal block size is performed during the installation of the software.

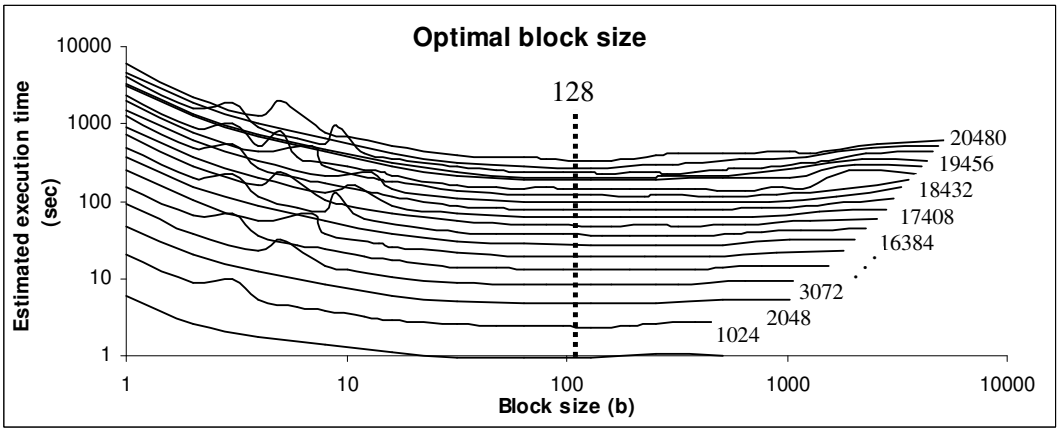


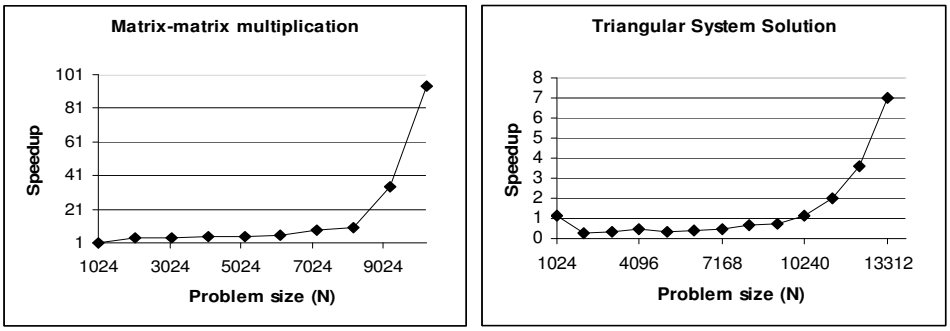
FIG. 5.3. *Optimal block size estimated by the HeteroPBLAS library for various problem sizes ($N \leq 20480$). The execution times are on a log scale.*

Finally, the optimal values of the 2D grid arrangement of processes (`p`,`q`) address the load imbalance caused by the computational "hot spots" where certain processes have more work to do between synchronization points than others. There is an optimal 2D process grid arrangement, (`p`,`q`), which depends on the communications characteristics of the network and determines the overlap of the communication with the computation. During the creation of a HeteroMPI group of processes in the context creation routine, the function `HMPI_Group_pauto_create` estimates the time of execution of the algorithm for each process arrangement evaluated. For each such estimation, it invokes the runtime mapping algorithm, which tries to arrange the processes along a 2D grid to optimally load balance the work of the processors. It returns the process arrangement that results in the least estimated time of execution of the algorithm. Table 5.1 shows the optimal 2D process grid arrangements determined by the context creation routines during the execution of the HeteroPBLAS programs. The total execution time of the HeteroPBLAS program is the sum of the context creation time and the algorithm execution time. It can be seen that for small problem sizes (for example, in the case for PDGEMM where $N \leq 4096$), the context creation times are large compared to the execution times of the parallel algorithms due to the evaluation of a large number of process arrangements. This number can be reduced using meaningful heuristics, which will be a subject for our future study.

Fig. 5.4 shows the execution times of sequential applications and HeteroPBLAS applications solving the same matrix-matrix multiplication and triangular system of equations. The sequential application calls optimized BLAS routines and is executed on the fastest processor (hcl16). The sequential applications performing matrix-matrix multiplication and solving triangular system of equations fail for problem sizes ($N \geq 10240$) and ($N \geq 13312$), respectively. It can be seen that the sequential program solving the triangular system

TABLE 5.1
Optimal 2D process grid arrangements (p,q) determined during the execution of the context creation routines.

Size of the matrix (N)	HeteroPBLAS PDGEMM Program			HeteroPBLAS PDTRSM Program		
	Predicted (p,q)	Context Creation Time (sec)	Execution Time (sec)	Predicted (p,q)	Context Creation Time (sec)	Execution Time (sec)
1024	(2,6)	1.8	0.34	(2,6)	0.5	0.27
2048	(2,6)	2	1	(2,7)	1	7
3072	(2,6)	2	3	(2,7)	1	21
4096	(2,6)	2	6	(2,7)	2	35
5120	(2,7)	2	11	(2,7)	2	80
6144	(2,7)	2	17	(2,7)	3	117
7168	(2,7)	2	27	(2,7)	3	160
8192	(2,7)	3	44	(2,7)	3	208
9216	(2,7)	3	48	(2,7)	4	307
10240	(2,7)	3	65	(2,7)	5	399
11264	(3,7)	11	106	(2,7)	5	491
12288	(3,7)	11	117	(2,7)	5	628
13312	(3,7)	11	164	(2,7)	6	737
14336	(3,7)	12	222	(2,7)	7	853
15360	(3,7)	12	385	(2,7)	8	919
16384	(3,7)	12	754	(2,7)	8	1285
17408	(3,7)	13	1204	(2,7)	9	1358
18432	(3,7)	13	1844	(3,7)	29	2033
19456	(2,14)	13	2729	(3,7)	33	2296
20480	(2,14)	28	4275	(3,11)	125	3017



(a) (b)

FIG. 5.4. *Speedup of the HeteroPBLAS programs over the sequential programs.*

(DTRSM) performs better than the HeteroPBLAS (and therefore PBLAS) program for the range of problem sizes ($N \leq 10240$). It is quite challenging work to find a procedure which automatically determines which program to use (either sequential or HeteroPBLAS) for this range of problem sizes. This is because HeteroPBLAS programs use performance models based on the PBLAS (ScaLAPACK) algorithms, which are clearly not accurate for small problem sizes. This is a subject for our future research.

The final set of experiments shown in Fig. 5.5 demonstrates the efficiency of the HeteroPBLAS program employing the level3 PDGEMM routine. Its efficiency is compared to that of the HeteroMPI program, which adopts the HoHe strategy using heterogeneous 2D block cyclic distribution of matrices [5]. We use the experimental approach to analysis of the performance of heterogeneous algorithms presented in [17]. The HeteroMPI program is close to optimal on the heterogeneous computing cluster. Since the execution time of the HeteroPBLAS program is practically the same as the HeteroMPI program, we can conclude that the efficiency of the HeteroPBLAS program is also close to optimal on this network.

6. Conclusions and future work. In this paper, we present a software library, called Heterogeneous PBLAS (HeteroPBLAS), which provides optimized parallel basic linear algebra subprograms for Heterogeneous Computational Clusters.

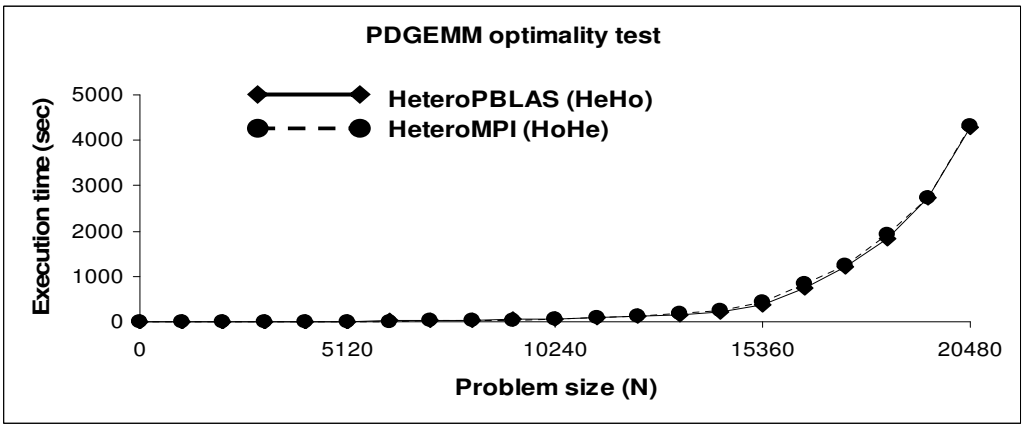


FIG. 5.5. *Execution times of the HeteroPBLAS and the HeteroMPI programs on the heterogeneous cluster. N is the size of the matrix. HeteroMPI program employs heterogeneous 2D block cyclic distribution of matrices.*

We show that the efficiency of the parallel routines is due to the most important feature of the library, which is the automation of the difficult optimization tasks of parallel programming on heterogeneous computing clusters. These tasks are the determination of the accurate values of the platform parameters such as the speeds of the processors and the latencies and bandwidths of the communication links connecting different pairs of processors, the optimal values of the algorithmic parameters such as the data distribution blocking factor, the total number of processes, the 2D process grid arrangement, and the efficient mapping of the processes executing the parallel algorithm to the executing nodes of the heterogeneous computing cluster.

We presented the user interface and the software hierarchy of the first prototype implementation of HeteroPBLAS. Our future work would involve testing of the software on multicore platforms. The software would then be integrated into the Heterogeneous ScaLAPACK package providing high performance routines for dense linear systems, least squares problems, and eigenvalue problems.

REFERENCES

[1] A. LASTOVETSKY AND R. REDDY, *HeteroMPI: Towards a Message-Passing Library for Heterogeneous Networks of Computers*, Journal of Parallel and Distributed Computing (JPDC), Volume 66, No. 2, pp.197–220, Elsevier, 2006.
[2] *Parallel Basic Linear Algebra Subprograms* (PBLAS). `http://www.netlib.org/scalapack/ pblas_qref.html`.
[3] *Basic Linear Algebra Subprograms (*BLAS*)*. `http://www.netlib.org/blas`.
[4] *Basic Linear Algebra Communication Subprograms (*BLACS*)*. `http://www.netlib.org/blacs`.
[5] A. KALINOV AND A. LASTOVETSKY, *Heterogeneous Distribution of Computations Solving Linear Algebra Problems on Networks of Heterogeneous Computers*, Journal of Parallel and Distributed Computing, Volume 61, No. 4, pp. 520–535, April 2001.
[6] A. LASTOVETSKY, D. ARAPOV, A. KALINOV, AND I. LEDOVSKIH, *A Parallel Language and Its Programming System for Heterogeneous Networks*, Concurrency: Practice and Experience, Volume 12, No. 13, pp. 1317–1343, November 2000.
[7] A. LASTOVETSKY, *Adaptive Parallel Computing on Heterogeneous Networks with mpC*, Parallel Computing, Volume 28, No. 10, pp. 1369–1407, October 2002.
[8] *Scalable* LAPACK. `http://www.netlib.org/scalapack/`.
[9] *Linear Algebra PACKage (*LAPACK*)*. `http://www.netlib.org/lapack/`.
[10] O. BEAUMONT, V. BOUDET, A. PETITET, F. RASTELLO, AND Y. ROBERT, *A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers)*, IEEE Transactions on Computers, Volume 50, No. 10, pp. 1052–1070, October 2001.
[11] Y. KISHIMOTO AND S. ICHIKAWA, *An Execution–Time Estimation Model for Heterogeneous Clusters*, In 13th Heterogeneous Computing Workshop (HCW 2004), Proceedings of 18th International Parallel and Distributed Processing Symposium (IPDPS'04), IEEE Computer Society (2004).
[12] A. KALINOV AND S. KLIMOV, *Optimal mapping of a parallel application processes onto heterogeneous platform*, 14th Heterogeneous Computing Workshop (HCW 2005), Proceedings of 19th International Parallel and Distributed Processing Symposium (IPDPS'05), IEEE Computer Society (2005).
[13] J. CUENCA, D. GIMÉNEZ, AND J. P. MARTINEZ, *Heuristics for work distribution of a homogeneous parallel dynamic programming scheme on heterogeneous systems*, Parallel Computing, Volume 31, No. 7, pp. 711–735, Elsevier, 2006.
[14] A. LASTOVETSKY, *Scientific Programming for Heterogeneous Systems—Bridging the Gap between Algorithms and Applications*, Proceedings of the 5th International Symposium on Parallel Computing in Electrical Engineering (PARELEC 2006), Bialystok, Poland, IEEE Computer Society Press, pp. 3–8, 13–17 Sept 2006.
[15] *Heterogeneous* ScaLAPACK. `http://hcl.ucd.ie/project/HeteroScaLAPACK/`.

[16] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, *Matrix Multiplication on Heterogeneous Platforms*, IEEE Transactions on Parallel and Distributed Systems, Volume 12, No. 10, pp. 1033–1051, 2001.
[17] A. Lastovetsky and R. Reddy, *On Performance Analysis of Heterogeneous Parallel Algorithms*, In Parallel Computing, Volume 30, No. 11, pp. 1195–1216, 2004.

Edited by: Marek Tudruj, Dana Petcu
Received: February 7th, 2009
Accepted: June 28th, 2009