

TM No. 941020

REFERENCE COPY

NAVAL UNDERSEA WARFARE CENTER
DETACHMENT NEW LONDON
NEW LONDON, CONNECTICUT 06320-5594



Technical Memorandum

**INTERFACE BETWEEN
TWO PARALLEL COMPUTERS**

Date: 28 February 1994

Prepared by:

A handwritten signature in dark ink, appearing to read 'W. Robert Bernecky', is written over a horizontal line.

W. Robert Bernecky
System Development Division
Submarine Sonar Department

Approved for public release; distribution unlimited.

Report Documentation Page			Form Approved OMB No. 0704-0188		
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE 28 FEB 1994		2. REPORT TYPE Technical Memorandum		3. DATES COVERED 28-02-1994 to 28-02-1994	
4. TITLE AND SUBTITLE Interface between Two Parallel Computers			5a. CONTRACT NUMBER		
			5b. GRANT NUMBER		
			5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) W. Bernecky			5d. PROJECT NUMBER		
			5e. TASK NUMBER		
			5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Undersea Warfare Center Detachment New London, New London, CT, 06320			8. PERFORMING ORGANIZATION REPORT NUMBER TM 941020		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research, Computer Technology Block			10. SPONSOR/MONITOR'S ACRONYM(S)		
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited					
13. SUPPLEMENTARY NOTES NUWC2015					
14. ABSTRACT This document describes the high-speed interface developed by NUWC to support data transport between heterogeneous, massively parallel computers. The interface provides a sustained data transfer rate of up to 3.5 Mbytes/second between an Intel iWarp and a Thinking Machines CM200a. The design is adaptable to other VMEbus-compatible interface boards.					
15. SUBJECT TERMS iWarp; Thinking Machine CM20a; VMEbus					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Same as Report (SAR)	18. NUMBER OF PAGES 23	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

ABSTRACT

This document describes the high-speed interface developed by NUWC to support data transport between heterogeneous, massively parallel computers. The interface provides a sustained data transfer rate of up to 3.5 Mbytes/second between an Intel iWarp and a Thinking Machines CM200a. The design is adaptable to other VMEbus-compatible interface boards.

ADMINISTRATIVE INFORMATION

The research described in this memorandum was performed under the Advanced Sonar Processing Architectures (ASPA) program, Principal Investigator Dr. J. Munoz. The sponsoring activity is the Office of Naval Research (ONR), Computer Technology Block, Program Manager Elizabeth Wald.

The author of this memorandum is located at the Naval Undersea Warfare Center, Detachment , New London CT 06320-5594. The technical reviewer for this document was Dr. Jose Munoz.

TABLE OF CONTENTS

1. INTRODUCTION	1
2. HARDWARE OF THE PARALLEL INTERFACE.....	3
2.1 Intel Standard Interface Board (iSIB).....	3
2.2 Thinking Machines CM/VME I/O Board (CM/VMEIO).....	5
3. SOFTWARE OF THE PARALLEL INTERFACE	6
3.1 iSIB Software	7
3.2 Software on the CM VMEIO host/ iWarp host.....	7
3.3 Software on the CM host.....	8
4. PARALLEL INTERFACE PERFORMANCE	8
5. REFERENCES	10
6. APPENDIX A.....	11

1. INTRODUCTION

This document describes the high-speed interface developed by NUWC to support data transport to, from and between massively parallel computers. This parallel interface (PIF) is currently being used to transfer data from an iWarp parallel computer [1] to a Thinking Machines Connection Machine CM200a (CM) [2] at a rate of up to 3.5 Mbytes/second. It has also been used to transfer data between the CM200a and a SUN 4 Workstation. This same interface design will be used to connect sonar sensor data to the iWarp system in planned sea-test trials of the Advanced Sonar Processing Architecture (ASPA) program.

The user (programmer) view of the PIF is presented in Figure 1. A user program running on the iWarp can read and/or write to special memory buffers. Conversely, a CM program may also read and/or write to these same buffers. Software routines are available that enable the user to easily implement double-buffered I/O schemes.

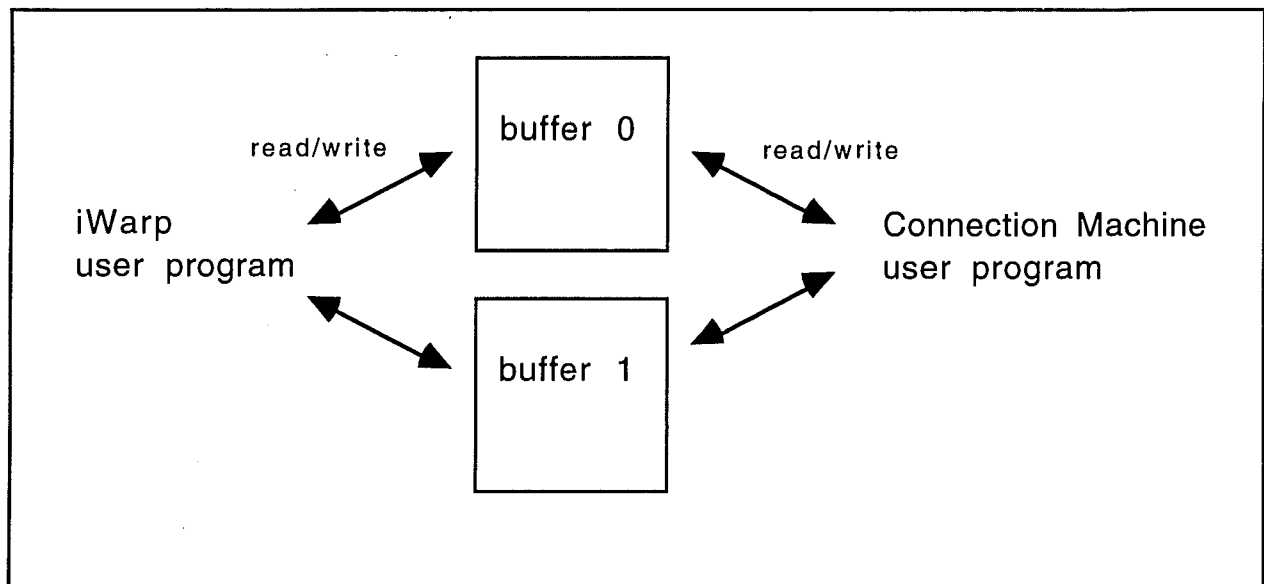


Figure 1 - Programmer's view of the parallel interface PIF.

The PIF is actually a set of software routines and a collection of existing hardware components that collectively provide to the user the data transfer paradigm illustrated in Figure 1.

For context, Figure 2 presents the major hardware elements of the composite system of two parallel computers and the associated interface components. Briefly, the two parallel computer systems are the iWarp, and the Connection Machine.

The iWarp is a Multiple-Instruction, Multiple-Data (MIMD) computer comprised of up to 256 cells interconnected in a mesh configuration with a proprietary 40 Mbyte/second per unidirectional bus (8 buses per cell) network. The iWarp system is interfaced to the outside world via an Intel SUN Interface Board (iSIB) that sits on the VME backplane [3] of a SUN 4 Workstation acting as the iWarp Host.

The CM is a Single-Instruction, Multiple-Data (SIMD) computer consisting of 8192 single-bit processors, 256 floating point chips, and an aggregate of 1000 Mbytes of memory. Unlike the iWarp host, the front-end (CM host) SUN 4 Workstation plays an integral role in controlling the execution of CM programs. Similar to the iWarp, the CM has an interface board (CM/VMEIO board) that, as does the iSIB, sits on the VME backplane of a SUN 4 Workstation (CM/VMEIO host). The CM/VMEIO host is distinct from the CM host. The system software that controls the CM/VMEIO board was developed by NUWC and is documented in [4].

As shown in Figure 2, the CM/VMEIO host and the iWarp host are the same SUN 4 Workstation.

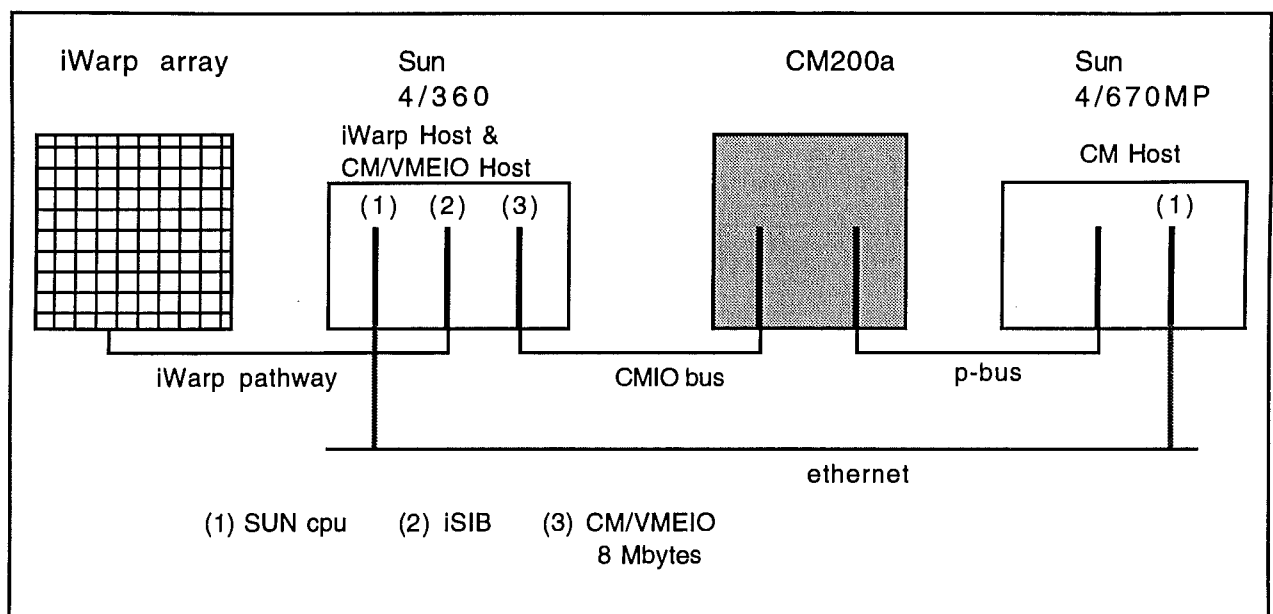


Figure 2 - Hardware configuration of PIF.

Figure 3 provides more detail on the hardware configuration most pertinent to the PIF proper. This hardware includes two VMEbus interface boards, the iSIB and the CM/VMEIO, provided respectively, by Intel and Thinking Machines. Both of these boards serve the same purpose, which is to interface the parallel computer to the *de facto* data transfer standard VMEbus[3]. The details involved in implementing the Parallel InterFace using the two VMEbus interface boards are presented in the remainder of this document.

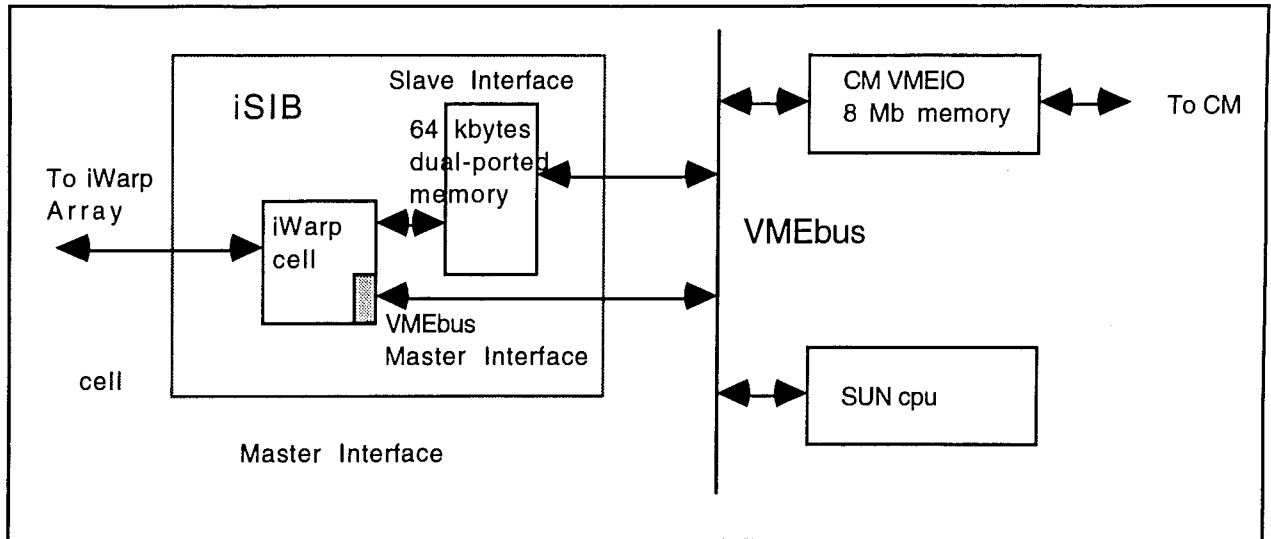


Figure 3 - Hardware details of iWarp/VMEIO Host.

The next two sections describe in more detail the hardware and software of the PIF. A final section presents some performance numbers.

2. HARDWARE OF THE PARALLEL INTERFACE

The Parallel InterFace (PIF) between the iWarp and the CM200a consists of the following items:

- A. SUN 4 Workstation (iWarp/VMEIO host) with VMEbus backplane.
- B. An Intel iSIB board that sits on the VMEbus of item A.
- C. CM/VMEIO board that also sits on the VMEbus of item A.
- D. SUN 4 Workstation acting as the CM host.
- E. Ethernet (or FDDI) local area net connecting items A and D.

2.1 Intel Standard Interface Board (iSIB)

A full description of the iSIB is given in [5]. The iSIB provides a communication path between an iWarp processor array and a SUN VMEbus. The iSIB is a single board (9U form factor, 15.75" by 14.4", 5 Volt, 16 Ampere) and contains an iWarp cell, a VMEbus slave interface and a VMEbus master interface.

The VMEbus slave interface is a 64 Kbyte dual-ported memory with a peak transfer rate of 80 Mbytes/second. Because the iWarp host software uses this interface to support data transfers between the SUN 4 Workstation host (iWarp host) and the iWarp array, this interface is unsuitable for I/O to the CM.

The master interface, Figure 3, gives the iWarp cell access to the VMEbus, and has a peak transfer rate of 18 Mbytes/second. It provides access to the VMEbus through two *windows* located in the local memory space of the iWarp cell: a main window (32 Mbytes) and a small window (16 Mbytes). Page registers are used to map these windows onto the VMEbus address space. I will note here that the PIF uses only the *small window of the master interface*. Use of the main window resulted in inconsistent performance (dropped data bits) indicative of a hardware design flaw.

```
/* VMEbus word access using small window */

#include <asm/isib.h>
/* defines isib constants, including:
-----
Extended Non-Privileged:Data Access */
#define AM_UEDATA      (AM_EXTENDED | AM_USER | AM_DATA) /* 0x09 */
#define SWR            *(int *)0x00c00008 /* Small Window Register */
#define SSR            *(int *)0x00c00010 /* Small Size Register */
#define SIB_SWINDOW    0x01000000 /* Small Window into VMEbus */
#define SSR_WORD       (0) /* 32-bit access */
/*-----*/

#define CM_addr        (0x8000 | AM_UEDATA) /* Page and addr modifier */

int i, *ptr= (int *) SIB_SWINDOW;

SWR= CM_addr; /* point to VME I/O shared memory */
SSR= SSR_WORD; /* word access only */

for(i=0;i<10;i++) *ptr++ = i; /* Example of writing to VME memory */
```

Figure 4- accessing iSIB small window Master Interface.

The small window is located in the iSIB's local memory space at byte address 0x0100,0000 to 0x01FF,FFFF. The window is 4 Mwords long, and can access the VMEbus using data sizes of 8, 16, and 32 bits. Two registers must be initialized to use this window:

- (1) Small Window Register (SWR); byte address 0x00C0,0008
- (2) Small Size Register (SSR); byte address 0x00C0,0010

(See Figures 4 and 5.)

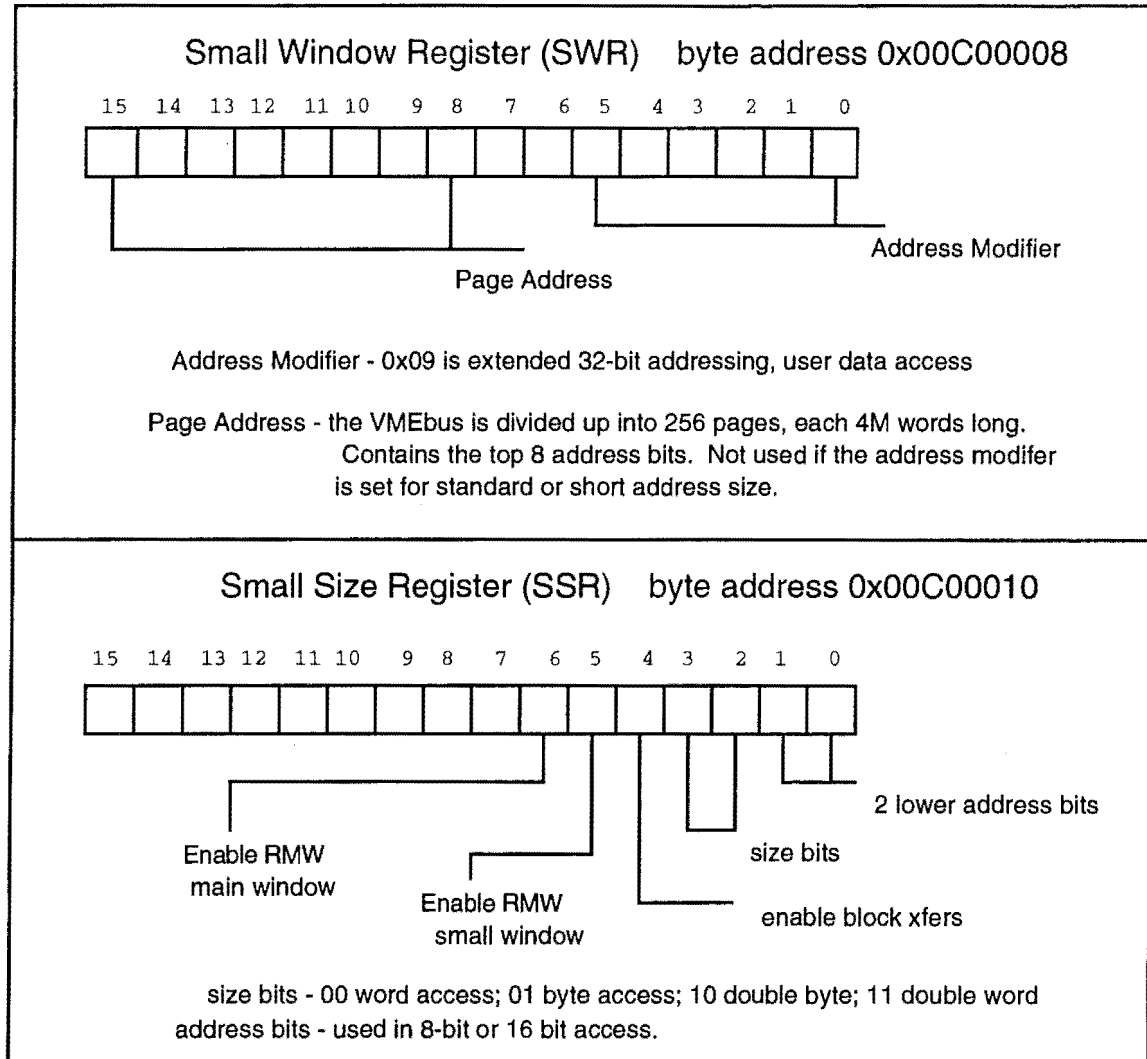


Figure 5- iSIB small window Master Interface Registers.

2.2 Thinking Machines CM/VME I/O Board (CM/VMEIO)

The CM/VMEIO is a single 9U board that connects the VMEbus to the CM's 64-bit multidrop CMI/O bus. The CM/VMEIO appears to any VMEbus master as 8 Mbytes of byte accessible memory (aligned or unaligned) addressed in extended 32-bit address space and 18 I/O registers addressed in short 16-bit address space. The starting location of the extended address space is set in a nine bit switch located on the board, and, in the current configuration, has a value of 0x8000,0000. Four address modifier codes are standard for extended addressing, including "Extended non-privileged data access" 0x09, the modifier used by the PIF.

Additional hardware details concerning the CM/VMEIO interface board are given in [6]. However, the user's view of the interface is defined by a NUWC-developed CM/VMEIO *driver*, which is additional system software incorporated into the CM's CMFS file server[4,9].

3. SOFTWARE OF THE PARALLEL INTERFACE

The PIF may be regarded as the set of software routines available to the user to facilitate the transfer of data between the iWarp and the CM. In general terms, this software includes initialization procedures, a synchronization service to support double-buffering, and a simplified (for the user) read/write memory abstraction.

Figure 6 indicates the user code required to transfer data between the iSIB and the CM. This (user) program runs on the iSIB's iWarp cell. Line 1 initializes the system by *mapping* the small window of the master interface to the memory on the CM/VMEIO board. Any reads or writes to the small window will now result in reads or writes (over the VMEbus) of the CM/VMEIO memory. The initialization also creates two buffers, blocks of memory on the CM/VMEIO board, referred to as buffer 0 and buffer 1. The user gains access to these shared buffers by invoking *iW_grab_buff* as indicated in Line 5. This procedure returns a pointer to buffer 0 or 1. Synchronization occurs implicitly, in that this procedure call does not return until the buffer is available. Initially, both buffers are free (to the iSIB). Writing data to the memory location indicated by this pointer results in data being transferred to the CM/VMEIO board. This occurs in Line 6. Similarly, any reads will result in data being transferred from the CM/VMEIO board to the iSIB. Finally, Line 7 releases the buffer and additionally, returns the buffer ID of the other buffer (e.g. buffer 1, if id= 0). The process of releasing the buffer again requires synchronization: the CM can not read a buffer until it has been released (indicating it has been filled) by the iWarp.

```

1  iW_init_db(2Mbytes);
2  id = 0;          /* start with buffer 0 */
3  Do forever
4  {
5      ptr = (int *) iW_grab_buff(id);
6      for (i=0; i<N; i++) *ptr++ = i; /* write out */
7      id= iW_release_buff(id); /* flips buffer id */
8  }
```

Figure 6- PIF user code executing on the iSIB.

The synchronization of the read/write access of the two buffers is accomplished by defining the notion of *ownership*. A user program may only read or write to a buffer that it owns. Ownership can only be changed by the current owner (either the iWarp or the CM), and is accomplished by releasing the buffer. Thus, to implement a double buffering scheme where both programs may simultaneously read and/or write to (separate) buffers, each must repetitively execute "grab buffer, perform I/O, release buffer". As noted above, the iSIB initially owns the two buffers.

There are three separate software components in the PIF, corresponding to the three hardware platforms involved in the interface. These are: (1) the iSIB; (2) the CM/VMEIO SUN host; and (3) the CM SUN host. The C programs implementing the PIF are given in Appendix A.

3.1 iSIB Software

The iSIB software of the PIF consists of three routines:

iW_init_db(*bytes*)

Sets up two buffers, each of size *bytes*. The maximum size is 4 Mbytes per buffer. The small window of the iSIB's master interface is mapped to VMEbus 32-bit extended address 0x80000000, corresponding to the current location of the CM/VMEIO memory. This value must be modified to accommodate any different memory placement. The small window is initialized to support 32-bit word access only. However, this may be modified to either 8-bit or 16-bit access[5]. Finally, ownership of the two buffers is assigned to the iWarp.

int * iW_grab_buff(*buff*); Returns a pointer to the requested buffer. Parameter *buff* is either 0 or 1. If the requested buffer is owned by the iWarp, a pointer to it is returned to the user. Otherwise, this routine uses the iMSG facility [7] to wait for a message from the iWarp's SUN host indicating that the buffer has been released by the Connection Machine. The iWarp's SUN host, in turn, waits (via an ethernet socket connection) for a message from the CM's SUN host indicating that the buffer has been freed.

int iW_release_buff(*buff*); Returns the ID of the other buffer. This routine releases ownership of the indicated buffer *buff*, allowing the CM to access it. This is accomplished by using the iMSG facility to send a message from the iSIB to the iWarp SUN host. The iWarp SUN host forwards this message via an ethernet socket connection to the CM SUN host.

3.2 Software on the CM VMEIO host/ iWarp host

The CM/VMEIO host is also the iWarp host, and is a SUN 4 Workstation. The purpose of this software is to act as a go-between, passing messages from the iSIB (via the iMSG protocol) to the CM's SUN 4 Workstation host (via ethernet). There is only one user routine that must be invoked:

host_init_db(*bytes*); Never returns.

This routine never returns to the user, and should be invoked as a background task under Unix. The parameter *bytes* is the size of each buffer, and must match the value used in **iW_init_db** (see Section 3.1), and **CM_init_db** (see Section 3.3). This code creates and binds a socket connection between the CM host and the VMEIO host (iWarp host). It also opens a shared memory connection between the iWarp host and the iSIB using an `img_open` call on the file `"/dev/iwusr0"`. It uses this shared memory (corresponding to the slave interface of the iSIB) to communicate with the iSIB. Two processes are forked off by this call. One process captures all messages sent to it from the CM host and forwards them to the iSIB. The other fields all messages from the iSIB and passes them on to the CM host.

3.3 Software on the CM host

The CM host software is essentially the mirror image of the iSIB software. However, since the user must invoke the system-defined CM/VMEIO routines[4], and perform data transformation tasks (to unscramble data in a serial format into data in a parallel format), two additional routines are included. The PIF routines are:

CM_init_db(*server*,*bytes*)

The parameter *server* is the network name of the VMEIO host. *bytes* is the size of each buffer. This initialization code: (1) opens an ethernet socket connection between the CM host and the VMEIO host; (2) issues CMFS_open calls to devices "sun4:/dev/vmemem0" and "sun4:/dev/vmemem1" for buffers 0 and 1, respectively; (3) assigns ownership of the two buffers to the iWarp.

CM_grab_buff(*buff*)

Waits until the CM host is the owner of buffer *buff*. This is accomplished by waiting for a message from the socket connection to the VMEIO host.

CM_release_buff(*buff*)

Releases buffer *buff* by assigning ownership to the iWarp and sending a message via the socket connection to the VMEIO host.

int CM_write_iW(*buff*, *data*, *len*) Returns ID of the other buffer. This routine grabs buffer *buff*. It uses CMFS calls [9] to transpose the parallel *data* (of size *len* bits per datum) and to then write this (serial formatted) data to the buffer. Finally, it releases the buffer and rewinds the vmemem device associated with the buffer.

int CM_read_iW(*buff*, *data*, *len*) Returns ID of the other buffer. Grabs buffer *buff*, reads the buffer, transposes the data into parallel format (parallel variable *data*, of *len* bits per datum), and finally releases the buffer.

CM_close_VME()

Closes the files associated with the buffers.

4. PARALLEL INTERFACE PERFORMANCE

The parallel interface performance has been evaluated in two different contexts. The first method was to measure the transfer rate achieved by the iSIB when writing to its small window, where the window has been mapped to the CM/VMEIO memory. The second method was to measure the end-to-end time transfer time of data from the iWarp array into the Connection Machine. This latter measure includes the time to convert the serial data to a parallel format suitable to the CM.

The iSIB test code used to measure the throughput of the small window (master interface) was written in C. Two different programs were tested. One wrote a constant zero value to the VMEIO memory; the second wrote a monotonically increasing integer value. Both

programs were compiled and executed with the optimize option disabled, and then enabled. Figure 7 indicates the transfer rate achieved for these various combinations.

	Time (seconds)	Rate (Mbytes/sec)
Unoptimized		
0 -> VME	18.5	1.81
Optimized		
0 -> VME	6.4	5.22
Unoptimized		
i -> VME	19.0	1.77
Optimized		
i -> VME	7.4	4.50
All cases are for a total of 33.5 Mbytes written to the small window of the iSIB master interface mapped to the 8 Mbyte CM/VMEIO memory board.		

Figure 7- PIF transfer rates in Mbytes/second.

The second method of evaluation measured the throughput of an end-to-end transfer of data from the iWarp to the CM. This performance metric includes all overhead costs, such as ethernet delays and serial-to-parallel data reformatting, and thus provides a true measure of achievable, and sustainable data transfer rates. The end-to-end test demonstrated a sustainable 3.5 Mbyte/second transfer rate between the iWarp and the Connection Machine.

It is appropriate to note that the efficiency of the PIF may be improved by passing the synchronization/control data directly over the VMEbus, obviating the (relatively) high-overhead associated with the ethernet connection. This consideration will be an important design goal when upgrading the PIF to support I/O transfers between the iSIB and a RIB, a VMEbus interface board (similar, in spirit, to the CM/VMEIO) that makes hydrophone data from a sensor array accessible to the VMEbus.

5. REFERENCES

1. Furnanz, L. , Kung H.T., et al, iWarp MacroArchitecture Specification, iWarp Tech Pub, INTEL Corp., Hillsboro OR, 1991.
2. Paris Reference Manual, Thinking Machines Corp., Cambridge MA, 1991.
3. Peterson, W., The VMEbus Handbook, VMEbus Int'l Trade Assoc, Scottsdale AZ, 1989.
4. Munoz, J. L., Bernecky W. R., *Invention Disclosure for Mapped Memory Interface for Communication between Multiple Computers*, NUWC Detachment New London CT, 1993.
5. Manseau, D. A., *iWarp Sun Interface Hardware External Product Specification*, INTEL Corp., Hillsboro OR, 1989.
6. Quinn, R., *User's Guide to the VMEI/O*, Thinking Machines Corp., Cambridge MA, 1990.
7. *IMSG manual pages*, iWarp Release 3.0, INTEL Corp., Hillsboro OR, 1992.
8. Network Programming Guide, #800-3850-10, Sun Microsystems Inc., Mountain View CA, 1990.
9. Connection Machine I/O System Programming Guide, Ver 6.1, Thinking Machines Corp., Cambridge MA, 1991.

6. APPENDIX A

/*

Advanced Sonar Processing Architectures
Parallel Interface Module
CM_iW_db.c

Author: W. R. Bernecky Sep '93
NUWC
New London, CT

Double buffer support routines. These are specialized to handle the case of the CM writing to/from 2 buffers on the iW board, and the iW host computer reading/writing from those same 2 buffers.

compile CM code: use CM's compiler on sun5
cc -DCM -O -c CM_iW_db.c
(include -lcmfs during link phase)

compile vme host:
cc -DVME -O -c CM_iW_db.c -I/iwarp/host/include -L/iwarp/host/lib -limsg

compile iWarp SIB code:
iwcc -O -c -o CM_iW_db CM_iW_db.c -lrts -limsg

Usage note: under C* the routines which expect CM_field_id_t vars must be passed the address of the C* variable.

*/

```
#define SIB 4
#define HOST 5
#define iW_owns (0x3)
#define CM_owns (0x7)
#include <stdio.h> /* everybody */
#ifdef __IWCC
#include <pathlib/pl.h> /* only SIB */
#include <asm/isib.h>
#else
#include <sys/types.h> /* only Sun */
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>
#include <signal.h>
#endif
#ifdef CM /* CM */
#include <cm/paris.h>
#include <cm/cm_stat.h>
#include <cm/cm_file.h>
#include <cm/vmemem.h>
#include <cm/cm_ioctl.h>
#else /* vme host or SIB */
#include <iwsys/iwsys.h>
#include <imsg/imsg.h>
#endif
```

```
static struct double_buffer_type
{
```



```

int *vme_mem_addr;    /* ptr into iW memory */
int socket;           /* ethernet connection to other machine */
int fd;               /* iW device file descriptor */
int owner;            /* owner of shared mem buffer: CM or iW */
} Data[2];

typedef struct
{
    int buff;          /* Buffer id= 0 or 1 */
    int value;         /* message value; either CM_owns or iW_owns */
} message;

#ifdef CM /* CM code */
void CM_grab_buff();
void CM_release_buff();
/*
-----
    Connection Machine Code:
        CM_init_db("sun4",1Mb)
        CM_read_iW(bufferID, parallel var, length of var)
        CM_write_iW(bufferID, parallel var, length of var)
        CM_close_VME()
-----
*/

/* Initializes double buffer scheme:
    opens socket connection between CM_host and VME_host
    opens 2 shared memory segments in VME I/O memory
    sets ownership of both buffers to iWARP
*/
int CM_init_db(server,bytes)
    char *server;
    int bytes;          /* size of each buffer */
{
    int code,j,sock,stat;
    struct sockaddr_in client; /* socket data structure */
    struct hostent *hp;        /* host addr of SERVER */

    /* open a socket connection between cmhost and vme host */
    client.sin_family= AF_INET;
    hp= gethostbyname(server);
    bcopy( (char *) hp->h_addr, (char *) &client.sin_addr, hp->h_length);
    j=stat= -1;
    while(++j<10 && stat<0)
    {
        if(j==5) sleep(1);
        Data[0].socket=
            Data[1].socket= sock= socket(AF_INET, SOCK_STREAM,0); /* create socket */
        client.sin_port= 4312; /* socket */
        stat= connect(sock,(struct sockaddr *) &client,sizeof(client));
    }
    if(stat<0)
        {perror("iW 2b: Connect error"); exit(-1);}

    CMFS_close_all_files();

```

```

/* open 2 buffers in the VME I/O memory */
stat=Data[0].fd= CMFS_open("sun4:/dev/vmemem0",CMFS_O_RDWR,0666);
if(stat<0) {CMFS_perror("Open err vmemem0"); exit(-1);}
stat=Data[1].fd= CMFS_open("sun4:/dev/vmemem1",CMFS_O_RDWR,0666);
if(stat<0) {CMFS_perror("Open err vmemem1"); exit(-1);}

if ((bytes%sizeof(int))!=0) /* integral number of words */
    bytes= ((int)(bytes/sizeof(int))+1)*sizeof(int);
code= 0xC0000000 | (sizeof(int)<< 16);
code |= CMVMEM_IOC_OPEN; /* change size from 8 MB */
stat= CMFS_ioctl(Data[0].fd,code,&bytes);
if(stat<0) CMFS_perror("ioctl for open0");
stat= CMFS_ioctl(Data[1].fd,code,&bytes);
if(stat<0) CMFS_perror("ioctl for open1");

code= 0xC0000000 | (sizeof(int)<<16);
code |= CMVMEM_IOC_OFFSET; /* move start of 2nd buffer */
stat= CMFS_ioctl(Data[1].fd,code,&bytes);
if(stat<0) CMFS_perror("ioctl for offset1");
Data[0].owner= Data[1].owner= iW_owns;
return(0);
}

/* This routine first grabs the buffer (if not already the owner)
   writes data from the CM into the buffer,
   and releases the buffer.
*/
int CM_write_iW(buff,data_field,len) /* CM writes to iW memory */
int buff; /* ... toggles buffer id */
CM_field_id_t data_field;
int len; /* size of data field */
{
    int stat,isync;

    CM_grab_buff(buff); /* make sure CM is owner */
    CMFS_cm_to_standard_byte_order(data_field,len);
    stat= CMFS_transpose_always(data_field,len,
        CMFS_IO_WRITE_TO_SERIAL_DATA,
        CMFS_write_to_row_major);
    if(stat<0) perror("CM write error");
    if((stat= CMFS_write_file_always(Data[buff].fd,data_field,len))<0)
        perror("CM write error");
    CM_release_buff(buff); /* release buffer to iWarp */
    /* rewind write ptr - perhaps this will change? */
    stat= 0xC0000000 | (sizeof(int)<<16);
    stat |= CMVMEM_IOC_REWIND;
    isync= 0;
    stat= CMFS_ioctl(Data[buff].fd,stat,&isync);
    if(stat<0) perror("CM rewind error");
    return((buff==0)?1:0); /* flip buffers */
}

/* If CM is not the owner of the buffer, this routine waits for a
   message from the socket.
*/

```

```

void CM_grab_buff(buff)
    int buff;
{
    message isync;

    while(Data[buff].owner!= CM_owns)
    {
        read(Data[buff].socket,&isync,sizeof(isync)); /* wait for iWarp */
        if((isync.buff==0)|| (isync.buff==1))
            Data[isync.buff].owner= isync.value;
        else
            perror("Synchronization problem: bad buff id\n");
        if((isync.value!=CM_owns)&&(isync.value!=iW_owns))
            perror("Synchronization problem: bad value\n");
    }
}

/* Sets the ownership of the buffer to iWarp. Sends a message via
the socket connection to the iWarp to let it know it is now
the owner of the buffer.
*/
void CM_release_buff(buff) /* makes buffer avail to iW host */
    int buff;
{
    message isync;

    isync.buff= buff;
    isync.value= iW_owns;
    if(Data[buff].owner==CM_owns) /* should always be so */
        write(Data[buff].socket,&isync,sizeof(isync)); /* tell iWarp the news */
    Data[buff].owner= iW_owns;
}

/* This routine first grabs the buffer (if not already the owner)
reads data from the VME I/O shared memory into the CM
and releases the buffer.
*/
int CM_read_iW(buff,data_field,len) /* CM reads iW memory */
    int buff,len;
    CM_field_id_t data_field;
{
    int stat,isync;

    CM_grab_buff(buff);
    stat= CMFS_read_file_always(Data[buff].fd,data_field,len);
    if(stat<0) perror("CM read error");
    /* data comes in slice-wise. Convert to field-wise */
    stat= CMFS_transpose_always(data_field,len,
        CMFS_IO_READ_FROM_SERIAL_DATA,
        CMFS_read_from_row_major);
    if(stat<0) perror("CM read error");
    CMFS_cm_to_standard_byte_order(data_field,len);
    CM_release_buff(buff);
    return(buff?0:1);
}

/*

```

a nice thing to do, else the fileserver will eventually complain

```

*/
void CM_close_VME()
{
    CMFS_close(Data[0].fd);
    CMFS_close(Data[1].fd);
    CMFS_close_all_files();
}

#endif

#ifdef __IWCC__
/*
-----
    This code runs on iWarp SIB
        iW_init_db(1Mbytes)
        ptr= (int *) iW_grab_buff(bufferID)
        for(i=0;i<N;i++) *ptr++ = data;           move data to VME memory
        bufferID= iW_release_buff(bufferID)       flips bufferID
-----
*/
#define CM_ADDR          (0x80000000)    /* addr of vme memory */
#define CM_ADDR_SWR      (0x8000 | AM_UEDATA) /* 0x8009 value for iSIBs SWR */
#define ONE_MWORD        (1024*1024)

/* Initialize double buffer scheme on SIB:
   sets up pointers to VME I/O memory
   sets small window control registers
   sets ownership of buffers to IWARP
*/

int iW_init_db(bytes)
    int bytes;
{
    int words,*vme_ptr;

    words= bytes/sizeof(int);
    if (words*sizeof(int) != bytes) words++;
    if (words>ONE_MWORD) fprintf(stderr,"More than 4 Mbytes per buffer err\n");
    /* map SIB small window to CM's vmeio memory */
    vme_ptr= (int *) (SIB_SWINDOW | (SIB_MASK_SWINDOW & CM_ADDR));
    Data[0].vme_mem_addr= vme_ptr;
    vme_ptr+= words;           /* ptr to 2nd buff */
    Data[1].vme_mem_addr= vme_ptr;
    Data[0].owner= Data[1].owner= iW_owns;    /* both buffers are mine */

    /* set up iSIB's small window control registers */
    SWR= CM_ADDR_SWR;          /* pt to VME I/O shared memory */
    SSR= SSR_WORD;             /* word access only */
    return(0);
}

/* if the iWarp does not own the buffer, this routine waits
   for a message from the VME host (which acts as a go-between)
   iWarp SIB <----> VME host <----> CM host

```

```

    RETURNS POINTER TO VME I/O MEMORY
*/
int *iW_grab_buff(buff)
    int buff;
{
    int stat;
    message isync;
    imsg_id mid;
    imsg_status isp;

    while(Data[buff].owner!=iW_owns)
    {
        stat= imsg_block(IMSG_PORT_USR); /* wait for a message */
        if(stat<0)
        { fprintf("Grab block err\n"); exit(-1); }
        mid= imsg_recv(IMSG_PORT_USR, (char *)&isync, sizeof(isync)); /* read msg */
        if(mid==IMSG_BADID)
        { fprintf("Grab recv err\n"); exit(-1); }
        do stat= imsg_stat(mid, &isp); while ((stat>=0)&&(!isp.is_done));
        if((stat<0)&&isp.is_error)
        { fprintf("Grab stat err\n"); exit(-1); }
        Data[isync.buff].owner= isync.value; /* grab buffer */
    }
    return(Data[buff].vme_mem_addr); /* pointer to buffer */
}

/* releases ownership of a buffer to the CM.
   FLIPS BUFFER ID
*/
int iW_release_buff(buff)
    int buff;
{
    int stat;
    message isync;
    imsg_id mid;
    imsg_status isp;

    isync.buff= buff;
    isync.value= CM_owns;
    if(Data[buff].owner== iW_owns) /* should always be so */
    {
        mid= imsg_send(HOST,IMSG_PORT_USR, (char *)&isync, sizeof(isync));
        if(mid==IMSG_BADID)
        { fprintf("Rel send err\n"); exit(-1); }
        do stat= imsg_stat(mid, &isp); while ((stat>=0)&&(!isp.is_done));
        if((stat<0)&&isp.is_error)
        { fprintf("Rel stat err\n"); exit(-1); }
    }
    Data[buff].owner= CM_owns;
    return(buff?0:1);
}

#endif

```

#ifdef VME

/*

=====

This code runs on Sun VME I/O host
host_init_db(bytes)
NEVER RETURNS!

*/

int host_init_db(bytes)

int bytes; /* size of each buffer */

{

int code,i,sock,stat,size,fd;

int proc0,proc1,proc2;

int *memptr; /* pts to SIB/VME dual-ported mem */

struct sockaddr_in client; /* socket data structure */

/* open socket connection between VME I/O host and CM host */

client.sin_family= AF_INET;

client.sin_addr.s_addr= INADDR_ANY;

sock= socket(AF_INET,SOCK_STREAM,0); /* create a socket */

if(sock<0) {printf("Socket error\n"); exit(-1);}

client.sin_port= 4312; /* socket */

bind(sock,(struct sockaddr *) &client,sizeof(client));

listen(sock,1); /* use the socket */

Data[0].socket= Data[1].socket=

accept(sock,(struct sockaddr *) 0,(int *) 0);

/* open shared mem connection between VME I/O host and SIB */

stat= imsg_open("/dev/iwusr0",&fd,&memptr,&size,HOST); /* open shared mem on iW:

if(stat<0) perror("Can not open iWarp mem");

/* dual-port mem: 0 buffer id from iWarp: 0 or 1; -1 -> exit

1 value (iW_owns or CM_owns)

2 buffer id into iWarp

3 value (iW_owns or CM_owns)

*/

Data[0].vme_mem_addr= (int *) memptr;

/* read out area */

Data[1].vme_mem_addr= (int *) (memptr+2);

/* send in */

/* start processes to act as communication agents between

SIB and CM host */

if (proc0= fork()) iW_to_CM();

/* never return */

else

CM_to_iW();

}

/* Field any messages from iWarp's SIB and pass to CM.

*/

int iW_to_CM()

{

int stat,*data;

message isync;

imsg_id mid;

imsg_status isp;

data= Data[0].vme_mem_addr; /* pointer to shared mem */

for(;;)

```

{
    stat= imsg_block(IMG_PORT_USR);
    if (stat<0) { perror("Error iW_to_CM: imsg_block\n"); kill(0,SIGKILL);}
    mid= imsg_recv(IMG_PORT_USR, (char *) data, sizeof(isync));
    if(mid==MSG_BADID) {perror("iW_to_CM bad message id\n"); kill(0,SIGKILL);}
    do stat= imsg_stat(mid,&isp); while ((stat>=0)&&(!isp.is_done));
    if (((*data!=0)&&(*data!=1))||((stat<0)&&isp.is_error))
    {
        if(*data== -1)
            printf("Exiting...\n");
        else
            perror("iW_to_CM: bad data\n");
            kill(0,SIGKILL);
    }
    isync.buff= *data;          /* buffer id */
    isync.value= *(data+1);
    stat= write(Data[*data].socket,&isync,sizeof(isync)); /* pass on to CM */
    if (stat<0) { perror("Error iW_to_CM: socket\n"); kill(0,SIGKILL);}
} /* forever */
}

int CM_to_iW()
{
    int stat,*data,i;
    message isync;
    imsg_id mid;
    imsg_status isp;

    for(;;)
    {
        data= Data[1].vme_mem_addr; /* pointer to send into iwarp mem */
        stat= read(Data[0].socket,&isync,sizeof(isync)); /* wait for CM */
        *data= isync.buff;          /* buff id */
        *(data+1)= isync.value;     /* data value */
        mid= imsg_send(SIB, IMG_PORT_USR, data, sizeof(message)); /* to iWarp */
        if((mid==MSG_BADID)|| (isync.value== -1))
        {
            if (isync.value== -1) printf("exiting due to CM...\n");
            else
                perror("iW_to_CM: bad data\n");
                kill(0,SIGKILL);
        }
        do stat= imsg_stat(mid,&isp); while ((stat>=0)&&(!isp.is_done));
        if(stat&&!isp.is_error)
        { perror("CM->iW stat error\n"); kill(0,SIGKILL); }
    } /* for ever */
} /* CM_to_iW */

#endif

```

DISTRIBUTION LIST

Internal Distribution

Code

0261	(NL Library (2))
0262	(NPT Library (2))
2094	(J. DePrimo)
21	(W. Coggins)
2121	(W. Fischer, M. Schindler)
2122	(J. O'Sullivan)
2123	(G. Bowman, R. Choma, S. Dzerovych, J. Ianniello, J. Law M. Maguire, N. Owsley, T. Tetlow)
2133	(H. Schloemer)
2141	(P. Davis)
2142	(D. Abraham)
2143	(J. Marsh)
215	(H. Watt)
2151	(T. Choinski, D. DaRos, D. Organ)
2152	(T. Anderson, I. Ferber, R. Latourette)
2153	(W. Bernecky (5), A. Edmonds, S. Harrison, R. Howbrigg, J. Ionata, B. Iwatake, L. Karasevich, M. Krzych, J. Munoz)
2154	(G. O'Brien)
2191	(R. Murdock)

External Distribution

K. Bromley
NRAD
San Diego, CA

T. DeYoung
ARPA
3701 N. Fairfax Dr.
Arlington, VA

B. Wald
ONR
800 N. Quincy St.
Arlington, VA

B. Wasilauski
NRAD
San Diego, CA

Total: 47