Macro- and Micro- Parallelism in a DBMS

Martin Kersten, Stefan Manegold, Peter Boncz, and Niels Nes

CWI, Kruislaan 413, 1098 SJ, Amsterdam, The Netherlands

Abstract. Large memories have become an affordable storage medium for databases involving hundreds of Gigabytes on multi-processor systems. In this short note, we review our research on building relational engines to exploit this major shift in hardware perspective. It illustrates that key design issues related to parallelism pose architectural problems at all levels of a system architecture and whose impact is not easily predictable. The sheer size/complexity of a relational DBMS and the sliding requirements of frontier applications are indicative that a substantial research agenda remains wide open.

1 Introduction

Database management systems have become a commodity system-software component to manage huge amounts of business data in a reliable and efficient manner. Its application space encompasses the whole spectrum of storage systems, ranging from databases fitting on a smart-card up to dealing with the petabyte archives produced in nuclear physics experiments. Likewise, it spans the complete space of responsiveness, from sub-second transaction management in telecom and financial sectors up to management of long-living transactions in aircraft construction.

This broad applicability and wide-spread use of a DBMSs make their design still an art, balancing end-user requirements with state-of-the-art software/hardware technology. The easy part is the functional requirement list. A new system should support an (object-)relational data model with its algebraic operators, transaction management and facilities to extend the system architecture with application-specific code.

The more difficult part is to predict the resources needed to manage the physical database itself and to predict hardware trends to satisfy these needs. For over two decades, commercial database systems have been designed from the assumption that a database is stored on disk with too little memory to keep a hot-set resident. Furthermore, they assume that the processing power offered by a single CPU is often insufficient to satisfy the application needs for cycles. Given manufacturing limitations to satisfy infinite memory and CPU power, a substantial effort has been devoted to realize parallel and distributed database technology.

1.1 Main-Memory Database Systems

Within the solution space for DBMS architectures we have focused our attention on systems with a sizable main-memory and deployment of parallel processing. The key assumptions underlying this part of the design space are that *the database hot-set can be economically held in main-memory*, *operating system technology will evolve and should be relied upon*, and *commodity hardware and parallel processing can be used to leverage the shortage of CPU power*. The two reference database architectures developed are: PRISMA and Monet.

PRISMA The PRISMA project (1986–1992) [3, 22] was a large national project geared at advancing computer science technology in the area of language design, operating system design, database system design, and network technology. The central theme was to develop a parallel object-oriented programming language with supportive operating system on which a complete SQL-based DBMS should run. The hardware platform consisted of 100 Motorola microprocessors with a total of 1.5 GByte main-memory. A sizable amount for the era it was constructed and used. The processors were linked into a configurable network to facilitate experimentation with network topologies. Each processor pair shared a disk (50GB) for persistence.

Monet The Monet project (1993–) was set-up to explore in more detail the effect of main-memory database technology. Furthermore, it took an offbeat approach in the design of the DBMS internals. The relational database tables were broken down into binary tables only, the relational algebra took a *materialize all intermediates* approach, indexing and operator optimization became automatic, and resource competition, such as transaction management, was moved to the intermediate language. Monet has been in production both experimentally and commercially since 1995. It runs under a variety of operating systems, i.e. NT, Linux, Solaris, IRIX, AIX. The largest experimentation platform is a SGI Origin 2000 with 32 CPUs and 64GB of RAM.

1.2 Parallel Database Technology

Both systems illustrate extreme approaches in terms of software architectures addressing the key design issues of a DBMS:

- *Persistent storage*, for which conventionally disk-farms and RAID systems are being used to secure the database content. Their capabilities define the bandwidth and latency to be expected when access randomly data.
- *Communication infrastructure*, which provides the framework to offload work over multiple systems. The full range of system architectures can be considered, e.g. shared-everything, shared-nothing, shared-disk, SIMD, etc..
- *Physical layout*, which encompasses the data structures to organize the records in memory and on persistent store, as well as the index structures needed to achieve reasonable performance.
- *Execution paradigm*, which dictates the way relational algebra expressions are being evaluated. The predominant approach is to use an operator pipeline,

which makes resource management of intermediate results relatively easy at the expense of repeated context switches amongst the operators. The alternative approach is to materialize the result of every operator before proceeding to the next level in the algebraic expression. The advantage is a simplified parallelism technique at the expense of possibly substantial storage overhead.
– *Transaction management*, which involves all techniques to support concurrent access to the database and to safeguard operations against accidental loss.
– *Query optimizer*, which involves the intelligence layer to derive a optimal plan of execution for a given database query. Query optimizers commonly generate a large portion of the semantic-equivalent formulations of a database query and select a good plan by running a cost-metric over each plan.

For a more detailed introduction to these topics we refer to the excellent textbook such as by Valduriez and Oszu [18] and the plethora of research results accessible through the digital library http://www.informatik.uni-trier.de/simley/db/index.html.

In the remainder of this note we illustrate the choices and the lessons learned from building and deployment of just two large-scale experimental systems.

2 Macro Parallelism

The PRISMA project [21] can be characterized as a system architecture geared at exploring opportunities for macro parallelism, i.e. the system used functional components and large storage fragments as the unit of distribution and parallel processing.

The relational tables were broken down into horizontal fragments using a hash-distribution over their primary keys. These fragments were distributed over the processor pool by the operating system, without any influence of the DBMS software. Each table fragment was controlled by a small-footprint relational algebra engine. SQL queries where translated into a distributed relational algebra expression, whose execution was handled by a distributed query scheduler. Likewise, a distributed concurrency manager was installed for transaction management.

The implementation language POOL (Parallel Object-oriented Language) [2, 1] was developed by research groups at Philips Natlab and universities. Together they realized a language-specific operating system and compiler toolkit for this language, with the ambition that parallelism can be transparently handled at those levels. It was the (contractual) target language for the database designers. The language implementation did not provide any control over the object granularity (initially). Rather, every object -small and large- was mapped into a separate process, which communicates with its environment through message passing. It was the task of the network infrastructure to resolve locality and a distributed scheduler ensured load distribution. Typically a small POOL program led to several thousands of objects distributed over the processor pool.

From the perspective of the database designers this level of transparency and object granularity caused major problems. Traversing simple binary search trees to organize information in the database became excessively expensive, e.g. memory references were cast into inter-process communication. Furthermore, the query optimizer of a DBMS is able to construct a proper execution plan, based on size, CPU, and communicating cost estimates. The underlying platform made this hardly useful, because the placement of intermediate results as well as the location of the operator threads was decided by the operating system without knowledge about the global (optimal) plan of action. Halfway of the project this had to be rectified using an advisory scheme to ensure locality of objects on a single processor and to identify where expensive message passing could be replaced by memory references.

Despite the limitations and problems caused by the implementation platform, the PRISMA project demonstrated that main-memory distributed database techniques are effective to speed-up performance on large processor clusters. Parallelism techniques were geared towards macro-entities in terms of size and granules of execution. Novel relational-join algorithms were discovered [20, 13] to achieve linear speed-up over >60 processors, schemes for dynamic query scheduling were developed [19], and progress was made in semantic query optimization and reliability.

3 Micro Parallelism

The most disappointing result of the PRISMA project was the in-surmounted problem to make macro-parallelism transparent at the system programming language interface. In addition, the software architecture of PRISMA/DB followed traditional lines, mimicking techniques nowadays common in commercial systems. As a result we started from scratch in 1993 with a new system architecture, nicknamed *Monet* [1].

The main design considerations underlying PRISMA/DB were carried over, i.e. the hot-set is memory resident, rely on the operating system, and move transaction decisions as high as possible in the application infrastructure. In addition, we envisioned a need to open up the kernel to accommodate better user-defined functions to support non-administrative applications. The most drastic steps where taken in the physical database design and the query execution paradigm. In combination with the coding style it lead to focusing on micro parallelism, i.e. dealing with critical instruction threads to extract the performance offered by a main-memory database on a multi-pipeline processor.

A long standing implementation approach is to physically cluster the fields of database records, to group those records into pages, and subsequently map them to segments (files or disk volumes). Instead, we took an orthogonal approach by breaking up relational tables into collections of binary tables. The underlying reason was that this way we simplified introduction of abstract data types, e.g. polygons, to organize their storage without concern on the total record layout. Furthermore, both columns of a binary table could be easily extended with search

[1] http://www.cwi.nl/~monet

accelerators. Such auxiliary structures were, however, constructed on the fly and basically never saved in persistent store.

Since the primary store for tables is main-memory, one doesn't have the luxury to permit sloppy programming without experiencing a major performance degradation. Sloppy programming in disk-based systems doesn't have that effect unless the system becomes CPU-bound. The consequence was that the database kernel algorithms were carefully engineered using code-expansion techniques to avoid type-analysis at the inner layers of the algorithms. For example, Monet version 4 contains about 55 implementations of the relational *select* operator, 149 for the unary operators, 355 for the relational *join* and *group* operations, and 72 for table aggregations.

The query execution paradigm was shifted towards concrete materialization of intermediates, thereby stepping away from the still predominant operator pipeline approach. The primary reason was to optimize the operators in isolation, i.e. each relational operator started out with just-in-time, cost-model based optimization decisions on the specific algorithm to use and beneficial search accelerators.

The primary interface to Monet became a textual-based Monet Interface Language [4]. This language is the target for both SQL-based and object-oriented front-ends [9, 11]. The overhead incurred by textual interaction with those front-ends was negligible, an observation shared nowadays in the community with the broad deployment of XML-based interaction schemes.

With a mature database kernel emerging from our lab in 1996 we embarked upon a series of performance studies. The first studies were aimed to assess the main-memory based approach against the traditional database techniques and its scalability beyond the main-memory size. To illustrate, in [9] we demonstrated the achievable performance on the TPC-D benchmark, and [5] illustrates that the engine could beneficially be used to support an object-oriented front-end. A short side-track assessed its performance capabilities as an active DBMS kernel [14].

3.1 Architecture-aware Optimization

Our recent studies showed that database systems — when designed and implemented "the traditional way" — do not lack CPU power [16, 15]. Actually, current database system are not even able to fully exploit the massive CPU power that nowadays super-scalar CPUs provide with their ever-rising clock speeds and inherent micro-parallelism. Rather, when executing database code, CPUs are stalled most of the time waiting for data to be brought in from main memory. While memory bandwidth has been improving reasonably (though not as rapidly as I/O-bandwidth or especially CPU speed) over the recent past, memory latency has stagnated or even got worse. As opposed to scientific programs, database operators usually create a random memory access pattern such as pointer-chasing. Hence, the performance is limited by latency rather than by bandwidth, making memory access the major performance bottleneck.

Choosing for vertically decomposed tables in Monet, was already a first step to improve memory performance as this avoids moving "superfluous" data around. In [6], we demonstrate, how properly designed cache-conscious algorithms can eliminate the memory access bottleneck by reducing random memory access to the fastest level of the systems cache memory hierarchy. Once memory access is optimized (i.e., the number of data cache misses is minimized), the costs of sloppy programming become obvious. The penalties for instruction cache misses and branch mispredictions — formerly "hidden" behind memory access — now dominate the performance. Using code-expansion techniques in Monet, we managed to eliminate both instruction cache misses (by having small-footprint operators) and most branch mispredictions (by avoiding type-dependent branching and/or function calls in the innermost loops) [16]. While now being "almost" CPU bound, the code is still not able to fully exploit the CPU inherent parallelism, e.g. moving from a 4 to a 9 instruction pipeline architecture did not significantly improve performance. We believe that there are three reasons for this. First, algorithm-inherent conditionals still require branches and hence keep the code "unpredictable" for compilers and the CPU itself. Second, loop bodies are too small to provide enough "meat" for parallelism. And third, the work to be done per data item is usually too small to keep more than one execution pipeline busy while loading the next data from memory.

3.2 3-Tier Query Optimization

Since 1999 we have shifted our focus on the middle-tier layer of a DBMS, i.e. its query optimizer infrastructure. The state-of-the-art in query optimization is for over a decade dictated by cost-based optimizers [17]. However, these model all assume that every query basically runs in isolation on a given platform, the database is in a cold-state, and that CPU- and I/O- activity can be analytically described. Although ideal assumptions for a laboratory setup and a sound basis to construct analytical models, it is far from reality.

Database systems are mostly called from applications that determine a context of interest, where queries are semantically related. Likewise, ad-hoc interactive session typically show quite an overlap amongst queries as the user successively refines it to locate the information of interest. Aside from personal use of a database, it is more common that at any time there are tens to hundreds of users interacting with a database, causing quite some overlap in interest and, indirectly, sharing/caching of critical table fragments.

To avoid inaccurate predictions and better exploit opportunities offered by a large query stream, we developed a novel just-in-time optimization scheme. Our hypothesis is that the optimization process can be split into a three-tier framework without jeopardizing the effectiveness of the optimization process. Instead of exploring a single huge search space in one optimization phase per query, we employ three smaller optimizers — *Strategic, Tactical*, and *Operational.*

The *strategic* optimizer exploits the application logic and data model semantic (e.g. rule constraints) for finding a good plan. At this level, the cost of a plan is only based on factors that are independent from the state of the DBMS, like intermediate result size and sort order, thus making volume reduction the prime optimization target. The plan generated by the strategic optimizer only describes the partial order amongst the database operators. The *tactical* optimizer consumes the query streams and minimize the resource requirements at runtime by exploiting the overlap encountered. It rewrites the plans such that (within the limits of the available resources, e.g. memory) the total execution time of all plans are minimized. The *operational* optimization works at operator evaluation time. The decisions taken focus on algorithm selection, which is based on the properties of their parameters and the *actual* state of the database. This technique is heavily exploited in Monet [4] already and proven effective in concrete real-life products.

This architecture is currently being build into Monet Version 5 and is expected to lead to a drastic speed-up for database intensive applications.

4 Applications

Development of novel database technology is greatly speed-up using concrete and challenging application areas. In the context of the PRISMA project this aspect was largely neglected, i.e. we were looking for a 'pure' parallel SQL database solution. During the development of Monet we used three major application areas to steer functionality and to assess its performance: geographical information systems, data mining, and image processing.

4.1 Geographical Information Systems

The first challenge for the Monet architecture was to support geographical applications [7]. Therefore, we realized a geographical extension module to implement the Sequoia benchmark [12]. This benchmark, developed at University of Berkeley, focuses on three aspects: user-defined functions, query performance, and scalability.

Its implementation pushed the development of extensibility features, such as an easy to use API generator for C/C++ based extension modules. Moreover, it demonstrated that many of the algorithms prevalent in this area could effectively use the restricted storage model and algebraic structure.

The query performance obtained was an order of magnitude better than reported by the competition. The queries ran in fractions of seconds without the need to rely on parallel processing at all.

The scalability aspect of the benchmark (>1GB) made full reliance on a database stored in main memory impossible. The approach taken in Monet to use memory-mapped files over binary tables turned out to be profitable. A surprise was that clustering data in main-memory still made a big difference, effectively killing the to date notion that access cost in main-memory can be considered uniform. A signal that cache-aware algorithms were needed even in a database management system.

4.2 Data Mining

The second challenge for the Monet architecture arose when it became the prime vehicle for data mining in a commercial setting. In 1995 we established the company Data Distilleries (www.datadistilleries.com), which developed and sold solutions for the analytical CRM space using Monet at its core.

The negative effect of this move for the Monet research team was that functionality and stability became somewhat dictated by the environment. In practical terms it meant that transaction processing features where put aside to focus on query-dominant environments. Moreover, the application interface language (MIL) was enhanced to simplify off-loading tasks to the database server. Finally, an extensive quality control system was set up to support overnight compilations and testing on all platforms supported.

On the positive side, early exposure to real end-user needs pushes the architecture to its limits. The database and query loads to be supported exploded in size, filling complete disk farms, and the functionality required for data mining called for extending the database kernel with efficient grouping and aggregation operations.

The effects of this engineering phase became visible in 1998, when we compared the behavior of Monet against Oracle. For this purpose we designed a benchmark reflecting the typical interaction between the data mining software and the database kernel. Subsequently we assessed the behavior of both systems in a memory-bound and disk-bound setting. For Monet it proved to provide the required supreme performance and also scaled beyond the main-memory limitations without additional tuning of the system (see [8]).

4.3 Image Processing

Given the good performance observed in both data mining and geographical information systems stressed our desire to find application domains that could both benefit from database technology and that posed real performance challenges. An area satisfying this criteria is image retrieval from multi-media databases.

In this context we are currently experimenting with indexing techniques based on traditional color-feature vectors scaled to handle over a million images. Moreover, we are looking for effective techniques that permit sub-image retrieval, an area often ignored in the image processing community. In [11, 10] we have demonstrated that such indices can be readily supported in Monet without impairing the performance.

Furthermore, we have introduced the notion of *query articulation*, the process to aid querying an image database using image spots marked by the user as relevant for retrieval. Finding such spots in the image collection is supported by the sub-image indexing scheme. Adequate performance will be obtained with the 3-tier query optimizer, because many interactions lead to overlapping subqueries.

5 Conclusions

In this short note we have covered fifteen years of research with a focus on database technology using large main-memories and parallelism at a macro and micro scale. The large body of expertise obtained [2] illustrate that to some extend we are confronted with a legacy problem. The commercial DBMS software architecture and its mapping to current hardware is far from optimal. Whether it is economical justified to replace them totally remains to be seen, but new markets may be conquered using database solutions currently available in laboratories only.

Recognition of this state of affairs provides a sufficient base to pursue system architecture research for DBMS and the applications it supports. The parallel research community can contribute to the evolution of database technology in many ways. To illustrate, just-in-time optimization and scheduling techniques at the high-end of the memory hierarchy should be improved. Likewise, cache-aware indexing schemes and relational operator algorithms may prove to form the basis for an order of magnitude performance improvement. But, the proof of the eating is in the pudding, being deployment of the solution in a concrete application setting with recognized benefit for the end-user.

References

1. P. America and J. J. M. M. Rutten. A Layered Semantics for a Parallel Object-Oriented Language. *Formal Aspects of Computing*, 4(4):376–408, 1992.
2. P. America and F. van der Linden. A Parallel Object-Oriented Language with Inheritance and Subtyping. In *Conference on Object-Oriented Programming Systems, Languages, and Applications / European Conference on Object-Oriented Programming (OOPSLA/ECOOP)*, pages 161–168, Ottawa, Canada, October 1990.
3. P. M. G. Apers, C. A. van den Berg, J. Flokstra, P. W. P. J. Grefen, M. L. Kersten, and A. N. Wilschut. PRISMA/DB: A Parallel Main Memory Relational DBMS. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 4(6):541–554, December 1992.
4. P. A. Boncz and M. L. Kersten. MIL Primitives for Querying a Fragmented World. *The VLDB Journal*, 8(2):101–119, October 1999.
5. P. A. Boncz, F. Kwakkel, and M. L. Kersten. High Performance Support for OO Traversals in Monet. In *Proceedings of the British National Conference on Databases (BNCOD)*, volume 1094 of *Lecture Notes in Computer Science*, pages 152–169, Edinburgh, United Kingdom, July 1996.
6. P. A. Boncz, S. Manegold, and M. L. Kersten. Database Architecture Optimized for the New Bottleneck: Memory Access. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 54–65, Edinburgh, United Kingdom, September 1999.
7. P. A. Boncz, W. Quak, and M. L. Kersten. Monet and its Geographical Extensions: a Novel Approach to High-Performance GIS Processing. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, volume 1057 of *Lecture Notes in Computer Science*, pages 147–166, Avignon, France, June 1996.

[2] see http://www.cwi.nl/htbin/ins1/publications

8. P. A. Boncz, T. Rühl, and F. Kwakkel. The Drill Down Benchmark. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 628–632, New York, NY, USA, August 1998.
9. P. A. Boncz, A. N. Wilschut, and M. L. Kersten. Flattening an Object Algebra to Provide Performance. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 568–577, Orlando, FL, USA, February 1998.
10. H. G. P. Bosch, A. P. de Vries, N. Nes, and M. L. Kersten. A case for Image Querying through Image Spots. In *Storage and Retrieval for Media Databases 2001*, volume 4315 of *Proceedings of SPIE*, pages 20–30, San Jose, CA, USA, January 2001.
11. H. G. P. Bosch, N. Nes, and M. L. Kersten. Navigating Through a Forest of Quad-Trees to Spot Images in a Database. Technical Report INS-R0007, CWI, Amsterdam, The Netherlands, February 2000.
12. J. Dozier, M. Stonebraker, and J. Frew. Sequoia 2000: A Next-Generation Information System for the Study of Global Change. In *Proceedings of the IEEE Symposium on Mass Storage Systems (MSS)*, pages 47–56, L'Annecy, France, June 1994.
13. M. A. W. Houtsma, A. N. Wilschut, and J. Flokstra. Implementation and Performance Evaluation of a Parallel Transitive Closure Algorithm on PRISMA/DB. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 206–217, Dublin, Ireland, August 1993.
14. M. L. Kersten. An Active Component for a Parallel Database Kernel. In *International Workshop on Rules in Database Systems (RIDS)*, number 985 in Lecture Notes in Computer Science, pages 277–291, Athens, Greece, September 1995.
15. S. Manegold, P. A. Boncz, and M. L. Kersten. Optimizing Database Architecture for the New Bottleneck: Memory Access. *The VLDB Journal*, 9(3):231–246, December 2000.
16. S. Manegold, P. A. Boncz, and M. L. Kersten. What happens during a Join? — Dissecting CPU and Memory Optimization Effects. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 339–350, Cairo, Egypt, September 2000.
17. S. Manegold, A. Pellenkoft, and M. L. Kersten. A Multi-Query Optimizer for Monet. In *Proceedings of the British National Conference on Databases (BNCOD)*, volume 1832 of *Lecture Notes in Computer Science*, pages 36–51, Exeter, United Kingdom, July 2000.
18. M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, Englewood Cliffs, NJ, USA, 1991.
19. C. A. van den Berg. *Dynamic query processing in a parallel object-oriented database system.* PhD thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, 1994.
20. A. N. Wilschut, J. Flokstra, and P. M. G. Apers. Parallel Evaluation of Multi-Join Queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 115–126, San Jose, CA, USA, May 1995.
21. A. N. Wilschut, J. Flokstra, and P. M. G. Apers. Parallelism in a Main-Memory DBMS: The performance of PRISMA/DB. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Vancouver, BC, Canada, 1995.
22. A. N. Wilschut, P. W. P. J. Grefen, P. M. G. Apers, and M. L. Kersten. Implementing PRISMA/DB in an OOPL. In *International Workshop on Database Machines (IWDM)*, pages 97–111, Deauville, France,, June 1989.