2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing

# Experimental Study of Six Different Implementations of Parallel Matrix Multiplication on Heterogeneous Computational Clusters of Multicore Processors

Pedro Alonso
Department of Information Systems and Computation,
Polytechnic University of Valencia
Cno. Vera s/n, 46022 Valencia, Spain
palonso@dsic.upv.es

Ravi Reddy and Alexey Lastovetsky
School of Computer Science and Informatics,
University College Dublin
Belfield, Dublin 4, Ireland
{manumachu.reddy,alexey.lastovetsky}@ucd.ie

***Abstract*—Two strategies of distribution of computations can be used to implement parallel solvers for dense linear algebra problems for Heterogeneous Computational Clusters of Multicore Processors (HCoMs). These strategies are called Heterogeneous Process Distribution Strategy (HPS) and Heterogeneous Data Distribution Strategy (HDS). They are not novel and have been researched thoroughly. However, the advent of multicores necessitates enhancements to them. In this paper, we present these enhancements. Our study is based on experiments using six applications to perform Parallel Matrix-matrix Multiplication (PMM) on an HCoM employing the two distribution strategies.**

***Keywords- Heterogeneous ScaLAPACK; HeteroMPI; multicore clusters; matrix-matrix multiplication; heterogenous clusters***

## I. INTRODUCTION

Parallel platforms employing multicores are becoming dominant systems in High Performance Computing (HPC). Almost 90% of the supercomputing systems in the Top500 list are based on dual- or quad-core architectures [1]. This rapid widespread utilization of multicore processors is due to several factors [2]. Therefore, computers containing multicore processors will become ubiquitous soon and will be widely deployed in clusters purposely built to tackle the most challenging scientific and engineering problems. A cluster built from such computers (HCoM), will be inherently heterogeneous due to the different number of cores/processors, its processing capabilities and the multilevel hierarchy of interconnected sets of them. Therefore, the advent of multicores poses many challenges to writing parallel solvers for dense linear algebra problems for an HCoM. Addressing these challenges would entail redesign and rewriting of parallel algorithms to take into account the increased TLP (Thread Level Parallelism) and the hierarchical nature of communications satisfying the criteria of fine granularity (as cores are associated with relatively small local memories) and asynchronicity to hide the latency of memory accesses. Algorithms hitherto considered unscalable for being communication-intensive or due to high granularity have to be revisited. These criteria can be satisfied when an algorithm can generate a set of independent tasks having a high ratio of floating point calculations to data required, that is, all the tasks involved are of Level 3 BLAS.

These solvers must take into account the aforementioned heterogeneities and provide "scalable" parallelism where speedups obtained are proportional to the number of cores as one scales from 4-16-128 and more cores. They must be written using hybrid programming models (e.g. MPI [3] plus OpenMP [4]). These solvers must also be automatically tuned for an HCoM, which means that they must automate the following complex optimization tasks [5]: the accurate estimation of platform parameters (speeds of processors, latencies and bandwidths of communication links, etc.); the use of efficient communication models that would reflect the hierarchical nature of communications and would accurately predict the time of different types of communications; the determination of the optimal values of algorithmic parameters such as data distribution blocking factor and two-dimensional processor grid arrangement; and, finally, an efficient mapping of the processes executing the parallel algorithm to the computers.

In this paper, we propose and analyse two strategies of distribution of computations:

- *Heterogeneous Process Distribution Strategy (HPS)*: In this strategy, more than one process is executed per computer. All the processes get the same amount of data. The number of processes on the computer multiplied by the number of threads run per process is equal to the number of cores in the computer. Given a problem size, there exist an optimal number of processes to be executed on a computer and an optimal number of threads to run per process. This is a different definition from the traditional HeHo strategy [6] since the number of processes executed per computer in the HPS strategy will be always equal to the number of cores. Therefore, the HeHo condition of proportionality of the number of processes run per processor to its speed is relaxed.

1066-6192/10 $26.00 © 2010 IEEE
DOI 10.1109/PDP.2010.52

- *Heterogeneous Data Distribution Strategy (HDS)*: In this strategy, one process is executed per computer (the computer may have one or more processors). The volume of data allocated to a computer is proportional to the speed of the computer. The number of threads run per computer will be equal to the number of cores in the computer to ensure that all the cores are fully utilized. It should also be noted the terms computer or process or processor are used interchangeably using this strategy, which is why in our modified definition we consider a computer with one or more processors (multicore or not) as a single entity, that is, one process is executed per computer even though the computer may have one or more processors (multicore or not).

The HPS strategy is a multiprocessing approach that is used to accelerate legacy parallel linear algebra programs on HCoMs. It allows the complete reuse of high-quality legacy parallel linear algebra software such as ScaLAPACK [7] on HCoMs with no redesign efforts and provides good speedups. The Heterogeneous ScaLAPACK library [8], currently under development, uses this strategy and is built on the top of HeteroMPI [9] and ScaLAPACK. It provides automatically tuned parallel linear algebra programs for HCoMs but most importantly performs all the aforementioned critical automations of the complex optimization tasks.

The rest of the document is organized as follows. The next section briefly outlines six applications used in this study to perform the Parallel Matrix-matrix Multiplication (PMM). Sections III to V introduce in more detail three of them, specifically designed and implemented for this study. Section VI presents the results of experiments with the applications. Section VII presents some conclusions and future work.

## II. Applications Description

### A. Homogeneous ScaLAPACK Application Using HPS

This application calls the PDGEMM routine of the PBLAS subproject, which implements the parallel outer-product algorithm of two dense matrices on a 2D process grid [7]. This routine falls in the set of HPS applications in our study.

### B. MPI Application Using HDS

The heterogeneous parallel algorithm [10] used to compute this matrix product is a modification of the ScaLAPACK outer-product algorithm. This application requires, as input, the 2D computer grid arrangement to use during the execution of the PMM. More details will follow in Section III.

### C. HeteroMPI Application Using HDS

This application calls the HeteroMPI routines to determine the optimal values of the algorithmic parameters: the subset of computers used for computations and their 2D arrangement. Afterwards, the selected computers that form the optimal 2D computer grid perform the heterogeneous PMM described in Section B. More details will be presented in Section IV.

### D. MPI Application Using HPS

This application requires two inputs, which are the number of threads to run per process and the 2D process grid arrangement to use during the execution of the PMM. It uses homogeneous distribution of computations, that is, each process gets the same amount of data. It reuses the code of the MPI application utilizing the HDS strategy (Section B) for the particular case of homogeneous data distribution among processes.

### E. HeteroMPI Application Using HPS

This application reuses the code of the HeteroMPI application utilizing the HDS strategy (Section C) with some exceptions. It uses homogeneous distribution of computations, that is, each process gets the same amount of data. The number of threads per process must be preconfigured. This is due to a shortcoming in HeteroMPI, which is the feature that would detect the optimal (process, thread) combination in the HPS strategy. This application is the application presented in Section D instrumented with HeteroMPI.

### F. Heterogeneous ScaLAPACK Application Using HPS

This application is written using Heterogeneous ScaLAPACK routines and reuses the PBLAS routine PDGEMM. The number of threads to run per process must be preconfigured. This application can be considered as the one shown in Section A (ScaLAPACK) instrumented with HeteroMPI. However, HeteroScaLAPACK provides users with additional tools to facilitate the interface with HeteroMPI. A summary of the software is presented in Section V.

## III. The MPI Application Using HDS

This section presents the PMM application multiplying matrix $A$ and matrix $B$, $C=A\times B$, where $A$, $B$, and $C$ are dense matrices of size $m\times k$, $k\times n$, and $m\times n$ matrix elements respectively on a 2D heterogeneous processor grid of size $p\times q$, $P_{ij}, \forall i \in [1,p] \wedge j \in [1,q]$ provided as input. Each matrix element is a square block of size $b\times b$. The heterogeneous parallel algorithm [10] used to compute this matrix product is a modification of the ScaLAPACK outer-product algorithm. One process is executed per computer even though the computer may have one or more processors (multicore or not).

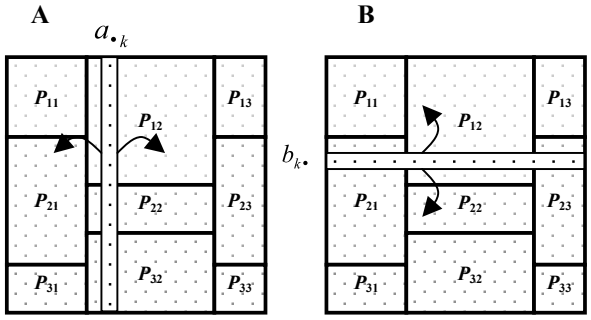



Figure 1. One step of the h eterogeneous PMM on a 2D processor grid of 3×3. First, each $b\times b$ block of the pivot column $a_{\bullet k}$ of matrix A (emitting the curly arrows) is broadcast horizontally, and each $b\times b$ block of the pivot row $b_{k\bullet}$ of matrix B (emitting the curly arrows) is broadcast vertically. Then, each $b\times b$ block $c_{ij}$ of matrix C is updated, $c_{ij} = c_{ij} + a_{ik} \times b_{kj}$.

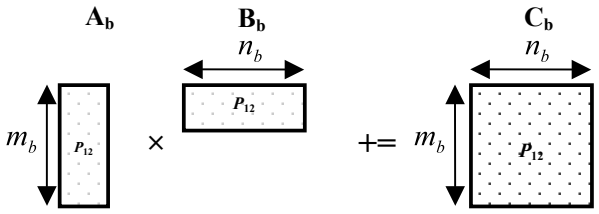



Figure 2. The computational kernel (shown here for processor $P_{12}$ for example) performs a matrix update of a dense matrix $C_b$ of size $m_b \times n_b$ using $A_b$ of size $m_b \times 1$ and $B_b$ of size $1 \times n_b$. The matrix elements represent b×b matrix blocks.

To perform the PMM, matrices *A*, *B*, and *C* are divided into rectangles such that there is one-to-one mapping between the rectangles and the computers, and the area of each rectangle is proportional to the speed of the processor owning it (Fig. 1). The procedure of data distribution invokes the data partitioning algorithm [6], [10], which determines the optimal 2D column-based partitioning of a dense matrix of size $m \times n$ over a 2D heterogeneous processor grid of size $p \times q$. The area is partitioned into uneven rectangles so that they are arranged into a 2D grid of size $p \times q$ and the area of a rectangle is proportional to the speed of the processor owning it. The inputs to the procedure are: the rectangular area size ($m \times n$), the 2D processor grid ($p \times q$) and the absolute speeds (represented with a single number) of the processors. The outputs are the heights and the widths of the rectangles. This procedure is described in [11].

For this application, the core computational kernel performs a matrix update of a matrix $C_b$ of size $m_b \times n_b$ using $A_b$ of size $m_b \times 1$ and $B_b$ of size $1 \times n_b$ as shown in Fig. 2. The size of the problem is represented by $m_b$ and $n_b$. We use a combined computation unit, which is made up of one addition and one multiplication to express the volume of computation. Thus, the total number of computation units (namely, multiplications of two $b \times b$ matrices) performed during the execution of the benchmark code will be approximately equal to $m_b \times n_b$. The absolute speed of the processor exposed by the application when solving the problem of size ($m_b$,$n_b$) can be calculated as $m_b \times n_b$ divided by the execution time of the matrix update.

## IV. HeteroMPI Application Using HDS

The application presented in this section is composed of two parts. First, it calls the HeteroMPI routines to determine the optimal values of the algorithmic parameters, which are the computers to be used in the execution of the PMM and their 2D arrangement. Then, the computers of the optimal 2D computer grid perform the heterogeneous PMM explained in Section III. Again, one process is executed per computer even though the computers may have one or more processors (multicore or not).

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

The first step in this process of automation is the writing of the *performance model* of the parallel algorithm, the PMM algorithm in our case. Performance model is a tool supplied to the programmer to specify its high-level knowledge of the main features of the underlying parallel algorithm that impact the execution performance in order to assist in finding the most efficient implementation 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;
- The total volume of data to be transferred between each pair of processes in the group;
- The order of execution of the computations and communications by the parallel processes in the group, that is, how exactly the processes interact.

HeteroMPI provides a dedicated performance model definition language (PMDL) for writing this *performance model*. The model and the PMDL are borrowed from the mpC programming language [12], [13]. The PMDL compiler compiles the performance model written in PMDL to generate a set of functions, which make up the algorithm-specific part of the HeteroMPI runtime system. These functions are called by the mapping algorithms of HeteroMPI runtime system to estimate the execution time of different configurations of the parallel algorithm. The HeteroMPI runtime system solves the problem of selection of the optimal set of processes running on different computers using the mapping algorithms explained in [9] and [13]. HeteroMPI considers the executing heterogeneous network as a multilevel hierarchy of interconnected sets of heterogeneous multiprocessors [13]. The mapping algorithms use an estimation of platform parameters: speed of processors and communication links. The speed of each processor is characterized by the execution time of a serial code provided by the application programmer (benchmark code) and it is supposed to be representative for the computations. The code is performed at runtime at points of the application specified by the application programmer. The communication model is seen as a hierarchy of homogeneous communication layers characterized by the latency and bandwidth. Unlike the speed model of the processors, the communication model is static, a shortcoming that would be addressed in our future work. Its parameters are obtained during the initialization of the HeteroMPI runtime and are not refreshed later.

The optimal values of the algorithmic parameters that HeteroMPI allows to determine are: the data distribution blocking factor and the 2D processor grid arrangement. The performance model of PMM and the estimation procedure are explained in detail in [11].

## V. Heterogeneous ScaLAPACK Application Using HPS

This section presents the Heterogeneous ScaLAPACK application, which utilizes the HPS strategy. The high-level building blocks of Heterogeneous ScaLAPACK package [8] are HeteroMPI and ScaLAPACK (Fig. 3). The principal routines in Heterogeneous ScaLAPACK package are the *context creation functions* for the ScaLAPACK routines (which include the PBLAS routines as well). There is a context creation function for each and every ScaLAPACK routine. It provides

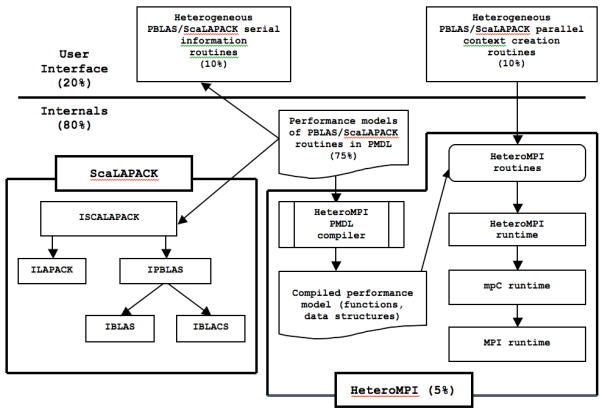



Figure 3. Flow of the Heterogeneous ScaLAPACK context creation routine call. The percentages give the breakup of Heterogeneous ScaLAPACK development efforts.

a context for the execution of the ScaLAPACK routine but most importantly, performs the critical work of automating the difficult optimization tasks.

Context creation routines keep the interface of the corresponding LAPACK routine except the pointers to data matrices are not passed as arguments. These routines return a handle to a HeteroMPI group of processes that is subsequently used in the actual execution of the computational routine. The Heterogeneous ScaLAPACK context creation and destruction routines call functions of HeteroMPI runtime system. The Heterogeneous ScaLAPACK information functions calculate the total number of floating-point operations performed by each process and the total number of communications in bytes between each pair of processes involved in the execution of the homogeneous ScaLAPACK routine. These routines are serial and can be called by any process. The block ISCALAPACK ('I' standing for instrumented) in Fig. 3 represents the instrumented code of ScaLAPACK, which reuses the existing code base of ScaLAPACK completely. The performance model definitions contain the instrumented code components. The HeteroMPI compiler compiles this performance model to generate a set of functions. During the creation of the context, the mapping algorithms of HeteroMPI runtime system call these functions to estimate the execution time of the ScaLAPACK routine. With this estimation, the context constructor routine determines the optimal values of the algorithmic parameters such as the 2D process grid arrangement and efficient mapping of the processes.

The Heterogeneous ScaLAPACK program uses the multiprocessing HPS strategy, 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.

## VI. Experimental Results

A small local heterogeneous cluster (Rosebud) consisting of multicore computers, SMPs, and single-processor workstations is used in the experiments. The specifications of this cluster are shown in Table I. Nodes *R01* and *R02* are single-processor workstations, nodes *R03* and *R04* are SMPs with two processors each, nodes *R05* and *R06* are computers with four Itanium dual-core processors, and *R07* and *R08* are computers with two Itanium dual-core processors. All the computers are running Linux OS. *R01* to *R04* have 32-bit OS whereas the rest have 64-bit OS. The communication network is based on 1 Gbit Ethernet. The software used is OpenMPI-1.2.8, ScaLAPACK-1.8.0, HeteroMPI-1.2.0, Heterogeneous ScaLAPACK-1.0.6-BETA, Intel *mkl* 9.0 (which includes a multithreaded version of BLAS) and Intel *icc* 9.1.

For all the applications utilizing the HDS strategy, the number of threads configured to run per node or computer is the number of cores (number of processors for the SMP nodes). The heterogeneity of the network due to the heterogeneity of the computers is calculated as the ratio of the absolute speed of the fastest computer to the absolute speed of the slowest computer. The absolute speed of the computers is obtained by performing a local DGEMM update of two matrices $2048\times99$ and $99\times2048$ where 99 is the optimal blocking factor. As one can see, *R05* and *R06* are the fastest computers and *R01* and *R02* are the slowest. The heterogeneity in this case is $\approx 15$. If we exclude the computers *R01* and *R02*, the heterogeneity would be $\approx 5$.

TABLE I. Specifications of the eight computers in the Rosebud cluster

| Node name | Main memory (kB) | No. of procs. | No. of cores | HDS | |
|---|---|---|---|---|---|
| | | | | No. of threads | Absolute speed (Mflops) |
| R01 | 1035492 | 1 | - | 1 | 2295 |
| R02 | 1035688 | 1 | - | 1 | 2295 |
| R03 | 3635424 | 2 | - | 2 | 6515 |
| R04 | 3635424 | 2 | - | 2 | 6515 |
| R05 | 8240240 | 4 | 8 | 8 | 34600 |
| R06 | 8240512 | 4 | 8 | 8 | 34600 |
| R07 | 8240528 | 2 | 4 | 4 | 19130 |
| R08 | 8240672 | 2 | 4 | 4 | 19130 |

The number of threads shown in Table I have been obtained by running the sequential matrix-matrix multiplication (level-3 BLAS routine DGEMM) with different problem sizes on each node. There are two trends that can be observed in the execution performance [11]. The first trend concerns problem sizes before the computer starts paging. For these problem sizes, the execution performance of the applications reduces when the number of threads exceeds the number of processors (in the case of single-processor workstation or SMP) or the number of cores in the case of a multicore computer. The sec-

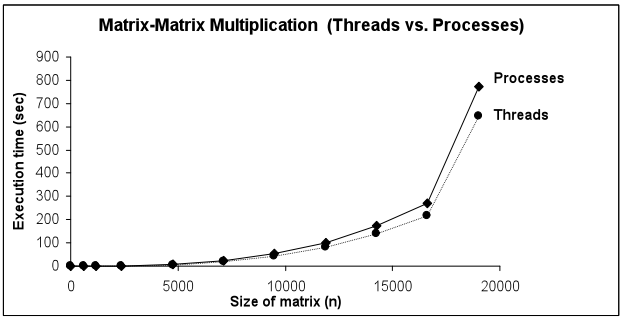



Figure 4. The solver using threads is more efficient than the parallel solver on a computer performing the same matrix-matrix multiplication.

ond trend relates to the execution performance in the area of paging. It can be concluded that there is no definite rule to use for the optimal number of threads except that the number of threads to run per process must be greater than the number of processors or the number of cores.

Inside a multicore node, our study [11] shows that it is more efficient to use a sequential matrix-matrix multiplication routine executed with the number of threads set equal to the number of cores than to use a MPI application with the number of processes equal to the number of cores (each process running one thread) as shown in Fig. 4. This justifies the feature of the HDS strategy, which is that a computer must be considered as a single entity instead of each of its processors for the distribution of computations. In addition, the solution space of 2D processor grid arrangements to evaluate will increase enormously if the (processor, thread) combinations need to be considered as is the case in the original definition of the strategy. Therefore, by treating a computer as a single entity, we eliminate this complexity.

One approach investigated in [15] determines the optimal blocking factor for each computer, which can be a drawback. This is because a single value of the blocking factor must be used as input to the routines in the legacy linear algebra packages (this is a interface requirement). Modifying this approach to determine the single optimal value to use in the parallel application is not trivial. As a result of our experiments, we saw that the optimal values of the blocking factor are in the range 54<=*b*<=144. We use the value of 99 in the experiments. This procedure to determine the best value of the blocking factor (as explained in Section IV) is executed separately and before the execution of all the parallel applications. So ideally this procedure must be executed during the installation of the software (as is done for the Heterogeneous ScaLAPACK).

For all the experimental results shown in the tables, the processes are arranged in the 2D grid in decreasing order of their speeds along each process row and along each process column. For example, a 2×4 computer grid arrangement containing all the computers shown in Table I will have the computers arranged as follows: *{R05, R06, R07, R08, R04, R03, R02, R01}*. Similarly, a 3×8 process grid arrangement involving the computers {*R05, R06, R07, R08*} shown in Table I, the number of processes run per computer being 8, 8, 4, 4, respectively, will have the processes arranged as follows: {*R05*, *R05*, *R05*, *R05*, *R05*, *R05*, *R05*, *R05*, *R06*, *R06*, *R06*, *R06*, *R06*, *R06*, *R06*, *R06*, *R07, R07, R07, R07, R08, R08, R08, R08}*.

TABLE II. MPI-HDS AND HETEROMPI APPLICATION UTILIZING THE HDS STRATEGY (EXECUTION TIME S IN SECONDS)

| Size of the matrix (n) | MPI-HDS | | HeteroMPI-HDS | | | |
|---|---|---|---|---|---|---|
| | p×q | Exec. time | Predict. p×q | Predict. exec. time | Group creation time | Actual time |
| 594 | 1×1 | 0.03 | 1×1 | 0.04 | 0.1 | 0.15 |
| 1188 | 1×1 | 0.12 | 1×1 | 0.18 | 0.4 | 0.52 |
| 2376 | 1×1 | 1 | 1×1 | 1 | 1 | 2 |
| 4752 | 1×1 | 5 | 1×1 | 6 | 2 | 7 |
| 7128 | 1×1 | 18 | 1×1 | 19 | 3 | 21 |
| 9504 | 1×1 | 42 | 1×1 | 44 | 4 | 46 |
| 11880 | 1×1 | 81 | 1×1 | 85 | 5 | 86 |
| 14256 | 2×2 | 133 | 2×2 | 131 | 6 | 140 |
| 16632 | 2×2 | 191 | 2×2 | 191 | 8 | 201 |
| 19008 | 2×2 | 250 | 2×2 | 263 | 8 | 260 |
| 21384 | 2×2 | 350 | 2×2 | 347 | 9 | 360 |
| 23760 | 2×2 | 453 | 2×2 | 452 | 11 | 467 |
| 26136 | 2×2 | 577 | 2×2 | 576 | 11 | 593 |
| 28512 | 2×2 | 756 | 2×2 | 757 | 12 | 790 |
| 30888 | 3×2 | 1006 | 3×2 | 1010 | 13 | 1048 |
| 33264 | 3×2 | 1203 | 3×2 | 1225 | 15 | 1460 |
| 35640 | 3×2 | 2091 | 3×2 | 2167 | 15 | 2993 |
| 38016 | 3×2 | 3423 | 3×2 | 3552 | 17 | 4766 |
| 40392 | 3×2 | 5575 | 3×2 | 5773 | 18 | 7156 |

The experimental results of the MPI-HDS application are shown in the second and third columns in Table II. The second column shows the optimal 2D computer grid arrangements. The data distribution used is column-based [6], [10]. The number of threads configured to run per node is shown in Table I. The execution of the MPI-HDS application consists of two parts. First, all the computers execute the benchmark code, whose time of execution is used to estimate their speeds, and then they execute the column-based data partitioning algorithm to partition the matrices. Second, they perform the PMM.

TABLE III. EXECUTION TIMES OF MPI-HDS APPLICATION (IN SECONDS)

| Size of matrix (n) | MPI-HDS | | | | | |
|---|---|---|---|---|---|---|
| | 2×2 | | 3×2 | | 4×2 | |
| | DPA time | PMM Time | DPA time | PMM Time | DPA time | PMM time |
| 28512 | 11 | 745 | 12 | 852 | 32 | 896 |
| 30888 | 10 | 1006 | 14 | 1006 | 18 | 1155 |
| 33264 | 11 | 1607 | 19 | 1203 | 220 | 1285 |
| 35640 | 13 | 3251 | 19 | 2091 | 810 | 2143 |
| 38016 | 15 | 4863 | 30 | 3423 | 1196 | 3442 |
| 40392 | 22 | 8857 | 29 | 5575 | 1365 | 5673 |

Table III shows the execution time of the first part (DPA) and the second part (PMM). The interesting computer arrangement to observe in this table is 4×2 where all the computers of the network are used. It can be seen that for problem sizes exceeding 30888, times spent in the execution of the first part (DPA) are large. This is because the slow computers *R01*

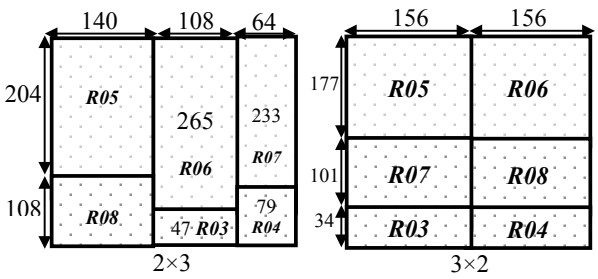



Figure 5. N=30888. The column-based distribution for the arrangement 3×2 is actually cartesian whereas that of the arrangement 2×3 is non-cartesian. The number of communications (horizontal+vertical) at each step of the PMM for the arrangements {2×3, 3×2} is {24,18}. The matrix elements are square blocks of size 99×99.

and *R02* start paging as the problem size exceeds 30888 and spend large times executing the benchmark code. The optimal data distribution determined however does not include the slow computers and as a result the processor arrangement used in the PMM is 3×2.

It has also been observed that the 2D computer grid arrangements {3×2, 4×2} perform better than {2×3, 2×4}, respectively. The reason why the column-based data distribution for 3×2 is efficient is because the matrix data distribution is actually cartesian (each processor has only two neighbours) as shown in Fig. 5. The column-based matrix data distribution for 2×3 is not cartesian, which results in more communications.

We performed further experiments to investigate the two matrix data distributions {column-based, cartesian}. The best performing 2D computer arrangements is 3×2 with both the column-based and the cartesian data distribution. In fact, the data distribution for the best performing 2D computer arrangement that is {3×2, column-based} is actually cartesian. The 2D computer grid arrangement {2×3, column-based} performs poorly due to non-cartesian distribution resulting in large number of communications. The 2D computer grid arrangement {2×3, cartesian} performs poorly due to load imbalance resulting from non-optimal matrix data distribution. It is very difficult to achieve the proportionality (volume of data to the speed of the processors) in the case of cartesian data distribution.

The experimental results highlight the importance of a tool that can provide features that can determine the optimal values of the algorithmic parameters such as the total number of computers and the 2D computer grid arrangement. HeteroMPI is one such tool. The experimental results of the HeteroMPI-HDS application are shown in Table II. The second column shows the best execution times and the 2D computer grid arrangements from the MPI-HDS application. These are compared with the predictions of HeteroMPI. The results of the HeteroMPI-HDS application in the third column are organized as follows. The first sub-column shows the optimal 2D computer grid arrangement predicted by HeteroMPI. The second sub-column shows the predicted time of execution of the PMM. The third sub-column shows the time taken by HeteroMPI group constructor routine to evaluate all the possible 2D computer grid arrangements and to arrive at the best 2D computer grid arrangement. Last sub-column shows the total execution time of the HeteroMPI-HDS application. This includes the time taken to execute the benchmark code and the time taken to determine the best 2D computer grid arrangement. The predictions of HeteroMPI are spot-on and the predictions of the execution times are accurate within 10%.

TABLE IV. SPEEDUP OF MPI-HDS OVER SEQUENTIAL MATRIX-MATRIX MULTIPLICATION

| Size of matrix (n) | Speedup |
|---|---|
| 594 to 11880 | 1 |
| 14256 | 1.04 |
| 16632 | 1.14 |
| 19008 | 2.3 |
| 21384 | 5.4 |
| 23760 | 10 |

Table IV shows the speedup of the MPI-HDS application over the sequential matrix-matrix multiplication application run on *R05*. The number of threads run in the sequential application is equal to the number of cores, which is equal to 8.

Table V shows the best time obtained with the different applications using the HPS strategy. For the MPI-HPS application, there will be an optimal (process, thread) combination for each problem size. Combination (2×2, 2t) involves 4 processes in *R05* with each process running 2 threads. Combination (2×2, 4t) involves one process per computer (*R05*-*R08*) running 4 threads each. As the problem size increases, it can be seen that the optimal number of threads per process increases from 2 to 4 due to the role played by communications.

For the HeteroMPI-HPS application, the execution time shown in Table V (col. 3) includes the execution of the benchmark code, the evaluation of the possible 2D process grid arrangements and the PMM computation. At the moment,

TABLE V. BEST CONFIGURATION (PROCESSOR ARRANGEMENT, NUMBER OF THREADS) FOR FOUR HPS APPLICATIONS TO PERFORM PMM. EXECUTION TIMES IN SEC.

| Size of the matrix (n) | MPI-HPS | Hetero-MPI-HPS | ScaLA-PACK-HPS | HeteroSca-LAPACK |
|---|---|---|---|---|
| 4752 | 7 (2×2, 2 t) | 4 (2×2, 2 t) | 12 (2×8, 1t) | 28 (5×6, 1t) |
| 7128 | 22 (2×2, 2 t) | 26 (2×2, 2 t) | 34 (3×8, 1t) | 77 (3×9, 1t) |
| 9504 | 52 (2×2, 2 t) | 58 (2×2, 2 t) | 64 (3×8, 1t) | 80 (2×14, 1t) |
| 14256 | 149 (2×2*, 4 t) | 157 (2×4, 2 t) | 163 (3×8, 1t) | 246 (4×7, 1t) |
| 16632 | 221 (2×2*, 4 t) | 236 (2×4, 2 t) | 245 (3×8, 1t) | 380 (2×14, 1t) |
| 23760 | 545 (2×2*, 4 t) | 572 (2×4, 2 t) | 610 (3×8, 1t) | 789 (4×7, 1t) |
| 30888 | 1090 (2×2*, 4 t) | 1205 (3×4, 2 t) | 1290 (3×8, 1t) | 1557 (2×14, 1t) |
| 33264 | 1320 (2×2*, 4 t) | 1668 (2×2, 4 t) | 1819 (3×8, 1t) | 1966 (2×14, 1t) |
| 40392 | 4355 (2×2*, 4 t) | 5537 (2×2, 4 t) | 8224 (7×4, 1t) | 4851 (4×7, 1t) |

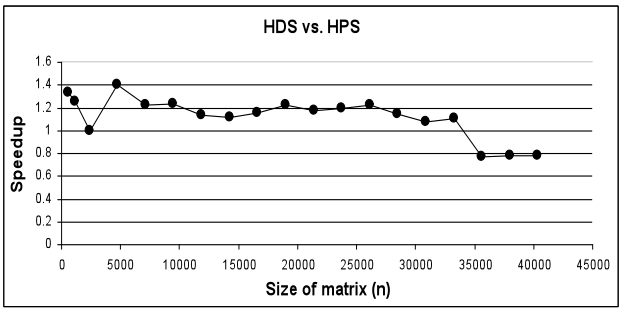



Figure 6. Speedup of the MPI application utilizing the HDS strategy over the MPI application utilizing the HPS strategy.

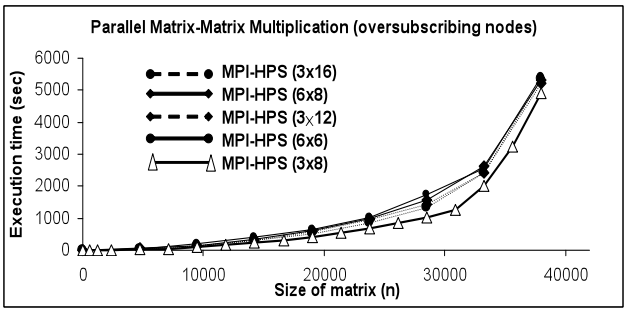
the number of threads must be pre-configured. The prediction of the 2D process grid arrangement is as correct as in the case of the HeteroMPI-HDS algorithm. The prediction of the execution time is not so. One of the reasons could be the inaccuracy of the communication model used for the shared-memory communications between the processes inside a computer. This issue is currently under investigation.

The application performing the worst is ScaLAPACK with the configuration where one process is run per core and one thread is run per process. The 2D process grid arrangements for the MPI-HPS shown in Table IV are the following (number of processes in brackets): (2×2): *R05* (2), *R06* (2); (2×2)*: *R05* (1), *R06* (1), *R07* (1), *R08* (1).

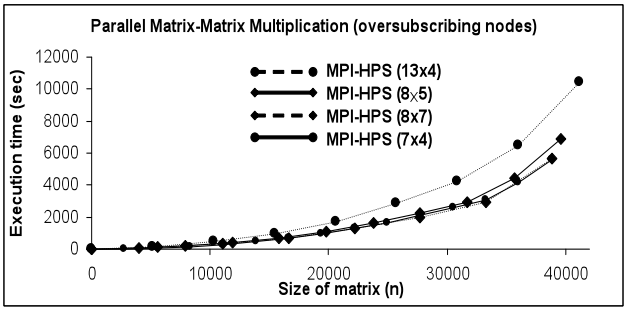
The (process, thread) combinations for the 2D process grid arrangements for the ScaLAPACK-HPS application shown in Table V are the following: (2×8): *R05* (8,1), *R06* (8,1); (3×8): *R05* (8,1), *R06* (8,1), *R07* (4,1), *R08* (4,1); (7×4): *R05* (8,1), *R06* (8,1), *R07* (4,1), *R08* (2,1), *R03* (2,1), *R04* (2,1).

In the experiments with the Heterogeneous ScaLAPACK application, all the computers are used in the execution of the application. The slowest computers *R01* and *R02* are however not picked during the execution of the PMM. Given the number of threads per process is preconfigured as input to the application, the Heterogeneous ScaLAPACK runtime would then determine the optimal number of processes and the optimal 2D process arrangement. In this case, the best execution times are achieved with only one thread per process.

Fig. 6 shows the speedup of the MPI-HDS application over the MPI-HPS application calculated as the ratio of the execution time of MPI-HPS application to the execution time of MPI-HDS application. The results reveal that the two strategies can compete with each other. For the range of problem sizes ($n \leq 35640$), the MPI applications employing HDS perform the best since they fully exploit the increased thread-level parallelism (TLP) provided by the multicore processors. However, for large problem sizes, the non-cartesian nature of the data distribution may lead to excessive communications that can be very expensive. For such cases, the HPS strategy has been shown to out-perform the HDS strategy. Fig. 7 shows the effect of oversubscribing the multicore computers and SMPs (running more processes than cores or processors). The number of threads run per process on each computer is 1. Our experiments allow us to conclude that once the number of processes exceeds the number of cores in the computer, the




(a)




(b)

Figure 7. Execution times of the MPI application utilizing the HPS strategy. (a). The multicore computers are oversubscribed. (b). The multicore and the SMP machines are oversubscribed.

execution performance of the application goes down before the region of paging ($n \leq 40000$). The computers can be oversubscribed in the region of paging. The 2D process grid arrangements shown in the Fig. 7(a) are the following with the number of processes in brackets: 3×16: *R05*(16), *R06*(16), *R07*(8), *R08*(8); 6×8: *R05*(16), *R06*(16), *R07*(8), *R08*(8); 3×12: *R05*(12), *R06*(12), *R07*(6), *R08*(6); 6×6: *R05*(12), *R06*(12), *R07*(6), *R08*(6); 3×8: *R05*(8), *R06*(8), *R07*(4), *R08*(4). The 2D process grid arrangements shown in the Fig. 7(b) are the following with the number of processes in brackets: 13×4: *R05*(16), *R06*(16), *R07*(8), *R08*(8), *R03*(2), *R04*(2); 8×5: *R05*(12), *R06*(12), *R07*(6), *R08*(6), *R03*(2), *R04*(2); 8×7: *R05*(16), *R06*(16), *R07*(8), *R08*(8), *R03*(4), *R04*(4); 7×4: *R05*(8), *R06*(8), *R07*(4), *R08*(4), *R04*(2), *R03*(2). The grid arrangements 3×8 and 7×4 (Fig. 7(a) and (b), respectively) are just shown for comparison. There still remains a problem of how many processes to run per computer and which 2D process grid arrangement to use. The problem becomes more complicated when threads are taken into account. This necessitates the importance of a tool, which can determine the optimal values of these algorithmic parameters automatically.

## VII. Conclusions and Future Work

In this paper, we presented enhancements to two pre-existing strategies of distribution of computations for HCoMs: HPS and HDS, to implement parallel solvers for dense linear algebra problems. We performed experiments using six applications utilizing the various distribution strategies to perform the PMM on a local HCoM. The results revealed that the two strategies can compete with each other. The conclusions to be

drawn from these results are:

- The HDS strategy is the best strategy to use since it allows to fully exploit the increased thread-level parallelism (TLP) provided by the multicore processors. However, for large problem sizes, the non-cartesian nature of the data distribution may lead to excessive communications that can be very expensive. For such cases, the HPS strategy has been shown to outperform it.
- HeteroMPI is a valuable tool to implement heterogeneous parallel algorithms on HCoMs using the HDS strategy since it accurately predicts the execution time of the parallel algorithm, accurately detects the optimal values of the algorithmic parameters such as the total number of processors and the 2D processor grid arrangement.
- The software (HeteroMPI, Heterogeneous ScaLAPACK) must be enhanced to provide accurate predictions of the optimal (process, thread) combination in the case of the HPS strategy. Since Heterogeneous ScaLAPACK is built on the top of HeteroMPI, the feature only needs to be added to HeteroMPI.
- No package is currently available using HDS strategy. A package based on this strategy requires great effort redesigning and reimplementing the linear algebra routines and writing the associated performance models. Heterogeneous ScaLAPACK is however completely implemented except for the eigenvalue solvers. It uses the legacy ScaLAPACK and its associated performance models are written. In addition, the HeteroMPI-HPS strategy followed by Heterogeneous ScaLAPACK have been shown to be quite competitive to the HDS strategy in single core processor clusters [16], [17], [18] as well as multicore processor clusters.
- HeteroMPI has been proven to be a valuable tool in single core processor clusters [9] by oversubscribing a fast processor with a number of processes proportional to its speed. From the analysis of the results of this paper, this can be accomplished too in multicore processor clusters but with one limitation: the total number of processes per node cannot be larger than the number of cores of the node, otherwise, the performance falls down drastically due to resource contention of the concurrent processes and loss of some aggregate computational computation power of the heterogeneous cluster. More investigation needs to be done with incoming growth in the number of cores per node.

Our future work would involve addition of features to the software (HeteroMPI, Heterogeneous ScaLAPACK) to determine the optimal (process, thread) combination in the HPS strategy. We would also look at improvements to the communication models in the software, which would accurately predict the time of different types of communications.

## ACKNOWLEDGEMENT

We acknowledge the support of the VIDI of the Polytechnic University of Valencia, the Generalitat Valenciana and the Spanish Research Project TIN2008-06570-C04-02. The study was also in part supported by the Science Foundation Ireland.

## REFERENCES

[1] Ranking of supercomputers according to the LINPACK benchmark. http://www.top500.org.

[2] J. Dongarra, D. Gannon, G. Fox, and K. Kennedy, "The Impact of Multicore on Computational Science Software," CTWatch Quarterly, Volume 3, No. 1, February 2007.

[3] The Message Passing Interface Standard. http://www-unix.mcs.anl.gov/mpi/.

[4] An API for multi-platform shared-memory parallel programming in C/C++ and Fortran. http://openmp.org/wp/.

[5] A. Lastovetsky, "Scientific Programming for Heterogeneous Systems - Bridging the Gap between Algorithms and Applications," Proceedings of the 5th International Symposium on Parallel Computing in Electrical Engineering (PARELEC 2006), Bialystok, Poland, IEEE Computer Society Press, pp. 3-8, 13-17 Sept 2006.

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

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

[8] Heterogeneous ScaLAPACK. http://hcl.ucd.ie/project/HeteroScaLAPACK/.

[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. http://hcl.ucd.ie/project/HeteroMPI/.

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

[11] P. Alonso, R. Reddy, and A. Lastovetsky, "Experimental Study of Six Different Parallel Matrix-Matrix Applications for Heterogeneous Computational Clusters of Multicore Processors," Technical Report UCD-CSI-2009-2, University College Dublin, 2009.

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

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

[14] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix Multiplication on Heterogeneous Platforms", IEEE Transactions on Parallel and Distributed Systems, Volume 12, No. 10, pp.1033-1051, 2001.

[15] P. Alonso and A. M. Vidal, "Cauchy-like system solution on multicore platforms," Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA 2008), May 13-16, NTNU, Trondheim, Norway.

[16] R. Reddy, A. Lastovetsky, and P. Alonso, "Heterogeneous PBLAS: Optimization of PBLAS for Heterogeneous Computational Clusters," In Proceedings of the 7th International Symposium on Parallel and Distributed Computing (ISPDC 2008), pp. 73-80, IEEE Computer Society Press.

[17] R. Reddy, A. Lastovetsky, and P. Alonso, "Scalable Dense Factorizations for Heterogeneous Computational Clusters," In Proceedings of the 7th International Symposium on Parallel and Distributed Computing (ISPDC 2008), pp. 49-56, IEEE Computer Society Press.

[18] R. Reddy, A. Lastovetsky, and P. Alonso, "Parallel solvers for dense linear systems for heterogeneous computational clusters," In Proceedings of the 23rd International Parallel and Distributed Processing Symposium (IPDPS 2009).