The PRISMA Experiment

Peter M.G. Apers  L.O. Hertzberger  Ben J.A. Hulshof
University of Twente  University of Amsterdam  Philips Research Laboratories

Abstract

The design of a large database management system on top of an object-oriented platform which is implemented on a tightly coupled multiprocessor system requires a large number of design decisions. There exist no methodology to evaluate the consequences and compare the differences between them. Looking into experimental methodology in physics we find a lot of similarities and techniques we think that are useful to solve our problems. A discussion is presented in which way such experiments have to be set up and of the tools that are required. The status of the project and our experience up till now is presented.

1 Introduction

In the last five years the possibility to build parallel computers has become more and more appealing. On the one hand, switching speed of electronic components has increased to such a height that the speed of light poses severe limitations on the physical length of interconnections in a computer. On the other hand, integration technology of semiconductors will soon reach a point where a complex conventional computer architecture only occupies a fraction of the effective space on a silicon chip. One of the possibilities to fill up the available space on a silicon chip is to design regular parallel computers, where a basic processor and communication design can be copied as often as required.

Although the hardware possibilities are there to build a parallel computer system, the models and the experience to effectively exploit parallelism in computer systems is still to a great extent missing. The Japanese Fifth Generation Computer System Project [1] had a large influence convincing the computer science community that parallel systems had a great potential. This project was aiming at exploiting implicit, more fine-grained parallelism, as was believed to be present in a language as Prolog.

The ESPRIT project 415 [2], headed by Philips Research Laboratory in Eindhoven, also had as its target designing a parallel computer, however, its aim was to exploit explicit, more coarse-grain parallelism as present in application programs. For that purpose a language called Parallel Object-Oriented Language [3] was designed. With the help of this language the user can structure his application programs as a collection of processes (objects), which can only interact explicitly via the exchange of messages. The combination of parallel machine and language provided an object-oriented platform called Distributed Object-Oriented Machine (DOOM) [4].

It was felt that to gain more experience with explicit parallelism, it was necessary to construct large applications. Therefore the PRISMA project [5] was started as a collaboration between Philips Research Laboratory in Eindhoven and a number of Computer Science Departments at Dutch universities and the CWI, the Centre for Mathematics and Computer Science. Although targetting to design a Database and Knowledgebase machine, it became clear in the course of the project that only parallelism in databases was understood sufficiently well to be able to design a parallel implementation.

This paper describes the evolution in our approach and way of thinking about parallelism during the course of the four year PRISMA project. More specifically, attention will be paid to how we think

The PRISMA project is supported by the Dutch "Stimuleringsprojectteam Informaticaonderzoek" (SPIN)
that a better understanding of the exploitation of parallelism in large application can be obtained and what methodologies are necessary. In this discussion the experience gained from the database application on top of an object-oriented platform will serve as a guide-line.

2 Project evolution and constraints

In this section it will be illustrated how our ideas of the project changed over time and what the constraints were the project had to deal with.

When the PRISMA project started the aim was to design a high performance database machine with a maximum of functionality. The major reason for this was, that it was expected that one of the spin-offs of the project would be a commercial product.

The main aim to realize the high performance was to exploit parallelism where and whenever possible. To make optimal use of current technology, the main memory approach was chosen. The further the project evolved, the more we found that the demands of functionality and speed could not be realized in the given time frame. Because of this, and because it was realized that there was not sufficient experience with such highly parallel applications as a database management system to justify our design decisions, we realized that a methodology was required that allowed us to compare and evaluate these decisions. Such a methodology should allow to compare the consequences of different forms and implementations of parallelism in the application. Looking around into other fields of science, like physics and chemistry, we came to the conclusion that an experimental approach (see Section 3) was the best suited methodology.

Before discussing this approach any further, it is important to analyse the consequences of the fact that PRISMA was based on the DOOM project. In the DOOM project the target had been to design a flexible and extensible machine. This had resulted in the choice for a distributed memory machine with a separate Communication Processors (CP) [6] responsible for communication and message routing. This CP was necessary to discharge the Data Processor (DP) from the abundance of message passing as generated by the object-oriented language. When we started, we decided to use the same hardware as the DOOM project and also to make use of the object-oriented platform as provided by POOL where and whenever possible. This was one of the arguments why the total database management system was coded in the POOL language. An additional goal was to evaluate the object-oriented programming style as a rapid prototyping facility.

For the PRISMA project, DOOM mainly set constraints on the language (object-oriented language) and machine level (distributed memory). In addition it was decided to implement issues that are to a large extend dictated by the application (the database system) like memory management, stable storage, etc via the flexible but slower software solutions and not by an inflexible hardware approach. Although we realized that this would imply speed penalties, it offers the flexibility to experiment, also at this level, with different solutions.

3 Why an experimentalist approach

In computer science for a long time the habit has been to design a model or a small experiment (for instance based on Fibonacci numbers) to present this model or some results of an experiment and thereafter conclude that a new idea would work far better and consequently emerge on a completely different track. Such an approach has the following disadvantages:

- there is no continuity built up in a certain research direction.
- it becomes almost impossible to compare results obtained from similar research because results are not obtained with similar techniques or under comparable assumptions.

At this moment it can be observed that a number of research groups have left that track and are looking for a more solid basis to evaluate and compare research results.
We found that research in parallel computing is one of these areas demanding for an approach allowing to better compare results obtained at different locations. Now that the field has left the stage where an elegant theoretical model of concurrency or a complicated design of a network switch will convince the scientific community that parallel computing is an attractive solution, the time has come where large implementations based on real parallel hardware have to prove that parallel computing is as promising as was predicted. It then becomes necessary to develop more solid research methodologies, also allowing comparison of results obtained by different groups.

In the course of the project we decided that to be able to do so it is helpful to look at methodologies used in other areas of science like physics. In particular the methodology and the problems that have to be solved in experimental physics such as high energy or low energy physics show a certain similarity. Although experiments in parallel computing do not require a similar preparation time and are running for a shorter period in time than their physics counterparts, there are similarities and one could learn from their methodology of experimentation.

In physics an experiment is performed to test a certain model or a number of hypotheses. For this purpose an experiment is designed. Such an experiment might require a lot of equipment like detectors (including read-out electronics and data acquisition computing) to measure the phenomena of interest. The design of all that equipment on itself often represents a large effort. Consequently the preparation phase of such an experiment might require a number of years and the experiment is often carried out a number of years after it was prepared. Because it can only partly be foreseen whether the choice for equipment was correct, an experiment must possess the flexibility to allow changes in equipment during the time the experiment is running, which often also could be a number of years. During this period data is collected and analysed. The results of this analysis often will demand a modification of the model. Moreover, this could require to test other properties sometimes demanding considerable modification of the experiment or making a new experiment necessary. In all phases of experimentation simulation plays an important role. Simulation can be used to get a better feeling for the essential parameters to test a model, or a design. This can be either a theoretical model, the design of a detector or the behaviour of the essential parameters one wants to test in an experiment.

So what is the resemblance with parallel computing and what can we learn from the example in physics. The advances in parallel computing have come from the progress in the key technologies of e.g. electronics (in particular chip technology) and computer design. In the last field, processor, memory and network switch design have been the most important issues for the actual realization of parallel computers. Nevertheless, as it was already observed, the progress in the field has not been so fast as was expected a number of years ago. In particular it turns out to be difficult to efficiently exploit the processing power present. Partly this is caused by factors in the hardware like an unbalance in the correspondence between processing and memory or processing and the performance of the network switch. The largest problem, however, is the lack of understanding of the role parallelism plays. More specific the mapping of the parallelism from the application onto the parallel hardware demands for far more research. This research encompasses modelling of parallelism as well as experimentation with parallel implementations. In the PRISMA project we have chosen for the experimentation with different implementations of the parallelism as present in a database management system.

We believe that the problems envisaged during these experiments show a certain similarity with those in physics that were discussed and that consequently we could learn from them. During the planning phase of the PRISMA machine it was realized that a large number of design decisions had to be taken for which no clear incentive existed which solution would be the best under the given circumstances. Moreover, we discovered that our ideas of how to solve the problems we were facing, changed during the time we were building up our equipment. For example, when POOL was designed it was believed that the language should not make any difference between data and code objects, and that the users of POOL should have no influence on the allocation of their objects over the processors of the machine. At the beginning of the PRISMA project we realized that these assumptions resulted not only in severe execution speed penalties, but, what was more important,
restricted the possibilities to evaluate different implementation decisions. And, last but not least, also the area of data management itself evolved over time. Therefore a modification to the language became necessary and demanded for flexible solutions.

In conclusion, the ideas about modelling parallel database applications as well as implementing them on a parallel object-oriented platform is similar to designing an experimental facility allowing to perform different types of experiments in parallelism. Like in physics such experiments should show continuity and being designed with sufficient flexibility in mind as to allow series of experiments with different parallel implementations. Also like in physics we have to accept that because of the long development time and the fact that experiments can run for a considerable period in time, the hardware will not always be according to the latest state-of-the art in technology. This does not have to be a disadvantage as long as it is, to a reasonable approximation, possible to scale and calibrate the results of the experiments. In this paper we will further concentrate on how such an experiment should be built and what type of extra tools it will require.

4 Design and realization of an experiment

When we are designing an experiment roughly three groups of questions play a predominant role. These are:

- What is the experiment we want to do?
- What do we want to measure?
- How do we build the experiment?

In Table 1 the three main groups are listed.

In the first group we find the type of questions we have to ask to make sure that we are setting up a correct experiment. In particular that we have given sufficient thought towards the aims and scope of the experiment. In Figure 1 the basic cycle of experimentation is presented. When starting to design an experiment one of the basic questions is which model or what set of assumption do we want to test. When we do not have a sufficiently clear idea, simulation studies of the model under test could help us to get a better feeling towards which parameters would play a crucial role in our experiment. In this phase it is important to study results of others to prevent duplication of effort. Also it is essential that we are able to restrict ourselves and define a clear program for the series of experiments that have to be performed.

![Figure 1: Cycle in experimentation](image-url)
Design of experimental facility

- What is the experiment I want to do?
  - which model or assumption(s) do we want to test
  - what are results of others
  - where can we do better
  - how can we restrict ourselves

- What do we want to measure?
  - do we measure what we want to
  - what data can we expect
  - what biases do we introduce

- How do we build the experiment?
  - what type of flexibility is necessary
  - what type of tools are required and which are critical
  - which data acquisition chain is required
  - what dataprocessing tools are necessary (visualization)
  - can we build triggers
  - what is the foreseen lifetime

Table 1: Design of experimental facility

In the second group questions related to the actual measurement activities are presented. An important issue there is whether we really measure those parameters which are important for the study of the model under test. And if we know which parameters to measure, the issue is which type of sensors and probes are needed. A question is: are those probes not disturbing the parameters and processes we plan to measure. In that case we are producing data that could contain a lot of biases. To have a better impression of what type of data and what type of biases we could expect we again can use simulation. We can try to simulate the influence our sensor or probe has on the measurement and to find a method to calibrate our measurements for biases we think we will introduce.

In the third group of Table 1 the questions concerning the actual realization of the experiment are listed. Here we have to concentrate on such issues as whether the experiment will run for a large period in time and consequently the type of flexibility that will be required. But also the type of equipment is important. What do we have to develop and what can we buy of the shelf. It has to be stressed here that the more we push technology in our experiment the more we will be obligated to design our own equipment which will result in a longer development time for our total experiment. In almost all types of experiments and specially in computer experiments this device plays a predominant role. This is partly in the read-out and control of an experiment and for the other part in the data processing, data analysis and the visualization of the results. In experiments where the computer is
the only equipment of experimentation, such as in parallel computing experiments, the double role of the computer, e.g. as the purpose of the experiment and as an aid in experimentation, has to be realized. This implies that specially during the measurement process additional tools have to be developed (like tracers and profilers) that help us to perform the measurements. However, what is more essential we also have to develop tools that allow the flexibility to do the experiments. In case of experimentation with parallel implementations like in the PRISMA project we had to develop annotations allowing the experimenter to indicate parallelism.

5 Database Experiment

In this section we address the questions “What experiment do we want to perform?”, “Which parameters do we need to measure?”, and “How do we build the experiment?”, related to the database system.

5.1 What experiment do we want to perform?

In this subsection we discuss the assumptions we want to test, compare our approach with others, and indicate how we confined ourselves to set up the experiment.

In very global terms the goal of the PRISMA project was to find out whether the usage of parallelism and main memory as the primary storage medium would increase the performance of a relational database system, and whether this performance would be scalable. This question is of course too general to be tested so we had to be more specific.

The architecture of PRISMA/DB is very traditional, the reason being that simplicity is often the best in a parallel environment. Advances made in centralized environments often do not apply to distributed environments. The only point where we deviated from traditional approaches were the data managers. We choose for data managers that could manipulate only one type of relation, which is a parameter that is given at creation time. These data managers are called One-Fragment Managers (OFM).

PRISMA/DB consists of components such as parsers, (SQL and PRISMAlog), a query optimizer, a transaction manager, a data dictionary, and an OFM. None of these components has parallelism in it. Parallelism is obtained by executing these components on different processors. During execution there might be an arbitrary number of instances of these components depending on the query load, assigned to different processors. This architecture emphasizes medium to coarse grain parallelism.

A database management system as described above can of course be implemented in various programming languages running on top of various hardware configurations. An additional goal of the PRISMA project is to see whether an object-oriented language, to be more specific POOL-X, has the right abstraction level for implementation. POOL-X [3] shields the hardware architecture, a shared-nothing, 100-node multi-processor system, from the database system implementor, including as a matter of fact the main memory aspect. The language provides a way of specifying communication protocols between objects, without having to worry about low level implementation details such as buffering.

The current trend in computer science is to define languages with an increasing abstraction level to make life easier for the application programmer (4 and 5GL). At the same time the burden for the implementor of these languages, to come up with an efficient implementation, is growing. It is our conviction that a separate language is necessary to describe performance knowledge of the application, to make the implementation efficient. Especially, coarse grain parallelism is so inherent to the application that no compiler can ever detect it. Therefore, POOL-X has a language of pragma's to capture performance knowledge of the database management system. We started with pragma's only for allocation directives. But after a first implementation other pragma's were added to indicate that the object is actually a local data structure (see Section 7).

All of these assumptions mentioned above are subject of our experiment, in the sense that we
want to find out whether we have taken the right decisions. Some of these decisions can be altered during the project based on experiments, others are so inherent to the project that finding out that they were wrong will lead to new projects with altered assumptions.

As mentioned, comparing computer science research done in different groups is rather difficult. Because there are too many degrees of freedom different groups tend to take different assumptions as their starting point. On very few points there seems to be consensus. Furthermore, there seems to be a tendency to have projects with mixed goals, a functionality goal and a performance goal. Without giving all the references (these can be found in [5]), some of the projects have as functionality goals a standard SQL interface, a logical query language interface, extensibility, etc. while others are restricted to join operators, transaction processing environment, etc. Also the hardware environment might differ enormously, e.g. centralized versus distributed, main memory versus disk based, bus-based network versus point-to-point connections, shared nothing versus shared memory, etc. It is obvious that comparing the performance of systems with different functionality is like comparing apples and pears. Also, the various hardware implementations have different scalability characteristics. Finally, a lot of projects never exit the simulation phase, so one never knows whether the result also holds for real implementations.

The PRISMA project has its own mixture of functionality and performance goals. In spite of the heavy performance goals (we wanted to be significantly faster than other database machines), there were also quite extensive functionality goals: a standard SQL interface to facilitate the support of existing applications and a logic query language interface, called PRISMAlog, to make the support of knowledge-based applications possible. As far as the performance goals are concerned we wanted to take a leap in the number of processors by choosing a 100-node system. Also, because the prediction is that main memory is becoming cheaper, every processor has 16 Mb of memory.

To obtain all the goals set, we of course also had to limit ourselves. Limitations that were given up front were the hardware architecture and the implementation language POOL-X. Other confinements are: choosing a traditional architecture of the database management system, taking simple solutions for some of the functionality aspects, only considering medium to coarse grain parallelism, only considering complex, ad hoc queries, not covering certain topics in detail (e.g. fault tolerance).

5.2 What do we need to measure

In this subsection we look at the parameters to be measured and the consequences of measuring.

If we want to know the overall performance of the database system or if we want to get a better insight in parallel execution of queries we have to determine the relevant parameters. For example, does the amount of communication traffic between objects say anything about the overall performance of the database system. Also, does just running benchmarks and obtaining response time and throughput information give a better insight in parallel execution. The question “which are the right parameters to be measured” is difficult to be answered up front. Seeing the results of one test run, one realizes that other parameters should have been measured to gain better insight. So, the database system and its environment should be set up in such a way that measuring the right parameters should be relatively easy. As in many other projects, also in the PRISMA project this aspect has been underdeveloped. Work in this area will continue after the official termination of the project in October 1990. Another problem is the presentation of the parameters measured. Most of the time one is flooded with data. The presentation problem is addressed in the next subsection.

Based on our current experience we have decided on the following list of parameters to start with: response time of (sub)queries, throughput of the database system, CPU utilization, network utilization, profiler information concerning the activities of objects. Where possible we need a breakdown per database component to get a good idea about the bottle-necks in the system. As mentioned before the parameters may change after the actual testruns.

As with all experiments in physics we have to be well aware of the fact that measuring a system also influences its behaviour. So, there should be a constant validation of results obtained from the measured system.
5.3 Setting up the experiment

In this subsection we look at which parts of the system we want to change and what tools we need to perform the experiment.

Although, many aspects of performance may be a topic for experimenting, here, we focus only on parallelism. Various forms of parallelism are used in the manipulation of the data (queries and updates, including constraint maintenance) and in the handling of queries and updates (parsing, optimization, etc.).

The manipulation of the data can be experimented with by allocating the base OFMs and the intermediate OFMs to different processors or by using different strategies for specific operators, e.g. for the join, or for queries altogether. This experiment can be undertaken by changing the query optimizer such that it produces different strategies or different allocation directives.

The handling of queries can be experimented with by allocating the components of the database management system to different processors, or, maybe, splitting up a component to be able to assign its task to several processors. The latter comes close to changing the architecture of the database management system. This experiment can be done by changing the allocation pragma's of the objects that together constitute the database management system.

6 The POOL experiment

In this section we address the questions “What experiment do we want to perform”, “Which parameters do we need to measure?” and “How do we build the experiment?” related to the POOL approach. Note that POOL-X can be viewed as a tool towards the PRISMA/DB and the PRISMA/DB as one major application towards POOL-X, therefore there is a strong correlation between experimental aspects concerning the POOL language.

6.1 What experiment do we want to perform?

The main intention of the POOL language is to offer the application programmer a flexible, clear, powerful and safe programming model for parallel application. Flexible in the sense that the programmer must be able to write a large variety of parallel programs, ranging from small to large, from dynamic to static, and from medium to coarse grain. Also a clear model is chosen: the object-oriented approach. Each object protects itself by running its own sequential program. The interaction between objects is realized via message-passing using explicit communication. The approach is made powerful by removing as much resource management from the programmer as possible, so for instance the programmer is not bothered by allocation/deallocation of messages and deallocation of redundant objects. Finally the language offers optimal safety by using a strong typing mechanism. By writing several POOL applications, amongst others DBMS, we want to validate the before mentioned claims on the expressiveness of the POOL language.

Of course the implementation of such a powerful language is not trivial. Therefore we also want to investigate whether the implementation can be made sufficiently fast to compete with other (parallel) languages. Because the performance of several features in the language, such as communication, is very difficult to estimate on beforehand, and also because we wanted to have a real parallel environment for experimentation, we decided not to use simulations, but to build and implement an object-oriented platform, consisting of compiler, operating system and hardware. When starting the implementation the “typical behaviour” of POOL programs was not known. Therefore a mainly software-oriented approach was chosen. Only for the communication special hardware was proposed, since this was expected to be crucial for performance. Furthermore we made a choice to use a distributed tightly coupled hardware platform, since this was expected to best fit the execution model of POOL and is also preferable because of extensibility, scalability and modularity. This complies with the choice already made in the DOOM project. From measurements we hope to derive guide-lines for hardware support in possible second generation systems. For flexibility we chose to
implement the operating system fully by ourselves directly on the bare hardware, other reasons were
the expected inefficiency of general purpose operating systems and less implementation effort. For
similar reasons we also chose to implement the compiler fully by ourselves.

Note that there is a potential conflict between the goal for expressiveness and the goal for efficiency
of the implementation.

During the design of the system we have imposed several restrictions on ourselves. First we
concentrated on the execution of a single POOL program at a time. Furthermore limited effort was
put into fault tolerance. Only error detection is supported for communication, using checksums and
sequence numbering. In order for the DBMS to recover after a crash a stable storage facility was
offered.

6.2 What do we need to measure?

The expressiveness of the POOL language is not measurable in an ordinary sense. This aspect can
only be "evaluated" by writing several applications and interact with the programmer to get some
insight into his experiences.

The performance measurement of the POOL approach as a whole can only be done by running
several applications on top of the object-oriented platform and derive figures from the behaviour of
the applications. These figures can be used to modify the compiler and/or operating system.

Some components of the object-oriented platform deserve special attention, since these are ex-
pected to be bottle-necks and/or potential candidates for hardware support. The most important
components are :

1. Scheduling
2. Memory management
3. Communication
4. Garbage collection
5. Object allocation
6. Tuple manipulation (only DBMS)
7. I/O system

The garbage collector takes care of removal of redundant objects. The I/O system allows inter-
action with the disks attached to the machine and some host systems connected via ethernet, it also
supports the stable storage.

Note that speed up should be measured from application level, not from the object-oriented
platform, since the application is primarily responsible for a proper parallelization of his algorithm.
However, the object-oriented platform should give support in two ways. First it should offer some
tools to measure the speed up and secondly it should give some characteristics of the implementation
(for instance ratio communication/processing speed, communication delay) in order to let the pro-
grammer tune its application. But it is not directly clear which characteristics should be provided
and this might also depend on the application. A similar difficulty is to relate figures measured in
the object-oriented platform to a POOL program.

To provide characteristics of the object-oriented platform, appeared to be a difficult task. There-
fore a study was done, to get some insight on how this should be done as input for future systems.
6.3 Setting up the experiment

As already mentioned in Section 1, we chose to follow the DOOM approach at the hardware level. For experimentation the Communication Processor (CP) hardware offers two programmable features. First by installing routing tables, one can select different networks and also exploit dynamic or static routing. Second the ratio of buffers for packets from the node the CP is attached to and packets on the way can be varied.

By fully implementing the operating system and compiler ourselves we also allowed a large degree of experimentation. To measure the performance of the components mentioned in Section 6.2 either a profiler can be used and/or some specific counters/time measurements can be incorporated into the components.

7 Tools for experimentation

In this section we will discuss some of the tools we need to perform the above experiments. First of all, we need the pragma’s to explicitly control the parallelism. Furthermore, we need tools to automatically perform testruns with benchmarks and profilers and performance monitors to produce the required data. And, finally, we need presentation tools to present the parameters measured at the right abstraction level. Sometimes, one wants detailed information, and sometimes just global information. Besides this, simulation tools are required to justify design decisions that have far-reaching consequences for the overall architecture.

As was mentioned we consider annotations as additions to the original language, but also as handles to allow us to experiment with different implementations. An annotated program should also run on a sequential implementation of the language. So, in contrast to the additions to other imperative languages to obtain parallelism (e.g. message passing primitives), annotations do not change the result of the computation and can be considered as a kind of comments made by the programmer, that will be passed to the lower levels of the system, where they may be used to generate or speed up parallel execution. Such annotations are said to be transparent.

Apart from the syntactical differences, we can distinguish at least three different levels where parallelism can be annotated [7]:

- At the first level one is concerned with annotating possible parallel computations. These are the most fundamental annotations where the programmer expresses his expectation that a computation may be executed as a concurrent task.

- The second level is where the programmer ensures that parallel computations have a sufficiently coarse grain size. Especially in recursive programs one may start off with coarse grain parallel tasks, but during the recursion the grain size continuously decreases and has to be stopped somehow to prevent the architecture to be flooded with fine grain tasks.

- The third level is concerned with allocation of tasks and loadbalancing. Apart from generating a number of coarse grain parallel tasks by annotations of level 1 and 2, the programmer may know a clever way to distribute those tasks efficiently on the parallel architecture in question. These allocation annotations may be dependent on physical characteristics of the underlying architecture.

Apart form the annotations concerning parallelism it appeared necessary for POOL to include annotations that increase the efficiency of compiled sequential code. In many cases a special behaviour of an object can be annotated such that the compiler can avoid the overhead that is incurred for full blown objects.

As performance handles towards the application level, two facilities are offered, monitoring and profiling:
• The monitor gives a first glimpse of the performance of the system by displaying the CPU, communication and memory occupation of each node in a graphical way during execution of the program.

• Two profilers are supported, the first one measures the system in a usual way, based on function like calls, the second one extends this scheme to the object level.

When using a profiler it is important that the user is able to select that data that is useful for the type of study he is performing. For that reason the profiler consist of a tool collecting all raw data and putting it into a file, as well as some extra facilities that allows the user to wrap around in that part of the data he is interested in. The presentation tools of the profiler allow the user to make among others histograms and present them on any type of terminal.

Some preliminary work has been done on performance modelling of the PRISMA machine. It uses information of the structure of POOL-X programs mapping it on tasks models, with as characteristic input the amount of work and the amount of communication involved in tasks. The parameters for these models will be obtained from the real POOL-X programs, using a trace tool collecting this information.

8 Experience and Status

In this section a short description of the status of the project will be given. In addition some of the experience with PRISMA/DB and POOL-X will be presented.

8.1 PRISMA/DB

Determining the functional specification of PRISMA/DB was harder than expected. There were many design decisions to be taken regarding topics on which there is very little consensus. Not all design decisions can be accounted for. Some were taken in the implementation phase. Looking back there was a constant struggle between functionality requirements and performance requirements. Many design decisions were discussed in a qualitative sense, too few decisions were actually validated by simulation. But we gained a lot of experience in building DB0 and were able to correct some of our mistakes in building DB1.

Building DB1 took a little more than 2 man years. It comprises 40,000 lines of POOL-X code. Most of the tools to perform the experiments are implemented. So, we now enter the experimentation phase. First testruns show that execution on one processor is well within the range of an implementation language like C, except for occasional interrupts of the garbage collector.

8.2 POOL-X

Already in a very early stage of the DOOM project it was recognized that the compiler/operating system was unable to automatically allocate objects over the system in a optimal way. Therefore pragmas were added to the object-oriented platform, to let the programmer explicitly control the allocation. But after the first implementation, the performance of the system was still poor. This was due to the fact that the compiler was unable to optimize on local communication to simple record like objects. Instead of direct access the more expensive communication primitives were used. As a consequence another type of pragma's was introduced to specify some simplified object behaviour types.

Currently the implementation with this new pragma's is finished. Some preliminary measurements show promising results on the performance. More measurements are necessary, however, which might lead to adjustments in the compiler/operating system. To begin with we will use the general profiler, before adding specific facilities to the components. The monitor and first profiler are operational.

In conclusion we can say that during the PRISMA project the emphasis more and more shifted towards the methodology aspects of the project.
References


