ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439

ANL/MCS-TM-223

FLIC: A Translator for Same-Source Parallel Implementation of Regular Grid Applications

by

John Michalakes

Mathematics and Computer Science Division

Technical Memorandum No. 223

February 1997

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

Contents

FLIC: A Translator for Same-Source Parallel Implementation of Regular Grid Applications

by

John Michalakes

Abstract

FLIC, a Fortran loop and index converter, is a parser-based source translation tool that automates the conversion of program loops and array indices for distributed-memory parallel computers. This conversion is important in the implementation of gridded models on distributed memory because it allows for decomposition and shrinking of model data structures. FLIC does not provide the parallel services itself, but rather provides an automated and transparent mapping of the source code to calls or directives of the user's choice of run-time systems or parallel libraries. The amount of user-supplied input required by FLIC to direct the conversion is small enough to fit as command line arguments for the tool. The tool requires no additional statements, comments, or directives in the source code, thus avoiding the pervasiveness and intrusiveness imposed by directives-based preprocessors and parallelizing compilers. FLIC is lightweight and suitable for use as a precompiler and facilitates a *same-source* approach to operability on diverse computer architectures. FLIC is targeted to new or existing applications that employ regular gridded domains, such as weather models, that will be parallelized by data-domain decomposition.

Keywords: source translation, same-source strategy, regular grids, weather modeling, distributed-memory parallel computing, parallel tools.

1 Introduction

Adapting simulation codes to distributed-memory parallel computing is difficult and error prone, since it involves replacing the single-address space memory model with a distributed one. The task is especially demanding if one wishes to preserve source compatibility with the non-distributed memory version of the model. Such a same-source approach is desirable because it lessens the burden of maintaining the software. It is, however, is difficult to achieve because the modifications required for distributed memory are usually extensive.

One critical problem is the handling of loops that iterate over decomposed dimensions, for example, the north/south and east/west loops in an atmospheric model. The starting and ending indices must be adjusted for the fact that each processor has only a section of the original domain. Another is the loss of the identity relationship between global and local indices. A reference `U(i,j)` in the source model will be wrong on distributed memory because the i, jth element on the processors is not the i, jth element of the logical domain. Although the loss of global/local index identity can be avoided by retaining the original dimensions of the model's data structures, doing so defeats memory scaling. The model will not be able to take advantage of the larger aggregate memory available from more processors unless array dimensions are shrunken to allocate only the memory needed on each processor. Finally, boundary conditions specified implicitly as loop ranges do not hold, because the loop ranges are global whereas the loop variable is used as a local index.

All of these problems have been successfully addressed using run-time systems and parallel libraries. A number of available packages [2, 3, 5, 6] provide the necessary looping and index translation information to the program at run time. However, looping statements and index expressions must be modified from their original form to use this run-time information, introducing enough parallel artifact to hinder a same-source software approach.

FLIC, a Fortran loop and index converter, is a parser-based pre-compiler that bridges the gap between the non-parallel source code and the parallelized version by automatically generating the calls or directives needed by the run-time system. FLIC requires no directives of its own in the source code; the amount of input required to guide the translation is small enough to fit on the FLIC command line (within a UNIX Make file, for example) and general enough to be specifiable once for all the modules of the program.

FLIC requires only that the user enumerate the Fortran-defined constants that are used to specify the size of decomposed array dimensions in the program. If, for example, the state arrays for model variables such as temperature, wind velocity, and moisture are dimensioned IX for the north/south dimension and JX for the east/west dimension, it is sufficient to specify

```
flic -mdim=ix -ndim=jx file.F
```

From this specification, and by looking at the array references in each loop body, FLIC will identify and convert all the loops that iterate over the m and n dimensions. FLIC will also identify decomposed arrays that are being indexed by nonloop-variable expressions and generate the correct global-to-local index translation—all without touching the source file. Thus, the tool avoids the pervasiveness and intrusiveness imposed by directives-based preprocessors and parallelizing compilers, in which directives must be scattered throughout the source code.

Another critical problem for implementing single-address space codes on distributed memory architectures is interprocessor communication. FLIC does not analyze data dependencies and generate communication. However, this is well within the capability of source translators [1, 4] and is a future enhancement for the FLIC tool.

2 Overview

FLIC is a precompiler, as distinguished from a preprocessor. Unlike macro preprocessors—M4, CPP, or Perl—which rely on pattern matching, FLIC is built atop UNIX lex and yacc and contains the full lexical, syntactic, and semantic specification for Fortran77 with extensions (such as Cray POINTER statements). The translator works from an abstract syntax tree and is fully capable of disambiguating Fortran nuances (e.g., when is an assignment statement actually a statement function?), and it has full access to type, class, and dimension information for all identifiers in the program.

Preprocessors do play a role in the translation process. FLIC is more powerful than a preprocessor, but it is also more difficult to modify. Therefore, FLIC does not generate a full Fortran expansion of the distributed-memory target code. Doing so would unnecessarily limit the range of possible target expansions. Rather, it generates FLIC macros that are expanded by using M4, CPP, or another preprocessor. Users who do not wish to write expansions for the macros FLIC generates can take advantage of the ready-made sets of macro expansions included in the FLIC distribution. Users who develop new macro expansions and wish to have them included are invited to contact the author.

2.1 FLIC Input and Output

In addition to a file containing Fortran source code, FLIC requires a small amount of additional information from the user. The primary information required is a list of the defined constants that are used to declare the decomposed dimensions of model arrays. Additionally, FLIC also needs to know whether the routine being translated is called from within a loop over a decomposed dimension.

The principal advantage of this approach is that a great deal of relevant knowledge about the program can be inferred from a very small amount of information, namely, the set of defined constants that are used to declare decomposed array dimensions. Even in a large model, the number of these constants is comparatively quite small. For example, in the 50,000-line PSU/NCAR MM5 code, about a dozen defined constants, easily listed on the FLIC command line, are used to declare the north/south and east/west dimensions of the hundreds of arrays in the model.

2.1.1 Loop Statements

To properly translate loop statements, FLIC needs enough information to be able to tell whether a given loop in the routine is over one of the decomposed dimensions. If it is, it should be translated; otherwise it is left alone. The current version of FLIC allows up to two dimensions of a domain to be decomposed; these are referred to as M and N. In an atmospheric model, for example, the north/south and east/west dimensions are candidates for decomposition over processors; one of these would be designated M and the other N. FLIC uses this rule:

> *A loop is over a decomposed dimension if, within the body of the loop, a decomposed dimension of an array is indexed by an expression that depends on the value of the loop variable. The loop is converted provided the ending loop range is* not *an expression of the defined constant that declared the array dimension.*

Since FLIC has ready access to the loop variable from the source code and can statically infer whether index expressions are dependent on it, the only missing piece of information is which dimensions of the array being indexed are over decomposed dimensions. Providing FLIC with the list of defined constants that are used to declare the M and N decomposed dimensions of model arrays allows it to look for these among the array declarations in the routine and to categorize each array accordingly. For example,

```
REAL UA(IX,JX,KX)
...
DO 10 J = 2, JL-1
```

```
        DO 10 K = 1, KL
          DO 10 I = 2, IL-1
            UA(I,J,K) = ...
 10   CONTINUE
      ...
```

is a fragment containing a triply nested loop that contains a reference to the three-dimensional model array UA. The tool has no way of inferring from the code itself that the I loop and the J loop are decomposed over M and N, respectively. If, however, we inform the tool that IX is used to declare the M dimension and JX is used to declare the N dimension, the tool can infer everything else from the rule stated above, generating

```
      REAL UA(IX,JX,KX)
      ...
      FLIC_DO_N(J,2,JL-1)
        DO K = 1, KL
          FLIC_DO_M(I,2,IL-1)
            UA(I,J,K) = ...
 10         CONTINUE
          FLIC_ENDDO
        ENDDO
      FLIC_ENDDO
      ...
```

The beginnings and ends of the loops over I and J are replaced with FLIC macros; the K loop is left alone. In the case of the I loop, an index expression dependent on the loop variable was found as the first index of UA in the loop body. The first dimension of UA is known to be over M because it is defined with IX. Therefore, the I loop is inferred to be over M, a decomposed dimension. Since the ending loop range is not an expression of IX (which would imply that the loop is already specified over the local and not global dimension), the loop is translated (likewise, with the J loop). Within the body of the converted loop, the loop variable is considered to be local and hence may be used as-is to index local arrays. If the loop variable appears in a logical expression, it may be converted to global.

Note that the tool also unravels the common termination on a labeled CONTINUE statement. That statement is left intact, within the innermost loop body, on the chance that the loop body contains a GOTO statement to that label.

2.1.2 Global and Local Indices

In addition to identifying and translating loops over decomposed dimensions, FLIC must identify and correctly handle instances where global indices must be converted to local indices, and vice versa.

Global values appearing as indices to local arrays must be converted to the correct local value, for example,

```
REAL UA(IX,JX,KX)
...
DO J = 1, JX
  DO K = 1, KX
    UA(3,J,K) = ...
  END DO
END DO
```

This construct would be found in sections of code that set boundary values. The value used as the first index of UA is wrong because "3" refers to the position in the undecomposed domain, whereas that index in local memory refers to another cell entirely. FLIC is able to recognize this situation using the following rule:

> *When a decomposed array dimension is indexed by an expression that is* not *dependent on the value of a loop variable, that expression will be considered* global *and in need of conversion.*

In the example above, the first dimension of UA is a decomposed dimension, and the constant "3" is clearly not dependent on either of the loop variables, J or K. Therefore, the expression "3" must be treated as global and transformed accordingly.

A simple transformation would be to replace the index in-place. However, for efficiency, FLIC defines and sets temporary variables outside the loop, if possible, to avoid recalculating the constant index expression each time through the K loop:

```
REAL UA(IX,JX,KX)
...
IFLIC_0 = FLIC_G2L_M(3,IX)
DO J = 1, JX
  DO K = 1, KX
    UA(IFLIC_0,J,K) = ...
  END DO
END DO
```

With regard to the actual generation of local indices from global indices, this is up to the particular set of macro expansions. The ready-made set of macros that maps to the RSL parallel library will expand FLIC_G2L_M(3,IX) to

```
min(max(3+idif,1),IX)
```

IDIF is a variable whose value is set by RSL at run time to the difference between a global index and a local index on each processor. Adding IDIF to "3" gives the local index. Of course, this is the local index only if the processor actually has the subrange of the M dimension containing the third element; hence, the min/max calls. If the processor has this element from the global dimension, the local index will fall between 1 and IX, the size of the local dimension. Otherwise, the resulting value will be either 1 or IX, and these can be treated as garbage values in the decomposed data structure. Again, this is an example of one possible expansion for the G2L macro; certainly other techniques are possible, depending on the needs of the underlying parallel software. The important point, however, is that the appearance of the source code is unaffected.

2.1.3 Boundary Tests

Boundary tests present the inverse problem: local indices will appear in statements where they should be global. In the case where the boundary conditions are implicit in a loop range, the conversion is handled as part of the overall loop conversion, since the starting and ending ranges are included as arguments to the FLIC_DO_M and FLIC_DO_N macros that replace the loop statements.

However, expressions that depend on loop variables often may be found in the logical expressions of conditional statements to test for position in the global domain before executing (or not executing) a set of statements:

```
      REAL UA(IX,JX,KX)
      ...
      DO 100 J = 1, JX
        DO 100 K = 1, KX
          IF ( J .EQ. 1 ) THEN
            UA(I,J,K) = ...
            ...
          ENDIF
 100  CONTINUE
```

FLIC will generate the following transformation, using the rule:

> *An instance a loop-variable over a decomposed dimension occurring in a logical expression will be converted from local to global.*

FLIC will translate the above code to

```
      REAL UA(IX,JX,KX)
      ...
      FLIC_DO_N(J,1,JX)
        DO K = 1, KX
          IF ( FLIC_L2G_N(J) .EQ. 1 ) THEN
            UA(I,J,K) = ...
            ...
          ENDIF
 100      CONTINUE
        END DO
      FLIC_ENDDO
```

A possible expansion for FLIC_L2G_N(J), again from the RSL/FLIC mapping, is (J-JDIF).

2.2 Iterative Scope

FLIC automatically handles routines that contain iteration over both the M and N dimensions using the rules described in the preceding sections. Sometimes, however, a routine may be called from within a loop; in other words, at least one of the loop statements that sets indices within the routine is outside the definition of the routine. For example, a routine containing I loops may be called from within a J loop of another routine and have the J index passed in as an argument. Such a routine is *slab callable*, since a full sweep of the domain involves calling the routine once for each J-indexed I slab. Although only the I loops are visible to the translator, the J index must also be handled, even though the J loop itself is inaccessible. Whole program analysis, such as is performed by the APR tools, is the ultimate solution to this problem. However, with some additional guidance from the command line, FLIC can handle this situation.

The `-m=`*list* and `-n=`*list* options provide a way of specifying an externally defined loop and loop variable on the command line. For example, for the routine described in the previous paragraph, the external J loop is be specified as

```
flic -n=j file.f
```

This indicates that the routine in file.f is called from within a loop over the decomposed dimension N and that the loop variable is J inside the routine. Specifications may be given for files containing more than one routine using entries of the form *routine:variable*, for example,

```
flic -n=sub1:j,sub2:j file.f
```

In this case, only the routines SUB1 and SUB2 would be considered slab callable. Other routines in the file would be unaffected by the option.

2.3 FLIC Macros

This section describes the macros that FLIC generates in the target code:

FLIC_DO_M(*var, start, end*)	Replaces DO loops over the M decomposed dimension. *Var* is the loop variable; *start* and *end* are the starting and ending global indices.
FLIC_DO_N(*var, start, end*)	Replaces DO loops over the N decomposed dimension.
FLIC_ENDDO	Replaces ENDDO or labeled CONTINUE statement at the end of a converted loop.
FLIC_G2L_M(*index-expr*)	Converts *index-expr* from global to local for M indices.
FLIC_G2L_N(*index-expr*)	Converts *index-expr* from global to local for N indices.
FLIC_L2G_M(*index-expr*)	Converts *index-expr* from local to global for M indices.
FLIC_L2G_N(*index-expr*)	Converts *index-expr* from local to global for N indices.
define(INSIDE_MLOOP)	This M4 definition is generated by FLIC for routines that are callable within a loop over M. It is a flag to the preprocessor that the FLIC_DO_N macros may need to be expanded differently.
define(INSIDE_NLOOP)	This M4 definition is generated by FLIC for routines that are callable within a loop over N. It is a flag to the preprocessor that the FLIC_DO_M macros may need to be expanded differently.

These macros are then expanded to the appropriate distributed-memory target code by using a macro preprocessor such as CPP or M4 (preferred because it allows multiline expansions). FLIC is distributed with ready-made macro packages. This section provides the specification for the FLIC macro interface, which may be used to construct new macros. Users who develop new macro expansions and wish to have them included are invited to contact the author.

3 Usage

This section describes how FLIC can be integrated and used in the compilation mechanism for a model. Input to the tool includes the source code and specifications that will be used to direct the transformation. These specifications may come from the command line, a file, or the user environment.

3.1 Obtaining FLIC

FLIC is available on the World Wide Web at

```
http://www.mcs.anl.gov/Projects/FLIC
```

This page also contains installation information and the latest usage information for the tool.

3.2 Command Line Arguments

The following is a command line summary for the tool.

-extend	Allow extended source lines (to 132 characters).
-m=*list*	
-n=*list*	Specify externally set loop indices for slab-callable routines called from within an M or N loop. *List* is a comma separated list of [*routine:*]*loop-var* items. *Loop-var* is the name of the externally set loop variable as it is known within the routine. *Routine* is the name of the routine the specification applies to. If *routine* is not given, the specification applies to all routines in the source file.
-mdim=*list*	
-ndim=*list*	Specify a comma-separated list of identifiers that are used to specify the M or N decomposed dimension when arrays are declared.
-F=*file*	Specify a file containing FLIC directive information. Specifications in the file will supersede command line settings.
-H=*string*	Include *string*, indented to column 7, at the beginning of the declarations section of the module.
-CPP=*string*	Include *string*, prepended with a # character, at the beginning of the declarations section of the module.
-toolname=*string*	Use a different string than FLIC in the macro names.

3.3 Directives File

FLIC may read the transformation specifications from a file whose name is specified using the -F option on the command line. The specifications in the file will supersede those from the command line or the environment. FLIC directives begin with C in the first column so that they may be treated as comments, should the user opt to include these in the Fortran source. In this case, the -F option must specify the source file itself; otherwise the directives will be ignored. The syntax for the directives is as follows:

cflic m=*list*	
cflic n=*list*	Same effect as the -m and -n command line options.
cflic mdim=*list*	
cflic ndim=*list*	Same effect as the -mdim and -ndim command line options.

3.4 Environment

The transformation specifications may be specified in the environment by using shell environment variables. Settings from the environment are superseded by command line settings and by input from a directives file.

FLIC_M *string*	
FLIC_N *string*	Same effect as -m and -n command line options.
FLIC_MDIM *string*	
FLIC_NDIM *string*	Same effect as the -mdim and -ndim command line options.

3.5 Compiling with FLIC

FLIC is designed to be used during compilation of a model, typically from within a UNIX Make file. The following fragment is an example rule for make that compiles model source files *name*.F down to relocatable object form *name*.f.

```
OBJ=solver.o ...

all : $(OBJ)
        $(LINK) -o model $(OBJ) ...

.F.o:
        cpp $(INCLUDES) -C -P $*.F | flic -F=FLICFILE > $*.flic
        m4 FLIC2RSL.m4 $*.flic | cpp -C -P >$*.f
        $(FC) -c $(OPTIM) $(DEBUG) $(F77_FLAGS) $*.f
```

In this example, the source file solver.F contains CPP directives that are independent of the parallelization (#include directives and conditional compilation directives, for example). The first line passes solver.F through the C preprocessor, cpp; the output is piped into FLIC. [1] The piped-in code that FLIC sees will be entirely Fortran (no CPP directives) and completely expanded. FLIC recognizes and expands normal Fortran INCLUDE statements, so it is not necessary to use CPP as a preprocessor to FLIC if the only concern is expansion of included files. FLIC transforms the source file, using directives in the file FLICFILE; output is directed to a temporary file, solver.flic. This file is then input to the M4 macro preprocessor, along with a set of macros targeting RSL, in the file FLIC2RSL.m4. This particular set of macro expansions generates a small number of additional directives that are expanded by piping M4 output to a second call to cpp. Output is directed to a file solver.f, which is then input to the Fortran compiler.

4 Conclusion

FLIC transforms loops and array indices in a Fortran source file for implementation on distributed-memory parallel computers. FLIC analyzes the code, identifying loops over decomposed dimensions and local and global index expressions, and generates a small set of

[1] Note the use of the -C and -P options to cpp. The -C option makes the preprocessor ignore and pass through things that might look like C-style comments. This prevents occurrences of the string "/*" in a Fortran FORMAT statement from voiding the remainder of the file. The -P option suppresses the generation of line control information in the cpp output. These are highly recommended whenever cpp is used to preprocess Fortran source.

in-line macros that are then easily expanded to target the user's distributed-memory library or run-time system of choice. A number of macro-expansions are provided with the FLIC distribution. The tool is lightweight enough to be included in a Make file, so that only the compiler need ever see the transformed code. The tool does not address data dependency or interprocessor communication, but still provides considerable savings in programmer time and effort, since loop and index translations are by far the most extensive and error-prone modifications required to convert a code to distributed memory. Providing access to these program loops and index expressions through macros as a precompiling step also facilitates other translations such as cache blocking.

Because it is built using a fully enabled Fortran parser, FLIC is able to perform the necessary transformations using unaltered Fortran source files and a very small amount of input data from the command line or a small file, as opposed to tools that rely on pattern matching and that require directives scattered throughout the code. This capability eliminates the need to modify the source code, and enables a same-source approach to development and maintenance of codes for multiple computer architectures.

References

[1] R. Friedman, J. Levesque, and G. Wagenbreth, *Fortran Parallelization Handbook*, Applied Parallel Research Inc., Sacramento, California, June 1995.

[2] R. Hempel and H. Ritzdorf, *The GMD communications library for grid-oriented problems*, Tech. Rep. GMD-0589, German National Research Center for Information Technology, 1991.

[3] S. R. Kohn and S. B. Baden, *A Parallel Software Infrastructure for Structured Adaptive Mesh Methods*, in Proceedings of Supercomputing '95, IEEE Computer Society Press, 1996.

[4] S. Kothari, *Parallelization Agent for Legacy Codes*, draft technical report, Iowa State University, Ames, Iowa, 1996.

[5] J. Michalakes, *RSL: A Parallel Runtime System Library for Regular Grid Finite Difference Models Using Multiple Nests*, Tech. Rep. ANL/MCS-TM-197, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, December 1994.

[6] B. Rodriguez, L. Hart, and T. Henderson, *A Library for the Portable Parallelization of Operational Weather Forecast Models*, in Coming of Age: Proceedings of the Sixth ECMWF Workshop on the Use of Parallel Procesors in Meteorology, World Scientific, River Edge, NJ, 1995, pp. 148–161.