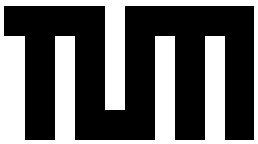


TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

Design, Implementation and Evaluation of Data Rivers for Efficient Intra-Query Parallelism

Clara Nippl, Stephan Zimmermann, Bernhard Mitschang

TUM-I9918
SFB-Bericht Nr. 342/08/99 A
November 99

TUM–INFO–11-I9918-150/1.–FI

Alle Rechte vorbehalten
Nachdruck auch auszugsweise verboten

©1999 SFB 342 Methoden und Werkzeuge für
die Nutzung paralleler Architekturen

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Design, Implementation and Evaluation of Data Rivers for Efficient Intra-Query Parallelism

Clara Nippl[1], Stephan Zimmermann[1], Bernhard Mitschang[2]

[1]Computer Science Department, Technische Universität München
D - 80290 Munich, Germany
e-mail: nippl@in.tum.de

[2]Institut für Parallele und Verteilte Höchstleistungsrechner, Universität Stuttgart
D - 70565 Stuttgart, Germany

Abstract *Upcoming applications such as OLAP, DSS and object-relational DBMS stress the demand for high performance and thus implicitly for efficient intra-query parallelism. In this paper, we evaluate the data river paradigm that has been designed for the management of intermediate query result sets that are produced as well as consumed by operators in a parallel database engine. We point out some aspects related to this paradigm that have a serious impact on query processing and efficiency. In addition we present an implementation based on a stringent modularization concept in combination with a set of parameters that on one hand provide necessary flexibility and on the other hand contribute to significant performance improvements. Furthermore, based on a thorough performance analysis we come up with a comprehensive set of parameter combinations that are recommended for specific situations covering the necessary spectrum of communication patterns typically found in parallel database engines.*

Keywords: Parallel Databases, Query Processing and Optimization, Resource Management

1. Introduction

The MIDAS project concentrates on optimization, parallelization and execution of queries coming from such areas as OLAP, decision support systems (DSS) and digital libraries [CJ+97]. We use designated operators to support special query types [NJM97] moving into the direction to support user-defined functions and object-relational queries [JM99, JM98]. In order to meet the demands w.r.t. performance in these scenarios, efficient intra-query parallelism is indispensable.

The *data river* paradigm [Gr95, DeG92] has been devised to achieve inter- and intra-operator parallelism between producing and consuming instances by means of special communication operators. Meanwhile this concept is used in many (object-)relational PDBMSs under various synonyms, e.g. the *send/receive* operators in DB2 UDB [JMP97, Ibm98], the *split/merge* operators in Gamma [DeG92], or the *Exchange* operator in Informix Dynamic Server and Volcano [Grae94, Zo97, Inf98]. However, in the course of validating our implementation of this concept, we have identified a set of parameters that significantly impact the performance of parallel queries, in terms of speedup and resource consumption. These parameters concern dataflow control, the granularity for data transport, as well as specific measures to minimize overhead and thus increase efficiency. Based on this insight we developed a clean modularization of the data river paradigm that provides this set of parameters and that resulted into an extensible approach covering the whole spectrum for communication found and needed in parallel query processing. To our knowledge, there exists no other publicly available report dealing with these aspects and its consequences at this level of parallelism and detail.

The rest of the paper is organized as follows. First, we give an introduction to our testbed parallel database system and provide the basic concepts and terminology concerned with intra-query parallelism, thereby developing the primitives of the data river paradigm. Section 3 details our design and

implementation of data rivers, coming up with a modularization and parameterization approach. As a first summary, a set of parameter combinations is investigated that is recommended for specific situations covering the necessary spectrum for communication. Efficient embedding of the data river paradigm into query processing is discussed in Section 4. Finally, Section 5 covers related work, while a conclusion is given in Section 6.

2. Issues in Intra-Query Parallelism

In this section, we present the terminology and concepts used for parallel query execution in MIDAS.

2.1 The Testbed Parallel Database System MIDAS

MIDAS (**M**un**I**ch Parallel **DA**tabase **S**ystem) is a prototype of a parallel database system running on a hybrid architecture, comprising SMP nodes combined in a shared-disk manner [BJ96]. For portability reasons we embedded the whole system into the Parallel Virtual Machine environment (PVM [Ge94]). One major goal of our project is the design of an adaptive cost-based parallel query execution integrating scheduling and load balancing. In order to reach this goal, we had to solve two problems: first, at compile-time, the generation of a parametric parallel query execution plan (parallel QEP or PQEP) and second, at run-time, the adaptive, parallel execution of this plan. The first task is performed by the parallelizer TOPAZ [NM98a] that is implemented based on the Cascades Query Optimizer Framework[1] representing a cost-based and rule-driven approach. The parallelizer's cost model comprises besides CPU-costs also communication costs, memory usage, and disk accesses and it supports inter-operator as well as intra-operator parallelism by means of pipeline and data parallelism. The goal is to identify a parameterized PQEP, whose parameter ranges determine lower and upper bounds on the reasonable degree of parallelism (DOP) and corresponding resource consumption for the execution of the PQEP. This parameterized PQEP allows the run-time component, called Query Execution Control (QEC), to derive individual PQEPs with fixed parameter values that are adjusted to the run-time system state. Thus, QEC comprises the run-time responsibilities of load balancing, scheduling, and resource allocation. It performs a fine-tuning of the PQEP and schedules different portions of the PQEP to different *execution units*.

2.2 Basic Concepts and Notation

The MIDAS operators are self-contained software objects designed according to the *iterator* (or *Open-*

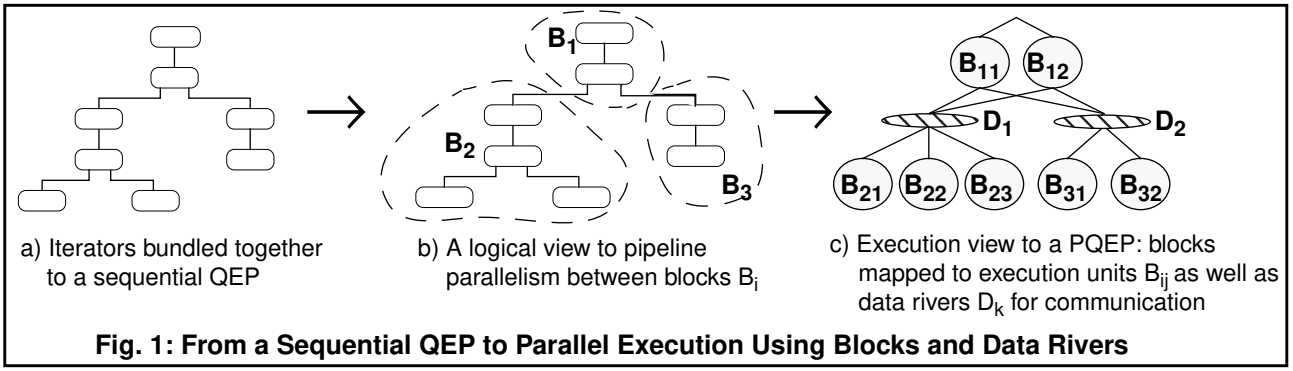


Fig. 1: From a Sequential QEP to Parallel Execution Using Blocks and Data Rivers

1. The Cascades Optimizer Framework [Grae95] is used in Microsoft's SQL Server and Tandem's NonStop SQL as well. Due to space limitations, we will detail the strategies used by the parallelizer only as far as it is relevant to this paper.

Next-Close) processing model [Grae94]. In this model that is used also in object-relational DBMSs [GB+96], queries are structured by bundling together the appropriate operators (iterators) into a QEP (Fig. 1a). In a sequential DBMS, each QEP is processed by a single execution unit. In the course of parallelization, this QEP is broken down into several subplans or *blocks* [TD93] (Fig. 1b) that define the granularity or unit of parallelism [HS93]. In our earlier work [NM98a] we have shown that it is vital to perform a cost-based analysis to identify the optimal granularity for parallelism, i.e. number of blocks for a given query, number of operators within a block, and degree of parallelism assigned to a block. This is achieved in MIDAS by the parallelizer TOPAZ. Some characteristics of PQEPs resulting from this approach are the following:

- cost-related degrees of parallelism and adjusted block sizes saving scarce resources
- parameters allowing a fine-tuning of the execution plan to different application scenarios
- usage of all possible communication patterns to realize efficient intra-query parallelism.

2.3 Anatomy of the Data River Concept

As determined by the parallelizer and at run-time adjusted by the QEC, a block is assigned to one or several *execution units*, according to its degree of parallelism. The flow of tuples among the various execution units is organized by the concept of *data rivers* (D_1 and D_2 in Fig. 1c). This mapping supports data parallelism within a block as well as pipeline parallelism in between blocks. The data river concept [Gr95, DeG92], based on split and merge of intermediate result tuple sets, is adopted also by many (object-)relational DBMS as it complies best with the iterator concept for the operators. If multiple producers add records to the same river that is consumed by several consumers, the river consists of several parallel *data streams*. In this way, parallelism is transparent for operators or operator instances, as they still operate sequentially on these data streams that constitute the data river.

In Fig. 2 we detail the data river concept (data streams are visualized as black bars) to distinguish the following basic communication patterns that comprehensively support intra-query parallelism:

- **Pipelining:** This is the most simple way of intra-query communication. In Fig. 2a left, the output of producer B_2 is consumed by consumer B_1. In this case, the data river consists of a single data stream. In the more general case, when consumer and producer have the same degree of parallelism N, the corresponding data river consists of N distinct streams (Fig. 2a right).
- **Replication:** In this case, all consumer instances read the same input. In Fig. 2b left, a single block produces a data stream, that is read by all consumer instances. In the more general case (Fig. 2b right), N instances produce N data streams that are read by all consumer instances. Thus, for replication N data streams are necessary, N being the degree of parallelism of the producer block.

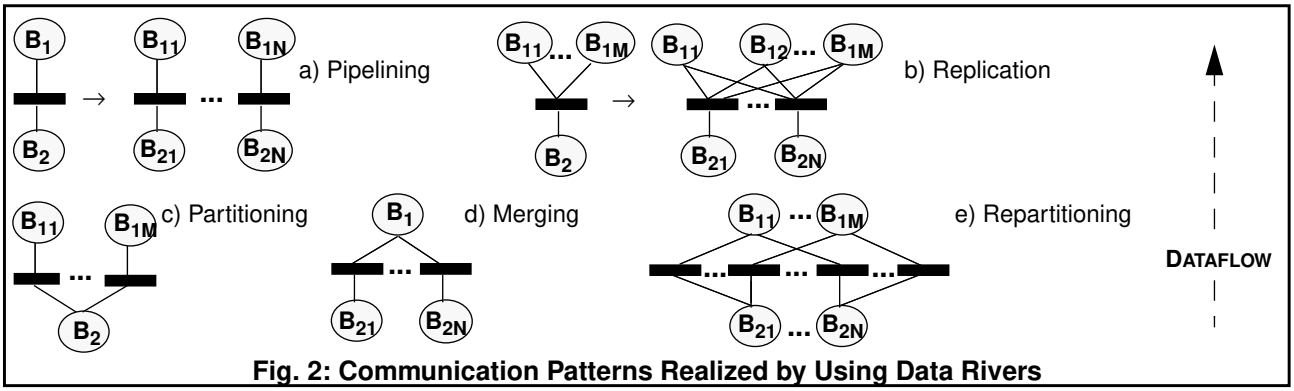


Fig. 2: Communication Patterns Realized by Using Data Rivers

- **Partitioning:** In Fig. 2c, the tuples produced by B_2 are partitioned according to a given criteria and written into the corresponding data stream that is read by an instance of consumer B_1. In this case M data streams are needed, where M is the degree of parallelism of the consumer block.
- **Merging:** In order to produce a final result, each producer instance fills a separate data stream with its intermediate results (see Fig. 2d). These are read by the consumer which merges the tuples coming from the different data streams according to the requirements for further processing (like sort ordering etc.). Please note that another possibility would be to have a single data stream, that is jointly filled by all producer instances. The disadvantage of this communication pattern is that the merging has to be done on the producer side. Hence, any requirements for data stream properties at the consumer side have to be taken into consideration already at the producer side. Moreover, the producer instances cannot operate independently from one another. Thus the independency issue, i.e. being self-contained processing objects, would not be satisfied for operators and operator instances.
- **Repartitioning:** Different degrees of parallelism or different partitioning strategies on both producer and consumer side imply a repartitioning of the data as shown in Fig. 2e. For a *N:M* redistribution, as each of the N producers write into M partitions, each mapped to a data stream, N x M data streams are necessary to build this type of data river.

3. Implementation Concepts for Data Rivers

In the following we describe the design and implementation of the data river paradigm as developed in the PDBMS MIDAS. Please note that in the logical concept presented in Section 2, there is no dependency to any specific execution architecture. In this section, we concentrate on an implementation concept for data rivers that preserves this independency to a large extent. Thereby, we identify the following basic aspects of the data river approach: data flow, data partitioning and data merging. Each aspect is implemented by a separate module and a set of parameters to offer maximum adaptability to specific situations. Then extensibility measures to the data rivers concept are also valuable for forthcoming hybrid heterogeneous architectures [NZT96, HFV96].

In the following we concentrate on this modularization approach and show the impact of the defined parameters on performance. Thereby, a thorough performance analysis is conducted and a comprehensive set of parameter combinations is identified that are recommended for specific situations.

3.1 Communication Segments as a Concept to Implement Data Rivers

In order to organize the flow of intermediate and result tuples, the parallelizer equips each block instance with two communication operators: *send* and *receive*. These follow the same iterator concept as all the other operators thus hiding all particularities and implementation aspects of the data river concept. Each block instance, excluding the top one, has as root a *send* operator, which transmits the resulting tuples to one or more parent blocks. Many DBMSs use a special communication subsystem for the implementation of the data river between the *send* and *receive* operators [BFG95, Zo97]. In contrast, we decided to view a data stream as a special temporary relation mapped to so-called *communication segments (CS)* that are handled by the storage system and by the database buffer management. CSs are temporary segments that are only visible inside a transaction and that are deleted at the end of the query or transaction. The same kind of segments are used for instance by the *sort* operator to store initial runs. Thus, *send* and *receive* are implemented to write (respectively to read

from) CSs. With this concept of mapping each data stream to a CS, MIDAS uses the same model for permanent and intermediate data sets. Additionally, the communication between consumers and producers takes place in an efficient buffered manner.

MIDAS supports different data organizations within a CS. In addition to the very general notion of a tuple stream floating from producer to consumer execution units, one can use appropriate data structures to improve the execution of both producer and consumer processing. For example, sorting (the task of the producer) as well as sorted access (a consumer requirement) can be considerably simplified by a B-tree data organization of the CS representing the data stream; in addition, such a data structure allows for direct and repeated access. Thus, by storing intermediate results in 1 or several CSs, MIDAS allows the reuse of these results, a prerequisite to optimization for common subexpressions as well as to multi-query optimization [Qi96, Se88].

With the concept of data streams mapped to CSs, we could realize all communication patterns necessary for intra-query parallelism as described in Section 2.3. However, in order to make a certain communication pattern executable several settings for data partitioning, data merge, or data flow have to be provided. This will be discussed in the following subsections and exemplified by the QEP examples given in Fig. 3. Please note that in this representation, the operators bracketed by *send* and *receive (recv)* nodes are bundled together to a block. Blocks can have different degrees of parallelism depicted as a sequence of overlapping operators. For simplification purposes, we omit the data rivers and the associated data streams at the borders of a block, but we express the bundling of operators within a block by connection lines. Most operators (e.g. the *sort*), show certain parameters that describe the operator execution in more detail, as e.g. memory allocation and management, sort parametrization etc. These parameters can also be adapted to the system state at run-time by QEC.

3.2 Control of Dataflow

In MIDAS, the granule of dataflow between execution units that communicate via CSs is a *page* or alternatively a *subpage*, i.e. a fraction of a page, since MIDAS uses subpages for coherency control, locking, as well as logging/recovery to reduce overhead [Li94]. Flow control is achieved by a page-based locking scheme. For instance, consuming block instance(s) are stalled until at least one page of their input data streams, i.e. CSs, is filled up by the corresponding producer. A local reader can access the pages directly through the database buffer, so there is no need for memory copies. Otherwise, the transmission of the page is achieved through (PVM) messages and by memory to memory transfer. Each CS has a certain quota of buffer pages assigned to the producer. If this is exhausted, one of the following dataflow control strategies are used, depending on the *send* operator's parameters that are set by the parallelizer or dynamically reset by the QEC:

- **WRITEOUT:** The "newest" page is written to disk. The rationale behind this decision is that the older pages, that are still buffered, are the next to be read by the consumer.
- **NOBUF:** The original LRU replacement algorithm of the buffer management is used, hence no explicit quota has to be specified. However, in contrast to the WRITEOUT strategy, the CS pages may be replaced due to any page request to the database buffer. According to the LRU strategy, in this case the "older" pages are affected, although they are the next to be read by the consumer(s).
- **WAIT:** In this case the producer is stalled until the consumer requests more input.

Based on these definitions one can easily observe that WRITEOUT is just a NOBUF strategy restricted by a fixed buffer quota and a FIFO replacement strategy; WAIT refers to a fixed buffer quota without

any replacement, but with a hold request on the producer if the quota is exceeded. Obviously only the NOBUF and WRITEOUT strategies refer to materialization, hence we call these *send* operators at some places *materializing* nodes. If there is only one consumer instance (i.e. no replication or multi-query optimization), a page can be deleted as soon as it is read. This is achieved by an additional parameter, called *readonce*. Hence if this parameter is used in combination with WAIT, I/O can be avoided entirely.

3.3 Control of Data Partitioning

By streaming the output of one operator into the input of the other operator, the two can work concurrently giving inter-operator parallelism or pipelining. This is achieved by simply placing a *send-receive* pair on any edge of a QEP. Thus, the QEP in Fig. 3a has been split up in 3 blocks that communicate in a pipelining manner via CSs as described above.

Intra-operator parallelism in MIDAS is based on data partitioning. The splitting and merging of data streams is performed by the *send* and *receive* operators as well. For each partition a separate CS is created and filled by the *send* operator with corresponding tuples. In this way, each operator (or block) instance processes one partition of the data. The strategies implemented are **ROUND-ROBIN, HASH, RANGE,** and **USER-DEFINED partitioning** [JM99]. The particular technique used depends on the type of the operator that has to be parallelized. Thus the partitioning often has to keep track of the attribute values, like in the case of hash- or range-based partitioning. An example is given in Fig. 3b. Here, the *sort* operator has a degree of parallelism of 4, each instance processing a single partition of the initial data streamAs in the case of sorting a value-based partitioning is not necessary, a round-robin strategy has been chosen, indicated in Fig. 3b by the parameter *RR[4]* set for the *send* operator. The reason for this choice is that round robin partitioning is cheap an prohibits data skew.

In Fig. 3c, an example for repartitioning is presented, where a change from DOP=3 to DOP=4 has to be accomplished. As described in Section 2.3, for a *N:M* redistribution the data river consists of *N x M* data streams, each mapped to a separate CS instance. Thus, in this example 3 x 4 = 12 CS are used. For illustration purposes, in this example the *send* operator performs a hash partitioning on the (first) sorting attribute, indicated by parameter *H[4] [1]* of the *send* operator .

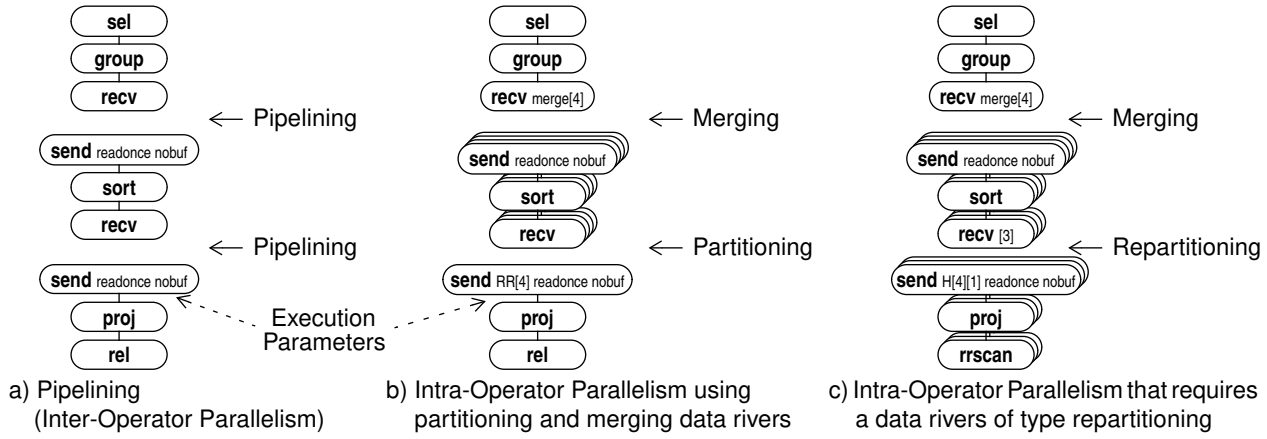


Fig. 3: Forms of Intra-Query Parallelism in MIDAS

3.4 Control of Data Merging

In order to combine several parallel data streams created by the *send* operator as described before, the *receive* operator has to read from multiple CSs as well. The communication between *send* and

receive can be based on polling or on a notification technique. In MIDAS, both alternatives are implemented, the first one being the default mode and the second one being triggered by the NOTIFY option. The decision on which mode to select is made by the QEC component, according to the relative costs of the consumer and producer instances. If the consumer is slower than the producers a polling technique is beneficial since most probably the first request for data can already be answered. In the opposite case, the NOTIFY alternative is more favorable.

With respect to the order in which the data streams are read, the following alternatives are possible:

- **MERGE:** In this case, the CSs containing locally sorted data streams are merged into a final sorted stream (see Fig. 3b and 3c).
- **SEQ:** Here one entire data stream is consumed before the next is worked on. As a side effect, a sorted output can be produced directly from range partitioned and locally sorted data streams.
- **ASYNCH:** In this mode, the tuples are read in their order of arrival. The output is unsorted. In contrast to the other modes presented, this one does not prescribe how the streams are merged. As soon as any page is available it is subject to consumption. Hence, we call this a *data-driven* consumption while the other two policies are called *demand-driven*.

3.5 Performance Measurements

With the concept presented, we could parallelize in MIDAS traditional and application-specific operators [NJM97]. This has been extended to user-defined functions [JM98]. In addition, parallel I/O is supported by new parallel scan operators that exploit specific storage structures and data fragmentation as well as physical properties of the resulting data streams (e.g. sorting, partitioning etc.).

As shown until now, apart of the decision on where to place a *send-receive* pair, several parameters related to the data river paradigm can influence the performance of a parallel execution plan. We have investigated these aspects by using a 100 MB TPC-D database running on a cluster of 4 Sun-ULTRA1 workstations with 143 MHz Ultra-SPARC processors and, for comparison purposes, also on a Sun-SPARC20 SMP, having 4 processors, each 100 MHz, and 4 disks.

As shown in Fig. 4, we have used a simple query consisting of the parallel scan of the LINEITEM table, performed by the *pscan* operator, followed by a grouping. In this scenario, the consumer subplan is slower than the producers. Since the LINEITEM table is physically partitioned onto 4 disks, the parallelizer has chosen to set the degree of parallelism of the scan operator to 4 as well. We have executed this query several times while modifying the *receive* modes. For each *receive* mode, the dataflow parameters of the *send* operator have been modified as well, as listed in the figure. Note that in the case

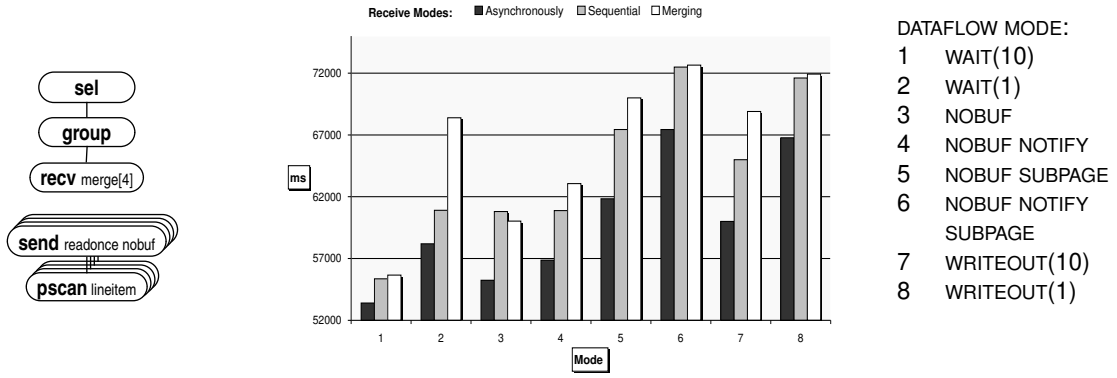
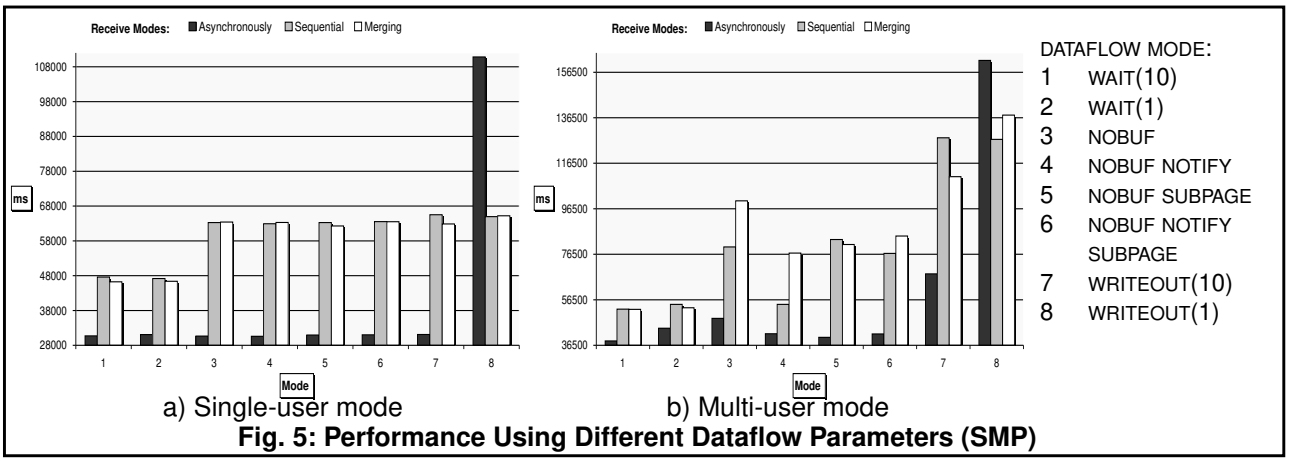


Fig. 4: Example Query and Performance Using Different Dataflow Parameters (Cluster)



a) Single-user mode b) Multi-user mode

Fig. 5: Performance Using Different Dataflow Parameters (SMP)

of an ASYNCH *receive* the output is not sorted; we have only listed it here for comparison purposes. The numbers in parenthesis show the buffer quota in pages for the WAIT and WRITEOUT strategies. The SUBPAGE option indicates that instead of a whole page, only a fraction of it (here 1/4) is used as dataflow granule. This leads to a finer granularity for the management of intermediate results, but also to more overhead due to the increased number of messages. In Fig. 5a the same test series are performed on the SMP machine, in Fig. 5b additionally using a noise factor as a constant load on the same system environment (i. e. disks and processors) simulating a multi-user scenario.

In real-life queries, the plans are more sophisticated and usually split into more blocks using also replication. In order to analyze these scenarios as well, we have parallelized and executed 16 different TPC-D queries on the above mentioned platforms. The *receive* mode parameters are determined by the parallelizer once for each query and left constant while changing the dataflow parameters. The average response times for the whole query set can be found in Table 1.

The results in Fig. 4 and Fig. 5 show that the ASYNCH *receive* mode produces the best performance, except for the WRITEOUT strategy on the SMP platform, when only one buffer page is allocated for intermediate results. This is due to the fact that all producer instances, in this scenario being faster than the consumer instance, have to spool their intermediate results to disk as soon as a page is filled up, thus causing a high disk contention. However, although the ASYNCH *receive* mode shows good results for all the other dataflow parameters analyzed, it is only usable if the final result doesn't have to fulfill any specific physical properties, as e.g. sorting. This property is only guaranteed by the other two (demand-driven) *receive* modes. Here, an advantage of the SEQ *receive* strategy over the MERGE one can be observed. This aspect is especially obvious for workstation cluster as the MERGE strategy generally involves a slightly increased communication overhead. However, the SEQ case requires a range-based partitioning on the sorting attribute. This constraint is not necessary in the case of the MERGE strategy. Hence the sort operator can be bundled into a block also with operators that require a different partitioning strategy than a range partitioning on the sort attribute (the details on block construction can be found in Section 4.2). In this case the results of the block instances only have to be combined by a MERGE *receive* node. Although several systems [Gr95] use only one (usually hash-based) partitioning strategy, the above discussion confirms that various strategies should be used for different problems.

Table 1: Average Response Times for TPC-D Query Set (ms)

	WRITEOUT(1) 1 USER	WRITEOUT(10) 1 USER	NOBUF 1 USER	WAIT(1) 1 USER	WAIT(10) 1 USER	WAIT(10) 2 USERS	NOBUF 2 USERS
SMP (4 x 100 MHz)	38713	38069	36615	35284	34824	61711	60123
Cluster (4 x 143 MHz)	23337	22472	22623	23794	23034	43724	41275

As for dataflow control, these tests show that a synchronized communication between subplans, using the WAIT option, is always favorable, especially if enough buffer pages are available. For this strategy, the difference between demand- and data-driven *receive* modes is also minimal. So why not use this dataflow strategy, combined with a demand-driven *receive*, without being obliged to find solutions for overflowing buffers as in the case of an ASYNCH *receive* and disk contention as in the case of materializing *send* operators? The answer is that although the demand-driven approach has the least synchronization overhead and produces correct outputs w.r.t. physical properties, it sometimes can produce deadlocks (see Section 4.1). One solution to this problem is the usage of an ASYNCH *receive* in parts of the PQEP where this is possible, or the usage of materializing *send* operators, like WRITEOUT or NOBUF. Another important remark is that in the simple example query we used for this test, the intermediate results have been read by a single consumer, making efficient garbage collection possible (*readonce* option). In this case the WAIT strategy eliminates completely the need for disk access and thus leads to a good performance. If replication or multi-query optimization is used, more consumers use the same intermediate results, hence these have to be materialized. In this case, the use of the WAIT option doesn't bring any remarkable benefits (see Table 1).

As expected in the example given in Fig. 4 (and Fig. 5) showing a slow consumer, the NOTIFY strategy has a negative influence on performance, due to the increased number of messages. Hence in such cases the (default) strategy based on polling should be used for communication between the *send* and *receive* operators. Using a finer granularity for communication (i.e. the SUBPAGE option) is beneficial e.g. on the SMP platform in a multi-user environment (Fig. 5 b); when the intermediate results have to be communicated over the network, larger units are preferable (Fig. 4). From the two materializing *send* strategies, NOBUF seems to be a good compromise when WAIT cannot be used because of deadlocks. However, the tests in Fig. 4 and Fig. 5 have been performed in an environment with a relatively large database buffer (1500 pages) that could be exploited by the NOBUF strategy. Column 2 and 3 of Table 1 show that the WRITEOUT strategy leads to a comparable performance even though having less buffer pages. This is due to the FIFO replacement strategy.

3.6 Recommendations for the Parametrization of Data Rivers

The results of our performance analysis can be described by a comprehensive set of parameter combinations that are recommended for specific situations. These recommendations are summarized in Table 2. In MIDAS, the parameters are first determined by the parallelizer, together with the decision on where to introduce *send/receive* nodes. The parallelizer takes the cost model and system statistics into account to find out which forms of parallelism provide optimal response time combined with minimal resource utilization. However, the outcome of this phase is only a preliminary result, as certain parameters can be further modified by the QEC at runtime. This is important, because the results in this section show that different parameters or parameter combinations are favorable, depending on the runtime environment, such as the platform where a block is scheduled, available database buffer,

Table 2: Parameter Combinations Recommended for Specific Situations

Situations		Receive Mode Parameter	Dataflow Mode Parameter	Data Partitioning Parameter
Resulting stream unsorted	• No special physical properties of resulting or intermediate data streams	ASYNCH	Buffer pages for communication • sufficient available: WAIT • else: NOBUF	ROUND-ROBIN (no data skew)
	• No special physical properties of resulting stream • Intermediate streams require partitioning on certain attributes	ASYNCH	Buffer pages for communication • sufficient available: WAIT • else: NOBUF	HASH
Resulting stream sorted	• No deadlock possible • Partitioning on sort attribute possible	SEQ	WAIT	RANGE (on sort attribute)
	• No deadlock possible • Partitioning required on attributes R, different from sort attribute	MERGE (on sort attribute)	WAIT	HASH (on attributes R)
	• Deadlock possible • Partitioning required on attributes R, different from sort attribute	MERGE (on sort attribute)	Buffer pages for communication • sufficient available: WRITEOUT • else: NOBUF	HASH (on attributes R)
	• Deadlock possible • Partitioning on sort attribute possible	SEQ	Buffer pages for communication • sufficient available: WRITEOUT • else: NOBUF	RANGE (on sort attribute)
Replication		ASYNCH	Buffer pages for communication • sufficient available: WRITEOUT • else: NOBUF	REPLICATION
Cost of producer higher than that of consumer		-	in addition NOTIFY option	-
SMP, multi-user environment		-	in addition SUBPAGE option	-

current number of users etc. With this concept of parametric PQEPs we achieve the necessary flexibility to be able to adapt in the best possible ways to these runtime situations.

Table 2 is structured as follows. The first column separates situations for which the other columns determine the corresponding parameters. First, we discuss the scenarios where no special physical properties are requested for the final stream. After this we present various situations that require a sorted result. We continue with the parameter settings for replication. Furthermore, it is worthwhile to identify the situations where the cost of the producer is higher than that of the consumer as well as multi-user contexts in SMP environments.

Focusing first on the *receive* modes, we can conclude from Table 2 that the ASYNCH mode is favorable for almost all situations. Exceptions are provided only by scenarios that require a sorted output.

The dataflow mode parameter in Table 2 is set by the QEC component (see Section 2.1) depending in some situations on the available number of buffer pages. If there are enough pages available, a dataflow mode that shows good performance in combination with a sufficiently large buffer quota as e.g. WAIT or WRITEOUT, should be chosen. Otherwise, the NOBUF strategy is advisable, as this can make use of buffer pages that are eventually freed at runtime, also by other queries.

As for the partitioning parameters, we recommend a round-robin strategy for all cases where this is possible, as this partitioning does not result into any data skew. In all other situations, the partitioning has to keep track of the actual parallelization strategy.

Our practical experience showed that a buffer quota is necessary in any case. The size of the quota has to be adapted according to general system state, i.e workload, and in accordance to the difference of the processing rate between consumer and producer blocks. Exactly for that reason our parallelizer builds blocks having similar processing rates (cf. [NM98a]). Our experiments showed that if both producer and consumer processing rate are in the same range, a buffer quota of less than 10 proved to be sufficient.

4. Efficient Embedding of Data Rivers into Query Processing

In this section we introduce and discuss indispensable measures to improve parallel query execution.

4.1 Deadlock Situations Caused by Intra-Operator Parallelism

The dataflow granule in MIDAS is a page (see Section 3.2). As shown in Section 3.5, a synchronized communication for intermediate results, using the WAIT option for the *send* operator, combined with a demand-driven *receive* mode, is the most favorable parameter combination, if certain physical properties (e.g. sorting) of the final data stream are important. In this case, flow control has to stall producer instances to prevent buffer overflow, as well as consumer instances until pages are filled up. These waiting situations can produce deadlocks when intra-operator parallelism is used. The reason is that intra-operator parallelism abolishes the tree structure of a QEP and thus can generate cycles.

In [NM98b], we have presented cyclic data dependencies that can lead to deadlock scenarios, e.g. within data rivers, in between data rivers as well as caused by binary operators. The solutions to resolve these situations are based on a controlled introduction of non-blocking, i.e. materializing *send* nodes along the data cycle. To avoid speedup degradation due to disk contention, this has to be done in a cost-based manner, taking into account intermediate result cardinalities and operator costs. The corresponding strategies, described in more detail in [NM98b], are incorporated into the parallelizer.

4.2 Reducing the Number and Size of Data Rivers

If each operator is parallelized separately, the granule or unit of parallelism is one operator. However, this strategy is suboptimal for practical database execution plans. Especially object-relational or DSS queries contain beside some expensive also several low-cost operators. With the above mentioned strategy a data river is necessary for each operator in the QEP, even if specific operators don't contribute significantly to overall performance improvements. This leads to a loss of efficiency w.r.t. communication costs and resource utilization.

Thus, in contrast to other strategies [GI97, GGS96], we incorporated into our model also the possibility of bundling together several operators into a single block, i.e. to perform several operators within a single execution unit. The strategy is cost-based, as it accounts for operator costs, selectivities, and

Table 3: Influence of Block Construction on Performance

(average over 16 TPC-D queries is reported)	Small Blocks	Combined Blocks
Avg. Execution Time (ms)	41686	23002
Avg. Speedup (vs. Sequential Execution)	2,4	4,34
Avg. Number of Execution Units	13.625	7.5

intermediate result sizes to determine block boundaries. Since mutually adjusted processing rates are prerequisite to efficient pipelining [MD95], one goal is also to achieve mutually adjusted processing rates among communicating blocks. Additionally, this strategy of building blocks by taking into consideration the sizes of intermediate results and performing the required block partitioning where it is most favorable, is an important instrument to reduce the size and number of data rivers and implicitly reducing communication overhead as well.

The degree of parallelism of the resulting blocks is adjusted by the parallelizer to the actual block processing costs, i.e. the sum of the costs of the constituting operators. This guarantees overall efficiency and is in contrast to approaches used by other PDBMSs as e.g. choosing the same DOP for the whole QEP or limiting the considered degrees to a few alternatives [BF97, Or98, JMP97].

To validate this approach, we have parallelized and processed 16 different TPC-D queries on the same workstation cluster as in Section 3.5. In one test series small blocks have been used, in some cases comprising only one operator, in the other series these blocks have been combined, adapting the degree of parallelism of the final block accordingly. The results, summarized in Table 3, show that cost-based block building combined with adapted degrees of parallelism throughout the PQEP contributes significantly to obtain optimal speedups, in the same time reducing resource utilization.

4.3 Reducing the Number of Data Streams and Execution Units

In several situations, the insertion of a data river is not imposed by cost-based decisions, i.e. to reduce response time by means of parallelization. For instance, if the partitioning strategies of two low-cost operators are different, a data river has to be inserted merely for repartitioning purposes, although the costs don't justify a parallel processing of the operators. To reduce the increased resource consump-

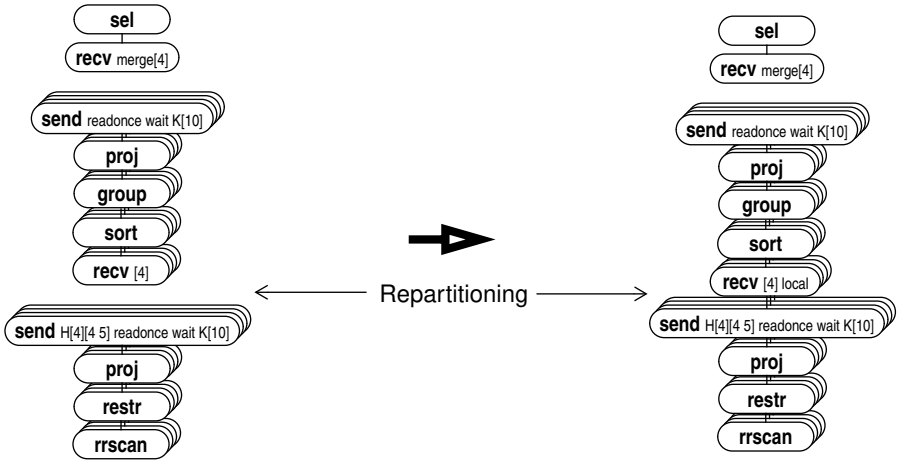


Fig. 6: Executing *send* and *receive* Within the Same Execution Unit

Table 4: Benefit of Execution Unit Combination

(average over applicable TPC-D queries)	Separate Execution Units	Combined Execution Units
Avg. Number of Execution Units	6,6	4,7
SMP avg. Execution Time (ms) - Single User	38985	35565
SMP avg. Execution Time (ms) - 2 Users	60136	52699
Cluster avg. Execution Time (ms) - Single user[1]	30171	26765

1. While comparing the results please note that the workstations in the cluster have more processing power than the SMP.

tion in this case, i.e. number of execution units and the communication costs, we have implemented the possibility to execute the *send* and *receive* operators within the same execution unit. Hence, the need for a data stream between *send* and *receive* operator instances that are processed within the same execution unit disappears. With this strategy, we deliberately renounce on parallel execution at some points which don't contribute to overall performance gains, thus saving resources.

As an example, consider TPC-D query Q1 (Fig. 6), parallelized for 4 processors and 4 disks. Here, a DOP of 4 is used throughout the PQEP. However, the grouping requires an attribute-based partitioning, in this case hash, indicated by the parameter (*H[4] [4 5]*). Thus, a repartitioning is necessary, realized by a data river of 4 x 4 = 16 data streams, leading to altogether 9 execution units. By bundling together *send* and *receive*, the number of execution units is reduced to 5. In addition, 4 data streams get substituted by communication within an execution unit, as visualized by the lines connecting inner-block operators. Hence the data river is now constituted only of 16 - 4 = 12 data streams.

The validation of this approach, using all applicable TPC-D queries and the same test scenario as before, can be found in Table 4. The results clearly show that this strategy leads to an increased efficiency, especially in a multi-user environment.

5. Related Work

Important research activities have been done in the area of parallel query optimization and execution. However, most previous work uses simplifying assumptions or concentrates on specific architectures and operator types that don't cover the requirement catalog coming from real-life application scenarios. In addition, language extensions or extensions to the database engine itself, as supported by object-relational DBMS, are hard to accomplish. Thus parallelization in many cases is restricted to join ordering in combination with restricted forms of parallelism [Has95] or is based on heuristics, as e.g. in XPRS [HS93]. Here, the exchange of intermediate results is done through temporary tables, but there is no discussion on how to organize these tables, which is one focus of our paper. Other approaches are highly specialized for specific, e.g. main-memory [BDV96] architectures or certain operator types. Thus, some research concentrated on scheduling of joins that maximizes the effect of specific forms of parallelism, elaborating concepts as right-deep [SD90], segmented right-deep [LC+93], zig-zag [ZZS93] and bushy trees [WFA95]. Although these strategies have achieved good performance for specific scenarios, they rely on the hash-join execution model and thus cannot be applied in a straightforward way to complex queries holding any kind of operators.

The most similar approach to the MIDAS execution system in a shared-memory environment is Volcano [Grae94]. Here, intermediate results are managed by a buffering control mechanism based on semaphores. Volcano adopts only a data-driven dataflow between the subplans, while having a de-

mand-driven approach within a subplan. Processes needed for the execution of subplans are forked dynamically through the *Exchange* operator. This again results into an architecture-specific solution. To reduce the number of processes, the *Exchange* operator was extended to run within a process' operator tree (similar to our approach described in Section 4.3), thus making inter-process communication demand-driven, too. The author claims this makes flow control obsolete. However, we cannot support this statement, as in the course of redistribution, the *Exchange* within a process also provides input for other processes, that can overflow. As shown in 4.1, this leads to a high deadlock probability, an issue that is not discussed in the Volcano paper. Furthermore, no experimental results are presented, which makes the developed techniques hard to be analyzed or to be compared to other approaches.

The Gamma prototype [DeG92] uses the *split* and *merge* operators in a similar way as MIDAS uses the *send* and *receive* operators. Apart from this, the query execution system of Gamma is quite different to ours, as it doesn't allow multiple operators to be processed within the same process. Later work [BFG95] has shown that this mechanism generates too many processes. Despite this result, the assumption that each operator is parallelized independently and the output of an operator being always repartitioned to serve as input to the next one is still the basis of more recent models [GI97, GGS96]. However, our results in Section 4.2 show that the optimal degree of parallelism of a set of operators differs from the optimal degree of parallelism of each separate operator, as in this way repartitioning can be avoided and larger blocks can be constructed, thus saving resources.

As for commercial databases, Informix Dynamic Server [Zo97, Inf98] uses some Volcano concepts for the query execution system. A special operator is used to pipe intermediate results from one set of operators to initiate another set of operators. A special dataflow manager is used to provide flow control and avoid spoolings to disk and probably also deadlocks, an issue not discussed there.

In DB2 UDB [Ibm98, JMP97, BFG95], the intermediate results are managed by so-called table queues that connect subsections of a plan. Only one (hash-based) redistribution strategy is used. But the major difference to MIDAS is that there is no synchronization between the sender and the receiver. The disadvantage of such a pure data-driven approach is that a special communication subsystem has to be implemented, that guarantees the correct order of arrival and the materialization/dropping of messages. Our measurements in Section 3.5 also show that the communication costs for such an approach are higher than in a synchronized communication. That is why in MIDAS we combine both methods, using the demand driven approach whenever possible and materializing only to avoid deadlocks. Another distinction to MIDAS is that the degree of parallelism of an operator is mainly determined by the partitioning of its inputs, while in MIDAS this is calculated according to the actual cost of the operator or the block the operator belongs to.

The Nonstop SQL/MX Database [Ta97] uses *split* and *collect* nodes for data partitioning that are similar to our *send* and *receive* nodes, except that only hash distribution is supported. To our knowledge, there exists no publication describing flow control mechanisms or resource management strategies. The degree of parallelism is chosen according to heuristics, like total number of processors, or number of outer table fragments for hash joins. In Oracle8 [Or98] multiple relational operators can be processed within the same process, but parallelism is limited to a 2-process depth at a time. Another simplifying aspect is that only one pre-determined degree of parallelism is used throughout the plan. This degree of parallelism can be only overridden by system limits and user hints. This approach of a uniform degree of parallelism for the entire PQEP is also adopted by Teradata [BF97], but proves to be suboptimal as confirmed by our experiments and performance analysis.

6. Conclusions and Outlook

In this paper we have presented and evaluated our approach to efficient management of intermediate query results in parallel database engines. Intra-query parallelism has been achieved by implementing the data river paradigm using communication segments. We have analyzed the importance of certain aspects in this paradigm, like dataflow control, partitioning, and merging of data streams as well as the granularity used for communication. Furthermore, we have shown how this paradigm can be significantly improved by block construction, cost-related degrees of parallelism and combination of execution units. We have pointed out the importance of these concepts towards achieving optimal speedup with minimal resource consumption. We have shown that the approach adopted by most commercial DBMSs of choosing a uniform degree of parallelism for the whole PQEP results in less efficiency, especially in forthcoming applications, like OLAP and parallel object-relational DBMS, where the operator costs in a plan can differ significantly.

All the above mentioned considerations have been incorporated into our rule- and cost-based parallelizer TOPAZ. Moreover, our approach to parameterize a PQEP allows to adapt important performance indicating parameters to the existing runtime situation. Since our data river implementation perfectly fits into state-of-the-art parallel database engine, any functional extensions to relational data processing (as e.g. recursion, (set-oriented) triggers, user-defined abstract data types, user-defined functions and user-defined tables) as being discussed for example in the object-relational context can still benefit from our approach in case of parallel processing. Even if the herein mentioned communication patterns do not suffice for some particular extension, we are still confident that new communication patterns can be easily reflected within our data river approach, because of its modularization issues. As for future work, we plan more experiments concentrating on OLAP and DSS scenarios as well as on object-relational test cases.

Acknowledgment
We gratefully acknowledge the cooperation of the whole MIDAS project staff with the prototype.

7. Literature

BF97 C. Ballinger, R. Fryer: Born to be Parallel, In: Bulletin of the IEEE Computer Society TCDE, Vol. 20, No. 2, 1997

BFG95 C. K. Baru, G. Fecteau, A. Goyal, H. Hsiao, A. Jhingran, S. Padmanabhan, G. P. Copeland, W.G. Wilson: DB2 Parallel Edition, In: IBM Sytems Journal, Vol 34, No 2, 1995.

BDV96 L. Bouganim, B. Dageville, P. Valduriez: Adaptive Parallel Query Execution in DBS3", EDBT Conf., Avignon, 1996.

BFV96 L. Bouganim, D. Florescu, P. Valduriez: Dynamic Load Balancing in Hierarchical Parallel Database Systems, In: Proc. of the 22nd VLDB Conference, Bombay, India, 1996.

BJ96 G. Bozas, M. Jaedicke, A. Listl, B. Mitschang, A. Reiser, S. Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project, Proceedings EUROPAR '96, 1996.

BV98 S. Brobst, B. Vecchione: DB2 UDB: Starburst, In: Database Programming & Design, Feb. 1998.

CJ+97 A. Clausnitzer, M. Jaedicke, B. Mitschang, C. Nippl, A. Reiser, S. Zimmerman: On the Application of Parallel Database Technology for Large Scale Document Management Systems, Proceedings IDEAS97, Montreal, Canada.

DaG95 D. Davidson, G. Graefe: Dynamic Resource Brokering for Multi-User Query Execution, In: Proceedings of the 1995 ACM SIGMOD Intl. Conf. on Management of Data, San Jose, 1995.

DeG92 D. DeWitt, J. Gray: Parallel Database Systems: The Future of High Performance Database Systems, In: CACM, Vol.35, No.6, pp.85-98, 1992.

GGS96 S. Ganguly, A. Goel, A. Silberschatz: Efficient and Accurate Cost Models for Parallel Query Optimization, In: Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems,, Montreal, 1996.

GI97 M. Garofalakis, Y. Ioannidis: Parallel Query Scheduling and Optimization with Time- and Space-Shared Resources, In: Proceedings of the 23rd VLDB Conference, Athens, Greece, 1997.

Ge94 Geist, A. et al.: PVM 3 User's Guide and Reference Manual. TR ORNL/TM12187, Oak Ridge Nat. Lab., 1994.

Grae94 G. Graefe: Volcano-An Extensible and Parallel Query Execution System, In: TKDE 6(1), February 1994.

Grae95 G. Graefe: The Cascades Framework for Query Optimization, In: Bulletin of the TCDE, Vol 18, No 3, Sept 1995.

Gr95 Gray, J.: A Survey of Parallel Database Techniques and Systems, In: Tutorial Handout VLDB Conf., Zurich, 1995.

GB+96 J. Gray, A. Bosworth, A. Layman, H. Pirahesh: Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals, In: Proc. of the Intl. Conf. on Data Engineering, New Orleans, 1996.

Has95 W. Hasan: Optimization of SQL Queries for Parallel Machines, PhD Thesis, Stanford University, 1995.

HFV96 W. Hasan, D. Florescu, P. Valduriez: Open Issues in Parallel Query Optimization, In: SIGMOD Record 25(3), 1996.

HS93 W. Hong, M. Stonebraker: Optimization of Parallel Query Execution Plans in XPRS, In: Distributed and Parallel Databases, pp. 9-32, 1993.

Ibm98 IBM DB2 Universal Database, Version 5, IBM Corp., 1998.

Inf98 Extended Parallel Option for Informix Dynamic Server: Technical Brief, Informix Software, http://www.informix.com.

JM99 M. Jaedicke, B. Mitschang: User-Defined Table Operators: Enhancing Extensibility for ORDBMS, Proc. VLDB Conference, Edinburgh, 1999.

JM98 M. Jaedicke, B. Mitschang: A Framework for Parallel Processing of Aggregate and Scalar Functions in Object-Relational DBMS, Proc. SIGMOD Conf., Seattle, 1998.

JMP97 A. Jhingran, T. Malkemus, S. Padmanabhan: Query Optimization in DB2 Parallel Edition, In: DE Bull. 20(2), 1997.

Li94 Listl, A.: Using Subpages for Cache Coherency Control in Parallel Database Systems, Proc. PARLE Conf., 1994.

LC+93 M.-L. Lo, M.-S. Chen, C. V. Ravishnakar, P. S. Yu: On Optimal Processor Allocation to Support Pipelined Hash Joins", In: Proc. of the 1993 ACM SIGMOD Intl. Conf. on Managemant of Data, Washington D. C., June 1993.

MD95 M. Mehta, D. DeWitt: Managing Intra-operator Parallelism in Parallel Database Systems, In: Proceedings VLDB Conference, Zurich, Swizerland, 1995.

NJM97 C. Nippl, M. Jaedicke, B. Mitschang: Accelerating Profiling Services by Parallel Database Technology, In: Proceedings of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications, Las Vegas, 1997.

NM98a C.Nippl, B. Mitschang: TOPAZ: a Cost-Based, Rule-Driven, Multi-Phase Parallelizer, Proc. VLDB Conference, New York City, 1998.

NM98b C.Nippl, B. Mitschang: Towards Deadlock-Preventing Query Optimization and Parallelization, In: Proc. Intl. Conf. on Parallel and Distributed Computing Systems, Chicago, Illinois, 1998.

NZT96 M. Norman, T. Zurek, P. Thanisch: Much Ado about Shared-Nothing, In: ACM Sigmod Records, 23(3), 1996.

Or98 Oracle8 Server Concepts, Oracle Corporation, 1998.

Qi96 Qian, X.: Query Folding, in: IEEE Data Engineering Conference, pp. 48-55, 1996.

Se88 Sellis, T.K.: Multi-Query Optimization, in: ACM Transactions on Database Systems, Vol. 13, No.1, 1988.

SD90 D. Schneider, D. DeWitt: Tradeoffs in processing complex join queries via hashing in multi-processor database machines, In: Proc. of the Intl. VLDB Conference, Melbourne, Australia, 1990.

Ta97 Nonstop SQL/MX Database, Tandem Computers Inc., 1997.

TD93 J. Thomas, S. Dessloch: A Plan-Operator Concept for Client-Based Knowledge Processing, Proc. 19th VLDB Conference, Dublin, Ireland.

Va93 Valduriez, P.: Parallel Database Systems: Open Problems and New Issues, In: Distributed and Parallel Databases, Vol.1, No. 2, pp.137-166, April 1993.

WFA95 A. Wilschut, J. Flokstra, P. Apers: Parallel Evaluation of Multi-Join Queries, In: Proc. SIGMOD Conf, 1995.

Zo97 C. Zou: XPS: A High Performance Parallel Database Server, In: Bulletin of the IEEE TCDE, Vol. 20, No. 2, 1997.

ZZS93 M. Ziane, M. Zait, B. Salamet: Parallel Query Processing with ZigZag Trees, In: VLDB Journal, 2(3), March 1993.

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

Liste aller erschienenen Berichte von 1990-1994
auf besondere Anforderung

342/01/95 A Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids

342/02/95 A Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law

342/03/95 A Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space

342/04/95 A Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes

342/05/95 A Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems

342/06/95 A Maximilian Fuchs: Formal Design of a Model-N Counter

342/07/95 A Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology

342/08/95 A Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies

342/09/95 A Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness

342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design

342/11/95 A Thomas Schnekenburger: The ALDY Load Distribution System

342/12/95 A Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm

342/13/95 A Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming

342/14/95 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks

342/15/95 A Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs

342/16/95 A Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik

342/17/95 A Georg Stellner: Using CoCheck on a Network of Workstations

342/18/95 A Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications

342/19/95 A Thomas Schnekenburger: Integration of Load Distribution into ParMod-C

342/20/95 A Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication

342/21/95 A Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control

342/22/95 A Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes

342/23/95 A Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation

342/24/95 A M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems

Reihe A

342/01/96 A Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries

342/02/96 A Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs

342/03/96 A Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes

342/04/96 A Thomas Huckle: Efficient Computation of Sparse Approximate Inverses

342/05/96 A Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification

342/06/96 A Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components

342/07/96 A Richard Mayr: Some Results on Basic Parallel Processes

342/08/96 A Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht

342/09/96 A P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme

342/10/96 A Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFSLib – A File System for Parallel Programming Environments

342/11/96 A Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions

342/12/96 A Javier Esparza: Reachability in Live and Safe Free-Choice Petri Nets is NP-complete

342/13/96 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Many-to-Many Data-flow Networks

342/14/96 A Giannis Bozas, Michael Jaedicke, Andreas Listl, Bernhard Mitschang, Angelika Reiser, Stephan Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project

342/15/96 A Richard Mayr: A Tableau System for Model Checking Petri Nets with a Fragment of the Linear Time μ -Calculus

342/16/96 A Ursula Hinkel, Katharina Spies: Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen

342/17/96 A Richard Mayr: Model Checking PA-Processes

342/18/96 A Michaela Huhn, Peter Niebert, Frank Wallner: Put your Model Checker on Diet: Verification on Local States

342/01/97 A Tobias Müller, Stefan Lamberts, Ursula Maier, Georg Stellner: Evaluierung der Leistungsfähigkeit eines ATM-Netzes mit parallelen Programmierbibliotheken

342/02/97 A Hans-Joachim Bungartz and Thomas Dornseifer: Sparse Grids: Recent Developments for Elliptic Partial Differential Equations

342/03/97 A Bernhard Mitschang: Technologie für Parallele Datenbanken - Bericht zum Workshop

342/04/97 A nicht erschienen

342/05/97 A Hans-Joachim Bungartz, Ralf Ebner, Stefan Schulte: Hierarchische Basen zur effizienten Kopplung substrukturierter Probleme der Strukturmechanik

342/06/97 A Hans-Joachim Bungartz, Anton Frank, Florian Meier, Tilman Neunhoeffer, Stefan Schulte: Fluid Structure Interaction: 3D Numerical Simulation and Visualization of a Micropump

342/07/97 A Javier Esparza, Stephan Melzer: Model Checking LTL using Constraint Programming

342/08/97 A Niels Reimer: Untersuchung von Strategien für verteiltes Last- und Ressourcenmanagement

342/09/97 A Markus Pizka: Design and Implementation of the GNU INSEL-Compiler gic

Reihe A

342/10/97 A Manfred Broy, Franz Regensburger, Bernhard Schätz, Katharina Spies: The Steamboiler Specification - A Case Study in Focus

342/11/97 A Christine Röckl: How to Make Substitution Preserve Strong Bisimilarity

342/12/97 A Christian B. Czech: Architektur und Konzept des Dycos-Kerns

342/13/97 A Jan Philipps, Alexander Schmidt: Traffic Flow by Data Flow

342/14/97 A Norbert Fröhlich, Rolf Schlagenhaft, Josef Fleischmann: Partitioning VLSI-Circuits for Parallel Simulation on Transistor Level

342/15/97 A Frank Weimer: DaViT: Ein System zur interaktiven Ausführung und zur Visualisierung von INSEL-Programmen

342/16/97 A Niels Reimer, Jürgen Rudolph, Katharina Spies: Von FOCUS nach INSEL - Eine Aufzugssteuerung

342/17/97 A Radu Grosu, Ketil Stølen, Manfred Broy: A Denotational Model for Mobile Point-to-Point Data-flow Networks with Channel Sharing

342/18/97 A Christian Röder, Georg Stellner: Design of Load Management for Parallel Applications in Networks of Heterogenous Workstations

342/19/97 A Frank Wallner: Model Checking LTL Using Net Unfoldings

342/20/97 A Andreas Wolf, Andreas Kmoch: Einsatz eines automatischen Theorembeweisers in einer taktikgesteuerten Beweisumgebung zur Lösung eines Beispiels aus der Hardware-Verifikation – Fallstudie –

342/21/97 A Andreas Wolf, Marc Fuchs: Cooperative Parallel Automated Theorem Proving

342/22/97 A T. Ludwig, R. Wismüller, V. Sunderam, A. Bode: OMIS - On-line Monitoring Interface Specification (Version 2.0)

342/23/97 A Stephan Merkel: Verification of Fault Tolerant Algorithms Using PEP

342/24/97 A Manfred Broy, Max Breitling, Bernhard Schätz, Katharina Spies: Summary of Case Studies in Focus - Part II

342/25/97 A Michael Jaedicke, Bernhard Mitschang: A Framework for Parallel Processing of Aggregat and Scalar Functions in Object-Relational DBMS

342/26/97 A Marc Fuchs: Similarity-Based Lemma Generation with Lemma-Delaying Tableau Enumeration

342/27/97 A Max Breitling: Formalizing and Verifying TimeWarp with FOCUS

342/28/97 A Peter Jakobi, Andreas Wolf: DBFW: A Simple DataBase FrameWork for the Evaluation and Maintenance of Automated Theorem Prover Data (incl. Documentation)

342/29/97 A Radu Grosu, Ketil Stølen: Compositional Specification of Mobile Systems

342/01/98 A A. Bode, A. Ganz, C. Gold, S. Petri, N. Reimer, B. Schiemann, T. Schnekenburger (Herausgeber): "Anwendungsbezogene Lastverteilung", ALV'98

342/02/98 A Ursula Hinkel: Home Shopping - Die Spezifikation einer Kommunikationsanwendung in FOCUS

342/03/98 A Katharina Spies: Eine Methode zur formalen Modellierung von Betriebssystemkonzepten

342/04/98 A Stefan Bischof, Ernst W. Mayr: On-Line Scheduling of Parallel Jobs with Runtime Restrictions

342/05/98 A St. Bischof, R. Ebner, Th. Erlebach: Load Balancing for Problems with Good Bisectors and Applications in Finite Element Simulations: Worst-case Analysis and Practical Results

342/06/98 A Giannis Bozas, Susanne Kober: Logging and Crash Recovery in Shared-Disk Database Systems

342/07/98 A Markus Pizka: Distributed Virtual Address Space Management in the MoDiS-OS

342/08/98 A Niels Reimer: Strategien für ein verteiltes Last- und Ressourcenmanagement

342/09/98 A Javier Esparza, Editor: Proceedings of INFINITY'98

342/10/98 A Richard Mayr: Lossy Counter Machines

Reihe A

342/11/98 A Thomas Huckle: Matrix Multilevel Methods and Preconditioning

342/12/98 A Thomas Huckle: Approximate Sparsity Patterns for the Inverse of a Matrix and Preconditioning

342/13/98 A Antonin Kucera, Richard Mayr: Weak Bisimilarity with Infinite-State Systems can be Decided in Polynomial Time

342/01/99 A Antonin Kucera, Richard Mayr: Simulation Preorder on Simple Process Algebras

342/02/99 A Johann Schumann, Max Breitling: Formalisierung und Beweis einer Verfeinerung aus FOCUS mit automatischen Theorembeweisern – Fallstudie –

342/03/99 A M. Bader, M. Schimper, Chr. Zenger: Hierarchical Bases for the Indefinite Helmholtz Equation

342/04/99 A Frank Strobl, Alexander Wisspeintner: Specification of an Elevator Control System

342/05/99 A Ralf Ebner, Thomas Erlebach, Andreas Ganz, Claudia Gold, Clemens Harlfinger, Roland Wismüller: A Framework for Recording and Visualizing Event Traces in Parallel Systems with Load Balancing

342/06/99 A Michael Jaedicke, Bernhard Mitschang: The Multi-Operator Method: Integrating Algorithms for the Efficient and Parallel Evaluation of User-Defined Predicates into ORDBMS

342/07/99 A Max Breitling, Jan Philipps: Black Box Views of State Machines

342/08/99 A Clara Nippl, Stephan Zimmermann, Bernhard Mitschang: Design, Implementation and Evaluation of Data Rivers for Efficient Intra-Query Parallelism

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B Jörg Desel: On Abstraction of Nets
342/3/90 B Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B Barbara Paech: Concurrency as a Modality
342/2/91 B Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox - Anwenderbeschreibung
342/3/91 B Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B W. Reisig: Concurrent Temporal Logic
342/1/92 B Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support
Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS