1 iWarp System Overview

This chapter provides an overview of the iWarp system to help you understand how to develop iWarp applications. This chapter describes the following parts of the iWarp system:

- the iWarp component
- communication in the iWarp system
- the iWarp programming environment
- the Programmed Communication Services (PCS)

iWarp Component

iWarp systems can support a wide range of configurations, from special-purpose systolic arrays to general-purpose distributed memory computers. The iWarp system simultaneously supports both fine-grained and course-grained distributed computation models in the same system. An iWarp system can be expanded to include 1,024 cells.

The iWarp component is the basic building block of the iWarp system. It is a custom VLSI single-chip processor that consists of approximately 600,000 transistors. The iWarp component architecture is designed for high-speed signal, image, and scientific computing.

Each iWarp component is a single-chip processor that requires only the addition of local memory to form a complete system building block that we call an iWarp cell. Figure 1-1 illustrates an iWarp cell.

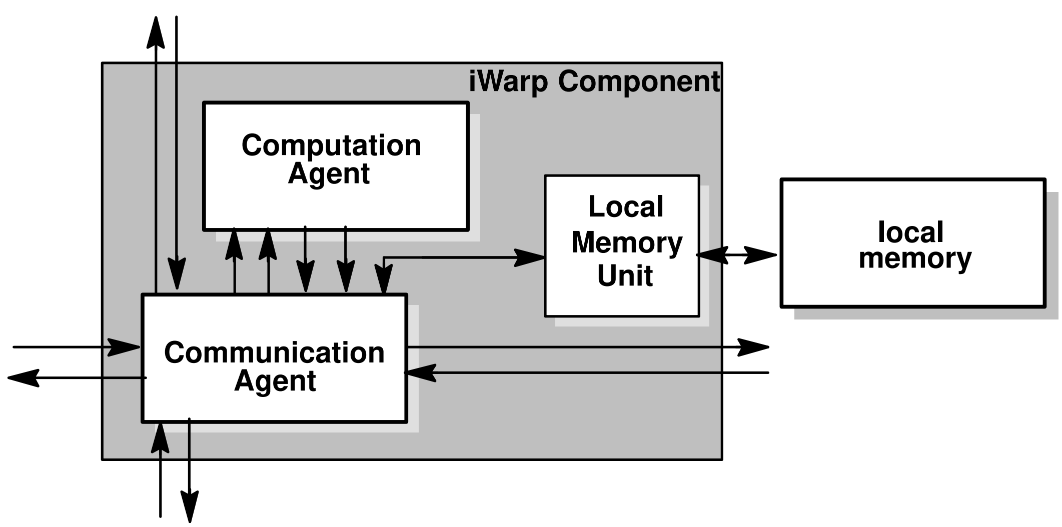

Figure 1-1: iWarp cell

The iWarp component contains three distinct sub-components:

- the computation agent
- the communication agent
- the local memory unit

The computation agent performs the normal, explicitly programmed computation. The communication agent handles input from and output to adjacent cells. The local memory unit provides access to local memory for both the computation and communication agents.

Independent communication and computation agents allow the communication activities of a cell to proceed without disturbing computation. Closely-coupled yet independently-controlled agents make it possible to efficiently overlap communication and computation, and provide greater efficiency for random communication. Nonadjacent cells in the array communicate without disturbing the computation on intermediate cells.

Figure 1-2 provides a block diagram of the iWarp component architecture.

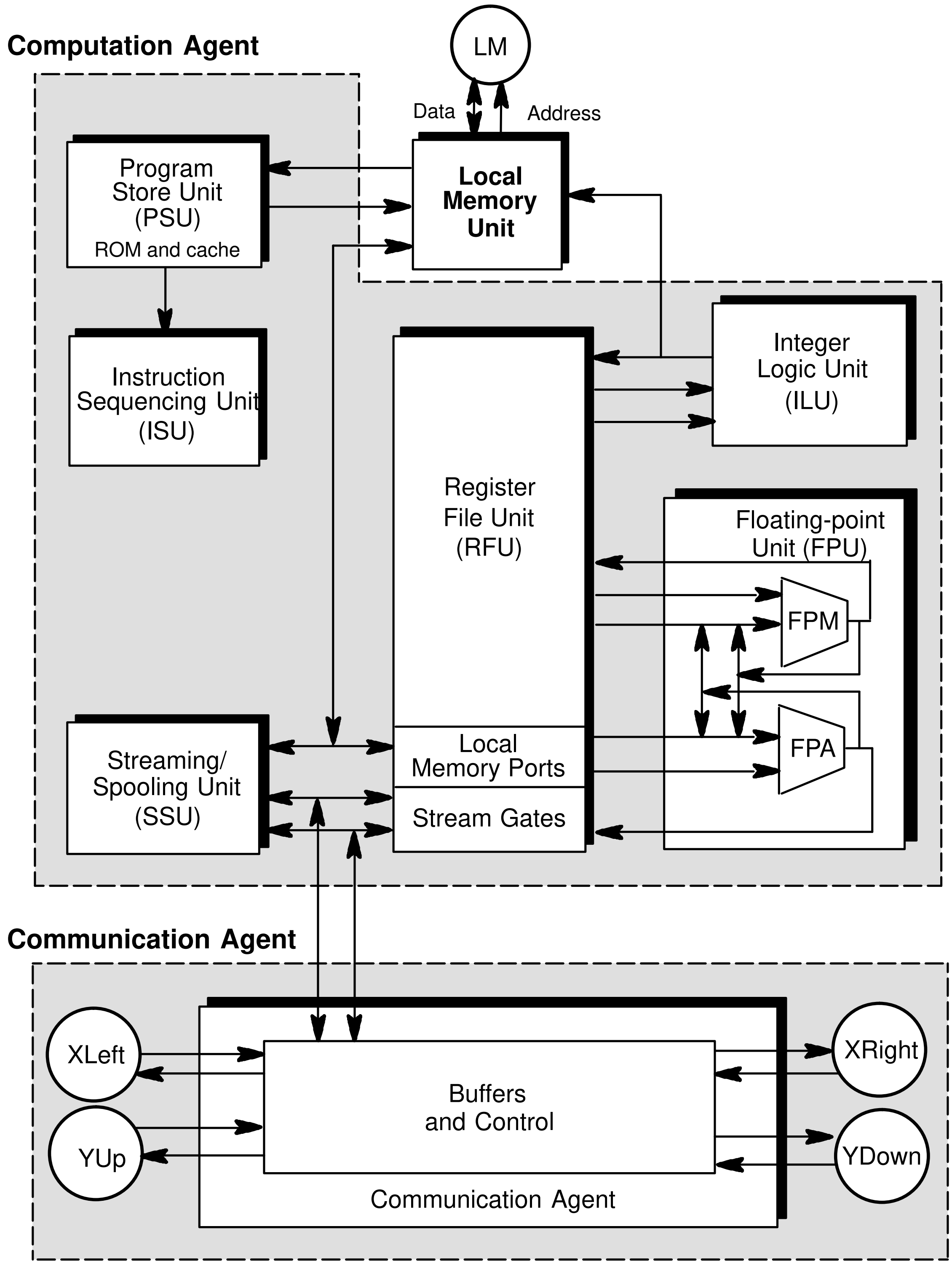

Figure 1-2 : iWarp component architecture block diagram

Communication in the iWarp System

The iWarp system supports both message-based and program-based (systolic) communication, thus providing support for a wide range of programming models and applications. You are not limited to a single communication style within an application however. When two cells communicate, one can perform message-based communication, while the other performs program-based communication.

The iWarp architecture also has two important innovations to support communication flexibility. These innovations are program access to communication and logical channels. Chapter 4 provides a detailed discussion of communication in the iWarp system, and describes the communication tools available to the programmer.

NOTE

The iWarp runtime system provides a reliable message-passing interface, but this interface is limited. For a description of the interface, refer to the iWarp `imsg` manual page.

Message-Based Communication

In message-based communication (also known as memory-based communication), the unit of communication and processing is a complete message. A complete message is built in the local memory of the sending cell before it is sent to the receiving cell. Only after the entire message has been received by the local memory of the receiving cell does the program running on the receiving cell operate on the data.

In message-based communication, messages are built in the local memories of cells and delivered by independent communication software, so the program running on a cell is not involved in communication. This separation of communication and computation allows a cell to operate on its local data while a message is delivered and buffered through its local memory.

Message-based communication is appropriate for applications that do not assume detailed knowledge about inter-cell communication. While message-based communication provides the advantages of separate communication and computation and ease of use, these advantages incur higher communication overhead.

Program-based Communication

In program-based communication (also known as systolic communication), the unit of communication and processing is a single word transfer. Each cell processes data immediately as each word arrives. The cell then sends the results of the computation to other cells as data of outgoing messages. This creates a long data stream flowing through the iWarp array.

Program-based communication provides the following advantages:

- fine-grained communication
- reduced access to local memory
- increased instruction-level parallelism
- reduced size for local memory

With these advantages comes increased complexity. Message queues in the communication agent of a cell are accessed sequentially, therefore the programmer must ensure that the reads and writes of message queues are properly sequenced so the correct data item appears at the front of the queue when it is supposed to be read.

The PathLib communication library contains routines for implementing program-based communication. Refer to Chapter 4 for a description of PathLib.

Program Access to Communication

For program-based communication, the iWarp system gives programs running on the computation agent direct access to the inputs and outputs of message queues in the communication agent. These locations can be bound to special registers, called gates, in the register file. Reading from a gate corresponds to receiving data from the queue, and writing to a gate corresponds to sending data to the queue.

In addition to supporting program-based communication, direct program access to the message queues can also speed up message-based communication. Message-passing communication is usually implemented by station-to-station delivery, where messages are copied from user space to system space in the sending cell, transmitted, and copied from system space to user space at the receiving cell. Direct access to message queues allows door-to-door delivery, where data is sent directly between the user spaces of the sending and receiving cells.

The iWarp program also provides program access to data routing. Under normal operation, the communication agent establishes a route between the sending and receiving cells, and each word of a message follows the same route. The iWarp system allows a program to alter the route in the middle of a message, so the rest of the data can be sent to another cell using a different route. This program access to data routing reduces the need to buffer data in memory, thus reducing size requirements for local memory.

Logical Channels

Logical channels allow extensive interconnection of cells in an iWarp array by providing multiple logical connections over a single physical connection. Up to 20 logical channels can be multiplexed over each of the the eight external physical busses in each cell. This allows the implementation of algorithms that require more cell connections than can be physically provided by the hardware. This high degree of connectivity is essential for program-based communication, since each cell may need to have simultaneous connections to several cells.

Logical channels are set up by the program executing on the sending cell. Once a logical channel is established, it remains until it is explicitly taken down by the cell that established it. This long-lived connection decreases communication latency since no further overhead is associated with the transfer of data once a logical channel is established. Logical channels allow non-adjacent cells to communicate as if they were adjacent, and they allow efficient simulation of communication network topologies that do not map directly onto the underlying hardware.

Logical channels can also be used to guarantee communication bandwidth for special classes of messages between a set of selected cells. The time-multiplexing of logical channels onto physical busses is scheduled such that a minimum bandwidth is guaranteed to be available to each logical channel, and therefore to the messages using that logical channel. Also, when a logical channel is idle, it does not consume any bandwidth, and the bandwidth reserved for it can be used by other logical channels.

Programming the iWarp System

The iWarp system provides a broad base of program development and file manipulation tools to develop application programs, as well as an execution environment for executing programs.

Programming Tools

The iWarp program development tools include the following:

- C compiler
- assembler
- linker
- disassembler
- profiler
- debugger
- archiver
- symbol table lister
- dumper
- sizer

These tools are described in Chapter 2 "Program Development Environment." To view the manual pages for the iWarp program development tools, use the iwman command. For information on the iwman command, see Appendix A "Commands Quick Reference."

Execution Environment

The iWarp Run Time System (iW/RTS) provides the execution environment for programs on the iWarp system. iW/RTS provides lgo to execute programs, and iW/RTS also includes the iWarp monitor, which provides a set of commands to develop array-program execution utilities.

Refer to Chapter 3 for a complete description of iW/RTS.

Parallel Program Generators: Apply

Each parallel program is based on a programming model that defines the performance characteristics for the parallel computer. For example, a programming model that can be used for image processing is the domain partition model. With the domain partition model, a problem domain, such as a grid space corresponding to an image, is partitioned so that each cell handles a subdomain.

Typically, when a programmer designs a parallel program, the programmer must build both the program and the programming model. This adds to development time and can result in a program that is difficult to change since the programming model is specific to the program.

For this reason, parallel program generators have been designed to improve parallel programming. Parallel program generators are application-specific tools that translate a user program into programs for each cell in the system, thus incorporating the low-level constructs required to implement the parallel programming model for that application.

The Apply parallel program generator is available with the iWarp system for image processing applications. Apply allows the programmer to write high-level image processing programs without having to be concerned with the implementation of the programming model.

The Apply programming language is for programming image-to-image computations where pixels of the output images are computed from corresponding rectangular windows of the input images. Each Apply procedure operates on a single pixel position. The Apply compiler generates a parallel program that executes over the entire image on an iWarp array. An Apply procedure does the following:

- accepts a pixel or window of pixels from each input image
- performs a computation on the input
- returns a pixel value for each output image

The following example illustrates an Apply program that averages pixel values from an input image to get a pixel value for the output image.

```
procedure avg(  imagein  : in array (−1..1, −1..1) of byte border 0,
                imageout : out byte)
is
begin
   imageout := (imagein(−1,1) + imagein(0,0) + imagein(1, −1)) / 3;
end avg;
```

Figure 1-3 provides an illustration of the operation performed by the Apply program in the example.

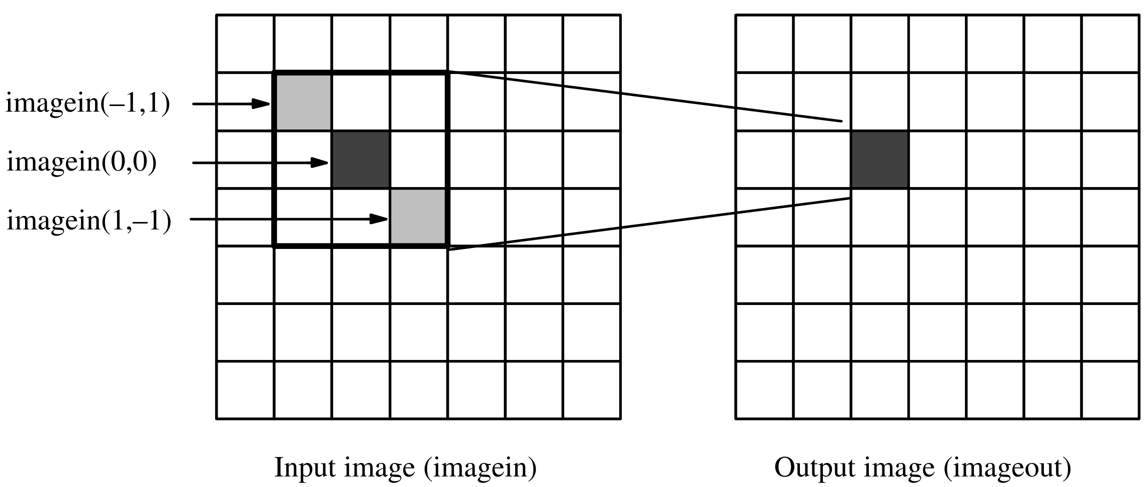

Figure 1-3: Apply Averaging Operation

You call Apply procedures from a C main program and pass the input and output images and other parameters as arguments. On the iWarp system, you can generate both single-cell (sequential) programs and multi-cell (parallel) programs with Apply.

For a complete description of the Apply language and parallel program generator, refer to *The Apply Manual.*

For descriptions of the tools used to develop programs on the iWarp system, see Chapter 2 "The Program Development Environment."

Programmed Communication Services (PCS)

The Programmed Communication Services (PCS) provides a set of tools and libraries for developing iWarp applications. PCS lets you do the following in iWarp applications:

- write cell programs that run in parallel on iWarp cells
- build cell networks for distributed parallel processing on iWarp arrays
- transfer data between cells in the cell network
- use systolic communication for data transfer
- use background processes and spooling for optimizing communication operations

The PCS libraries are a set of functions and macros for developing your PCS applications. The PCS libraries include the following:

- the data movement library
- the combiner library

You use the PCS data movement library to create cell programs. This library provides functions to do the following:

- initialize cell programs
- send data to other cells
- receive data from other cells
- spool data transfers using background processes
- detect runtime errors in applications

You use the PCS combiner library to create array programs. This library provides functions to create, configure, and reserve resources for cell networks.

For more information about the PCS data movement library and the combiner library, see Chapter 5, "Program Communication: PCS."

The PCS tools are specifically designed for developing PCS applications. The PCS tools include the following:

- PCS preprocessor (iwpp)
- PCS array combiner (iwcb)
- PCS array loader (iwamld)

You use these tools to do the following activities:

- optimize the PCS data movement calls in your source code
- create executable array modules that run on iWarp systems
- load and run the array modules on an iWarp system

For more information about the PCS tools, see Chapter 6, "PCS Tools."

Building PCS Applications

You use the PCS tools to develop PCS applications. Figure 1-4 shows the sequence for building PCS applications.

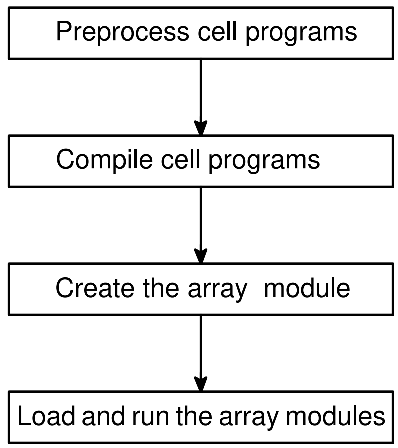

Figure 1-4: PCS Development Process

Develop your source code using the PCS libraries, then build your application program. Build PCS applications by following these steps:

1. Before compiling cell programs, use the PCS preprocessor to preprocess the PCS code in the source code files. The preprocessor converts PCS code into equivalent inline code. This avoids function call overhead and the code runs more efficiently. This is an optional step.
2. Compile the source code with the iWarp C compiler to produce the cell program. Compile each cell program separately without regard to code placement in the array module.
3. Create an array module using the PCS array combiner. The array combiner compiles and runs the array program to create the array module. This combines the application's cell programs, existing array modules, and a network description file to create an array module.
4. Load and run the array module on the iWarp system using the PCS array loader. You can run the application with the array loader or use a run script to run it later.

For more information about developing PCS applications, see the examples in Chapter 7, "Developing iWarp Programs."