Two-Dimensional Matrix Partitioning for Parallel Computing on Heterogeneous Processors Based on Their Functional Performance Models

Alexey Lastovetsky and Ravi Reddy

School of Computer Science and Informatics, University College Dublin, Belfield Dublin 4, Ireland
{Alexey.Lastovetsky,Manumachu.Reddy}@ucd.ie

Abstract. The functional performance model (FPM) of heterogeneous processors has proven to be more realistic than the traditional models because it integrates many important features of heterogeneous processors such as the processor heterogeneity, the heterogeneity of memory structure, and the effects of paging. Optimal 1D matrix partitioning algorithms employing FPMs of heterogeneous processors are already being used in solving complicated linear algebra kernel such as dense factorizations. However, 2D matrix partitioning algorithms for parallel computing on heterogeneous processors based on their FPMs are unavailable. In this paper, we address this deficiency by presenting a novel iterative algorithm for partitioning a dense matrix over a 2D grid of heterogeneous processors and employing their 2D FPMs. Experiments with a parallel matrix multiplication application on a local heterogeneous computational cluster demonstrate the efficiency of this algorithm.

Keywords: data partitioning algorithms, functional performance models, heterogeneous processors, parallel matrix multiplication.

1 Introduction

Traditional data partitioning algorithms for parallel computing on heterogeneous processors [1-3] are based on a performance model, which represents the speed of a processor by a constant positive number and computations are distributed amongst the processors such that their volume is proportional to this speed of the processor. The number characterizing the performance of the processor is typically its relative speed demonstrated during the execution of a serial benchmark code solving locally the core computational task of some given size.

The traditional constant performance models (CPMs) proved to be accurate enough for heterogeneous distributed memory systems if partitioning of the problem results in a set of computational tasks that fit into the main memory of the assigned processor. But these models become less accurate in the presence of paging. The functional performance model (FPM) of heterogeneous processors proposed and analysed in [3] has proven to be more realistic than the CPMs because it integrates many important

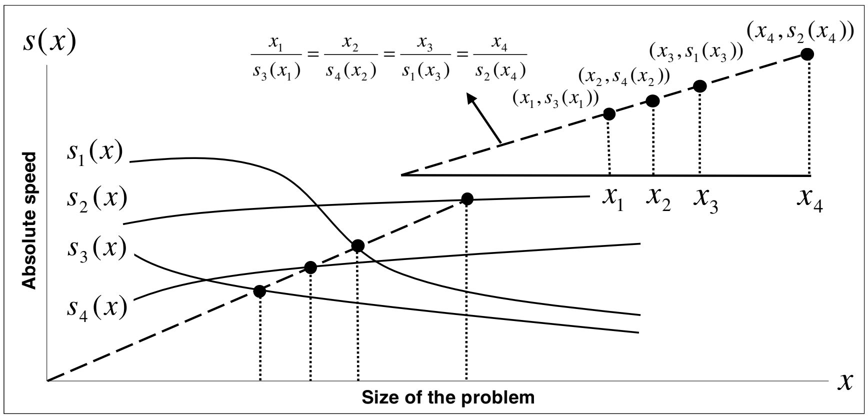


Fig. 1. Optimal data distribution showing the geometric proportionality of the number of chunks to the speed of the processor

features of heterogeneous processors such as the processor heterogeneity, the heterogeneity of memory structure, and the effects of paging. The algorithms employing it therefore distribute the computations across the heterogeneous processors more accurately than the algorithms employing the CPMs. Under this model, the speed of each processor is represented by a continuous function of the size of the problem. This model is application centric because, generally speaking, different applications will characterize the speed of the processor by different functions.

The problem of distributing independent chunks of computations over a one-dimensional arrangement of heterogeneous processors using this FPM has been studied in [4]. It can be formulated as follows: Given n independent chunks of computations, each of equal size (i.e., each requiring the same amount of work), how can we assign these chunks to p ($p<n$) physical processors P_1, P_2, ..., P_p with their respective full FPMs represented by speed functions $s_1(x)$, $s_2(x)$, ..., $s_p(x)$ so that the workload is best balanced? An algorithm solving it with a complexity of $O(p \times \log_2 n)$ is also presented. This and other similar algorithms, which relax the restriction of bounded heterogeneity of the processors [5] and which are not sensitive to the shape of speed functions [6], are based on the observation that the optimal data distribution points $(x_1, s_1(x_1))$, $(x_2, s_2(x_2))$, …, $(x_p, s_p(x_p))$ lie on a straight line passing through the origin of the coordinate system and are the intersecting points of this line with the graphs of the speed functions of the processors. This is shown in Figure 1. These algorithms are used as building blocks in algorithms solving more complicated linear algebra kernel such as the dense factorizations [7].

However, 2D matrix partitioning algorithms for parallel computing on heterogeneous processors and employing their FPMs are unavailable. We address this deficiency in this paper by presenting a novel iterative algorithm of optimal 2D data partitioning for parallel computing on a 2D grid of heterogeneous processors and employing their 2D FPMs. The algorithm assumes the 2D FPMs are given. Using experimental results with parallel matrix multiplication applications on a local heterogeneous computational cluster, we demonstrate the efficiency of this algorithm.

The rest of the paper is organized as follows. In Section 2, we present the contribution of this paper, which is the data partitioning algorithm. This is followed by experimental results on a local heterogeneous computing cluster in Section 3. For the experiments, we use parallel matrix multiplication applications demonstrating the efficiency of the algorithm.

2 Data Partitioning Algorithm

The data partitioning problem that we are trying to solve can be formulated as follows:

- Given
  - The problem size represented by a set of two parameters, (m,n), i.e., $m\times n$ independent chunks of computations each of equal size (i.e., each requiring the same amount of work). The problem size characterizes the amount and layout of data in two dimensions, for example, a dense matrix of size $m\times n$;
  - (p,q), the dimensions representing the 2D processor grid of size $p\times q$, P_{ij}, $i\in[1,p], j\in[1,q]$
  - The FPMs of the processors, $S=\{s_{ij}(x,y)$, $i\in[1,p], j\in[1,q]$. The execution time t to execute a problem size (x,y) on a processor i can be calculated using the formula $(x\times y)/s_i(x,y)$;
  - ε, the termination criterion, which represents the required relative accuracy of the solution.
- Assign each processor, P_{ij}, a block of r_{ij} rows and c_{ij} columns satisfying the conditions, $\sum_{i=1}^{p} r_{ij} = m, \forall j\in[1,q]$ and $\sum_{j=1}^{q} c_{ij} = n, \forall i\in[1,p]$, meaning that it is responsible for computing $r_{ij}\times c_{ij}$ computational chunks, such that
  - The rectangular partitions, $r_{ij}\times c_{ij}$ form a two-dimensional $p\times q$ arrangement, and
  - The maximum relative difference (MRD) between execution times on the processors is less than or equal to ε, i.e., MRD≤ε.

***Data Partitioning Algorithm (DPA-FPM-2D)*:** The inputs to the algorithm are

- The problem size represented by a set of two parameters, (m,n);
- (p,q), the dimensions representing the 2D processor grid of size $p\times q$, P_{ij}, ($i\in[1,p], j\in[1,q]$);
- The FPMs of the processors, $S=\{s_{ij}(x,y)\}$;
- The termination criterion, ε.

The outputs r and c are integer arrays of size $p\times q$. The (i,j)-th element of r is the number of rows allocated to processor P_{ij} and the (i,j)-th element of c is the number of columns allocated to processor P_{ij}. The algorithm can be summarized as follows:

- **Initialization:**
  - The execution times to execute the problem size, (m/p, n/q), on all the processors, P_{ij} ($i \in [1, p], j \in [1, q]$), are calculated using the formula $t_{ij}^1 = \left(\left(m/p\right) \times \left(n/q\right)\right) / s_{ij}^1(m/p, n/q)$. If MRD≤ε, the algorithm terminates. The optimal distribution is $r_{ij}=m/p$, $c_{ij}=n/q$;
  - Otherwise, the algorithm proceeds to the next step, for which the inputs are the single-number speeds, $s_{ij}^1\left(r_{ij}^1, c_{ij}^1\right)$ where $r_{ij}^1 = m/p$, $c_{ij}^1 = n/q$.
- **Iteration:** At step k+1,
  - The procedure, *HCOL*(m, n, p, q, s) illustrated in Figure 2, is invoked to determine the column-based data distribution, which is optimal for the single-number speeds, $s = s_{ij}^k\left(r_{ij}^k, c_{ij}^k\right)$. The resulting data distribution is $\left(r_{ij}^{k+1}, c_{ij}^{k+1}\right)$;
  - The execution times, t_{ij}^{k+1}, are then calculated using the formula $t_{ij}^{k+1} = \left(r_{ij}^{k+1} \times c_{ij}^{k+1}\right) / s_{ij}^{k+1}(r_{ij}^{k+1}, c_{ij}^{k+1})$. If MRD≤ε, the algorithm terminates. The optimal distribution is $(r_{ij}^{k+1}, c_{ij}^{k+1})$. If MRD>ε, the algorithm proceeds to the next step;
  - The MRD in each processor column is checked. For each of the processor columns where MRD>ε, the algorithm executes the procedure, *HSPF*(m, c, p, S), which returns the optimal distribution of m independent chunks over p heterogeneous processors P_i of respective speed functions $S=\{s_{ij}(x,c)\}$ where c is a constant. Here c is the size of the column block, which is the same for all the processors in a processor column. For the sake of simplicity, let us assume the existence of one such column x. The inputs to the procedure are then m, c_{1x}^{k+1}, p, $S=\{s_i(x, c_{1x}^{k+1})\}_{i=1}^{p}$. The resulting data distribution is $\left(r_{ix}^{k+1}\right), \forall i \in [1, p]$. The algorithm then proceeds to the next iteration, for which the inputs from this processor column are the speeds, $s_{ix}^{k+1}(r_{ix}^{k+1}, c_{1x}^{k+1}), \forall i \in [1, p]$. For the processor columns for which the MRD≤ε, the inputs are the speeds, $s_{ij}^{k+1}(r_{ij}^{k+1}, c_{ij}^{k+1})$.

***HCOL*:** This procedure invokes the data partitioning algorithm [1,3], which determines the optimal 2D column-based partitioning of a dense matrix of size m×n on a 2D heterogeneous processor grid of size p×q. The matrix is partitioned into uneven rectangles so that they are arranged into a 2D grid of size p×q and the area of a rectangle is proportional to the speed of the processor owning it. The inputs to the algorithm are

- The dense matrix of size m×n;
- (p,q), the dimensions representing the 2D processor grid of size p×q, P_{ij}, $i \in [1, p], j \in [1, q]$;
- The single number speeds of the processors, s_{ij}.
- The output is the heights and the widths of the rectangles, (r_{ij},c_{ij}).

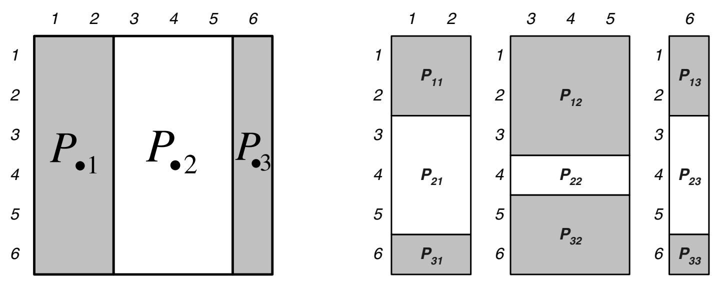


Fig. 2. Example of two-step distribution of a 6×6 square over a 3×3 processor grid. The relative speed of processors is given by {0.11, 0.25, 0.05, 0.17, 0.09, 0.08, 0.05, 0.17, 0.03}. (a) At the first step, the 6×6 square is distributed in a one-dimensional block fashion over processors columns of the 3×3 processor grid in proportion 0.33:0.51:0.16 ≈ 2:3:1. (b) At the second step, each vertical rectangle is distributed independently in a one-dimensional block fashion over processors of its column. The first rectangle is distributed in proportion 0.11:0.17:0.05 ≈ 2:3:1. The second one is distributed in proportion 0.25:0.09:0.17 ≈ 3:1:2. The third is distributed in proportion 0.05:0.08:0.03 ≈ 2:3:1.

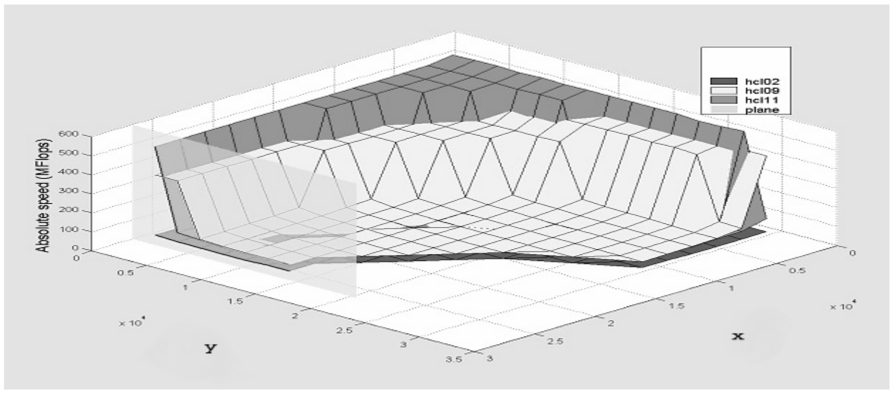
The algorithm can be summarized as follows:

- First, the area $m \times n$ is partitioned into q vertical slices, so that the area of the j-th slice is proportional to $\sum_{i=1}^{p} s_{ij}$ (see Figure 2(a)). It is supposed that blocks of the j-th slice will be assigned to processors of the j-th column in the $p \times q$ processor grid. Thus, at this step, the load *between* processor columns in the $p \times q$ processor grid is balanced, so that each processor column will store a vertical slice whose area is proportional to the total speed of its processors;
- Then, each vertical slice is partitioned independently into p horizontal slices, so that the area of the i-th horizontal slice in the j-th vertical slice is proportional to s_{ij} (see Figure 2(b)). It is supposed that blocks of the i-th horizontal slice in the j-th vertical slice will be assigned to processor P_{ij}. Thus, at this step, the load of processors *within* each processor column is balanced independently.

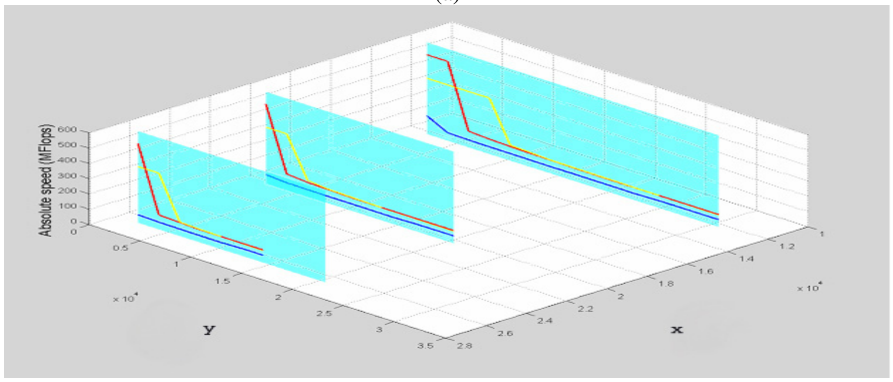
***HSPF*:** This procedure returns the optimal distribution of m independent chunks over p heterogeneous processors of P_1, P_2, ..., P_p of respective speeds $S=\{s_1(x,y), s_2(x,y), ..., s_p(x,y)\}$ (HSPF stands for Heterogeneous Set Partitioning using Functional model of heterogeneous processors). It is composed of two steps:

- Surfaces, $z_i=s_i(x,y)$, representing the absolute speeds of the processors are sectioned by the plane $x=c$ (as shown on Figure 3 (a) for 3 surfaces). A set of p curves on this plane (as shown in Figure 3 (b)) will represent the absolute speeds of the processors against variable y given parameter x is fixed;
- Apply the set partitioning algorithm [4] to this set of p curves to obtain an optimal distribution.

The DPA-FPM-2D algorithm can be intuitively explained as follows. The goal of the algorithm is to determine points $(r_{ij},c_{ij},s_{ij}(r_{ij},c_{ij}))$ in the (r,c,s) space such that $(r_{ij} \times c_{ij})/s_{ij}(r_{ij},c_{ij})=C$, where C is a constant (and also the execution time). Each iteration



(a)



(b)

Fig. 3. (a) Two surfaces representing the absolute speeds of 2 processors are sectioned by the planes x=c. (b) Curves on this plane represent the absolute speeds of the processors against variable y, given parameter x is fixed.

step facilitates convergence to the optimal solution along the coordinate c and the execution of *HSPF* procedure *within the processor columns* provides the optimal solution along the coordinate r by fixing the value of the coordinate c. During the execution of the procedure *HSPF*, the coordinate 'c' of the (r,c,s) space is kept constant and 1D FPMs of the processors, (r_{ix},s_{ix}(r_{ix},c_{1x})), are built against parameter r in the (r,s) plane, taking one processor column x for instance. These are a set of p curves representing the absolute speeds of the p processors against parameter r given parameter c is fixed. It can be visualized as the plane y=c intersecting the speed surfaces in the (r,c,s) space to give p curves in the (r,s) plane. The data partitioning algorithm [4] is then applied to this set of p curves to obtain optimal data distribution for these 1D FPMs so that $r_{ix}/s_{ix}(r_{ix},c_{1x})=C_1$, $i \in [1, p]$,where C_1 is a constant. Since c_{1x} is the same for all the processors in the column x, $(r_{ix}\times c_{1x})/s_{ix}(r_{ix},c_{1x})=C_2$ $\forall i \in [1, p]$, where C_2 is a constant too.

The speeds from the new data distribution are employed again to determine the new value of the coordinate c, which would be closer to the optimal solution, and the iteration procedure is repeated. Thus in this manner, the algorithm converges to the optimal solution.

3 Experimental Results

We use a small heterogeneous local network of 16 different Linux processors (hcl01-hcl16) for the experiments. The specifications of the network are available 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.

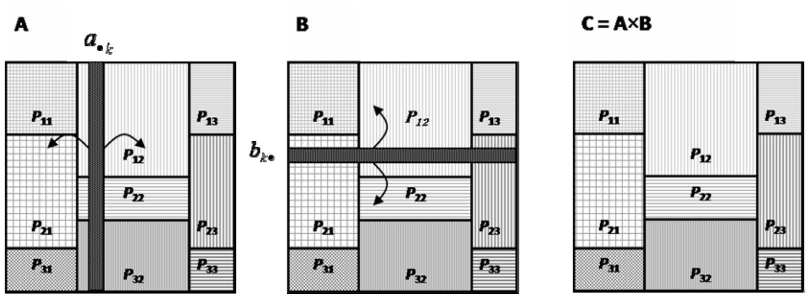


Fig. 4. One step of the algorithm of parallel matrix multiplication employing a 2D heterogeneous processor grid of size 3×3. Matrices A, B, and C are partitioned such that the area of the rectangle is proportional to the speed of the processor owning it. First, each b×b block of the pivot column $a_{\bullet k}$ of matrix A (shown with curly arrows) is broadcast horizontally, and each b×b block of the pivot row $b_{k\bullet}$ of matrix B (shown with curly arrows) is broadcast vertically. Then, each b×b block c_{ij} of matrix C is updated, $c_{ij}=c_{ij}+a_{ik}\times b_{kj}$.

Figure 4 shows the parallel matrix multiplication application used for the experiments. It implements the matrix operation C=A×B, multiplying matrix A and matrix B, where A, B, and C are dense matrices of size m×k, k×n, and m×n matrix elements respectively on a 2D heterogeneous processor grid of size p×q. We use dense square matrices and a 2D heterogeneous processor grid of size 3x3 for illustration purposes. Each matrix element is a square block of size b×b (value of b used in the experiments is 64). The heterogeneous parallel algorithm used to compute this matrix product is a modification of the ScaLAPACK outer-product algorithm [8]. We assume that only one process is configured to execute on a processor. The data partitioning problem is that the matrices A, B, and C must be divided into rectangles such that there is one-to-one mapping between the rectangles and the processors, and the area of each rectangle is proportional to the speed of the processor owning it.

For this application, the core computational kernel performs a matrix update of a matrix C_b of size m_b×n_b using A_b of size m_b×1 and B_b of size 1×n_b as shown in Figure 5. The size of the problem is represented by two parameters, 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. Therefore, 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 $m_b \times n_b$. Therefore, 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. This gives us a function, f: $\mathbf{N}^2 \rightarrow \mathbf{R}_+$, mapping problem sizes to speeds of the processor. The FPM of the processor is obtained by continuous extension of function f: $\mathbf{N}^2 \rightarrow \mathbf{R}_+$ to function g: $\mathbf{R}_+^2 \rightarrow \mathbf{R}_+$ (f(n,m)=g(n,m) for any (n,m) from $\mathbf{N}^2$).

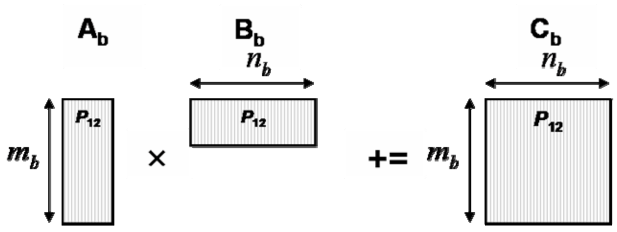


Fig. 5. The computational kernel (shown here for processor P_{12} for example) performs a matrix update of a dense matrix C_b of size $m_b \times n_b$ using A_b of size $m_b \times 1$ and B_b of size $1 \times n_b$. The matrix elements represent $b \times b$ matrix blocks.

The heterogeneity of the network due to the heterogeneity of the processors is calculated as the ratio of the absolute speed of the fastest processor to the absolute speed of the slowest processor. For example, consider the benchmark code of a local DGEMM update of two matrices 2560×64 and 64×2560, the absolute speeds of the processors hcl01-hcl16 in million flop/s performing this update are {130, 258, 188, 188, 188, 214, 125, 127, 157, 232, 147, 137, 157, 197, 194, 201}. As one can see, hcl02 is the fastest processor and hcl07 is the slowest processor. The heterogeneity is therefore 2.

Figure 6 shows the execution times of the sequential application and three parallel applications solving the same matrix multiplication problem. The execution of the parallel applications consists of two parts. First, all the processors execute a data partitioning algorithm to partition the matrices and then they perform the parallel matrix multiplication itself.

The first parallel application employs a data partitioning algorithm (DPA-CPM-2D) that uses the CPMs of the processors [1,3]. The constant single number speeds of the processors are calculated from the execution of a local DGEMM update of two matrices of sizes, $(m/p) \times b$ and $b \times (n/q)$ respectively where (m,n) is the problem size, p=4, q=4, and b=64. The second parallel application employs a data partitioning algorithm DPA-FPM-1D that optimally partitions the matrix over a 1D arrangement of processors based on their FPMs. The inputs to DPA-FPM-1D are the problem size (m, n), the 1D arrangement of 16 processors and their FPMs. The third parallel application employs DPA-FPM-2D. The parameters to DPA-FPM-2D are the problem size (m, n), p=4, q=4, ε=0.05, and the full FPMs of the processors.

The sequential application is executed on the fastest processor (hcl02). For problem sizes ((m,n), m>10240 & n>10240), the sequential application fails due to the problem

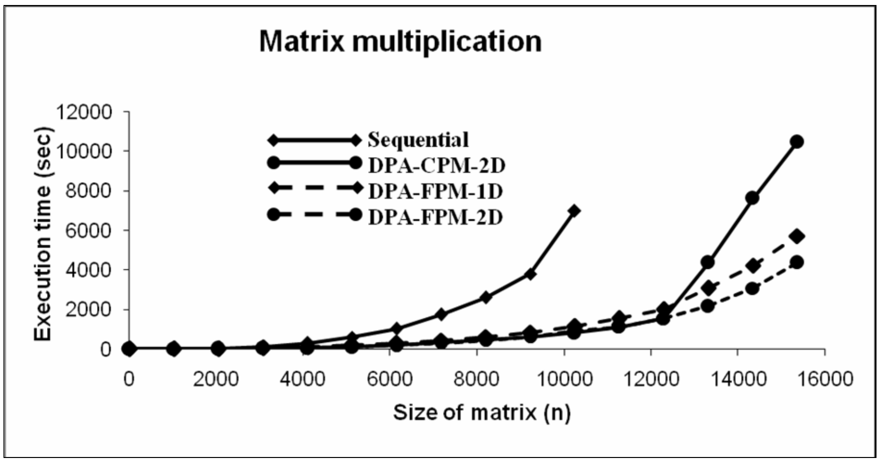


Fig. 6. Execution times of the parallel application employing DPA-FPM-2D, a parallel application employing 1D grid of heterogeneous processors and their associated FPMs, a parallel application employing a data partitioning algorithm using constant single-number speeds and a sequential application solving the same matrix multiplication problem.

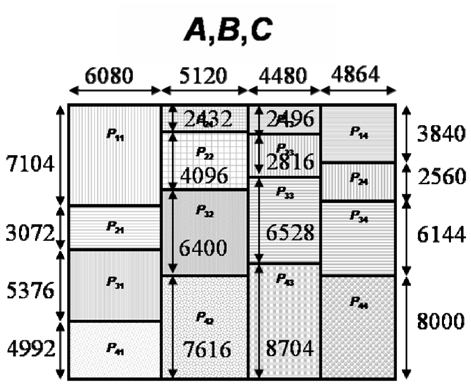


Fig. 7. Optimal data distribution using DPA-FPM-2D whose parameters are m=n=20544, p=4, q=4

size exceeding the memory limit of the processor. One or more processors start paging around the problem sizes ((m,n), 1≤m≤12288 & 1≤n≤12288). The parallel application employing the DPA-CPM-2D algorithm fails for problem sizes ((m,n), m>15360 & n>15360).

Figure 7 shows the optimal data distribution of the dense matrices A, B, and C determined by DPA-FPM-2D for the problem size (m,n)=(20544,20544). One can see that the parallel application employing the DPA-FPM-2D algorithm outperforms the other applications. The total number of iteration steps of DPA-FPM-2D observed are 1 for the problem sizes, ((m,n), 1≤m≤12288 & 1≤n≤12288) and 2 for the problem sizes in the region of paging, ((m,n), m>12288 & n>12288). Therefore, we can conclude the DPA-FPM-2D algorithm converges very fast. Its execution time is also found to be several orders of magnitude less than the execution time of the parallel matrix multiplication.

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant Number 08/IN.1/I2054.

References

[1] Kalinov, A., Lastovetsky, A.: Heterogeneous Distribution of Computations Solving Linear Algebra Problems on Networks of Heterogeneous Computers. Journal of Parallel and Distributed Computing 61(4), 520–535 (2001)

[2] Beaumont, O., Boudet, V., Rastello, F., Robert, Y.: Matrix Multiplication on Heterogeneous Platforms. IEEE Transactions on Parallel and Distributed Systems 12(10), 1033–1051 (2001)

[3] Lastovetsky, A., Reddy, R.: On Performance Analysis of Heterogeneous Parallel Algorithms. Parallel Computing 30(11), 1195–1216 (2004)

[4] Lastovetsky, A., Reddy, R.: Data Partitioning with a Functional Performance Model of Heterogeneous Processors. International Journal of High Performance Computing Applications 21(1), 76–90 (2007)

[5] Lastovetsky, A., Reddy, R.: Data Partitioning for Multiprocessors with Memory Heterogeneity and Memory Constraints. Scientific Programming 13(2), 93–112 (2005)

[6] Lastovetsky, A., Reddy, R.: Data Partitioning with a Realistic Performance Model of Networks of Heterogeneous Computers. In: 17th International Parallel and Distributed Processing Symposium. IEEE Computer Society Press, Los Alamitos (2004)

[7] Lastovetsky, A., Reddy, R.: Data distribution for dense factorization on computers with memory heterogeneity. Parallel Computing 33(12), 757–779 (2007)

[8] Petitet, A., Dongarra, J.: Algorithmic Redistribution Methods for Block-Cyclic Decompositions. IEEE Transactions on Parallel and Distributed Systems 10(12), 1201–1216 (1999)