EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN/ECP 95-29
11 December 1995

ON-LINE EVENT RECONSTRUCTION USING A PARALLEL IN-MEMORY DATABASE

E. Argante[†,‡], P. v.d. Stok[†], I. Willers[‡]

[†]Eindhoven University of Technology
Dept. Mathematics and Computing Science
P.O.Box 513, 5600 MB Eindhoven
the Netherlands

[‡]CERN
div. ECP
CH-1211 Geneva 23
Switzerland

Abstract

PORS is a system designed for on-line event reconstruction in high energy physics (HEP) experiments. It uses the CPREAD reconstruction program. Central to the system is a parallel in-memory database which is used as communication medium between parallel workers. A farming control structure is implemented with PORS in a natural way. The database provides structured storage of data with a short life time. PORS serves as a case study for the construction of a methodology on how to apply parallel in-memory databases to HEP software, providing systematic structuring of HEP data, easier parallelization and consequently a simpler development and maintenance of code. PORS runs on a SPARCcenter 2000 8-node shared memory computer.

Presented at the First IEEE International Conference on Engineering of Complex Computer Systems, Ft. Lauderdale, Florida, November 6-10, 1995

On-line event reconstruction using a parallel in-memory database

E. Argante[†,‡] P. v.d. Stok[†] I. Willers[‡]

[†] *Eindhoven University of Technology*
Dept. Mathematics and Computing Science
P.O.Box 513, 5600 MB Eindhoven
the Netherlands

[‡] *CERN*
div. ECP
1211 Geneva 23
Switzerland

Abstract

PORS is a system designed for on-line event reconstruction in high energy physics (HEP) experiments. It uses the CPREAD reconstruction program. Central to the system is a parallel in-memory database which is used as communication medium between parallel workers. A farming control structure is implemented with PORS in a natural way. The database provides structured storage of data with a short life time. PORS serves as a case study for the construction of a methodology on how to apply parallel in-memory databases to HEP software, providing systematic structuring of HEP data, easier parallelization and consequently a simpler development and maintenance of code. PORS runs on a SPARCcenter 2000 8-node shared memory computer.

1 Introduction

Parallel in-memory databases offer a powerful means of communication between the parallel workers of an application, while providing high performance through parallelization. In addition, they facilitate the structuring of data, and allow a separation of logical data presentation from physical data representation. The abstraction from physical data representation and low-level communication primitives makes parallel software less platform dependent. This paper shows that the application of database concepts can lead to logically structured software. It is argued that parallel in-memory databases are a viable alternative to current non-database software development techniques.

PORS (parallel on-line reconstruction system) is a case study which is designed to perform on-line event reconstruction using the CPREAD reconstruction program of the CPLEAR experiment at CERN, the European Laboratory for Particle Physics in Geneva, Switzerland. This is accomplished using a parallel in-memory database.

Event reconstruction is the process of converting raw detector data into interpretable physics results. Traditionally, event reconstruction is performed off-line, i.e. separate from the experiment. PORS is designed to perform on-line event reconstruction, i.e. as the experiment is running. Former attempts to apply databases in on-line HEP applications failed due to a lack of performance.

2 HEP software problems and solutions

Need for high performance. For new HEP applications and in particular the new Large Hadron Collider (LHC) experiments at CERN, the estimated increase of computing power requirements for the coming ten years is three orders of magnitude [1]. Parallel processing will become an essential way to tackle this problem. This especially holds for real-time applications where computational latency is important.

The scalable performance of PORS permits calculation and storage of reconstructed HEP data on-line.

No uniform way of structuring data. HEP data are often stored in an unstructured or not uniformly structured way. This holds for run-time data as well as more permanently stored data. Data are often viewed and treated as binary objects without any defined structure. Permanent data are stored on sequential media, most commonly magnetic tape.

Data formats can be specific to an institute or even to a HEP experiment. An example is Zebra [2], a library to extend Fortran with more elaborate data structure facilities. Also *within* Zebra, there is a lack of enforced consistency. There is no general method on how to model the data.

Databases help to enforce structuring of data according to well-defined rules. Real-world objects can be represented by tables. The columns of a table represent features of the object. Relations between ob-

jects are represented by columns or by separate tables. The table structuring rules (normalization) lead to a commonly understood data organization.

Hardware dependence of high performance parallel programs. A problem in parallel software is that to obtain good performance, the machine architecture has to be taken into account [3], [4], forcing applications to become machine dependent. Currently available communication paradigms (PVM [5] or MPI), are still not capable of combining a good abstraction level with high performance [4]. Aiming for the highest performance reduces the software's portability. Tuple space as defined by LINDA [6] abstracts completely from the underlying communication structure. However, the broadcasting of tuple space contents to all participants can lead to considerable communication overhead. PORS strikes a balance between LINDA and PVM by abstracting from the communication structure of the platform but allowing access to specified data items via SQL statements.

The PORS structure allows the easy implementation of rather complex control structures. Example are:

- Data driven farming. A worker specifies via SQL statements that it needs certain data from the database. At the arrival of the data the worker continues execution. The provision of data with a sufficient high rate leads to a high CPU utilization.
- Buffering. PORS provides flexible buffering which is easy to set-up and changeable at runtime. For example by specifying an appropriate data distribution on a distributed memory machine, data can be sent in advance instead of waiting until the data are needed. By using PORS, this can be accomplished by specifying the location of a table.
- Monitoring. Simple queries can be used to monitor the results of the parallel software. Queries can be used to change parameters in the system if a deficiency is monitored.

Parallelizing existing software. Currently, many companies have sequential software which is not parallelized, since they foresee that the gain does not compensate for the work and difficulties experienced in parallelization [3]. In HEP, farming of the sequential program is a solution which may not always be satisfactory, for example if computational latency is important.

PORS provides a smooth transition from sequential to parallel code. In the case study, this is accomplished by replacing the I/O statements by database statements. For more elaborate ways of coarse grain parallelism, like pipe-lining, the interface routines between the modules of the sequential program are replaced by database access routines. Data can be shared between multiple workers instead of every worker having its own local copy. Concurrency control mechanisms of the database take care that data consistency is preserved. This offers an easy way to obtain data parallelism.

Conforming to industrial standards. The commercial Oracle database can be used to store data with a long lifetime. By integrating a commercial database like Oracle into the system, there are a lot of additional advantages: structured storage of large volumes of permanent data; conformance to industrial standards; data representation to the user is platform independent; network access among different platforms; it offers tools (e.g. graphics or statistics tools) which are directly applicable to the data

3 Parallel in-memory databases

The purpose of a database is to store and retrieve data efficiently and conveniently. In a relational database, a collection of tables is used to represent data and relationships among the data. The SQL query language offers four types of access routines to retrieve information from a database and to change its contents: select, delete, insert and update. Indices are used to decrease search times through database tables. Logically associated access routines are gathered in transactions. A transaction is the atomic unit of database actions.

A parallel database allows concurrent access. Parallel transactions should not be allowed to destroy database consistency. Concurrency control algorithms preserve database consistency in the context of parallel transactions [7].

An application using the database, like PORS, consists of multiple workers accessing the database, each performing a task. The order in which data are accessed by an application is called its data access pattern. By examining this data access pattern of a worker, a set of transactions is designed to retrieve the required data. The set of all transactions of all workers of an application is called the transaction scheme. The transaction scheme determines the lay-out of the in-memory database, i.e. table definitions, data distribution and worker distribution.

Transactions of different workers may be executed in parallel, possibly resulting in data conflicts. For best performance, these data conflicts are evaluated and an appropriate efficient type of concurrency control is chosen to preserve data consistency, i.e. transaction scheme and type of concurrency control are optimized looking at the data access pattern of the application. This ensures that the overhead of concurrency control will be low, since it does not have to deal with all possible data access patterns. The steps described above constitute an important design strategy to support the high performance of the database. It is intended to distinguish classes of applications each for which a suitable type of concurrency control exists.

4 Case study: on-line event reconstruction for CPLEAR

In the case study (described in detail in [8]), CPREAD is integrated with the in-memory database to obtain event reconstruction with on-line performance. The case study comprises the design and implementation of the database and the integration of the application with the database. It should be noticed that the only CPREAD dependent part is the changeable concurrency control mechanism.

CPLEAR & CPREAD. CPLEAR is a HEP experiment at CERN which investigates the C-P violation phenomenon. The experiment has run for several years and about 100 people are involved. CPREAD is a 260k lines Fortran source code program which is used to reconstruct events produced by CPLEAR. About 100 GBytes of data have to be reconstructed annually. At run-time, the program size is 13 MBytes. CPREAD is often changed and adapted.

Description of the system in operation. Figure 1 shows the system implemented on an 8-node shared memory computer. The dotted box comprises the database. The database together with the database library form the reusable part of the system.

The CPLEAR event generator is a source of CPLEAR event data. It inserts Zebra [2] blocks with raw (i.e. non-reconstructed) events into a table of the in-memory database, at a specific rate. CPREAD workers retrieve Zebra blocks with raw events from this table via database routines. A worker reconstructs the events and reconstructed valid events together with reconstruction information are inserted into another table. Writers retrieve accepted events from this table to write them to permanent storage.

In the CPLEAR case study the transaction scheme provides data driven farming (a worker asks for data when it is ready to process data). There is no central master in the system controlling the placement of data. Transport of data to the right worker is done via the transaction scheme. Every worker takes its own decisions. The absence of a master enhances the system's scalability.

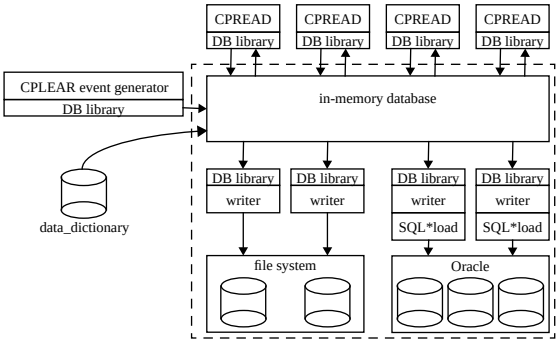

Figure 1: On-line event reconstruction for CPLEAR using a parallel in-memory database

Permanent storage can be a file system on hard disk or the Oracle 7.1 parallel database. In the case of the Oracle database, the SQL*loader tool is used to insert data into Oracle at high rate. Only data with a long lifetime, i.e. reconstructed events, are stored on permanent storage.

At run-time, workers can be added or removed from the system, i.e. the farm size can be changed dynamically. This does not lead to any data inconsistencies or data loss. It is a standard feature of PORS: PORS allows workers to connect or disconnect at run-time. A library (DB library) linked to all worker processes accessing the in-memory database connects the worker to the database, and provides database access routines and concurrency control. A data definition language is used to define the lay-out of the database.

Hardware. The system is implemented on a SPARCcenter 2000 shared memory computer. It has 512 MBytes memory and eight 40 MHz processors with 2 MBytes cache each.

Performance. Processes concurrently accessing the same table can introduce waits. Database overhead consists of executing access routines and maintaining database structures. Measurement of waiting times and overhead shows database performance.

The event reconstruction rate as function of the number of CPREAD workers is measured. 8 CPREAD workers gave the highest rate. A standard stand-alone CPREAD worker, i.e. without using the database, can reconstruct events at a rate of 15.5 Hz on the SPARCcenter computer (in this case it uses one processor). So assuming that full scalability were possible, 8 processors would process events at 124 Hz. The actual system does process events at 117 Hz with 8 processors. So 7 Hz are lost by the overhead caused by the database. Reasons for the decreased performance in comparison to the theoretical case are overhead generated by the event generator and the memory bandwidth of the shared memory machine.

Some typical execution times for access routines during run-time of PORS are provided. Actions are done on tables containing about 100 rows; the timings are averages with accuracy of two digits. Data copying is not included: select: 1 μs; delete: 7 μs; update: 12 μs; insert: 18 μs.

5 Conclusions

The case study showed that data driven farming with a changing farm size was efficiently realized thanks to the database concepts. Performance turned out to be quite scalable. Database overhead turns out to be small. Properties of the SPARCcenter machine turn out to be the predominant performance limiting factors.

At present, the system is not connected to the CPLEAR experiment to perform on-line event reconstruction. Necessary adaptations are the construction of an interface between experiment and system, and an increase of the permanent storage size.

6 Future work

The case study left many features of the database unused. Track fitting, a part of the reconstruction process, will be parallelized. Communication between the workers will be done via the database. The increased parallelism should provide a reduced latency and less memory usage. The reduced memory usage comes from reduced code replication. It is a heavy test for the database since the number of communications increases and the size of the communicated data packets decreases. It tests whether the database approach facilitates the parallelization of existing software.

The system will be ported to a Meiko CS-2 distributed memory computer to investigate the influence of different parallel machine architectures on the performance of PORS [9]. Data distribution will become an important issue.

Acknowledgements

Thanks to Marcel Meesters and Bob Dobinson from CERN for their help, and CPLEAR for giving access to the CPREAD program.

References

[1] Proc. of the 8th Conference in the series *Computing in high energy physics*, Santa Fe, USA, 1990

[2] *Zebra; an overview of the Zebra system*, CERN, Geneva, Switzerland, 1994

[3] W.F. McColl, *General purpose computing, Lec. on parallel computation*, proc. 1991, ALCOM spring school on parallel computation, Cambridge, UK

[4] J.J. Lukkien, *The Construction of a Small Communication Library*, Comp. science report, Eindhoven University of Technology, the Netherlands, 1995

[5] A. Geist et al., *PVM 3 User's Guide and reference manual*, Oak Ridge National Laboratory, 1993.

[6] R. Bjornson, N. Carriero, D. Gelernter, and J. Leichter,*Linda, the Portable Parallel*, Technical Report 520, Yale University Department of Computer Science, Jan. 1988

[7] Henry F. Korth and Abraham Silberschatz, *Database system concepts*, McGraw-Hill, 1991

[8] Erco Argante, *Real-time database access on a MPP*, Thesis of the Software Technology Programme, Eindhoven University of Technology, the Netherlands, 1994

[9] E. Argante, M.R.J. Meesters, I. Willers, P. van der Stok, *A database for on-line event analysis on a distributed memory machine*, Eindhoven University of Technology, the Netherlands and CERN, Geneva, Switzerland, 1995