F2C-ACC Users Guide

Version 2

Developed by: Mark Govett
National Oceanic and Atmospheric Administration (NOAA)
Earth System Research Laboratory (ESRL)

April 2010

Additional information and user support links

http://www.esrl.noaa.gov/gsd/ab/ac/Accelerators.html
help.accel.gsd@noaa.gov

I. Setup

1. Untar the gzipped file, and type "make" to build the compiler.
2. Install the m4 library (http://www.gnu.org/software/m4/).
   NOTE: Generated code relies on the m4 macro library to handle different operating system calling conventions between Fortran and C, collapse multi-dimensional arrays, and support some Fortran intrinsic functions.
3. A makefile is provided in the examples directory for building CUDA or C source code.
   a. Edit the Makefile as needed to define the location of your installed software. (GPU_HOME, CUDA, etc).
   b. Suffix runs are provided to generate .o (CPU objects), .cu (GPU objects), and .m4 targets.

II. Language Support

Common language constructs supported include most declarations, do-enddo, do-continue, if-else-endif, data, parameter, assignment statements, allocate, etc are supported.

At this time, language constructs not supported are the character, complex and derived types, all I/O statements, modules, while, where, forall, and select statements, and many Fortran intrinsic functions. In the event a language construct (eg. "interface" statement) is not supported, F2C-ACC will generate the following message that identifies the line in question:

```
F2C-ACC ERROR:  4,25 "Language construct not supported."
```

In this case, the error was on line 4, column 25. The message will be embedded directly in the output text so users can modify the generated C or CUDA code as necessary.

The following intrinsic functions are currently supported:

`abs, acos, asin, atan, ccos, cos, csin, max, min, pow, sign, sin, and tan.`

In the event a mapping between a Fortran intrinsic and C function is not available, or is not supported, a message will be generated that identifies the line in question:

ERROR" 5,25 "Fortran intrinsic not supported."

Where the line number and column number are given as 5 and 25 respectively in this example.

III. Parallelization Directives

A. Accelerator Region Directive

Defines a region for GPU acceleration. Generated code will copy all data with intent (IN) or (INOUT) to the GPU, and copy all data with intent (OUT) or (INOUT) back to the CPU after completion of the GPU kernels.

<u>Syntax:</u>

!ACC$REGION BEGIN (<thread sizes >, <block sizes >)
- thread sizes: number of threads in each dimension
- block sizes: number of blocks in each dimension

!ACC$REGION END

Examples:

```
!ACC$REGION ( <nz>, <nx, ny> ) BEGIN
```

- nz threads, nx by ny blocks

B. Accelerator Loop Parallelization

Defines loops level parallelism for the GPU device. Two types are currently supported: VECTOR and PARALLEL. The qualifier VECTOR is used for thread parallelism, and PARALLEL is used for block level parallelism. The directives are placed in the code immediately before the do-loop to be parallelized.

<u>Syntax:</u>

!ACC$DO VECTOR [(dim)]
!ACC$DO PARALLEL [(dim)]

Optional Arguments

dim: identifies the dimension of the thread or block parallelism

If no arguments are specified, the variable named in the do-loop extent will be compared to the variables listed in the ACC$REGION directive to determine the thread or block dimension. If the variable is not present in the directive, an error will be generated.

Examples:

`!ACC$DO PARALLEL (1)`

- Applies to the first dimension of the block, defined by ACC$REGION

`!ACC$DO PARALLEL`

- Dimension is determined by the loop extent of the do-loop (eg. *nx* in the loop: *`do I = 1, nx`*)

IV. Building Code

Code generation is available thru the F2C-ACC script. The following run-time options are supported:

--comment	Retain original array statements as comments
--FileType=[MODULE][INCLUDE]	File is a module or include file
--Fixed	Input files are f77 or f90 fixed format
--Free	Input files are fortran 90 free format
--Generate=[C][CUDA]	Language Options: CUDA and C, default is CUDA
--Kernel	File contains GPU kernel routines only
--OutputFile=[filename]	Name of the generated output file

Generated code relies on the GNU m4 library (http://www.gnu.org/software_m4/) to resolve array references, handle some intrinsic functions, and to resolve platform specific subroutine and function calling conventions.

The examples directory of this distribution contains a sample Fortran subroutine, a Makefile, and a README containing instructions for building source code.