iWarp Local Guide

David R. O'Hallaron
June 17, 1992

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

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

The work was supported in part by Defense Advanced Research Projects Agency (DOD) monitored by DARPA/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 note 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.

Use `/iwarp/bin/iwman` to read iWarp man pages. The `iwarp` man page gives a summary of the software provided by Intel. The `iwarp` bboard is another useful source of information about the state of the iWarp systems, new software releases, bug fixes, etc. Type `man bb` from a Mach machine for details.

Because the system is still evolving, be sure you have the most recent version of this guide, which you can find online in

```
/iwarp/doc/local.ps
```

from the iWarp Suns, or

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

from anywhere.

2. Hardware Overview

There are two 64–cell iWarp systems at CMU: Enterprise and Victory. Figure 1 shows the setup for each system. Enterprise and Victory run at 10 MHz. The Sun Interface Board (SIB) includes an iWarp cell and a 64 Kbyte dual–ported memory for talking to the Sun VME bus. Each cell (except for the SIB cell on Victory) contains 512 Kbytes of 25ns SRAM. The SIB on Victory contains 2 Mbytes of SRAM. The Intel numbering scheme for the cells 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.

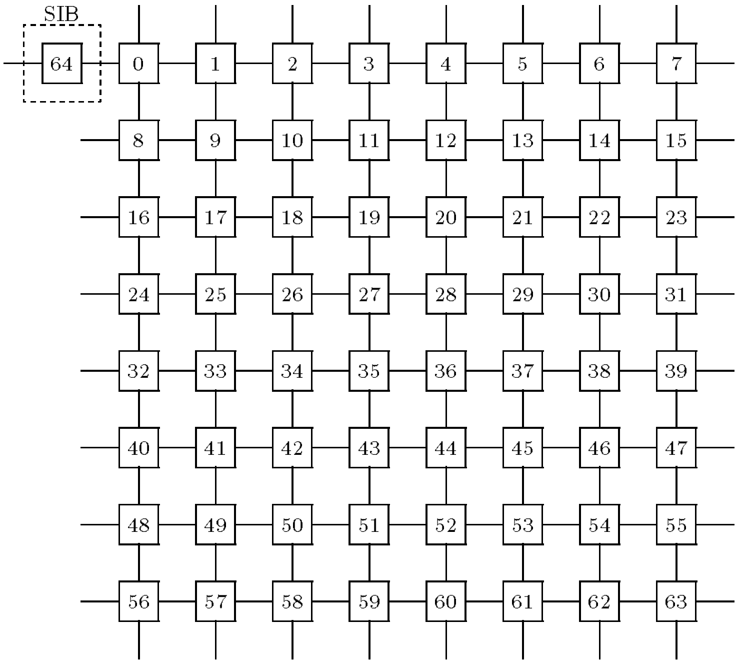

Figure 1: iWarp System

3.2. PCS

Programmed Communication Services (PCS) is a collection of tools and library routines, developed at Carnegie Mellon, for writing iWarp programs. 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.

3.2.1. 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.

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.

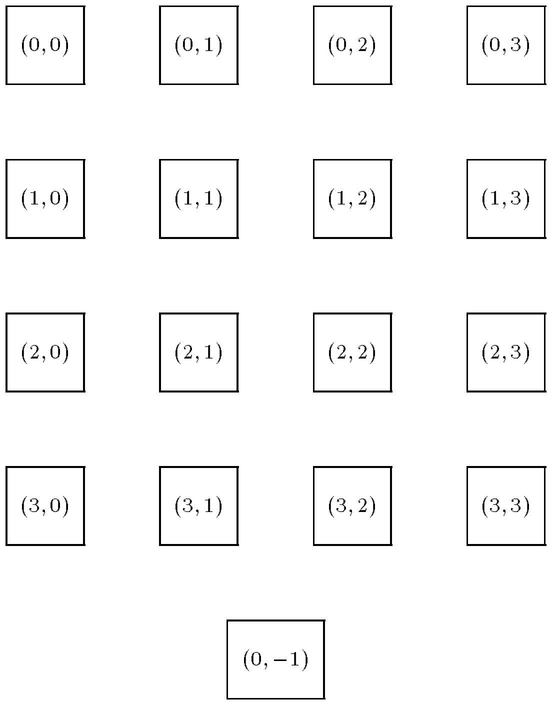

Figure 2: Numbering scheme for logical arrays

3.2.2. 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`.

`iwload` is the array–level loader. It inputs the array module produced by `iwcb` and loads and executes it on the iWarp array. The `iwload` 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 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.

3.2.3. 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.

3.3. RTS 2.5.2

Version 2.5.2 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 6 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 Workstation

To use iWarp 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`, and `gamay`. Please don't run compilations on the iWarp hosts `victory`, `enterprise`, and `stargazer`.

4.2. Setting up your Environment

In `.login`:

```
setpath -i1 /iwarp
```

Before running on Enterprise, set the following environment variables:

```
setenv IWARP_HOSTNAME enterprise
setenv IWARP_SIBNUM 2
setenv IWARP_CFGPATH /iwarp/etc2.5.2/cfg
```

Before running on Victory, set the following environment variables:

```
setenv IWARP_HOSTNAME victory
setenv IWARP_SIBNUM 1
setenv IWARP_CFGPATH /iwarp/etc2.5.2/cfg
```

4.3. Compiling and Running PCS Programs

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

```
/afs/cs/project/iwarp/member/droh/iwarp25/example/{array.c,cell.c,makefile}
```

The array program (`array.c`) is:

```
#include <stdio.h>
#include <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 = output_port_create("out",  src);
  in_port = input_port_create("in", dst);
  nw = network_create(out_port, in_port, NULL);

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

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

```
#include <stdio.h>
#include <math.h>
#include <pcs_def.h>
#include <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) {
    systolic_gate_bind(out, GATE0);
    send_direct_msg(out, GATE0, data, SIZE*sizeof(float), NULL);
  }
  else {
    systolic_gate_bind(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);
}
```

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 1
setenv IWARP_CFGPATH /iwarp/etc2.5.2/cfg
```

```
CC = iwcc
CFLAGS = -O -fsingle
TARGET_ARCH = iwarp
LIB = -L/usr/anja/synch/lib/2.5.2

all: array
  iwload-pcs array -t $(IWARP_HOSTNAME) $(IWARP_SIBNUM) $(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 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 `iwload-pcs`.

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, `iwload-pcs` 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`.

For more information: See [2].

Debugging iWarp Programs

There isn't a source–level debugger for cell programs yet; probably the easiest way to debug your programs is to use `printf's`. 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.

Another useful and relatively nonintrusive debugging and monitoring technique is to use the lights on the front panel. Each cell has a red, green, and yellow light that can be turned on and off from your program. See the `ledctl(2)` man page for details.

If you use X, you can watch the lights from your workstation using Tom Warfel's `iwspy` program. Executables for Sun4 OS, Sun4 Mach, Pmax, RT workstations are in

```
/afs/cs/project/iwarp/member/tew/iwspy/xclient/latest/*/iwspy
```

where "*" is either `Sun4Machbin`, `Sun4OSbin`, `Pmaxbin`, or `rt_r3`, depending on the kind of workstation you have. Usage is

```
iwspy system.sib &
```

where `system.sib` is either `victory.1` or `enterprise.2`.

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)`, `iwsize(1)`, and `ledctl(2)`.

5. Where to Find Documentation

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

```
/iwarp/xxx/docs/
```

Marian D'Amico (marian@cs) has hard copies of the Intel documents in a file cabinet in her office. Barbara Grandillo (bag@cs) has hard copies of some of the published iWarp papers in her office.

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.

7. Acknowledgements

Thanks to Mike Gillinov, Susan Hinrichs, Jim Stichnoth, and Jon Webb for their comments and suggestions.

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.