Adlib version 1.0, Release Notes

D.B. Carpenter

March 1, 1996

1 Overview of the release

Version 1.0 of Adlib is implemented as a C++ class library built on top of MPI. Please report bugs and suggestions for improvement to `dbc@npac.syr.edu`.

Section 2 of this document describes how to use the `ad++` command (information which is also provided by the `ad++` 'manual page'). Section 3 describes the suite of example programs provided with the Adlib release.

2 Using `ad++`

An Adlib program typically has the form

```
#include "ad++.h"

// ...

void main(int argc, char* argv[]) {
  AdlibInit(argc, argv) ;

  // ...

  AdlibFinalize() ;
}
```

An `AdlibInit` call with precisely the interface illustrated should be made before *any* other Adlib call. In particular, initialisation must precede any call to a constructor from an Adlib class. In general therefore, Adlib objects may not be declared as static entities (of course, *pointers* to such objects can be declared as static entities).

Adlib programs are compiled using a script called `ad++`, which calls the local C++ compiler with options to search the correct directory for Adlib header files, and to link in the Adlib library code.

The result of compilation is an executable file. The procedure for running this file depends on the underlying message-passing platform.

The usage of the ad++ command is

```
ad++ [options] [files] [libraries]
```

where the options are options recognised by the local C++ compiler.

3 The example suite

The Adlib release contains a set of example codes in the `examples` subirectory.

In this section we assume the provision of a script `prun`. This runs an executable generated by the `ad++` command on a specified number of processes. An Adlib program suitable for execution on 3 processors might be compiled and executed by

```
ad++ test.C
prun 3 a.out
```

An example version of `prun` is provided in the Adlib `bin` directory, but you will have to tailor this script according to your local message-passing environment.

There is presently only one subdirectory of `examples`.

3.1 `tests` directory

Simple codes designed primarily to test the Adlib library. The codes

```
test0.h90
test1.h90
test2.h90
...
test6.h90
```

are best executed on 4 processes, e.g.:

```
ad++ -o test1 test1.C
prun 4 test1
```

The codes

```
test7.C
test8.C
test9.C
test10.C
```

```
test14.C
test17.C
testGather.C
testMultiShift.C
testScatterComb.C
testStep.C
```

are best executed on 6 processes, e.g.:

```
ad++ -o test7 test7.C
prun 6 test7
```

The codes

```
testSArray.C
testGprintf.C
testGscanf.C
```

do not involve distributed data, so any number of processors will do, e.g.:

```
ad++ -o testSArray testSArray.C
prun 3 testSArray
```

A script `runtests` will compile and execute all the programs in the directory (except `testGscanf.C`, which requires keyboard input).