MOGUL: A Graphical Environment for Developing the LOGFLOW Parallel Prolog System

Péter Kacsuk, József Kovács and Norbert Podhorszki

Research Institute for Measurement and Computer Techniques of the
Hungarian Academy of Sciences
Konkoly-Thege. u. 29-33, Budapest, H-1121, HUNGARY
kacsuk@sunserv.kfki.hu, http://www.kfki.hu/~mszkihp/info/lpds/

Abstract. The paper describes the MOGUL graphical environment. MOGUL is applied in the design and development of the LOGFLOW parallel Prolog system which exploits inherent OR- and pipeline AND-parallelism of Prolog programs. It helps with editing, compiling and executing Prolog programs. It supports graphically the mapping of a sequential and a parallel Prolog engines to physical processors, the animation of the data driven execution mechanism as well as the animation of load-balancing decisions. These features are used for examination and evaluation of LOGFLOW, supporting the developers to improve the system performance. Besides, MOGUL provides a framework for creating Prolog programs and makes LOGFLOW available for users other than the developers.

1. Introduction

LOGFLOW is a distributed Prolog system running on multi-transputer machines and workstation clusters. The 3DPAM engines (Distributed Data Driven Prolog Abstract Machines) execute the predicates of a Prolog program in parallel based on a fine-grain data driven execution scheme. Coarse-grain pieces of work are executed by traditional sequential WAM (Warren Abstract) machines.

The main design issues in the development of LOGFLOW were as follow:

- correctness of the dataflow execution mechanism
- combining 3DPAM and WAM engines
- compiling Prolog programs into abstract codes both for the 3DPAM and WAM engines
- mapping of 3DPAM and WAM engines into the distributed processor space
- load balancing
- granularity control
- performance measurements

Accordingly to the design issues of LOGFLOW, MOGUL provides a framework for visualizing the relevant features of the parallel system design. There were many visual tools developed for supporting parallel programming ([2], [9], [11], [5], [3], [6]) but only some tools were developed for parallel logic programming ([1], [4], [13]). Since LOGFLOW is implemented on several multi-transputer machines and on workstation clusters as well, it was necessary to create a development system that is able to access remotely any of these hardware platforms. This is one of the main advantages of MOGUL that it releases the programmer of dealing with the necessary file transfers, conversations, etc. during the program development. The user can concentrate on the relevant Prolog related development issues.

There are two directions in developing such an environment as MOGUL. The visualization techniques and the evaluation methods of the results may support the system designers or the Prolog application programmers. System developers need techniques to evaluate the behaviour and the performance of LOGFLOW while Prolog programmers are interested in the sound and efficient execution of their applications. The two directions are not exclusive and an environment can support both groups of users but we had to choose between them because of the available manpower. MOGUL with its present modules is intended to support the developers of LOGFLOW but it can be extended with visualization and debugging modules supporting Prolog programmers in the future.

In the rest of the paper we present the MOGUL environment. Then, we show future directions in improvement of the environment. An existing PC/DOS application, the graphical toolset for transputer network configuration is also described. It can be ported into MOGUL.

2. MOGUL

MOGUL is based on the concept of projects. A project consists of the following steps:

1. Editing the Prolog program
2. Compiling the Prolog program
3. Executing the Prolog program
4. Visualizing the execution of the Prolog program
   a. Animating the dataflow execution
   b. Animating the decisions of the load balancer and granularity controller

These steps are made by a programmer who wants to execute different applications and evaluate the system. Another programmer, who is interested in the development of applications, uses only the first three stages. He would need another evaluation tool to test and improve the application.

Figure 1 shows the main window of MOGUL which is able to handle several projects simultaneously. These projects can be in different stages and MOGUL visualizes their current status. Each project is represented by a project line in the MOGUL environment. A project line contains two information fields and currently five buttons as shown in the figure.

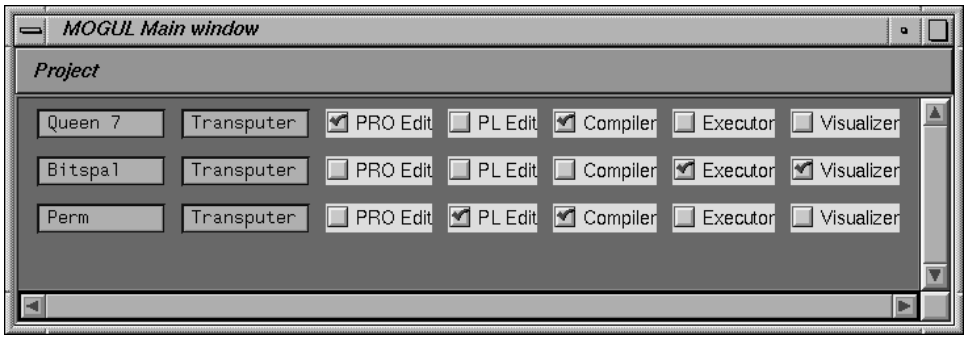

Figure 1. MOGUL Main window

The first information field contains the name of the project while the second one shows the hardware platform where the project is executed. For the time being Transputer based machines and workstation clusters are supported. A joint project with Kyushu University aims at porting LOGFLOW to a multithreaded architecture [8].

The five buttons are used to control the consecutive steps of a project. They are also supplied with an indicator showing if the corresponding step is under execution.

2.1 Editing Prolog programs

The first step in the development cycle is the creation of a Prolog application. Two types of Prolog engines are employed in LOGFLOW: the parallel 3DPAM and the sequential WAM. There are three reasons for separating the source codes written for them.

First, the structure of LOGFLOW enables us to connect different sequential engines to the system. For example, the present multi-transputer system uses an experimental WAM engine while the workstation cluster based system uses the high-performance SICStus Prolog system. It is not ensured, that a source file can be processed by the compilers of different Prolog systems. The 3DPAM compiler is the same for all architecture therefore, we decided to separate the source file from the one written for the actually used sequential Prolog compiler.

Second, the 3DPAM engine allows the user to control the parallel/sequential execution of each predicate. A predicate can be defined to be inherently sequential by an annotation in the source file and the 3DPAM engine will always create a sequential subquery when this predicate is to be executed. These annotations are included in the source code and processed by the compiler.

Third, the connected sequential WAM engine may have special built-in procedures, constants and annotations that are not available in 3DPAM and the 3DPAM compiler would fail to process the Prolog code containing them. However, the user may want to use these features of the sequential system to build an efficient application. In this case we have only to declare the clauses using them to be inherently sequential and exclude them from the source code written for the 3DPAM. The 3DPAM compiler does not need the definition of the sequential clauses, only their names and the arities that are defined in the annotations.

MOGUL can invoke any standard editor available under UNIX in a pop-up window. It is an initialization option of MOGUL to specify the editor the programmer wants to use. The “PRO Edit” button is used to invoke the editor to make the source file for the 3DPAM compiler. The output of the action is a file named after the project and extended with “.pro”. The effect of the “PL Edit” button is the same except that the file extension will be “.pl”. This file is the input of the WAM compiler.

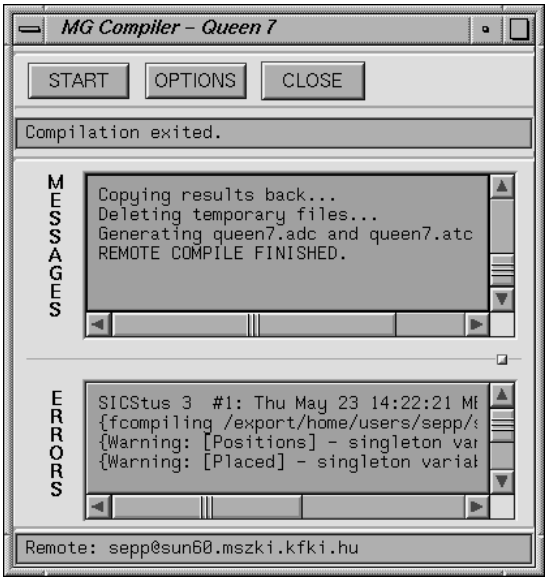

Figure 2 MOGUL Compiler window

2.2 Compilation

The main goal of the compiler window is to control and access the compiler program that can be located on any other machine in the network. Without the use of MOGUL one should

- copy the source files to a remote machine,
- login that machine,
- execute both compilers to create 3DPAM and WAM abstract code (with several additional files),

- execute the program which makes the connection between the two abstract codes (so the 3DPAM engine can do a sequential query for the WAM engine and can convert all the necessary data for this action)
- copy the result files back

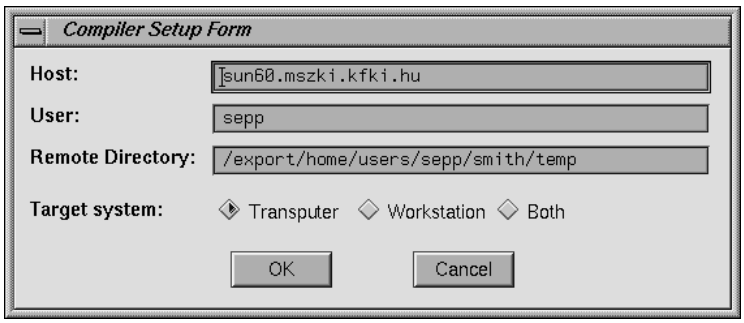

Figure 3 The compiler options window

When starting compilation a new compiler window (see Figure 2) appears. There are two text fields to show the output of the compiler program: error messages are separated from common messages. Two extra information fields are introduced for showing the state of compilation (upper field) and the internet address of the remote machine where the compilation is carried out (lower field).

In the OPTIONS menu the user is allowed to make the following settings, see Figure 3:

- Address of remote machine (Host), where the compilers are placed
- Username for login the remote machine (User)
- Temporary directory in which the files are to be copied (Remote Directory)
- Target system for which the Prolog program is to be compiled (Transputer, Workstation, Both)

The target system should be selected before the compilation and not before the execution. LOGFLOW implementations on different architectures use different WAM engines and therefore, different WAM compilers are invoked.

2.3 Execution

The execution would be also cumbersome without the help of MOGUL. The necessary file should be copied to the remote machine which will start and control the execution (in the case of a multi-transputer machine this is the host computer). After execution the result and monitoring files should be copied to another machine for further work.

The executor window of MOGUL shows the messages of LOGFLOW during the execution (see Figure 4). In the options window the same variables (host, user, remote temporary directory) as at compilation can be set (see Figure 5). Additionally, the types of the channels used in multi-transputer execution (virtual or real) can be determined. Another option is the monitoring. If it is set LOGFLOW collects all the necessary information about the

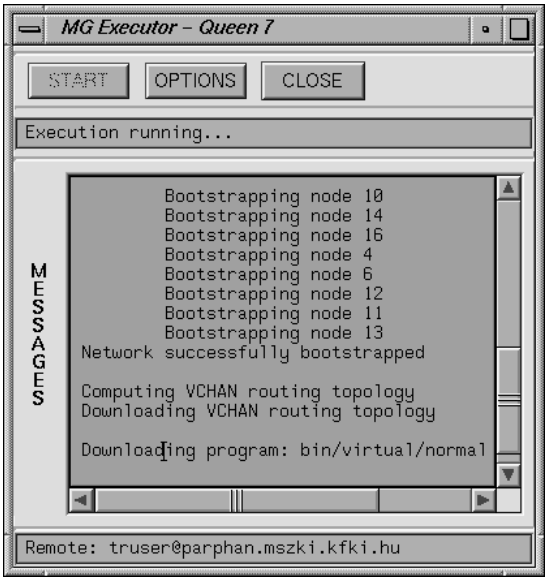

Figure 4. MOGUL executor window

execution that is needed for performance evaluation. Recently, LOGFLOW supports monitoring for load-balancing and for visualizing the dataflow execution. Only the load-balancing work can be visualized in the first version of MOGUL but it is planned to be extended with other visualization tools.

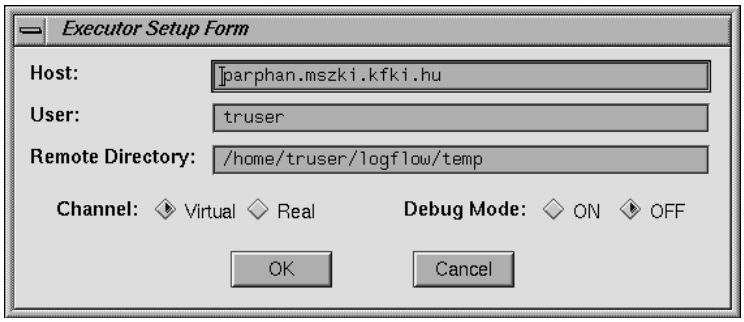

Figure 5 The executor options window

2.4 Visualizing the work of load-balancer

MOGUL supports post-mortem load-balancing visualization. During the execution of a Prolog program the LOGFLOW system is able to collect information about the state of the processors. This information is written into a file after execution has terminated. This method is called post-mortem visualization and was required for analysing the efficiency of the Wavefront Scheduler of the LOGFLOW system [7].
In the load-balancing visualizer there are columns representing the following information (see Figure 7 for the means of the signs):

- position: identifies the processors
- height: shows the quantity of tokens (jobs) on the processor
- colour: determines the type of token (job) on the processor
- pattern: gives information about which engine is currently active

On the MOGUL visualizer window (see Figure 6) menus, drawing area, control buttons, a slider and labels can be distinguished. There are possibilities to load debug files, show legend and view workload parameters in the menus. The user can control the following actions with the control buttons:

- jump to the start
- stop animation
- start animation
- step one message forward

During animation the program executes one step per second. The number of monitoring messages to be processed and animated by a single animation step can be set with the speed slider. Above the drawing area there are two labels containing the actual time position both in terms

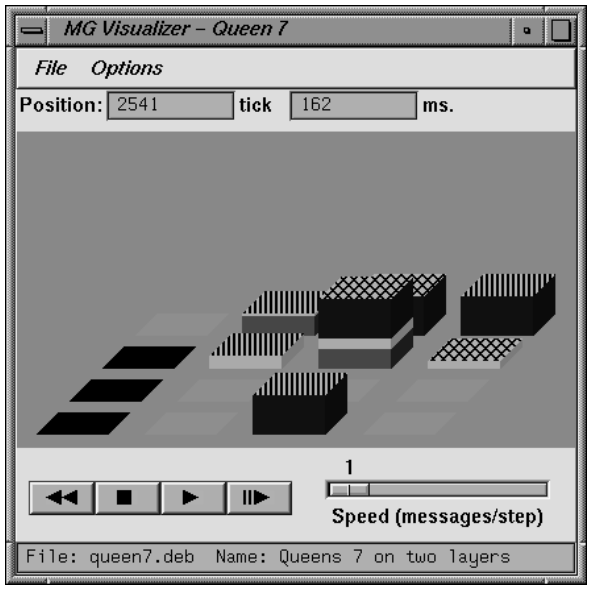

Figure 6. MOGUL visualizer

of clock ticks and in milliseconds. The user can set them by clicking the appropriate fields. When changing one of those values the program automatically jumps to the corresponding point in the profile generated by the monitoring system.

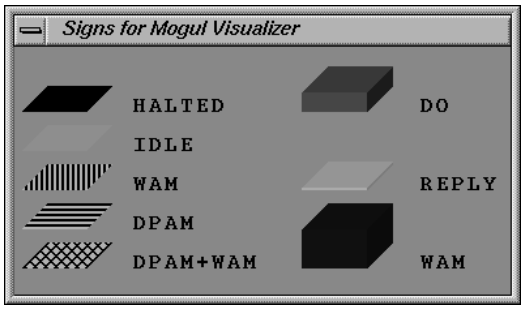

Figure 7 Signs for MOGUL Visualizer

Figure 6 shows a snapshot where 3 of the processors are turned off (black squares) and 5 of them are idle (gray squares). See also Figure 7, where the means of the signs are explained. Idle state means that the processor currently has no tokens to process. Three processors are executing both WAM and 3DPAM code (chequered top of the columns) and 4 processors are engaged in WAM activity (vertical lines on the top of the columns). These 7 processors also have tokens in their token queues waiting for being processed.

3. Other graphical tools for supporting LOGFLOW

3.1 Animation of the dataflow execution

The original idea of MOGUL derives from an MS-Windows software, the 3DPAM-WIN. It was developed earlier for supporting the first version of the LOGFLOW system. 3DPAM-WIN provides a Windows environment that supports the user from the initial editing until the final multi-transputer run

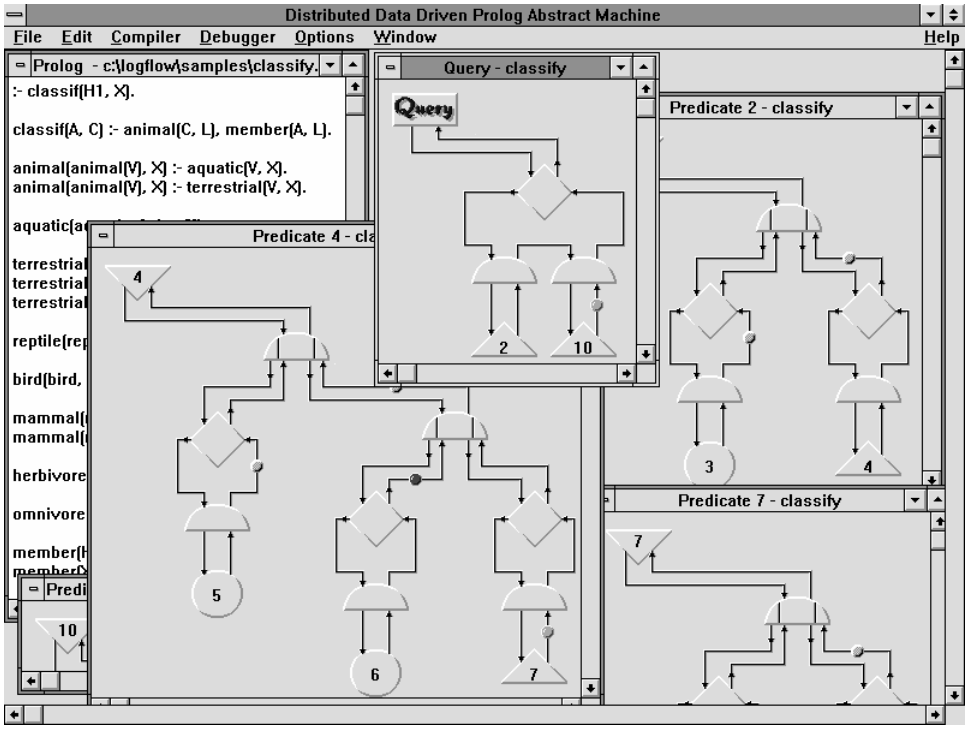

Figure 8. Snapshot of the animation

and animation. This concept was taken to MOGUL where the editor can be chosen from several standard editors and the compilation and execution can be performed by remote computers.

The 3DPAM-WIN was designed for post-mortem debugging facilities. The animation of the execution gives a detailed view of the distributed execution of Prolog programs based on the dataflow principle. If animation is requested the LOGFLOW monitors the events relevant for this visualization and stores them in a trace file. After execution the dataflow graph of the Prolog program is automatically drawn as illustrated in Figure 8. Queries and variable bindings move as tokens in the graph and visualized accordingly to the execution. The animation has two working modes: *movie* and *step-by-step*.

In the movie working mode the animation is running until a breakpoint is met. Breakpoints can be placed on the arcs of the dataflow graph. If a token arrives at a breakpointed arc, the movie working mode is interrupted and the system switches to step-by-step mode. In this mode the user can control the animation of the program execution. Pressing the space bar enables the firing of one node of the dataflow graph. A snapshot of the animal classification Prolog program is shown in Figure 8.

3DPAM-WIN gives a dynamic view (an animation) of the execution while other tools produce only a static trace of it. VisAndOr and Vista are good examples for that class of visualizers, see [1] and [13], respectively.

The existing version of 3DPAM-WIN is used for demonstration and to prove the conception of dataflow execution of Prolog programs in a distributed environment. It was used at the development of the LOGFLOW system. The step by step examination of the (at that time synchronous) execution on many processors could provide information about the sound work of LOGFLOW.

MOGUL will contain a visualization module that works similarly to the animation of 3DPAM-WIN. The original animation can be further developed to provide program debugging. If LOGFLOW collects enough information about the events of the execution, at any step of the animation the user can see the state of the system, for example, the variable bindings of any subgoal executed on any processor.

3.2 Graphical toolset for mapping LOGFLOW

Additionally to the tools of MOGUL, a graphical configurator toolset has been developed for porting LOGFLOW among various multi-transputer machines. This toolset is currently running on PC-s and it will be connected to MOGUL. This module will complete the environment: the configuration of the distributed system can also be done within MOGUL. It means, the user can use MOGUL from the first keystroke to the processing of the results sitting at a computer and he does not need to take care of cumbersome procedures. Of course, the configurator must be redesigned to provide the configuration of clusters of workstations, too. Here, we describe the transputer configurator toolset.

LOGFLOW has been ported to several different multi-transputer based machines and hence to facilitate the porting activity became crucial. We developed a graphical toolset that provides a user-friendly, easy-to-use environment for the system programmer whose task is to port LOGFLOW to various parallel architectures [12].

The execution of a Prolog program by LOGFLOW can be considered in three levels as shown in Figure 9. Abstractly, the program is executed on the network of Prolog Abstract Machines. A machine can have either a 3DPAM or a WAM engine, or both. The layout of the working engines is mapped onto the communication topology of the LOGFLOW system, which defines the neighbourhood and the routing paths of each node. The communication topology is further mapped to the hardware topology.

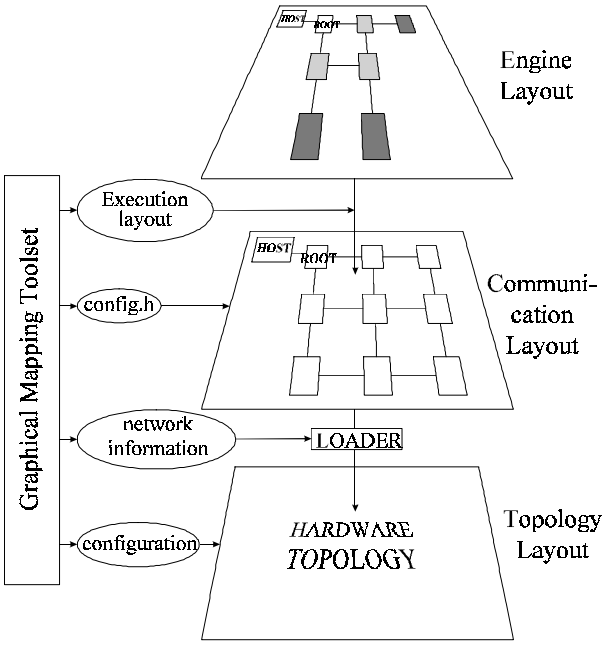

Figure 9. Three levels of execution

Notice that in case of LOGFLOW it is not the user program that should be mapped on to the processor space. The Prolog program is completely loaded to each processing element both in form of parallel and sequential code. The PEs are equivalent concerning *what* they execute. They are distinguished according to *how* they execute the code, i.e. if they employ 3DPAM engines or WAM engines, or both.

The ***lowest level*** (Topology Layout) is independent from LOGFLOW or Prolog. The configuration of the multi-transputer network is a general task. The topology can be defined by several pairs of numbers that form a program for the configurator chip C004 of the hardware. Instead of designing the topology on paper and defining these numbers by hand the graphical tool provides a convenient environment for the configuration.

The ***middle level*** (Communication Layout) is the communication architecture of the LOGFLOW system. The nodes of the architecture are assigned to the transputers of the lower level. LOGFLOW is dependent on the hardware architecture. A node must know which elements are its neighbours, in which directions it must send data towards another node or where to pass the global commands sent by the root processor. All these routing related data are stored in the `config.h` file, isolating them from the other part of the code. Since these data can be derived automatically from the topology, our graphical tool can easily generate them. An extra network information file is necessary to support the downloading of the system onto the network. This file is also generated by the tool.

The ***highest level*** (Engine Layout) is the layout of the Prolog engines on where the actual Prolog application is running. The LOGFLOW logically forms a two-layered architecture. The first layer consists of distributed Prolog abstract machines that are responsible for fine-grained logical computation and distribution of work. The second layer consists of conventional sequential Warren abstract machines that execute the coarse-grained computation. A node of the communication topology can be declared to be either a parallel (3DPAM) or a sequential (WAM) processing element or even it can be turned off.

The graphical toolset consists of four tools. The ***Tr-C004*** program performs the hardware

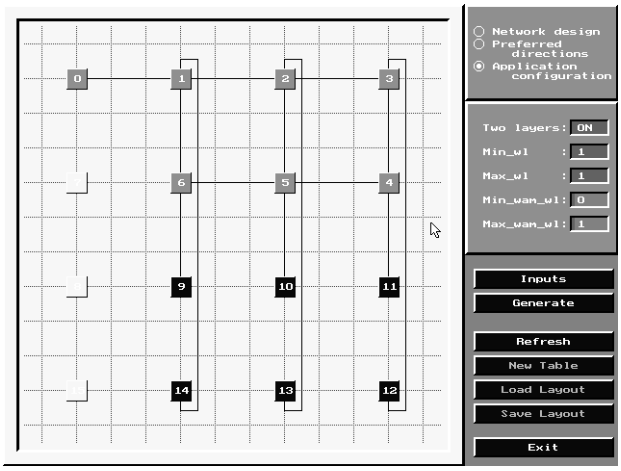

Figure 10. Construction of a physically two-layered LOGFLOW architecture in GR-Conf.

configuration. The tool generates the program for the configurator chip C004. The ***Tr-Loader*** generates the network information file that is used when the program is downloaded onto the transputer network. The ***Tr-Conf*** automatically generates the topology dependent data of the LOGFLOW system. The ***Gr-Conf*** is a graphical tool that enables the user to design a transputer network graphically, integrates the above tools and provides the same graphical environment for mapping the engine layout of the Prolog applications.

By the use of the graphical environment it is convenient to design a new multi-transputer topology and to port LOGFLOW to it. By the use of a mouse the whole configuration task can be completed. We can move the processors on the screen and place them in a grid. The creation or the deletion of a connection takes only two clicks on the appropriate two processors. The generation of the network configuration file costs only a button press. The same graphical environment and actions enable to define the two-layered LOGFLOW architecture as shown in Figure 10.

This graphical environment can also be used for configuring workstation clusters. However, it is simpler work. There is no need for hardware configuration and the workstation version of LOGFLOW needs only a simple description file about the processing elements. The mapping of the engines to the topology is the same as in multi-transputer systems.

Conclusion and further work

We presented the MOGUL graphical environment created for supporting the development and evaluation of the LOGFLOW parallel Prolog system. Taking into account the fact that the LOGFLOW system has been ported many times to several types of transputer machines, we can conclude, that a graphical, user friendly environment like MOGUL

- makes it easier for the programmer to do the porting job
- by its automatic checks (some of them will be integrated in the future) it can detect and thus, avoid some of possible configuration errors
- makes easier to detect other kind of errors (both configuration and programming ones)

Furthermore, the evaluation of the system (and post-mortem debugging, in general) is almost impossible without a good graphical tool, since trace files of several Megabytes must be scanned and analysed. The standalone tools integrated (or going to be integrated) into MOGUL have proved their usability and efficiency already. The MOGUL is a collection of these tools integrating them into a single, user friendly environment. It provides safe data flow between the components and it hides the lowest, hardware related layers from the programmer.

References

[1] M. Carro, L. Gómez and M. Hermenegildo: Some paradigms for visualizing parallel execution of Logic Programs, *Proceedings of the International Conference on Logic Programming, Hungary*, ed. D.S. Warren, MIT Press, pp. 184-201, 1993

[2] T. Delaitre, G.R. Justo, F. Spies, S.C. Winter: A graphical toolset for simulation modelling of parallel systems, *Parallel Computing* 22 (1997) No. 13. pp. 1823-1836.

[3] T. Fadgyas, W. Schreiner: Visualization of parallel programs: The PACVIS visualization tool, *Proc. of the 2nd Austrian-Hungarian Workshop on Transputer Applications*, KFKI-1994-09/M,N Report (Budapest, 1994), pp. 43-61.

[4] Y. Feldman, E. Shapiro: Temporal debugging and its visual animation, *Proc. of the 1991 Intl. Symposium on Logic Programming*, ed. V. Saraswat, K. Ueda, MIT Press, 1991, pp. 3-17.

[5] T. Ludwig, R. Wiesmüller, V. Sunderam, A. Bode: OMIS – On-line monitoring interface specification, LLR-TUM, Technical Univ. of Munich, Germany, and Emory Univ. USA, 1996.

[6] E. Luque, R. Suppi and J. Sorribes: Overwiev and new trend on PSEE, *IEEE Software*, 1992.

[7] P. Kacsuk: Wavefront scheduling in LOGFLOW, *Proceedings of Euromicro Workshop on Parallel and Distributed Processing,* Malaga, 1994. pp. 503-510.

[8] P. Kacsuk, M. Amamiya: Prolog on the multithreaded Datarol-II machine based on the logicflow execution model, *Proceedings of SPDP'96,* New Orleans, 1996

[9] P. Kacsuk , J.C. Cunha, G. Dózsa, J. Lourenco, T. Fadgyas, T. Antao: A graphical development and debugging environment for parallel programs, *Parallel Computing* 22 (1997) No. 13. pp. 1747-1770.

[10] P. Kacsuk, Zs. Németh, Zs. Puskás: Tools for mapping, load balancing and monitoring in the LOGFLOW parallel Prolog project, *Parallel Computing*, 22 (1997) No. 13. pp. 1853-1882.

[11] D. Kranzmüller, S. Grabner, J. Volkert: Debugging with the MAD environment, *Parallel Computing* 23 (1997) pp. 199-217.

[12] N. Podhorszki: Transputer network configuration tools for LOGFLOW, *Proc. of the 1st Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS),* N.Podhorszki, P.Kacsuk, G.Kotsis, G.Haring eds., Miskolc, 1996. pp. 221-222.

[13] E. Tick: Visualizing parallel logic programs with VISTA, *Fifth Generation Computer Systems*, pp. 934-942, 1992