Project Workspaces for Parallel Computing - The TRAPPER Approach

Dino Ahr and Andreas Bäcker

SCAI, Institute for Algorithms and Scientific Computing
GMD, German National Research Center for Information Technology
Sankt Augustin, Germany
{Dino.Ahr|Andreas.Baecker}@gmd.de
http://www.gmd.de/SCAI/

Abstract. PC clusters running Windows NT have been investigated as a low-cost alternative to parallel computers. One important reason why NT clusters are not broadly accepted yet is the lack of powerful development environments, which are a crucial success factor for software development projects. This paper presents the new Windows NT version of TRAPPER, an integrated development and visualization environment for parallel systems. Emphasis is put on a new component called the *project workspace tool*. Project workspaces enable user-friendly interaction with all components and settings required for the development of a parallel application. They automate many repetitive tasks that are typical to parallel system development.

1 Introduction

The widespread adaption of parallel computing has been obstructed due to two major reasons: Parallel hardware is still very expensive and there is a lack of powerful parallel programming environments. The first problem has been tackled by the utilization of networked workstations and personal computers (PC) as low-cost parallel computing platforms. In order to attack the second problem, the message passing platforms PVM [6] and MPI [7] have recently been ported to the Windows NT platform [2, 3].

Message passing platforms are an enabling technology for parallel computing, but additional tools are required to develop parallel software efficiently. Of particular interest are integrated development environments (IDE) which support all activities in the development process. Up to now, neither one of the development environments available for UNIX has been ported to Windows NT, nor have any new environments been developed.

In the first part of this paper, we present TRAPPER [11], an interactive development and visualization environment for parallel computing on Windows NT. TRAPPER has been ported to Windows NT, redesigned and improved in the context of the WINPAR[1] project [1]. Several new features have been added

[1] The WINPAR (Windows-based Parallel Computing) project is supported by the European Union under ESPRIT IV project No. 23516.

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 98–107, 1999.

and the graphical user interface (GUI) has been re-engineered to meet current standards of the Windows platform.

A very important requirement to an IDE on the Windows NT platform is user-friendliness, because demands from users being accustomed to carry out most of their daily work using interactive tools are very high. There exists a variety of development environments for parallel computing, but only few attention has been payed to the aspect of user-friendliness.

To enhance the user-friendliness of TRAPPER, a new component called the *project workspace tool* has been developed. The project workspace tool is the central component of TRAPPER: On the one hand, it offers a structured view on all tools, settings and files belonging to a software project. On the other hand, it serves as a steering and navigation tool which allows to launch tools, to trigger actions like the compilation process or to modify project settings. Moreover, the project workspace tool automatically cares for complete and consistent data and settings and performs a kind of garbage collection for files that are no longer needed. These features allow for an efficient cycle of work and relieve the programmer from many complicated and repetitive tasks.

In the next chapter we give a brief summary of existing development environments for parallel computing. The third chapter gives an introduction into TRAPPER. Chapter 4 describes the project workspace tool in-depth. The last chapter contains the conclusion and outlines future work on the project workspace tool.

2 Related Work

In the following we describe graphical software development environments for message-passing architectures. We refer to [13] for a detailed review of parallel programming tools and environments.

HeNCE (Heterogeneous Network Computing Environment) [4] is an X-based environment designed to assist in developing parallel programs on top of PVM that run on a heterogeneous networks of UNIX computers. HeNCE is composed of integrated graphical tools for creating, compiling, executing, animating and analyzing programs. HeNCE describes the behavioral aspects of the computation via a directed acyclic graph whereas TRAPPER describes the structural aspects of the application.

The GRADE (Graphical Application Development Environment) [8] environment provides tools to construct, execute, debug, monitor, and visualize PVM programs. The structure of the parallel application is specified down to the level of individual communication operations and is hence much more detailed than in TRAPPER. A distributed debugging engine assists the user in debugging programs on distributed memory computer architectures. The monitoring tool TAPE/PVM [10] and the visualization tool PROVE support performance monitoring and visualization of the program behavior. GRADE and TRAPPER are very similar except for that GRADE provides a distributed debugger and has a more elaborated graphical editor.

The EDPEPPS (Environment for the Design and Performance Evaluation of Portable Parallel Software) [5] environment comprises integrated tools for graphical parallel software design and mapping (PVMGraph), monitoring (TAPE/PVM), CPU modeling (cputime), simulation (SES/Workbench) and visualization, animation and analysis of program execution and predicted results (PVMVis). The EDPEPPS tool-set is targeted for a heterogeneous network of workstations and for the PVM parallel programming model. The main feature of this environment is the rapid prototyping approach to parallel software development, since the graphical editor together with the simulation tools allows performance analysis to be done without accessing the target platform.

The Tool-set [12] environment for PVM consists of a set of integrated tools which can be divided into automatic tools, e.g. the parallel file system, a checkpoint generator and a load balancer, and interactive tools with a GUI based on OSF/Motif, e.g. a source level debugger, a performance analyzer, a program flow visualizer and a deterministic execution controller. The core of the system is an OMIS (On-line Monitoring Interface Specification) [9] compliant monitoring system (OCM) for the PVM programming library on networks of workstations. *The Tool-set* environment focuses on on-line analysis and debugging of the parallel application and provides powerful and comprehensive facilities for this. It does not cover the graphical specification of a parallel application (along with code generation).

The environments discussed above all support PVM and run on UNIX. TRAPPER differs from them in that it supports both PVM (3.3) and MPI (1.1), and runs on Windows NT. Moreover, TRAPPER has been implemented on top of a portable GUI library and can be ported to UNIX with little effort. The project workspace tool is unique to TRAPPER.

3 The TRAPPER Visual Programming Environment

Figure 1 shows the tools of the TRAPPER development environment. Both, the parallel application and the target hardware, will be specified graphically with the *design manager* and the *graph editor*. The parallel application is represented by a process graph where nodes are processes and edges are message passing communication channels. TRAPPER generates code-skeletons for each process; the user only has to insert code for the process' functionality. The *mapping tool* determines the assignment of the processes to hardware nodes. The *platform settings* comprise paths and flags for compiler, linker, communication library and make tool available on the target hardware. This information is required for *generating* (by a click of a button) makefiles and configuration files which allow to *compile* and *run* the parallel program. The *monitoring settings* decide whether and how the application should be monitored (on-line/off-line). Trace data from the application, the communication platform and the hardware is collected by the *monitoring system*. For debugging and optimization purposes this data can be visualized and analyzed with the *visualization tool*. All tools are embedded

in the so-called *project workspace tool*, which will be covered in-depth in chapter 4.

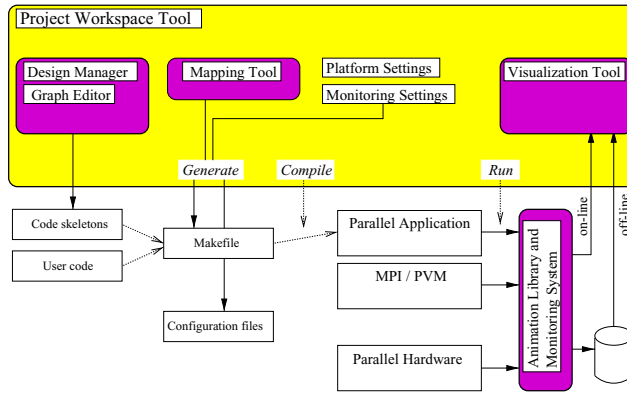

Fig. 1. The TRAPPER architecture

3.1 Designs

The central concept of TRAPPER is a *design*. A design is a graphical specification of a static process net or hardware configurations. Designs are created and modified using the *graph editor*. A design contains rectangles for its subnodes, ports denoting input/output channels and network interfaces, and links for connecting ports to nodes. Busses are used for multiplexing links. They denote group constructs in MPI and PVM program specifications and networks in hardware configurations.

Although the effort needed to specify a parallel system graphically introduces some overhead in early stages of the system development cycle, the graphical specifications gain a high value when the parallel system has to be animated. The node rectangles can be used to display the state of a process or a processor as well as to display contents of process variables and arrays. Links and ports can be highlighted to display communication activities. Since visualization tools are very helpful for debugging and performance tuning, the benefits obtained in later development stages outweigh the initial specification effort.

3.2 Software Development

TRAPPER supports a hybrid software development approach: The parallel structure of the application is described graphically by a process graph, whereas the sequential process code has to be written by the software developer. Each process design has an associated code file. Additional source code files can be associated with the whole application. TRAPPER simplifies coding of a process' code by generating code skeletons, which can later be extended.

3.3 Hardware Design

A graphical hardware design can be composed out of four different types of nodes: hardware clusters, single-processor computers, multi-processor computers and CPUs. These four basic types allow the user an exact and easy description of the hardware.

3.4 The Mapping Tool

The mapping tool performs the assignment of the application processes to nodes of a desired hardware design. TRAPPER provides automatic and manual mapping. The heuristic mapping algorithm computes a partitioning of the process graph with a well distributed computation load and a small communication load between partitions. The computation and communication loads are gained from the attributes of the software and hardware nodes.

3.5 The Monitoring System and the Animation Library

The monitoring system gathers run-time trace information about the parallel application. Trace events can either be stored in a binary trace file and then be analyzed retrospectively (*off-line*), or the events can be analyzed *on-line* while the parallel application is running.

On-line monitoring is realized via portable TCP/IP socket library calls. The benefit of using portable socket library calls is that a Windows NT workstation running TRAPPER can be used as a visualization front-end for a parallel application that runs on an arbitrary remote hardware.

The monitoring system is implemented as a C function library - the *TRAPPER animation library* - which has to be linked with the application. A language binding for Fortran is available. TRAPPER automatically instruments MPI and PVM applications by wrapping MPI and PVM function calls.

In addition to tracing of communication and process events, the animation library supports tracing of scalar, one-dimensional and two-dimensional program variables.

3.6 The Visualization Tool

The visualization tool supports the developer in understanding, debugging and optimizing his or her application. It displays and animates trace data generated through calls to the animation library. It uses the node rectangles in the graphical software and hardware designs to display state information and views of program variables (fig. 2). The visualization tool also provides time axis diagrams for process states, communication events, communication statistics, scalar program variables, and critical path analysis.

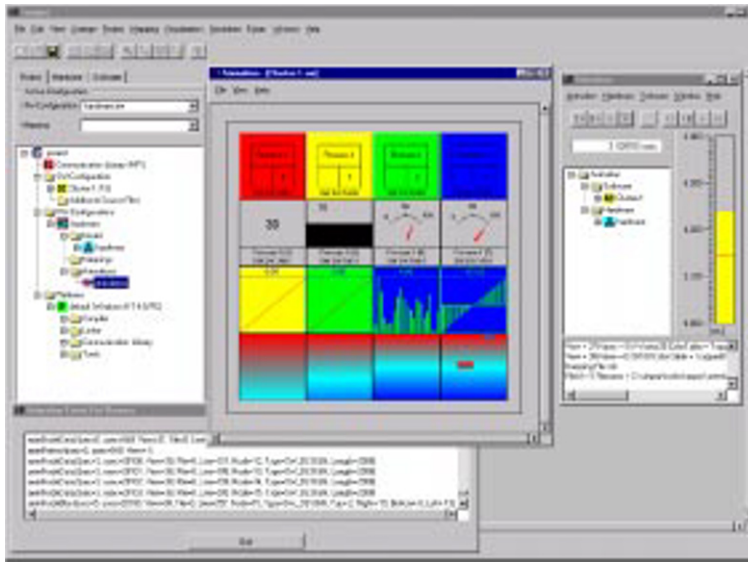

Fig. 2. TRAPPER screen-shot: The project tree (left) and the visualization tool (center)

4 The Project Workspace Tool

4.1 Motivation

Each tool in a development environment works upon data which are generally stored in files. For an environment like TRAPPER there are software and hardware design files, mapping files, source files, makefiles, configuration files for platforms and communication libraries, and animation files. A typical problem is to create an appropriate directory structure and to maintain all files according to this structure. Differentiation of file types is solely accomplished via file suffixes and it is sometimes difficult to bear in mind which files belong together, e.g. which software and hardware model files belong to a mapping. Furthermore the deletion of invalid or obsolete files and data is cumbersome and might be risky because valid files could be removed accidentally.

Another aspect concerns the maintenance of several similar program execution scenarios. For example, the developer tries different mappings or different target platforms to optimize the performance of an application. This could result in a lot of different mappings, configuration files, platform settings and associated animation files which are difficult to manage.

Furthermore, it is very convenient for a developer if the current state of work in the environment (e.g. the tools opened, the settings performed, the windows positioned on the screen, etc.) could be saved and restored later. This allows the developer to continue immediately with his work at the next session instead of configuring everything new.

4.2 Concept

The project workspace tool acts as a shell around the other tools of the TRAPPER environment. It has *structuring*, *visualizing* and *interactive* aspects.

As a fundament we designed a logical structure for the files and settings that are part of the TRAPPER development process. All these files and settings - inherently organized according to this structure - constitute the so-called *project workspace*. For each development project the user will create a separate project workspace. The project workspace tool cares for *consistency* and *completeness* of all elements added to the workspace. Moreover, the project workspace can be made persistent, i.e. the current state of work within the project workspace can be saved to disk and easily restored for the next session.

The visualization aspect of the project workspace tool is realized by a tree gadget[2] GUI - called the *project tree*. Within the project tree each item contained in a project workspace will be displayed as a tree node. The hierarchy of the structure is mirrored by a parent-child relation between the corresponding nodes. Hence all files and settings are ordered visually and can easily be found and accessed by the user. A design/sub-design relation on design level will be mapped by a parent-child relation of the corresponding nodes in the project tree. This technique allows to capture particularly complex designs with many nodes and hierarchy levels. The directory structure of the files contained in the project tree is hidden; the user doesn't have to care where a file is stored. Since folder nodes can be closed or opened the contents can be hidden respective shown allowing the developer to concentrate on the parts he or she currently works upon.

For instant access each node of the project tree provides context sensitive interaction facilities through a context menu.

We will now present the structural aspects of the project workspace and its visualization and interaction facilities within the project tree. At the same time we will discuss a sample application, the *Poisson problem* taken from [7], in order to illustrate the features and benefits of the project workspace tool.

4.3 The Project Workspace and the Project Tree

The general structure of a project workspace is depicted in fig. 3. Figure 4 shows the project tree for the Poisson problem example.

Communication Library. At the beginning of the development process a basic decision concerning the *communication library* has to be made by the developer. This could be either MPI or PVM. The example application which solves the Poisson problem will be implemented on top of MPI.

[2] A tree gadget is a GUI element consisting of *folder* nodes and *leaf* nodes arranged as a tree. Each node has an icon and a label. Folder nodes can contain further folder and leaf nodes.

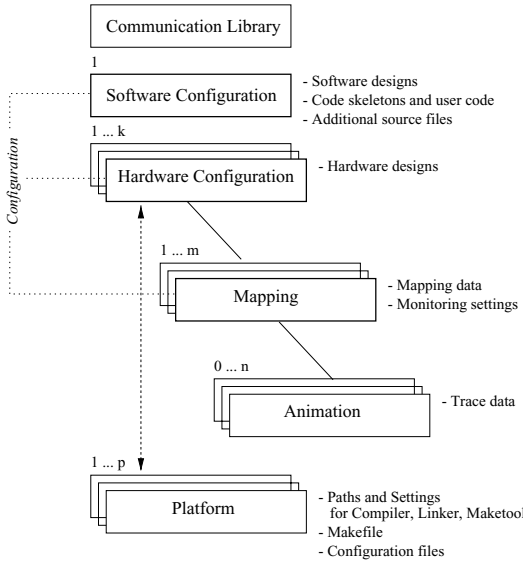

Fig. 3. Structure of the project workspace

Fig. 4. Screen-shot of the project tree

Software/Hardware Configuration and Mapping. The *software configuration* comprises software designs of the parallel application and associated code skeletons including inserted user code. In addition, it contains source files which are attached to the application. In our example the software design which represents the whole parallel application is called `PoissonProblemSolver`. It is composed of five `JacobiSolver` processes, programmed in C.

In order to execute the parallel application on different available hardware platforms there can be several *hardware configurations*. Each hardware configuration includes hardware designs and an arbitrary number of *mappings*. This enables the user to evaluate different mapping strategies.

In the above example we consider two different target platforms, named `Local` and `NT-Cluster`. The `Local` platform contains one single-processor computer and is used for implementation and testing the application. The `NT-Cluster` hardware consists of three single-processor computer nodes and is used in the stage of performance optimization. All nodes are attributed with Windows NT 4.0 as their operating system and Pentium as their cpu. For our application we focus on mappings for the hardware configuration `NT-cluster`. We create two mappings `mapping-lowCom` and `mapping-loadbalanced` which concentrate on minimizing the communication between the processors respective ensuring a good load balancing.

Configuration. A hardware configuration and a mapping define a unique execution scenario for a parallel application in the project workspace. We denote

such a pair as a *configuration*. The actions *Generate*, *Compile* and *Run* (fig. 1) are only available when there is at least one configuration. In general there are several configurations contained in a project workspace. Hence there has to be a particular configuration, called *active configuration*, to which these actions refer.

We assume that we want to solve the Poisson problem on the `NT-cluster`. Because communication is a bottleneck for connected PCs we choose the mapping `mapping-lowCom` for our purposes. Hence we have to choose the pair (`NT-cluster`, `mapping-lowCom`) as active configuration.

Since a mapping component identifies a configuration it is used as a container for monitoring settings and *animation* components. The monitoring settings influence whether and how the application generated for this configuration should be monitored. Each animation component contains trace data of an application run made for this configuration. Several trace files can be managed to compare application runs.

In order to obtain trace files we switch on the off-line mode in the monitoring settings of the active configuration (`NT-cluster`, `mapping-lowCom`). The *Generate* action produces a makefile and a MPI process group file. Via the *Compile* action the makefile will be processed by the make tool resulting in an executable which can then be started with the *Run* call. For the program run the monitoring system produces the trace file `poissonAnim`.

Platform. A *platform* is defined by the processor type and the operating system of a hardware design. For each hardware design contained in the project there exists an associated platform dataset which comprises settings and paths for compiler, linker, communication library and make tool. This data is required for the generation of makefiles and configuration files. TRAPPER provides settings for each platform in a *defaults database*. The defaults database eventually has to be adapted once by the system administrator of a target platform; after that the user doesn't have to care about these issues.

Consistency and Completeness. The *consistency* mechanism of the project workspace tool cares for deleting invalid data. For example, if the software configuration has been modified because a process node has been added/removed all existing mappings will become invalid. Hence all mapping data including animation data will be deleted automatically to ensure the consistency of the project workspace.

Completeness checks of required settings will be performed by the mapping tool and the generation process. In case of failure the user is led directly to the concerned dialog.

5 Conclusion

We have described the concept and the implementation of project workspaces for the development of parallel applications. Experiences have shown that the

project workspace tool is a suitable tool for speeding up application development. For people who are new to an already running development project, project workspaces can be a valuable aid in discovering content and structure of the project. TRAPPER is therefore an ideal supplementary tool for education in parallel programming.

References

[1] D. Ahr, A. Bäcker, O. Krämer-Fuhrmann, R. Lovas, H. Mierendorff, H. Schwamborn, J. G. Silva, and K. Wolf. WINPAR - Windows based Parallel Computing. In E. H. D'Hollander, G.R. Joubert, F. J. Peters, and U. Trottenberg, editors, *PARALLEL COMPUTING: Fundamentals, Applications and New Directions*, pages 495–502. Elsevier, Amsterdam, The Netherlands, 1998.

[2] A. Alves, L. Silva, J. Carreira, and J. G. Silva. WPVM: Parallel Computing for the People. In *Proceedings of HPCN'95*, pages 582–587, 1995.

[3] Marc Baker. MPI on NT: The current status and Performance of the available environments. In *EuroPVM/MPI'98*, Liverpool, UK, September 1998.

[4] A. Beguelin, J. Dongarra, G. A. Geist, R. Manchek, and K. Moore. HeNCE: A Heterogeneous Network Computing Environment. *Scientific Programming*, 3(1):49–60, 1994.

[5] T. Delaitre, M. J. Zemerly, P. Vekariya, G. R. Justo, J. Bourgeois, F. Schinkmann, F. Spies, S. Randoux, and S. C. Winter. EDPEPPS: A Toolset for the Design and Performance Evaluation of Parallel Applications. In D. Pritchard and J. Reeve, editors, *Euro-Par'98*, pages 113–125, September 1998.

[6] A. Geist, A. Beguelin, J. Dongarra, J. Weicheng, R. Manchek, and V: Sunderam. *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[7] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, 1994.

[8] P. Kacsuk, J. C. Cunha, G. Dózsa, J. Lourenço, T. Fadgyas, and T. Antão. A Graphical Development and Debugging Environment for Parallel Programs. *Parallel Computing*, 22(13):1747–1770, February 1997.

[9] Thomas Ludwig and Roland Wismüller. OMIS 2.0 – A Universal Interface for Monitoring Systems. In M. Bubak, J. Dongarra, and J. Waśniewski, editors, *EuroPVM/MPI'97*, pages 267–276, November 1997.

[10] E. Maillet. TAPE/PVM: An Efficient Performance Monitor for PVM Applications - User Guide. WWW: `ftp://ftp.imag.fr/pub/APACHE/TAPE/`, March 1995.

[11] L. Schäfers, C. Scheidler, and O. Krämer-Fuhrmann. TRAPPER: A Graphical Programming Environment for Parallel Systems. *Future Generations Computer Systems*, 11(4-5):351–361, August 1995.

[12] R. Wismüller, T. Ludwig, A. Bode, R. Borgeest, S. Lamberts, M. Oberhuber, C. Röder, and G. Stellner. The Tool-Set Project: Towards an Integrated Tool Environment for Parallel Programming. In X. De, K. E. Großpietsch, and C. Steigner, editors, *APPT'97*, pages 9–16, Koblenz, Germany, September 1997.

[13] M. J. Zemerly, T. Delaitre, and G. R. Justo. Literature Review (2), EDPEPPS EPSRC Project (GR/K40468) D6.2.2, EDPEPPS/32, Centre for Parallel Computing, University of Westminster, London. WWW: `http://www.cpc.wmin.ac.uk/~edpepps/reports/edpepps32.ps.gz`, July 1997.