nodes, each using the identical `incr` primitive to operate on different sized input vectors.

Much of the power of the state structure derives from the fact that the fields are arbitrarily defined by the user. For example, an n-point complex FFT can use the state structure to hold the size of the input vector and a collection of dynamically precomputed complex coefficients.

4.3 Input and Output Functions

If we ever plan to associate `incr` with a source or sink node, then we must write input and output functions for that node so that it can transfer data to or from the host computer. We typically place these functions in the same file as the primitive. For example, this input function is invoked each time an `incr` source node reads its input ports.

```
incr_in(inp, sp)
float *inp;
struct incr_state *sp;
{
    int i;

    for (i=0; i<sp->size; i++)
        scanf("%f", &(inp[i]));
}
```

4.4 Dynamic Initializers

As we learned in the previous section, some or all of the elements of the state structure can be initialized *statically* when a a node is created or *dynamically* at run time before the graph is executed. We will learn how to statically initialize state structures in the next section. To dynamically initialize a state structure, we provide a C function. For example, given the following state structure,

```
struct fft_state {
    int n;          /* number of complex points */
    int isign;      /* 1 is fft, -1 is ifft */
    complex wp[MAXN]; /* twiddle factors */
};
```

we might use a dynamic initializer of the following form to precompute the twiddle factors of an n-point complex FFT.

```
void fft_init(sp)
struct fft_state *sp;
{
    ...
}
```

5 Writing Graph Generators

Once a collection of node functions is assembled, the next step is to create a flowgraph using a graph generator. A graph generator is a C program that uses a library of ASSIGN functions to emit a flowgraph on the standard output.

For the initial implementation of ASSIGN we have chosen to provide a procedural interface rather than a visual interface. Each approach has its advantages and disadvantages. The chief advantage of the procedural approach is that we can exploit the tremendous expressive power of C, including iteration, conditionals, functions, and recursion. This expressive power is extremely helpful for generating large graphs with hundreds or thousands of nodes. Another key advantage is that command line arguments and manifest constants can be used to parameterize the graph generator, so that one graph generator can produce an entire family of graphs.

The main disadvantage of the procedural approach is that people tend to think of flowgraphs as graphical objects. We are investigating existing visual interfaces, and if we are able to find a system with enough expressive power to handle large parameterized graphs, it will not be difficult to replace the existing interface.

The library of graph generator functions provided by ASSIGN includes functions that create an empty graph, associate source and include files with a graph, add an atomic node to a graph, add a graph node to a graph, add input and output ports to a node, add a state structure to a node, add input and output functions to a node, add an edge to a graph, and emit a graph on the standard output. Complete descriptions of these functions, as well as numerous examples, can be found in [8].

6 Related Work

ASSIGN continues the tradition of building parallel program generators for restricted classes of applications. Examples of program generators for image processing, matrix computations, and nested loop computations can be found in [3, 15, 11], respectively.

The flowgraph model has been studied extensively. Some seminal theory on scheduling flowgraphs and an extensive survey of previous results can be found in [5].

Several systems for compiling flowgraph descriptions of signal processing applications have been developed in recent years, including Gabriel[4] at Berkeley and ZC[10] at Carnegie Mellon. Gabriel is a task–level parallel program generator that was originally designed to generate code on single processor systems or small shared memory parallel systems. However, there has been recent work to