iWarp Local Guide

David R. O'Hallaron
October 19, 1993

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

/afs/cs/project/iwarp/doc/local.ps

The work was supported in part by Advanced Research Projects Agency (DOD) monitored by ARPA/CMO under Contract MDA972-90-C-0035, and in part by the Air Force Office of Scientific Research under grant F49620-92-J-0131.

Abstract

This document explains to new users how to use the iWarp systems at Carnegie Mellon. We give an overview of the hardware and software, explain how to run programs, and explain what to do if you have problems or questions.

1. Introduction

This is the iWarp Local Guide. Its purpose is to give new users a high–level introduction to the iWarp systems at Carnegie Mellon. It includes an overview of the hardware and software, a description of how to compile and run an example program, some hints for debugging iWarp programs, where to find documentation, and where to go for help.

Be sure you have the most recent version of this guide, which you can find online in

```
/afs/cs/project/iwarp/doc/local.ps
```

The current version of the iWarp software is found in

```
/iwarp
```

which exists on all the SunOS machines. The iWarp software can also be accessed from non SunOS machines in AFS land through

```
/afs/cs/project/iwarp
```

but the mapping between the two is confusing and convoluted. The simplest solution is to ask Michael Gillinov (mdg) to add a `/iwarp` link to your non SunOS machine.

1.1. iWarp manual pages

Use the command `/iwarp/bin/iwman` to read iWarp manual pages. For example:

```
iwman iwarp
```

will display the manual pages that give a summary of the software provided by Intel. A full list of current man pages is available in Appendix A of this document.

1.2. The iWarp bboard (newsgroup)

The `iwarp` bboard is another useful source of information about the state of the iWarp systems, new software releases, bug fixes, etc. and is a general way of communicating with iWarp project members.

`Bboard` is a term used within SCS (School of Computer Science) to refer to local newsgroups. Whilst the name `iwarp` is recognized as a bboard on SCS machines, it can also be accessed as a newsgroup across all CMU machines as `cmu.cs.proj.warp.iwarp`.

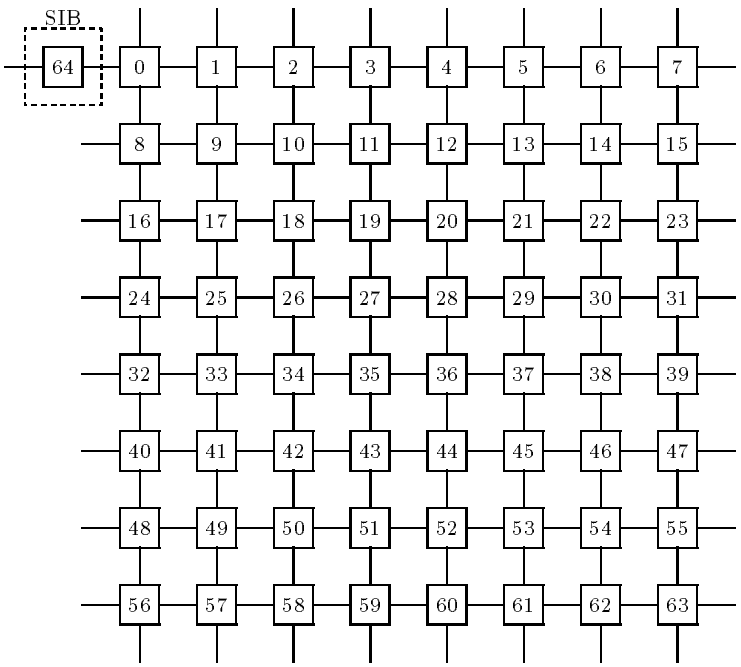

Figure 1: iWarp System

Various bboard/newsgroup reading packages are available at CMU; `bb` and `rn` are basic readers that are typically available on any Unix machine.

For SCS machines, information about the bboard system and various readers is available in the SCS general guide. SCS uses a newer threaded version of rn called `trn`. We also have an Emacs and Gnu-Emacs based reader called `bags`, a subset of the Rand MH mail program called `msh`, and an X-windows based program called `xrn`. Manual pages are available for all of these (i.e. `man bags`).

For Andrew machines, there are a number of readers including `batmail`, `vui` and `messages`.

2. Hardware overview

There are currently three 20 MHz 64–cell iWarp systems at CMU: Enterprise, Victory, and Stargazer. Stargazer is reserved for system development. Victory and Enterprise are available for general use. Figure 1 shows the setup for a typical 64–cell system. The Sun Interface Board (SIB) includes an iWarp cell and a 64 Kbyte dual–ported memory for talking to the Sun VME bus. Each SIB contains 2 MBytes of 25ns SRAM. The Intel numbering scheme for a 64–cell array is shown in the figure.

For more information: See [1, 5, 6].

3. Software Overview

3.1. C Compiler

The iWarp C compiler (`iwcc`) is a K&R C compiler with some extensions that support direct access of the systolic gates and assembly language inlining.

For more information: See [4] and the man page `iwcc(1)`.

3.2. Fortran compilers

The iWarp FORTRAN compiler (`iwf77`) is an optimizing version of f77 supported and distributed by Intel. It produces COFF (Common Object File Format) object code, and recognizes VMS extensions and various optimization methods including preprocessing and inlining.

Fx is a prototype parallelizing Fortran compiler developed at Carnegie Mellon for the iWarp system.

For more information: See the man page `iwf77(1)`. See

```
/afs/cs/project/iwarp/member/fx/doc/userguide
/afs/cs/project/iwarp/member/fx/examples
```

for the Fx User's Guide and example programs.

3.3. PCS

Programmed Communication Services (PCS) is a collection of tools and library routines for writing iWarp programs. Originally developed at Carnegie Mellon, it is now supported and distributed by Intel. PCS provides an array–level view of the iWarp, and efficient and direct program access to the iWarp's communication hardware. This section gives a brief overview of the PCS programming model, and the tools and library routines that support the model.

3.3.1. PCS programming model

PCS programs are based on the notion of a *module*. Modules are hierarchical entities, powerful abstractions that allow programs to be constructed from smaller programs. A module is either a *cell program*, which is a single-threaded C `main()` program that runs on one cell, or a collection of instances of modules and the networks that connect them. Networks can be simple point-to-point networks, or more sophisticated multi-source, multi-drop networks.

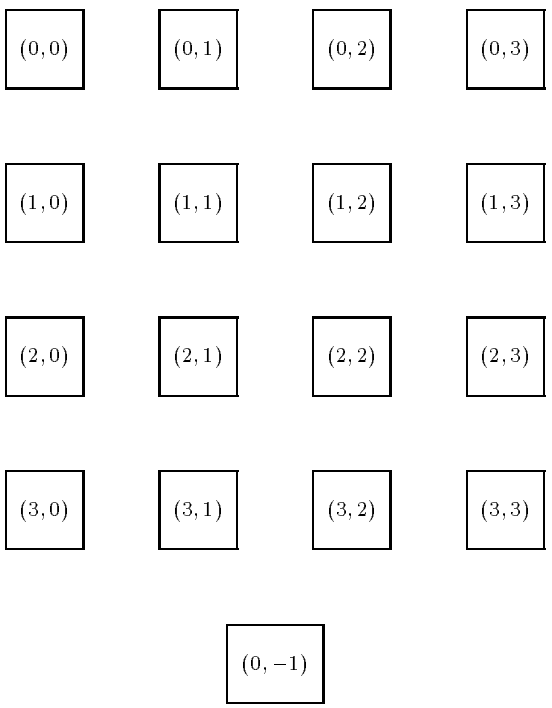

Figure 2: Numbering scheme for logical arrays

Modules run on *logical arrays* of cells. The cells in a logical array are numbered as shown in Figure 2. On our systems at Carnegie Mellon, there is one auxiliary cell $(0, -1)$ which corresponds to the SIB.

3.3.2. PCS tools

`iwcb` is the array–level linker. It inputs a C program called an *array program*, and outputs a single file, called an *array module*, that can be loaded and executed on the iWarp array. The array program describes in a textual way instances of modules and the networks that connect them. Modules and networks can be placed and routed manually, by the user, or automatically, by `iwcb`.

`iwamld` is the array–level loader. It inputs the array module produced by `iwcb` and loads and executes it on the iWarp array. The `iwamld` program is smart enough to produce only one load image for cells that run the same cell program. Since each distinct load image requires on the order of 1-2 minutes to compile and link, there is a strong motivation during code development to write array programs that consist of instances of only a few different types of cell programs.

Other PCS tools include `iwmd`, a handy X program that graphically displays the array modules produced by `iwcb`, and `iwpp`, a preprocessor that inlines function calls to the PCS library.

3.3.3. PCS library

PCS includes an extensive library of communications routines for accessing the networks defined by the array program. Examples include routines for transferring single words, blocks of data, and entire messages via the spooling gates or the systolic gates.

For more information: See [2] for details on PCS. See also `/iwarp/examples` for numerous examples of PCS programs.

3.4. RTS 3.1

Version 3.1 of RTS (Run Time System) is a simple runtime system that supports `malloc()` and Unix file I/O from any cell, asynchronous message passing between arbitrary cells on the array, and asynchronous message passing at up to 3 MB/s between arbitrary cells and the Sun host. An RPC-based Unix command interface is provided for basic functions such as locking and unlocking the array, loading code on a cell, running code on a cell, peeking and poking cell memory locations and registers, and querying completion status. These commands can be run from any SunOS workstation on the network.

For more information: See the man pages for `iwintro(1)`, `iwmon(1)`, `iwrts(1)`, `intro(2)`, `imsg_send(2)`, and `imsg_recv(2)`.

4. Running iWarp Programs

4.1. Finding a Sun SunOS Workstation

To use iWarp locally, you will need an account on the iWarp Suns. The iWarp software is installed on Sun-4 workstations running SunOs 4.1.1. A list of these machines can be found online in

`/afs/cs/project/iwarp/machines`.

Best bets are the public Sun-4/330's: `tokay`, `chenin`, `pinot`, and `gamay`. Most of the lab machines have 24MB of memory, however `pinot` and `gamay` each have 40MB making them quite attractive for large compilations. Please don't run compilations on the iWarp hosts `victory`, `enterprise`, and `stargazer`.

4.2. Setting up your SunOS Environment

In `.login`:

```
setpath -i1 /iwarp
setpath -i2 /iwarp/misc
```

Before running on Stargazer, set the following environment variables:

```
setenv IWARP_HOSTNAME stargazer
setenv IWARP_SIBNUM 0
```

Before running on Victory, set the following environment variables:

```
setenv IWARP_HOSTNAME victory
setenv IWARP_SIBNUM 0
```

Before running on Enterprise, set the following environment variables:

```
setenv IWARP_HOSTNAME enterprise
setenv IWARP_SIBNUM 0
```

The simplest way to switch back and forth between machines is to create files containing these commands and then source the files.

4.3. Compiling and running PCS programs

4.3.1. An example PCS program

Here is a simple PCS program that sends a message from from one cell to another. The program illustrates some basic things like how to synchronize the clocks on a collection of cells, how to write to files, how to transfer data between cells, how to blink the front panel lights, how to measure execution time, and how to use the C math library. Sources are in

```
/iwarp/misc/examples/local_guide/{array.c,cell.c,makefile}
```

The array program (`array.c`) looks like this:

```
#include <stdio.h>
#include <pcs/cb_def.h>

user_main()
{
  int i;
  cellid_type src, dst;
  port_type *out_port, *in_port;
```

```
  network_type *nw;

  src.row = 0; src.col = 0; src.type = ARRAY;
  dst.row = 1; dst.col = 1; dst.type = ARRAY;

  out_port = create_output_port("out", src);
  in_port = create_input_port("in", dst);
  nw = create_network(out_port, in_port, NULL);

  place(src, "cell.o");
  place(dst, "cell.o");
}
```

This array program defines a 2×2 logical array, with a point-to-point network connecting cell $(0,0)$ to cell $(1,1)$. Cells $(0,1)$ and $(1,0)$ are undefined and inactive. Connections are created directionally from one cell (`src`) to another (`dst`). Cell types are either `AUX` for the SIB or `ARRAY` for all others. The `place` function directs what COFF files are loaded on the specified cell. It can be called multiple times to place additional object files on the same cell. In this example, the same object file `cell.o` is placed on both active cells.

The cell program (`cell.c`) that runs on cells $(0,0)$ and $(1,1)$ is:

```
#include <stdio.h>
#include <math.h>
#include <pcs/pcs_def.h>
#include <pcs/pcs_time.h>
#include <ksupp/blink.h>
#include <iwsys/getcfg.h>

#define SIZE 1024

float data[SIZE];

extern int getcfg();
struct iwcfg cfg;

port_type *out, *in;
port_list_type ports[] = {
  {"out", &out},
  {"in", &in}};

main(argc, argv)
int argc;
char *argv[];
{

  int i, stime, clocks, bytes, mhz;
```

```
cellid_type *cellid;

/* create the network */
pcs_init(ports, NUM_PORTS(ports), NULL, 0);

/* initialize the barrier package */
getcfg(&cfg);
sb_init(cfg.cellid);

/* initialize the data */
cellid = get_cellid();
if (cellid->col == 0)
  for (i=0; i<SIZE; i++)
    data[i] = (float)sqrt((float)i);

/* transfer the message */
ledctl(LED_YELLOW_ON);
sb_synch(100,2,0);
stime = GetTimer1();
EnableTimer1();
if (cellid->col == 0) {
  bind_systolic_gate(out, GATE0);
  send_direct_msg(out, GATE0, data, SIZE*sizeof(float), NULL);
}
else {
  bind_systolic_gate(in, GATE0);
  receive_limited_msg(in, GATE0, data, SIZE*sizeof(float));
}
DisableTimer1();
sb_synch(200,2,1);
ledctl(LED_YELLOW_OFF);

/* print the performance */
clocks = (stime-GetTimer1())*8;
bytes = SIZE*sizeof(float);
mhz = cfg.syshz/1000000;
printf("%d bytes\n", bytes);
printf("%d clocks\n", clocks);
printf("%.6f seconds\n", (float)clocks/(float)cfg.syshz);
printf("%.2f clocks/byte\n", (float)clocks/(float)bytes);
printf("%.2f mbytes/second\n", ((float)bytes/(float)clocks)*(float)mhz);

/* print the results */
if (cellid->col) {
  printf("\n");
  for (i=0; i<8; i++)
    printf("%f\n", data[i]);
```

```
    }

    /* close the network */
    pcs_de_init();

    exit(0);
}
```

Notes

The `getcfg` function retrieves configuration information about the iWarp array and the local cell, and places it into a structure. Cell position (`row`, `col`), logical number (`cellid`) and clock speed (`syshz`) are all member variables of this structure.

Barrier synchronization, which is explained further in the `misc` section of this document, is used to coordinate the execution of the cells at various points. It is initialized with the `sb_init` function, and invoked with the `sb_synch` function. The cells are synchronized at two points in this program, before and after the primary action of the program.

The yellow LED is used to give some low-expense visual feedback about cell activity in this program. More information on the LEDs and how to view them is given later in this document. The function used here is `ledctl` which is loaded from the `ksupp/blink.h` include file. That file shows the various arguments to `ledctl` which are self-explanatory.

The timers in this program are a feature of PCS; calls to `GetTimer1` return integer values which count in descending order when enabled. Each decrement of the Timer value corresponds to approximately 8 (eight) clock cycles. In this case, timer values from the beginning and end of the primary computational segment are compared to calculate the overall elapsed time.

4.3.2. Compiling and Executing the Example Program

To compile and run this program on Victory, login to a Sun-4 that runs SunOS 4.1.1 and invoke the following makefile. Before you invoke the makefile, make sure you are authenticated to AFS on Victory by using the `kinit` program, and that the following environment variables are defined

```
setenv IWARP_HOSTNAME victory
setenv IWARP_SIBNUM 0


CC = iwcc
CFLAGS = -O -fsingle
TARGET_ARCH = iwarp
LIB = -L/iwarp/misc/barrier

all: array
  iwamld array -W1,$(LIB),-lm,-lsynch
```

```
array: array.c cell.o
  iwcb -row 2 -col 2 -o array array.c

clean:
  rm -f *.o cb1 array-* array_* array go std[eo]* *~
```

When you invoke this makefile, the following things happen. First, the C compiler inputs `cell.c` and outputs the cell program `cell.o`. Next, `iwcb` inputs the array program `array.c` and outputs the array module `array`, which is then loaded and executed on Victory by `iwamld`.

In general, the standard output from physical cell $xxxx$ can be found in

```
./stdoutxxxx
```

Similarly for standard input, the input for physical cell $xxxx$ is in

```
./stdinxxxx
```

and the output from physical cell $xxxx$ to standard error is in

```
./stderrxxxx
```

where xxxx is a 4-digit decimal number with leading zeros. By default, `iwamld` loads the logical array into the top left hand corner of the physical array. So in the example program logical cell $(0,0)$ is loaded onto physical cell 0 and logical cell $(1,1)$ is loaded onto physical cell 9. When the program executes, the output appears in files `stdout0000` and `stdout0009`. The physical cell numbers correspond to a 64-cell iWarp array as used in the example; the numbers would differ for other array sizes.

For more information: See [2].

4.4. The iWarp misc collection

The example program makes use of several iWarp packages which are found in the iWarp misc directory:

```
/iwarp/misc
```

This area contains collections of useful tools and packages that are locally maintained. The first two are employed in the example program.

If not already done, you can update your environment with:

```
setpath -i1 /iwarp/misc
```

to create paths to these various collections.

4.4.1. Barrier Synchronization

The synch library gives the programmer control over the execution times of groups of cells by setting "roadblocks" for each cell which are not passed until all cells in the group are at that point.

In the example program, we use the function `sb_synch` to start the timing of the two cells together. This sort of usage is especially helpful in larger arrays where the disparity in time can be significant, and can make accurate timing comparisons otherwise impossible. The exact usage of `sb_synch`, `sb_init`, and other functions is explained in the `synch(3)` man page.

In the makefile for the example, notice the `LIB` variable which includes the barrier collection, and the `-lsynch` flag in its unorthodox format to `iwamld` which loads the physical library `/iwarp/misc/barrier/libsynch.a`.

The barrier synchronization package is in :

```
/iwarp/misc/barrier
```

and includes documentation and examples.

4.4.2. LED routines (LED controlling)

The example program makes use of the status lights on the iWarp to give feedback about program execution. Each cell on the iWarp has three status lights red,yellow,green which can be controlled by the program executing on that cell. The `ledctl(2)` man page gives details on the standard method of blinking these LEDs. A more robust package of routines is found in:

```
/iwarp/misc/led
```

in the include file `led.h`. The `README` file gives an explanation of the various functions.

4.4.3. iwspy (LED watching)

`iwspy` is an X-windows based tool that displays the monitor lights for an iWarp array and is an enormously helpful debugging aid. `iwspy` can be run (architecture-independent) as:

```
/iwarp/misc/bin/iwspy
```

Usage is

```
iwspy [ system ].[ sib ] &
```

where `system` is the name of the iWarp array you are using and `sib` is the number of the SIB you are utilising ("iwspy `victory.0` &" for example).

`iwspy` is also available for use in Gnu-Emacs providing an ascii representation of the status lights in a separate window. The lisp load file is found in:

```
/iwarp/misc/iwspy/lisp/iwspy.el
```

4.4.4. calctime

`calctime` is a tool that estimates numbers of execution cycles of a cell program based on profile data and object codes. When using `calctime`, the object code gathering profile data and the object code being used in cycle calculation are different. Thus, the tool can eliminate from its timings the effects of profiling and/or workarounds. In addition to being more accurate than the standard profiler (iwprof), `calctime` also provides more detailed statistical information. it can be found in:

```
/iwarp/misc/calctime
```

The `README` file in that directory gives details whilst the `examples` directory contains a sample profiling session.

4.4.5. espl

`espl` (Efficient Spooling Pathway Library) as a package of inserts (written in assembly) that provides very fast communication between cells. It is bundled as an include file for iWarp C programs in:

```
/iwarp/misc/include/espl.h
```

See the `espl(3)` man page for details.

4.4.6. Kernels

This is a collection of signal-processing and matrix routines. written in assembly for inclusion in iWarp C programs, their primary use is for benchmarking the compiler but can also be useful for applications. the directory is:

```
/iwarp/misc/kernels
```

and the `README` file is quite informative.

4.4.7. Message Passing

Included as a starting point for message passing on the iWarp,

```
/iwarp/misc/mp
```

is a collection of various message passing models. The model providing buffering and visual feedback is probably the best choice for initial testing and comparison. Contact Tom Stricker for more information on iWarp message passing.

4.5. Debugging iWarp Programs

A symbolic debugger for the iWarp is now available in:

```
/iwarp/bin/iwdb
```

`iwdb` is an extension of the familiar Gnu debugger (gdb). It provides much of the functionality of gdb with added control management for cells and cell groupings. The `iwdb(1)` man page has details.

For the more adventurous, `RTS` provides some low–level debugging facilities for peeking and poking memory and register locations. See the `iwmon(1)` man page for details.

The `LED` routines and `iwspy` packages mentioned previously are very helpful debugging tools.

Be warned that although iWarp has memory protection hardware, it is not used by the current version of RTS. So it is very easy to overwrite the monitor or stack and crash the system.

Some catastrophic failures will leave a cell spinning in an event handler, with no way to use `RTS` to probe the machine. These kinds of failures light up the red LED on the front panel. If your program turns on the red LED, see Peter Lieu (pjl@cs.cmu.edu), who can show you techniques for finding out what caused the failure.

Another problem to be aware of during debugging is that if the monitor crashes on any cell (a common occurrence because of the lack of memory protection in the current version of RTS), then you may not necessarily see all of the printf's issues by the cell programs.

User programs are loaded at address 72 Kbytes, leaving 440 Kbytes for program, static and dynamic data, and stacks. A C "hello, world" program requires roughly 30 Kbytes. Use `iwsize` to check the size of an executable file.

For more information: See [3] for details on the iWarp assembly language. Also see the man pages for `iwmon(1)` and `iwsize(1)`.

5. Where to Find Documentation

Historical CMU iWarp documentation and papers can be found online in

```
/afs/cs/project/iwarp/doc/
```

See the README file for details. Online postscript versions of some of Intel's documentation can be found in

```
/afs/cs/project/iwarp/doc-intel/
```

Release notes for version xxx of Intel's software are in

```
/afs/cs/project/iwarp/docs/
```

The iWarp papers archive is in

```
/afs/cs/project/iwarp/papers
```

Barbara Grandillo (bag@cs) has hard copies of the Intel iWarp documents in her office (WeH 8212).

6. Reporting Problems and Getting Help

Peter Lieu (pjl@cs) is the chief point of contact between Carnegie Mellon iWarp users and Intel. If you need help, or have found a bug, send mail to Peter.

A iWarp manual pages

IWAMLD (1) - PCS array loader
IWCB (1) - PCS array combiner
IWMD (1) - PCS array module display
IWPP (1) - PCS preprocessor
PCS_INTRO (1) - Introduction to the iWarp Programmed Communication Services
iwprof (1) - display profile data
cell sets (3) - Create sets of cells to load/run on the iWarp system
diwas (1) - iWarp disassembler
espl (3) - Efficient Spooling Pathway Library
imsg_block (3) - block until input message available
imsg_info (3) - return information about first message on input queue
imsg_open (3) - open HOST device used for iW/RTS message communication
imsg_recv (3) - asynchronously receive a message
imsg_send (3) - asynchronously send a message
imsg_stat (3) - return information about a pending send or receive
isib (4) - driver for Intel iWarp Sun-Interface Board
isibio (4) - Intel iWarp SIB ioctl operations
iwar (1V) - archive and library maintainer for portable archives
iwarp (1) - iWarp Software Summary
iwas (1) - iWarp assembler
iwbrklock (1) - break another owner's lock on an iWarp array.
iwbrklock, iwclrstat, iwdepcsr, iwdepmem, iwexcsr, iwexmem, iwhpi, iwkill, iwload, iwlock, iwreset, iwrun, iwstat, iwtime, iwunlock, iwvers, lgo - Unix command interface to the iWarp monitor daemon
iwcc (1) - iWarp optimizing C compiler
iwccomq (1) - iWarp global optimizer
iwcfg (5) - iWarp array configuration file
iwcfg, cfg_getsystem, cfg_readcfg, cfg_free, cfg_packcfg, cfg_selectcell, cfg_getinfo, cfg_getattr, cfg_getmemmap - host-side iWarp configuration information interface
iwclrstat (1) - clear cell exit status buffer
iwconf (1) - the iWarp interactive diagnostic monitor
iwcpp (1) - the C language preprocessor
iwdb (1) - The iWarp symbolic debugger
iwdepcsr (1) - deposit to control-space register on an iWarp cell
iwdepmem (1) - deposit data to memory on an iWarp cell
iwdump (1) - dump selected parts of an object file
iwexcsr (1) - examine control-space register in an iWarp cell
iwexmem (1) - examine memory on an iWarp cell
iwf77 (1) - iWarp FORTRAN 77 compiler (with VMS extensions)
iwfile (1) - determine the type of a file by examining its contents
iwhpi (1) - start iW/RTS Host Proxy Interface server
iwkill (1) - send a kill signal to a cell
iwld (1) - link editor
iwliw (1) - Optimizing code generator for iWarp
iwload (1) - load iWarp executable object files on cells.
iwlock (1) - gain exclusive access to an iWarp array.

iwman (1)	- display reference manual pages specific to iWarp
iwmond (8)	- iWarp Monitor daemon
iwnm (1)	- print name list
iwreset (1)	- restart iwrts
iwrts (7)	- The iWarp Run-Time System
iwrun (1)	- execute iWarp cell programs
iwsize (1)	- print section sizes in bytes of common object files
iwstat (1)	- retreive iWarp array status information
iwstrip (1)	- remove symbols and relocation bits from an object file
iwtime (1)	- report execution time of an iWarp cell program
iwunlock (1)	- remove a lock on an iWarp array
iwvers (1)	- report version of iwmon server
lgo (1)	- simplified interface to iwmon
mdl_loadargs (3)	- Download argc, argv, and envp onto an iWarp array
mdl_loadcoff (3)	- Download a COFF file onto an iWarp array
synch (3BS)	- iWarp barrier synchronization library (libsynch)

References

[1] S. Borkar et al. *iWarp Macroarchitecture Specification, Version 3.0.* Intel Corporation, March 1990.

[2] S. Hinrichs. *User's Guide for the Programmed Communication Services (PCS) Tool Chain.* Carnegie Mellon University, 1991.

[3] P. Miller. *iWarp Assembly Language Specification Release 2.0 for Macroarchitecture Specification, Version 3.0.* Intel Corporation, September 1990.

[4] J. Reinders. *iWarp C Guide.* Intel Corporation, November 1989.

[5] S. Borkar et. al. iWarp: An integrated solution to high-speed parallel computing. In *Supercomputing '88*, Kissimmee, FL, November 1988.

[6] S. Borkar et. al. Supporting systolic and memory communication in iWarp. In *17th Annual International Symposium on Computer Architecture.* IEEE Computer Society and ACM, May 1990.