1 *Overview*

Audience

This manual is designed for programmers who are developing parallel image processing applications for the iWarp supercomputer. To use Adapt, you should have a good knowledge of the C programming language.

Using This Manual

This manual is intended for the new or experienced Adapt user, and contains both tutorial and reference information. The manual is structured as follows:

Chapter 1:	Provides an overview of Adapt and this manual.
Chapter 2:	Describes the Adapt language.
Chapter 3:	Describes how to develop and execute an Adapt program.
Appendix A:	Provides a definition of the Adapt language in the Extended Backus-Naur form.
Appendix B:	Describes WebLib, a library of image processing routines.

Conventions

Documentation conventions

- *italic type* indicates items you must provide.
- `courier font` is used for program and command examples.

In a syntax description:

- **[ ]** Square brackets enclosing qualifiers indicate the qualifiers are optional.
- **{ }** Braces enclosing qualifiers indicate that one or more of the qualifiers is required.
- **|** A vertical rule separating qualifiers indicates that only one of the qualifiers can be selected.
- **...** A horizontal ellipsis indicates that the preceding optional item can appear one or more times in succession.
- **⋮** A vertical ellipsis indicates the continuation of statements within a program example.
- Spaces are used between entities to improve readability in both syntax descriptions and programs but unless otherwise noted have no significance.

List of Related Documents

The following documents provide more information about the iWarp system:

- *Introduction to iWarp*
- *iWarp Programmer's Guide*
- *iWarp C User's Guide*
- *iWarp FORTRAN User's Guide*
- *iWarp Debugger Manual*
- *Assign User's Guide*

Adapt Overview

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. Adapt is an application-specific parallel program generator for image processing that translates a user's sequential program into programs for each cell in the system. Adapt allows the programmer to write high-level image processing programs without having to be concerned with the implementation of the programming model.

The Adapt language is based on the split and merge programming model. In this model, an image is divided into parts, each part is processed independently, and the results are combined. The Adapt programmer simply provides a local operator, which is applied to every pixel, and a combine operator that merges the independent results into one.

You call Adapt procedures from a C main program, passing 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 Adapt.

Adapt is an extension of the Apply language. Apply is only capable of performing local image processing operations such as edge detection, smoothing, and thresholding. In addition to local operations, Adapt can also perform global image processing operations such as

- connected components
- run-length encoding
- image warping

With minor syntax changes, you can modify Apply programs to be compiled by the Adapt compiler. You can make these changes automatically by running the app2ad program. This program is described in Chapter 3.

Adapt Functions

Adapt image processing functions consist of the following sections:

First	An initialization function that can be run at the beginning of any row of an image.
Next	A raster-order function that is applied across an image. This section of the Adapt function is the only section required if you are implementing an Apply-style local operator program.
Combine	A merging function that combines the outputs of two regions of an image to produce a final output for the image. You can only use the combine function on groups of consecutive rows of an image.
Last	A termination function that is applied after the output of an entire image is computed.

There are four types of parameters in Adapt programs:

- input images
- output images
- global input parameters
- global output parameters

Input and output images are iterated over by the *Next* function. Global input parameters are broadcast to all instances of *Next*, and are also available during *First*, *Combine*, and *Last*. Their values cannot be changed. Global output parameters are given values as a result of processing in *First*, and *Next*. Global output parameter values from two different regions of an image are combined in the *Combine* function, and the *Last* function can also modify these values.

It is not necessary to include all of the sections in every Adapt program. If images or images and global input variables are being processed, only *Next* is required. If images and global output variables are being processed, both *Next* and *Combine* are required. *First* and *Last* are always optional.

Adapt Examples

The following Sobel edge detection example illustrates the use of Adapt. In the example, Sobel edge detection is performed by convolving the input image with two 3 x 3 masks. The horizontal mask measures the gradient of horizontal edges, and the vertical mask measures the gradient of vertical edges. Diagonal edges are operated on by both masks, allowing edge orientation and strength to be measured for all edges. Figure 1-1 illustrates the masks.

$$\begin{array}{cc} \begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix} & \qquad \begin{vmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{vmatrix} \\ \text{HORIZONTAL} & \qquad \text{VERTICAL} \end{array}$$

Figure 1-1: Sobel Convolution Masks

The following code illustrates the Adapt implementation of the Sobel edge detection. This Adapt program contains only a *Next* section, because only images are being processed. The program defines a rectangular window around the current pixel in the input image that is accessed by the *Next* function. This is an example of an Apply-style local operator program.

The lines are numbered in comment form to help explain the program, but line numbers are not part of the Adapt language.

```
procedure sobel (inimg   :  in image array (-1..1, -1..1)   --1
                                   of byte
                               border 0,
                  threshold: float,
                  magnitude: out float)

is                                                          --2

   horiz, vert  :  integer;                                 --3

next                                                        --4

begin                                                       --5

   horiz := inimg(-1,-1) + 2 * inimg(-1,0) + inimg(-1,1)    --6
         - inimg(1,-1) - 2 * inimg(1,0) - inimg(1,1);

   vert  := inimg(-1,-1) + 2 * inimg(0,-1) + inimg(1,-1)    --7
         - inimg(-1,1) - 2 * inimg(0,1) - inimg(1,1);

   magnitude  :=  sqrt(FLOAT(horiz) * FLOAT(horiz)          --8
                  + FLOAT(vert) * FLOAT(vert));

   if magnitude < threshold then                            --9
      magnitude := 0.0;                                     --10
   end if;                                                  --11
end next                                                    --12

end sobel;                                                  --13
```

Line 1 defines the input, output, and constant parameters to the procedure. The input parameter *inimg* is an input image with a 3 x 3 window. The constant parameter *threshold* is a threshold value. Edges that are weaker than this threshold are suppressed in the output magnitude image, *magnitude*.

Line 1 also defines the input image window. It is a 3 x 3 window centered about the current pixel processing position. **BORDER 0** specifies that the border values are filled with 0 when the window extends beyond the edges of the image. The border keyword is described fully in Chapter 2.

Lines 2, 4, 5, 12, and 13 contain keywords that define separate segments of the Adapt procedure.

Line 3 defines the local variables *horiz* and *vert*, which hold the results of the horizontal and vertical Sobel edge operator.

The computation of the Sobel convolutions is implemented in the expressions on lines 6 through 8. Suppression of edges that are weaker than the threshold is implemented on lines 9 through 11.

The next example is an Adapt implementation of a histogram. This example incorporates all of the sections of an Adapt function, because both images and output global variables are used.

```
procedure histogram ( img : in image byte,                      --1
                      hist: out array(0..255) of float)         --2

is                                                              --3
   count: array(0..255) of integer;                             --4

first                                                           --5
begin                                                           --6
   for i in 0..255 loop                                         --7
      count(i) := 0;                                            --8
   end loop;                                                    --9
end first;                                                      --10

next begin                                                      --11
   count(img) := count(img) + 1;                                --12
end next;                                                       --13

combine                                                         --14
begin                                                           --15
   for i in 0..255 loop                                         --16
      count(i) := count(i) + _count(i);                         --17
   end loop;                                                    --18
end combine;                                                    --19

last                                                            --20
   pixels: integer;                                             --21
begin                                                           --22
   pixels := 0;                                                 --23
   for i in 0..255 loop                                         --24
      pixels := pixels + count(i);                              --25
   end loop;                                                    --26
   for i in 0..255 loop                                         --27
      hist(i) := float(count(i)) / pixels;                      --28
   end loop;                                                    --29
end last;                                                       --30

end histogram;                                                  --31
```

The procedure header (lines 1 and 2) defines one input image and one global output variable, the histogram. The output variable is declared to be a floating-point value, because the final histogram resulting from the program is a frequency of each pixel rather than a pixel count. Line 4 declares an internal global program variable that is available to all sections of the program.

The *First* section (lines 5 through 10) zeroes an integer histogram called *count*. The *Next* section (lines 11 through 13) increments *count* for the current pixel. The *Combine* section (lines 14 through 19) adds the *count* arrays from the upper and lower regions of the image and stores the result in the histogram for the merged region. Note on line 17 that the *count* variable for the lower region is preceded by an underscore to distinguish it from the upper region variable. A variable preceded by an underscore is by definition a global variable in Adapt. The underscore variable is the means by which the count from another cell is obtained for combining results. The *Last* section (lines 20 through 30) adds all the histogram elements to get the total number of pixels in the image, and divides the frequency count table by this total to get the final frequency count.

Chapter 2 provides a complete description of the Adapt language.