2008 International Symposium on Parallel and Distributed Computing

Scalable Dense Factorizations 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 discusses the design and the implementation of the LU factorization routines included in the Heterogeneous ScaLAPACK library, which is built on top of ScaLAPACK. These routines are used in the factorization and solution of a dense system of linear equations. They are implemented using optimized PBLAS, BLACS and BLAS libraries for heterogeneous computational clusters. We present the details of the implementation as well as performance results on a heterogeneous computing cluster.

1. Introduction

This paper discusses the design and the implementation of the LU factorization routines included in the Heterogeneous ScaLAPACK library. These routines are used in the factorization and solution of a dense system of linear equations.

Heterogeneous ScaLAPACK [1] is a software package providing optimized parallel linear algebra programs for heterogeneous computational clusters (HCCs). It is built on top of ScaLAPACK [2] and reuses its software fully. It is currently under development. At the moment, it contains full implementation of Heterogeneous PBLAS, which provides optimized parallel basic linear algebra subprograms for HCCs. The building blocks of Heterogeneous PBLAS are PBLAS [3], BLACS [4] and BLAS [5].

There are a few research contributions to compute LU factorization on heterogeneous computational clusters (HCC). However there is only a single proposal [6] mooting provision of LU factorization routines in the form of a library and the issues involved thereof. The authors 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.

A few contributions present multiprocessing approaches to solve linear algebra kernel on HCCs. 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.

To summarize their results, the multiprocessing strategy is easier 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 development efforts and good speedup. Furthermore software providing optimized parallel linear algebra programs on HCCs must automate the tedious and error-prone tasks of determining the accurate platform parameters such as speeds of the processors, latencies and bandwidths of the communication links connecting different pairs of processors and optimal algorithmic parameters such as number of processes, number of processors, number of processes per processor involved in the execution of the parallel algorithm and the mapping of the processes to the executing nodes of the HCC. The Heterogeneous ScaLAPACK library performs these automations. This paper demonstrates how they are achieved by presenting details of the implementation of a LU factorization routine.

978-0-7695-3472-5/08 $25.00 © 2008 IEEE
DOI 10.1109/ISPDC.2008.10

```
/* 1 */ algorithm pdgetrf(int M, int N, int IA, int JA,
/* 2 */                  int DESCA[DLEN1_], int p, int q) {
/* 3 */   coord I=p, J=q;
/* 4 */   node {I>=0 && J>=0: bench*(hscal_pdgetrf_tcomp(I, J, M, N, IA,
/* 5 */         JA, DESCA, p, q)/hscal_pdgetrf_bench(M, N));};
/* 5 */   link (K=p, L=q) {
/* 6 */        I>=0 && J>=0: length*(hscal_pdgetrf_tcomm(I, J, K, L, M,
/* 7 */         N, IA, JA, DESCA, p, q)) [I, J] -> [K, L];
/* 8 */  };
/* 9 */  parent[0,0];
/* 10 */ scheme {
/* 11 */  int bf, i__3, i__4, jb, jn; double *tcomp, *tcomm;
/* 12 */  hscal_pdgetf2_s(&M, JB_l, ..., p, q, tcomp, tcomm);
/* 13 */  if (jb+1 <= N) {
/* 14 */    hscal_pdlaswp_s("F", "R", ..., p, q, tcomp, tcomm);
/* 15 */    hscal_pdtrsm_s("L", "L", "N", "U", ..., p, q, tcomp, tcomm);
/* 16 */    if (jb+1 <= M) hscal_pdgemm_s("N", "N", ..., p, q, tcomp, tcomm);
/* 17 */  }
/* 18 */  for(j = jn+1; j <= min(M, N); j += bf) {
/* 19 */     i__3 = M - j + JA;
/* 20 */     hscal_pdgetf2_s(&i__3, ..., p, q, tcomp, tcomm);
/* 21 */     hscal_pdlaswp_s("F", "R", &i__3, ..., p, q, tcomp, tcomm);
/* 22 */     if (j - JA + jb + 1 <= N) {
/* 23 */       hscal_pdlaswp_s("F", "R", &i__3, ..., p, q, tcomp, tcomm);
/* 24 */       hscal_pdtrsm_s("L", "L", "N", "U", &jb, &i__3, ...,
/* 25 */                       p, q, tcomp, tcomm);
/* 26 */       if (j - JA + jb + 1 <= M) {
/* 27 */         i__3 = M - j - jb + JA;
/* 28 */          i__4 = N - j - jb + JA;
/* 29 */         hscal_pdgemm_s("N", "N", &i__3, &i__4, &jb, ...,
/* 30 */                         p, q, tcomp, tcomm);
/* 31 */       }
/* 32 */     }
/* 33 */  }
/* 34 */ };
/* 35 */ };

         /* Simplified scheme of PDGEMM performance model */
/* 1 */  scheme hscal_pdgemm_s(char *TRANSA, char *TRANSB, int n, int b,
/* 2 */                        int p, int q, double *tcomp, double *tcomm) {
/* 3 */    int i, j, k ;
/* 4 */    for(k = 0; k < n; k+=b) {
/* 5 */       par(i = 0; i < p; i++)
/* 6 */          par(j = 0; j < q; j++)
/* 7 */             if (j != ((k/b)%q))
/* 8 */               (100.0*b*b*(n/(b*p))/TCOMM(i, ((k/b)%q), i, j, p, q)
/* 9 */                                   %% [i,((k/b)%q)]->[i,j];
/* 10 */      par(i = 0; i < p; i++)
/* 11 */         par(j = 0; j < q; j++)
/* 12 */            if (i != ((k/b)%p))
/* 13 */              (100.0*b*b*(n/(b*q))/TCOMM(((k/b)%p), j, i, j, p, q)
/* 14 */                                  %% [((k/b)%p),j]->[i,j];
/* 15 */      par(i = 0; i < p; i++)
/* 16 */         par(j = 0; j < q; j++)
/* 17 */           ((100.0*2*b*b*b*(n/(b*p))*(n/(b*q)))/tcomp[i*q+j]) %% [i,j];
/* 18 */     }
/* 19 */ };
```

Figure 1. Description of the performance model of the PDGETRF routine in the mpC's performance model definition language.

The rest of the paper is organized as follows. We start with the implementation of the LU factorization in the Heterogeneous ScaLAPACK library. This is followed by experimental results of execution of Heterogeneous ScaLAPACK programs employing the LU factorization routines on a local network of heterogeneous computers. We conclude the paper by outlining our future research goals.

2. Heterogeneous ScaLAPACK LU Factorization

In this section, we present the implementation details of the LU factorization routine (PDGETRF) of a general distributed matrix, which uses partial pivoting with row interchanges, in the Heterogeneous ScaLAPACK library. The details are equally applicable to other LU factorization routines supported in ScaLAPACK. We refer the readers to [7] for the description of the LU Factorization algorithm and the parallel implementation of the ScaLAPACK PDGETRF routine.

2.1 Performance model of ScaLAPACK LU Factorization

The first step in the implementation is the description of its performance model using a performance model definition language (PMDL). The performance model allows application programmer to specify their 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 (which is a output parameter);
- The total volume of computations to be performed by each of the processes during the execution of the algorithm;
- The total volume of data to be transferred between each pair of processes during the execution of the algorithm;
- The order of execution of the computations and communications by the parallel processes, that is, how exactly the processes interact during the execution of the algorithm (which computations are performed in parallel, which are serialized, which computations and communications overlap, etc.).

The PMDL uses most of the features in the specification of network types of the mpC language [8]. 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 execution time of the parallel algorithm. This happens during the creation of the heterogeneous context of the ScaLAPACK routine (the steps are outlined in the following section).

The description of the performance model has been the most complicated and tedious effort. The key design issues were (a) accuracy to facilitate accurate prediction of the execution time of the ScaLAPACK routine, (b) efficiency to execute the performance model in reasonable execution time, (c) reusability to reuse the performance models as building blocks for the solution to dense linear system of equations and (d) preservation to preserve the key design features of underlying ScaLAPACK package.

The performance model definition of PDGETRF ScaLAPACK routine shown in Figure 1 demonstrates the complexity of the effort of writing a performance model. Lines 1-2 is a header of the performance model declaration. It introduces the name of the performance model **`pdgetrf`** parameterized with the scalar integer parameters **`M`**, **`N`**, **`IA`**, **`JA`**, **`p`** and **`q`** and a vector parameter, which is the descriptor array **`DESCA`** for the matrix *A*. Parameters **`M`** and **`N`** are the rows and columns of the matrix *A*. Parameters **`IA`** and **`JA`** are the row index and the column index in the matrix. Parameters **`p`** and **`q`** are output parameters representing the number of process rows and columns 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. Lines 4-5 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. The auxiliary function `hscal_pdgetrf_tcomp` calculates the absolute total volume of computations performed by the abstract processor with coordinates **`(I,J)`** during the execution of the PDGETRF routine. It is presumed that the benchmark code, which is used for estimation of speeds of processors, performs a local GEMM update of two dense **`m×b`** and **`b×n`** matrices where **`b`** is the optimal data distribution factor determined by the Heterogeneous ScaLAPACK runtime system and (**`m`**,**`n`**) are heuritics determined based on the problem size. The auxiliary function `hscal_pdgetrf_bench` calculates the absolute total volume of computations performed by the processor during the execution of the benchmark code, which is **`2×m×b×n`**. The lines of node declaration specify that the volume of computations to be performed by the abstract processor with coordinates **(I,J)** is (hscal_pdgetrf_tcomp(…)/hscal_pdgetrf_bench(…)) times bigger than the volume of computations performed by the benchmark code.

Lines 5-8 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. The auxiliary function hscal_pdgetrf_tcomm calculates the total volume of data in matrix elements transferred between processors with coordinates **(I,J)** and **(K,L)** during the execution of the PDGETRF routine. The total volume of data in bytes transferred from processor P_{IJ} to processor P_{KJ} will be given by (hscal_pdgetrf_tcomm(…))**×sizeof(double**).

Line 10 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 describes the first block of **b** columns separately. Then it describes **min(M,N)** successive steps of the algorithm. At each step **k**,

- Line 20 describes the LU factorization of the current panel by the current column of processes using PDGETF2;
- Line 21 describes the row interchanges to the left and right of the current panel using PDLASWP;
- Lines 24-25 describe the broadcast of L_{11} and computation of block row of U_{12} using PDTRSM and finally
- Lines 29-30 describes the broadcast of L_{21} and U_{12} followed by updating of trailing matrix A_{22} using PDGEMM.

Due to space limitations, we would only highlight the important points in the scheme of the PDGEMM routine. The scheme hscal_pdgemm_s 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 [9,10] for more details of the description of the performance model of PDGEMM. This definition is an extensively stripped down version of the actual, which can be studied from from the file /PBLAS/SRC/pm_pdgemm.mpc in the Heterogeneous ScaLAPACK package. The scheme declaration describes **(n/b)** successive steps of the algorithm. At each step **k**,

- Lines 5-9 describe vertical communications related to matrix A. Only processors from the same row of the processor grid send each other elements of matrix A. **(100.×b×b×(n/(b×p))/TCOMM(…))** 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 macro **TCOMM**(I,J,K,L,p,q) returns the total volume of data in bytes transferred between processors with coordinates **(I,J)** and **(K,L)** during the execution of the PDGETRF routine. 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 10-14 describe horizontal communications related to matrix B. Only processors from the same column of the processor grid send each other elements of matrix B. **(100.×b×b×(n/(b×q))/TCOMM(…))** 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 15-17 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.×2×b×b×b×(n/(b×p))×(n/(b×q))/tcomp[i×q+j])** percent of the volume of computations it performs during the execution of the algorithm. The array reference **tcomp[i×q+j]** returns the total volume of computations performed by the processor with coordinates **(i,j)** during the execution of the PDGETRF routine. 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 complete performance model descriptions of the routines PDGETRF, PDGETF2 and PDLASWP can be

```
   int main(int argc, char **argv) {
      int nprow, npcol, pdgetrfctxt, myrow, mycol, c__0 = 0, LLD_a;
/* Problem parameters */
      int  *M, *N, *IA, *JA, *DESCA, *IPIV, INFO;
      double *A,;
/* Initialize the heterogeneous ScaLAPACK runtime */
      hscal_init(&argc, &argv);
/* Get the heterogeneous PDGETRF context */
      hscal_pdgetrf_ctxt(M, N, IA, JA, DESCA, &pdgetrfctxt);
      if (!hscal_in_ctxt(&pdgetrfctxt))
         hscal_finalize(c__0);
/* Retrieve the process grid information */
      Cblacs_gridinfo(pdgetrfctxt, &nprow, &npcol, &myrow, &mycol);
/* Initialize the array descriptor for the matrix A */
      descset_(DESCA, ..., &pdgetrfctxt, &LLDa);  /* for Matrix A */
/* Distribute matrices on the process grid using user-defined pdmatgen */
      pdmatgen_(&pdgetrfctxt, ...); /* for Matrix A */
/* Call the SCALAPACK 'pdgetrf' routine */
      pdgetrf_(M, N, A, IA, JA, DESCA, IPIV, INFO);
/* Release the heterogeneous PDGETRF context */
      hscal_free_ctxt(&pdgetrfctxt);
/* Finalize the Heterogeneous ScaLAPACK runtime */
      hscal_finalize(c__0);
   }
```

Figure 2. Basic steps involved in calling the heterogeneous ScaLAPACK routine `PDGETRF`.

studied from the files `pm_pdgetrf.mpc`, `pm_pdgetf2.mpc` and `pm_pdlaswp.mpc` in the directory `/SRC` of the Heterogeneous ScaLAPACK package. The performance models of the PBLAS routines PDTRSM and PDGEMM can be found in the directory `/PBLAS/SRC`.

2.2 Model of the Heterogeneous ScaLAPACK Program

Figure 2 shows the essential steps involved in calling the ScaLAPACK PDGETRF routine in a Heterogeneous ScaLAPACK program. These are:

1. Initialize the heterogeneous ScaLAPACK runtime using using the operation
   ```
   int hscal_init(int * argc,
                  int *** argv)
   ```
   where `argc` and `argv` are the same as the arguments passed to `main`. This routine must be called before any other Heterogeneous ScaLAPACK context management routine and must be called once. It must be called by all the processes running in the Heterogeneous ScaLAPACK application;
2. Get the heterogeneous PDGETRF context using the context constructor routine **`hscal_pdgetrf_ctxt`**. The function call **`hscal_in_ctxt`** returns a value of 1 for the processes chosen to execute the PDGETRF routine or otherwise 0;
3. Execute the homogeneous ScaLAPACK PDGETRF routine;
4. Release the context using the context destructor operation
   ```
   int hscal_free_ctxt(int * ctxt);
   ```
5. When all the computations have been completed, the program is exited with a call to **`hscal_finalize`**, which finalizes the heterogeneous ScaLAPACK runtime.

The execution of the library PDGETRF context constructor routine **`hscal_pdgetrf_ctxt`** 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. In this case, the benchmark code performs 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 determined based on the problem size solved;
2. Finding the optimal values of the parameters of the parallel algorithm used in the ScaLAPACK 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 ScaLAPACK 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 [8,9]. It is based on the performance model of the ScaLAPACK routine and the performance model of the executing network of computers, which reflects the state of this network just before the execution of the ScaLAPACK 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 Heterogeneous ScaLAPACK program uses the multiprocessing approach, which allows more than one process involved in its execution to be run on each processor. The number of processes to run on each processor during the program startup is determined automatically by the Heterogeneous ScaLAPACK command-line interface tools. During the creation of a HeteroMPI group 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. At the same time, the mapping algorithm invoked tries to arrange the processors along a 2D grid so as to optimally load balance the work of the processors.

3. Experimental Results

The set of experiments is run on a small moderately heterogeneous local network of sixteen Linux workstations (hcl01-hcl16) whose specifications can be studied 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 [11].

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.

The speedup, which is shown in the figures, is calculated as the ratio of the execution time of the homogeneous ScaLAPACK program and the execution time of the HeteroScaLAPACK program. Dense square matrices of size N×N were used in the experiments. The homogeneous ScaLAPACK programs use the default parameters recommended by 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.

Figure 3 shows the execution times of the sequential LAPACK [12], ScaLAPACK and HeteroScaLAPACK programs solving the same LU factorization problem. The LAPACK library employs optimized BLAS library (ATLAS). The LAPACK program is executed on the fastest processor hcl06. For problem sizes (N<=3072), there are no benefits using ScaLAPACK. This means that just a single processor (hcl06) can be used for solving the LU factorization problem. However for the problem sizes beyond (N>3072), the HeteroScaLAPACK and ScaLAPACK programs perform significantly better with HeteroScaLAPACK being the best for reasons that are explained below. The LAPACK program starts paging for problem sizes (N>11264) and exhibits severe performance degradation.

Figures 4(a) and 4(b) show the experimental results from the execution of the ScaLAPACK and HetroScaLAPACK programs employing the routines PDGETRF and PDPOTRF. The ScaLAPACK program uses a 4×4 grid of processes (using one process per node configuration) adhering to the recommendations provided in the ScaLAPACK's user guide. Figure 4 (a) shows results for problem sizes where ScaLAPACK programs do not page and Figure 4(b) for problem sizes where ScaLAPACK programs page. For PDGETRF at around problem size (N=18432),

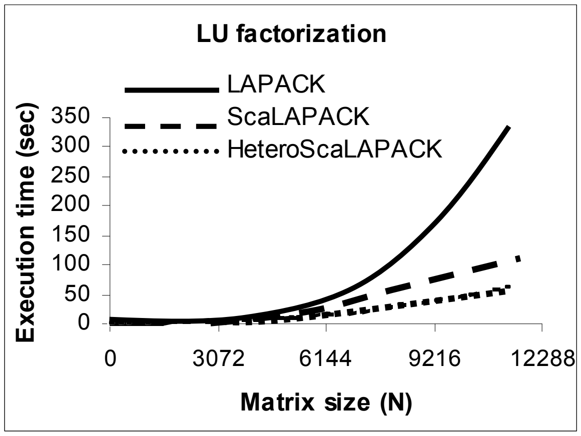
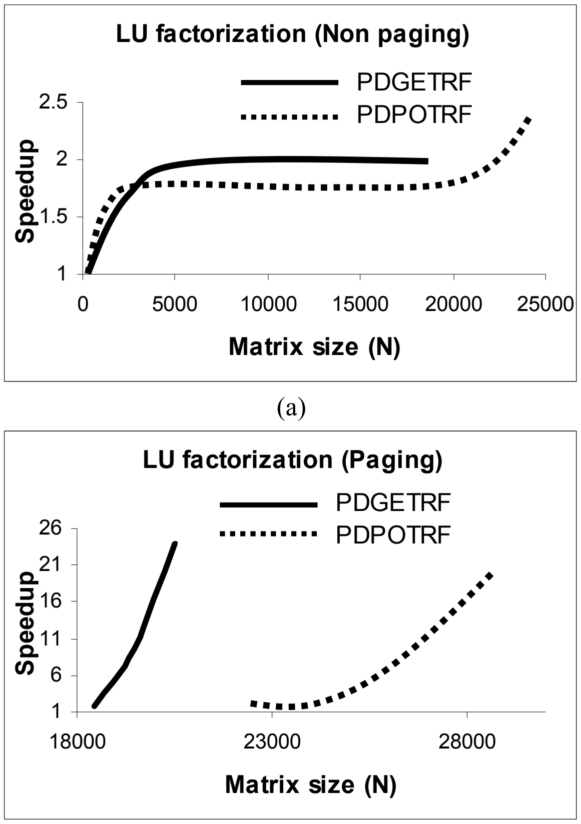


Figure 3. Execution times of LAPACK, ScaLAPACK and HeteroScaLAPACK programs solving the same LU factorization problem.

ScaLAPACK programs start paging leading to very poor performance. For PDPOTRF, paging starts happening around problem size (N=24576). The average speedups of HeteroScaLAPACK programs over ScaLAPACK programs for non-paging problem sizes are are 1.9 and 1.7 respectively. Similar speedups are obtained for the other LU factorization routines supporting different datatypes (PCGETRF, PSGETRF, PZGETRF, PCPOTRF, PSPOTRF and PZPOTRF). The ScaLAPACK programs failed for problem sizes beyond 20480 for PDGETRF and 28672 for PDPOTRF due to the problem sizes not fitting into one or more nodes used for the experiments.

There are a few reasons for the good speedups delivered by the Heterogeneous ScaLAPACK programs on HCCs for all problem sizes. 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.

Because the largest fraction of the work takes place in the update of the trailing matrix A_{22}, therefore, to obtain maximum parallelism all processors should participate in its update. Since A_{22} reduces in size as the computation progresses, a block cyclic data distribution is used to ensure that at any stage A_{22} is evenly distributed over all processors, thus obtaining their balanced load. Since



(b)

Figure 4. Speedup of Heterogeneous ScaLAPACK over ScaLAPACK. (a) Problem sizes where ScaLAPACK programs do not page and (b) Problem sizes where ScaLAPACK programs page.

the distribution of work becomes uneven as the computation progresses, a larger block size results in greater load imbalance, but reduces the frequency of communication between processors. There is, therefore, a tradeoff between load imbalance and communication startup cost which can be controlled by varying the block size, *b*. An optimal block size of 64 is used by the HeteroScaLAPACK library.

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 processors have more work to do between synchronization points than others. This is the case here due to partial pivoting being performed over rows in a single column of the processor grid while the other processors are idle. Similarly, the evaluation of each block row of the U matrix requires the solution of a lower triangular system across processors in a single row of the processor grid. 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

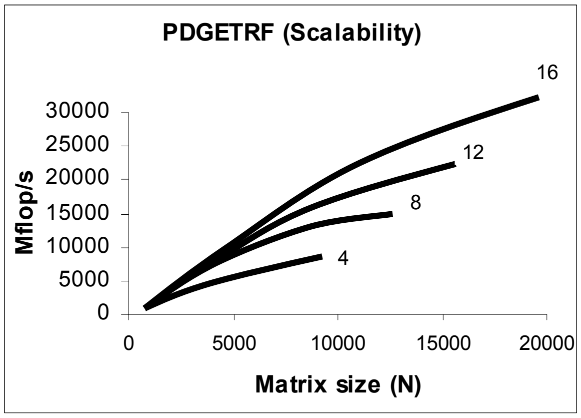


Figure 5. Performance of HeteroScaLAPACK PDGETRF as function of matrix size for different numbers of processors.

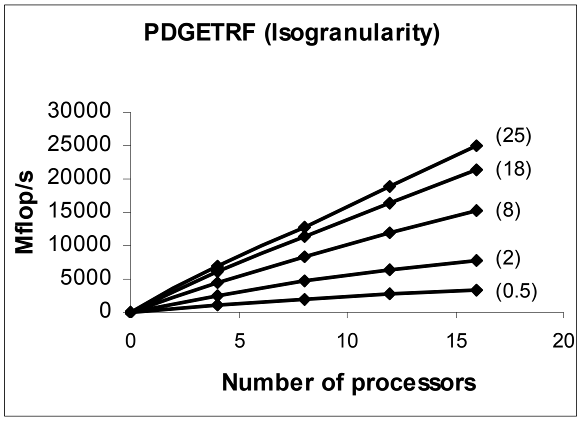


Figure 6. Isogranularity curves. The numbers in brackets represent the granularity in Mbytes per processor.

each process arrangement evaluated. For each such estimation, it invokes 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. There is thus an optimal aspect ratio, p/q, which depends on the communications characteristics of the network and determines the overlap of the communication with the computation. The optimal aspect ratios observed were in the range (1/5,1).

The performance results in Figures 5 and 6 is used to assess the scalability of the HeteroScaLAPACK factorization routine PDGETRF. Measured execution times are converted to million flop/s by assuming an operation count of $2N^3/3$, where N is the matrix size. The heterogeneity of the experimental network is maintained constant by retaining the fastest and slowest processors hcl06 and hcl09 in each set of processors. Figure 6 shows the isogranularity plots where the efficiency is investigated by observing how the performance per processor degrades as the number of processors increases for a fixed grain size, which is N^2/N_p where N_p is the number of processors. The scalability is assessed by the extent to which the isogranularity curves differ from linearity. The near-linearity of these plots show that the factorization routines are quite scalable on this network.

4. Conclusions and Future Work

We have presented the details of implementation of the LU factorization routines in the Heterogeneous ScaLAPACK library, which provides a subset of optimized LAPACK routines for heterogeneous computational clusters. Our future work would involve the writing of performance models of the routines for the factorization and solution of dense system of linear equations (PXYYSV) in ScaLAPACK. These would employ the performance models of the LU factorization routines described in this paper.

REFERENCES

[1] Heterogeneus ScaLAPACK. http://hcl.ucd.ie/project/HeteroScaLAPACK/.
[2] Scalable LAPACK. http://www.netlib.org/scalapack/.
[3] Parallel Basic Linear Algebra Subprograms (PBLAS). http://www.netlib.org/scalapack/pblas_qref.html.
[4] Basic Linear Algebra Communication Subprograms (BLACS). http://www.netlib.org/blacs/.
[5] Basic Linear Algebra Subprograms (BLAS). http://www.netlib.org/blas/.
[6] 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.
[7] A. Lastovetsky and R. Reddy, "Data distribution for dense factorization on computers with memory heterogeneity," Parallel Computing, Volume 33, No. 12, pp.757-779, December 2007.
[8] A. Lastovetsky, "Adaptive Parallel Computing on Heterogeneous Networks with mpC," Parallel Computing, Volume 28, No.10, pp.1369-1407, October 2002.
[9] 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.
[10] R. Reddy and A. Lastovetsky, "HeteroMPI+ScaLAPACK: Towards a ScaLAPACK (Dense Linear Solvers) on Heterogeneous Networks of Computers," Proceedings of the 13th IEEE International Conference on High Performance Computing (HiPC 2006), Bangalore, India, LNCS Volume 4297, pp.242-253, December 2006.
[11] Automatically Tuned Linear Algebra Software (ATLAS). http://math-atlas.sourceforge.net/.
[12] Linear Algebra PACKage (LAPACK). http://www.netlib.org/lapack/.