Column-Based Matrix Partitioning for Parallel Matrix Multiplication on Heterogeneous Processors Based on Functional Performance Models

David Clarke, Alexey Lastovetsky, and Vladimir Rychkov

School of Computer Science and Informatics, University College Dublin,
Belfield, Dublin 4, Ireland
David.Clarke.1@ucdconnect.ie
{Alexey.Lastovetsky,vladimir.rychkov}@ucd.ie

Abstract. In this paper we present a new data partitioning algorithm to improve the performance of parallel matrix multiplication of dense square matrices on heterogeneous clusters. Existing algorithms either use single speed performance models which are too simplistic or they do not attempt to minimise the total volume of communication. The functional performance model (FPM) is more realistic then single speed 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. To load balance the computations the new algorithm uses FPMs to compute the area of the rectangle that is assigned to each processor. The total volume of communication is then minimised by choosing a shape and ordering so that the sum of the half-perimeters is minimised. Experimental results demonstrate that this new algorithm can reduce the total execution time of parallel matrix multiplication in comparison to existing algorithms.

Keywords: Parallel matrix multiplication, functional performance models, heterogeneous platforms, load balance, data partitioning.

1 Introduction

In this paper, we deal with the problem of partitioning matrices across a cluster of heterogeneous processors in order to improve the performance of parallel matrix multiplication. Computation time can be minimised by partitioning the work so that all processors finish their work in the same time. Communication time can be reduced by arranging the partitioning in such a way as to minimise the total volume of communication. Communication time can also be reduced by measuring the interconnect speed between all nodes and choosing a partitioning based on this; however, this approach is beyond the scope of this paper.

Two-dimensional decomposition of matrices yields more efficient parallel algorithms then one-dimensional decomposition. Hence, ScaLAPACK [2], a linear

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 450–459, 2012.

algebra library designed for homogeneous platforms, implements the two-dimensional regular grid partitioning in the parallel outer-product routine. In addition, this routine has a blocking factor, b, designed to take advantage of processor cache. Each matrix block contains $b \times b$ elements, and each step of the routine involves updating one block.

To balance the load on heterogeneous platforms, irregular partitioning schemes are used. This approach is based on a concept of the *computational unit*, the smallest amount of work that can be given to a processor. All units require the exact same number of arithmetic calculations and have the same data storage requirements. The computational load is balanced by distributing computational units in proportion to the speeds of processors. In the case of two-dimensional matrix multiplication, the computational unit is the update of a $b \times b$ block. Each processor P_i is responsible for computing a rectangle of $m_i \times n_i$ blocks.

There are no existing algorithms to find the general solution of irregular partitioning. However, there are some algorithms that find sub-optimal solutions under certain restrictions on the arrangements of rectangles. One such algorithm (KL) [7] implements a column-based partitioning. Processors are arranged into columns, and all processors in a column are allocated rectangles of the same width. The widths of all the columns sum to the N dimension of the matrix. The heights of rectangles in a column sum to the M dimension of the matrix. This algorithm does not minimise the total volume of communication and uses a basic partitioning algorithm based on single speed values.

In another column-based algorithm (BR) [1], the area of rectangles is defined by the relative cycle-times of processors. The shape and ordering of rectangles is calculated to minimise the sum of half-perimeters $\sum_{i=1}^{\hat{p}} m_i + n_i$. Therefore, this algorithm partitions a matrix in proportion to processor performance in such a way as to reduce the total volume of communication.

Many traditional data partitioning algorithms use a similar approach for calculating relative processor performance, where processor speed is represented by a single positive number. These algorithms were designed for medium sized problems run on general purpose single core workstations. It has since been demonstrated in [8] that processor speed is not invariant with problem size. Speed represented by a function of problem size has proven to be more realistic than the constant performance models because it integrates many important features of heterogeneous processors such as the architectural heterogeneity, the heterogeneity of memory structure, the effects of paging and so on. A better partitioning can be achieved by using the algorithms based on functional performance models (FPM).

The functional performance model proposed in [8] is one-dimensional and represented by a line in 2D space. In two-dimensional matrix partitioning, the problem size is composed of two parameters, m and n. Hence, the functional performance model becomes a surface in 3D space, where the z axes represents speed. In [11], these surfaces are used as more realistic performance models of processors in order to improve the KL partitioning algorithm. It iteratively slices 2D plains through the 3D space at positions that represent the column width, reducing the problem to a series of one-dimensional partitioning. FPM-based algorithm is used to find optimal partitioning within each column, while the column widths are found using the basic partitioning algorithm based on single values.

This algorithm does find a good partitioning but it has a number of disadvantages: (i) communication cost is not taken into account and any prime number of processors cannot be used efficiently; (ii) convergence is not guaranteed because it uses the basic partitioning algorithm as demonstrated in [3]; (iii) building full 2D models is expensive.

In this paper, we present a modification of the (BR) algorithm. Instead of simplistic performance models of processors, we use more accurate functional performance models. We reduce the complexity of matrix partitioning from two parameters down to one by using the area of rectangles $d = m \times n$. This allows us to build less expensive one-dimensional functional performance models and to solve the partitioning problem in one step with help of the FPM-based algorithm. The result of this partitioning is the areas of rectangles, which are then arranged by the (BR) algorithm so that the total volume of communication is minimised. Therefore, we achieve more optimal data partitioning, which is based on more accurate performance model of processors, while also minimising communication volume.

The rest of the paper is organised as follows. In Section 2, we review existing algorithms for two-dimensional matrix partitioning designed for heterogeneous platforms. In Section 3, we present the main contribution of this paper, namely, the FPM-based modification of the BR algorithm. In Section 4, we present the experimental results for parallel matrix multiplication on a heterogeneous cluster.

2 Related Work

In this section, we summarize existing heterogeneous partitioning algorithms for parallel matrix multiplication. The common features of these algorithms are the following: (i) computational units are mapped to processors in proportion to their speed; (ii) to reduce the space of possible solutions of this mapping, a column-based restriction is applied. These algorithms determine the partitioning for a heterogeneous implementation of the blocked ScaLAPACK outer product [2].

2.1 Column-Based Partitioning (KL)

Column-based partitioning of matrices was first introduced in [7]. This algorithm distributes a unit square between $\hat{p}$ heterogeneous processors arranged into q columns, each of which is made of p_j processors, $j \in [1, ..., q]$:

- Let the relative speed of the i-th processor from the j-th column, P_{ij}, be s_{ij} such that $\sum_{j=1}^{q} \sum_{i=1}^{p_j} s_{ij} = 1$.

- Then, we first partition the unit square into c vertical rectangular slices such that the width of the j-th slice is $n_j = \sum_{i=1}^{p_j} s_{ij}$. This partitioning makes the area of each vertical slice proportional to the sum of the speeds of the processors in the corresponding column.
- Second, each vertical slice is partitioned independently into rectangles in proportion to the speed of the processors in the corresponding processor column.

This algorithm has some drawbacks. Namely, it does not take communication cost into account, and it relies on inaccurate, single-value performance model of the processor speed. These issues are addressed by the algorithms in Section 2.2 and 2.3 respectively.

2.2 Minimising Total Communication Volume (BR)

The BR algorithm [1] minimises the total volume of communication as follows. The objective is to tile the unit square into $\hat{p}$ non-overlapping rectangles, where each rectangle is assigned to a processor, in such a way as to achieve load balancing and minimise communication. Then, this unit square can be scaled to the size of the matrix. The general solution to this problem is NP complete, however, by applying a restriction that all processors in the same column have the same width, the authors of [1] were able to produce an algorithm of polynomial complexity.

First, the relative speed of each processor is calculated from the relative cycle-times t_i: $s_i = \frac{1/t_i}{\sum(1/t_i)}$. This speed gives the area d_i of the rectangle assigned to the processor P_i. However, there are degrees of freedom with regards to the shape and ordering of the rectangles.

In each iteration, the number of elements of matrix $\mathbf{A}$ that each processor either sends or receives is directly proportional to its height m_i and the number of elements of matrix $\mathbf{B}$ sent or received is proportional to its width n_i. The total volume of data exchange is proportional to the sum of the half perimeters $H = \sum_{i=0}^{p-1}(m_i + n_i)$. Communication cost can be reduced by minimising H. This is achieved by arranging the rectangles so that they are as square as possible. The optimum number of columns c and the optimum number of processors in each column r_j is calculated by the algorithm. The processors are sorted in order of increasing speed. A table is built to summarise the communication costs for 1 to p columns, i.e. from all processors in the same column to each processor in an individual column. The algorithm then works backwards through the table, selecting values for c and r_j which minimise the half perimeter.

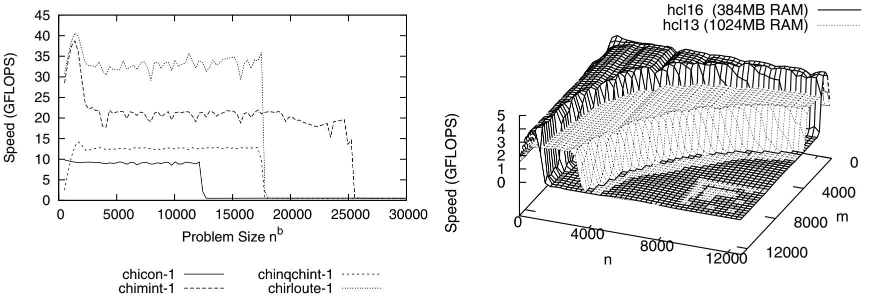
The main disadvantage of this algorithm is that cycle-times is not an accurate measure of the processor performance. This may result in poor performance of parallel matrix multiplication.

2.3 Functional Performance Model-Based Partitioning (FPM-KL)

The assumption that the absolute speed of the processor is independent of the size of the computational task becomes less accurate in the following situations:

- The partitioning of the problem results in some tasks either not fitting into the main memory of the assigned processor and hence causing paging or fully fitting into faster levels of its memory hierarchy.
- Some processing units involved in computations are not traditional general-purpose processors (say, accelerators such as GPUs or specialized cores). In this case, the relative speed of a traditional processor and a non-traditional one may differ for two different sizes of the same computational task even if both sizes fully fit into the main memory.
- Different processors use different codes to solve the same problem locally.

Functional performance models more accurately represent the speed of processors then traditional constant models [8]. The speed of each processor is represented by a continuous positive function of problem size (Fig. 1(a)). These functions are obtained by benchmarking a serial code that is equivalent to one step of the parallel routine. There are two approaches to building the models. If the application is to be run multiple times on a set of machines (or a sub-set of these machines) then an exhaustive full functional performance model can be built for each unique machine. This process is time consuming but needs to be done only once for each routine on each unique machine, it can be done at compile time. An alternative approach, suitable for more dynamic environments, where the available machines change regularly, is to dynamically build only the necessary parts of the models at run-time. This approach has been demonstrated in [10]. Dynamically built models are perfectly applicable to the algorithm proposed in the next section, however for clarity of results, we will use full functional performance models in the remainder of this paper.



(a) (b)

Fig. 1. (a) Functional performance models for 4 nodes from the Grid'5000 Lille site. (b) Two-dimensional models for two nodes from our local heterogeneous cluster, showing hcl16 is a faster node with less memory then hcl13.

The partitioning algorithm based on functional performance models proposed in [8] is designed for partitioning with one parameter. However, the ScaLAPACK outer-product routine requires two partitioning parameters, m_{ij} and n_j, for each processor P_{ij}. An two-dimensional iterative algorithm to overcome this shortcoming is proposed in [9]. The strategy is similar to that described in Section 2.1. However, functional performance models are used in place of simplistic single value performance models. The two parameters, m and n, gives two degrees of freedom which leads a model consisting of a surface in 3D space (Fig. 1(b)). The z axes represents processor speed.

Processors are arranged into a $p \times q$ grid. Initially column widths are given by $n_j = N/q \ \forall j$. Iterating:

1. A 2D plane is sliced through the 3D space at positions equal to n_j. This gives one-dimensional functional performance models which can be used by the algorithm in [8] to find the optimum partitioning within each column, m_{ij}. Single value speeds for this partitioning can then be found from the model s_{ij}.
2. If the maximum relative difference between execution times is less then some ϵ the algorithm finishes, otherwise it continues.
3. New column widths n_i are calculated in proportion the single value speed of each column $\sum_{i=1}^{p} s_{ij}$

This algorithm does find a good partitioning but it has a number of disadvantages: (i) it does not take communication cast into account; (ii) the processor grid is fixed and the algorithm is unable to change the ordering of the processors; (iii) it relies on a constant performance model to find the location of the next slice so there is no guarantee of convergence; (iv) building full 2D models requires more time consuming benchmarking (while a 1D model requires x experimental points to achieve a given accuracy, a 2D model requires x^2 points).

2.4 Other Related Work

A matrix partitioning algorithm for a heterogeneous combination of FPGA and general purpose processors is presented in [12]. Their model does consider memory hierarchy, however detailed knowledge of the architecture is requires and each memory level requires a parameter in their partitioning algorithm. Hence, it is not self adaptable to new environments.

The authors of [4] present interesting algorithms for minimising the total volume of communication and for partitioning with respect to both computational power and communication speed. However, these algorithms are targeted for a master-worker platform and so are not directly applicable to our target platform.

A different approach for load balancing is taken in [6]. The problem is broken down into many small processes, each requiring an equal amount of work and data storage. Processes are then mapped to processors in proportion to processor performance. This approach allows for easy adaptation of existing homogeneous algorithms to heterogeneous platforms, however it is unable to achieve fine grained load balancing without incurring an overhead penalty for running a large number of processes per processor.

3 New FPM-BR Two-Dimensional Matrix Partitioning Algorithm

The efficient heterogeneous ScaLAPACK outer-product routine requires two partitioning parameters for each processor. Load balancing with 1D functional performance models only works with problems with one degree of freedom. The existing 2D FPM-KL partitioning algorithm does not take communication cost into account while the BR algorithm minimises communication volume but uses a too simplistic model for processor performance. To overcome these shortcomings, we present a new FPM-BR algorithm that combines the strengths of these algorithms.

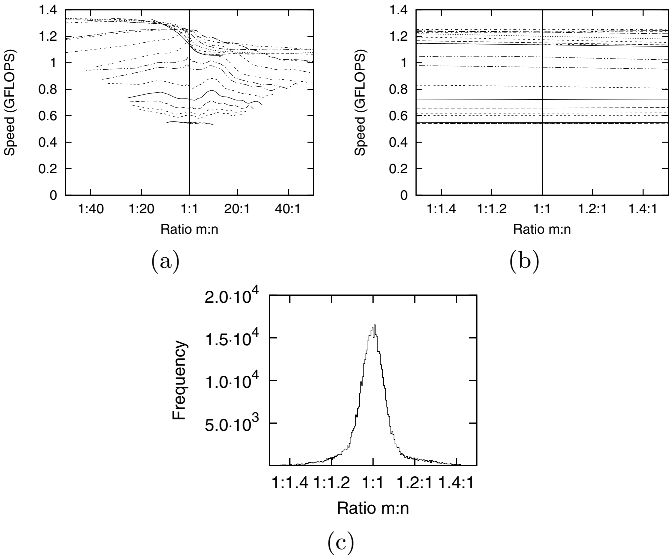
The height m_i and width n_i parameters can be combined into one parameter, area $d_i = m_i \times n_i$. Our computational unit is a $b \times b$ block, and benchmarking is done for square areas $m = n = \sqrt{d}$, for $0 < d \leq M \times N$. We can then partition using the one-dimensional FPM-based algorithm [8] to determine the areas of the rectangles that should be partitioned to each processor. The BR algorithm is then used to calculate the optimum shape and ordering of the rectangles so that the total volume of communication is minimised.

In the algorithm proposed above we have made the assumption that a benchmark of a square area will give an accurate prediction of computation time of any rectangle of the same area, namely $s(x, x) = s(x/c, c.x)$. However, in general this does not hold true for all c (Fig. 2(a)). Fortunately, in order to minimise the total volume of communication the BR algorithm arranges the rectangles so that they are as square as possible. We have verified this experimentally by partitioning a medium sized square dense matrix using our new algorithm for 1 to 1000 nodes from the Grid'5000 platform (incorporating 20 unique nodes), and plotted the frequency of the ratio $m : n$ in Fig. 2(c). Fig. 2(b), showing a detail of Fig. 2(a), illustrates that if the rectangle is approximately square the assumption holds.

4 Experimental Results

To demonstrate the effectiveness of the new FPM-BR matrix partitioning algorithm we applied it to a heterogeneous MPI implementation of the blocked ScaLAPACK outer product routine [2]. The high performance, cross-platform multi-threaded GotoBLAS2 [5] library was used for the BLAS implementation. Dense square matrices are filled with random numbers. A block size of $b = 16$ was chosen, increasing block size allows the GotoBLAS2 dgemm subroutine to make more efficient use of cache levels, however this reduces the granularity available to the partitioner. The total matrix dimension is given by $N^b = N \times 16$, where N is the dimension used by the partitioner algorithm.

A benchmark to build the functional performance model must be done independently of other nodes. Serial code, which closely resembles one iteration of the parallel code, is timed. Memory is allocated and freed in the same order and MPI point-to-point communications are sent to itself. Statistics are applied so that benchmarks are repeated until a specified confidence interval has been achieved.



(a)

(b)

(c)

Fig. 2. Showing speed against the ratio of the sides of the partitioned rectangles. Lines connect rectangles of equal area. The centerline at 1 : 1 represents square shape. In general speed is not constant with area (a). However when the ratio is close to 1 : 1, speed is approximately constant (b). (c) Shows the frequency distribution of the ratio of $m : n$ using the new partitioning algorithm for 1 to 1000 machines (incorporating 20 unique hardware configurations).

Table 1. Lille Site Hardware Specifications

Nodes	Processor	Cores	Memory
20	2.6GHz Opteron	4	4
20	2.83GHz Xeon	8	8
19	2.4GHz Xeon	8	16
5	2.4GHz Xeon	8	8

Four partitioning algorithms (even homogeneous, BR, FPM-KL, FPM-BR) are applied to parallel matrix multiplication on 64 nodes from Grid'5000 Lille site. The total execution time for a range of problem sizes was recorded and plotted in Fig. 3. The nodes are from 4 interconnected clusters with 4 unique hardware configurations (Table 1, Fig. 1(a)). Our new FPM-BR algorithm was able to efficiently partition for all problem sizes up a maximum size of $N^b = 160000$ at which point all of the available memory is used. The BR algorithm works successfully for medium sized problems but fails for problems with $N^b > 80000$ because it uses a too simplistic model of processor speed. The FPM-KL algorithm is also able to partition up to the maximum size but performance is lower than FPM-BR because the total volume of communication is not minimised.

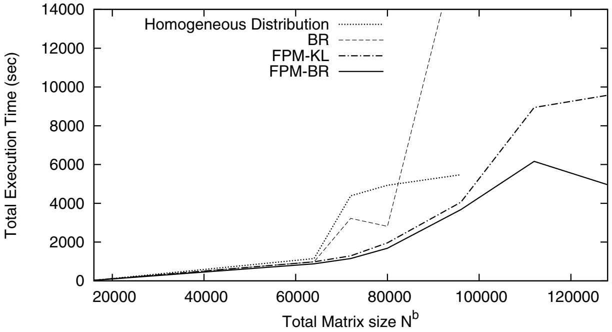


Fig. 3. Total time to execute parallel square dense matrix multiplication for a range of problem sizes using the three algorithms discussed in this paper and an even homogeneous distribution. The experiment is conducted on 64 nodes from Grid'5000 Lille site (incorporating 4 unique hardware configurations).

The speedup for FPM-BR algorithm over FPM-KL algorithm is more pronounced for non-square number of processes, for example 14 as shown in Fig. 4. The total volume of communication is reduced by 17.1% and there is a corresponding 13.6% reduction in total computation time. The difference can be accounted for by an increase, with the FPM-BR algorithm, in the number of point-to-point communications to send matrix **A** horizontally. Namely in the first iteration processor 03 must send to 7 processors (04, 14, 10, 12, 08, 05, 06) (Fig. 4(b)). With the FPM-KL algorithm, processor 03 needs only send horizontally to 3 processors (10, 13, 14) (Fig. 4(a)). Collective communications are used to broadcast elements of matrix **B** vertically.

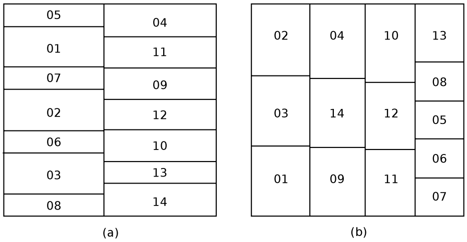


Fig. 4. Matrix partitioning for 14 heterogeneous nodes, with a problem size of $N = 840$. Using: (a) FPM-KL and (b) FPM-BR algorithms. The normalised total volume of communication is 9 and 7.457. Total computation time was 192 sec and 166 sec respectively.

The presented experimental results demonstrate that by combining functional performance models with the BR algorithm we are able to achieve both optimisation goals, namely partitioning the workload in proportion to processor speed and reducing the total volume of communication. This algorithm also allows us to use the simpler one-dimensional models rather then the more complex 2D models to partition for the two-parameter matrix multiplication routine.

Acknowledgments. This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant Number 08/IN.1/I2054. Experiments were carried out on Grid'5000 developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr). We are grateful to Arnaud Legrand who provided sample code for minimising the total volume of communication.

References

1. Beaumont, O., Boudet, V., Rastello, F., Robert, Y.: Matrix Multiplication on Heterogeneous Platforms. IEEE Trans. Parallel Distrib. Syst. 12(10), 1033–1051 (2001)
2. Blackford, L., Choi, J., Cleary, A., et al.: ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance. In: Supercomputing, p. 5. IEEE (1996)
3. Clarke, D., Lastovetsky, A., Rychkov, V.: Dynamic Load Balancing of Parallel Computational Iterative Routines on Highly Heterogeneous HPC Platforms. Parallel Process. Lett. 21(2), 195–217 (2011)
4. Dongarra, J., Pineau, J.F., Robert, Y., Vivien, F.: Matrix Product on Heterogeneous Master-Worker Platforms. In: PPoPP 2008, pp. 53–62. ACM (2008)
5. Goto, K., van de Geijn, R.A.: Anatomy of high-performance matrix multiplication. ACM Trans. Math. Softw. 34(3), 1–12 (2008)
6. Kalinov, A., Klimov, S.: Optimal Mapping of a Parallel Application Processes onto Heterogeneous Platform. In: Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, p. 123. IEEE (2005)
7. Kalinov, A., Lastovetsky, A.: Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers. In: Sloot, P.M.A., Hoekstra, A.G., Bubak, M., Hertzberger, B. (eds.) HPCN-Europe 1999. LNCS, vol. 1593, pp. 191–200. Springer, Heidelberg (1999)
8. Lastovetsky, A., Reddy, R.: Data Partitioning with a Functional Performance Model of Heterogeneous Processors. Int. J. High Perform. Comput. Appl. 21(1), 76–90 (2007)
9. Lastovetsky, A., Reddy, R., Rychkov, V., Clarke, D.: Design and Implementation of Self-Adaptable Parallel Algorithms for Scientific Computing on Highly Heterogeneous HPC Platforms. [cs.DC] (2011), arXiv:1109.3074v1
10. Lastovetsky, A., Reddy, R.: Distributed Data Partitioning for Heterogeneous Processors Based on Partial Estimation of Their Functional Performance Models. In: Lin, H.-X., Alexander, M., Forsell, M., Knüpfer, A., Prodan, R., Sousa, L., Streit, A. (eds.) Euro-Par 2009. LNCS, vol. 6043, pp. 91–101. Springer, Heidelberg (2010)
11. Lastovetsky, A., Reddy, R.: Two-Dimensional Matrix Partitioning for Parallel Computing on Heterogeneous Processors Based on Their Functional Performance Models. In: Lin, H.-X., Alexander, M., Forsell, M., Knüpfer, A., Prodan, R., Sousa, L., Streit, A. (eds.) Euro-Par 2009. LNCS, vol. 6043, pp. 112–121. Springer, Heidelberg (2010)
12. Zhuo, L., Prasanna, V.K.: Optimizing Matrix Multiplication on Heterogeneous Reconfigurable Systems. In: PARCO 2007, pp. 561–568 (2007)