Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 6, Number 2, pp. 13–22. http://www.scpe.org

APPLICATION OF P-GRADE DEVELOPMENT ENVIRONMENT IN METEOROLOGY

RÓBERT LOVAS‡, PÉTER KACSUK‡, ÁKOS HORVÁTH¶ AND ANDRÁS HORÁNYI¶

Abstract. The main objective of a meteorological nowcasting system is to analyse and predict in ultra-short range those weather phenomena, which might be dangerous for life and property. The Hungarian Meteorological Service developed a nowcasting system (MEANDER), and its most computational intensive calculations have been parallelised by means of P-GRADE graphical programming environment. In this paper the application of P-GRADE environment is demonstrated in a real-size meteorological problem. We give an overview on the parallelisation of MEANDER system applying P-GRADE environment at the different stages of parallel program development cycle; specification, design, debugging and performance analysis. However, the paper focuses on a novel approach of parallel debugging, which combines ideas from the field of formal methods and model checking techniques with advanced parallel debugging methods in a user-friendly way.

Key words. Parallel programming, graphical development environment, numerical weather prediction, debugging, formal methods, performance analysis

1. Introduction. MEsoscale Analysis Nowcasting and DEcision Routines, MEANDER [1][2] developed by the Hungarian Meteorological Service (HMS) has a crucial task in the protection of life and property. Taking into account all the available meteorological observations the numerical weather prediction system can help the meteorological service to issue weather warnings, which are essential for storm warning at Lake Balaton, for aviation, and for other industrial partners. MEANDER system consists of several computational intensive procedures that cannot be executed in time without efficient parallelisation.

On the other hand, MTA SZTAKI and University of Miskolc developed a graphical programming environment, called P-GRADE [3][4] that is able to support the entire life-cycle of parallel program development. In a joint project of HMS and MTA SZTAKI, we applied the P-GRADE environment in order to parallelise the complex sequential FORTRAN and C/C++ code of MEANDER system.

In this paper we introduce briefly the fundamentals of MEANDER system (see Section 2) and P-GRADE programming environment (see Section 3) as well as the way of parallelisation over the design (see Section 4), the debugging (see Section 5) and the performance analysis (see Section 6) phases of program development. However, the paper focuses on a novel approach of parallel debugging (see Section 5), which combines ideas from the field of formal methods and model checking techniques with advanced parallel debugging methods in a user-friendly way. Furthermore, our experiences concerning the usage of P-GRADE, and some comparative performance measurements involving supercomputer and cluster environments, are also presented to illustrate the efficient applicability of P-GRADE environment on both platforms. Finally, we summarise the related works (see Section 7), and our achievements (Section 8).

2. MEANDER nowcasting system. The main objective of a meteorological nowcasting system is to analyse and predict in the ultra short-range (up to 6 hours) those weather phenomena, which might be dangerous for life and property. Typically such events are snowstorms, freezing rain, fog, convective storms, wind gusts, hail storms and flash floods. MEANDER nowcasting system [1][2] of the Hungarian Meteorological Service is not only capable to predict those severe weather events but, if needed, it is also able to issue automated warnings.

For the development of MEANDER nowcasting system the following main meteorological observations could be used as input data: (i) Hourly surface measurements are available on about a 40 km horizontal coverage. (ii) Some vertical soundings of the atmosphere (their number and availability is rather limited in space and time). (iii) Radar data are available at every 15 minutes using different radars covering the whole Carpathian Basin. (iv) Satellite images on infrared, water vapour and visible channels. (v) Lightning detection data over Hungary.

The development was constrained by few considerations, which were as follows: (i) The analysis and 3 hour forecasting should be available within *20 minutes* after the measurement time (this point is crucial in order to provide a timely analysis and prediction). (ii) The system should be able to provide "present weather" information for all the gridpoints of the domain of interest (the expression "present weather" means in meteorology

‡MTA SZTAKI Computer and Automation Research Institute, Hungarian Academy of Sciences
1518 Budapest, P.O. Box 63, {rlovas|kacsuk}@sztaki.hu
¶Hungarian Meteorological Service
1525 Budapest, P. O. Box 38, {horvath.a|horanyi.a}@met.hu

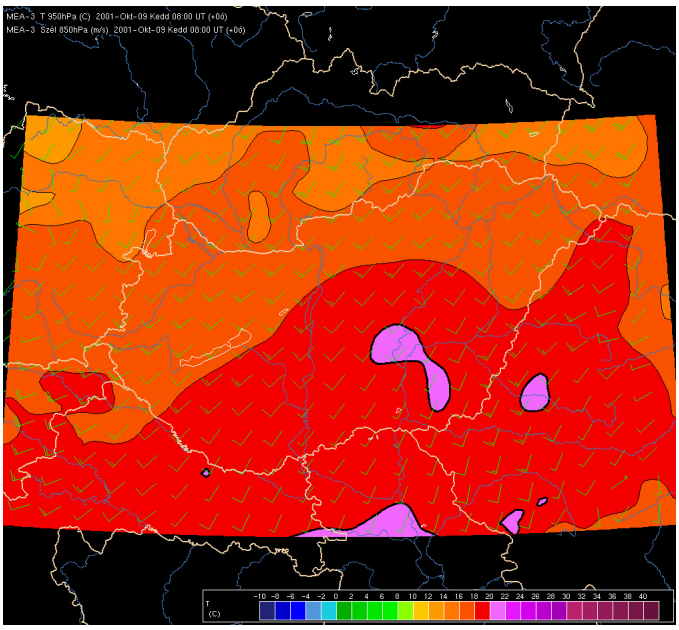

FIG. 2.1. *Visualisation of interpolated basic parameters (wind, temperature)*

a special description of the main characteristics of the actual state of the atmosphere, like thunderstorm, fog, etc.). (iii) The system should be capable to provide automated warnings in case of specified weather phenomena (this aspect is an important one in order to prevent the consequences of the 'dangerous' weather).

Based on these constraints and considerations MEANDER system is going to be developed around three main segments: analysis (the determination of the present state of atmosphere), the linear segment (extra- and interpolation between the analysis fields and the forecasts of the numerical weather prediction models), dynamical modelling part (very high resolution numerical models with special physical parameterisation schemes for the precise prediction of weather events related to precipitation). The warning interface is smoothly fitted into the entire system. In the course of developments, until now, basically the first two segments are fully ready and operationally used, while the last one is under development and tests. Hereafter, briefly the analysis segment will be detailed, where the parallelisation of the computer code was developed.

The core of the analysis system is the so-called optimal interpolation scheme, where the basic meteorological variables are interpolated into the grid of the nowcasting system (basically the grid covers Hungary). The basic parameters are surface pressure, humidity, wind values, geodynamical height and temperature (see Fig. 2.1). To these basic parameters some others are added using radar and satellite information. From all the available information some derived quantities (these are typically type of cloudiness, visibility, phase of precipitation, some 'thunderstorm' parameters, moisture convergence, turbulence, convective available potential energy, etc.) are computed in order to get a deeper insight into the dynamics of the meteorological objects.

The parallelisation of the analysis segment can be based on the domain decomposition of model grid into subgrids (and the computations were carried out independently on the different subgrids) as it is described in Section 4.

On the other hand, the parallel execution of different calculations of basic and derived quantities (i.e. functional decomposition) can be a feasible way of parallelisation if there are enough computational resources.

3. P-GRADE development environment. The central idea of P-GRADE system [3][4] is to support each stage of parallel program development life-cycle by an integrated graphical environment where all the graphical views (applied at the different development stages) are related back to the original graph that defines the parallel program. The graph of parallel application is designed by the user but P-GRADE is responsible for generating all communication library calls automatically on the basis of the graphically described code. Since graphics hides all the low-level details of communication, P-GRADE is an ideal programming environment for average (or even beginner) developers who are not experienced in parallel programming, e.g. for chemists, biologists or meteorologists.

P-GRADE also supports the fast re-engineering of existing sequential applications by means of the applied hybrid graphical language, called GRAPNEL that is based on a hierarchical design concept supporting the bottom-up design methodologies beside the widespread top-down design concept. GRAPNEL provides predefined and scalable process communication templates that allow the user to describe the fundamental behaviour of the program with regular process topologies. In this way, the developer is able to construct complex and reliable applications in a relatively easy and fast way.

4. Design of parallelised MEANDER system. P-GRADE consists of three hierarchical design layers: (i) *Application Layer* is a graphical layer, which is used to define the component processes, their communication ports as well as their connecting communication channels. Shortly, the Application Layer serves for describing the interconnection topology of the component processes (see Fig. 4.1, Application window). (ii) *Process Layer* is also a graphical layer where different types of graphical blocks are applied: loop construct, conditional construct, sequential block, input/output activity block and graph block. The graphical blocks can be arranged in a flowchart-like graph to describe the internal structure (i.e. the behaviour) of individual processes (see Fig. 4.1, Process windows). (iii) *Text Layer* is used to define those parts of the program that are inherently sequential and hence only pure textual languages like C/C++ or FORTRAN can be applied at the lowest design level. These textual codes are defined inside the sequential blocks of the Process layer (see Fig. 4.1, at bottom of Process window labelled *Process: visib_s → visib_s_0*).

The usage of predefined and scalable process communication templates enables the user to generate complex parallel programs quickly. A communication template describes a group of processes, which have a pre-defined regular interconnection topology. P-GRADE provides such communication templates for the most common regular process topologies like Process Farm (see Fig. 4.1, Template window), Pipe, 2D Mesh and Tree, which are widely used among scientists. Ports of the member processes in a template are connected automatically based on the topology information.

In this project we mostly followed the domain decomposition concept in order to parallelise the six most time consuming modules of MEANDER, such as assimilation of incoming satellite images (see Fig. 4.1, 'satel_m' process and 'satel_s' Process Farm), processing of radar images (see Fig. 4.1, 'radar_m' process and 'radar_s' Process Farm), or calculation of derived quantities, i.e. the visibility (see Fig. 4.1, 'visib_m' process and 'visib_s' Process Farm).

Generally, we created one coordinator (master) process and one Process Farm template for each parallelised MEANDER routine. The 'master' process is responsible for dividing the model grid of MEANDER (over the Carpathian Basin) into subgrids as well as for the assignment of these partial tasks to 'worker' processes belonging to the Process Farm. When the worker received the 2D or 3D input data (via the communication channels and ports) to be processed on its part of grid, the worker invokes the corresponding FORTRAN or C/C++ sequential function from a sequential block (see Fig. 4.1, at the bottom of window labelled *Process: visib_s → visib_s_0*).

Preparing the parallelisation, the complex sequential functions of the original MEANDER source code were collected into separated object files. We rewrote (particularly parameterised) the sequential functions and also linked to the P-GRADE application. The simpler or shorter functions were inserted directly into the sequential blocks of Process Layer without changes.

Finally, the master process collects the partial results from the workers relying on the collective communication facilities of P-GRADE, and the master stores the collected information into a meteorological database in netCDF format.

As the Application Window depicts (see Fig. 4.1), the master processes of MEANDER calculations are connected to each other; they passes control messages to each other ensuring the appropriate scheduling among the calculation modules.

Each master process is also connected to a visualisation subsystem (*Ready + MetVis processes*), which is responsible for the qualitative on-the-fly visualisation of intermediate results on meteorological grids (see Fig. 4.1, *MetVis* window). Thus, the users are able to check the progress of calculation in run-time.

5. Debugging. After the successful design of MEANDER system, this section discusses in details the major issues as well as our novel solutions concerning the debugging phase of program development, since the reliability and the correctness are crucial criteria of MEANDER system.

During the debugging stage we took all the advantages of DIWIDE debugger [5], the built-in distributed debugger of P-GRADE environment. DIWIDE debugger provides the following fundamental facilities of par-

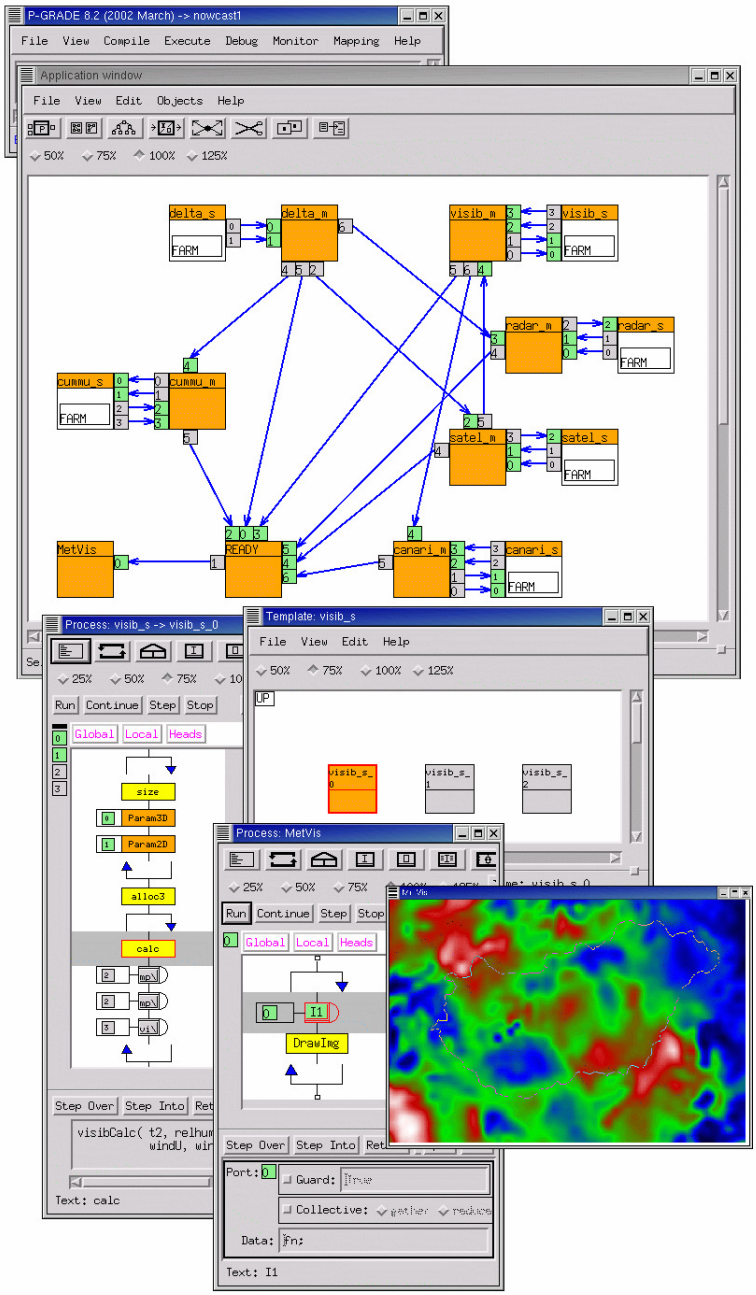

FIG. 4.1. *Parallelised MEANDER calculations in P-GRADE*

allel debugging; the step-by-step execution on both graphical and textual levels, graphical user interface for variable/stack inspection, and for individual controlling of processes.

Besides, the presented MEANDER system also served as a testbed for a novel parallel debugging methodology, which has been partially implemented in P-GRADE framework. This experimental debugging toolset tries to unify and also improve the research achievements of more completed projects [6][7][8]. The major aims of our debugging methodology can be summarised as follows:

- Handling the non-deterministic behaviour of P-GRADE programs.
- Application of formal methods and model checking techniques in parallel debugging, such as Petri-nets or temporal logic specifications.
- Reducing the needed user interactivity.

5.1. Temporal Logic Specification. As the first step of debugging, the user must specify the correctness properties (i.e. the expected program behaviour) of the developed GRAPNEL program using temporal logic (TL) formulas, since temporal logic has proved as an adequate framework for describing the dynamic behaviour of a system (program) consisting of multiple asynchronously executing components (processes) [11][12][17].

We defined the fundamental properties regarding the scheduler and the visualisation parts of the parallelised MEANDER system with three TL asserts. The first assertion (AssertLoad) expresses a basic resource requirement: the upper limit of the simultaneously working processes in Process Farms, which must be always fulfilled over the entire execution. The next assertion (AssertVisual) is used to express another property concerning the reliability of the visualisation subsystem; if any MEANDER routine has finished, its results must be eventually visualised in the future. Finally, the third assertion (AssertScheduling) formulates the expected execution order of MEANDER routines since the calculations have strict dependences to each other, and we also need proper scheduling to avoid the overload of available computational resources.

(1). *AssertLoad*: $[](\text{NumberOfActiveWorkers}() < 16)$
(2). *AssertVisual*: $(\forall x \in \mathbf{N} : 0 \leq x < 6)\ \text{CalcFinished}(x) \Rightarrow \Diamond\text{ResultVisualized}(x)$
(3). *AssertScheduling*: $\text{DeltaFinished}() \Rightarrow \Diamond(\text{CummuStarted}() \wedge \text{RadarStarted}() \wedge \text{SatelStarted}())$
$\wedge\ \text{SatelFinished}() \Rightarrow \Diamond\text{VisibStarted}() \wedge \text{VisibFinished}() \Rightarrow \Diamond\text{CanaryStarted}()$

The atomic predicates referenced from the above asserts, such as *NumberOfWorkers*, are implemented in dynamically loaded C libraries, and they use a simple predicate-DIWIDE interface [7] to obtain the actual state information of individual processes, e.g. actual values of variables, label of currently executed GRAPNEL block, or contents of already received messages.

5.2. Coloured Petri-net model. At the next stage of debugging, the formalism of coloured Petri-nets (CPN) [10] was chosen for expressing and composing a model for GRAPNEL programs because CPNs are well founded, have been widely used to specify parallel software systems and are supported by a number of tools for validation, simulation, analysis and verification.

Some minor limitations and restrictions of GRAPNEL programs are required for the automated transformation to CPN model, e.g. at Process Layer the expressions of conditional and loop constructs must be dependent only on global control variables, which are declared in a separated section of GRAPNEL program. If a GRAPNEL program meets the given limitation rules, the new experimental GRP2CPN tool of P-GRADE system is able to generate a coloured Petri-net model from the developed GRAPNEL program for further simulation and analysis purposes.

The fundamental principles of the transformation from GRAPNEL programs to coloured Petri-nets are described in [6]. However, in our experimental debugging framework the place fusion approach was followed instead of the presented environmental place approach [6], and the positioning information of graphical items at Application and Process Layers are also exported into the CPN model in order to get topologically similar Petri-net to GRAPNEL program (compare the Process Metvis in Fig. 4.1, and its Petri-net model in Fig. 5.1). These modifications enable the user to identify easily the corresponding parts of GRAPNEL program and CPN model, and to simulate the behaviour of GRAPNEL programs from various aspects (liveness and home properties, deadlocks, etc.). The generated hierarchical CPN model is described in XML using the Document Type Description of Design/CPN tool [9] provided for importing and exporting models.

An example, Fig. 5.1 depicts a coloured Petri-net model generated from *MetVis* process of the parallelised MEANDER system. Basically, the presented Petri-net model has been constructed in a sub-page of our hierarchical model, and built up from the Petri-net patterns of FOR-type loop construct (denoted as 'LOOPS' and 'LOOPE'), input communication activity ('I1'), and sequential block ('DrawImg'). At the beginning of the model, a sequential block-like construct is located modelling the initialisation of global variables. In the left side of the Fig. 5.1, the end of the input communication channel/port is represented by two places (used for blocked message passing), and the global declaration of colours and tokens are placed.

Relying on the simulation facilities and the reachability analysis offered by Design/CPN tool the modelled parts of the nowcasting system has been verified successfully.

5.3. Improved macrostep-based execution. The general problem of constructing and replaying consistent global states of parallel/distributed systems must be also considered because the evaluation of erroneous situations depends on accurate observations. To solve these problems, Macrostep Engine (a part of DIWIDE dis-

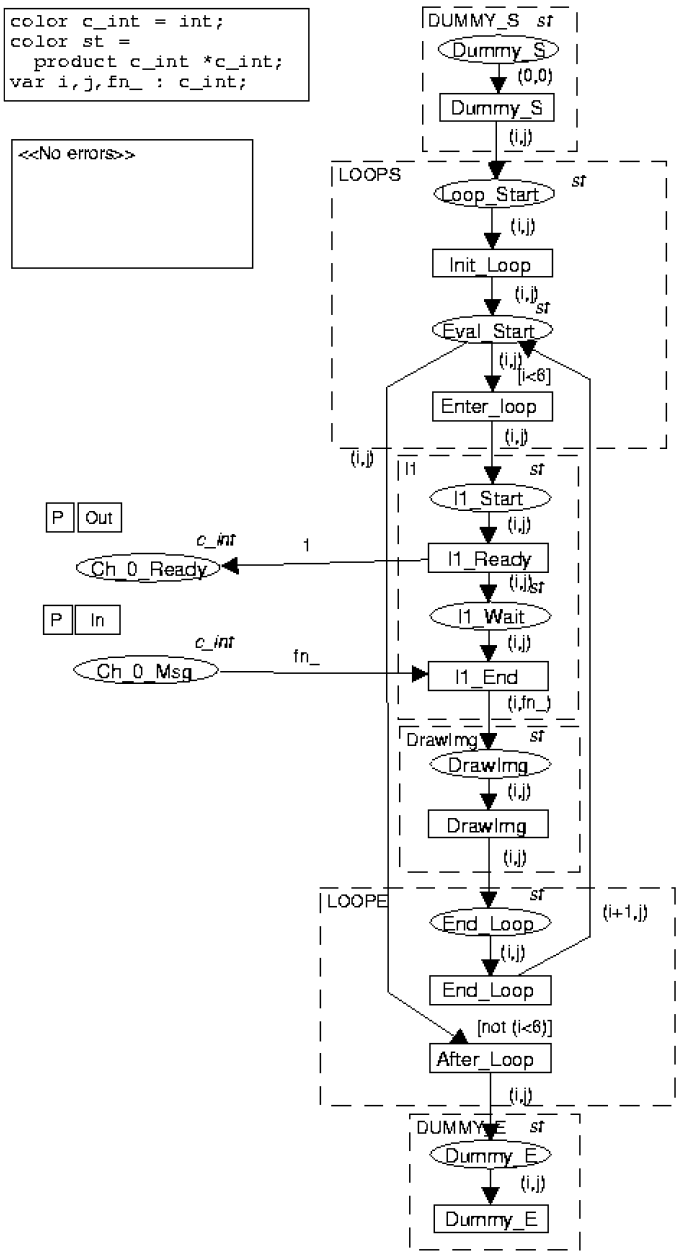

FIG. 5.1. *Coloured Petri-net model of MetVis process*

tributed debugger) provides strategies for the observation as well as replaying of consistent computation states relying on the so-called macrostep-by-macrostep execution. The idea of macrostep-based execution is based upon the concept of collective breakpoints[1] , which are placed on the interprocess communication primitives in each GRAPNEL process. The set of executed code regions between two consecutive collective breakpoints is called a macrostep. A detailed definition of macrostep concept is given in [8].

In this project we realised the analogy between the step-by-step execution mode of sequential programs (by local breakpoints) and the macrostep-by-macrostep execution mode of parallel programs.

The Execution Tree [7][8] (see Fig. 5.2) is a generalisation of a single execution path (based on macrosteps); it contains all the possible execution paths of a parallel program assuming that the non-determinism of the current

[1] A collective breakpoint is a set of local breakpoints, one for each process, that are all placed on communication instructions.

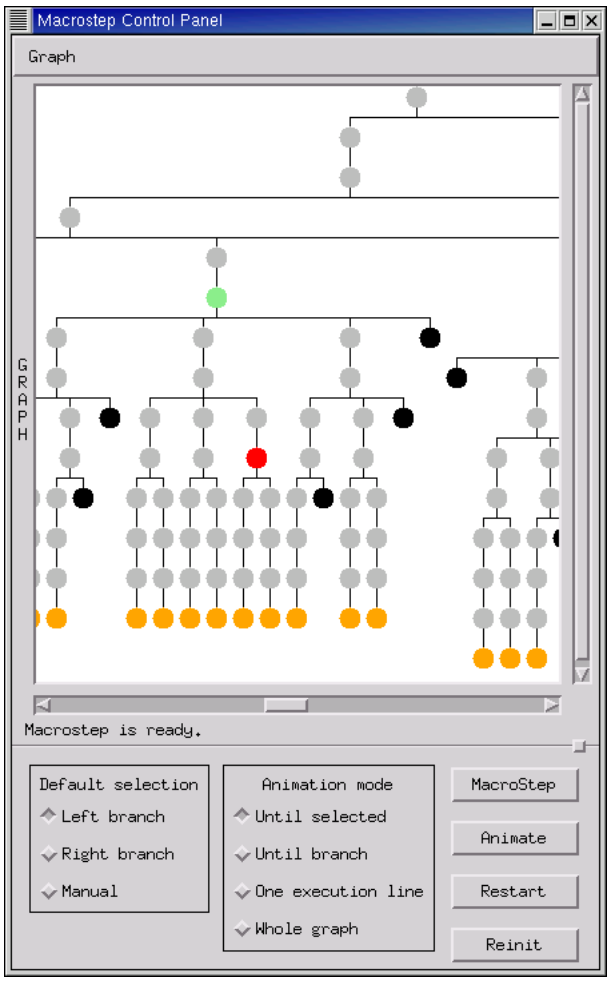

FIG. 5.2. *Macrostep Control Panel*

program is inherited from wildcard message passing communications. In idealistic case, the task of Macrostep Engine would be the exhaustive traversing of the complete Execution Tree. In fact, when a developer tries to debug a real-size program systematically, the user has to face the problem of combinatorial explosion of possible execution paths. Another drawback of the original macrostep concept is that Macrostep Engine is able to detect only a few types of critical failures itself, such as deadlocks or wrong terminations.

In order to improve the macrostep concept, Macrostep Engine is able to cooperate with an external Temporal Logic Checker [7] (TLC) Engine comparing the user-defined TL specification (see Section 5.1) to the observed program behaviour macrostep-by-macrostep. Hence, the debugger framework has new capabilities to detect a wider spectrum of programming/implementation errors, and to cut the branches of Execution Tree earlier than the original macrostep concept.

Meanwhile, a CPN analyser tool[2] can be applied for simulating the program execution from some critical aspects (synchronisation, number and ordering of passed messages, termination, etc.) by means of the automatically generated CPN model (see Section 5.2). The simulation (i.e. the construction of Occurrence Graph [9] of Petri-net model) and the analysis of the results are faster by magnitudes of orders than the real program execution since most of the user's sequential code are not involved in the model. Based on the discovered failures the developer can steer the verification (i.e. traversing of Execution Tree by macrosteps) towards the found suspicious situations.

If an erroneous situation is detected, DIWIDE enables the user to inspect either the global state of GRAP-

[2]Actually we used Design/CPN v4.0.2.

NEL application or the state of any individual process. Depending on the situation, the programming bug can be fixed using the graphical editor of P-GRADE, or the entire application can be replayed to get closer to the origin of the detected erroneous situation.

By the combined efforts of DIWIDE distributed debugger, Macrostep Engine, Temporal Logic Checker, and coloured Petri-net generator/analyser we eliminated several critical programming bugs from the parallel version of MEANDER system.

6. Performance analysis. According to the available computational resources the actual number of worker processes in each scalable Process Farm can be set by a dialog window in P-GRADE. Currently, the Hungarian Meteorological Service works with an SGI Origin 2000 server equipped by 16 MIPS R12000/450MHz processors, and MTA SZTAKI installed a self-made Linux cluster containing 29 dual-processor nodes (Pentium III/500MHz) connected via Fast Ethernet. The parallel version of MEANDER system has been tested on both platforms. Based on our measurements the execution times of individual parallelised calculations depend particularly on the floating-point calculation performance of applied CPUs (assuming the same number of involved CPUs). To take an example, the visibility calculation was executed on the SGI supercomputer within 43 sec (see Fig. 6.1, in the middle of PROVE window), and it took approximately 116 sec to calculate on MTA SZTAKI's cluster. The rate between the execution speeds is almost equal to the reciprocals of the processors' SPECfp rates due to the computational (and not communication) intensive feature of the visibility calculation as PROVE performance visualisation tool [4] depicts. Some calculations, such as the processing of satellite images, were executed on the same time on both platforms; but the processing of radar data was slower on the cluster by a magnitude of order due to the long initialisation time (sending radar images) and the relatively short processing time.

In details, PROVE, as a built-in tool of P-GRADE system, can visualise either event trace data, i.e. message sending/receiving operations, start/end of graphical blocks in a space-time diagram (see Fig. 6.1, Prove window), or statistical information of the application behaviour (see Fig. 6.1, Process State and Execution Time Statistics Windows). In all the diagrams of PROVE tool, the black colour represents the sequential calculations, and two different colours used for marking the sending and the receiving of messages. The arcs between the process bars show the message passing between the processes.

Hereby, PROVE assisted us to focus on performance bottlenecks and to fine-tune of our application. On HMS's 16-processsor SGI computer the parallelised MEANDER calculations are executed within 2 minutes (ca. 15 times faster than the sequential version on an HP workstation). By the end of the joint project, almost the same performance results were achieved on MTA SZTAKI's cluster utilising all the 58 processors, the difference between the execution times was less than 5%. However, we had to take into consideration the saturation of speedup curves by each calculation as well as the limited size of operational memory offered by the cluster when the sizes of Process Farms were increased on the cluster.

Finally, a specialised visualisation tool, HAWK (Hungarian Advanced Workstation) developed by the Hungarian Meteorological Service provides a detailed and quantitative visualisation of the calculated meteorological parameters for the specialists and for other external partners (see Fig. 2.1, temperature and wind at 850 hPa on the basic grid of MEANDER).

7. Related works. Over the past few years P-GRADE system has been installed on several high performance clusters and supercomputers in the frame of various research projects and co-operations. Beside the presented meteorological application, P-GRADE is applied by physicist, chemists and students for designing parallel applications, and two other real-size applications were developed and demonstrated on scientific forums; a simulator for collision of galaxies (N-body problem), and an urban traffic simulator. Relying on the achievements of the presented work a national project has been launched for supporting complex collaborative work and its application for air pollution forecasting (elaboration of smog alarm plans) by means of P-GRADE environment and GRID infrastructure [16].

To provide more efficient support for long-running parallel applications on clusters, MTA SZTAKI have developed new experimental tools in P-GRADE framework with facilities for distributed checkpointing, process migration, dynamic load balancing, fault-tolerance execution and interfacing to job schedulers.

This paper focuses mainly on the debugging issues (see Section 5) of parallel applications following the active control approach, similarly to other state-of-the-art debugging frameworks [13][14][15][17]. The major advantages of the developed system comparing to the related works can be summarized as follows.

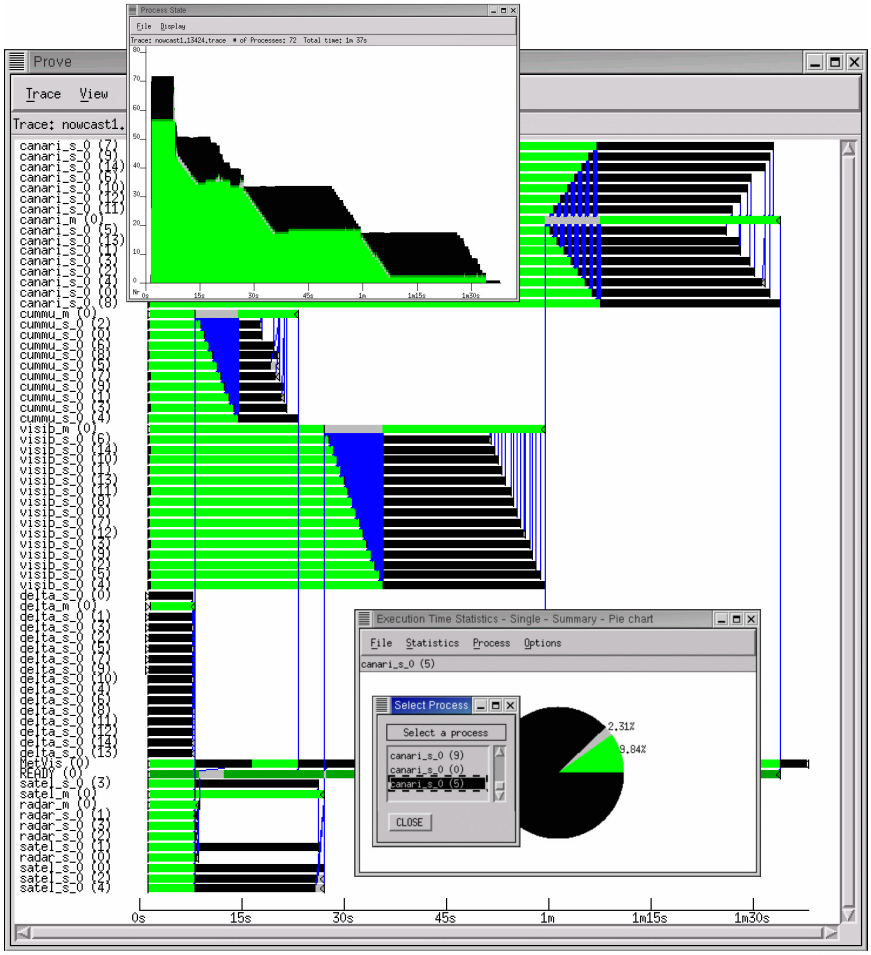

FIG. 6.1. *Performance visualisation with PROVE on SGI*

- Application of temporal logic assertions (instead of traditional predicates) offers more powerful expressiveness for the users to specify correctness criteria, and the universe, where the assertions are evaluated, is the result of real program execution (and not modeling).
- Automatic generation of coloured Petri-net model based on the GRAPNEL program code, which can assist to the user in steering of debugging activity.
- Macrostep-based active control enables the user to debug the parallel application similarly to the step-by-step execution of sequential programs.

However, our experimental debugging framework is strongly tightened to the GRAPNEL graphical language thus, it cannot be applied directly for legacy parallel application contrary to MAD environment [13] or STEPS/DEIPA [14].

8. Summary. As the presented work demonstrates, P-GRADE environment provides easy-to-use solutions for parallel program development even for non-specialist programmers at each stage of software development life-cycle. Hungarian Meteorological Service parallelised successfully the most time consuming parts of its model for ultra-short range weather forecast with P-GRADE system. Hence, the execution time of the entire processing (from the measurement to the issuing of warnings) matches to the 20-minute time constraint in order to predict on time those weather phenomena which might be dangerous for life and property.

The parallel version of the ultra-short range weather prediction system has been demonstrated successfully at the 5th DataGRID conference involving the server of Hungarian Meteorological Service, the PC cluster of MTA SZTAKI, and some mobile devices placed on the site of the conference. We would like to thank József

Kovács and Márk Rajnai for their assistance with various aspects of the successful demonstration.

The presented nowcasting meteorological system provided valuable feedbacks for the further development of P-GRADE, particularly for the experimental debugging framework, which combines ideas from the field of formal methods and model checking techniques with advanced parallel debugging methods in a user-friendly way.

Acknowledgments. The work presented in this paper was supported by the Ministry of Education under Nos. OMFB–02307/2000, OMFB–02313/2000, the Hungarian Scientific Research Fund (OTKA) No. T042459 and IHM 4671/1/2003.

REFERENCES

[1] Á. Horváth: *Nowcasting System of the Hungarian Meteorological Service*, Fifth European Conference on Applications of Meteorology, OMSZ Budapest, 2001. pp. 13.
[2] I. Geresdi and Á. Horváth: *Nowcasting of precipitation type*, Időjárás Vol. 104. No.4. 2000. December, pp. 241–252.
[3] P. Kacsuk, J. C. Cunha and S. C. Winter (editors): *Parallel Program Development for Cluster Computing: Methodology, Tools and Integrated Environments*, Nova Science Publ., New York, 2001
[4] P. Kacsuk, G. Dózsa, T. Fadgyas and R. Lovas: *GRADE: a Graphical Programming Environment for Multicomputers*, Journal of Computers and Artificial Intelligence, Slovak Academy of Sciences, Vol. 17, No. 5, 1998, pp. 417–427.
[5] J. Kovács, P. Kacsuk: *The DIWIDE Distributed Debugger on Windows NT and UNIX Platforms*, Distributed and Parallel Systems, From Instruction Parallelism to Cluster Computing, Editors.: P. Kacsuk and G. Kotsis, Cluwer Academic Publishers, pp. 3–12, 2000.
[6] Z. Tsiatsoulis, G. Dózsa, Y. Cotronis, P. Kacsuk: *Associating Composition of Petri Net Specifications with Application Designs in Grade*, Proc. of the Seventh Euromicro Workshop on Parallel and Distributed Processing, Funchal, Portugal, pp. 204–211, 1999.
[7] J. Kovács, G. Kusper, R. Lovas, W. Shreiner: *Integrating Temporal Assertions into a Parallel Debugger*, Proceedings of the 8th International Euro-Par Conference, Paderborn, Germany, pp. 113–120., 2002
[8] P. Kacsuk: *Systematic Macrostep Debugging of Message Passing Parallel Programs*, Journal of Future Generation Computer Systems, Vol. 16, No. 6, pp. 609–624, 2000.
[9] K. Jensen: *Design/CPN Reference Manual*, Aarhus University-Metasoft,1996
[10] K. Jensen: *Coloured Petri Nets: A High Language for System Design Analysis*, Advances in Petri nets, LNCS 483, 342–416, Springer
[11] L. Lamport: *The Temporal Logic of Actions*, ACM Transactions on Programming Languages and Systems, 16(3):872–923, May 1994.
[12] Z. Manna and A. Pnueli: *The Temporal Logic of Reactive and Concurrent Systems - Specification*, Springer, Berlin, 1992.
[13] D. Kranzlmüller, S. Grabner, and J. Volkert: *Debugging with the MAD environment*, Environment and Tools for Parallel Scientific Computing (editors: J. Dongarra and B. Tourancheau), volume 23 of Parallel Computing, pp. 199–217. Elsevier, 1997.
[14] J. Lourenco, J. C. Cunha, H. Krawczyk, P. Kuzora, M. Neyman, and B. Wiszniewski: *An integrated testing and debugging environment for parallel and distributed programs* Proc. 23rd EUROMICRO Conference, pages 291–298, Budapest, Hungary, 1997. IEEE ComputerSociety.
[15] A. Tarafdar and V. K. Garg: *Predicate control for active debugging of distributed programs*, Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP-98), pages 763-769, Los Alamitos, March 30–April 3 1998. IEEE Computer Society.
[16] I. Foster and C. Kesselman: *Globus: a Metacomputing Infrastructure Toolkit*, International Journal of Supercomputing Application, May 1997.
[17] M. Frey and M. Oberhuber: *Testing and Debugging Parallel and Distributed Programs with Temporal Logic Specifications*, Proc. of Second Workshop on Parallel and Distributed Software Engeneering 1997, pages 62–72, Boston, May 1997. IEEE Coputer Society.

Edited by: Zsolt Nemeth, Dieter Kranzlmuller, Peter Kacsuk, Jens Volkert
Received: April 17, 2003
Accepted: June 5, 2003