EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN/ECP 95-28
11 December 1995

A DATABASE FOR ON-LINE EVENT ANALYSIS ON A DISTRIBUTED MEMORY MACHINE

E. Argante[†,‡], M.R.J. Meesters[†,‡], P. van der 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

Parallel in-memory databases can enhance the structuring and parallelization of programs used in High Energy Physics (HEP). Efficient database access routines are used as communication primitives which hide the communication topology in contrast to the more explicit communications like PVM or MPI. A parallel in-memory database, called SPIDER, has been implemented on a 32 node Meiko CS-2 distributed memory machine. The SPIDER primitives generate a lower overhead than the one generated by PVM or MPI. The event reconstruction program, CPREAD, of the CPLEAR experiment, has been used as a test case. Performance measurements showed that CPREAD interfaced to SPIDER can easily cope with the event rate generated by CPLEAR.

Presented at Computing in High Energy Physics '95
Rio de Janeiro, Brazil, September 18-22, 1995

A DATABASE FOR ON-LINE EVENT ANALYSIS ON A DISTRIBUTED MEMORY MACHINE

E. Argante[†,$], M.R.J. Meesters[†,$], P. van der Stok[$], I. Willers[†]

[†] *CERN, div. ECP,*
1211 Geneva 23, Switzerland

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

Parallel in-memory databases can enhance the structuring and parallelization of programs used in High Energy Physics (HEP). Efficient database access routines are used as communication primitives which hide the communication topology in contrast to the more explicit communications like PVM or MPI. A parallel in-memory database, called SPIDER, has been implemented on a 32 node Meiko CS-2 distributed memory machine. The SPIDER primitives generate a lower overhead than the one generated by PVM or MPI. The event reconstruction program, CPREAD, of the CPLEAR experiment, has been used as a test case. Performance measurements showed that CPREAD interfaced to SPIDER can easily cope with the event rate generated by CPLEAR.

1 Introduction

To meet ever increasing demands for computing capacity —the expected increase is more than a factor 1000 for the coming ten years— parallel computing will become the most cost effective solution[3]. Parallel in-memory databases can enhance the process of parallelization. Communication is transparent to the user: access routines, used for storage and retrieval of data, take care of communication implicitly.

Our objective is to investigate how database techniques can be applied to improve High Energy Physics (HEP) software. In particular, we aim to simplify the process of parallelization of software and to enhance the structuring of data. Also, we aim to increase the portability of user programs. Furthermore, event reconstruction programs must be able to run on-line, i.e. event reconstruction must keep up with the frequency at which events are generated by the experiment.

Farming of the sequential program is the obvious choice to increase the event reconstruction frequency. Latency reduction of the event reconstruction by parallelizing the program is interesting for other reasons: (1) reduction of data storage and (2) faster feed back on experiment progress. The decreased latency of a simplified reconstruction program makes it possible to insert these programs into the triggers to increase their discriminating power. This diminishes the need for the storage of raw data. A low latency makes it possible to detect changes in the accuracy of the measurements sufficiently quickly to adapt the calibration constants of the reconstruction software on an event to event base.

In addition, a homogeneous structure of the HEP programs composed of relatively independent parts allows us to postpone the decision concerning which computer platform to use to satisfy the real-time performance requirements.

The huge volume of data and their complex relations make database concepts well suited to solve the data structuring problem. However, the low performance of databases seemed to exclude this approach until now. In this paper it is shown that a parallel in-memory database can cope very well with the expected event rates.

SPIDER, Scalable, Parallel In-memory Database for Event Reconstruction, has been designed and implemented to be used for HEP software, in particular for event reconstruction programs. As a test case, CPREAD was interfaced with SPIDER. CPREAD is the event reconstruction program of CPLEAR, a CERN experiment. SPIDER is designed to operate on a distributed memory machine, and implemented on a MEIKO CS-2, a 32 node machine, each node containing two 100 MHz SPARC processors.

2 Parallel In-Memory Databases for a Distributed Memory Machine

Databases are designed to manage large bodies of information. A relational database consists of a collection of tables. Every table has a number of rows and columns. This allows modeling of real world objects and their relations. Database access is established via *select*, *insert*, and *delete* operations. The *select* operation retrieves rows that satisfy a given predicate. The *insert* and *delete* operations store and remove rows from a table respectively.

In an in-memory database, the data reside in real-memory in contrast with a disk-based database where the data reside on hard-disks. In-memory storage allows faster access of data than disk-based access. However, in-memory storage capacity is more expensive than disk storage capacity. For the CS-2, total in-memory storage capacity adds up to 128 MBytes $\times$ 32 Nodes = 4.1 GBytes.

Parallel databases allow concurrent access, i.e. operations can be executed in parallel. We distinguish intra and inter access parallelism. The former allows a single operation to be executed on more than one processor, the latter allows several operations to be executed concurrently.

Access to data on a distributed memory machine can be characterized as NUMA: Non Uniform Memory Access. Data accessed in the local memory of a CPU can be accessed in shorter time than data accessed in memory of another CPU. Typical times for a *select* of 23 KBytes on the CS-2 are 3 μs for a local select (no copy performed and data present in cache), 950 μs on another CPU's memory and 5 ms when read from disk.

Processes, together comprising a HEP program, can be interfaced to SPIDER. To obtain high performance, these processes must be balanced over the processors, such that all processors have an equal load. Due to the NUMA effect, processes should access most frequently accessed data locally. SPIDER permits the distribution of complete rows of a given table over several nodes. The user can specify the allocation of processes and rows to nodes via a Data Definition Language (DDL).

3 How can HEP Software Benefit from SPIDER?

3.1 Enhance Parallelization

Communication in SPIDER is established via data access routines. Their high abstraction level eases load balancing and the communication deadlock problem. With communication paradigms like PVM or MPI, communication is established via message passing. The simplest is node to node communication: via a *send* on one node, and a *receive* on the other. Destination and source nodes must be specified in the *send* and *receive* statements. Therefore, the communication pattern has to be known. During the design a program is divided into processes which can be distributed over the processors. Insight is required in the time consumption of the processes to achieve load balancing. In contrast, SPIDER operations like *select* and *insert* can be used in these processes, instead of communication primitives. In the SPIDER operations no reference to the location of the data need to be specified. After all processes have been designed, their communication pattern is determined by binding the tables and processes to nodes in the DDL file. Thus, the design of the processes and the communications between them can be separated, which eases load balancing. The user does not explicitly address the communication between the processes. Communication still takes place between processes but is completely handled by SPIDER. In the implementation of SPIDER, it is assured that every node contains a process that is always willing to receive data from any other nodes. Consequently, the user only needs to handle deadlock on the level of his algorithm, knowing that the database primitives themselves do not deadlock.

Data consistency is also no user worry; via locking mechanisms concurrent access is controlled. All locking is performed locally: no communication is needed.

3.2 Enhance Structuring of Data

Data formats for structuring HEP data are often institute or experiment specific. An example is Zebra[2], a library which offers an extension to Fortran data structures. No uniform method exists on modeling HEP data in Zebra.

Databases allow structuring of HEP data[9]: physical objects can be represented by means of tables. The columns in tables are used to represent properties of these objects. Relations between objects can be represented by extra columns or tables. Many methods, like the E-R model[5], are available to structure data for databases. SPIDER can be used along with these structuring methods.

3.3 High Performance

SPIDER differs from conventional databases in its simplicity: it contains minimal data security, concurrency control and fault tolerance. Relational database operations like joins, views and support for transaction processing are kept minimal. For use in on-line event reconstruction programs these features did not prove to be necessary. On the other hand, a few database operations are added which speed up programs considerably. For example, it offers the operation *select_and_delete*, which has the same functionality as the sequence of a *select* and a *delete* operation.

Another difference to conventional databases is the choice to store frequently used data in-memory rather than on disk.

To be effectively used for parallel computers, the database must be scalable. The SPIDER test cases showed that performance increases linearly with the number of nodes. To ensure this, SPIDER does not have centralized control: the database is distributed over all nodes. Each node decides where to access data.

3.4 Hardware Independence

HEP software programs normally last for many years, much longer than the computers they run on. Usage of low level communication libraries would increase performance at the expense of the portability of these programs.

When ported to another computer, only the communication within SPIDER has to be changed; the user-programs can remain the same. The communication primitives needed to build SPIDER are the node to node communications *send*, *receive* and a node to node synchronisation primitive *signal*. Currently, the manufacturer specific Elan Widget communication library is used. This results in fast communications, at the cost of reduced portability of SPIDER.

4 Case Study: On-Line Event Reconstruction

To test how SPIDER performs when interfaced to HEP programs, we interfaced it with CPREAD, the event reconstruction program of the CPLEAR experiment. CPLEAR investigates the CP violation phenomenon. CPREAD is a 200k line Fortran source code program, with which about 100 GBytes of data are reconstructed annually. At run-time, the program size is 13 MBytes.

A first configuration that we tested was a farm of CPREAD instances (see left side figure 1). Every node contained a CPREAD instance. The input of an instance were "raw" events, the output were the reconstructed events. To measure performance, we constructed a process (called REG) that inserted raw events into the database with an adjustable frequency. For permanent storage, interesting reconstructed events can be written to disk.

This test case uses SPIDER in a simple form: the REG inserts rows in table 0, each row containing a raw event. Each CPREAD instance performs a *select_and_delete* on a row from table 0 (a raw event), reconstructs the event and inserts the reconstructed event as a row in table 1. In the DDL, it is specified that every node running a CPREAD instance contains a part of table 0 and table 1. The REG distributes the rows over the different parts of table 0 in a round robin fashion: rows are distributed in cyclic order (see figure 1). If the *select* on the local part of the table fails (e.g. an empty table), the *select* is communicated to other nodes.

The second configuration is designed to investigate the ease of parallelizing existing programs when interfaced to SPIDER, see right side of figure 1. A parallelization of a part of CPREAD on sub-event level was used: track fitting[10]. Since every event contains 2 to 6 charged tracks, each of 2 to 6 nodes can calculate one track. Input and output of the track fitting processes is done via *select_and_delete*

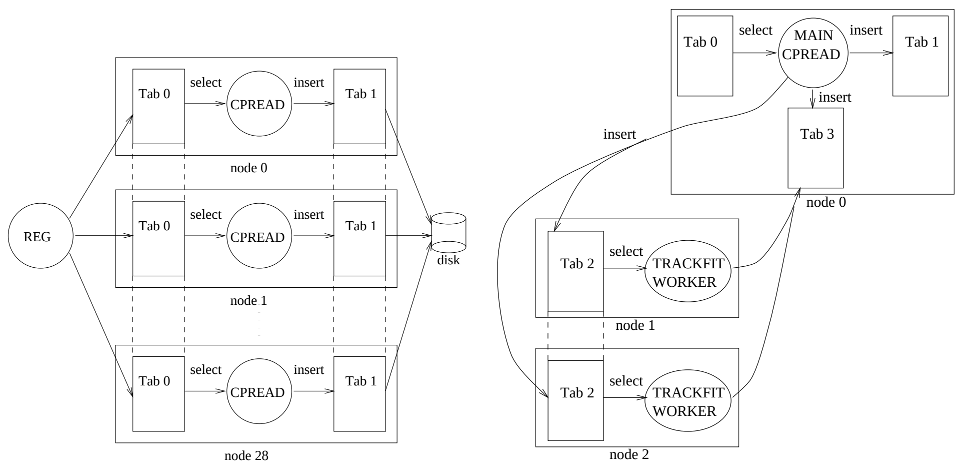

Figure 1: Select and insert by CPREAD and REG (left) and parallel track fitting (right)

and *insert* operations. Therefore, two extra tables are specified, table 2 for "raw tracks" and table 3 for "fitted tracks". Advantages of this type of parallelization are reduced latency and less memory usage, since each node does not need to contain all CPREAD code, but only a part. This is especially relevant for large programs like CPREAD.

4.1 Measurements

Here we describe measurements on the farm of CPREAD instances. We used an input file of 3000 raw events, generated by the CPLEAR experiment. For the CPREAD farm case, the system (operating on 28 nodes) reconstructed events with a rate of 560 Hz. A sequential version without the database operating on the same input file achieved a rate of 23 Hz on one node. This means that the theoretical limit is $23 \times 28 = 644$ Hz. Thus, we loose 84 Hz due to communication and database overhead. Notice that if only one node is used, CPREAD *with* the database performs at a rate of 22Hz, giving a limit of $22 \times 28 = 616$ Hz. This implies that $644 - 616 = 28$ Hz is lost due to overhead of the database, and $616 - 560 = 56$ Hz is lost due to communication overhead and memory access contention.

The measured event reconstruction rate of 560 Hz allows this system to run on-line since the event generation rate of CPLEAR varies between 200 and 500 Hz.

REMARK: At this moment every node can have only one CPREAD instance (this is a limitation of the current implementation). The reconstruction rate will increase considerably when two instances per node (one instance for every CPU) can be scheduled. More results are given in [7].

5 Summary and Conclusions

Compared with other communication paradigms, SPIDER communications are faster than PVM[4] and about the same as MPI[8] on the CS-2. For example, a *select* which retrieves a tuple of 1 KBytes from another node takes 135 μs with SPIDER. With PVM and MPI, a communication of 1 KBytes takes 170 and 140 μs respectively. The main disadvantage of PVM and MPI is that they offer less abstraction: despite their large communication possibilities they are essentially message passing paradigms.

Compared with other structuring facilities, like Zebra, which are used at CERN, the structuring methods offered by database theory are more standardized and are more widely used and researched.

Compared with conventional databases, SPIDER has less overhead, due to minimized security, concurrency control, and fault tolerance. Furthermore, it is faster since it is memory based rather than disk based.

Parallel in-memory databases like SPIDER enhance the process of parallelizing HEP software, which is shown with the parallel track fitting case. SPIDER combines functionalities normally offered by a diversity of tools, such as data structuring tools, communication paradigms and data retrieval tools.

The above mentioned advantages of SPIDER are offered without significant loss of performance. As our test case has shown, SPIDER even offers the opportunity to perform event reconstruction on-line for experiments like CPLEAR.

We wish to thank the CPLEAR experiment for 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. CN, ECP and PPE divisions *Zebra; an overview of the Zebra system*, CERN, Switzerland (1994).
3. W.F. McColl, *General purpose computing*, Lectures on parallel computation, proceedings 1991 ALCOM, Cambridge, UK (1991).
4. A. Geist et al, *PVM 3 User's guide and reference manual*, Oak Ridge National Laboratory (1993).
5. H.F. Korth et al, *Database system concepts*, Mc-Graw-Hill (1991).
6. J.J. Lukkien, *The Construction of a Small Communication Library*, Comp. science report, Eindhoven, the Netherlands (1995).
7. M.R.J. Meesters, *On-line event reconstruction using a parallel in-memory database*, Thesis of the Softw. Techn. Prog., Eindhoven, the Netherlands (1995).
8. P.S. Pacheco, *A users guide to MPI*, Univ. of San Francisco, CA, USA (1995).
9. Programming techniques Group, ECP, *ADAMO Entity-Relationship Programming System, Version 3.3* Cern, Geneva, Switzerland (http://www1.cern.ch/Adamo/ADAMO_ENTRY.html) (1993)
10. R. Schiefer *Parallel track fitting in CP-READ*, CERN, Geneva, Switzerland (1995).