An Object-Oriented and Parallel Simulation of a Power-Plant

Klaus Wolf[1], António Mano[2], Sérgio Prata dos Santos[2], Jean-Marc Letteron[3]

[1] GMD - German National Research Center for Computer Science
Schloß Birlinghoven, D-53754 Sankt Augustin, GERMANY
Phone: +49-2241-14 2557, Fax: +49-2241-14 2181, Email: klaus.wolf@gmd.de

[2] EDP - Electricidade de Portugal, S.A./PROET;
Av. Estados Unidos América, 55; 1700 Lisboa, PORTUGAL
Phone: +351 1 847 0180, Fax: +351 1 840 9419, Email: sergio@softpar.edinfor.pt

[3] SEMA GROUP, 16 Rue Barbes, F-92126 Montrouge cedex, FRANCE
Phone: +33 1 40 92 40 92, Fax: +33 1 46 56 96 53
Email: Jean-marc.Letteron@sema-taa.fr

Abstract. The simulation of complex physical systems using realistic models is still a challenge to the design of the simulation program as well as to it's performance. While former simulation-environments were either well structured but slow or hard-coded and fast, the SOFTPAR Esprit Project 8451 environment offers a real solution for both requirements. The simulation of a coal fired power generation plant profits from this software factory. Designed in the standard HOOD method and coded in paradise C++ style, the simulator can run on a wide range of parallel machines. A performance evaluation shows the profit in clear numbers of gained performance-speedup.

1 Introduction

The simulation of complex physical systems using realistic models can provide us a wealth of information and experimentation possibilities, some of which are impossible to safely perform on a real system.

However, the large execution times necessary for complex model simulations using standard sequential techniques can severely impair the usefulness of the model. Parallel computing is a possible way to address this issue but it raises several questions regarding, for instance, model decomposition, information distribution, synchronization and data consistency.

The simulator described in this article deals with such topics and was developed at PROET/EDP in the scope of the SOFTPAR ESPRIT project 8451 [Softpar]. The tools implemented in the Softpar project provide parallel application programmers with a complete software factory Concerto supporting all life-cycle activities from design to code and test. The high performance parallel simulation of a coal fired unit model of a power plant, developed in a joint program between EDP and INESC to test the effects of different types of controllers on the energy generation process, was used as a demonstrator.

2 Power Plant Simulation Requirements

The need for a specific power plant model simulation led to the project to use parallel computing in that context. The model concerns a 314MW coal fired unit of a power generation plant and is non-linear, concentrated parameters type and time invariant with a special focus on the boiler-turbogenerator coordinated control problem. The simulation of other similar models and configuration flexibility is also part of the requirements. The key specifications guiding the simulator development are:

- **Simulation Accuracy:** The simulations outputs must be well within the models error tolerance.
- **Performance:** The simulation performance must be greatly increased and make good use of the available parallel environment. Simulation times close to real-time for the main submodules of the model are expected using an 8-node X'plorer parallel machine.
- **Flexibility:** Experimentation with different controllers and configurations is part of the simulator development motivation and it is important to handle these features in a user-oriented way since they are expected to be fully used.
- **Run-Time Interaction:** Provided that the simulation time is increased to real-time magnitude, the model use and development can benefit from a user on-line interaction with the simulator, allowing the modification of variables and parameters to be instantly expressed in the simulation results. This feature creates new possibilities to the use and usefulness of the simulator.

The development of a user-friendly interface transmitting the simulator capabilities is also part of the project specifications.

The power plant model was previously implemented in MATRIXx. It is structured into several modules, the main ones being: *Fuel feeding, Air Gas Circuit, Water boiling circuit, Steam circuit* and *Turbo Generator.* It has been run on a Sparc UNIX workstation. Due to the high number of equations and the complexity of the model, only a subset of the entire model could be run at the same time. The simulation tool took about 3 days to complete the simulation of 2 hours of operation for a model subset corresponding to 1/4 of the entire model.

The running time of the model subsets was considerably long in part due of the nature of the simulation. Because some of the components of the model have short time constants, a very small time step needed to be used in order to provide a reasonable simulation accuracy. Because the MATRIXx system package can only provide a common time step for all the system, the time step used for this fast running components was used also for slower components that would not require it.

3 Application Developement

3.1 Functional Description

The mathematical model of the power-plant was taken from the existing implementation in MATRIXx and was re-implemented using the SOFTPAR tools.

An important requirement for the simulator was also its modifiability and configurability in order to assist the further development of the models. During the design phase it was noticed that a generalized concept for the simulator was possible without implying substantial changes for adaptation of the general concept to the dedicated subsystems included in the applications. This included the necessity to develop a general translator to generate code from the MATRIXx descriptions.

The specifications for the application were changed in order to accommodate this possibility. This change greatly increased the simulator flexibility without increasing significantly the complexity of the system. The configurability intended for the simulator was considered as a way to improve the performance of the simulation in run-time. The possibility of a good parametrization of the system was considered as essential for optimization of the task of each system component. By sufficient abstraction of a component's task it can be represented by its physical and mathematical properties and unnecessary work for separate implementation is saved. Another important requirement in view of very complex models was the possibility to simulate only some parts of the system. The interaction with the user required also the development of an application running on the user host machine. This application was also developed using HOOD [HOOD].

The application primarily requires a way to decompose the model equations in order to efficiently distribute and execute the solving work in parallel. The uncertainty regarding the execution environment (potentially heterogeneous) and the model equations characteristics led to the creation in run-time of the simulation structure responsible for the simulation work and distribution. This structure is built by the engine at startup time according to the user specifications and permits to deal with heterogeneous environments and unbalanced equation decompositions. Working with general models requires that the simulator engine contains no information regarding any concrete model. The actual model data is given by an external translator that generates code from the model equations in the form of engine-derived objects. These modules form the main building blocks of the simulation structure.

The execution of the simulation is performed by dividing the simulation time in time steps and executing the simulation procedures for that time in each of the model components. The time steps can be of different sizes for different modules and corresponds to their information exchange rate with the rest of the system.

3.2 HOOD-Design

The main structure of the simulator engine core is presented in the HOOD diagram in figure 1. The *engine controller* active object is responsible for keeping the system working and attending to the user external requests either on configuration or simulation time. The *name server* object keeps the mapping between the model variable names and their internal representation. The *heartbeat module* provides the simulation structure with the execution pulses for each time step, feeding the external model inputs into the system. The *run-time simulation structure module* contains the base and initial context for the simulation

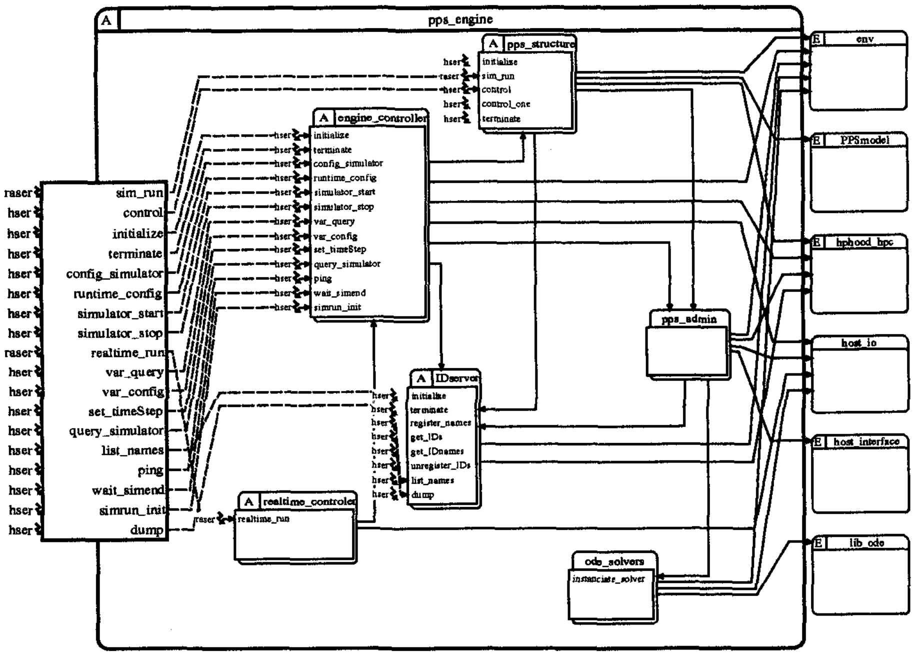

Fig. 1. Top level Design Structure of the application

structure and the *pps_main module* contains its main building blocks. *Ode solvers module* contains the objects with the differential equation solvers used in the solving steps.

The runtime-simulation structure consists basically on three types of HOOD objects: the *Subsystems*, which contain the simulation support and interface code with the model objects; the *Systems* or administration objects, which are responsible for grouping sets of Subsystems or other administration blocks; and the *active context* objects which provide the machine contexts to spawn the structure in the parallel environment and can transparently encapsulate either one of the passive objects. The structure objects are all true HOOD class objects deriving from a common ancestor, giving the system a very flexible and efficient way to deal with distribution by transparently handling out-of-context structure members. The creation of remote objects is done through the instantiation of local interface objects and remotely spawned HOOD active objects, which means that the actual code for the distribution process is left to the HOOD generator tool. The communication primitives are also transparently handled by the same tool. The remote evocation of simulation related methods uses the RASER (Reported Asynchronous Execution Request) protocol in order to allow the sender to keep working (or make other calls) until the replys are collected.

The simulation execution step is performed by feeding the structure root with the model external input variables and distributing that information, together with the variable values from the previous step, through the structure. The equation solving work is executed in the structure leafs (model Subsystems) and the result values are retrieved again through the administration intermediate objects. All the information paths are calculated and prepared during the structure start-up in order to optimize the execution during simulation time. Administration objects are aware of the in-context or out-of-context character of their sons, and use that information for the parallelization of the simulation algorithms.

The decomposition of the model into true HOOD objects that are distinguished only on run-time and their transparent interchangeability creates the opportunity to give the system a very high flexibility regarding specific and local configurations. This is used to allow, for instance, different time steps for structure branches, local defined and configured differential equation solving methods, and run-time setup and query of specific modules or variables.

3.3 Additional Fine-Coding in paradise C++

The target language independence of the design phase is not particularly important or visible in the automatically generated paradise C++ [HPC++] [SwEng] code because HOOD provides a good mapping of its structures in this language. The generated code, although more mechanical, is natural enough to be considered modifiable. The user inserted code is well integrated inside the generated structure and since only the structure and the interface is automatically generated, the code efficiency is close to hand generated code and the interface is fast and efficient.

The simulator run-time configuration was implemented through the HOOD active classes which can be spawned in run-time and allow the application to expand according to its needs. This flexibility is supported by paradise C++ possibility to create and delete new processes in run-time and connect them with the existing structure. The flexible mapping of the application virtual nodes to physical nodes is also the result of explicit paradise C++ support.

4 Simulation Performance Evaluation

The performance obtained so far, both in terms of accuracy and computation times, meets the simulation expectations. Simulation times for sub-models were boosted to real-time magnitude operation with an accuracy well within the model error tolerances.

The simulation performance depends on a set of dependent parameters and configurations and a balance between them must be achieved in order to have the simulation executing with the intended profile. Factors determining the performance of the simulation include specific model properties (like fast dynamics), user model organization, time step values, local configurations and precision settings of differential solvers and mapping of active contexts into physical nodes.

The accuracy for the entire power plant model is still under evaluation since the previous system did not allowed the full model to be run and no previous results for complex combined tests exits. Global accuracy is however, expected to be within the model tolerance since the sub-models execution yielded a very good accuracy. Global performance for the entire model, using worst case configurations from the sub-models for 20 seconds of simulation time and different hardware configurations was nearly linear up to 8 processors used. A speedup of 6.14 using 8 nodes and comparing with execution on 1 node (where no message passing is used due to local optimizations) indicates the distribution effectiveness of the system. A gain of around fifty times on the sub-models execution, when comparing with the previous system, and the possible on-line interactivity with the simulation, defines new ways to deal and experiment with the model expression. Tests on a 8 node workstation cluster yielded similar speedup values but with less absolute performance, mainly due to inferior performance of the computational nodes.

5 Conclusion

The automatic handling, by HOOD/paradise C++ , of the process management and remote method evocation protocols allows the designer to focus on the real application problems, leaving the complexity of distribution details to be automatically generated. Moreover, paradise C++ close support to HOOD design features and distribution in general, assures that the generated interface and management code is small, efficient, well localized and easy to be understood.

The large improvement in performance obtained with the present simulator, supports an increased interest in working with complex model simulations in the power generation field. Faster access to simulation results and on-line interaction creates new possibilities to develop and work with new models.

The implementation of the simulator using high level tools like HOOD and paradise C++ is uncommon in the high performance computing world, but demonstrates, in our opinion, that a realistic approach to automatic code generation, focusing on the distribution system management, can greatly reduce development work and complexity in distributed environments.

References

[Goncalves1] A.J. Goncalves *Modelo Para o Controlo Coordenado Caldeira-Turboalternador de uma Central Termoelectrica*; IST Lisboa, 1993

[Goncalves2] A.J. Goncalves *Modelo Para o Controlo Coordenado Caldeira-Turboalternador de uma Central Termoelectrica - Codigo e resultados de simulacao*; IST Lisboa, 1993

[HPC++] Wolf, Lang, Holtz (GMD), *HPC++ - High Performance C++* HPCN95 Proceedings May 1995, Springer LNCS 919

[SwEng] Lang, Wolf (GMD), Letteron (SEMA Group) *Software-Engineering and Parallel Object-Oriented Programming* Parallel Computing: State-of-the-Art Perspective, Proceedings of the International Conference ParCo95, Elsevier Science Editor

[HOOD] Letteron, Bancroft (Sema Grp), Gerlich, Debus (DORNIER) Wolf, Lang, Holtz (GMD) *HOOD and Parallelism in the Softpar project*, HPCN95 Proceedings May 1995, Springer LNCS 919

[Softpar] CEC ESPRIT PROJECT 8451. *Softpar A Software Factory for the development of Parallel Applications* Softpar Project Report, SOFTPAR-S-29.2, January 1996