Optimal Data Distributions for LU Decomposition

THOMAS RAUBER GUDULA RÜNGER *

Computer Science Dep., Universität des Saarlandes, 66041 Saarbrücken, Germany

Abstract. The paper considers the well-known problem of LU decomposition to study a method to derive data distributions for parallel computers with a distributed memory organization. The importance of the paper lies not so much in the special application but with the principle that the problem of finding an optimal data distribution is formulated as an optimization problem. This is possible by using a parameterized data distribution and a rigorous performance prediction technique that allows us to derive runtime formulas containing the parameters of the data distribution. The parameters are determined in such a way that the total runtime is minimized, thus also minimizing the communication overhead and the load imbalance penalty.

1 Introduction

An important issue in the design of parallel programs for distributed memory machines (DMMs) is the choice of a suitable data distribution. The layout of the data structures of a parallel program among the processors of a parallel machine strongly influences the performance of the program. An inappropriate data distribution may lead to a large communication overhead and a load imbalance which may reduce the speedup considerably. The goal is to find a data distribution that minimizes the overall execution time of a parallel program, thus minimizing the communication overhead and showing a good load balance.

In this paper, we describe a technique to derive a suitable data distribution for any algorithm that uses arrays of arbitrary dimension as data structures. The technique is based on the use of a parallel programming model [RRW95] in which formulas for the global execution time of an algorithm can be derived. These formulas depend on the problem size and on the machine parameters that are used by the programming model. The numerical evidence of the formulas derived in this programming model has already been shown for many numerical applications like Newton iteration, extrapolation methods, and different Runge–Kutta methods, see [RRW95] and the references therein.

This paper extends the performance prediction technique of the programming model by introducing parametrized data distributions. The use of these data distributions leads to runtime formulae that not only depend on the problem size and the machine parameters but also on the parameters of the data distribution.

* both authors are supported by DFG

This enables us to apply optimization techniques to determine the parameters of the data distribution such that the global execution time is minimized.

We apply this technique to derive a data distribution for the Gaussian LU decomposition of a matrix. The LU decomposition is chosen as example for two reasons. First, the problem has been extensively studied in the past, see [vdV94] and the references therein. Thus an optimal data distribution for this problem has already been derived by other approaches [FWM94, vdV90] and has been verified by experiments on various parallel machines. This gives us the possibility to verify the results of our analysis. Second, finding an optimal data distribution for LU decomposition is not trivial because both the communication overhead and the load balancing issue have to be taken into consideration. Finally, direct solvers for linear systems are important because they don't impose special requirements on the problem as many iterative solvers do.

The rest of the paper is organized as follows: Section 2 gives an overview of the programming model used. Section 3 introduces a special class of parametrized data distributions which is used in Section 4 for the derivation of formulas for the global execution time of a parallel implementation of the LU decomposition. Section 5 shows how these formulas can be used to determine the parameters of the data distribution such that a minimal global execution time results.

2 Parallel computation model

The important information for the evaluation of a parallel program is the *global execution time*. The global execution time is the time between the start and the termination of a computation of a program and, in case of a parallel program, it consists of the time for computations and the time for communications.

For the theoretical prediction of the execution times, we use the parallel computation model from [RRW95]. The model describes a parallel machine by four parameters: (1) the number p of processors, (2) the time t_{op} to execute an arithmetic operation, (3) the byte transfer time t_c for point-to-point messages, and (4) the startup time τ for point–to–point messages. For a specific machine, the values for t_{op}, t_c, and τ are determined by appropriate benchmark programs.

Parallel programs are specified in an SPMD (*single program multiple data*) programming scheme. The *data exchange* between processors is performed in a synchronous communication phase that is expressed by communication primitives like single–to–single transfer, single accumulation, and single broadcast.

The runtime $t(Prog)$ of a program *Prog* executed by a set G processors is

$$t(Prog) = \max_{q \in G} t_w(q) + \max_{q \in G} t_c(q)$$

where $t_w(q)$ is the computation time for processor q and $t_c(q)$ is the communication time for processor q. The computation times $t_w(q)$ are determined from the arithmetic operations of the program. The communication time depends on the communication primitives used. The costs of one of the communication primitives for a specific parallel machine are expressed by a formula that depends on

the number of communicated data, the number of participating processors, and the machine parameters τ and t_c. The transfer time of a message of M bytes between two processors P_1 and P_2 is $t_{s\_s}(M) = \tau(M) + M \cdot t_c(M)$, independent of the special interconnection network of the DMM. The runtime formulae for the other communication primitives depend on the special machine. For a hypercube network, [JH89] addresses the exact runtimes of the primitives. We use the formulae from [RRW95] for our implementations on an Intel iPSC/860:

$$t_{s\_broad}(p, M) = (1 + \log p)(Mt_c + \tau) \tag{1}$$

$$t_{s\_gather}(p, M) = \frac{p-1}{\log p} Mt_c + \left(\frac{p}{\log p} + \log p\right)\tau$$

3 Data Distributions

In order to determine an optimal data distribution that has the minimal global execution time for the LU–decomposition, we consider a theoretical execution time function that contains the information about different data distributions as parameters. For the distribution of an array A, we adopt the parametrized data distribution of [DHR94]: Let A be an array with d dimensions of size $n_0 \times \ldots \times n_{d-1}$. The elements of A are addressed by elements from an index set $I_A \subseteq N^d$. We assume that the indices of dimension i range between 0 and $n_i - 1$. Let $P = \{q_0, \ldots, q_{p-1}\}$ be the processors of the target machine.

Definition 1. (distribution function) A function $\gamma_A : N^d \to P$ is called a *distribution function* for A. A distribution function γ_A partitions the elements of I_A into p index sets $I_0, \ldots, I_{p-1} \subseteq I_A$ with $I_q = \{k \in I_A | \gamma_A(k) = q\}$.

We consider distribution functions that are described by *distribution* vectors of the form $((m_0, b_0), \ldots, (m_{d-1}, b_{d-1}))$ with $p = \prod_{i=0}^{d-1} m_i$ and $1 \leq b_i \leq n_i$. The value m_i, $0 \leq i \leq d-1$, is the number of processors in dimension i. Dimension i is divided up evenly among the m_i processor groups. For simplicity we assume $n_i/m_i \in N$. The value b_i specifies the block size in dimension i. The two–dimensional case is illustrated in Figure 1.

Definition 2. (Parametrized distribution function) We logically arrange the processors in a d–dimensional grid, i.e. processor q is specified by a grid address $G(q) = (j_0, \ldots, j_{d-1})$ with $0 \leq j_i < m_i$ for $0 \leq i < d$. The distribution function for a distribution vector $((m_0, b_0), \ldots, (m_{d-1}, b_{d-1}))$ is

$$\gamma_A(e_0, \ldots, e_{d-1}) = \left(\left\lfloor \frac{e_0}{b_0} \right\rfloor \bmod m_0, \cdots, \left\lfloor \frac{e_{d-1}}{b_{d-1}} \right\rfloor \bmod m_{d-1}\right) \tag{2}$$

Remark: If $b_i = 1$ for $i = 0, \ldots, d-1$, the function (2) describes a *cyclic distribution*. If $m_i \cdot b_i = n_i$, the function (2) describes a block distribution.

Here, we consider the case $d = 2$ and $n_0 = n_1$, i.e. A is a quadratic matrix, see Figure 1. Each processor owns contiguous blocks of array elements of size $b_0 b_1$. A *superblock* is built by $m_0 m_1$ of these blocks. The number of superblocks in dimension $i = 0, 1$ is $\lceil n/(m_i b_i) \rceil$. For simplicity we assume $n/(m_i b_i) \in N$.

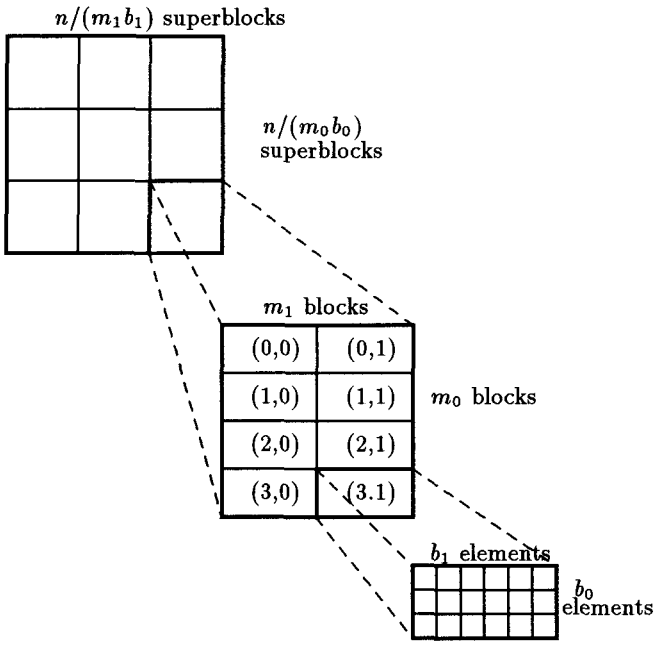

Fig. 1. Distribution of an array with $d = 2$ dimensions among $p = 8$ processors. The figure shows the case $m_0 = 4$, $m_1 = 2$, $b_0 = 3$, and $b_1 = 6$.

Definition 3. (row groups, column groups) For $d = 2$ and a twodimensional grid–numbering of the processors, the set of processors is divided into a partition of m_0 row groups $R_0, \ldots, R_{m_0-1}$ and into a partition of m_1 column groups $C_0, \ldots, C_{m_1-1}$, i.e.

$$\bigcup_{i=0}^{m_0-1} R_i = \bigcup_{i=0}^{m_1-1} C_i = P \quad \text{and} \quad R_i \cap R_j = \emptyset \quad \text{and} \quad C_i \cap C_j = \emptyset \quad \text{for} \quad i \neq j$$

The row and column groups are:

$$R_i = \{q \in P | G(q) = (i, \cdot)\} \qquad C_i = \{q \in P | G(q) = (\cdot, i)\} \tag{3}$$

For $0 \leq i < m_0$ and $0 \leq j < m_1$, it is $|R_i| = m_1$ and $|C_j| = m_0$.

Lemma 4. *For distribution function (2), each row i of A is stored in a single row group R_k with $k \equiv i \pmod{m_0 b_0}$ or $\lfloor i/b_0 \rfloor = \lfloor k/b_0 \rfloor$. Similarly, each column j of A is stored in a single column group C_k with $k \equiv j \pmod{m_1 b_1}$ or $\lfloor j/b_1 \rfloor = \lfloor k/b_1 \rfloor$.*

For a hypercube network, we have $p = 2^x$ and therefore $m_0 = 2^{x_0}$ and $m_1 = 2^{x_1}$ with $x = x_0 + x_1$. We map the two–dimensional processor grid into

a hypercube with *reflected Gray Codes* (RGC) [BT88]. The x–bit RGC is a sequence of 2^x distinct binary numbers with x bits each.

Definition 5. (RGC sequence) The 1–bit RGC sequence is $\{0, 1\}$. Let $\{b_0, b_1, \ldots, b_{q-1}\}$ be the $(x-1)$–bit RGC sequence with $q = 2^{x-1}$. The corresonding x–bit RGC sequence is $\{0b_0, 0b_1, \ldots 0b_{q-1}, 1b_{q-1}, \ldots 1b_0\}$.

We generalize the recursive construction of the RGC sequence: Suppose that $\{a_0, a_1, \ldots a_{m_0-1}\}$ and $\{b_0, b_1, \ldots b_{m_1-1}\}$ are the x_0–bit and x_1–bit RGC sequences. We construct the $m_0 \times m_1$ matrix of x–bit strings $\{a_i b_j | i = 0, \ldots, m_0 - 1; j = 0, \ldots, m_1 - 1\}$

$$\begin{bmatrix} a_0b_0 & a_0b_1 & \cdots & a_0b_{m_1-1} \\ a_1b_0 & a_1b_1 & \cdots & a_1b_{m_1-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_0-1}b_0 & a_{m_0-1}b_1 & \cdots & a_{m_0-1}b_{m_1-1} \end{bmatrix} \tag{4}$$

that represents the mapping of a $m_0 \times m_1$ mesh into a hypercube. The processor p with $G(p) = (i, j)$ is the x–cube node with identity number $a_i b_j$. The rows and columns of matrix (4) represent the row and column groups (3).

Lemma 6. *For a hypercube network, the row (column) groups (3) are hypercubes with m_1 (m_0) processors.*

The fact that row and column groups represent independent substructures makes it possible to assume independent executions on the row and column groups. This is true for every topology in which a grid can be embedded.

4 Parallel Implementation of LU Decomposition

In this section, we describe a general parallel implementation of the LU decomposition that is based on parametrized data distributions as described in the last section and derive formulas for the global execution time that depend on the parameters of the data distribution and the machine model.

4.1 LU Decomposition of a matrix

We consider the LU decomposition of a matrix A without pivoting in order to keep the example comprehensible. The following short description of the LU decomposition introduces the notation that we use for the parallel implementation. The matrix A is factorized into $A = LU$ with a unit diagonal, lower triangular matrix L and an upper triangular matrix U. To compute the n^2 unknown entries of L and U, we proceed row by row (Gauß). The entries of L and U are stored in a single matrix A^{n-1}. We compute A^{n-1} by executing $(n-1)$ elimination steps. Elimination step m consists of subtracting suitable multiples of the mth equation from the remaining equations $m+1, \ldots, n$. The multiples are determined in

such a way that the unknown x_m is eliminated from these equations. In general, after the $(m-1)$th elimination step we are left with matrix

$$A^{(m-1)} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,m-1} & a_{1m} & \cdots & a_{1n} \\ l_{21} & a_{22}^{(1)} & \cdots & a_{2,m-1}^{(1)} & a_{2m}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \ddots & \ddots & \vdots & \vdots & & \vdots \\ \vdots & & \ddots & a_{m-1,m-1}^{(m-2)} & a_{m-1,m}^{(m-2)} & \cdots & a_{m-1,n}^{(m-2)} \\ \vdots & & & l_{m,m-1} & a_{mm}^{(m-1)} & \cdots & a_{mn}^{(m-1)} \\ \vdots & & & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & \cdots & \cdots & l_{n,m-1} & a_{nm}^{(m-1)} & \cdots & a_{nn}^{(m-1)} \end{bmatrix}$$

In step m, x_m is eliminated from equations $i = m+1, \ldots, n$ by subtraction of $a_{im}^{(m-1)}/a_{mm}^{(m-1)}$ times equation m from each of these equations. This yields the matrix in which the first m rows and $m-1$ columns are the same as those of $A^{(m-1)}$. The remaining elements in column m of $A^{(m)}$ are replaced by $l_{im} = a_{im}^{(m-1)}/a_{mm}^{(m-1)}$ and the other elements are given by $a_{ij}^{(m)} = a_{ij}^{(m-1)} - a_{im}^{(m-1)} a_{mj}^{(m-1)}/a_{mm}^{(m-1)}$ for $i, j = m+1, m+2, \ldots, n$.

4.2 Parallel Implementation

For notational convenience, we use $Ro(k)$ and $Co(k)$ to denote the row group that holds the row with number k and the column group that holds the column with number k. Using distribution function (2), $Ro(k)$ and $Co(k)$ are defined by:

$$Ro(k) = R_i \quad \text{for} \quad k \equiv i \pmod{m_0 b_0}$$
$$Co(k) = C_i \quad \text{for} \quad k \equiv i \pmod{m_0 b_0}$$

We consider the forward elimination step $m \rightarrow m+1$ without pivoting. For the array element with index (m, m) there is exactly one processor $r \in P$ with $(m, m) \in I_r$. The following operations are necessary for the elimination step:

1. Broadcast of the pivot row: In order to eliminate the mth column element $a_{im}^{(m)}$ for $i > m$, a suitable multiple of the pivot row $(a_{mm}^{(m-1)}, \ldots, a_{mn}^{(m-1)})$ has to be subtracted from the remaining rows $m+1, \ldots, n$. Therefore, the pivot row has to be sent to the relevant processors. The pivot row is distributed among the processors $q \in Ro(m)$. The element $a_{mj}^{(m-1)}$ of column j has to be sent only to those processors owing some elements of the same column j, i.e. to all $q \in Co(j)$. Thus each processor $q \in Ro(m)$ performs a single broadcast operation to all processors in its column group (group broadcast). The number of elements to be transmitted is $\#\{(m, j) \in I_q | j \geq m\}$.
2. Computation of elimination factors: The processors in the column group of r compute the values l_{im}, i.e. each $q \in Co(m)$ computes l_{im} if $(i, m) \in I_q$. Processor $q \in Co(m)$ computes $\#\{(i, m) \in I_q | i > m\}$ elements.

3. Broadcast of elimination factors: Processor $q \in Co(m)$ broadcasts all computed values l_{im} to all processors owing some elements of row i, i.e. to all $s \in Ro(i)$. The number of elements transmitted is $\#\{(i,m) \in I_q | i > m\}$.
4. Computation of new matrix elements: Each processor q computes $a_{ij}^{(m)} = a_{ij}^{(m-1)} - l_{im} a_{mj}^{(m-1)}$ for all elements $(i,j) \in I_q$. That are $|\{(i,m) \in I_q; i > m\}| \cdot |\{(m,j) \in I_q; j > m\}|$ elements.

The LU decomposition executes $n-1$ of these steps. There are synchronization points between the broadcast of the pivot row and the computation of the elimination factors l_{im} and between the broadcast of these factors and the computation of the new matrix elements. But there is no need for a synchronization between the computation and the transmission of the values l_{im}. In the two computation phases (2. and 4.) all processors involved in the computation are executing in parallel.

4.3 Runtime Prediction

We assume that the processors are arranged in a topology in which the disjoint row or column groups represent independent substructures. Examples are a hypercube network in which the substructures are also hypercubes and d–dimensional grid structures in which the substructures are $(d-1)$–dimensional grids. For these topologies, the broadcast operations of the row or column groups can be performed in parallel. Therefore, the maximal communication time needed by the dominating group determines the global communication time. The following lemma summarizes the resulting global execution time of the complete method according to the performance model from Section 2. The global execution time specified represents an upper bound to the exact runtime.

Lemma 7. *The global execution time of the LU–decomposition estimated according to the computational model of Section 2 is:*

$$\begin{aligned} T_1 = \sum_{m=1}^{n-1} \Big\{ & \max_{q \in Ro(m)} t_{s\_broad}(\#Co(q), \#\{(m,j) \in I_q | j \geq m\}) \\ & + \max_{q \in Co(m)} \#\{(i,m) \in I_q | i > m\} \cdot t_{op} \\ & + \max_{q \in Co(m)} t_{s\_broad}(\#Ro(m), \#\{(i,m) \in I_q | i > m\}) \\ & + \max_{q} (\#\{(i,m) \in I_q | i > m\} \cdot \#\{(m,j) \in I_q | j > m\}) * 2t_{op} \Big\} \end{aligned}$$

For the parametrized distribution function (2) the number of elements to be computed or broadcasted are estimated by upper bounds containing the parameters m_0, b_0, m_1, b_1. For the broadcast of the pivot row m, the number of elements to be transmitted by each processor $q \in Ro(m)$ is

$$\#\{(m,j) \in I_q | j \geq m\} \leq \left\lceil \frac{n-m+1}{m_1 b_1} \right\rceil b_1 \leq \left(\frac{n-m+1}{m_1 b_1} + 1 \right) b_1 = \frac{n-m+1}{m_1} + b_1$$

where $\left\lceil \frac{n-m+1}{m_1 b_1} \right\rceil$ is the number of superblocks and b_1 is the number of elements of row m in each superblock. The number of elements l_{im} to be computed or broadcasted by each processor $q \in Co(m)$ is

$$\#\{(i,m) \in I_q | i > m\} \leq \left\lceil \frac{n-m}{m_0 b_0} \right\rceil b_0 \leq \left(\frac{n-m}{m_0 b_0} + 1 \right) b_0 = \frac{n-m}{m_0} + b_0$$

The parametrized data distribution and the corresponding estimations of the size of the index subsets result in the global execution time that is expressed in the following lemma. Note that the estimation of the number of elements broadcasted or computed is independent of the individual processor. Therefore there is no need to use the maximum function.

Lemma 8. *The parameterized global execution time of the LU-decomposition implemented with the parametrized data distribution* $((m_0,b_0),(m_1,b_1))$ *is*

$$\begin{aligned} T_2(m_0,b_0,m_1,b_1) = \sum_{m=1}^{n-1} \Big(& t_{s\_broad}(m_0, \frac{n-m+1}{m_1} + b_1) \\ & + (\frac{n-m}{m_0} + b_0) t_{op} + t_{s\_broad}(m_1, \frac{n-m}{m_0} + b_0) \\ & + (\frac{n-m}{m_0} + b_0)(\frac{n-m}{m_1} + b_1) 2t_{op} \Big) \end{aligned}$$

where $m_0 m_1 = p$ *and* $1 \leq b_i \leq n/m_i$ *for* $i = 0, 1$.

For a hypercube network, we substitute the time t_{s_broad} for a single broadcast operation by equation (1) and transform the addition over m according to $\sum_{m=1}^{n-1} m = n(n-1)/2$ and $\sum_{m=1}^{n-1} m^2 = n(n-1)(2n-1)/6$. We get for example:

$$\begin{aligned} & \sum_{m=1}^{n-1} t_{s\_broad}(m_0, \frac{n-m+1}{m_1} + b_1) \\ & = \sum_{m=1}^{n-1} (1 + \log m_0) \left(\left(\frac{n-m+1}{m_1} + b_1 \right) t_c + \tau \right) \\ & = (1 + \log m_0) \left(\left(\frac{n(n-1)}{2} \frac{1}{m_1} + \frac{n-1}{m_1} + (n-1) b_1 \right) t_c + (n-1)\tau \right) \end{aligned}$$

Lemma 9. *The global execution time of the LU-decomposition implemented with the parametrized data distribution* $((m_0,b_0),(m_1,b_1))$ *on a hypercube network is*

$$\begin{aligned} T_3(m_0,b_0,m_1,b_1) = & (1 + \log m_0) \left(\left(\frac{n(n-1)}{2} \frac{1}{m_1} + \frac{n-1}{m_1} + (n-1) b_1 \right) t_c + (n-1)\tau \right) \\ & + \left(\frac{n(n-1)}{2} \frac{1}{m_0} + (n-1) b_0 \right) t_{op} \\ & + (1 + \log m_1) \left(\left(\frac{n(n-1)}{2} \frac{1}{m_0} + (n-1) b_0 \right) t_c + (n-1)\tau \right) \\ & + \left(\frac{n(n-1)(2n-1)}{6} \frac{1}{p} + \frac{n(n-1)}{2} \left(\frac{b_1}{m_0} + \frac{b_0}{m_1} \right) + (n-1) b_0 b_1 \right) 2t_{op} \end{aligned}$$

where $m_0 m_1 = p$ *and* $1 \leq b_i \leq n/m_i$ *for* $i = 0, 1$.

Special cases of the parametrized distributions are the

- row–cyclic distribution with $((m_0, b_0), (m_1, b_1)) = ((p, 1), (1, 1))$
- column–cyclic distribution with $((m_0, b_0), (m_1, b_1)) = ((1, 1), (p, 1))$

The decision which of these distributions is better depends on the number of processors p and the system size n. We consider the difference

$$T_3((p,1),(1,1)) - T_3((1,1),(p,1)) = (n\text{–}1)\left((1+log(p))t_c + \frac{n}{2p}t_{op} - \frac{1}{p}t_c - \frac{n}{2}t_{op}\right)$$

Lemma 10. *The row–cyclic distribution is better than the column–cyclic distribution (i.e.* $T_3((p,1),(1,1)) - T_3((1,1),(p,1)) > 0$*) if*

$$n < \frac{t_c}{t_{op}} \frac{2(p(1+log(p)) - 1)}{p-1}$$

In some situations, the row– and column–cyclic distributions are suitable for the use of LU decomposition as a module within more complicated problems. An example for this situation can be found in [RR95] which describes the implementation of a diagonal–implicitly iterated Runge–Kutta (DIIRK) method. The DIIRK method executes several Newton iterations where each iteration of the Newton method uses an LU decomposition.

5 Optimal Data Distribution for LU Decomposition

In this section, we derive an *optimal* parametrized data distribution for the LU decomposition, i.e. the data distribution that minimizes the global execution time.

Definition 11. The optimal data distribution for a parallel program *Prog* is the data distribution $((\bar{m}_0, \bar{b}_0), \ldots, (\bar{m}_{d-1}, \bar{b}_{d-1}))$ that minimizes the parametrized global execution time $T_{Prog}((m_0, b_0), \ldots, (m_{d-1}, b_{d-1}))$ of *Prog*, i.e.

$$T_{Prog}((\bar{m}_0, \bar{b}_0), \ldots, (\bar{m}_{d-1}, \bar{b}_{d-1})) = \min_{\substack{(m_i, b_i) \\ i=0,\ldots,d-1}} T_{Prog}((m_0, b_0), \ldots, (m_{d-1}, b_{d-1}))$$

where $\prod_{i=0}^{d-1} m_i = p$ and $1 \leq b_i \leq n_i/m_i$.

The following lemma states that the block size of an optimal data distribution is 1 in each dimension, if the runtime formula for the execution time of a broadcast operation can be separated into a part that only depends on the number of processors and into a part that only depends on the number of data items transmitted.

Lemma 12. *(Block size of optimal data distribution for LU decomposition) Consider a network in which the time for a single broadcast operation can be represented as* $t_{s\_broad}(p, M) = f(p)g(M)$ *with* g *monotonically increasing for* $M > 0$. *The optimal data distribution minimizing* $T_2((m_0, b_0), (m_1, b_1))$ *of Lemma 8 fulfills* $b_0 = b_1 = 1$.

Proof. We consider T_2 as a function of b_0 and b_1. The broadcast terms of T_2 are monotonically increasing in b_0 or b_1. The computation term is a positiv quadratic function in b_0 and b_1, i.e. T_2 contains positive terms in b_0, b_1, and $b_0 b_1$. Therefore, T_2 has its minimum at the left hand boundary of the intervall $1 \leq b_i \leq n/m_i$, $i = 0, 1$, which is independent of m_0 and m_1.

Note that the separation of the variables required by the lemma is possible for most networks like trees, meshes, and hypercubes. For a hypercube network, it is for example according to equation (1): $f(p) = 1 + \log p$ and $g(M) = \tau + Mt_c$. Because of Lemma 12 we now have to minimize $T_2((m_0, 1), (m_1, 1))$.

For this minimization, we have to know the dependence of the runtime of the broadcast operation on the number of participating processors i.e. we need a concrete runtime formula. From now on we consider the hypercube network as a specific network and minimize $T_3((m_0, 1), (m_1, 1))$ which we denote by $T_3(m_0, m_1)$. The function T_3 consists of three components: a constant part T_C that is independent of m_0 and m_1, a symmetric part T_S that is a symmetric function in m_0 and m_1, and an asymmetric part T_A.

$$\tilde{T}_3(m_0, m_1) = T_C(m_0, m_1) + T_S(m_0, m_1) + T_A(m_0, m_1)$$

$$T_C(m_0, m_1) = 2(n-1)(t_c + \tau) + 3(n-1)t_{op} + (n-1)(t_c + \tau)(\log m_0 + \log m_1) + \frac{n(n-1)(2n-1)}{6}\frac{1}{p}2t_{op} \quad (5)$$

$$T_S(m_0, m_1) = \frac{n(n-1)}{2}(t_c + 2t_{op})\left(\frac{1}{m_0} + \frac{1}{m_1}\right) + \frac{n(n-1)}{2}t_c\left(\frac{\log m_0}{m_1} + \frac{\log m_1}{m_0}\right) \quad (6)$$

$$T_A(m_0, m_1) = (n-1)t_c\left(\frac{1}{m_1} + \frac{1}{m_1}\log m_0\right) + \frac{n(n-1)}{2}\frac{1}{m_0}t_{op} \quad (7)$$

The constant component T_C is independent of m_0 and m_1 (because of $\log m_0 + \log m_1 = \log(m_0 m_1) = \log p$) and does not influence the optimal choice for m_0 and m_1. The symmetric part T_S contains the row broadcast and part of the column broadcast of the pivot row. The asymmetric part T_A contains the communication overhead of broadcasting the pivot element within the column groups and a term that reflects the load imbalance of computing the elimination factors l_{im} only in one column group. For T_S the minimum can be computed analytically, if the arguments are assumed to be real values.

Lemma 13. *(Optimal data distribution for the symmetric part) If T_S is considered as a function of real values, then $T_S(m_0, m_1)$ is minimal for $m_0 = m_1 = \sqrt{p}$.*

Proof. The analytical solution is obtained by differentiating $T_S(m_0, p/m_0)$:

$$\frac{d}{dm_0}T_S(m_0, p/m_0) = \frac{n(n-1)}{2}(t_c + 2t_{op})\left(\frac{1}{p} - \frac{1}{m_0^2}\right) + \frac{n(n-1)}{2}t_c\left(\left(\frac{\log m_0}{p} - \frac{1}{m_0^2}\log\frac{p}{m_0}\right) + \left(\frac{1}{p\ln 2}\left(1 - \frac{p}{m_0^2}\right)\right)\right)$$

It is $\frac{d}{dm_0}T(\sqrt{p}, \sqrt{p}) = 0$ because of $1 - p/m_0^2 = 0$ and $\frac{1}{p}\log m_0 - \frac{1}{m_0^2}(\log p - \log m_0) = 0$ for $m_0 = \sqrt{p}$.

For an integer solution we have to find integer values m_0 and m_1 near $\sqrt{p}$ that fulfill $m_0 m_1 = p$.

The data distribution characterized by $m_0 = m_1 = \sqrt{p}$ is the optimal solution for the symmetric part T_S and also for $T_S + T_C$. The asymmetric part T_A has not been taken into account for the derivation of the solution. This is only justified if the influence of T_A is small. Figure 2 shows for two fixed number of processors that the influence of T_A on the optimal selection of m_0 and m_1 is small. The following lemma shows that T_A is small compared to the global execution time $T_S + T_C + T_A$ for an arbitrary number of processors.

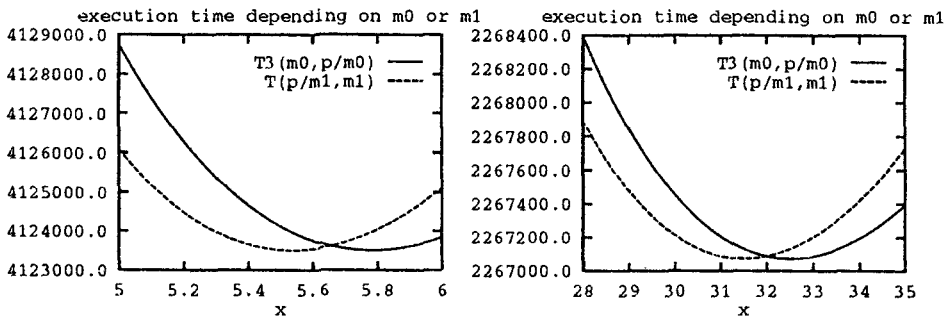

Fig. 2. Global execution time $\tilde{T}_3(m_0, m_1)$ in μs with variable m_0 and setting $m_1 = p/m_0$, and variable m_0 and setting $m_1 = p/m_0$ for $p = 32$ and $p = 1024$ processors. The intersection point of the curves lying at $\sqrt{p}$ is the optimal solution of the symmetric part.

Lemma 14. *Let T_C, T_S, and T_A be the constant, the symmetric, and the asymmetric parts of the global execution time $T_C + T_S + T_A$ according to (5), (6), and (7). The fraction of the asymmetric part T_A on the global execution time is*

$$\frac{T_A}{T_C + T_S + T_A} \leq \frac{1}{X} + \frac{1}{Y}$$

with $X = X(n,p)$ and $Y = Y(n,p)$:

$$X = \frac{n}{2} + \frac{3n}{2p(1+\log p)} + \frac{n}{\log p}\frac{t_{op}}{t_c} + \frac{(2+\log p)(t_c+\tau) + 3t_{op}}{(1+\log p)t_c} + \frac{2n(2n-1)t_{op}}{6p(1+\log p)t_c}$$

$$Y = 11 + 2n + (n+1)\frac{t_c}{t_{op}} + \frac{2}{3p}(2n+1) + \frac{2}{n}(1+\log p)\frac{t_c+\tau}{t_{op}}$$

Proof. The claim follows by substituting T_C, T_S, and T_A and estimating m_0 and m_1 appropriately.

Note that X and Y do not depend on the special choice of m_0 and m_1 but only on n and p. Figure 3 shows that the value of $1/X + 1/Y$ is smaller than 0.3% for all values of n and p.

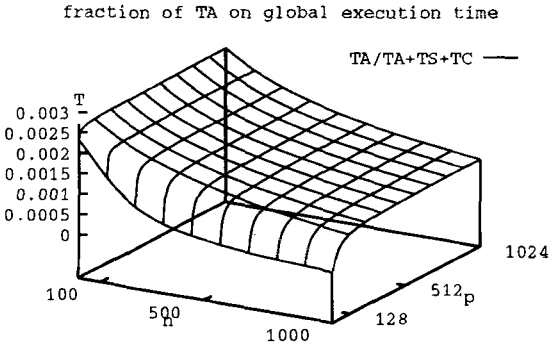

Fig. 3. Value of $1/X(n,p)+1/Y(n,p)$ for different system sizes and number of processors.

6 Conclusions

This article shows that the problem of finding a good data distribution for numerical algorithms can be reduced to an optimization problem, if parametrized data distribution functions are used and if the algorithms are formulated within an appropriate computation model. This is illustrated by considering the LU decomposition of a matrix as a representative example. The technique can also be applied to other numerical algorithms that work with arrays of arbitrary dimension. We think that the technique could be a starting point to solve the data distribution problem in parallelizing compilers [DHR94].

References

[BT88] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computing.* Prentice Hall, New York, NY, 1988.

[DHR94] A. Dierstein, R. Hayer, and T. Rauber. A Branch–and–Bound Algorithm for Array Distributions. In *2nd Euromicro Workshop on Parallel and Distributed Processing*, pages 528–535, 1994.

[FWM94] G.C. Fox, R.D. Williams, and P.C. Messias. *Parallel Computing Works!* Morgan Kaufmann Publishers, 1994.

[JH89] S.L. Johnsson and C.T. Ho. Optimum Broadcasting and Personalized Communication in Hypercubes. *IEEE Transactions on Computers*, 38(9):1249–1268, 1989.

[RR95] T. Rauber and G. Rünger. Performance Predictions for Parallel Diagonal-Implicitly Iterated Runge-Kutta Methods. In *9th Workshop on Parallel and Distributed Simulation*, Lake Placid, USA, 1995.

[RRW95] T. Rauber, G. Rünger, and R. Wilhelm. An Application Specific Parallel Programming Paradigm. In *International Conference on High-Performance Computing and Networking*, 1995.

[vdV90] E.F. van de Velde. Data redistribution and concurrency. *Parallel Computing*, 16:125–138, 1990.

[vdV94] E.F. van de Velde. *Concurrent Scientific Computing.* Springer, 1994.