HeteroMPI+ScaLAPACK: Towards a ScaLAPACK (Dense Linear Solvers) on Heterogeneous Networks of Computers

Ravi Reddy[1] and Alexey Lastovetsky[2]

[1] GS Laboratory Private Limited, Pune, India
ravi@gs-lab.com
[2] School of Computer Science and Informatics, University College Dublin, Belfield, Dublin 4, Ireland
Alexey.Lastovetsky@ucd.ie

Abstract. The paper presents a tool that ports ScaLAPACK programs designed to run on massively parallel processors to Heterogeneous Networks of Computers. The tool converts ScaLAPACK programs to HeteroMPI programs. The resulting HeteroMPI programs do not aim to extract the maximum performance from a Heterogeneous Networks of Computers but provide an easy and simple way to execute the ScaLAPACK programs on such networks with good performance improvements. We demonstrate the efficiency of the resulting HeteroMPI programs by performing experiments with a matrix multiplication application on a local network of heterogeneous computers.

1 Introduction

In this paper, we present a tool, which ports conventional parallel programs that are designed to run on *massively parallel processors* (MPP) such as Scalable Linear Algebra Package (ScaLAPACK) programs [1] to Heterogeneous Message Passing Interface (HeteroMPI) programs [2] for Heterogeneous Networks of Computers (HNOCs). The resulting HeteroMPI programs do not aim to extract the maximum performance from a heterogeneous network but provide an easy and simple way to execute the conventional parallel programs on HNOCs with good performance improvements. Before we describe the details of the porting procedure, we present briefly the ScaLAPACK and HeteroMPI packages.

ScaLAPACK is a well-known standard package of high-performance linear algebra routines for distributed-memory message passing MIMD computers and networks of workstations supporting PVM [3] and/or MPI [4]. It is a continuation of the LAPACK project [5], which designed and produced analogous software for workstations, vector supercomputers, and shared-memory parallel computers. Both libraries contain routines for solving systems of linear equations, least squares problems, and eigenvalue problems.

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

Y. Roberts et al. (Eds.): HiPC 2006, LNCS 4297, pp. 242 – 253, 2006.

The first step in this process of automation is the specification of the performance model of the heterogeneous parallel algorithm in a performance model definition language. Performance model is a tool supplied to the programmer to specify his or her high-level knowledge of the application in a generic form. This knowledge is used by the HeteroMPI runtime system to find the most efficient implementation of the heterogeneous parallel algorithm on HNOCs.

The second step involves the writing of a HeteroMPI application. A typical HeteroMPI application consists of the following steps:

1. Accurate determination of the platform parameters using HeteroMPI characterization API;
2. Optimal data partitioning using HeteroMPI data partitioning API. This step of heterogeneous decomposition is parameterized by the platform parameters determined in the first step;
3. Determination of the optimal algorithmic parameters using HeteroMPI estimation API;
4. Efficient mapping of processes to the computers of the executing heterogeneous network. HeteroMPI group management operations automate this step.
5. Finally the execution of the HeteroMPI program using the HeteroMPI's command line interface.

The tool that we present in this paper mainly assists scientists trying to port their homogeneous parallel algorithms to HNOCs. It is usually a difficult design task to come up with a practical and efficient heterogeneous counterpart of a homogeneous parallel algorithm on HNOCs. The problem of optimal heterogeneous data distribution has proved to be NP-complete even for such a simple linear algebra kernel as matrix multiplication on HNOCs [6]. Once the heterogeneous parallel algorithm is designed, its portable and efficient implementation on heterogeneous platforms requires writing of a lot of complex code to automate several tedious and error-prone tasks [7]. The scientists can use this tool for porting their homogeneous parallel algorithms for HNOCs without any rewriting or redesigning. *It can be seen as a first step towards the realization of a ScaLAPACK for HNOCs.*

The tool takes two inputs. The first input is a ScaLAPACK program containing the homogeneous parallel algorithm that solves the problem on MPPs. The other input is the performance model of the homogeneous parallel algorithm employed in the ScaLAPACK program described in HeteroMPI's performance model definition language. It generates a HeteroMPI program, which uses a multiprocessing algorithm consisting of the following steps:

- The whole computation is partitioned into a large number of equal chunks;
- Each chunk is performed by a separate process;
- More than one process is allowed to be run on each processor. During the creation of a HeteroMPI group of processes, 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 relative 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 during the creation of a HeteroMPI group of processes, the mapping algorithm invoked tries to arrange the processors along a 2D grid so as to optimally load balance the work of the processors.

We start with literature survey on the multiprocessing approaches to solving parallel problems and proposals for heterogeneous ScaLAPACK. Then we describe the details of the porting procedure of the ScaLAPACK programs to HeteroMPI programs. This is followed by experimental results with a matrix multiplication application on a local network of heterogeneous computers demonstrating the efficiency of the resulting HeteroMPI programs. We conclude the paper by outlining our future research goals.

2 Literature Survey

The section surveys related papers from the literature. The papers surveyed are mainly: papers presenting proposals for heterogeneous ScaLAPACK and papers presenting multiprocessing approaches to solve parallel problems on HNOCs.

Beaumont *et al*. [8] discuss data allocation strategies to implement matrix products and dense linear system solvers on heterogeneous computing platforms 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. They formally state the optimization problem to be solved and present both an exact solution (with exponential cost) and a heuristic solution.

Kalinov and Lastovetsky [9] analyze two strategies:

- HeHo - heterogeneous distribution of processes over processors and homogeneous block 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 [10, 11]. The first strategy is implemented using calls to ScaLAPACK; the second strategy is implemented with calls to LAPACK and BLAS [12]. 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 that 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 essential 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 [13] 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 [14] 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.

3 Porting a Legacy ScaLAPACK Program

This section is divided into three sub-sections. We start with the legacy ScaLAPACK program that is to be ported. This is followed by description of the homogeneous parallel algorithm used in the ScaLAPACK program in HeteroMPI's performance model definition language. In the second sub-section, we explain the structure of the HeteroMPI program output by the porting procedure. Finally we explain the issues involved in the porting procedure and how they are resolved.

3.1 Inputs

There are two inputs provided to the tool. The first input is the ScaLAPACK program computing matrix multiplication using the routine **`PDGEMM`**. There are four basic steps involved in calling a ScaLAPACK routine. The reader is directed to the ScaLAPACK users' guide [15] for more details.

The second input is the performance model definition **`pdgemm`** of the matrix multiplication routine **`PDGEMM`**. HeteroMPI allows application programmers to describe a performance model of their implemented homogeneous algorithm. This model allows specification of all the main features of the underlying parallel algorithm that have an essential impact on application execution performance on HNOCs. 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, and
- The order of execution of the computations and communications by the involved parallel processes in the group, that is, how exactly the processes interact during the execution of the algorithm.

HeteroMPI provides a small and dedicated model definition language for specifying this performance model. This language uses most of the features in the specification of network types of the mpC language. A compiler compiles the description of this

```
/* 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/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 */ };
```

Fig. 1. Specification of the performance model of the homogeneous algorithm employed by **PDGEMM** in the HeteroMPI's performance definition language

performance model to generate a set of functions. The functions make up an algorithm-specific part of the HeteroMPI runtime system.

The tool takes as input the performance model definition **pdgemm** shown in Figure 1. This performance model definition describes the simplest scenario performed by the **pdgemm** routine in ScaLAPACK, which uses outer-product algorithm using the *logical LCM hybrid algorithmic blocking* strategy [16]. The performance model definition describes the parallel matrix-matrix multiplication of two dense square matrices *A* and *B* of size **n×n**. The distribution blocking factor **b** used in the matrix-matrix multiplication is assumed to be equal to the algorithmic blocking factor. The performance model definition also assumes that the matrices are divided into whole number of blocks of size equal to distribution blocking factor, that is, (**n**%(**b**×**p**)) and (**n**%(**b**×**q**)) (see explanation of variables below) are both equal to zero.

The reader is referred to [11,17] for explanation of the main constructs, namely **coord**, **parent**, **node**, **link**, and **scheme**, used in a description of a performance

```
int main(int argc, char **argv) {
    static int p, q, n, t, input_p, output_p;
    int* mdlparams;
    HMPI_Group gid;
    HMPI_Init(&argc, &argv);
    // Estimation of speeds of the processors
    if (HMPI_Is_member(HMPI_PROC_WORLD_GROUP)
       HMPI_Recon(&dgemm, &input_p, 2, &output_p);
    // Model parameter initialization
    if (HMPI_Is_host())
       mdl_params[0] = n; mdl_params[1] = 64; mdl_params[2] = t;
    // HMPI Group creation
    if (HMPI_Is_host())
        HMPI_Group_heuristic_auto_create(&gid, &HMPI_Model_pdgemm,
                                         &hfunc, mdl_params);
    if (HMPI_Is_free())
        HMPI_Group__heuristic_auto_create(&gid, &HMPI_Model_pdgemm,
                                          NULL, NULL);
    // Execution of the algorithm
    if (HMPI_Is_member(&gid)) {
       MPI_Comm algocomm = *(MPI_Comm*)HMPI_Get_comm(&gid);
       HMPI_Group_topology(&gid, &nd, &dp);
       p = dp[0]; q = dp[1]; // optimal process grid arrangement
       ictxt = Csys2blacs_handle(algocomm);
       //Legacy ScaLAPACK program pdgemm code using ictxt
    }
    // HMPI Group Destruction
    if (HMPI_Is_member(&gid))
       HMPI_Group_free(&gid);
    HMPI_Finalize(0);
}
```

Fig. 2. The most relevant fragments of generated HeteroMPI code computing matrix-matrix multiplication using **PDGEMM** on heterogeneous networks

model. Briefly, 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*. It is assumed that the benchmark code multiplies two **b×t** and **t×b** matrices. Parameter **b** is the size of the distribution blocking factor. 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 coordinate system to which the processor nodes of the network are related. Line 4 is a *node declaration*. It relates the virtual processors to the coordinate system declared and specifies the (absolute) volume of computations to be performed by each of the processors. Lines 5-13 are a *link declaration*. This specifies the links between the virtual processors, the pattern of communication among the abstract processors, and the total volume of data to be transferred between each pair of virtual processors during the execution of the algorithm. Line 14 is a *parent declaration*. It specifies the coordinates of the parent processor node in a given coordinate system. Line 15 introduces the *scheme declaration*. The **scheme** block describes how exactly virtual processors interact during the execution of the algorithm.

3.2 Target HeteroMPI Program

The HeteroMPI program shown in Figure 2 resulting from the porting procedure performs typically the following steps:

1. The initialization of HeteroMPI runtime using the function **HMPI_Init**;
2. This is followed by dynamic refreshment of the estimation of the processor speeds using the characterization function **HMPI_Recon**. The benchmark code used in the call to **HMPI_Recon** is a serial BLAS version of the parallel ScaLAPACK routine. In this case, the BLAS routine **dgemm** multiplying two dense matrices is used to dynamically refresh the processor speeds. The benchmark code allocates, multiplies, and frees two **b×t** and **t×b** matrices where **b** is the distribution blocking factor and **t** is is equal to the size of the matrix used in the parallel application divided by the square root of the total number of processes that are available for computation. This is a heuristic used because some of the processes may not be chosen by the mapping algorithm employed by the HeteroMPI group constructor function (presented subsequently) to participate in the execution the parallel application.
3. Creation of a HeteroMPI group of processes using the group management function **HMPI_Group_auto_create** to obtain a handle to the HeteroMPI group of MPI processes. This function detects the optimal number of processes that can execute the parallel application, that is, finds the optimal arrangement of processes in a grid. During the creation of a HeteroMPI group of processes, the mapping of the parallel processes in the group is performed such that the number of processes running on each processor is proportional to its speed. At the same time, the processors are arranged along the 2D grid **p×q** so as to optimally load balance the work of the processors. The mapping algorithm is explained in detail in [11]. Since the number of 2D process grid arrangements is large, the HeteroMPI program uses the HeteroMPI function **HMPI_Group_heuristic_auto_create** instead of the HeteroMPI function **HMPI_Group_auto_create**, which evaluates all the possible 2D process grid arrangements. The function **HMPI_Group_heuristic_auto_create** uses heuristics to reduce the number of process arrangements to evaluate. The design and implementation of the HeteroMPI group constructor functions are explained in detail in [17];
4. The function **HMPI_Group_heuristic_auto_create** returns an HeteroMPI handle to the group of MPI processes in **gid**. The second parameter **HMPI_Model_pdgemm** is a handle that encapsulates all the features of the performance model. These features are in the form of a set of functions generated by the compiler from the description of the performance model. The third parameter **hfunc** is a heuristic function used to reduce the number of 2D process arrangements to evaluate. The fourth parameter **mdl_params** is an input parameter to the performance model, which consists of problem size to be solved, the algorithmic blocking factor used (which is equal to the distribution blocking factor) and the size of matrix used in the benchmark code. The only input provided by the application programmer is the problem size to be solved;
5. Conversion of the handle to the HeteroMPI group of MPI processes obtained previously to an MPI communicator using the function call **HMPI_Get_comm**;
6. Conversion of the MPI communicator to an integer BLACS handle, which can be passed into grid creation routine. This is done using the interim BLACS routine **Csys2blacs_handle**;

7. Creation of the BLACS context using the integer BLACS handle. This is done using the interim BLACS routine **Cblacs_gridinit**;
8. The legacy ScaLAPACK code is then executed using the BLACS context obtained;
9. This is followed by freeing the group using operation **HMPI_Group_free** and the finalization of HeteroMPI runtime system using operation **HMPI_Finalize**.

It can be seen that the HeteroMPI program automates the most tedious and error-prone tasks that are involved in porting a homogeneous parallel application.

3.3 Porting Issues

There are three important issues to be considered in the porting procedure.

1. The total number of processes to be allocated to each participating computer when the user starts up the application. Some basic rules to choose the number of processes to allocate per each processor can be followed:
- First of all, the number of processes running on each computer should not be less than the number of processors of the computer just to be able to exploit all the available processor resources. So the lower bound on the number of processes to be run on a computer is given by the number of processors on the computer.
- The upper bound on the number of processes executed on each processor is roughly equal to the ratio of speed of the fastest processor to speed of the slowest processor on the executing network of computers.
2. The blocking factor used to distribute the rows and the columns of the matrices involved in the computation. It is observed that for a process arrangement, execution times are the same no matter what algorithmic blocking factor is used. However to ensure efficient data distribution, ScaLAPACK [15] recommends that any blocking factor between 32 to 64 be used to distribute the rows and the columns of the matrices involved in the computation of the linear algebra kernel. The tool uses a value of 64;
3. The optimal arrangement of processes in the grid. This is determined by the HeteroMPI group constructor functions **`HMPI_Group_auto_create`** or **`HMPI_Group_heuristic_auto_create`**.

4 Experimental Results

A local network of 15 different heterogeneous Linux workstations hcl01 to hcl15 is used in the experiments. The computers used in the experiments are connected to communication network, which is based on 2 Gbit Ethernet with a switch enabling parallel communications between the computers. The experimental results are obtained by averaging the execution times over a number of experiments. Figure 3 shows the experimental results using the routine **`pdgemm`** performing parallel matrix-matrix multiplication on this heterogeneous network. The speedup calculated is the ratio of the execution time of the ScaLAPACK program over the execution time of the HeteroMPI+ScaLAPACK program. The reader is referred to [17] for details on the execution of the HeteroMPI program using HeteroMPI's command line interface.

Table 1. Optimal process grid arrangements (**p**,**q**) detected by the HeteroMPI group constructor function **`HMPI_Group_heuristic_auto_create`**. **`n`** is the size of the matrix. The third column gives the time taken to refresh the speeds of the processors at runtime. The fourth column gives the time taken to evaluate the process arrangements during the creation of the HeteroMPI group of processes that would execute the parallel application. The last column gives the execution time of the parallel application.

N	(p,q)	Processor speed update time (sec)	HeteroMPI Group creation time (sec)	Execution time (sec)
1024	(4,2)	0.09	1.29	17
2048	(8,2)	0.10	2.59	20
3072	(6,3)	0.21	1.38	21
4096	(8,2)	0.26	1.32	26
5120	(10,2)	0.31	2.70	30
6144	(6,3)	0.37	7.02	41
7168	(7,2)	0.44	1.76	53
8192	(8,2)	0.51	2.83	69
9216	(9,2)	0.58	5.33	100
10240	(10,2)	0.67	7.85	138
11264	(11,2)	0.76	6.36	215
12288	(12,2)	0.88	48.19	266
13312	(13,2)	1.18	10.73	312
14336	(14,2)	3.41	23.64	354
15360	(15,2)	8.97	54.65	405
16384	(16,2)	11.78	34.83	513
17408	(17,2)	14.28	23.99	772
18432	(18,2)	24.15	100.94	956
19456	(19,2)	30.49	32.45	1323
20480	(8,4)	33.59	41.97	2063

The absolute speeds of the processors are obtained based on serial version **`dgemm`** of the corresponding parallel routine **`pdgemm`**. The absolute speeds in million floating point operations per second (MFlop/s) is obtained by multiplication of two dense 1536×1536 matrices for the processors. The absolute speeds are {2171, 2099, 1761, 1787, 1735, 1653, 1879, 1635, 3004, 2194, 4580, 1762, 4934, 4096, 2697}. It can be seen that the fastest processor is *hcl13* and the slowest processor is *hcl08*. It should be noted that a process is run per processor to obtain these measurements. The ratio of

absolute speed of the fastest processor to the absolute speed of the slowest processor is 4934/1635 = 3. This is the number of processes run on each processor in the network during the execution of the parallel application. So the total number of processes available to the HeteroMPI+ScaLAPACK program for computation is 25×3 = 75 since there are 25 processors in the network. The HeteroMPI+ScaLAPACK program detects the optimal process grid arrangement from the set of all possible 2D process grid arrangements of 75 processes in a reasonable amount of time as presented in Table 1. The number of possible 2D process arrangements can be calculated to be 338 (using the formula m×(1+1/2+1/3+…+1/m) where m=75). The ScaLAPACK program uses a 5×5 grid of processes (using one process per processor configuration).

Table 1 shows the optimal process grid arrangements determined by the HeteroMPI group constructor functions for the problem sizes experimented. The second column gives the optimal process grid arrangements for the problem sizes shown in first column. The third column gives the time taken to refresh the speeds of the processors at runtime during the **HMPI_Recon** function call. The fourth column gives the time taken to evaluate the process arrangements during the creation of the HeteroMPI group of processes using the **HMPI_Group_heuristic_auto_create** function call. This time varies due to different number of process arrangements evaluated for a given values of **n** and **b**. The last column gives the execution time of the parallel application. It includes the processor speed update time and the group creation time. It can be seen that the processor speed refreshment time and the group creation time are much less than the actual execution time of the parallel application. The function **HMPI_Group_heuristic_auto_create** uses heuristics to reduce the number of 2D process grid arrangements (**p**,**q**) to evaluate. One such heuristic used is that one-dimensional process arrangements where either **p** or **q** or both is equal to 1 are not evaluated.

The speedups of the HeteroMPI+ScaLAPACK program over ScaLAPACK program for these problem sizes are shown in Figure 3. As can be seen from the results, the resulting HeteroMPI programs deliver good performance improvements on HNOCs for problem sizes beyond 12288. There are two reasons for such good speedups observed. 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, 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. It can be seen that for problem sizes larger than 12288, 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 ScaLAPACK program, good load balance is not achieved. However just running more than 25 processes in the execution of the ScaLAPACK program would not resolve the problem. This is because in such a case the optimal process arrangement and the efficient mapping of the process arrangement to the executing computers of the underlying network must also be determined. This is a complex task automated by HeteroMPI.

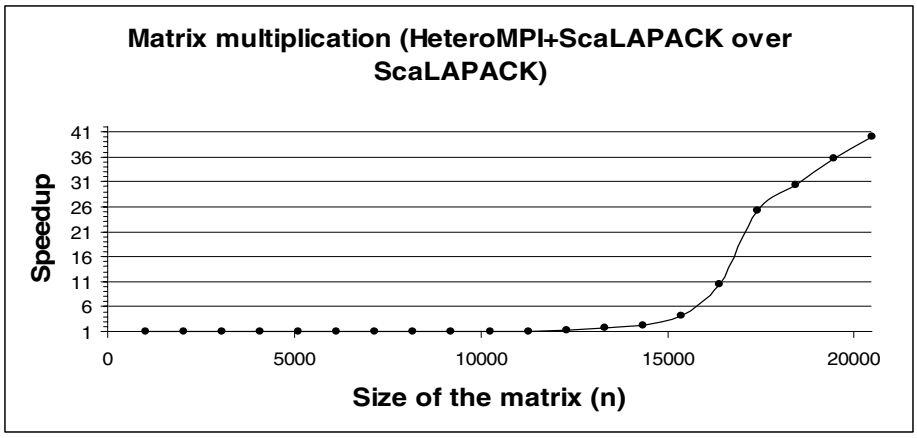

Fig. 3. Speedup of the HeteroMPI+ScaLAPACK program over the ScaLAPACK program employing matrix-matrix multiplication using the routine **pdgemm**

The second reason is the optimal 2D grid arrangement of processes. During the creation of a HeteroMPI group of processes, the function **`HMPI_Group_heuristic_auto_create`** estimates the time of execution of the algorithm for 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.

5 Conclusions and Future Work

In this paper, we have presented a tool that ports ScaLAPACK programs to heterogeneous platforms. The tool converts the ScaLAPACK programs to HeteroMPI programs. These HeteroMPI programs do not aim to extract the maximum performance from a heterogeneous network but provide an easy and simple way to execute the conventional parallel programs on HNOCs with good performance improvements. We have taken the first step towards the realization of a heterogeneous ScaLAPACK for HNOCs. Our future work will involve the development of a Heterogeneous ScaLAPACK library, which will include dense linear solvers of ScaLAPACK redesigned for HNOCs. The design and implementation of this library will include: (a) Design of performance models for each of the level-1, level-2, and level-3 PBLAS routines; (b) Design of performance models for each of the dense linear solvers of ScaLAPACK routines.

References

[1] Blackford, L., Choi, J., Cleary, A., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.: ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers – Design Issues and Performance. In: Proceedings of the 1996 ACM/IEEE Supercomputing Conference. IEEE Computer Society, CD-ROM/Abstracts Proceedings, Pittsburgh PA USA (1996)

[2] Lastovetsky, A., Reddy. R: HeteroMPI: Towards a Message-Passing Library for Heterogeneous Network of Computers. Journal of Parallel and Distributed Computing 66 (2006) 197-220
[3] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V. S.: PVM: Parallel Virtual Machine, Users' Guide and Tutorial for Networked Parallel Computing. The MIT Press: Cambridge, MA (1994)
[4] Dongarra, J., Huss-Ledermann, S., Otto, S., Snir, M., Walker, D.: MPI: The Complete Reference. The MIT Press (1996)
[5] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hmmarling, S., McKinney, A., Ostrouchov, S., Sorenson, D. LAPACK Users' Guide. Release 1.0, SIAM, Philadelphia (1992)
[6] Beaumont, O., Boudet, V., Rastello, F., Robert, Y.: Matrix Multiplication on Heterogeneous Platforms. IEEE Transactions on Parallel and Distributed Systems 12 (2001) 1033-1051
[7] Lastovetsky, L.: Scientific Programming for Heterogeneous Systems - Bridging the Gap between Algorithms and Applications. In: Proceedings of the 5th International Symposium on Parallel Computing in Electrical Engineering (PARELEC 2006), IEEE Computer Society Press (2006)
[8] Beaumont, O., Boudet, V., Petitet, A., Rastello, F., Robert, Y.: A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers). IEEE Transactions on Computers 50 (2001) 1052-1070
[9] Kalinov, A., Lastovetsky, A.: Heterogeneous Distribution of Computations Solving Linear Algebra Problems on Networks of Heterogeneous Computers. Journal of Parallel and Distributed Computing 61 (2001) 520-535
[10] Lastovetsky, A., Arapov, A. Kalinov, A., Ledovskih, I.: A Parallel Language and Its Programming System for Heterogeneous Networks. Concurrency: Practice and Experience 12 (2000) 1317-1343
[11] Lastovetsky, A.: Adaptive parallel computing on heterogeneous networks with mpC. Parallel Computing 28 (2002) 1369-1407
[12] Dongarra, J., Croz, J.D., Duff, I.S., Hammarling, S.: A set of level-3 basic linear algebra subprograms. ACM Transactions on Mathematical Software 16 (1990) 1-17
[13] Kishimoto, Y., Ichikawa, I.: 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)
[14] Kalinov, A., Klimov, S.: Optimal mapping of a parallel application processes onto heterogeneous platform. In: 4th Heterogeneous Computing Workshop (HCW 2005), Proceedings of 19th International Parallel and Distributed Processing Symposium (IPDPS'05), IEEE Computer Society (2005)
[15] Blackford, L., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.: ScaLAPACK User's Guide. SIAM (1997)
[16] Petitet, A., Dongarra, J.: Algorithmic Redistribution Methods for Block-Cyclic Decompositions. IEEE Transactions on Parallel and Distributed Systems 10 (1999) 1201-1216
[17] Reddy, R.: HeteroMPI: A Message Passing Library for Heterogeneous Networks of Computers. PhD Dissertation, University College Dublin, Dublin, Ireland (2005)