Multilevel Hierarchical Matrix Multiplication on Clusters

Sascha Hunold
Department of Mathematics, Physics and Computer Science
University of Bayreuth, Germany

Thomas Rauber
Department of Mathematics, Physics and Computer Science
University of Bayreuth, Germany

Gudula Rünger
Department of Computer Science
Chemnitz University of Technology, Germany

ABSTRACT

Matrix-matrix multiplication is one of the core computations in many algorithms from scientific computing or numerical analysis and many efficient realizations have been invented over the years, including many parallel ones. The current trend to use clusters of PCs or SMPs for scientific computing suggests to revisit matrix-matrix multiplication and investigate efficiency and scalability of different versions on clusters. In this paper we present parallel algorithms for matrix-matrix multiplication which are built up from several algorithms in a multilevel structure. Each level is associated with a hierarchical partition of the set of available processors into disjoint subsets so that deeper levels of the algorithm employ smaller groups of processors in parallel. We perform runtime experiments on several parallel platforms and show that multilevel algorithms can lead to significant performance gains compared with state-of-the-art methods.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*Parallel programming*; G.1.0 [**Numerical Analysis**]: General—*Parallel algorithms*

General Terms

Algorithms, Performance

Keywords

Strassen's algorithm, matrix multiplication, multiprocessor tasks, task parallelism

1. INTRODUCTION

Matrix-matrix multiplication is one of the core computations in many algorithms from scientific computing or numerical analysis. On a single processor ATLAS [12] or PHiPAC [1] create very efficient implementations by adjusting the computation order to the specific memory hierarchy and its properties. Parallel approaches include many methods based on decomposition like Cannon's algorithm, or the algorithm of Fox. Efficient implementation variants of the latter are SUMMA or PUMMA, see [11] for more references. Matrix-matrix multiplication by Strassen or Strassen-Winograd benefits from a reduced number of operations but require a special schedule for a parallel implementation. Several parallel implementations have been proposed in [3, 6, 8].

The current trend to use clusters of PCs or SMPs for scientific computing suggests to revisit matrix-matrix multiplication and to investigate efficiency and scalability of different versions on clusters. In this context mixed programming models like mixed task and data parallelism are important since efficiency and scalability can be improved by using multiprocessor tasks (M-tasks). Task parallel implementations or mixed matrix-matrix multiplications have already been proposed in literature. One possibility of parallelizing Strassen's algorithm is to distribute the seven intermediate results onto a group of processors of size 7^i, preferably in a ring or torus configuration [2, 4]. Other approaches include to mix the common Fox BMR method (broadcast multiply roll) with Strassen's algorithm [8]. Another mixed parallel algorithm that exploits the complexity reduction of Strassen's algorithm on the top level and combines it with the performance of ScaLAPACK on the bottom layer is given in [3].

In this paper we consider matrix-matrix multiplication on clusters of PCs or SMPs and investigate the performance of several parallel hierarchical algorithms and their realizations. In particular, we investigate new multilevel algorithms with different building blocks, including well-known parallel algorithms like Strassen multiplication as well as new algorithms that have been designed to exploit the memory hierarchy of recent microprocessors by an increased locality of memory references. We show that a suitable combination of the building blocks can lead to significant performance improvements compared with an execution of the algorithms in isolation. The building blocks are expressed as M-tasks to exploit mixed task and data parallelism. The combination of the M-tasks is performed with the Tlib library [9].

The composed methods use the Strassen algorithm on the upper level to create coarse-grained M-tasks that are assigned to disjoint processor groups. For the intermediate level efficient variants like PDGEMM from ScaLAPACK and a new hierarchical decomposition-based algorithm tpMM are used. For the lowest level we use BLAS or ATLAS for the multiplication of smaller blocks on single processors. It is a crucial point to choose a good schedule for Strassen and

ICS'04, June 26–July 1, 2004, Saint-Malo, France.
Copyright 2004 ACM 1-58113-839-3/04/0006 ...$5.00.

to pick the right cutoff of the hierarchical decomposition for the coupling of the different levels. Depending on the cluster platform or parallel machine different strategies lead to the most efficient implementation.

The rest of the paper is organized as follows. Section 2 resumes the matrix multiplication algorithms used as building blocks. Section 3 describes different combinations of the building blocks to multilevel hierarchical methods. Section 4 gives some runtime estimations and Section 5 shows experimental results on different clusters. Section 6 concludes the paper.

2. BUILDING BLOCKS

In this section, we give a short overview of the algorithms that are used as building blocks for multilevel algorithms.

2.1 Strassen's Algorithm

Strassens algorithm is used as a building block for the top level of the multilevel methods. The algorithm is discussed intensively in the literature and we only give a short summary of the method for notational reasons [5, 10]. If the matrices A and B are of dimension $R^{n\times n}$ with an even n, the matrix product $C = A \times B$ can be expressed as:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \tag{1}$$

where C_{11}, C_{12}, C_{21}, and C_{22} are determined by:

$$\begin{aligned} Q_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ Q_2 &= (A_{21} + A_{22})B_{11} \\ Q_3 &= A_{11}(B_{12} - B_{22}) \\ Q_4 &= A_{22}(B_{21} - B_{11}) \\ Q_5 &= (A_{11} + A_{12})B_{22} \\ Q_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ Q_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned} \tag{2}$$

and

$$\begin{aligned} C_{11} &= Q_1 + Q_4 - Q_5 + Q_7 \\ C_{12} &= Q_3 + Q_5 \\ C_{21} &= Q_2 + Q_4 \\ C_{22} &= Q_1 + Q_3 - Q_2 + Q_6 \end{aligned} \tag{3}$$

This scheme reduces the time complexity of a matrix-matrix multiplication from $O(n^3)$ to $O(n^{2.8})$, if the algorithm is applied fully recursively. Another approach is the so-called Strassen-Winograd variant of this algorithm [5]. The method by Winograd minimizes the required matrix additions in equation 2 which reduces the overall complexity.

2.2 Task parallel matrix multiplication (tpMM)

In the following, we give a short description of tpMM. A more detailed description and an evaluation of tpMM in isolation is given in [7]. tpMM is designed to work with $p = 2^i$ processors for arbitrary $i \in \mathbb{N}$. The processors are hierarchically grouped into clusters of size 2^{l-1}, $1 \le l \le \log p + 1$. This results in a group hierarchy with $\log p + 1$ levels in which the leaf groups at level 1 contain single processors. The groups are successively combined with one of their neighboring groups to form larger and larger groups until the root group contains all processors.

The input matrices A and B are of size $m \times n$ and $n \times k$ where p divides m and k without remainder; the result matrix C is of dimension $m \times k$. The initial data distribution is a row block-wise distribution for matrix A and a column block-wise distribution for matrix B, so that each processor q stores a block of $s = m/p$ rows of A and $s' = k/p$ columns of B. In addition, the initial column blocks of B are virtually grouped into larger column blocks according to the hierarchical clustering of the processors into groups.

The algorithm tpMM computes the result matrix C in $\log p + 1$ steps. In step l, $1 \le l \le \log p + 1$, the $2^{\log p - l + 1}$ processor groups G_{lk}, $1 \le k \le p/2^{l-1}$, work in parallel. Processor group G_{lk} computes the diagonal block C_{lk} of C, which contains the $(s \cdot 2^{l-1})^2$ entries c_{ij} with $2^{l-1} \cdot (k-1) \cdot s + 1 \le i, j \le 2^{l-1} \cdot k \cdot s$. In summary, each processor q is responsible for the computation of s rows of the result matrix C.

The computation of a diagonal block C_{lk} by the processor group G_{lk} is performed in parallel by all processors in G_{lk}. If G_{lk} consists of a single processor q, this processor computes one initial diagonal block C_{lk} by using its local entries of A and B. Otherwise, the computation of the two diagonal sub-blocks $C_{l-1,2k-1}$ and $C_{l-1,2k}$ of C_{lk} have already been completed in the preceding step by the two subgroups $G_{l-1,2k-1}$ and $G_{l-1,2k}$ of G_{lk} and the computation of C_{lk} is completed by computing the remaining sub-blocks $C'_{l-1,2k-1}$ and $C'_{l-1,2k}$.

To do this, the column blocks $B_{l-1,2k-1}$ and $B_{l-1,2k}$ of matrix B that are needed for the computation of $C'_{l-1,2k-1}$ and $C'_{l-1,2k}$, respectively, are first exchanged between the processors of the corresponding groups. This can be done in parallel since the processors of the group can be grouped into pairs which exchange their data. After the transfer operations are completed, the sub-groups $G_{l-1,2k-1}$ and $G_{l-1,2k}$ of G_{lk} compute the sub-blocks $C'_{l-1,2k-1}$ and $C'_{l-1,2k}$, respectively, in parallel by recursive calls.

At each point in time, each local memory needs to store at most s rows of A and s' columns of B. Only columns of B are exchanged between the local memories, see Figure 1 for an illustration.

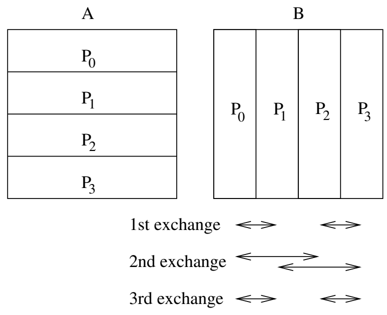

Figure 1: Communication pattern of tpMM.

2.3 Ring method

The Ring method is based on the same initial partitioning of the matrices A and B among the processors as algorithm tpMM, but it does not use the hierarchical organization. This has the advantage that an arbitrary number p of processors can be used. Similar to tpMM, only column blocks of B are exchanged between the processors, here in a ring-like way, see Figure 2. For the computation of C, processor P_i, $i = 0, \ldots, p-1$, uses its row block of A and its current column block of B to compute one sub-block of C. After this computation, P_i receives a new block of B from $P_{(i-1)\%p}$ and sends its current block of B to $P_{(i+1)\%p}$. P_i can then compute the next block of C. After $p-1$ steps, the complete matrix C has been computed in a distributed way.

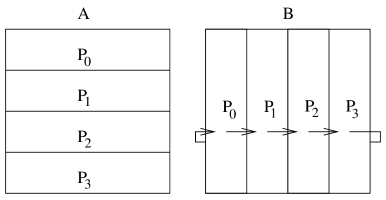

Figure 2: Communication pattern of the ring method.

2.4 PDGEMM

PDGEMM is a function declaration from the PBLAS set that was developed as a part of the ScaLAPACK project. There exist numerous implementations of this function, vendor-specific or free realizations as in ScaLAPACK. The algorithm that lies behind this function interface differs in most libraries. PDGEMM is available on almost all parallel systems and has become the de-facto standard for fast parallel matrix-matrix multiplication. We use this algorithm as a building block for multilevel algorithms to be considered in the next section.

3. MULTILEVEL COMBINATIONS

In this section, we describe the multilevel combinations of the building blocks with up to three levels. The top level may consist of one or more recursions of the Strassen algorithm accompanied by a recursive splitting of processors into disjoint groups. The intermediate level is realized by one of the algorithms tpMM, PDGEMM or Ring. And the lower level picks an appropriate one-processor implementation of matrix-matrix multiplication to exploit the processor most efficiently. For parallel subtasks (i.e. the processor group contains more than one processor) PDGEMM, tpMM or the Ring method are used. Local matrix updates are performed by an optimized BLAS implementation, either ATLAS on Linux Clusters or ESSL on the *IBM Regatta p690*. The resulting hierarchy of algorithms is shown in Fig. 3.

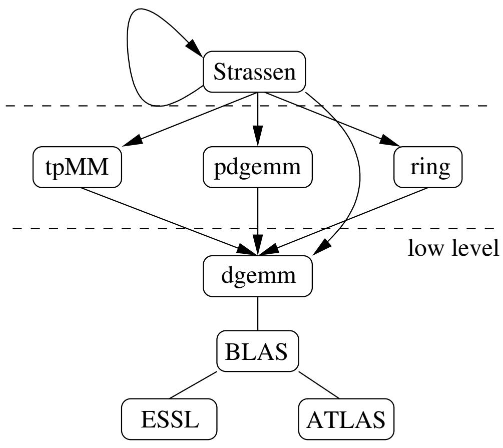

Figure 3: Computational hierarchy.

3.1 Multiprocessor-tasks for Strassen's algorithm

For the definition of multiprocessor tasks we use a result-oriented view on Strassen's algorithm. According to formula (3), four concurrent tasks are formed where each task is responsible to complete one quarter C_{ij}, $i, j = 1, 2$, of the result matrix C. The computations of the sub-problems Q_i, $i = 1, \ldots, 7$, are assigned to these tasks according to two allocations schemes:

scheme (1) Each task $T_{C_{ij}}$ performs two sub-computations Q_i and all tasks are assigned with an equal number of processors.

scheme (2) Three tasks compute two sub-problems and one task is responsible for only one intermediate result. Therefore, the available processors are distributed proportionally according to each task's amount of work, i.e. a task that has to compute two sub-results receives 2/7 of the available processors.

Both cases entail positive and negative impacts on the workload and the communication overhead.

Advantages of scheme (1). To balance the workload a redundant computation is introduced. It leads to an increasing overall complexity of the algorithm. But there are also improvements. Since all tasks in one recursion level are processed by the same number of processors the communication pattern is kept simple. Moreover replicated tasks also lead to less communication because sub-results already reside on the right task (if carefully chosen).

Advantages of scheme (2). The second case avoids replicated tasks and so it can take full advantage of Strassen's algorithm reducing the number of multiplications required. On the other hand an unequal number of processors per task leads to more communication needed to exchange data between tasks.

For the parallel processing of one recursion step of Strassen's algorithm we divide the set of processors into four disjoint groups and assign the tasks $T_{C_{ij}}$ to those groups. We assume that the processors assigned to task $T_{C_{ij}}$ also store the sub-matrices A_{ij} and B_{ij},. A dependence analysis has shown that a minimum communication overhead can be achieved by the specific allocation schemes in Table 1 for scheme (1) and Table 2 for scheme (2), respectively.

Table 1: Composition of tasks $T_{C_{ij}}$ for Scheme (1).

$T_{C_{11}}$	$T_{C_{12}}$	$T_{C_{21}}$	$T_{C_{22}}$
Q_1	Q_3	Q_4	Q_1
Q_7	Q_5	Q_2	Q_6

Table 2: Composition of tasks $T_{C_{ij}}$ for Scheme (2).

$T_{C_{11}}$	$T_{C_{12}}$	$T_{C_{21}}$	$T_{C_{22}}$
Q_1	Q_3	Q_4	Q_6
Q_7	Q_5	Q_2	

A more detailed description of the task structure is presented in Table 3 for scheme (1) and Table 4 for scheme (2). The replicated assignment of sub-task Q_1 to task $T_{C_{22}}$ in Table 3 for scheme (1) minimizes the required data exchange during the execution of the algorithm.

Table 3: Internal task structure of the multilevel version of Strassen's algorithm for Scheme (1).

Task $T_{C_{11}}$	Task $T_{C_{12}}$	Task $T_{C_{21}}$	Task $T_{C_{22}}$
Send B_{11}	Recv B_{22}	Recv B_{11}	Send B_{22}
Recv B_{22}	Send B_{22}	Send B_{11}	Recv B_{11}
Send A_{11}	Recv A_{11}	Recv A_{22}	Send A_{22}
Recv A_{22}	Send A_{11}	Send A_{22}	Recv A_{11}
Recv A_{12}	Send A_{12}	Send A_{21}	Recv A_{21}
Recv B_{21}	Send B_{12}	Send B_{21}	Recv B_{12}
$T_1 = B_{11} + B_{22}$	$T_1 = B_{12} - B_{22}$	$T_1 = B_{21} - B_{11}$	$T_1 = B_{11} + B_{22}$
$T_2 = A_{11} + A_{22}$	Strassen(Q_3, A_{11}, T_1)	Strassen(Q_4, A_{22}, T_1)	$T_2 = A_{11} + A_{22}$
Strassen(Q_1, T_2, T_1)	$T_1 = A_{11} + A_{12}$	$T_1 = A_{21} + A_{22}$	Strassen(Q_1, T_2, T_1)
$T_1 = A_{12} - A_{22}$	Strassen(Q_5, T_1, B_{22})	Strassen(Q_2, T_1, B_{11})	$T_1 = A_{21} - A_{11}$
$T_2 = B_{21} + B_{22}$			$T_2 = B_{11} + B_{12}$
Strassen(Q_7, T_1, T_2)			Strassen(Q_6, T_1, T_2)
Recv Q_4	Send Q_3	Send Q_4	Recv Q_3
Recv Q_5	Send Q_5	Send Q_2	Recv Q_2
$T_1 = Q_1 + Q_7$	$C_{12} = Q_3 + Q_5$	$C_{21} = Q_3 + Q_5$	$T_1 = Q_1 + Q_6$
$T_1 = T_1 + Q_4$			$T_1 = T_1 + Q_3$
$C_{11} = T_1 - Q_5$			$C_{22} = T_1 - Q_2$

Table 4: Internal task structure of the multilevel version of Strassen's algorithm for Scheme (2).

Task $T_{C_{11}}$	Task $T_{C_{12}}$	Task $T_{C_{21}}$	Task $T_{C_{22}}$
Send A_{11}	Recv A_{11}	Recv A_{22}	Send A_{22}
Recv A_{22}	Send A_{11}	Send A_{22}	Recv A_{11}
Send B_{11}	Recv B_{22}	Recv B_{11}	Send B_{22}
Recv A_{12}	Send A_{12}	Send B_{11}	Recv B_{11}
Recv B_{21}	Send B_{12}	Send B_{21}	Recv B_{12}
Recv B_{22}	Send B_{22}	Send A_{21}	Recv A_{21}
$T_1 = A_{11} + A_{22}$	$T_1 = B_{12} - B_{22}$	$T_1 = B_{21} - B_{11}$	$T_1 = A_{21} - A_{11}$
$T_2 = B_{11} + B_{22}$	Strassen(Q_3, A_{11}, T_1)	Strassen(Q_4, A_{22}, T_1)	$T_2 = B_{11} + B_{12}$
Strassen(Q_1, T_2, T_1)	$T_1 = A_{11} + A_{12}$	$T_1 = A_{21} + A_{22}$	Strassen(Q_6, T_1, T_2)
$T_1 = A_{12} - A_{22}$	Strassen(Q_5, T_1, B_{22})	Strassen(Q_2, T_1, B_{11})	
$T_2 = B_{21} + B_{22}$			
Strassen(Q_7, T_1, T_2)			
Send Q_1			Recv Q_1
Recv Q_4	Send Q_3	Send Q_4	Recv Q_3
Recv Q_5	Send Q_5	Send Q_2	Recv Q_2
$T_1 = Q_1 + Q_7$	$C_{12} = Q_3 + Q_5$	$C_{21} = Q_3 + Q_5$	$T_1 = Q_1 + Q_6$
$T_1 = T_1 + Q_4$			$T_1 = T_1 + Q_3$
$C_{11} = T_1 - Q_5$			$C_{22} = T_1 - Q_2$

The subdivision of Strassens algorithm represents the hierarchical top-level. As one can see in Table 3 and Table 4 a task parallel execution scheme of Strassen's algorithm intensifies the number of communications. To analyze the impact of different cutoff levels of Strassen, that is, the level when the recursion of Strassen is stopped and another algorithm takes over to solve the sub-problem, we propose well-suited algorithms at lower levels.

The next subsection describes the algorithms Strassen-tpMM, Strassen-PDGEMM, and Strassen-Ring. Due to the algorithmic structure, Strassen-tpMM and Strassen-PDGEMM use the scheme from Table 1. Strassen-Ring uses the scheme from Table 2.

3.2 Combining Strassen and tpMM

For the combination of Strassen with tpMM, we assume that the number p of processors can be represented as $p = 4^i 2^j$ for $i \geq 1$ and $j \geq 0$ where $l = 4^i$ reflects the fact that four new processor groups are built at each recursion level of the Strassen algorithm and 2^j processors are used for the execution of tpMM after the Strassen recursion has stopped. The input matrices A and B have size $n \times n$ with $n \geq p$. For simplicity, we assume $n = l \cdot 2^k$.

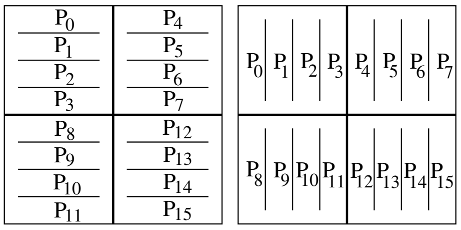

Figure 4: Mapping of processors to matrices A and B for *Strassen_tpmm* using 16 processors

The processors are arranged in two virtual rectangular grids, one for matrix A and one for matrix B. The grids are chosen such that after the Strassen recursion has stopped, matrix A is distributed row-blockwise and B is distributed column-blockwise, as it is required by tpMM. The processor grid for B with r rows and c columns with $r \cdot c = p$ is chosen such that $r \leq c$ and $(r,c) = \underset{(r,c)}{min}\{c - r | r \cdot c = p\}$, i.e., a quadratic layout is preferred. The processor grid for A is defined similarly with r' and c' and the roles of r' and c' exchanged. The processor grid for A is also used for the result matrix C. Altogether, each processor stores $\frac{n}{r'} \times \frac{n}{c'}$ elements of A and $\frac{n}{r} \times \frac{n}{c}$ elements of B, see Figure 4. The task parallel implementation uses a result-orientated decomposition into four sub-tasks $T_{C_{ij}}$ according to Table 1.

The implementation *Strassen_tpmm* calls Strassen until each sub-group contains four or less processors. A group size of four processors is chosen because experimental results have shown that tpMM always performs well on 4 processors. tpMM is built on top of BLAS, hence local matrix updates are performed by DGEMM included in the BLAS routines. We use the ATLAS implementation of DGEMM on Linux clusters (*Dual Xeon 3 GHz Cluster (BT)*, *Dual Xeon 2 GHz Cluster (C)*) and on *IBM Regatta p690* the library ESSL provided by IBM.

3.3 Combining Strassen and PDGEMM

For the number p of processors, we assume $p = 2^d \cdot 4^i$ with $i \geq 1$ and $d \in \{0,1\}$. The matrices A, B and C have size $n \times n$. The processors are mapped row-blockwise onto the blocks of A, B and C so that the blocks have size $\frac{n}{r} \times \frac{n}{c}$. Figure 5 shows an example of a virtual grid corresponding to a first-level morton ordering due to the algorithmic structure of *Strassen_pdgemm*.

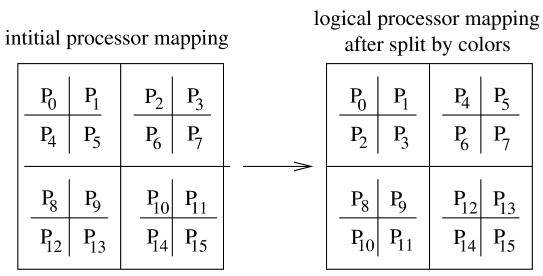
intitial processor mapping
logical processor mapping after split by colors
P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15
P0 P1 P4 P5 P2 P3 P6 P7 P8 P9 P12 P13 P10 P11 P14 P15

Figure 5: Mapping of 16 processors to sub-blocks of matrices A, B and C in order to perform one level of Strassen in *Strassen_pdgemm*.

The top-level Strassen method is called as often as specified by the user. The maximum number of recursions is $\log_4 p$. When the Strassen recursion stops, PDGEMM continues working on the sub-problem if there is more than one processor in each group. If the group size is one, DGEMM is called directly.

3.4 Combining Strassen and Ring

The combination of Strassen and Ring is derived from the *Strassen_tpmm* approach. On the bottom level both algorithms use a similar computational structure, but a different communication scheme. The unbalanced work distribution resulting from the scheme in Table 2 requires an uneven distribution of matrix B, see Figure 6. Although the algorithm can be executed for an arbitrary number of processors, at least 7 processors should be used for the algorithm to be efficient.

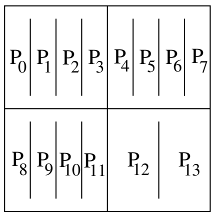

Figure 6: Mapping of 14 processors to sub-blocks of B for *Strassen_ring*.

4. THEORETICAL EVALUATION

In this section, we evaluate the costs of Strassen in terms of time complexity. We consider the complexity functions for scheme (1) and scheme (2) separately. The value of l_c, the number of recursions of the top level Strassen, is

$$l_c \leq \begin{cases} \lfloor \log_4 p \rfloor, \text{for scheme (1)} \\ \lfloor \log_7 p \rfloor, \text{for scheme (2)} \end{cases}$$

4.1 Communication costs

The communication costs are evaluated using the formula $\alpha + m \cdot \beta$ for single transfer operations where α is the network latency, β corresponds to the inverse of the network bandwidth and m is the message length.

4.1.1 Scheme (1)

The time spent on communication operations in this case can be determined from the structure of task $T_{C_{11}}$ (Table 3). Eight communications are required to distribute the submatrices at the start of the task and to receive the necessary subresults at the end. A communication can be performed perfectly in parallel due to the group structure's one-to-one mapping of processors of different groups. Hence,

$$T_{Comm}(n,p) = l_c \cdot 8 \cdot \left(\alpha + \frac{n^2}{p}\beta\right) \quad (4)$$

4.1.2 Scheme (2)

The bottleneck of the task parallel version of Strassen-ring is the use of intergroup communications, e.g. for distributing the submatrices A_{ij}, B_{ij} and the exchange of Q_{ij} and C_{ij}. Since in each step the main group is split into 4 subgroups according to scheme (2), see Section 3.1, $3 \cdot \frac{2}{7}$ of the available processors are assigned to three tasks and the rest of $\frac{1}{7}$ of the processors is assigned to task $T_{C_{22}}$. In this case, the number of matrix elements assigned to processors that need to exchange submatrices might differ, e.g. processors from task $T_{C_{11}}$ hold less elements than processors executing task $T_{C_{22}}$. Thus, communication between subgroups gets more irregular and may lead to serialization. We examine the worst case behavior.

After the first split of p processors, three of the subgroups (s_i, $i = 0 \ldots 3$) contain

$$\left\lfloor \frac{2}{7} \cdot p \right\rfloor \leq p_{s_i} \leq \left\lfloor \frac{2}{7} \cdot p \right\rfloor + 1$$

processors and the fourth subgroup contains $\lceil \frac{p}{7} \rceil$ processors. To gain information about the maximum degree of serialization it is sufficient to consider the smallest and largest number of processors of any group, which are $\lceil \frac{p}{7} \rceil$ and $\lfloor \frac{2}{7} \cdot p \rfloor + 1$, respectively. Since each processor of the smaller group receives a message from corresponding processors of the larger group, the estimation

$$\frac{\lfloor \frac{2}{7} \cdot p \rfloor + 1}{\lceil \frac{p}{7} \rceil} \leq \frac{2 \cdot p + 7}{p} = 2 + \frac{7}{p} \leq 3 \,, \quad \text{for } p \geq 7 \qquad (5)$$

shows that the maximum degree of serialization is 3, i.e. for one intergroup communication a processor from the largest group sends at most 3 messages to members from the smallest group. In practice, this factor is usually smaller than 2 when arranged with non-blocking calls to `MPI_Isend` and `MPI_Irecv`.

For scheme (2) there is also one additional communication in every recursion level since the redundant computation of Q_1 is avoided and so, the task $T_{C_{22}}$ has to receive 3 subresults Q_1, Q_2 and Q_3. Therefore, the task $T_{C_{22}}$ dominates the communication time. According to that, the time complexity for communication in this case is

$$T_{Comm}(n,p) \leq 9 \cdot l_c \cdot 3 \cdot \big(\alpha + \frac{n^2}{p}\beta\big) \qquad (6)$$

4.2 Cost models

With the overview about the time that is consumed by data transfer operations we can now examine the general complexity of the various versions of Strassen. Since the addition and the multiplication of two doubles are in general not performed in the same number of CPU cycles we will distinguish them as τ_a and τ_m respectively.

Cost functions for algorithms that follow scheme (1). A full recursion in the task parallel sense might occur in the case of `Strassen_pdgemm`, when there are $p = 4^i$, $i \geq 1$ processors available. After splitting a group that contains exactly 4 processors the program of `Strassen_pdgemm` applies directly the DGEMM routine for the local matrix-matrix multiplication. In this configuration a processor holds matrix blocks of size $\frac{n^2}{p}$. The overall complexity of Strassen that executes a full recursion (fr) for scheme (1) is:

$$\text{Strassen\_1\_fr}(n,p) = T_{Comm}(n,p) + 7l_c \cdot \big(\frac{n^2}{p}\tau_a\big) + 2^{l_c}\frac{n^3}{p\sqrt{p}} \cdot \big(\tau_m + \tau_a\big)$$

Strassen takes advantage of other parallel algorithms after performing a few recursion steps. At the cutoff level l_c other algorithms such as tpMM and PDGEMM complete the computation. Hence, the time for these algorithm is composed of the time spent in the upper level of Strassen and the lower level algorithms. The overall complexity for scheme (1) is

$$\begin{aligned}\text{Strassen\_1}(n,p) &= T_{Comm}(n,p) + 7 \cdot l_c \cdot \big(\frac{n^2}{p}\tau_a\big)\\ &+ 2^{l_c}\big\{\text{tpMM}(n',p'),\ \text{PDGEMM}(n',p')\big\} \qquad (7)\\ &\text{where } n' = \frac{n}{2^{l_c}} \text{ and } p' = \frac{p}{4^{l_c}}.\end{aligned}$$

Complexity functions for tpMM and PDGEMM will complete equation 7. In tpMM all processors own a block of $\frac{n^2}{p}$ elements of the $n \times n$ matrices A, B and C. At the start tpMM performs a local multiplication and exchanges data with the corresponding group, performs another multiplication and exchanges data. Together, tpMM needs p local updates and $2(p-1)$ communications (send block + recv block).

$$\text{tpMM}(n,p) = 2(p-1)\big(\alpha + \frac{n^2}{p}\beta\big) + \frac{n^3}{p}\big(\tau_m + \tau_a\big)$$

PDGEMM uses communication schemes of higher complexity such as *broadcasts*. That is why we need a few more variables for modeling the algorithm. We assume that the processor grid has size $r \times q$, where r denotes the number of rows and q the number of columns. In addition the network parameters α and β introduced above are further to differentiate. The values α_q^r and β_q^r (α_r^q and β_r^q) depend on the grid topology and communication pattern in a row (column). For the local multiplications (implemented with DGEMM) we use *time(DGEMM)* as done in [3]. The parameter *nb* denotes the blocking factor.

$$\begin{aligned}\text{PDGEMM}(n,p) &= \text{time}\big(\text{DGEMM}, \lceil\frac{n}{p}\rceil, \lceil\frac{n}{p}\rceil, n\big)\\ &+ \big(\alpha_q^r + \alpha_r^q\big)n^2 + \big(\beta_q^r + \beta_r^q\big) \cdot \lceil\frac{n}{nb}\rceil\end{aligned}$$

Cost functions for Strassen and Ring. If the algorithm is not performed with exactly four processors a full recursion as we have discussed in the last paragraph is not going to happen. Due to the fractioning of the number of processors either by $\frac{2}{7}$ or $\frac{1}{7}$ the group size depends on the assignment strategy of left over processors. Therefore, we present an upper bound:

$$\begin{aligned}&\text{Strassen\_ring}(n,p) \leq T_{Comm}(n,p)\\ &+ max\big\{l_c \cdot \big(\frac{n^2}{4^{l_c}}\tau_a\big) + l_c \cdot \frac{n^3}{8^{l_c}}\big(\tau_m + \tau_a\big),\\ &l_c \cdot \big(\frac{n^2}{4^{l_c}\frac{2}{7^{l_c}}p} \cdot \tau_a\big) + 2 \cdot l_c \cdot ring\left(\frac{n}{2^{l_c}}, \frac{2}{7^{l_c}}p\right)\big\}\\ &\text{and } \text{ring}(n,p) = O\left(tpmm(n,p)\right).\end{aligned}$$

5. EXPERIMENTAL EVALUATION

Performance tests for the algorithm from Section 3 have been were carried out on three platforms:

1. an IBM Regatta p690 cluster at the NIC Jülich,
2. a 48 Dual 3 GHz Xeon Linux cluster at the University of Bayreuth (BT), Gigbabit-Ethernet, 1 GB RAM per node,
3. and an SCI interconnected Linux cluster at the Technical University of Chemnitz (C), 16 nodes, Xeon 2 GHz.

The right choice of the low level BLAS implementation plays an important role for obtaining high performance. Since there is no hand-optimized DGEMM for Linux systems we use the automatically adjusted ATLAS routine (ATLAS version 3.6.0). On the *IBM Regatta p690* we use the ESSL function implementing DGEMM which is provided by IBM.

As MPI library we use MPICH (1.2.5) on the *Dual Xeon 3 GHz Cluster (BT)* and the SCI specific Scali MPI on the *Dual Xeon 2 GHz Cluster (C)*. The IBM Regatta comes with its own IBM implementation of MPI.

We present the following abbreviations of the different program versions that are used in the diagrams :

strassen_pdgemm refers to the top level Strassen that calls PDGEMM at the cutoff level. The number behind the suffix r specifies how many levels of recursions of Strassen's algorithm have been performed in parallel.

strassen_tpmm denotes Strassen's algorithm that calls tpMM when there are less than 4 processors in one processor group.

strassen_ring denotes the algorithm that implements Scheme (2). It starts with Strassen and calls Ring after $\log_7 p$ levels.

strassen refers to the same algorithm as strassen_pdgemm. However PDGEMM is never called, because the full recursion leads to a direct call of DGEMM. Again, the number behind r denotes the number of recursions.

PDGEMM stands for the ScaLAPACK routine.

Figure 7 and Figure 8 show the test results for the *Dual Xeon 3 GHz Cluster (BT)* and the *Dual Xeon 2 GHz Cluster (C)*, respectively. The results for both Linux clusters are very similar despite their different interconnection networks and different MPI implementations. The combinations of Strassen and tpmm as well as Strassen and Ring show the best results for 16 and 32 processors. In case of 16 processors, a full task parallel recursion of Strassen from Scheme (1) is as fast as the combinations of Strassen with tpMM and Ring. PDGEMM does not achieve such a high performance on the Linux clusters. As a result, the combination of Strassen and PDGEMM can also not compete with the other algorithms.

Figure 9 shows the MFLOPS/node measured on *IBM Regatta p690*. PDGEMM is the fastest algorithm for a smaller number of matrix elements ($n \leq 6144$). When the matrix dimension increases the combination of Strassen with Ring or tpMM achieves the best results. An exception can be observed for dimension 8192. For this size, the underlying implementation of DGEMM limits the performance of the combined algorithms: For dimension 8192, the data decomposition of Strassen combined with tpMM leads to submatrices A, B and C of size 2048×1024 ($p = 32$) and 2048×512 ($p = 64$). For these sizes, DGEMM exhibits a significant difference in performance:

m	n	k	MFLOPS
1024	2048	1024	~ 1959
512	2048	512	~ 1827
1024	1024	1024	~ 4068

In each recursion of Scheme (2) the processor groups are partitioned into 7 subgroups. If the number of processors is not a multiple of 7, some processors remain unassigned. Assigning them to any sub-group helps to make this group perform better but leads to unequal workload. In Figure 11 we examine the runtime of *Strassen_ring* with a varying number of processor pairs (7,8), (14,16), and (28,32). Except for a matrix dimension of 9216 a multiple of 7 processors results always in a larger MFLOPS rate. Thus, the experimental results show that using a number of processors that is a multiple of 7 leads to better parallel efficiency. A noticable performance loss can be observed for 7 processors at dimension 9216. In case of 7 processors, one processor has to store the entire submatrix of size 4608×4608. Additionally, this processor has to perform one recusion level where it has to hold 4 more submatrices of the same size. Thus, this processor makes use of about 1.1 GB of main memory which leads to swapping and as a result to less perfomance.

To gain information about the internal execution pattern of the algorithms the MPE library is used to trace MPI function calls and jumpshot-4 is used to display the recorded data. Figure 10 shows the profile of PDGEMM on the *Dual Xeon 3 GHz Cluster (BT)*. The profile of the complete runtime is given on the lefthand side. The lines between the first 4 rows are specific for jumpshot and represent a collection of arrows. The computation structure of PDGEMM is so fine-grained that we show a small portion of the diagram on the right which covers the pattern in the first 10 seconds. Darker portions in the right profile denote MPI communication calls, like `MPI_Send` or `MPI_Recv`.

The total time spent in each function can be retrieved using the histogram of the algorithm which is given in Figure 12. The darker bars on the left denote the communication time. Most of the time is consumed by DGEMM which is represented by the collection of tiny bars on the right.

The profile of *Strassen_tpmm* shows a different pattern, see Figure 13. The computation and communication structure is coarse-grained. The small arrows on the left show the initial communication phase of Strassen, see Table 3. The brighter bars in the center represent the local matrix multiplications and the darker rectangles denote communication. It can be observed that groups 1, 3, and 4 wait until group 2 (5th-8th processor from the top) is done. Especially processor 5 determines the overall performance. Since we have used a non-dedicated system for recording the profiles, the program might have to share one node with another user, which has been processor 5 for the tests presented.

The trace and the histogram of *Strassen_ring* are presented in Figure 14 and show very similar patterns as the profile of *Strassen_tpmm*. Hence, we only highlight the differences. Firstly, this trace uses only 14 processors in order to achieve a good workload balance. The trace shows the grouping of processors according to Scheme (2) into groups of processors of different sizes and also shows that the smallest group (2 processors on the bottom) is assigned to one subtask only.

Figure 15 pictures the communication and computation pattern of *Strassen_pdgemm*. The distribution of submatrices at the start of Strassen can be recognized in the first 10 sec. The thicker lines again represent preview arrows as explained above and thinner lines or bundles of arrows, as between processor 1 and 2, denote the transfer of messages. In the middle one can recognize the fine-grained structure of PDGEMM which dominates the overall execution time.

The traces show that the performance loss of PDGEMM on Xeon-Linux clusters is caused by slow local updates. Our experiments show that PDGEMM chooses an internal block size that is not large enough to reach the peak performance of the Xeon processor.

6. CONCLUSIONS

In this paper, we have shown that a suitable combination of different algorithms that are used as building blocks can be used to obtain multilevel algorithms which lead to significant performance improvements compared to an execution of the algorithms in isolation. In particular, it is possible to obtain algorithms that are nearly twice as fast on some cluster platforms as the original PDGEMM method from ScaLAPACK. An important point for the construction of the multilevel algorithms lies in the issue, which building blocks should be combined in which order and at which group size the building blocks are assembled. The evaluation shows that a combination of Strassen's method at the

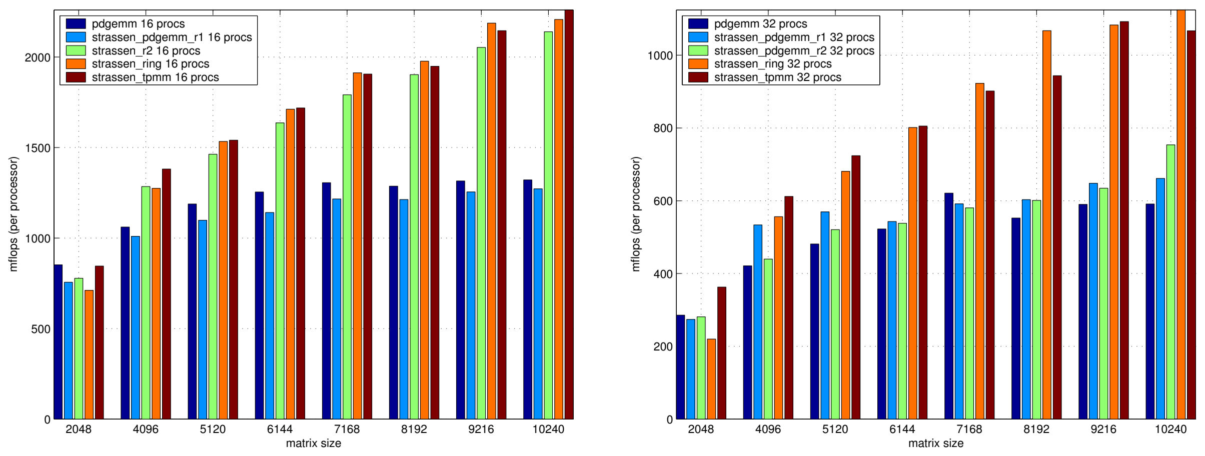

Figure 7: Experimental results on *Dual Xeon 3 GHz Cluster (BT)*.

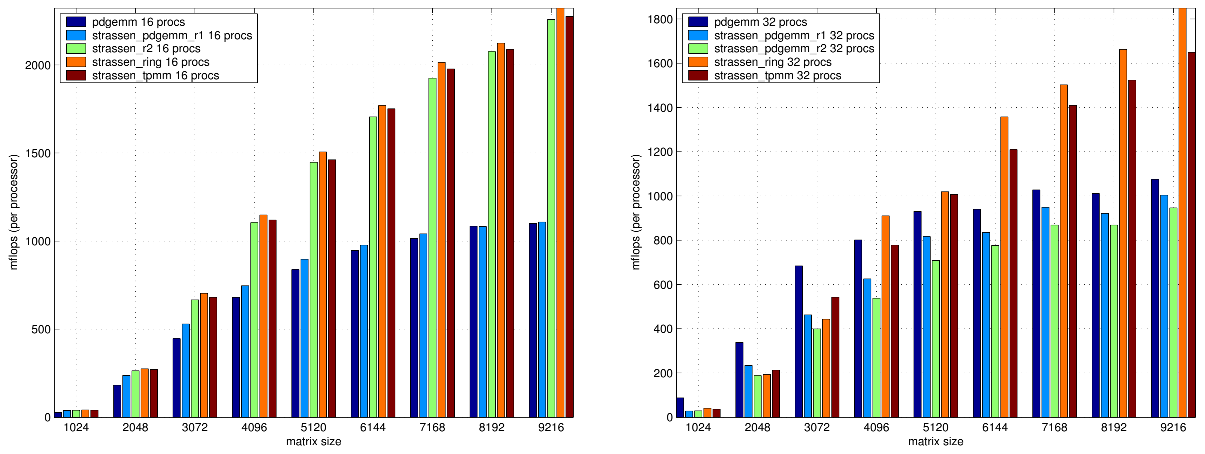

Figure 8: Experimental results on *Dual Xeon 2 GHz Cluster (C)*.

top level with special communication-optimized algorithms on the intermediate level and ATLAS or DGEMM at the bottom level leads to the best results.

7. REFERENCES

[1] J. Bilmes, K. Asanovic, C.-W. Chin, and J. Demmel. Optimizing matrix multiply using PHiPAC: A portable, high-performance, ANSI c coding methodology. In *Proc. of the International Conference on Supercomputing – ICS'97*, pages 340–347, 1997.

[2] C.-C. Chou, Y.-F. Deng, G. Li, and Y. Wang. Parallelizing Strassen's Method for Matrix Multiplication on Distributed-Memory MIMD Architectures. *Computers and Mathematics with Applications*, 30(2):49–69, 1995.

[3] F. Desprez and F. Suter. Impact of Mixed-Parallelism on Parallel Implementations of Strassen and Winograd Matrix Multiplication Algorithms. Technical Report RR2002-24, Laboratoire de l'Informatique du Parallélisme (LIP), June 2002. Also INRIA Research Report RR-4482.

[4] B. Dumitrescu, J.-L. Roch, and D. Trystram. Fast matrix multiplications algorithms on MIMD architectures. *Parallel Algorithms and Applications*, 4(2):53–70, 1994.

[5] G. Golub and C. Van Loan. *Matrix Computations.* The John Hopkins University Press, 1989.

[6] B. Grayson, A. Shah, and R. van de Geijn. A High Performance Parallel Strassen Implementation. Technical Report CS-TR-95-24, Department of Computer Sciences, The Unversity of Texas, 1, 1995.

[7] S. Hunold, T. Rauber, and G. Rünger. Hierarchical Matrix-Matrix Multiplication based on Multiprocessor Tasks. In *Proc. of the International Conference on Computational Science – ICCS 2004*, LNCS. Springer, June 2004.

[8] Q. Luo and J. B. Drake. A Scalable Parallel Strassen's Matrix Multiplication Algorithm for Distributed-Memory Computers. In *Proceedings of the 1995 ACM Symposium on Applied Computing*, pages 221–226. ACM Press, 1995.

[9] T. Rauber and G. Rünger. Library Support for Hierarchical Multi-Processor Tasks. In *Proc. of the Supercomputing 2002*, Baltimore, USA, 2002.

[10] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[11] R. A. van de Geijn and J. Watts. SUMMA: scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience*, 9(4):255–274, 1997.

[12] R. C. Whaley and J. J. Dongarra. Automatically Tuned Linear Algebra Software. Technical Report UT-CS-97-366, University of Tennessee, 1997.

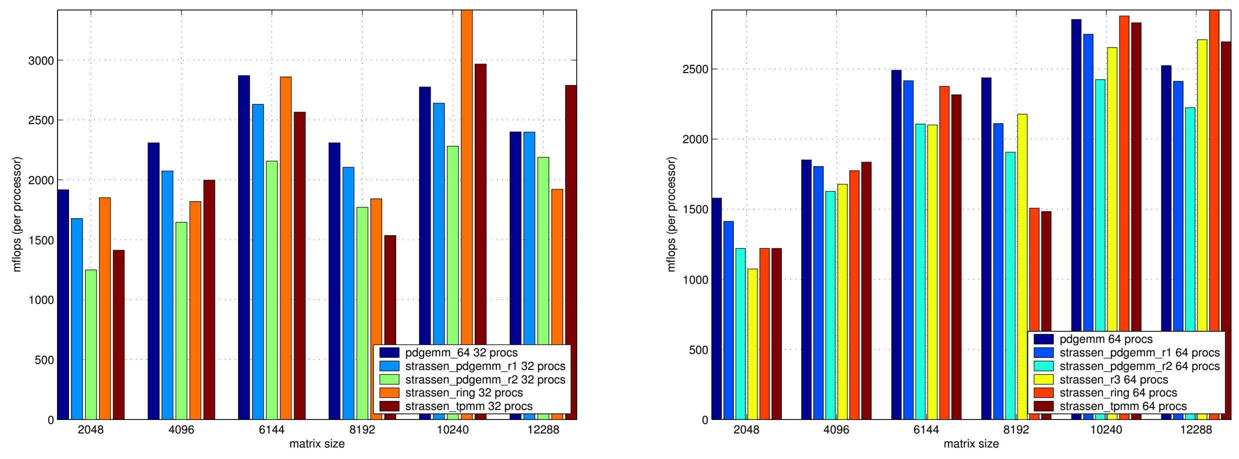

Figure 9: Experimental results on *IBM Regatta p690*.

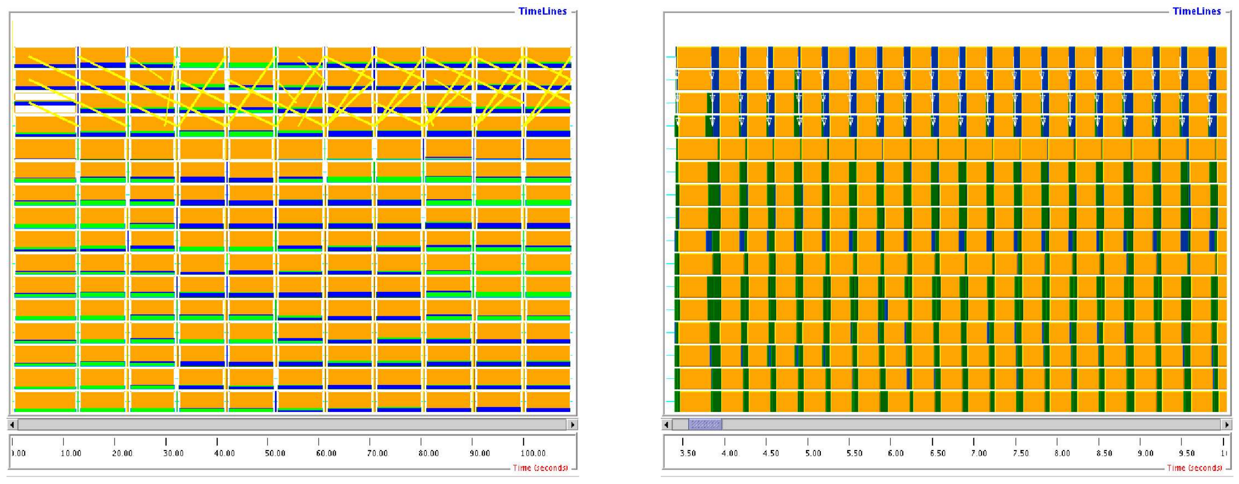

Figure 10: Trace of PDGEMM (left) and a specially-detailed trace for the time between 3 to 10 sec, p=16, n=10240, *Dual Xeon 3 GHz Cluster (BT)*.

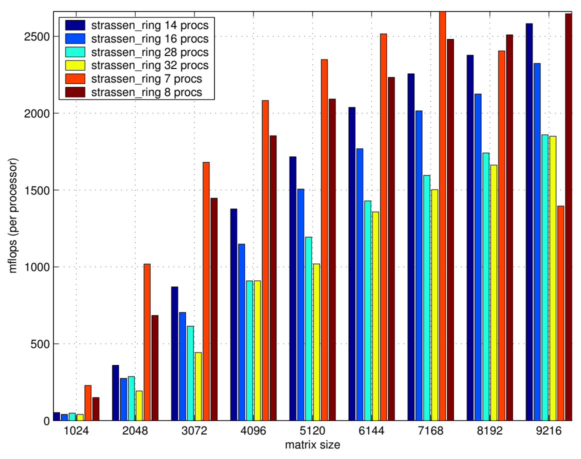

Figure 11: Efficiency comparison of Strassen_ring on *Dual Xeon 2 GHz Cluster (C)*.

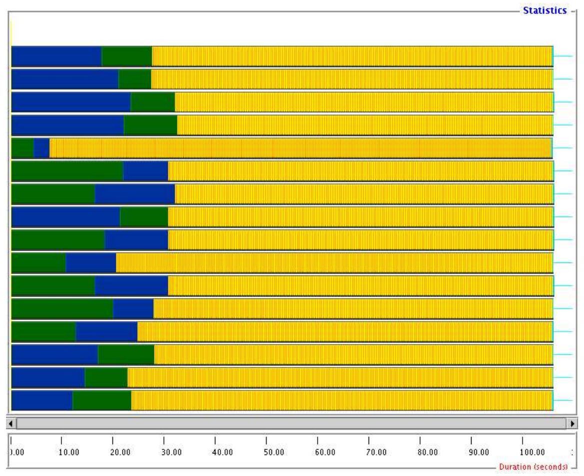

Figure 12: Histogram of PDGEMM, p=16, n=10240, *Dual Xeon 3 GHz Cluster (BT)*.

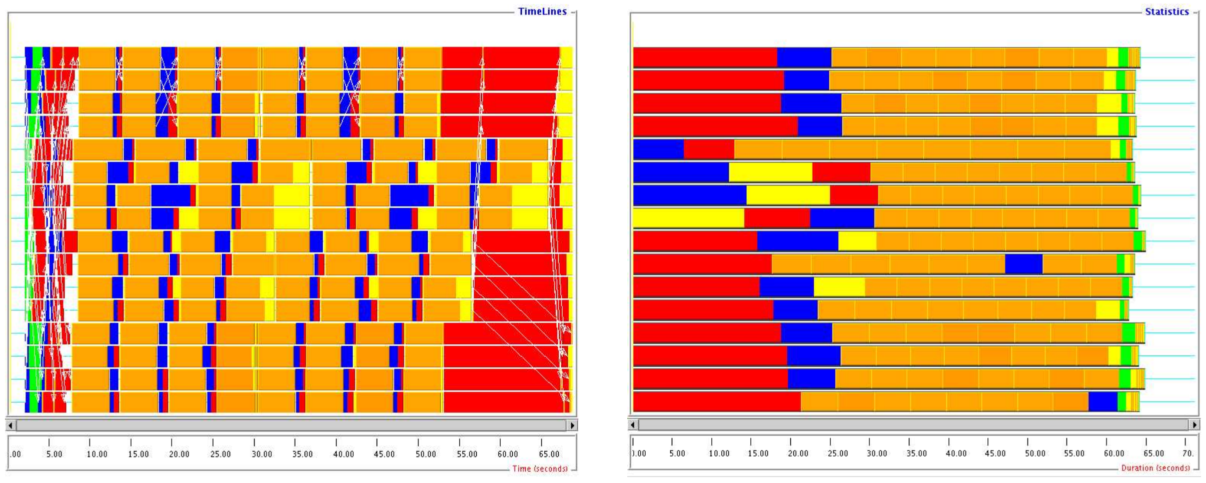

Figure 13: Trace and histogram of *Strassen_tpmm*, $p = 16, n = 10240$, *Dual Xeon 3 GHz Cluster (BT).*

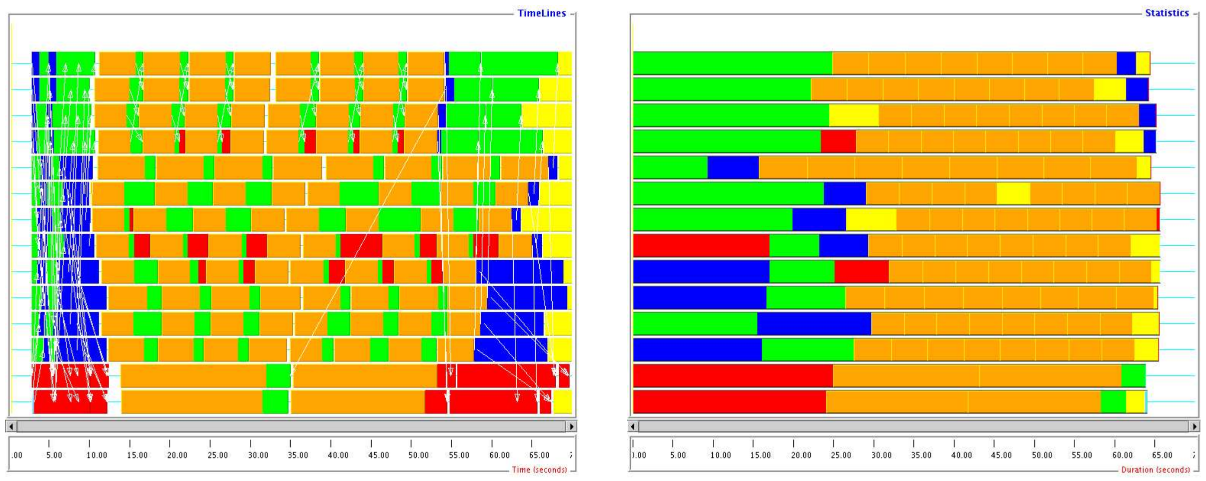

Figure 14: Trace and histogram of *Strassen_ring*, $p = 14, n = 10240$, *Dual Xeon 3 GHz Cluster (BT).*

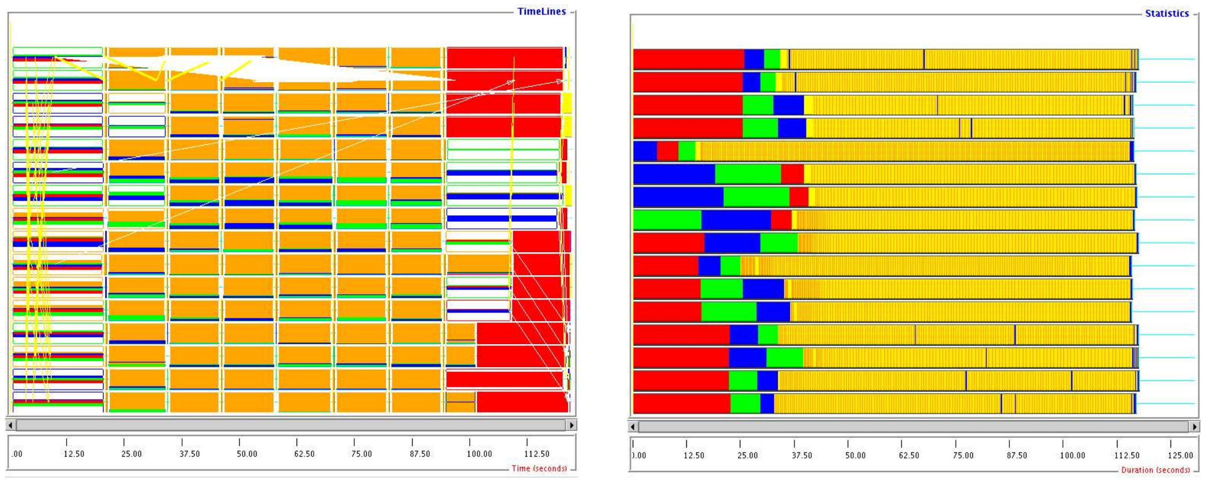

Figure 15: Trace and histogram of *Strassen_pdgemm*, one level of recursion (call to PDGEMM at group size of 4), $p = 16, n = 10240$, *Dual Xeon 3 GHz Cluster (BT).*