3 Developing Adapt Programs

This chapter describes how to compile, link, and load an Adapt procedure for parallel execution. The first step is to translate the Adapt procedure into a parallel representation with the Adapt compiler. The compiler generates two cellprograms: one for the master cell and one for the slave cells. These cell programs and the C main program are passed to the C compiler, where they are compiled and linked to form executable cellprograms. Figure 3-1 illustrates these steps.

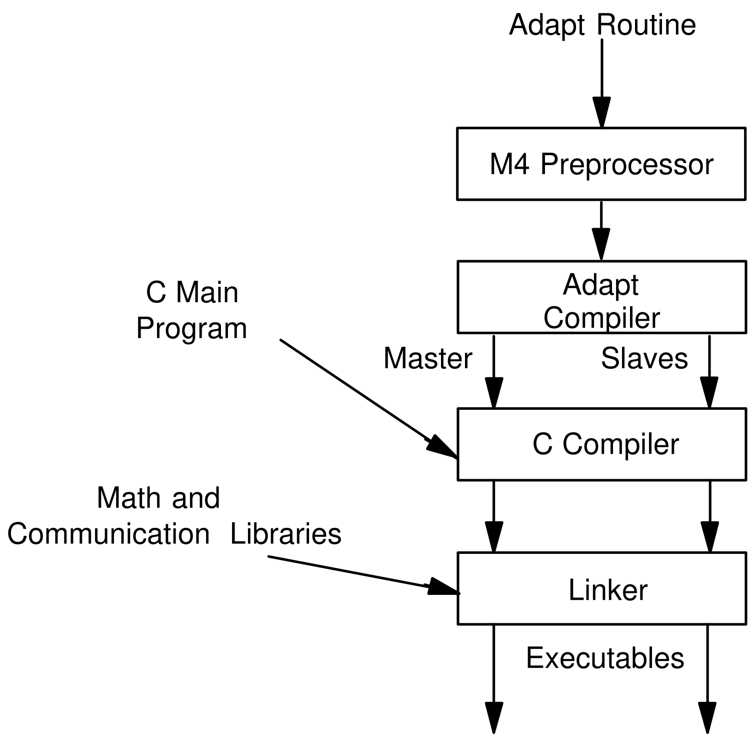

Figure 3-1: Adapt Parallel Compilation and Linking Process

NOTE

All Adapt procedures pass through the m4 preprocessor, allowing you to use m4 commands when writing Adapt procedures.

Compiling and Executing

The Adapt command invokes the Adapt compiler. The syntax for the Adapt command is as follows:

```
adapt [option ... ] file.ad ...  main.c
```

file.ad is the name of an Adapt procedure. The .ad file extension specifies that *file* is an Adapt source file. You can include more than one Adapt procedure on the command line. *main*.c is the main C program that calls the Adapt procedures. There must be only one main program on the command line, and it must come last.

The default output of the Adapt compiler is two iWarp executable files and a ‘‘go’’ script to load and run the executables on the iWarp array.

option can be any of the following:

-a	Stop after Adapt creates the parallel C programs. This allows advanced users to build their own programs or modify generated code. If you use the -a option, you do not need to specify a C main program on the command line.
-c *num*	Set the number of columns to compile for. The default is 2. The Adapt compiler generates code for a rectangular array of iWarp cells. The size of the rectangle is specified by the -c and -r options.
-H *host_name*	Specify the iWarp array attached to *host_name* as the target array. By default, the *host_name* is the value of the environment variable IWARP_HOSTNAME. If this variable is not set, *host_name* defaults to the current machine.
-I*pathname*	Add *pathname* to the list of directories to be searched for #include files with relative pathnames.
-L*dir*	Add *dir* to the list of directories to be searched for libraries. This option is passed directly to the array module loader (iwamld).
-l*x*	Link library lib*x*.a to the current program. This option is passed directly to the array module loader (iwamld).
-o *name*	Specify *name* as the name of the output file.
-r *num*	Set the number of rows to compile for. The default is 2.
-s	Generate sequential C code.
-S *sib_num*	Specify SIB number *sib_num* on the target iWarp array. By default, *sib_num* is the value of the environment variable IWARP_SIBNUM. If this variable is not set, *sib_num* defaults to 0.
-V	Print version information for each tool and perform compilation as specified.
-VV	Print version information for each tool but do not perform compilation.

-W*pass arg* [, *arg* ...]	Hand off the command specified by *arg* to *pass*, where *pass* is one of the following:
	u — compilation of main program (iwcc)
	m — compilation of Adapt master program (iwcc)
	s — compilation of Adapt slave program (iwcc)
	b — combiner (iwcb)
	l — array module loader (iwamld)
	q — compilation of sequential C programs for Sun host (cc)
-x	Print expanded compilation commands before executing.
-y	Print expanded compilation commands, but do not execute.
-Y*pass dir*	Get Adapt facility *pass* from directory *dir*, where *pass* is one of the following:
	a — Adapt
	c — iWarp C compiler (iwcc)
	b — combiner (iwcb)
	l — array module loader (iwamld)
	d — data movement library
	q — host C compiler (cc)

To execute your program, type “go” at the prompt. This runs the script generated by the Adapt compiler. The script loads and executes your program.

Calling Adapt Programs

Adapt programs are called from a C main program that runs on the SIB cell of the iWarp system. The main program must begin with a call to Initialize_Adapt, and it must end with a call to Terminate_Adapt. When you call an Adapt program from the main program, you must pass the height and width of the image as additional arguments to the Adapt procedure. The height and width arguments allow you to specify and change the size of the image at run time.

There are two ways to handle image variables passed to an Adapt procedure. The simplest way is to pass the address of the image, as with a typical procedure call. This method is used in the example that follows. In this example, the transfer of image data from the main program to the iWarp cells is handled by the Adapt compiler.

```
main()   {

   char a[HEIGHT] [WIDTH];
   char b[HEIGHT] [WIDTH];
   char c[HEIGHT] [WIDTH];

   Initialize_Adapt();

   /* read input image, etc. */

             •
             •
             •

   adapt_program1(a, b, HEIGHT, WIDTH);
   adapt_program2(b, c, HEIGHT, WIDTH);
```

```
    /* print results, etc. */

              •
              •
              •

    Terminate_Adapt();

}
```

You can get better performance and more image storage capacity by handling image allocation and transfer explicitly. The following example illustrates this method. The image allocation, distribution, and collection routines used in this example are described in the following section.

```
main()   {

    char img[HEIGHT] [WIDTH];
    int a_id, b_id, c_id;

    Initialize_Adapt();

    /* read input image, etc. */

              •
              •
              •

    a_id = ad_allocate_image(HEIGHT, WIDTH, sizeof(char));
    b_id = ad_allocate_image(HEIGHT, WIDTH, sizeof(char));
    c_id = ad_allocate_image(HEIGHT, WIDTH, sizeof(char));

    ad_distribute_image(img, HEIGHT, WIDTH, sizeof(char), a_id);

    adapt_program1(a_id, b_id, HEIGHT, WIDTH);
    adapt_program2(b_id, c_id, HEIGHT, WIDTH);

    ad_collect_image(img, c_id);

    /* print results, etc. */

              •
              •
              •

    Terminate_Adapt();

}
```

In this example, the ad_allocate_image function allocates space for images across the iWarp array. This function also returns an id number that is used to refer to the image. The ad_distribute_image function copies the image data from the main program to the previously allocated distributed space. In this example, when the Adapt procedures are called, the image id's are passed, instead of the image addresses as in the previous example. When adapt_program1 completes, its output image, b_id, is still distributed across the iWarp array. This allows adapt_program2 to be called immediately, without the need to distribute the image. When adapt_program2 completes, its output, c_id, is returned to the main program by ad_collect_image.

This example provides better performance because images are distributed and collected only when necessary. Since image id's are passed, the Adapt procedures know that image allocation and transfer has been handled explicitly by the programmer. If you do not use the image allocation, distribution, and collection routines, Adapt procedures will automatically allocate and distribute each input image and collect each output image.

Image Allocation, Distribution, and Collection Routines

You can use the following routines to explicitly allocate space and distribute and collect images.

ad_allocate_image

This routine allocates space for an image and returns an image id number. The id number refers to the space allocated by ad_allocate_image.

```
ad_image_id
ad_allocate_image(height, width, pixel_size)
int height, width;
int element_size;
```

height and *width* are the height and width of the image. *pixel_size* is the size of the pixel type in bytes.

ad_distribute_image

This routine copies the image at address *image_addr* to the distributed image space referred to by *image_id*.

```
void
ad_distribute_image(image_addr, height, width, pixel_size, image_id)
char*  image_addr;
int  height, width, pixel_size;
ad_image_id  image_id;
```

height and *width* are the height and width of the image stored at image_addr. They must be equal to the height and width of the image space referred to by *image_id*. *pixel_size* is the size of the pixel type in bytes.

ad_collect_image

This routine collects the distributed image *image_id*, and puts it at the address *image_addr* in the main program.

```
void
ad_collect_image(image_addr, image_id)
char*  image_addr;
ad_image_id  image_id;
```

Converting Apply Programs

You can convert Apply programs to Adapt programs with the app2ad filter. The app2ad filter accepts an Apply source file as input and produces an Adapt source file as output. The syntax for using app2ad is the following:

```
app2ad <apply_file> adapt_file
```

apply_file is the name of the Apply source file you wish to convert.

adapt_file is the name given to the converted source file.