The Warp programming environment

by B. BRUEGGE, C. H. CHANG, R. COHN, T. GROSS, M. LAM, P. LIEU, A. NOAMAN, and D. YAM
Carnegie-Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

This paper describes the environment for developing and executing Warp* programs. The center of the program development environment is a customized shell that ties together a compiler for the Warp array, the Warp run-time system, and a debugger. The compiler translates high-level language programs to microcode for the Warp machine. It achieves a high utilization of the computation power of the processor. The run-time system supports remote execution of Warp programs across a network and makes the Warp machine available as a shareable resource. The debugger permits symbolic debugging of Warp programs. The Warp programming environment makes the Warp machine an easily programmable and accessible attached processor in a UNIX™ environment.

* Warp is a service mark of Carnegie Mellon University.

INTRODUCTION

In our programming environment, Warp is modeled as an attached processor accessible from an interactive, programmable, command interpreter called the Warp shell. The shell provides traditional operating system commands as well as commands to execute programs on the Warp machine. Calling a Warp program is similar to invoking a procedure: the shell calls the Warp program and passes input and output data between the application and Warp. The run-time system provides low-level support such as securing exclusive access to the machine, downloading object code, and transferring data between the host and the Warp system.

For programming the Warp, we have designed a language called W2 and implemented an optimizing compiler. The programming model, as supported by the language, allows the user to see the machine as a linear array of sequential processors and hides the low-level details from users. From a W2 program, the compiler generates microcode for the Warp array and the interface unit, as well as C programs for the I/O processors.[1]

In this paper we first describe the objectives of the Warp programming environment (WPE), and the system configuration. Then we describe the two methods for using the Warp system. The primary method is the interactive mode through the Warp shell; a library of existing Warp routines as well as user programs can be invoked interactively through shell commands. Program development is done almost exclusively with this method. The second method, used mainly for real-time systems, is the direct mode, for users who cannot afford the overhead of an interactive system. We then describe the support software in WPE: the run-time system, compiler, and debugger. We conclude with a review of the current status and a brief discussion of our experience to date.

Objectives of WPE

The primary objective of WPE is to simplify the use of the Warp machine. WPE is a uniform environment to edit, compile, debug, and execute W2 programs. Its audience includes the user who calls routines from a W2 library, the programmer who develops new algorithms for Warp, as well as the implementor who writes support software.

WPE must support efficient multiple user access because the use of the Warp hardware in a typical user session is sporadic. By allowing multiple user sessions to overlap and by serializing the use of the hardware, the Warp machine can be better utilized. WPE also provides multiple machine access; if there is more than one Warp array available, a user has the choice of connecting to any of these machines. It also provides network transparency, the user sees no difference whether he uses the Warp array remotely from his personal workstation or logs in directly to the Warp host machine.

WPE is designed to be development machine-portable. The shell, compiler, and debugger are written in Common LISP, which runs on many workstations, and the TCP/IP protocol is used in inter-machine communication. Our current release of WPE runs on SUN-3 under BSD UNIX 4.2. WPE is also designed to be target machine-portable. It has been in use for our prototype system, and it can be used with the successor Warp architectures: the production architecture implemented with printed circuit boards as well as the VLSI Warp which is currently in the design stage.

System Configuration

Figure 1 shows the configuration of WPE. Each workstation, a SUN-3, runs one or more *Warp shells*. The workstations communicate with a machine called the *Warp host*. This is another SUN-3 which is physically connected via a bus repeater to the external host and Warp array.[2] The Warp server executes on the Warp host and is the intermediator between users and the Warp array and external host.

TWO MODES OF ACCESSING WARP

There are two methods of running programs on Warp. Users may use the Warp shell which provides an interactive interface to the constituents of WPE such as the compiler, run-time system, debugger, and servers. Or, if absolute performance is necessary, users may program the machine in direct mode, without the overhead of a command interpreter.

The Warp Shell

The Warp shell binds together the components of WPE. Shell commands can be used to invoke the compiler, run a program on the array, and call debugging functions. The Warp shell is based on an extensible shell written in Common LISP.[3] The extensibility makes it possible to support different classes of users. Specifically, the Warp shell distinguishes between the novice and the experienced user. For example, the implementation language Common LISP and the components of the environment are completely hidden from a novice. This is useful for programmers interested in using the Warp shell to execute W2 programs from a library. On the other hand, the LISP implementation and all the software components comprising the Warp environment are easily available when de-