External Product Specification for the iWarp Programmed Communication Services (PCS) Array Loader

Thomas Gross and Susan Hinrichs
August 1991
Version 2.0

Abstract

The PCS array loader loads and executes array modules created by the iWarp PCS array combiner on a physical iWarp system. The array loader also performs system specific resource checks and creates network initialization tables based on information in the array module.

This document describes the functionality of the array loader, how it is used, and how it fits in the program development tool chain.

1 Introduction

This document is the External Product Specification (EPS) for the iWarp array loader **iwload**. This document describes the general behavior of the array loader, specifies how the array loader interacts with the environment of the machine it is executed on, and how the user interacts with the array loader.

An iWarp system consists of one or more iWarp cells; each iWarp cells consists of an iWarp component and some local memory. The iWarp component provides sophisticated support for inter-processor (cell-to-cell) communication. We assume the iWarp system is arranged in a regular $m \times n$ torus.

The array loader executes as part of the program development tool chain. It is executed on the host system and loads cell programs onto the iWarp array.

1.1 General array loader functionality

An array module is analogous to an executable file in the uniprocessor world. It represents the compiled version of a programmers multi-cell array program. The array module contains code for single cell programs, descriptions of program connections, and communication resource reservations. The array loader manipulates the array module, so the array program it represents is loaded and executed on the specified iWarp array. Array modules are created by the iWarp array combiner, which is described in a separate EPS.

All iWarp arrays are not configured identically. Some arrays have different number of cells, different amounts of memory, or different auxiliary cells. Checking for physical array specific details is delayed until the array loader. The array module was created by the combiner with a general logical array in mind. The array loader tries to map this logical array to the target physical array and exits with an error if this mapping is impossible.

iwload links the necessary system libraries with the cell programs and creates communication initialization tables for each cell from the information in the array module. Then it passes the completed cell programs to the lower level cell loader **iwml**.

1.2 Array loader's place in the iWarp tool chain

The program tool chain includes all software tools that aid in the translation and execution of programs for the iWarp system. There are five distinct steps that have to be taken to transform a problem specification (of an application problem) into a set of object codes executing on an iWarp system.

We assume the problem to be mapped onto an iWarp system is initially specified without reference to the specifics of the target array. That is, the problem assumes a single thread of control and a single data space; there are no explicit communication operations. From this, a set of programs, one for each cell, must be generated. At the time these individual programs are generated, the computation and the data are distributed over multiple cells, and appropriate communication statements for data exchange and synchronization must be created. Such translation can be done by a *parallel program generator*, a program that produces a parallel program for execution on the target array, or the translation can be done by a programmer. Parallel program generators are important to make the programming of parallel systems practical, and several parallel program generators have been build at Carnegie Mellon, and one of them (Apply) has been retargetted and ported by Intel Corp. to iWarp. A program for a single cell is called a *cell program* and is written in C or any other language for which there are iWarp compilers.

Cell programs use *ports* to identify connections between different cell programs. A port is a local name, in a single cell program, that refers either to a incoming or outgoing pathway, as depicted by Figure 1. In a later step, the information about port connectivity (port `xout` of cell (i,j) is connected to port `xin` of cell (i, j+3)) is used to determine which pathways are needed for this application. Cell programs can use the PCS data movement library to access the communication networks through these ports.

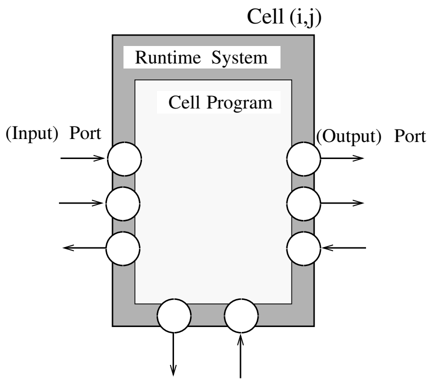

Figure 1: Single-cell program with ports

The second step of translating a collection of cell programs for execution on an iWarp array is *compiling*. Each cell program (for a single cell of the logical array) is compiled with a suitable compiler. Note that this compiler is a 'single-cell' compiler; that is, the compiler has no knowledge of the final placement of the cell program in the array. This compiler does not deal with the interactions of one cell program with another cell program in the array. A side effect of this model of translation is that the same object code (produced by the local compilation) can be used in various scenarios, e.g. as part of a 3×3 as well as a 4×4 application, provided that the same cell program can be used. This is quite useful when creating libraries, etc.

The third step is *linking* the compiled cell programs. For each cell program, unresolved names to external objects (functions, variables) are taken care of. Note that cell programs can only refer to objects on the same cell, so this step is no different than the linking step usually included in compiling for a conventional uniprocessor. The linker takes care only of intra-cell references. As in any other tool chain, the linker can be invoked multiple times.

The iWarp communication agent provides for a variety of connections and communication paths between individual cells. The set-up of these connections requires the cooperation of all cells in the array. So the next step is to produce the initialization code from the user's description of the communication paths in the machine. This step is called *array combining* because at this time the individual cell programs (which so far are unrelated) are combined into an array module. The functionality for this step is provided by the iWarp combiner **iwcb**, which is discussed in this document.

Figure 2 shows a simple example of three cell programs that have been combined. Notice that there is a non-neighbor communication connection between the first and third cell.

Multiple cell programs can be combined to create an *iWarp array module*. In practice, it makes no difference if the cell programs that are combined into a module have been written 'by hand' or have

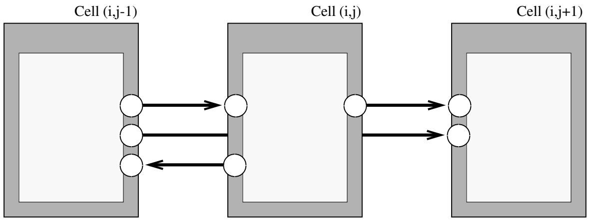

Figure 2: Multiple cell programs combined

been generated by a parallel program generator; they are combined the same way. The only difference is that if a parallel program generator is used, the parallel program generator will produce the input for the combiner as well.

The last step is process *activation*. This step includes the assignment (placement) of cell object code onto physical iWarp cells, relocation of the object code (if needed), and any linkage with the runtime system that is necessary to provide input/output capabilities to the application executing on the cells, as well as the downloading of the program codes (and initialization of memory, communication agent etc. as appropriate). At that time, the program is ready to execute. The iWarp array loader takes care of process activation.

Figure 3 illustrates the sequencing of the various translation steps that turn a collection of cell programs into an array module that can be executed on an iWarp array.

1.3 Scope, purpose, and background of this document

This array loader EPS does not repeat any information about the single-cell tool chain. Information about these tools can be found in the EPS of each tool.

The array loader assumes a static view of the iWarp array. That is, during the execution of a program on the array, the number of cells, the properties of each cell, and the physical connections between the cells are constant. No re-mapping of an application program takes place during the execution of an application program. Future software tools may address the needs of fault tolerant systems that have to move computations and data from one cell to another.

This EPS tries to follow the EPS of the array combiner where possible and borrows some text and organization from the array combiner EPS.

1.4 Product identification

The iWarp array loader **iwload** executes on a SUN host and loads programs and data on the iWarp array. The **-V** option of the array loader forces the loader to emit a line with the current release number, thus disclosing the exact identity of the loader in use. A preliminary version of the array loader has been in use at Carnegie Mellon since October 1990.

1.5 Assumptions about iWarp systems

The array loader assumes it is working on an iWarp array configured as a $m \times n$ torus with one or more auxiliary cells attached to the edges. The loader assumes it can get data about the configuration of a particular array from a configuration data base.

The array loader also assumes the existence of a runtime system. This runtime system provides services to load data onto individual cells and to return results from the cells to the host.

1.6 Assumptions about the host system

The iWarp array loader invokes various parts of the iWarp program development tool chain. Therefore, the complete iWarp tool chain must be available on the machine on which the combiner is executed. In addition, the iWarp array loader may use the Unix tool **make** to intelligently link cell programs and create initialization tables. Therefore, the existence of **make** on the host is assumed.

1.7 Functions not described or provided

This EPS does not describe the iWarp array combiner, which interacts closely with the iWarp array loader. The EPS of the array combiner is described in a separate document.

1.8 Development phases

Releases of the iWarp array loader are synchronized with the development of the iWarp systems.

1.9 Relation to other iWarp software

Figure 4 shows how the array loader **iwload** relates to other iWarp program development tools.

1.9.1 iWarp array loader and iWarp archiver

iwload uses the iWarp archiver **iwar** to extract files from the array module. The array module is created by **iwcb** using **iwar**.

1.9.2 iWarp array loader and iWarp array combiner

The iWarp array loader creates the array module by executes a user specified array program. The array module is not specific to a particular iWarp array but is defined for a general logical array of cells. **iwload** maps this logical array to the target physical iWarp array. **iwload** create initialization data structures from initialization tables created by the array combiner.

1.9.3 iWarp array loader and iWarp cell loader

iwload uses the iWarp cell loader to actually load the completed cell programs onto the array.

2 Purpose and general functions of the array loader

This sections reviews the general functions of the iWarp array loader **iwload** and outlines how a user communicates specific request to the array loader. The user passes requests through command line arguments and environment variables.

2.1 Array loader functions

iwload performs the following functions:

1. Check that there are sufficient communication resources on the target iWarp array to satisfy the connections defined in the array module.
2. Create communication initialization tables for each cell from the connection information in the array module.
3. Link cell programs with system libraries.
4. Load cell programs on cells of the array with proper initialization data.
5. Start execution on loaded cells.

2.2 Input

The primary input to the array loader is an array module created by the iWarp array combiner **iwcb**. The array module contains cell programs, a mapping of cell programs onto a logical array, and descriptions of program connections.

3 Array loader command line options

3.1 Sample array loader command

The command shown below assumes there is an array module `foo` created by the array combiner. The array loader is told to execute this array module on the iWarp array attached to the Sun named victory on the interface board 1.

```
iwload foo -t victory 1
```

3.2 Command line options

-V Print the version number and continue.

-VV Prints the version number, prints usage information, and exit.

-d Prints additional debugging information.

-e The execute flag. In addition to checking and linking, it executes the array program on the physical array. By default **iwload** will execute the array module.

-n The no execute flag. This flag prevents **iwload** from directly executing the array program. Instead it creates a script call **go** to load and execute the array program on the designated machine.

-offset *row,col* This defines where the logical cell (0,0) is loaded. By default it is loaded on physical cell (0,0).

-t *host_name sib_num* This specifies the target iWarp array by naming the host and sib number the array is attached to. By default, the target iWarp array is assumed to be on the current machine on sib 1.

-c *error level* This sets the level of error checking at run time. Currently, this is just a binary setting. An *error level* of 1 turns on all error checking. An *error level* of 0 turns off all error checking. By default all error checking is turned on.

iwcc *arguments* Any other arguments like -*l* or -L will be passed on to **iwcc** when the runtime libraries are linked in.

-*row,col* [*arg* *] Defines arguments to be used with the cell program on virtual cell (*row,col*).

array_module Produced by **iwcb**. This describes the array program to be loaded and contains the cell programs.

3.3 Environment variables

The proper version of the initialization libraries is determined from the environment variable **IWARP_ARCHITECTURE**. If you want to use a non-standard initialization library, set the environment variable **PCSINITLIB** to the non-standard library location. The use a non-default version of the runtime system, set the environment variable `RTS_VERSION` to the desired version number.

If no program has been assigned to a particular cell, and a network is routed through that cell a "dummy" program is loaded on the cell. The dummy program must call `pcs_init` to initialize the logical channels, so networks can be routed through the cell. By default the dummy program is **/iwarp/lib/pcs-dummy.o**. If you want to use your own dummy program, set the **PCSDUM** environment variable to that path name.

If the same files are placed on two cells and the same arguments are passed to these cells, those cells will share the same cell executable. This reduces the amount of time spent compiling and linking.

4 Related documents

- *iWarp linker EPS*, Herbert G. Mayer, Version 0.5 of March 16, 1990. Intel internal document.

  Describes the general behavior of the iWarp single-cell linker **iwld**.

- iWarp runtime system EPS, ?? Intel internal document.

  Describes the behavior of the iWarp runtime system.

- *iWarp PCS array combiner EPS*, Thomas Gross, Version 2.0 of July 8, 1990. Carnegie Mellon/Intel internal document.

  Describes the iWarp array combiner.

- *iWarp cell loader EPS*, ???? Intel internal document.

  Describes the iWarp cell loader.

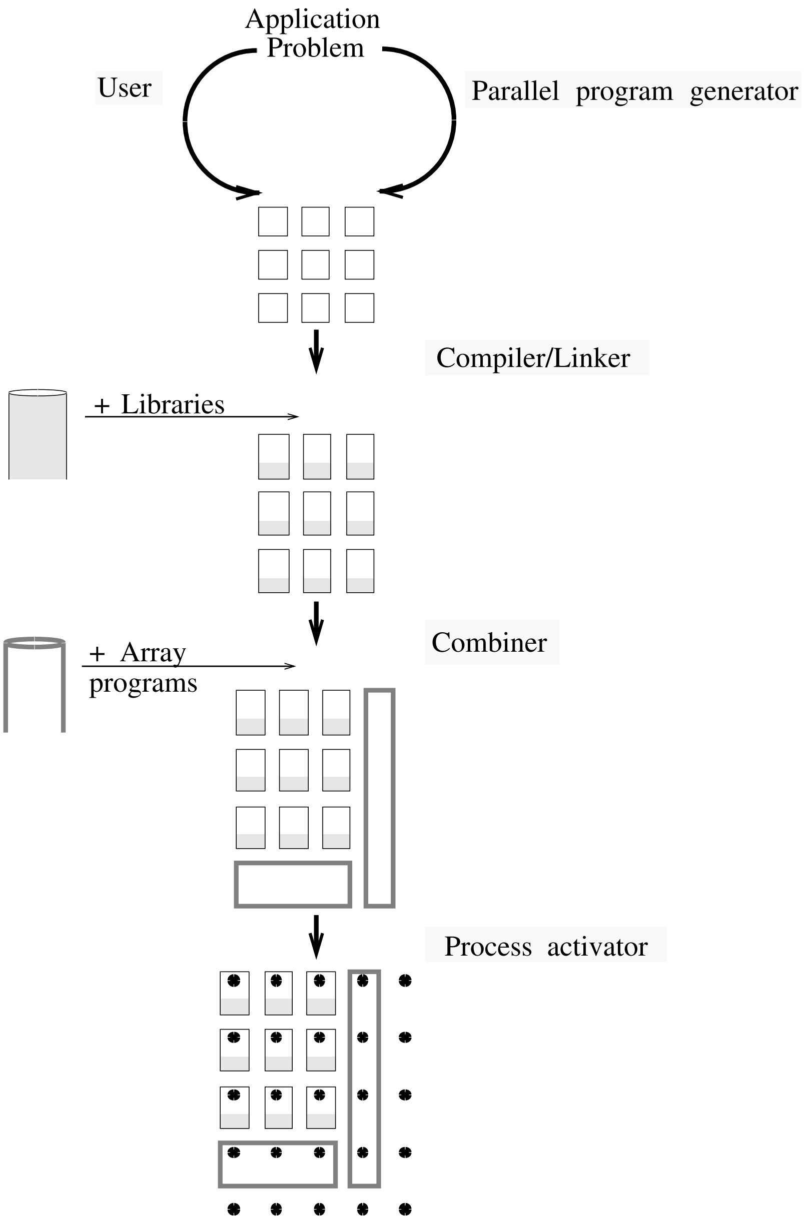

Figure 3: Program translation steps

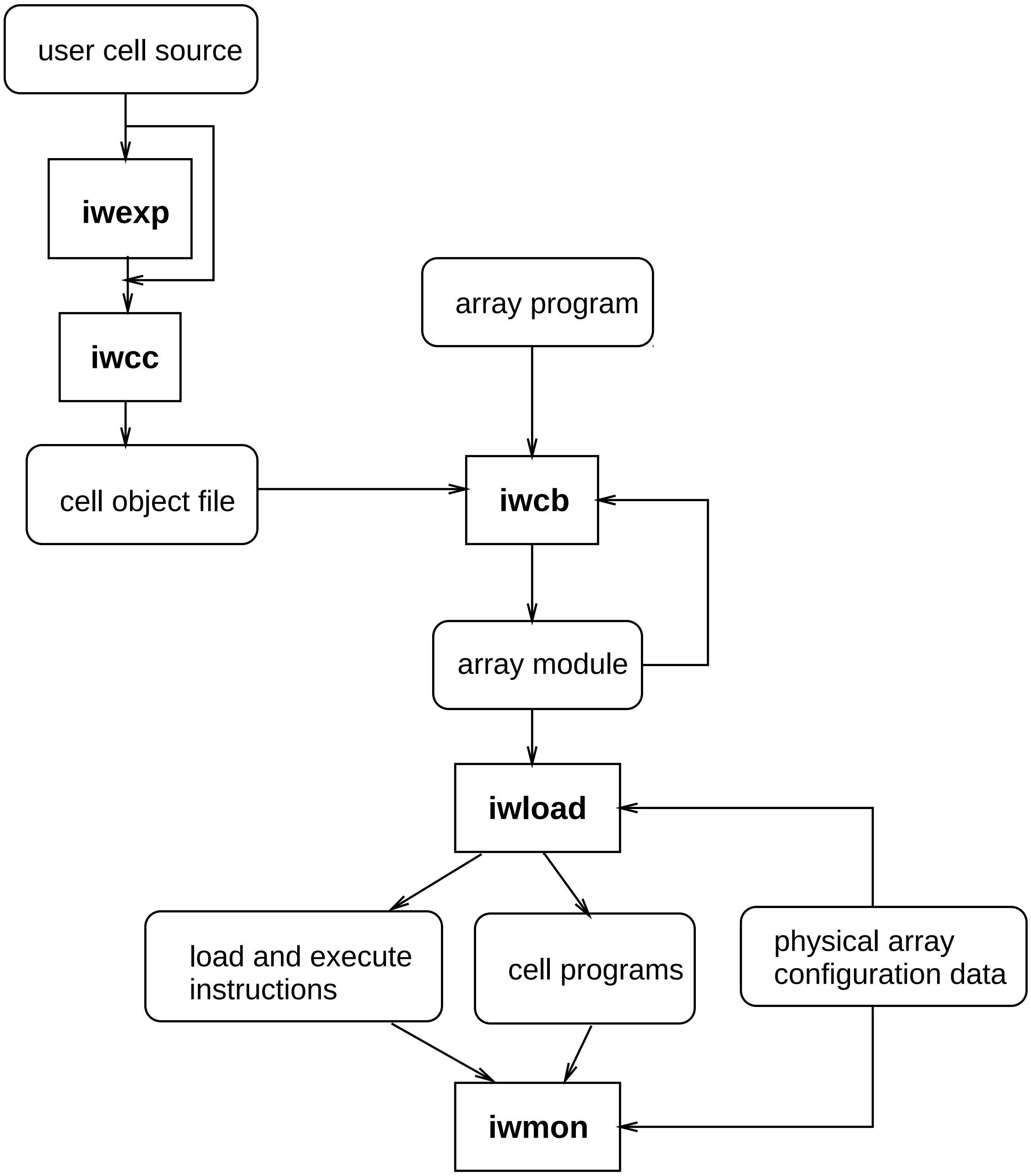

Figure 4: Flow chart of the iWarp tool chain