Heterogeneous PBLAS: Optimization of PBLAS for Heterogeneous Computational Clusters

Ravi Reddy
School of Computer Science and Informatics,
University College Dublin
manumachu.reddy@ucd.ie

Alexey Lastovetsky
School of Computer Science and Informatics,
University College Dublin
alexey.lastovetsky@ucd.ie

Pedro Alonso
Department of Information Systems and Computation,
Polytechnic University of Valencia
palonso@dsic.upv.es

Abstract

This paper presents a package, called Heterogeneous PBLAS (HeteroPBLAS), which is built on top of PBLAS and provides optimized parallel basic linear algebra subprograms for heterogeneous computational clusters. We present the user interface and the software hierarchy of the first research implementation of HeteroPBLAS. This is the first step towards the development of a parallel linear algebra package for heterogeneous computational clusters. We demonstrate the efficiency of the HeteroPBLAS programs on a homogeneous computing cluster and a heterogeneous computing cluster.

1. Introduction

Parallel Basic Linear Algebra Subprograms (PBLAS) [1, 2] is a parallel set of BLAS [3], 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 [4], which is a well-known standard package providing high-performance linear algebra routines for distributed-memory message passing MIMD computers supporting PVM [5] and/or MPI [6].

To the best of the authors' knowledge, there have only been proposals for implementation of PBLAS on heterogeneous computing clusters (HCC). Beaumont *et al.* [7] discuss data allocation strategies to implement matrix products and dense linear system solvers on HCCs as a basis for a successful extension of the ScaLAPACK library to heterogeneous platforms. They show that extending the standard ScaLAPACK block-cyclic distribution to heterogeneous 2D grids is difficult. In most cases, a perfect balancing of the load between all processors cannot be achieved and deciding how to arrange the processors along the 2D grid is a challenging NP-complete problem.

There are a few research contributions presenting multiprocessing approaches to solve linear algebra kernel on HCCs. Kalinov and Lastovetsky [8] analyze two strategies:

- HeHo - heterogeneous distribution of processes over processors and homogeneous block cyclic distribution of data over the processes;
- 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.

Both strategies were implemented in the mpC language [9, 10]. The first strategy is implemented using calls to ScaLAPACK; the second strategy is implemented with calls to LAPACK [11] and BLAS. They compare the strategies using Cholesky factorization on a network of workstations. They show that for heterogeneous parallel environments both the strategies HeHo and HoHe are more efficient than the traditional homogeneous strategy HoHo (homogeneous distribution of processes over processors and homogeneous distribution of data over the processes as implemented in ScaLAPACK). The main disadvantage of the HoHe strategy is non-Cartesian nature of the data distribution. This leads to additional communications that can be expensive in the case of large networks. The HeHo strategy is easy to accomplish. It allows the reuse of high-quality software, such as ScaLAPACK, developed for homogeneous distributed memory systems in heterogeneous environments and to obtain a good speedup with minimal expenses. Kishimoto and Ichikawa [12] adopt a multiprocessing approach 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 [13] investigate the HeHo strategy where the performance of the processor is given as a function of

978-0-7695-3472-5/08 $25.00 © 2008 IEEE
DOI 10.1109/ISPDC.2008.9

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*. [14] analyse automatic optimization techniques in the design of parallel dynamic programming algorithms on heterogeneous platforms. The main idea is to automatically determine the optimal values of a number of algorithmic parameters such as (number of processes, number of processors, processes per processor). To summarize, the multiprocessing strategy is easy to accomplish. It allows the complete reuse of high-quality software such as ScaLAPACK, which is developed for homogeneous distributed memory systems, in heterogeneous environments with minimal expenses and good speedup.

In this paper, we present Heterogeneous PBLAS (HeteroPBLAS), which provides optimized parallel basic linear algebra subprograms for HCCs. The design of the package adopts the multiprocessing approach and thus reuses the PBLAS software completely. This can be seen as the first step towards the development of a parallel linear algebra package for HCCs, which will be called Heterogeneous ScaLAPACK and built on top of ScaLAPACK.

We start with the presentation of the user interface to the HeteroPBLAS package. Then we describe the different software components and building blocks of the first research implementation of the interface. This is followed by experimental results of execution of PBLAS programs on a homogeneous computing cluster and a heterogeneous computing cluster. We conclude the paper by stating our future research goals.

2. HeteroPBLAS User Interface

The main routine is the context creation function, which provides a context for the execution of the PBLAS routine. There is a context creation function for each and every PBLAS routine. This function frees the application programmer from having to specify the process grid arrangement to be used in the execution of the PBLAS routine. It tries to determine the optimal process grid arrangement.

All the context creation routines have names of the form **`hscal_pxyyzzz_ctxt`**. The second letter, **`x`**, indicates the data type. The next two letters, **`yy`**, indicate the type of matrix (or of the most significant matrix). The last three letters **`zzz`** 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 [15] group of MPI processes in **`ictxt`** and a return value of **`HSCAL_SUCCESS`** on successful execution. It differs from the PDGEMM call in the following ways:

- It returns a context but does not actually execute the PDGEMM routine;
- The matrices *A*, *B* and *C* containing the data are not passed as arguments;
- It has 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;
- A return value of **`HSCAL_SUCCESS`** indicating successful execution or otherwise an appropriate error code;
- The context element in the descriptor arrays **`desca`**, **`descb`** and **`descc`** need not be filled.

`hscal_pdgemm_ctxt` is a collective operation and must be called by all the processes running in the HeteroPBLAS application. 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. During the creation of the HeteroMPI group of MPI processes, the HeteroPBLAS runtime system tries to detect the optimal process arrangement as well as solves the problem of selection of the optimal set of processes running on different computers of the heterogeneous network. It should be noted that this problem of mapping, in general, is a NP-complete problem. The solution to the problem is based on the following:

- The performance model of the PBLAS routine. This is in the form of a set of functions generated by a compiler from the description of the performance model of the PBLAS routine;
- The performance model of the executing network of computers, which reflects the state of this network just before the execution of the PBLAS routine. This model takes into account the material nature of communication links and their heterogeneity [10].

The mapping algorithms used to solve the problem of selection of processes are detailed in [10, 15]. The reader is referred to the HeteroPBLAS programmer's manual for more details of the HeteroPBLAS user interface. It also presents the essential, which are also very few, differences between calling a homogeneous PBLAS routine and a heterogeneous PBLAS routine using code snapshots.

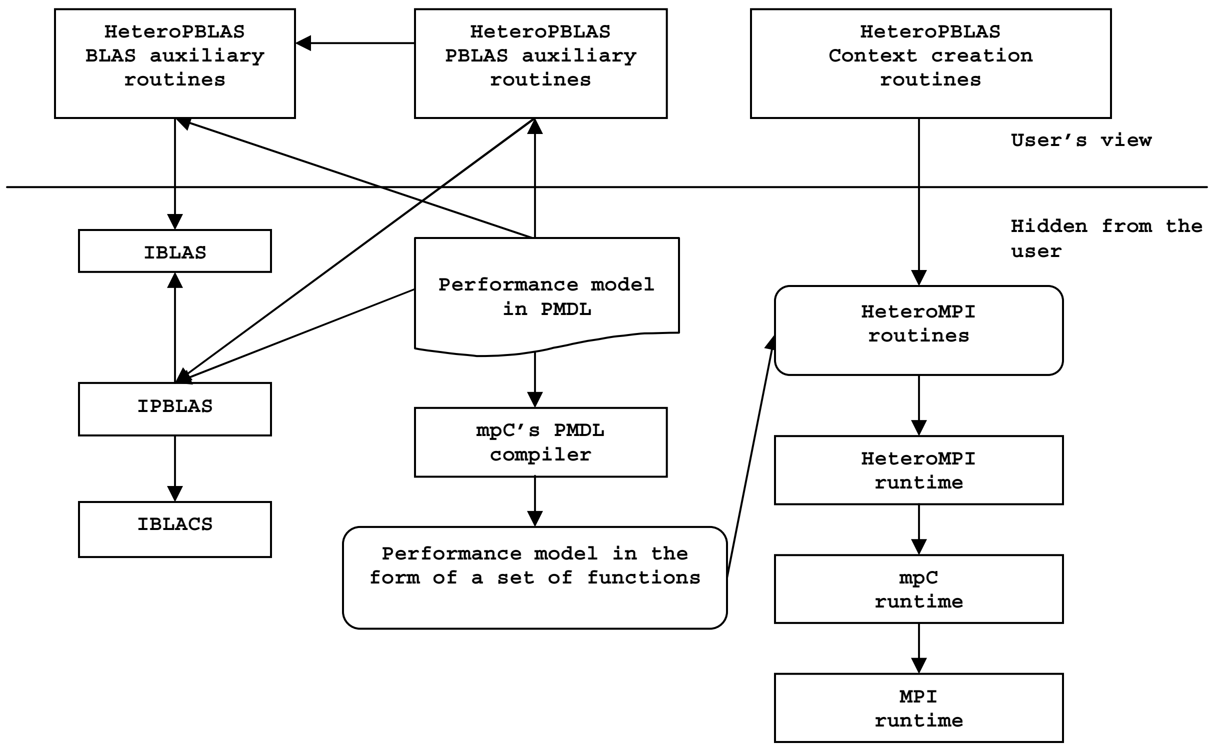

HeteroPBLAS BLAS auxiliary routines
HeteroPBLAS PBLAS auxiliary routines
HeteroPBLAS Context creation routines
User's view
Hidden from the user
IBLAS
Performance model in PMDL
HeteroMPI routines
IPBLAS
mpC's PMDL compiler
HeteroMPI runtime
IBLACS
Performance model in the form of a set of functions
mpC runtime
MPI runtime

Figure 1. Heterogeneous PBLAS software hierarchy.

3. HeteroPBLAS Software Design

The software hierarchy of HeteroPBLAS package is shown in Figure 1. The package can be downloaded from the URL: http://hcl.ucd.ie/Software/HeteroScaLAPACK. The building blocks are HeteroMPI, BLACS [16], PBLAS and BLAS and are not contributions of this paper. The HeteroPBLAS context creation routines call interface functions of HeteroMPI, which invoke the HeteroMPI runtime. The HeteroPBLAS auxiliary functions of PBLAS, BLACS and BLAS call the instrumented PBLAS, BLACS and BLAS code shown in the software hierarchy diagram as IPBLAS, IBLACS and BLAS respectively. The instrumented code reuses the existing code base completely. The only modifications are (a) Replacement of the serial BLAS computation routines and the BLACS communication routines by calls to estimation functions determining the number of arithmetical operations performed by each process and number of communications in bytes performed by a pair of processes respectively and (b) Wrapping of the parallel regions of the code in mpC `par` loops. An optimized set of BLACS for HCCs as well as a well-defined interface of corresponding auxiliary functions will be provided in future releases of the software.

The first step in the implementation of the context creation routine for a PBLAS routine is the description of its performance model using a performance model definition language (PMDL). The performance model allows an application programmer to specify his or her high-level knowledge of the application that can assist in finding the most efficient implementation on HCCs. This model allows specification of all the main features of the underlying parallel algorithm that have an essential impact on application execution performance on HCCs. These features are

- The total number of processes executing the algorithm;
- The total volume of computations to be performed by each of the processes in the group during the execution of the algorithm;
- The total volume of data to be transferred between each pair of processes in the group during the execution of the algorithm;
- 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.

The PMDL uses most of the features in the specification of network types of the mpC language [9, 10]. The mpC compiler compiles the description of this performance model to generate a set of functions, which make up the algorithm-specific part of the mpC runtime system. These functions are called by the mapping algorithms of mpC runtime to estimate of the cost of

```
/* 1 */ algorithm pdgemm(int n, int b, int t, int p, int q)
/* 2 */ {
/* 3 */   coord I=p, J=q;
/* 4 */   node {I>=0 && J>=0: bench*((n/(b*p))*(n/(b*q))*(n*b)/(t*t));};
/* 5 */   link (K=p, L=q)
/* 6 */   {
/* 7 */      I>=0 && J>=0 && I!=K :
/* 8 */        length*((n/(b*p))*(n/(b*q))*(b*b)*sizeof(double))
/* 9 */              [I, J]->[K, J];
/* 10 */     I>=0 && J>=0 && J!=L:
/* 11 */       length*((n/(b*p))*(n/(b*q))*(b*b)*sizeof(double))
/* 12 */             [I, J]->[I, L];
/* 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/(b*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/(b*p))) %% [((k/b)%p),j]->[i,j];
/* 28 */       par(i = 0; i < p; i++)
/* 29 */         par(j = 0; j < q; j++)
/* 30 */           ((100.0×b)/n) %% [i,j];
/* 31 */     }
/* 32 */   };
/* 33 */ };
```

Figure 2. Description of the performance model of the PDGEMM routine in the mpC's performance model definition language.

execution of the parallel algorithm. This happens during the creation of the context (see the steps outlined below).

The description of performance models of all the PBLAS routines (about 123 of them) has been the most intricate effort in 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 PBLAS package.

The performance model definition of PDGEMM PBLAS routine shown in Figure 2 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 [10, 15] 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 to be equal to the algorithmic blocking factor.

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**, **t**, **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. Parameter **t** is used for the benchmark code, which is assumed to multiply two **t×b** and **b×t** matrices. Parameters **p** and **q** are output parameters representing 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 abstract processors with this coordinate system to form a **p×q** grid. It specifies the (absolute) volume of computations to be performed by each of the processors. The statement **bench** just specifies that as a unit of measurement, the volume of computation performed by some benchmark code be used. It is presumed that the benchmark code, which is used for estimation of speed of physical processors, multiplies two dense square **t×b** and **b×t** matrices. The line 4 of node declaration specifies that the volume of computations to be performed by the abstract processor with coordinates **`(I,J)`** is **`((n/(b*p))*(n/(b*q))*(n*t/t*t))`** times bigger than the volume of computations performed by the benchmark code.

Lines 5-13 are a *link declaration*. This specifies the links between the abstract processors, the pattern of communication among the abstract processors, and the total volume of data to be transferred between each pair of abstract processors during the execution of the algorithm. Lines 7-9 of the link declaration describe vertical communications related to matrix A. Only abstract processors from the same row of the processor grid send each other elements of matrix A. The volume of data in one **`b×b`** block is given by **`(b*b)*sizeof(double)`** and so the total volume of data transferred from processor P_{IJ} to processor P_{KJ} will be given by **`(n/(b×p))×(n/(b×q))×b×b×sizeof(double)`**.

Lines 10-13 of the link declaration describe horizontal communications related to matrix B. Obviously, only abstract processors from the same column of the processor grid send each other elements of matrix B. The volume of data in one **`b×b`** block is given by **`(b*b)*sizeof(double)`** and so the total volume of data transferred from processor P_{IJ} to processor P_{IL} will be given by **`(n/(b×p))×(n/(b×q))×b×b×sizeof(double)`**.

Line 15 introduces the *scheme declaration*. The **`scheme`** block describes how exactly abstract processors 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 abstract processor with the coordinates (`i`). Each communication unit is of the form $e\%\%[i] \rightarrow [j]$ specifying transfer of data from abstract processor with coordinates `i` to the abstract processor 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`**,

- Lines 20-23 describe vertical communications related to matrix A. **`(100.*(n/(b*q))`** percent of data, that should be in total be sent from processor P_{IJ} to processor P_{KJ} , will be sent at the step. The **`par`** algorithmic patterns imply that during the execution of this communication, data transfer between different pairs of processors is carried out in parallel;
- Lines 24-27 describe horizontal communications related to matrix B. **`(100.*(n/(b*p))`** percent of data, that should be in total be sent from processor P_{IJ} to processor P_{IL} , will be sent at the step;
- Lines 28-30 describe computations. Each abstract processor updates each 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, the processor 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 abstract processors perform their computations in parallel.

The example just described demonstrates the complexity of performance model description of even the simplest case of PDGEMM PBLAS routine. 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`. The performance model files start with prefix `pm_` followed by the name of the PBLAS routine and have a file extension `mpc`.

The execution of a HeteroPBLAS context creation routine consists of the following steps:

1. Updating the estimation of the speeds of the processors using the HeteroMPI routine `HMPI_Recon`. A benchmark code representing the core computations involved in the execution of the PBLAS routine is provided to this function call to accurately estimate the speeds of the processors. For example in the case of the PDGEMM routine, the benchmark code provided is a local GEMM update of `m×b` and `b×n` matrices where `b` is the data distribution blocking factor and `m` and `n` are local number of matrix rows and columns respectively;
2. Finding the optimal values of the parameters of the parallel algorithm used in the PBLAS routine, such as the algorithmic blocking factor and the data distribution blocking factor, using the HeteroMPI routine `HMPI_Timeof`;
3. Creation of a HeteroMPI group of MPI processes using the HeteroMPI's group constructor routine `HMPI_Group_pauto_create`. One of the inputs to this function call is the handle, 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 PBLAS routine. During this function call, the HeteroMPI runtime system detects the optimal process arrangement as well as solves the problem of selection of the optimal set of processes running on different computers of the heterogeneous network. The selection process is described in detail in [10, 15]. It is based on the performance model of the PBLAS routine and the performance model of the executing network of computers, which reflects the state of this network just before the execution of the PBLAS routine;

4. The handle to the HeteroMPI group is passed as input to the HeteroMPI routine `HMPI_Get_comm` to obtain the MPI communicator. This MPI communicator is translated to a BLACS handle using the BLACS routine `Csys2blacs_handle`;
5. The BLACS handle is then passed to the BLACS routine `Cblacs_gridinit`, which creates the BLACS context. This context is returned in the output parameter.

The HeteroPBLAS program uses the multiprocessing approach, which allows more than one process involved in its execution to be run on each processor. The multiprocessing approach can be summarized as follows:

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

Thus, while distributed evenly across parallel processes, data and computations are distributed unevenly over processors of the heterogeneous network so that each processor performs the volume of computations proportional to its speed. The number of processes to run on each processor during the program startup is determined automatically by the HeteroPBLAS command-line interface tools.

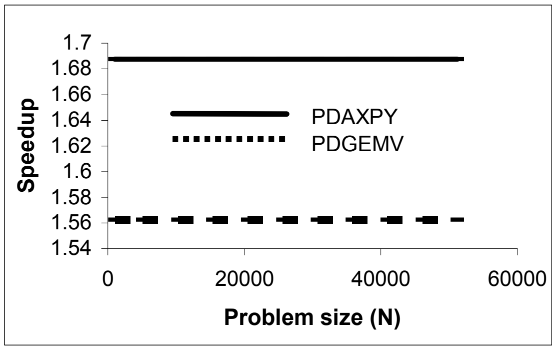
The future versions of the HeteroPBLAS software would support three execution models. The first execution model, which is currently supported, is the simplest. Only the estimation of the cost of execution (execution time) of the PBLAS routines is provided. The cost of redistribution of data between the slaves are not taken into consideration. The second execution model supports the master-slave pattern. In this model, the master distributes data amongst the slaves. The results are returned to the master. The cost of distribution of data by the master amongst the slaves and the cost of accumulation of results at the master from the slaves will be taken into consideration. The third model is the most complicated allowing a mixture of master-slave and slave-to-slave models. In this model, the master distributes data amongst the slaves. The slaves execute one or more calls to a PBLAS routine. The slaves then communicate the results to a different group of slaves, which execute one or more calls of a different PBLAS routine. Finally, the results are returned to the master. So in this model, the cost of redistribution of data between the slaves in addition to the costs of distribution of data amongst the slaves by the master and the cost of accumulation of results at the master from the slaves will be taken into consideration.

4. Experimental Results

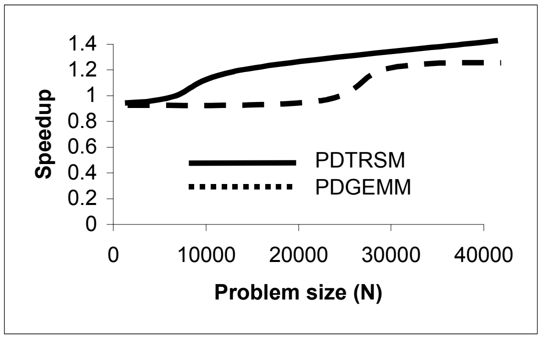
We present three sets of experiments. The first set of experiments is run on a homogeneous computing cluster (https://www.cs.utk.edu/help/doku.php?id=clusters) consisting of 64 Linux nodes with 2 processors per node with Myrinet interconnect. The processor type is Intel EM64T. The software used is MPICH-1.2.7, ScaLAPACK-1.8.0 and ATLAS [17], which is an optimized BLAS library. Only 32 nodes (64 processors) are used in the experiments.

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×N and vectors of size N were used in the experiments. The homogeneous PBLAS programs uses the default parameters recommended by the ScaLAPACK user's guide [3]. We chose two level-3 routines, which are 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 sub-matrix) decreases with each step.

The first set of experiments is composed of two parts. Figures 3(a) and 3(b) show the experimental results of the first part. Figure 3(a) shows the experimental results from the execution of the PBLAS level-1 routine PDAXPY and level-2 routine PDGEMV on the homogeneous cluster. The homogeneous PBLAS programs use a 1×64 grid of processes (using one process per processor configuration). Figure 3(b) show the experimental results from the execution of the PBLAS level-3 routines PDGEMM and PDTRSM respectively. The homogeneous PBLAS program uses 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

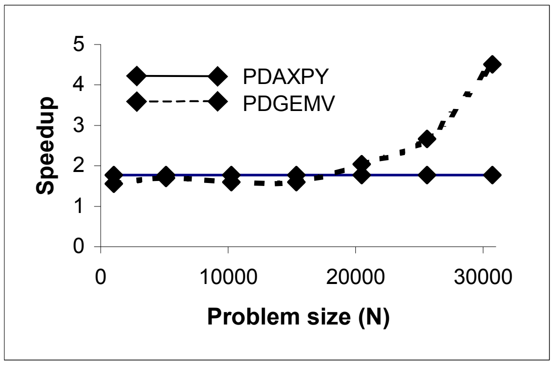


(a)

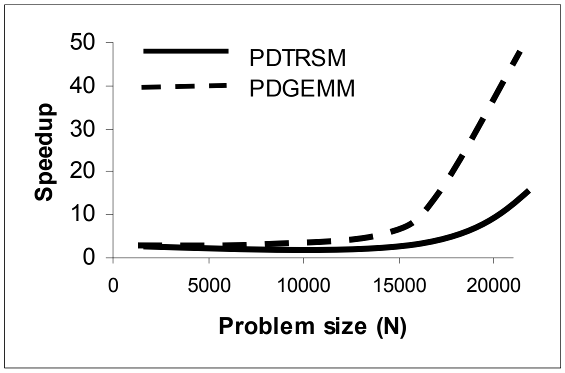


(b)

Figure 3. The network used is the homogeneous Grig cluster. N is the size of the vector/matrix.



(a)



(b)

Figure 4. The network used is the heterogeneous cluster. N is the size of the vector/matrix.

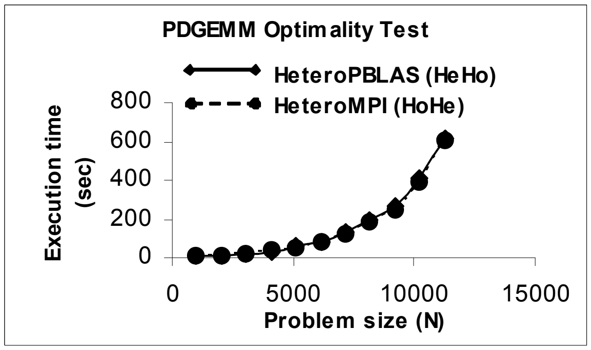


Figure 5. Execution times of the HeteroPBLAS and the HeteroMPI programs on the heterogeneous cluster. HeteroMPI program employs heterogeneous 2D block-cyclic distribution of matrices.

discernible overhead during the execution of HeteroPBLAS programs. The maximum overhead of about 7% incurred in the case of level-3 routines occurs during the creation of the context. The execution times of HeteroPBLAS programs for level-1 and level-2 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 but not the optimized parameters whereas the HeteroPBLAS programs use accurate platform parameters and the optimal algorithmic parameters such as the optimal block factor and the optimal process arrangement. The parameters for the homogeneous PBLAS programs must be tweaked for just comparision with the HeteroPBLAS programs but this process is tedious and is automated by HeteroPBLAS, which is one of the results of this work.

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 absolute speeds of the processors, in million flop/s, performing a local GEMM update of two matrices 3072×64 and 64×3072 are {8866, 7988, 8958, 8909, 9157, 9557, 8907, 8934, 2179, 5940, 3232, 7054, 6824, 3268, 3144, 3769}. Therefore, hcl06 is the fastest processor and hcl09 is the slowest processor. 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, which is 4.4. Figure 4(a) shows the experimental results from the execution of the PBLAS level-1 routine PDAXPY and level-2 routine PDGEMV. The homogeneous PBLAS programs use a 1×25 grid of processes (using one process per processor configuration). Figure 4(b) shows the experimental results from the execution of the PBLAS level-3 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 super-linear 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. The second reason is the optimal 2D grid arrangement of processes. 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 mapping algorithm, which tries to arrange the processors along a 2D grid so as 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.

The third set of experiments shown in Figure 5 demonstrates the efficiency of the HeteroPBLAS program employing the level-3 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 [8]. We use the experimental approach to analysis of the performance of heterogeneous algorithms presented in [18]. 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.

We would present experimental results on multicore architectures in our future work.

5. Conclusions and Future Work

In this paper, we have presented a package, called Heterogeneous PBLAS (HeteroPBLAS), providing parallel basic linear algebra subprograms for Heterogeneous Networks of Computers (HCCs). Our future work will involve the development of the Heterogeneous ScaLAPACK package. The contents of this package will include: (a) The heterogeneous PBLAS package presented in this paper (b) The context creation and auxiliary routines for the ScaLAPACK routines (c) An optimized set of Basic Linear Algebra Communication Subprograms (BLACS) for HCCs and (d) A tool that would automatically transform ScaLAPACK programs to heterogeneous ScaLAPACK programs designed to run efficiently on HCCs.

References

[1] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. W. Walker and R. C. Whaley, "A Proposal for a Set of Parallel Basic Linear Algebra Subprograms," In Proceedings of the Second Workshop on Parallel Scientific Computing, Lyngby, Denmark, LNCS Volume 1041, Springer-Verlag, pp.107-114, 1996.

[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] Scalable LAPACK. http://www.netlib.org/scalapack/.

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

[6] J. Dongarra, S. H. Lederman, S. Otto, M. Snir, and D. Walker. MPI: The Complete Reference. The MIT Press, 1996.

[7] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert, "A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers)," In IEEE Transactions on Computers, Volume 50, No. 10, pp.1052-1070, October 2001.

[8] A. Kalinov and A. Lastovetsky, "Heterogeneous Distribution of Computations Solving Linear Algebra Problems on Networks of Heterogeneous Computers," In Journal of Parallel and Distributed Computing, Volume 61, No. 4, pp.520-535, April 2001.

[9] A. Lastovetsky, D. Arapov, A. Kalinov, and I. Ledovskih, "A Parallel Language and Its Programming System for Heterogeneous Networks," In Concurrency: Practice and Experience, Volume 12, No. 13, pp.1317-1343, November 2000.

[10] A. Lastovetsky, "Adaptive Parallel Computing on Heterogeneous Networks with mpC," In Parallel Computing, Volume 28, No.10, pp.1369-1407, October 2002.

[11] Linear Algebra PACKage (LAPACK). http://www.netlib.org/lapack/.

[12] Y. Kishimoto and S. Ichikawa, "An Execution-Time Estimation Model for Heterogeneous Clusters," In 13th Heterogeneous Computing Workshop (HCW 2004), in Proceedings of 18th International Parallel and Distributed Processing Symposium (IPDPS'04), IEEE Computer Society (2004).

[13] A. Kalinov and S. Klimov, "Optimal mapping of a parallel application processes onto heterogeneous platform," In 14th Heterogeneous Computing Workshop (HCW 2005), in Proceedings of 19th International Parallel and Distributed Processing Symposium (IPDPS'05), IEEE Computer Society (2005).

[14] J. Cuenca, D. Giménez, and J-P. Martinez, "Heuristics for work distribution of a homogeneous parallel dynamic programming scheme on heterogeneous systems," in Parallel Computing, Volume 31, No. 7, pp.711-735, Elsevier, 2006.

[15] A. Lastovetsky and R. Reddy, "HeteroMPI: Towards a Message-Passing Library for Heterogeneous Networks of Computers," in Journal of Parallel and Distributed Computing (JPDC), Volume 66, No. 2, pp.197-220, Elsevier, 2006.

[16] Basic Linear Algebra Communication Subprograms (BLACS). http://www.netlib.org/blacs/.

[17] Automatically Tuned Linear Algebra Software (ATLAS). http://math-atlas.sourceforge.net/.

[18] A. Lastovetsky and R. Reddy, "On Performance Analysis of Heterogeneous Parallel Algorithms," In Parallel Computing, Volume 30, No. 11, pp.1195-1216, 2004.