Open Issues in Parallel Query Optimization

WAQAR HASAN*
Hewlett-Packard Labs, USA
hasan@cs.stanford.edu

DANIELA FLORESCU
INRIA, France
Daniela.Florescu@inria.fr

PATRICK VALDURIEZ
INRIA, France
Patrick.Valduriez@inria.fr

Abstract

We provide an overview of query processing in parallel database systems and discuss several open issues in the optimization of queries for parallel machines.

1 Introduction

Parallel database systems combine data management and parallel processing techniques to provide high-performance, high-availability and scalability for data-intensive applications [10, 35]. By exploiting parallel computers, they provide performance at a cheaper price than traditional mainframe solutions. Further, they are the solution of choice for high transaction throughput in OLTP systems as well as low response times in decision-support systems. Finally, parallel databases are the only viable solution for very large databases.

SQL, the standard language for programming database access, is a high-level, set-oriented, declarative language. This permits SQL compilers to automatically infer and exploit parallelism. Users do not have to learn a new language and application code does not need to be rewritten to benefit from parallel execution. This is to be contrasted to the use of lower-level languages in scientific computing which often requires re-writing application code to take advantage of parallel machines.

A key to the success of parallel database systems, particularly in decision-support applications, is *parallel query optimization*. Given a SQL query, parallel query optimization has the goal of finding a parallel plan that delivers the query result in minimal time. While considerable progress has been made, several problems remain open. Further, solutions to the optimization problem are sensitive to the query language expressive power, the underlying execution mechanisms, the machine architecture, and variations in the cost metric being minimized. New applications, demands for higher performance from existing applications, innovations in query execution mechanisms and machine architectures are changing some of the underlying assumptions thereby offering new challenges.

Parallel query optimization offers challenges beyond those addressed by past research in fields such as distributed databases [30] or classical scheduling theory [18]. While distributed and parallel databases are fundamentally similar, research in distributed query optimization was done in the early 1980s, a time at which communication over a network was prohibitively expensive and computer equipment was not cheap enough to be thrown at parallel processing. Most work [27] focused on minimizing work (total resource consumption) rather than response time. Response time was considered [2] only for execution spaces that allowed independent parallelism but did not allow pipelined or partitioned parallelism. The latter two forms of parallelism have also not been addressed in classical scheduling theory.

In this paper, we describe some open issues in parallel query optimization and propose some directions of research. We first provide a brief overview of parallel architectures (Section 2), and of parallel query execution (Section 3) and optimization (Section 4). Section 5 introduces the new issues and Section 6 concludes.

2 Parallel Machine Architectures

Parallel system architectures range between two extremes, *shared-memory* and *shared-nothing* (see Figure 1). There are interesting intermediate architectures such as shared-disk which we omit for brevity.

In shared-memory systems all processors may access all memory modules and all disks. Examples are HP T500, Bull's Escala, SGI Challenge, Cray CS6400 as well as mainframes such as IBM3090 and Cray YMP. Examples of shared-memory parallel database systems include research prototypes such as XPRS [25], DBS3 [4] and Volcano [16], as well as commercial products such as Informix 7.2 Online Dynamic Server [26], Oracle 7.3/Parallel Query Option [29] and IBM DB2/MVS [28].

In shared-nothing systems, each processor has exclusive access to its main memory and a subset of the disks. Tandem Himalaya, IBM SP2 [1] as well as clusters of workstations connected by commodity interconnects such as ATM are examples of shared-nothing machines. Examples of shared-nothing parallel database systems include commercial products such as Tandem NonStop-SQL [34, 12], IBM DB2 Parallel Edition [3], ATT GIS Teradata as well as research prototypes such as Bubba [5] and Gamma [11].

The main advantage of shared-memory is simplicity. Since meta-information (directory) and control information (e.g., lock table) is shared by all processors, writing database software is not very different than for single-processor computers. However, balancing processor and disk loads presents a problem. As compared to shared-nothing, load balancing problems are simpler since any

**Current address: Informix Software, 4100 Bohannon Drive, Menlo Park, CA 94025, USA*

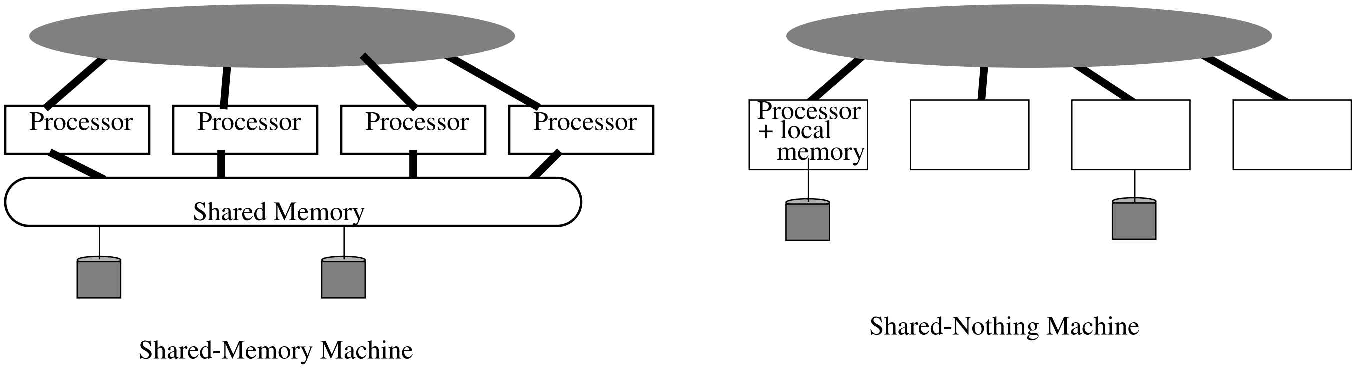

Figure 1: Shared-Memory and Shared-Nothing Architectures

processor may access any disk, jobs may be pre-empted cheaply and communication costs are low.

Sharing memory among processors leads to three problems: limited scalability, high cost and low availability. As the number of processors increase, conflicting accesses to the shared-memory rapidly degrade performance. Retaining performance requires special-purpose and costly design of the bus and memory. Therefore, scalability is limited to tens of processors. The architecture hurts availability since the memory space is shared by all processors. A memory fault may affect most processors.

Shared-nothing architectures solve scalability and availability problems by reducing interference between processors. Tandem and Teradata have demonstrated commercial installations with hundreds of processors. As remarked earlier, load balancing is harder in shared-nothing systems. Shared-memory systems can offer the best price-performance when the numbers of processors is small. This has led to hybrid architectures which consist of a shared-nothing system in which each node is a shared-memory multi-processor. Examples are Encore 93 and ATT GIS P90 [7].

3 Parallel Query Execution

A procedural plan for a SQL query is conventionally represented as an annotated query tree. Such trees encode procedural choices such as the order in which operators are evaluated and the method for computing each operator. Each tree node represents one (or several) relational operators. Annotations on the node represent the details of how it is to be executed. For example a join node may be annotated as being executed as a hash-join and a base relation may be annotated as being accessed by an index-scan. The EXPLAIN statement of most SQL systems allows such trees to be viewed by a user.

The work in computing a query may be partitioned using three forms of parallelism: *independent*, *pipelined* and *partitioned*. Two operators neither of which uses data produced by the other may simultaneously run on distinct processors. This is termed independent parallelism. Since operators produce and consume sets of tuples, the tuples output by a producer can sometimes be fed to a consumer as they get produced. Such concurrency is termed *pipelining* and, when the producer and consumer use distinct processors, is termed pipelined parallelism. Intra-operator parallelism based on partitioning of data is termed *partitioned* parallelism.

There are intrinsic limits on the benefit from parallel execution due to *constraints on available parallelism* and due to the *overheads* of parallel execution.

Available parallelism is constrained by several factors. Inter-operator parallelism is constrained by timing constraints between operators. For example, a hash join works by first building a hash table on one operand and then, probing the hash table for matches using tuples of the second operand. Since the hash table must be fully built before being probed, there is a *precedence* constraint in the computation. As another example, an operator that scans a table may pipe its output to the operator that build a hash table. Such concurrency eliminates the need to buffer intermediate results. It, however, places a *parallel* constraint in the computation. In many machine architectures, data on a specific disk may only be accessed by the processor that controls the disk. Thus *data placement* constraints limit both inter and intra-operator parallelism by localizing scan operations to specific processors. For example, if an Employee table is stored partitioned by department, a selection query that retrieves employees from a single department has no available parallelism.

Using parallel execution requires starting and initializing processes. These processes may then communicate substantial amounts of data. These *startup* and *communication* overheads increase total work. The increase is significant and may offset the advantages of parallel execution in some cases [12].

Example 3.1 Figure 2 shows a query tree and the corresponding operator tree. Thin edges are pipelining edges that represent parallel constraints. Thick edges are blocking edges that represent precedence constraints. A simple hash join is broken into `Build` and `Probe` operators. Since a hash table must be fully built before it can be probed, the edge from `Build` to `Probe` is blocking. A sort-merge join sorts both inputs and then merges the sorted streams. The merging is implemented by the `Merge` operator. In this example, we assume the right input of sort-merge to be pre-sorted. The operator tree shows the sort required for the left input broken into two operators `FormRuns` and `MergeRuns`. Since the merging of runs can start only after run formation, the edge from `FormRuns` to `MergeRuns` is blocking.

The operator tree exposes the available parallelism and timing constraints among operators. Partitioned parallelism may be used for any operator. Pipelined parallelism may be used between two operators connected by a pipelining edge. Two subtrees with no (transitive) timing constraints between them may run independently (eg: subtrees rooted at `FormRuns` and `Build`). □

4 Parallel Query Optimization

The optimization problems in the context of parallel machines can be understood with respect to the two-phase view [24, 21] shown in Figure 3. The breakup into phases provides a way of conquering problem complexity similar to the use of phases in programming language compilation. It eases both the understanding of the problems as well as the development of solutions.

The first phase, JOQR (for Join Ordering and Query Rewrite), produces an annotated query tree that fixes aspects such as the order of joins and the strategy for computing each join. The second phase, *parallelization*, converts the annotated query tree into a parallel plan. The JOQR phase includes problems similar to conventional query optimization. Parallelization does not have a counterpart in conventional optimization.

Critical aspects of parallel execution that interact with the decisions of the first phase can be incorporated into the models and algorithms used for JOQR. For example, expensive repartitioning of data is needed if one or both operands of a join are not partitioned on the join attribute. Such repartitioning may be avoided by changing the choice of the join predicate or the order of joins [23, 21, 20]. We remark that some two-phase approaches reuse a conventional optimizer for the JOQR phase [24].

Following the design of conventional optimizers (such as in Starburst [19]) JOQR may be broken into two steps. The first rewrites queries based on heuristics while the second uses a cost-model to fix the order of operations and selects methods for computing each operator (for example join and access methods).

Parallelization may also be broken into two steps. The first extracts parallelism by macro-expanding the annotated query tree to an operator tree. The operator tree identifies the pieces of code (operators) that should be considered to be atomic by the scheduler as well as the timing constraints between operators. The second step schedules the operator tree on the parallel machine. The goal of a scheduler is to allocate machine resources so as to exploit the available parallelism while respecting timing and data placement constraints.

The two phases pose optimization problems at different levels of abstraction. An optimization problem is modeled by specifying an *execution space* that defines the space of choices and a *cost model* that assigns a cost to each execution. Given such a model, *search algorithms* that minimize cost may be devised.

Models employed for the two phases are usually quite different due to the nature of the problems. JOQR focuses on algebraic transformations and selection of strategies for each high-level operator. The model therefore abstracts away facets such as allocation of machine resources. The cost metric is either work (i.e. total resource consumption) or a very rough guess of response time. The parallelization phase, on the other hand, takes a fixed procedural plan and focuses on allocating machine resources. The model includes a detailed view of machine resources and the cost metric is response time.

It is clearly not essential for optimization algorithms to be developed using a strict two-phase view. For example JOQR may generate a set of plans or the two phases may be integrated.

5 Some Open Issues

We now discuss several open issues in parallel query optimization. Many of these issues apply generally to query optimization but parallelism makes them harder.

5.1 Heterogeneous Machines

A standard assumption in most research is to consider all nodes of a parallel machine to be identical. However, it is desirable for database software to work in heterogeneous environments.

One often touted advantage of parallel machines is the ability to incrementally add components (processors, disks). By the time a user needs more computing power, newer and faster components are likely to be available.

Another scenario for heterogeneity is the existence of a large number of diverse machines in most office environments. The machines differ in processor speed, the amount of memory, and the speed and number of attached disks. Many of these machines are under-utilized, especially at night. Commodity interconnects such as Myrinet, FDDI or an ATM switch may be used to turn idle machine cycles into a useful parallel machine.

Distributed information systems may also be enabled by the ubiquity of WANs such as the Internet. At one

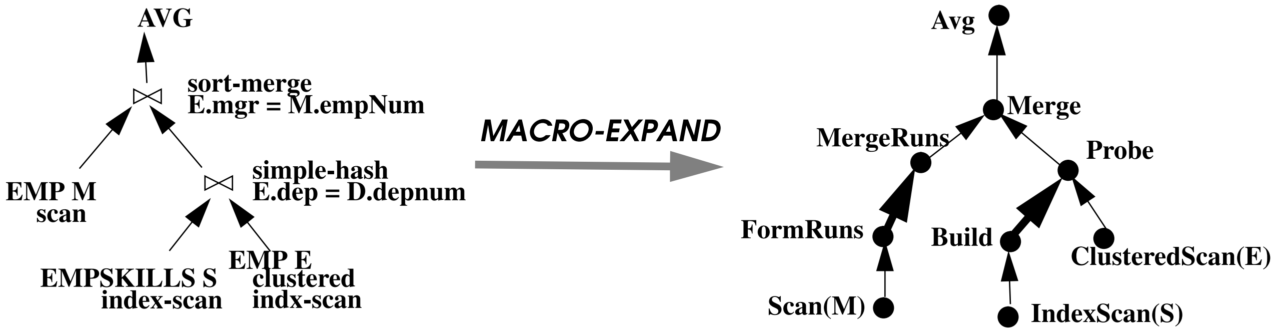

Figure 2: Macro-Expansion of Query Tree to Operator Tree (Parallelism Extraction)

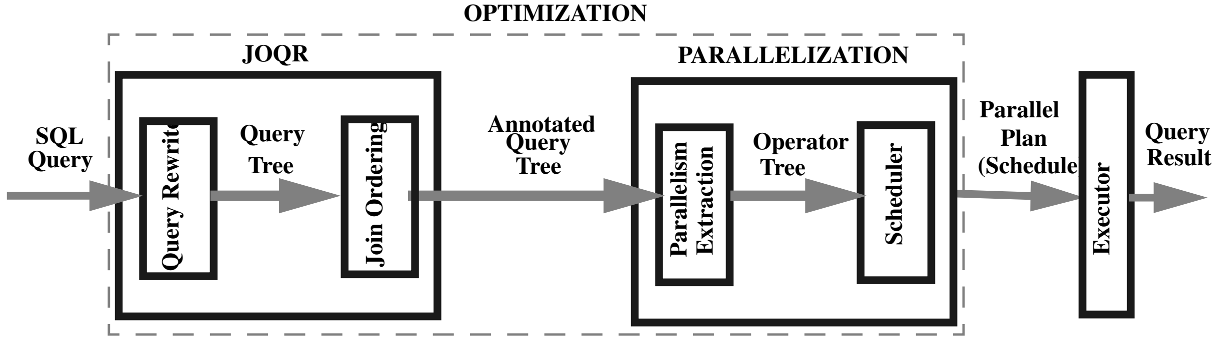

Figure 3: Parallel Query Optimization: A Two-phase View

end of the spectrum, when there is strong central control, these problems may be modeled by considering the system to be a heterogeneous parallel system. At the other end, decentralized resources management poses extremely challenging problems [33]. Further, these new environments may require new optimization objectives such as minimizing monetary cost to the end-user given response time constraints, or minimizing response time given a fixed budget.

5.2 Dynamic/Pre-emptive Optimization

The machine resources available for executing a query may change while the query is in execution. For example, another query may complete and release resources. This motivates the need for dynamic revision [6] of scheduling decisions.

We observe that the additional freedom to revise scheduling decisions gave two advantages in classical scheduling problems such as multi-processor scheduling. Firstly, pre-emptive schedule are better than non-preemptive schedules. Secondly, the algorithmic problems get simplified.

It can be costly to pre-empt a query that uses a large number of resources on a parallel machine. Any pre-emptive scheme must account for the trade-off between the cost and benefit of pre-emption.

Optimization decisions other than scheduling may also benefit from revision at execution time [31, 17]. Join ordering is sensitive to estimates of intermediate result sizes. It is well known that such estimates may have large errors and better information may be available at execution time.

5.3 Space-Time Tradeoffs

Exploiting parallelism poses a host of scheduling problems that may be characterized along two dimensions: the *machine* model and the *task* model. The machine model represents resources such as processors, disks, memory and the network. The task model consists of operator tree and the degrees of freedom that the scheduler is allowed (i.e. the use of partitioned, pipelined and independent parallelism).

One challenge is to incorporate space-time tradeoffs in the task model. It is well known that additional memory can be exploited to reduce the I/O and CPU cost of operations such as sorting. In a parallel machine, more memory is obtained by spreading computation over a larger number of processors – thus I/O and CPU can be traded for memory and communication. It is challenging to devise models and algorithms that minimize response time subject to limits on maximum memory usage while taking this trade-off into account.

5.4 Extended/Non-Relational Data Models

Most research on parallel query optimization is in the context of the relational model. All parallel database products are relational. The penetration of database technology into new domains such as technical and scientific applications is extending DBMS functionality with object and rule capabilities. Examples of new functionality (as in SQL3 and ODMG standards) are sequence or graph data structures, path expressions, foreign functions (written in programming languages), and passive and active rules. Procedural extensions to database systems incorporate control features such as sequencing, conditionals and loops.

One challenge is to determine how the models and algorithms developed for parallel query optimization need to change for these extended data models. The FAD language of Bubba and the Flora language of the IDEA system [13] have operators that express various forms of parallelism over an object data model. The SVP model [32] allows sets, sequences and parallelism to be captured in a unified framework formalizing divide-and-conquer mappings. These languages are possible formal foundations for further research in parallel database languages and parallel query optimization.

5.5 Evaluation of Algorithms

Two important aspects of evaluating an algorithm are the quality of the plans and the running time of the algorithm. Standard benchmarks such as TPC-D provide a measure of overall system performance. They do not measure time spent on optimization and cannot isolate the quality of the optimizer from the quality of other system components. Further, they are not designed to stress test parallel query optimizers. Thus, one challenge is to develop standard criteria for evaluating optimization algorithms.

We believe it is important to measure the quality of plans by comparison with the *optimal* plan. One metric for the quality of plans is the *performance ratio* [15] which is the ratio of the cost of the produced plan to the cost of the optimal plan. This measure has several advantages. Firstly, the fact that it is a relative measure, allows the quality of plans generated to be measured independent of the quality of other system components. Secondly, it provides a measure of potential benefits from algorithmic advances. Lastly, both the average performance ratio as well as the worst-case performance ratio are of interest. Further, the ratio may be measured either experimentally or by analysis. It is worth noting that performance ratio can be computed without incurring the prohibitive effort of actually finding the optimal plan. Using an easy to compute lower bound on the optimal cost yields a pessimistic estimate of the performance ratio.

The second aspect of evaluating an algorithm is its running time. Since there is a tradeoff between the time spent in optimizing a query versus executing it, the evaluation criteria depend on the number of times the query will be executed. Two important cases are canned and interactive queries. A canned query is executed many times and an interactive query exactly once. Optimization time is not a major concern for canned queries. The tradeoff is thus important for interactive queries. Ideas such as *approximation schemes* in which better plans are obtained by expending more effort may be useful.

5.6 Cost Models

It is desirable to let users decide whether the cost of running a query is worth the benefit from the query result. This requires the ability to accurately predict query execution time. While this is a challenging problem even for sequential machines, factors such as data skew [36, 9] pose additional challenges for parallel machines. More work is needed to develop and validate accurate cost models.

Database systems are increasingly deployed in interactive systems where it is important to minimize the time to produce the first few tuples of the query result rather than the time to complete the query. This new optimization objective poses fresh challenges.

Two-phase optimization is useful to leverage the difficult problems but creates two cost models, each at a different level of abstraction. Unifying or ensuring the consistency of the two cost models is interesting. Validation of cost models is also hard as existing benchmarks do not deal with skew and parallelism. This is the more general problem of benchmarking parallel query optimizers.

Since accurate cost estimation is hard, the complementary approach of developing optimization techniques that compensate for the lack of knowledge by delaying decisions to runtime is a useful direction.

6 Conclusions

Parallel query optimization is a key technology that has already contributed to the success of parallel database systems. New requirements from applications, demands for higher performance from existing applications, innovations in query execution mechanisms and machine architectures are changing some of the underlying assumptions thereby offering new challenges.

In this paper, we have briefly introduced what we consider the major issues to be addressed: heterogeneous architectures, dynamic/pre-emptive optimization, space-time tradeoffs, new language features, evaluation of optimization algorithms, and accuracy of the cost model. Although all these issues can be addressed individually, they are not independent. To make substantial progress, it is important to build parallel query optimizers and stress them against real data and applications.

References

[1] T. Agerwala, J.L. Martin, J.H. Mirza, D.C. Sadler, D.M. Dias, and M. Snir. SP2 System Architecture. *IBM Systems Journal*, 34(2):152–184, 1995.

[2] P.M.G. Apers, A.R. Hevner, and S.B. Yao. Optimization Algorithms for Distributed Queries. *IEEE Transaction on Software Engineering*, 9(1), 1983.

[3] C.K. Baru, G. Fecteau, A. Goyal, H. Hsiao, A. Jhingran, S. Padmanabhan, G.P. Copeland, and W.G. Wilson. DB2 Parallel Edition. *IBM Systems Journal*, 34(2):292–322, 1995.

[4] B. Bergsten, M.Couprie, , and P. Valduriez. Prototyping DBS3, a Shared-Memory Parallel Database System. In *First International Conference on Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1991.

[5] H. Boral, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez. Prototyping Bubba, A Highly Parallel Database System. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), March 1990.

[6] L. Bouganim, D. Florescu, and P. Valduriez. Dynamic Load Balancing in Hierarchical Parallel Database Systems. In *Proceedings of the Twenty Second International Conference on Very Large Data Bases*, September 1996.

[7] F. Carino and P. Kostamaa. Exegesis of DBC/1012 and P-90 - Industrial Supercomputer Database Machines. In *Parallel Architectures and Languages Europe*, Paris, France, June 1992.

[8] C. Chekuri, W. Hasan, and R. Motwani. Scheduling Problems in Parallel Query Optimization. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1995.

[9] D. DeWitt, J. Naughton, D. Schneider, and S. Seshadri. Practical Skew Handling in Parallel Joins. In *Proceedings of the Eighteenth International Conference on Very Large Data Bases*, Vancouver, British Columbia, Canada, August 1992.

[10] D. J. DeWitt and J. Gray. Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM*, 35(6):85–98, June 1992.

[11] D.J. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H.-I. Hsiao, and R. Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), March 1990.

[12] S. Englert, R. Glasstone, and W. Hasan. Parallelism and its Price: A Case Study of NonStop SQL/MP. 1995. Sigmod Record, Dec 1995.

[13] D. Florescu, J-R. Gruser, M. Novak, P. Valduriez, and M. Ziane. Design and Implementation of Flora, A Language for Object Algebra. *Information Sciences*, 1995.

[14] S. Ganguly, W. Hasan, and R. Krishnamurthy. Query Optimization for Parallel Execution. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 9–18, June 1992.

[15] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, 1979.

[16] G. Graefe. Encapsulation of Parallelism in the Volcano Query Processing System. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, May 1990.

[17] G. Graefe and K. Ward. Dynamic Query Optimization Plans. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, May 1989.

[18] R.L Graham, E.L. Lawler, J.K. Lenstra, and A.H.G Rinnooy Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[19] L.M. Haas, J.C. Freytag, G.M. Lohman, and H. Pirahesh. Extensible Query Processing in Starburst. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, June 1989.

[20] A. Hameurlain and F. Morvan. Exploiting inter-operation parallelism for sql query optimization. In *Proceedings of the International Conference On Database and Expert Systems Applications*, Greece, September 1994.

[21] W. Hasan. *Optimization of SQL Queries for Parallel Machines*. PhD thesis, Stanford University, 1995. http://www-db.stanford.edu/pub/hasan/1995/thesis.ps.

[22] W. Hasan and R. Motwani. Optimization Algorithms for Exploiting the Parallelism-Communication Tradeoff in Pipelined Parallelism. In *Proceedings of the Twentieth International Conference on Very Large Data Bases*, pages 36–47, Santiago, Chile, September 1994.

[23] W. Hasan and R. Motwani. Coloring Away Communication in Parallel Query Optimization. In *Proceedings of the Twenty First International Conference on Very Large Data Bases*, Zurich, Switzerland, September 1995.

[24] W. Hong. *Parallel Query Processing Using Shared Memory Multiprocessors and Disk Arrays*. PhD thesis, University of California, Berkeley, August 1992.

[25] W. Hong and M. Stonebraker. Optimization of Parallel Query Execution Plans in XPRS. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, December 1991.

[26] Informix. INFORMIX-OnLine Extended Parallel Server for Loosely Coupled Cluster and Massively Parallel Processing Architectures, July 1995. http://www.informix.com.

[27] G. Lohman, C. Mohan, L. Haas, D. Daniels, B. Lindsay, P. Selinger, and P. Wilms. Query Processing in R*. In W. Kim, D. Reiner, and D. S. Batory, editors, *Query Processing in Database Systems*. Springer Verlag, 1985.

[28] C. Mohan, H. Pirahesh, W.G. Tang, and Y. Wang. Parallelism in Relational Database Management Systems. *IBM Systems Journal*, 33(2), 1994.

[29] Oracle. Oracle Parallel Server, 1995. http://www.oracle.com.

[30] M.T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, 1991.

[31] S. Roy. *Adaptive Methods in Parallel Databases*. PhD thesis, Stanford University, 1991. Stanford CS Report STAN-CS-91-1397.

[32] P. Valduriez S. Parker, E. Simon. SVP, a Data Model Capturing Sets, Streams and Parallelism. In *Proceedings of the Eighteenth International Conference on Very Large Data Bases*, Vancouver, British Columbia, Canada, August 1992.

[33] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An Economic Paradigm for Query Processing and Data Migration in Mariposa. In *Third International Conference on Parallel and Distributed Information Systems*, Austin, Texas, September 1994.

[34] Tandem. Query Processing Using NonStop SQL/MP, 1995. http://www.tandem.com.

[35] P. Valduriez. Parallel Database Systems: Open Problems and New Issues. *Distributed and Parallel Databases: An International Journal*, 1(2):137–165, April 1993.

[36] C.B. Walton, A.G. Dale, and R.M. Jenevein. A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, Barcelona, Spain, September 1991.