Using the Force to Write Parallel HEP Programs

by
Harry F. Jordan

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

9

9

Introduction

The basic idea behind using the force method of parallel programming is to write a program (a force) which can be executed by any number of processes, including one process. The fork into parallel processes is done before the force is ever executed and the number of processes can be chosen on the basis of hardware efficiency only. Experience indicates that it is not hard to write programs which are independent of the number of executing processes on a shared-memory MIMD computer. Several such applications programs have been written for the HEP computer in HEP Fortran 77, which is standard Fortran 77 with very rudimentary parallel extensions. The constructs which seemed to be useful in these programs were formalized into a set of statements which can be macro-expanded into HEP Fortran 77.

Some essential ideas to keep in mind when writing a force is that each statement is executed by all processes of the force unless an explicit synchronization prevents this. The distinction between a local variable, which has a separate memory cell assigned for each process referencing it, and a global variable, which has one memory cell which is used by all processes, is important to keep straight. There are force constructs which make it easy to specify that a section of code is to be executed by only one process. Other constructs specify that the body of a loop is to be executed for all values of the loop index in parallel with one process handling each index value. Unless explicit force statements prevent it, a call is executed by all processes so that subroutines executed by all processes of the force are also possible.

Description of the Force Macros

The user of the force macros writes a single parallel main program, one or more parallel force subroutines (optional) and one or more single stream subroutines to be executed by a single process (also optional). In writing the parallel main program and parallel subroutines the macros described below may be used. The single stream subroutines and all of the code except the macros in the parallel modules is HEP Fortran and familiarity with that language is assumed. Asynchronous variables would normally be global or common variables but otherwise are used as in HEP Fortran. The CREATE statement, however, is inconsistent with the idea of specifying parallelism at the highest level of program

structure and should not appear in a force. The macros are divided into two classes: declaration macros and executable macros.

Declaration Macros

Force <module> of <nproc> on <npem> ident <me>

- This must be the first statement of the single parallel main program. It will set up the parallel environment and give access to the execution time parameters:

<nproc> - the number of processes in the force

<npem> - the number of PEMs used by the force

<me> - a unique identifier for each process

The statement also establishes <module>G as the name of the global common block.

Join

- This should be the last executable statement in the parallel main program. It is listed with declarations because it is paired with the Force statement. It causes all processes to quit except one which returns to the driver. It would often immediately precede the END statement.

Global <v1>, <v2>, ... , <vn>

- This statement declares <v1>, <v2>, ..., <vn> to be global variables so that the same storage cell(s)

is(are) used by all processes of the force. This is done by declaring the variable(s) to be in a labeled common block <module>G built from the module name of the parallel main program or of the parallel subroutine containing the Global statement. The Global macro will only handle simple variable names for <vi>. Any type declarations or dimensions must be specified in separate statements.

Local <declaration>

- The syntax of this statement differs from that of Global in that a valid Fortran declaration must follow the keyword. Since the default declaration in parallel Fortran is local, meaning that each process has a separate variable of the same name, the keyword is merely stripped off the declaration by the macro.

End declarations

- This macro call terminates the declarations section of the parallel Force main program and begins its executable code. It marks the place to insert declarations generated automatically by the macros and generates some executable code to set up the process identifier and record the logical task number.

Forcesub <module>(p1,...,pn) of <nproc> on <npe> ident <me>

- This is the first statement of a parallel subroutine and is completely analogous to the Force statement except for the parameter list following the subroutine name. Since the unique identifier of a process can only be passed through a parameter in a call, the macro automatically appends ,<me> to the parameter list to generate:

SUBROUTINE <module>(p1,...,pn,<me>)

Declarations, Local and Global statements can come between a Forcesub statement and the following End header. The current macro does not allow for a null argument list. There is no special statement to terminate a parallel subroutine. The Fortran RETURN statement should be the last executable statement.

End header

- This is used in connection with the previous statement to bracket the declarations section of a

parallel subroutine. It marks the point to insert automatically generated declarations for the subroutine.

Executable Macros

Forcecall <module>(p1,...,pn)

- This generates a call to a parallel subroutine. It differs from a CALL only in that the process identifier <me> is appended to the parameter list.

Barrier

- This statement should be executed by all processes of the force. When all have reached the Barrier statement a single process will execute statements between it and the following End barrier statement. After the singly executed code section is complete, all processes will resume execution after the End barrier.

End barrier

- Paired with the previous statement, this one delimits a section of code executed by a single instruction stream. Synchronized parallel execution begins after this statement.

Pcase on <var>

- A parallel case construct consists of a series of independent sections of code, each of which is to be executed by a single process. The sections are delimited by a Pcase, zero or more Usect and an End pcase statements. Execution of multiple cases is pre-scheduled using the local variable <var> which will have the value i during the execution of the ith case.

If there are more processes in the force than there are independent code sections then all will be executed simultaneously. Otherwise some of the sections will be executed sequentially by the same process. A parallel case with only one code section is similar to a Barrier in that the code is executed by a single process but no synchronization of other processes occurs.

Usect

- This statement separates multiple single stream code sections of a parallel case. It is not used if there is only one section.

End pcase

- The parallel case construct is terminated by this statement. Note that some processes may proceed past this point while portions of the parallel case are still being executed.

Critical \$<var>

- Mutual exclusion is accomplished by critical sections, begun by a Critical statement and ended by End critical. Two critical sections on the same global variable, \$<var>, cannot execute simultaneously. The implementation constructs a local variable <var>L to be used as a temporary.

End critical

- This statement is paired with the nearest unmatched preceding Critical statement to delimit a critical section. Nested critical sections are allowed.

Presched DO <n> <i>=<i1>,<i2>,<i3>

- Parallel loops introduced by this statement must have bodies such that instances of the body for different values of <i> are independent and can thus be executed in parallel. Pre-scheduling partitions different values of <i> evenly over processes in a manner fixed at compile time. The step size <i3> is optional and is taken as one if missing. Pre-scheduled loops are used when the execution time of the loop body is independent of <i>.

<n> End presched DO

- This statement terminates the body of a pre-scheduled DO loop. The statement number <n> must match that on the Presched DO statement.

Selfsched DO <n> <i>=<i1>,<i2>,<i3>

- This statement is an alternative for introducing a parallel loop. Its behavior is the same as that of a Presched DO except that scheduling of the work is done

at execution time. A process takes the next unassigned value of <i> as soon as it is free. This tends to even the workload over processes when the execution time of the loop can vary significantly for different <i> values. The implementation generates an asynchronous global variable with name \$<i>G.

<n> End selfsched DO

- This statement ends the body of the self-scheduled DO loop with statement number <n>.

Examples of Macro Expansion

Several examples of the use of the macros in Fortran 77 and the expansion of the source code using the macros into HEP Fortran 77 running under the HEPOS system are given.

The source text:

```
Force PAR of NPROC on NPEM ident ME
  DIMENSION A(50),B(5,5)
  INTEGER COUNT
Global A,B,COUNT,$X,$Y
Local INTEGER I,J,K
End declarations
```

Parallel main program

```
Join
END
```

becomes:

```
C Force PAR of NPROC on NPEM ident ME
  SUBROUTINE PMAIN( LTASKP )
C
  COMMON /PARCOM/ NPEM, NPROC
  COMMON /PARCOM/ $IC, $FINISH, $NUMPROC
  COMMON /PARCOM/ $NPEMB, $BLOCK(4)
  COMMON /PARCOM/ NPR(4), $NPRB(4)
  COMMON /PARCOM/ PSWS(50,4), LTASK(200)
  INTEGER NPEM, NPROC
  LOGICAL $FINISH, $BLOCK
  LOGICAL UNIQUE
  INTEGER ME
C
  DIMENSION A(50),B(5,5)
  INTEGER COUNT
C Global A,B,COUNT,$X,$Y
  COMMON /PARG/ A,B,COUNT,$X,$Y
C Local INTEGER I,J,K
  INTEGER I,J,K
C End declarations
  ME = $NUMPROC + 1
  $NUMPROC = ME
  LTASK(ME) = LTASKP
C
```

Parallel main program

```
C Join
  ME = $IC - 1
  IF ( ME .EQ. 0 ) THEN
```

```

    $FINISH = .TRUE.
  ELSE
    $IC = ME
  ENDIF
  RETURN
END

```

9

9

Subroutine definition and call

Definition - The source text:

```

Forcesub PARSUB(Q,MEAN) of NPROC on NPEM ident ME
  REAL Q,MEAN
  Global $Z
  Local REAL ZZ
  End header

```

Parallel subroutine

```

RETURN
END

```

becomes:

```

C    Forcesub PARSUB(Q,MEAN) of NPROC on NPEM ident ME
C    SUBROUTINE PARSUB(Q,MEAN,ME)
C
C      COMMON /PARCOM/  NPEM, NPROC
C      COMMON /PARCOM/  $IC, $FINISH, $NUMPROC
C      COMMON /PARCOM/  $NPEMB, $BLOCK(4)
C      COMMON /PARCOM/  NPR(4), $NPRB(4)
C      COMMON /PARCOM/  PSWS(50,4), LTASK(200)
C      INTEGER NPEM, NPROC
C      LOGICAL $FINISH, $BLOCK
C      LOGICAL UNIQUE
C      INTEGER ME
C
C      REAL Q,MEAN
C      Global $Z
C      COMMON /PARSUBG/ $Z
C      Local REAL ZZ
C      REAL ZZ
C      End header

```

Parallel subroutine

```
RETURN
END
```

Call - The source text:

```
Forcecall PSUB(P1,P2,P3)
```

becomes:

```
C      Forcecall PSUB(P1,P2,P3)
      CALL PSUB(P1,P2,P3,ME)
```

9

9

Parallel case construct

The source text:

```
Pcase on IC
  V1 = .5
  U1 = V1*V1
Usect
  V2 = 8.2
Usect
  V3 = 22.1
  U3 = V3/7.
  W3 = U3 + V3**2
End pcase
```

becomes:

```
C      Pcase on IC
      IC = ME
      IF (IC .EQ. 1) THEN
        V1 = .5
        U1 = V1*V1
C      Usect
      IC = IC + NPROC
      ENDIF
      IF (IC .EQ. 2) THEN
        V2 = 8.2
```

```
C      Usect
      IC = IC + NPROC
      ENDIF
      IF (IC .EQ. 3) THEN
      V3 = 22.1
      U3 = V3/7.
      W3 = U3 + V3**2
C      End pcase
      IC = IC + NPROC
      ENDIF
```

9

9

Barriers and critical sections

The barrier source text:

```
Barrier
X = 7.5
End barrier
```

becomes:

```
C      Barrier
      CALL BAR(UNIQUE, ME)
      IF (UNIQUE) THEN
      X = 7.5
C      End barrier
      CALL LETGO(UNIQUE, ME)
      ENDIF
```

The critical section source text:

```
Critical $LOCK
STACK(I) = T
I = I + 1
End critical
```

becomes:

```
C      Critical $LOCK
      $LOCK = LOCKL
      STACK(I) = T
      I = I + 1
C      End critical
      LOCKL = $LOCK
```

9

9

Running a Force Program on HEP

The macro expansion of a force program into HEP Fortran is done in a Unix environment using the stream editor, sed, and the m4 macro processor. A one line shell script, hep-force, pipelines the output of sed, using the script, forscript, into m4 with the macro file, forcemac, included. The result is that a pattern matching macro expansion turns

a Fortran 77 program written with force constructs into a HEP Fortran subroutine with standard name PMAIN, perhaps along with other subroutines.

The output of the macro expansion is then compiled and linked in the HEPOS environment to form a complete executable file. Linking requires four other files; PARDRV.HFO, TSKDRV.HFO, GETSPI.HAO and BAR1.HAO corresponding to source files; PARDRV.HFS, TSKDRV.HFS, GETSPI.HAS and BAR1.HAS respectively. PARDRV is a single stream main program in logical task 1 which creates a copy of TSKDRV in each logical task (each PEM). No purpose is served by having other than one logical task per PEM. The TSKDRV processes create multiple processes evenly spread over PEMs and executing copies of the PMAIN(LTASK) subroutine produced by the Force macro. The number of PEMs and number of processes are specified as input to the driver at run time but must be consistent with specifications in the linker command file.

The number of logical tasks linked must be the same as the number of PEMs requested by the run control cards and equal to (or greater than) the number of tasks input to the driver. The number of copies of PMAIN and any other parallel subroutines linked in each task must be greater than or equal to the number of processes used for execution divided by the number of PEMs. The current driver is tailored to measuring performance with differing numbers of processes and requires four integer input values: the number of tasks, the number of processes to use to execute the program the first time, the maximum number of processes to use and the step size in number of processes. The parallel program will then be executed with each specified number of processes and System Performance Indicator statistics will be printed by the driver for each execution. To execute only once without modifying the driver the starting and maximum numbers of processes can be set equal. Example job files for linking and running a two PEM job are shown below.

9

9

```
ASN 6,PR:
ASN 5,*
TASK TASK1
LIBRARY [001001]RTL1.HAO
INCLUDE PARDRV.HFO
INCLUDE TSKDRV.HFO
INCLUDE GETSPI.HAO
COPIES 30
  PMAIN
  SGEFA
  INCLUDE SGETST.HFO
  INCLUDE BAR1.HAO
  INCLUDE [001001]ITCALL.HAO
TASK TASK2
LIBRARY [001001]RTL1.HAO
INCLUDE TSKDRV.HFO
INCLUDE GETSPI.HAO
COPIES 30
  PMAIN
  SGEFA
  INCLUDE SGETST.HFO
  INCLUDE BAR1.HAO
  INCLUDE [001001]ITCALL.HAO
END
/*
ASN 7,SGETST.HLL,A,PR,EXWD,REC:128,NEW,KEEP
LUN 3
RUN [001001]LINKER
//
```

Link Command File for a Two PEM Job

```
JOB SGETSTR 30000
TASK 1 4000,7F2,300,NP30,PEM0
TASK 2 4000,7F2,300,NP30,PEM1
ASN 5,*
  2  NUMBER OF TASKS (PEMS)
  10  STARTING NUMBER OF PROCESSES
  60  MAXIMUM NUMBER OF PROCESSES
  10  STEP SIZE FOR NUMBER OF PROCESSES
      problem dependent input data
/*
ASN 6,PR:
RUN SGETST.HLL
//
```

Run Command File for a Two PEM Job

APPENDIX

A complete example of a force program follows. It involves the LU decomposition subroutine SGEFA from LINPACK.

```
Force LUD of NPROC on NPEM ident ME
Global NEQN, IPVT, ITM1, ITM2
INTEGER NEQN, IPVT(500), ITM1, ITM2
Global INFO, IONCE
INTEGER INFO, IONCE
Global A, TIME, RATE
REAL A(500,500), TIME, RATE
Local INTEGER I, J
End declarations

C
  IF (ME .EQ. 1) THEN
    IF (IONCE .EQ. 0) THEN
      700 READ(5,700) NEQN
        FORMAT(I4)
        IONCE = 1
    ENDIF
  ENDIF

C
  Barrier
  End barrier

C
  Presched DO 200 I = 1, NEQN
    DO 100 J = 1, NEQN
      IF (I .NE. J) THEN
        A(I,J) = I + J
      ELSE
        A(I,I) = I*I + 1
      ENDIF
    100 CONTINUE
    200 End Presched DO

C
  Barrier
  End barrier

C
  IF (ME .EQ. 1) CALL CLOCK(ITM1)

C
  Forcecall SGEFA(A,500,NEQN,IPVT,INFO)

C
  Barrier
  End barrier

C
```



```

IF (ME .EQ. 1) THEN
  CALL CLOCK(ITM2)
  TIME = 1.0E-7*(ITM2 - ITM1)
  RATE = (2.0*NEQN*NEQN*NEQN/3.0)*1.0E-6/TIME
  WRITE (6, 400) NPROC, NEQN, TIME, RATE

```

```

400      FORMAT(// ' USING', I4, ' PROCESSES, AN ORDER', I5,
+          'MATRIX IS DECOMPOSED IN', F15.6, ' SECONDS FOR A RATE OF',
+          F10.3, ' MELOPS.')
      ENDIF
      Join
      END

```

```

Forcesub SGEFA(A, LDA, N, IPV, INFO) of NPROC on NPEM ident ME
INTEGER LDA, N, INFO, IPV(LDA)
REAL A(LDA, 1)

```

C

```

      Local REAL ABSMAX, T
      Local INTEGER KK, I, MAXI, J, KP1
      Global IALL
      INTEGER IALL
      Global $ALLMAX, PIV
      REAL $ALLMAX, PIV
      End Header

```

C

```

      Barrier
      INFO = 0
      PURGE $ALLMAX
      $ALLMAX = 0.0
      End Barrier
      DO 2000 KK = 1, N-1
      ABSMAX = 0.0
      Presched DO 100 I = KK, N
      T = ABS( A(I, KK) )
      IF (ABSMAX .LT. T) THEN
      MAXI = I
      ABSMAX = T
      ENDIF

```

100

```

      End Presched DO
      T = $ALLMAX
      IF (T .LT. ABSMAX) THEN
      IALL = MAXI
      T = ABSMAX
      ENDIF

```

```

C      $ALLMAX = T
      Barrier
      IF ($ALLMAX .EQ. 0.0) THEN
        INFO = KK
        IPVTT(KK) = KK
      ELSE
        IPVTT(KK) = IALL
      ENDIF
      End Barrier
C
      IF (INFO .EQ. KK) GOTO 1000
      MAXI = IALL
      IF (MAXI .NE. KK) THEN
        Presched DO 110 J = KK, N

          T = A(MAXI, J)
          A(MAXI, J) = A(KK, J)
          A(KK, J) = T
110      End Presched DO
      ENDIF
C
      KP1 = KK + 1
      Barrier
      PIV = -1.0/A(KK, KK)
      End Barrier
C
      Selfsched DO 130 I = KP1, N
        T = PIV*A(I, KK)
        A(I, KK) = T
        DO 120 J = KP1, N
          A(I, J) = A(I, J) + T*A(KK, J)
120      CONTINUE
130      End Selfsched DO
C
      Barrier
      $ALLMAX = 0.0
      End Barrier
C
1000      CONTINUE
C
2000      CONTINUE
      RETURN
      END
```