Combining Building Blocks for Parallel Multilevel Matrix Multiplication

Sascha Hunold\textsuperscript{1} \hspace{1cm} Thomas Rauber\textsuperscript{1} \hspace{1cm} Gudula Rünger\textsuperscript{2}

\textsuperscript{1} Department of Computer Science
University of Bayreuth

\textsuperscript{2} Department of Computer Science
Chemnitz University of Technology

PMAA'06, Rennes
Outline

1 Introduction

2 Specific Approach

3 Execution environment: Tlib

4 Algorithm combinations
   - Recursive splitting – Strassen algorithm
   - tpmm - a new recursive algorithm for memory hierarchies
   - Tuning of PDGEMM for better performance
   - Combination of different algorithms

5 Experimental results

6 Conclusions
matrix multiplication is one of the core computations in many algorithms from scientific computing; → many efficient realizations have been invented;

efficient sequential implementations: ATLAS, PHiPAC;
efficient parallel implementations: SUMMA, PDGEMM;

execution platforms differ in important characteristics memory hierarchy, communication architecture
efficient implementation must exploit these characteristics;
→ adaptivity is an important property;
Introduction

- **multiprocessor task** (M-task) implementations have been successful for many platforms for different application areas;
  - **advantage**: reduction in communication time, better scalability;
- **approach**: M-task implementation of matrix multiplication;
- **goal**: competitiveness with ScaLAPACK and vendor-specific implementations;
1 Introduction

2 Specific Approach

3 Execution environment: Tlib

4 Algorithm combinations
   - Recursive splitting – Strassen algorithm
   - tpmm - a new recursive algorithm for memory hierarchies
   - Tuning of PDGEMM for better performance
   - Combination of different algorithms

5 Experimental results

6 Conclusions
Specific Approach

- **hierarchical M-task implementation** with combination of different algorithms to exploit specific characteristics of the execution platform;
- **lower level**: efficient adaptive and/or highly tuned **sequential algorithm** to use single-processor architecture;
- **upper level**: hierarchical (recursive) algorithm to improve communication behavior → **Strassen** algorithm;
- **intermediate level**: hierarchical (recursive) algorithm **tpmm** to improve memory access and communication behavior for multi-processor nodes;
Specific Approach

**result:** multi-level algorithm which allows adaptations to different execution platforms:

- different **combinations** of algorithms;
- different **levels of recursions** of upper level Strassen algorithm;
- different lower level **sequential algorithms**;
- different **block sizes** for low-level algorithms;
- different **schedulings** for upper level Strassen algorithm;
Outline

1. Introduction

2. Specific Approach

3. Execution environment: Tlib

4. Algorithm combinations
   - Recursive splitting – Strassen algorithm
   - tpmm - a new recursive algorithm for memory hierarchies
   - Tuning of PDGEMM for better performance
   - Combination of different algorithms

5. Experimental results

6. Conclusions
Execution environment: Tlib

- The runtime library Tlib allows the realization of hierarchically structured M-tasks programs;
- Tlib is based on MPI and allows the use of arbitrary MPI functions.
- The Tlib API provides support for
  * the creation and administration of a dynamic hierarchy of processor groups;
  * the coordination and mapping of nested M-tasks;
  * the handling and termination of recursive calls and group splittings;
  * the organization of communication between M-tasks;
- The splitting and mapping can be adapted to the execution platform.
Interface of the Tlib library

Type of all **Tlib tasks** (basic and coordination):

```c
void *F (void * arg, MPI_Comm comm, T_Descr *pdescr)
```

- void * arg packed arguments for F;
- comm MPI communicator for internal communication;
- pdescr pointer to a descriptor of executing processor group;

The function F may contain calls of Tlib functions to **create sub-groups** and to **map sub-computations** to sub-groups.
Splitting operations for processor groups

int T_SplitGrp( T_Descr * pdescr,
               T_Descr * pdescr1,
               float per1,
               float per2)

- pdescr: current group descriptor
- pdescr1: new group descriptor for two new subgroups
- per1 relative size of first subgroup
- per2 relative size of second subgroup

int T_SplitGrpParfor( int n,
                      T_Descr * pdescr,
                      T_Descr * pdescr1,
                      float p[])

int T_SplitGrpExpl( T_Descr * pdescr,
                    T_Descr * pdescr1,
                    int color,
                    int key)
Concurrent execution of M-tasks

**Mapping** of two concurrent M-tasks to processor groups:

```c
int T_Par( void * (*f1)(void *, MPI_Comm, T_Descr *),
           void * parg1, void * pres1,
           void * (*f2)(void *, MPI_Comm, T_Descr *),
           void * parg2, void * pres2,
           T_Descr *pdescr)
```

- `f1` function for first M-Task
- `parg1`, `pres1` packed arguments and results for `f1`
- `f2` function for second M-Task
- `parg2`, `pres2` packed arguments and results for `f2`
- `pdescr` pointer to current group descriptor describing two groups
Outline

1. Introduction

2. Specific Approach

3. Execution environment: Tlib

4. Algorithm combinations
   - Recursive splitting – Strassen algorithm
   - tpmm - a new recursive algorithm for memory hierarchies
   - Tuning of PDGEMM for better performance
   - Combination of different algorithms

5. Experimental results

6. Conclusions
A combination of different algorithms for matrix multiplication can lead to very competitive implementations: Strassen method, tpMM, Atlas;

Illustration:
Recursive splitting – Strassen algorithm

- **Matrix multiplication** \( C = AB \) with \( A, B, C \in IR^{n \times n} \):
  - Decompose \( A, B \) into square blocks of size \( n/2 \):
    \[
    \begin{pmatrix}
    C_{11} & C_{12} \\
    C_{21} & C_{22}
    \end{pmatrix} =
    \begin{pmatrix}
    A_{11} & A_{12} \\
    A_{21} & A_{22}
    \end{pmatrix}
    \begin{pmatrix}
    B_{11} & B_{12} \\
    B_{21} & B_{22}
    \end{pmatrix}
    \]

- Compute the **submatrices** \( C_{11}, C_{12}, C_{21}, C_{22} \) separately according to
  \[
  \begin{align*}
  C_{11} &= Q_1 + Q_4 - Q_5 + Q_7 \\
  C_{21} &= Q_2 + Q_4 \\
  C_{12} &= Q_3 + Q_5 \\
  C_{22} &= Q_1 + Q_3 - Q_2 + Q_6
  \end{align*}
  \]

- Compute \( Q_1, \ldots, Q_7 \) by **recursive calls**:
  \[
  \begin{align*}
  Q_1 &= \text{strassen}(A_{11} + A_{22}, B_{11} + B_{22}) \\
  Q_2 &= \text{strassen}(A_{21} + A_{22}, B_{11}) \\
  Q_3 &= \text{strassen}(A_{11}, B_{12} - B_{22}) \\
  Q_4 &= \text{strassen}(A_{22}, B_{21} - B_{11}) \\
  Q_5 &= \text{strassen}(A_{11} + A_{12}, B_{22}) \\
  Q_6 &= \text{strassen}(A_{21} - A_{11}, B_{11} + B_{12}) \\
  Q_7 &= \text{strassen}(A_{12} - A_{22}, B_{21} + B_{22})
  \end{align*}
  \]
Strassen algorithm – task scheduling

- organization into four tasks
  Task\_C11(), Task\_C12(), Task\_C21(), Task\_C22():

<table>
<thead>
<tr>
<th>Task C11</th>
<th>Task C12</th>
<th>Task C21</th>
<th>Task C22</th>
</tr>
</thead>
<tbody>
<tr>
<td>on group 0</td>
<td>on group 1</td>
<td>on group 2</td>
<td>on group 3</td>
</tr>
<tr>
<td>compute $Q_1$</td>
<td>compute $Q_3$</td>
<td>compute $Q_2$</td>
<td>compute $Q_1$</td>
</tr>
<tr>
<td>compute $Q_7$</td>
<td>compute $Q_5$</td>
<td>compute $Q_4$</td>
<td>compute $Q_6$</td>
</tr>
<tr>
<td>receive $Q_5$</td>
<td>send $Q_5$</td>
<td>send $Q_2$</td>
<td>receive $Q_2$</td>
</tr>
<tr>
<td>receive $Q_4$</td>
<td>send $Q_3$</td>
<td>send $Q_4$</td>
<td>receive $Q_3$</td>
</tr>
<tr>
<td>compute $C_{11}$</td>
<td>compute $C_{12}$</td>
<td>compute $C_{21}$</td>
<td>compute $C_{22}$</td>
</tr>
</tbody>
</table>

- realization with Tlib: Task\_C11() and Task\_q1()
void * Strassen (void * arg, MPI_Comm comm,
        T_Dscr *pdescr){
    T_Dscr descr1;
    void * (* f[4])(), *pres[4], *parg[4];
    float per[4];

    per[0]=0.25; per[1]=0.25; per[2]=0.25; per[3]=0.25;
    T_Split_GrpParfor (4, pdescr, &descr1, per);
    f[0] = Task_C11; f[1] = Task_C12;
    T_Parfor (f, parg, pres, &descr1);
    assemble_matrix (pres[0], pres[1], pres[2], pres[3]);
}
void * Task_C11 (void * arg, MPI_Comm comm,
   T_Descr *pdescr){
   double **q1, **q4, **q5, **q7, **res;
   int i,j,n2;

   /* extract arguments from arg including matrix size n */
   n2 = n/2;
   q1 = Task_q1 (arg, comm, pdescr);
   q7 = Task_q7 (arg, comm, pdescr);
   /* allocate res, q4 and q5 as matrices of size n/2 times n/2 */
   /* receive q4 from group 2 and q5 from group 1 using parent comm. */
   for (i=0; i < n2; i++)
      for (j=0; j < n2; j++)
          res[i][j] = q1[i][j]+q4[i][j]-q5[i][j]+q7[i][j];
   return res;
Strassen algorithm – Task_q1()

```c
void *Task_q1(void *arg, MPI_Comm comm, T_Descr *pdescr) {
    double **res, **q11, **q12, **q1;
    struct struct_MM_mul *mm, *arg1;
    *mm = (struct_MM_mul *) arg;
    n2 = (*mm).n / 2;
    /* allocate q1, q11, q12 as matrices of size n2 times n2 */
    for (i=0; i < n2; i++)
        for (j=0; j < n2; j++) {
            q11[i][j] = (*mm).a[i][j]+(*mm).a[i+n2][j+n2]; /* A11+A22 */
            q12[i][j] = (*mm).b[i][j]+(*mm).b[i+n2][j+n2]; /* B11+B22 */
        }
    arg1 = (struct_MM_mul *) malloc (sizeof(struct_MM_mul));
    (*arg1).a = q11; (*arg1).b = q12; (*arg1).n = n2;
    q1 = Strassen (arg1, comm, pdescr);
    return q1;
}
```
tpmm for memory hierarchies

**block structure** of tpmm: \( p = 2^i \) processors

\[
\text{tpMM}(n,p) = \begin{align*}
& \text{for } (l = 1 \text{ to } \log p + 1) \\
& \quad \text{for } k = 1, \ldots, p/2^{l-1} \text{ compute in parallel} \\
& \quad \text{compute } \text{block } (C_{lk});
\end{align*}
\]
tpmm for memory hierarchies

tpmm: data distribution for matrix \( B \) and computation order for 8 processors:

\[
\begin{array}{cccccccc}
\text{P}_0 & \text{P}_1 & \text{P}_2 & \text{P}_3 & \text{P}_4 & \text{P}_5 & \text{P}_6 & \text{P}_7 \\
\text{P}_1 & \text{P}_0 & \text{P}_3 & \text{P}_2 & \text{P}_5 & \text{P}_4 & \text{P}_7 & \text{P}_6 \\
\text{P}_2 & \text{P}_3 & \text{P}_0 & \text{P}_1 & \text{P}_6 & \text{P}_7 & \text{P}_4 & \text{P}_5 \\
\text{P}_3 & \text{P}_2 & \text{P}_1 & \text{P}_0 & \text{P}_5 & \text{P}_6 & \text{P}_7 & \text{P}_4 \\
\text{P}_4 & \text{P}_5 & \text{P}_6 & \text{P}_7 & \text{P}_0 & \text{P}_1 & \text{P}_2 & \text{P}_3 \\
\text{P}_5 & \text{P}_6 & \text{P}_7 & \text{P}_4 & \text{P}_0 & \text{P}_1 & \text{P}_2 & \text{P}_3 \\
\text{P}_6 & \text{P}_7 & \text{P}_5 & \text{P}_4 & \text{P}_1 & \text{P}_2 & \text{P}_3 & \text{P}_0 \\
\text{P}_7 & \text{P}_0 & \text{P}_1 & \text{P}_2 & \text{P}_3 & \text{P}_4 & \text{P}_5 & \text{P}_6 \\
\end{array}
\]

bottom level: Atlas for one-processor matrix multiplication
tpmm performance in isolation

MFLOPS per processor for 4 and 8 processors on IBM Regatta:
Combining Strassen and tpMM

- $p = 4^i 2^j$ processors with $i \geq 1$ and $j \geq 0$:
  4 processor groups are build at each recursion level of Strassen;
  $2^j$ processors are used for the execution of tpmm;
- data layout for 16 processors:
Combining Strassen and PDGEMM

- $p = 4^i \cdot p = 2^d$ processors with $i \geq 1$ and $d \in \{0, 1\}$;
- The processors are mapped row-blockwise onto the blocks of $A$, $B$ and $C$ so that the blocks have size $\frac{n}{r} \times \frac{n}{c}$.
- data layout for 16 processors:
Tuning of PDGEMM

- impact of blocking factor on PDGEMM performance:

Opteron cluster with Infiniband interconnect
Tuning of PDGEMM

- impact of logical block size on PDGEMM performance:

Opteron cluster with Infiniband interconnect

Hunold, Rauber, Rünger

Building Blocks for Multilevel Matrix Multiplication

PMAA’06, Rennes
Outline

1 Introduction

2 Specific Approach

3 Execution environment: Tlib

4 Algorithm combinations
   - Recursive splitting – Strassen algorithm
   - tpmm - a new recursive algorithm for memory hierarchies
   - Tuning of PDGEMM for better performance
   - Combination of different algorithms

5 Experimental results

6 Conclusions
Experimental evaluation: IBM Regatta p690

$p = 32$ and $p = 64$: MFLOPS for different algorithmic combinations:

![Graph showing MFLOPS for different algorithmic combinations with matrix sizes ranging from 2048 to 12288 and MFLOPS values from 0 to 3000. The algorithms compared include pdgemm, strassen_pdgemm_r1, strassen_pdgemm_r2, strassen_ring, and strassen_tpmm for both 32 and 64 processors.](image)
Experimental evaluation: Dual Xeon Cluster 3 GHz

MFLOPS for different algorithmic combinations:

MFLOPS per processor for 16 and 32 processors
Experimental evaluation: Infiniband Cluster

MFLOPS for different algorithmic combinations:

MFLOPS per processor for 32 and 64 processors
Experimental evaluation: SGI Altix

\( p = 16 \): MFLOPS for different algorithmic combinations:

![Graph showing MFLOPS for different matrix dimensions and algorithms]
Outline

1. Introduction
2. Specific Approach
3. Execution environment: Tlib
4. Algorithm combinations
   - Recursive splitting – Strassen algorithm
   - tpmm - a new recursive algorithm for memory hierarchies
   - Tuning of PDGEMM for better performance
   - Combination of different algorithms
5. Experimental results
6. Conclusions
Conclusions

- The combination of different algorithms as **building blocks** leads to competitive parallel implementations;
- There are many different ways to combine the building blocks; for different platforms, **different combinations lead to the best performance**;
- **automatic support** for finding the best combination is useful;
- outlook: the combination of different algorithms might be especially useful for **heterogeneous platforms**;
- further information:
  - ai2.uni-bayreuth.de
  - rauber@uni-bayreuth.de
  - ruenger@cs.tu-chemnitz.de