2004 춘계학술발표회 논문집
한국원자력학회

중대사고해석 코드(MIDAS)개발을 위한 CF 팩키지 개편

A Restructuring of CF Package for MIDAS Computer Code

박선희, 김고려, 김동하, 조성원
한국원자력연구소
대전광역시 유성구 덕진동 150

요 약

MELCOR 계산에 사용자 지정 기능을 적용토록 해 주는 CF 팩키지에 대하여 사용자 편의성 위주의 데이터 전달구조를 지닌 중대사고 해석 종합전산코드인 MIDAS의 팩키지로 추가하기 위하여 개편하였다. 이를 위해 기존 MELCOR 코드의 데이터 전달 방법을 수정하여 CF 팩키지에 적용하였다. 기존의 MELCOR 코드는 FORTRAN77을 사용하여 구현하였으며, 이로 인해 메모리의 낭비가 있고 내용을 파악하기 어려운 단점이 있으며, CF 팩키지의 특성상 data 자체가 다른 팩키지 데이타의 위치정보이기 때문에 어려움이 더하다. FORTRAN90이 발표된 후 저장장소의 동적 할당과 사용자가 정의한 데이터 타입을 사용할 수 있게 되었으며, 이 기능을 적용하여 메모리를 효과적으로 사용하고 코드의 내부를 쉽게 이해할 수 있도록 하였다. 본 논문에서 언급된 CF 팩키지에 대한 개편은 모듈 구성과 부프로그램 변환을 수행하여 계산 수행 부분인 MELCOR 부분은 물론 data file을 작성하는 MELGEN 부분까지를 대상으로 하였다. 검증을 위하여 개편 이후의 계산 결과와 기존의 결과를 비교하였으며, 결과의 추이가 같음을 확인하였다. 검증 결과 코드 전체로 확장하여 사용할 수 있는 타당성을 제시하였고, 코드의 어느 부분에서나 변수 파악을 용이하게 하고 기존 모델의 개선이나 추후 개발되는 새로운 모델의 접목을 편리하게 하여 코드의 국산화 작업을 가속시킬 수 있을 것으로 예상된다.

Abstract

CF package, which evaluates user-specified ‘control functions’ and applies them to define or control various aspects of computation, has been restructured for the MIDAS computer code. MIDAS is being developed as an integrated severe accident analysis code with a user-friendly graphical user interface and modernized data structure. To do this, data transferring methods of current MELCOR code are modified and adopted into the CF package. The data structure of the current MELCOR code using FORTRAN77 causes a difficult grasping of meaning of the variables as well as waste of memory, difficulty is more over because its data is location information of other package’s data due to characteristics of CF package. New features of FORTRAN90 make it possible to allocate the storage dynamically and to use the user-defined data type, which lead to an efficient memory treatment and an easy understanding of the code. Restructuring of the CF package addressed in this paper includes module development, subroutine modification, and treats MELGEN, which generates data file, as well as MELCOR, which is processing a calculation. The verification has been done by comparing the results of the modified code with those from the existing code. As the trends are similar to each other, it hints that the same approach could be extended to the entire code package. It is expected that code restructuring will accelerate the code domestication thanks to direct understanding of each variable and easy implementation of modified or newly developed models.

1. 서론

국내 국산화된 중대사고 해석코드 개발의 필요성이 대두되면서 원자력연구소에서는 기존의 중대사고 해석 코드인 MELCOR를 근간으로 기존 모델에 대해 개선하거나 추가하여 중대사고해석 코드인 MIDAS를 개발하고 있다.

기존 MELCOR 코드에서의 데이터 저장 방법은 데이터 별로 고정적인 크기의 저장장소를 확보하고, 그 한도내에서 가변적인 크기의 데이타를 처리한다. 효율적인 데이터 전달 및 저장을 위하여 포인터 변수를 사용하고 있으며 CF 팩키지가 저장하고 있는 데이터는 그 자체가 다른 팩키지의 위치정보이다. 이로 인해 코드내부에 대한 전반적인 이해가 어려우며, 특히 다른 모델을 추가할 경우 연계 작업이 복잡하여 많은 어려움이 따르고 있다.

하지만 FORTRAN90이 발표된 후 저장장소의 동적할당이 가능하게 되어 필요한 만큼의 저장장소 확보를 코드의 수행시에 할 수 있게 되어 메모리의 낭비를 없앨 수 있게 되었고, 사용자가 정의한 data type을 사용할 수 있게 되어 데이터 분류를 내용 위주로 할 수 있게 되었으며, 이러한 장점들이 코드 개발에도 많이 활용되고 있다.[1,2,3]

이러한 추세에 따라 기존 코드의 데이터 저장 및 전달 방법을 재구성하여 코드의 이해도(readability)를 높이고 동적 메모리 관리 (DMM) 기법을 사용하여 메모리의 낭비를 없앴으며, 코드 개발자나 사용자가 코드의 내부를 쉽게 이해할 수 있도록 하였다. 또한 구성된 모듈을 적용하여 subroutine 개편시 도출변수(derived type variables)로의 변경을 자동으로 수행하는 프로그램을 개발하여 사용하였다[4,5].

코드의 전체적인 개편에 앞서 간단한 팩키지에 대한 개편 시안을 이미 개발하여 타당성을 분석하였으며[6,7,8,9], 간단한 팩키지부터 전체 코드로 확장하는 전략으로 수행하고 있다. 개편 대상 팩키지는 여러 팩키지 중 CF 팩키지(사용자로 하여금 특정 기능을 위한 function을 정의하도록 하여, 계산 실행시에 적용할 수 있도록 하는 팩키지)로 선정하여 코드 구조를 개편하였으며, MELCOR 부분은 물론 계산 수행을 위한 restart file을 작성하는 MELGEN 부분까지를 대상으로 하였고, 개편 이전의 결과와 이후의 결과를 비교하여 검증작업을 수행하였다.

2. 기존 구조

MELCOR code는 3부분으로 되어 있으며, 입력자료 점검과 계산에 필요한 restart file 작성을 하는 MELGEN, 작성된 restart file을 입력으로 한 계산 수행, log file 및 plot file의 작성을 하는 MELCOR, 그리고 PLOT과 관련된 수행부분이 있다.

모든 수행부분에는 물리적인 현상에 관련된 COR, HT, SPR, TF, CF, RN2 등 20개의 팩키지가 있으며, 각 팩키지간에 data를 공유하여 수행되며, 각 팩키지마다 수십개의 subroutine들이 있고, message를 기록하는 subroutine, code 수행 흐름을 제어하는 subroutine 등 전체적으로 수백개의 subroutine으로 구성되어 있다[10].

2.1 Restart file

코드 개편을 위하여 먼저 data의 저장 및 전달 방법을 파악하였다. MELCOR 수행시 RESTART file의 read 또는 write를 수행하는 subroutine은 MXXRS과 MXXRSW이며, 조건에 따라 read 또는 write를 수행한다. 이 과정은 각 팩키지별로 순차적으로 수행되고, 각 팩키지별 real, integer, logical, character의 4개 데이터 타입별 array에 대한 읽기와 쓰기를 수행한다.

2.2 Database

MELCOR 코드는 모든 팩키지의 data를 효율적으로 전달 및 저장하기 위해 4가지 타입의 저장 장소를 예약하여 사용하고 있다[8,9,11]. 또한 각 데이터 타입의 고정된 크기 내에서 사용자의 입력에 따라 저장 용량의 크기를 최적화하여 데이터를 저장하며, database 이동은 2개의 단계를 거치게 된다[8,9,11]. Database 이동의 1단계에서는 데이터 타입별 array의 갯수와 해당

팩키지에서 사용하는 array의 시작지점을 argument로 전달한다. Database 이동의 2단계에서는 데이타 타입별 팩키지내에서 사용되는 변수에 대한 array내의 포인터를 argument로 전달한다. Database 이동의 작업단계에서는 argument로 전달된 변수들을 지역변수로 사용하고 있다.

2.3 포인터 변수

포인터 변수를 분석하기 위하여 database의 이동에 관련된 subroutine인 xyzDBC subroutine과 xyzDBD subroutines에 있는 comment 및 argument pass를 분석하였다. CF 팩키지에 대한 단계별 포인터 변수는 다음과 같다.

(1) CF 팩키지의 1단계 포인터 변수

1단계의 포인터 변수는 모든 팩키지에 대해 동일하게 적용되는데(그림 1), 몇 개의 subroutine 내에 ‘CFDB’라는 common block으로 정의되어 있으며(그림 2), 4개의 data type별로 2개씩의 변수를 사용한다

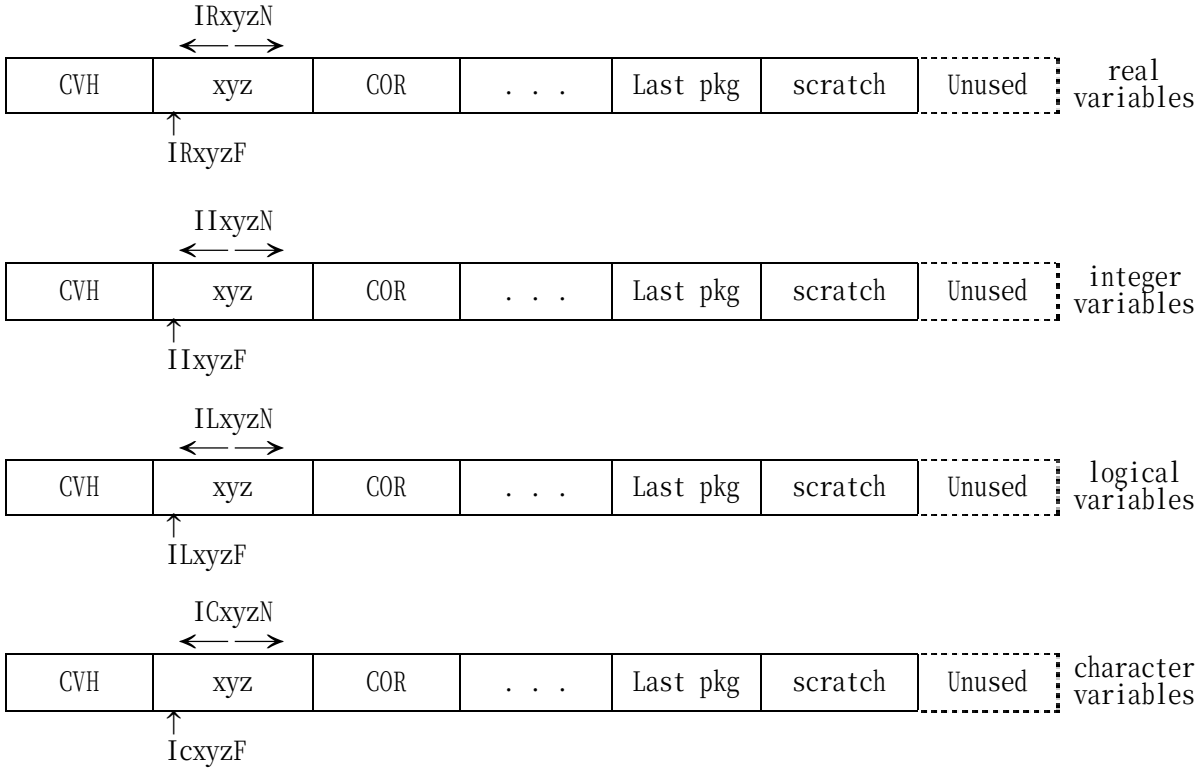

그림 1. 1단계의 포인터 변수의 내용

```
*- INCLUDE CFDB
C
C     POINTERS TO CONTROL FUNCTION DATA BLOCKS IN DATABASE COMMONS
C
      COMMON /CFDB/ IRCFF , IRCFN , IICFF , IICFN ,
     1              ILCFF , ILCFN , ICCFF , ICCFN
C
*-
```

그림 2. CF 팩키지의 1단계 포인터 변수 정의

(2) CF 팩키지의 2단계 포인터 변수

2단계의 포인터 변수는 팩키지에 따라 다르게 적용되는데, 몇개의 subroutine 내에 ‘xyzPNT’ 또는 이와 비슷한 이름의 common block으로 정의되어 있으며, 4개의 data type에 대한 database 중 어느 하나를 가리키도록 정의된다. 전체의 팩키지중 CF 팩키지에 대한 포인터 변수 정의는 그림 3과 같다.

CF 팩키지에 대한 포인터 변수가 의미하는 내용은 팩키지내의 subroutine에서 찾아볼 수 있으며, 이를 근거로 CF 팩키지에 대한 모듈을 구성하였다. 또한 개편대상의 CF data는 CF 팩키지 이외에 여러 팩키지에서도 참조하고 있음을 확인하였다.

이것으로 CF 팩키지를 개편하였을 때 기존의 MELCOR와 접목하여 검증이 가능함을 확인하였다.

```
*- INCLUDE CFPNT
C
C     POINTER COMMON BLOCK FOR CONTROL FUNCTION PACKAGE
C     VERSION 1.8.0 MODIFIED  2/17/89
C
      COMMON /CFPNT/ KLCFVO, KLCFVN, KRCFVO, KRCFVN, KCFSCL, KCFADC,
     1       KCFLBD, KCFUBD, KCFNMS, KCFNUM, KCFNAM, KCFTYP, KNMARG,
     2       KPTARG, KNMMIS, KPTMIS, KPTFUN, KNMMIN, KCFMSO, KCFMSN,
     3       KARSCL, KARADC, KCHARG, KNARGA, KNARGB, KNARGC,
     4       KARVAL, KLARVL, KCFHSH,
     5       KLINIT, KLATST, KSHOTO, KSHOTN, KCHGFL, KCHMSG,
     6       NUMCF,  NNCF  , NUMLCF, NNLCF , NUMRCF, NNRCF , NUMARG,
     7       NAXARG, NNMIS , NMBMIS, NDMHSH, IOFLCF, IOFRCF, IOFMIS
C
*-
```

그림 3. CF 팩키지의 2단계의 포인터 변수 정의

3. 개발 내용

코드의 전체적인 개편에 앞서 간단한 팩키지에 대해 먼저 개편을 하여 타당성을 분석하고 일부 팩키지에 대하여 검증 작업을 거친 후 전체 코드로 확장하는 방법으로 진행하였다. 또한 restart file을 작성하는 MELGEN 부분을 기존 방법대로의 읽기/쓰기를 그대로 유지하도록 개편하였고, restart file 작성 이후의 계산 수행 부분인 MELCOR 부분도 개편대상으로 하였다.

MELCOR 1.8.4를 대상으로 하였으며, 사용 언어는 FORTRAN90으로 하였다. 데이터 전달 구조는 코드 개발자나 사용자로 하여금 코드 내부의 이해가 쉽고 사용이 용이하도록 데이터 저장 및 전달 체계를 기존 MELCOR의 데이터 타입별 단일 배열에서 FORTRAN90 언어의 특성을 활용할 수 있는 도출변수 형태(derived data type)로 바꾸어 팩키지별로 모듈화하였다. 데이터 전달구조 개편의 전체적인 특성은 그림 4와 같다.

개편 팩키지는 사용자가 정의하여 기능을 적용할 수 있는 CF 팩키지 중 RN1 팩키지에서 사용된 CF 팩키지로 하였으며, RN1 팩키지내의 CF 팩키지에서 사용하는 변수를 모듈화된 구조로 전환하고, 동적 저장 기법을 이용하여 계산 효율을 높였다. MELCOR 전체의 수행을 제어하는 subroutine, CF 팩키지에 속해 있는 60여개의 subroutine 및 CF 데이터를 사용하는 RN1 팩키지의 subroutine 130여개를 분석하였으며, 개발된 자동 변환 프로그램 (MELtoMID)의 사용으로 수작업을 최소화하여 관련 subroutine을 변환하였다[4,5].

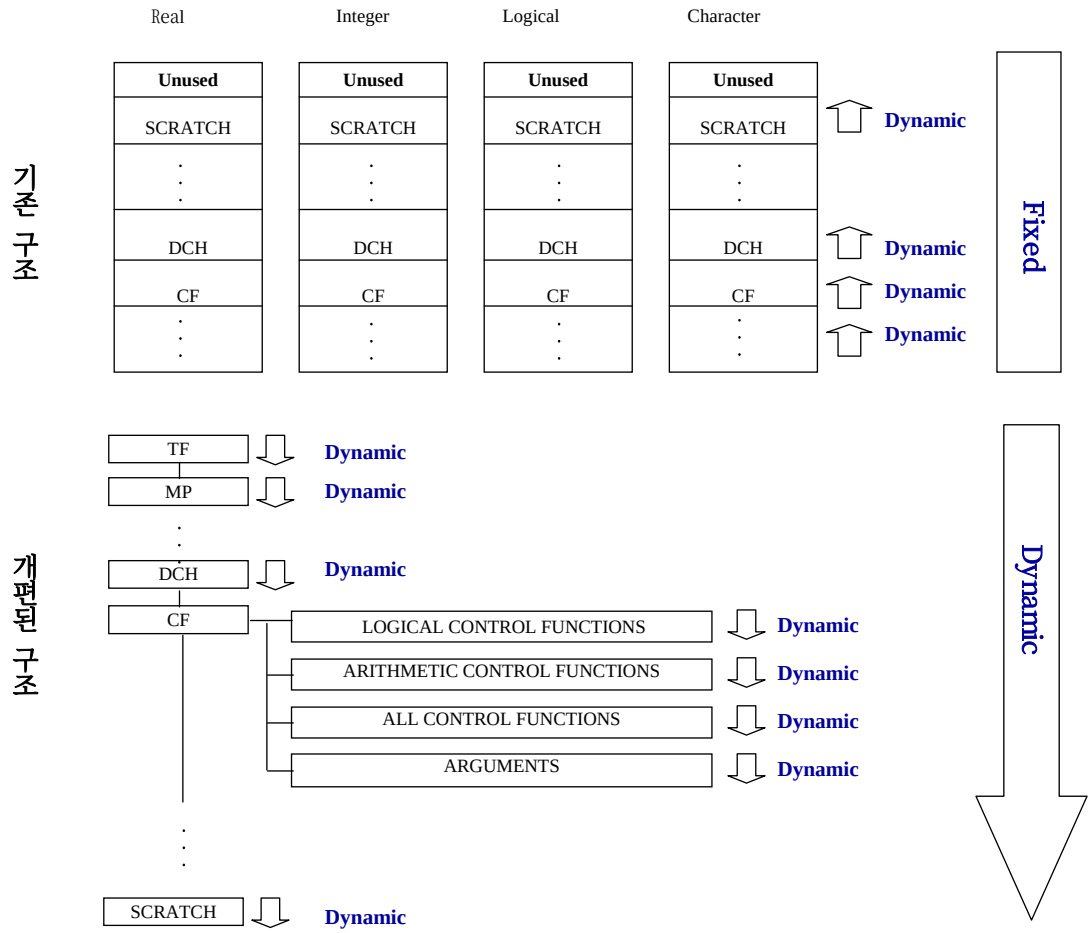

그림 4. 데이터 전달구조 개편의 전체적인 특성

3.1 모듈 구성

Subroutine CFDB1, CFDB2, CFDB3, CFDBC, CFDBE, CFDBI 등에서 subroutine CFDB4, CFDBD, CFDBF, CFDBZ, CFDBY, CFDBJ 등을 호출하면서 포인터 변수를 사용하였으며, subroutine CFDB5, CFCDBC, CFDB5, CFP1A, CFLDBC, CFRDBC, CFTDBC, CFRUN 등에서는 포인터변수에 대응하는 지역변수 (local variable)를 사용하고 있다. 2단계의 포인터 변수 분석에 근거하여 구성된 CF 팩키지의 모듈은 그림 5와 같다.

3.2 Subroutine 개편

Subroutine의 개편에 앞서 다른 팩키지의 개편과 같이[8,9] 먼저 FORTRAN77로 쓰여진 기존 MELCOR 코드 전체를 FORTRAN90 언어로 변환하였으며, 전후의 결과가 같음을 입증하였다[11].

RN1 팩키지와 관련된 CF 팩키지의 subroutine의 경우, argument가 전달되는 모든 단계를 확인한 결과, 포인터 변수를 직접 사용하는 subroutine은 28개였으며, 다른 subroutine은 포인터 변수를 직접 사용하지 않고 subroutine 내의 local variable을 사용하여 argument를 전달하거나 계산하고 있었다. 변경된 내용 중 특이한 것은 dimension으로 선언된 array를 member variable 형태로 변경하는 것이다. 이에 대한 내용을 표 1에 나타냈다.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
    MODULE CF_MDL
!
    INTEGER :: NCF_FLAG,   NCF_NARG
!
        TYPE CF_A1 ; SEQUENCE
           INTEGER    :: CFN1,     CFN2,     CFN3
!                                 Argument 1 - 3
           CHARACTER  :: CFPNAM*4, CFGNAM*4, CFMNAM*8
!                                 Member variable
        END TYPE
        TYPE (CF_A1), ALLOCATABLE :: CF_ARG1(:)
!
    END MODULE CF_MDL
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

그림 5. 구성된 CF 팩키지의 모듈

표 1. Array의 변경내용

기존 방식에서의 사용 :

```
        ARVALU (N) = XREALX (INARGB (N) + IJK * INARGC (N) )
```

신규 방식에서의 사용 :

```
        IF ( INARGB(N) .NE. 0 ) THEN
              ARVALU (N) = XREALX (INARGB (N) + IJK * INARGC (N) )
        ELSE
           ARVALU(N) = GETCF(CF_ARG1(MMM-1+N)%CFN1, CF_ARG1(MMM-1+N)%CFN2, &
                       CF_ARG1(MMM-1+N)%CFN3, IJK, CF_ARG1(MMM-1+N)%CFPNAM,&
                       CF_ARG1(MMM-1+N)%CFGNAM,CF_ARG1(MMM-1+N)%CFMNAM )
        ENDIF
```

개편 전후의 subroutine 비교는 다음과 같다. CF를 사용하는 Subroutine RN1DB5에서는 argument에서 포인터 변수가 삭제된 것과(그림 6), CF를 사용하는 subroutine CFDBV에서는 지역변수대신 직접 변수가 사용되었음을 알 수 있다(그림 7). 추가된 function GETCF는 그림 8에 나타내었다.

<> CF 팩키지를 사용하는 subroutine RN1DB5의 개편 이전

```
      SUBROUTINE RN1DB5(MUMARG,CHARG,
     +                  IDSUR,
     +                  ICVNUM,KCVFLG,NVOL,NNVOL,
     +                  NHSNUM, IBVL, IBVR, NHS, NNHS,
     +                  INARGA,INARGB,INARGC,IGOODG)
C
       . . . . . .

C          'RN1-AMG-'
C
C       CRACK THE CHARACTER STRING TO GET THE SECTION, CLASS AND CV
```

```
C
        CALL CRACHR(CHARG(IX), CHDEL, 2, 1, N)
         . . . . . .
C
        INARGA(IX)=0
C
C       *** SET POINTER TO AEROSOL SECTION IN GAS PHASE   ***
C
C-------TOTAL AEROSOL
        IF(INTKRA(5).EQ.1)IRORT = IT1AGO
C-------RADIOACTIVE AEROSOL
        IF(INTKRA(5).EQ.2)IRORT = IR1AGO
C
        INARGB(IX) =   IRRN1F - 1
     +               + IRORT - 1
     +               + NUMSEC*NUMCLS*(IVOL-1)
     +               + NUMSEC*(INTKRA(4)-1)
     +               + INTKRA(3)
C
        INARGC(IX) = IORRN1
C
        GO TO 100
C
305     CONTINUE
C
        . . . . . .

      RETURN
      END
```

<> CF 팩키지를 사용하는 subroutine RN1DB5의 개편 이후

```
!     SUBROUTIN RN1DB5 (MUMARG, CHARG,      ICVNUM, KCVFLG,NVOL,NNVOL,&
        NHSNUM, IBVL, IBVR, NHS, NNHS,INARGA,PNAME,INARGB,INARGC,IGOODG)
!
       USE RN1_MDL
       USE CF_MDL
!
       . . . . . .

!          'RN1-AMG-'
!
!       CRACK THE CHARACTER STRING TO GET THE SECTION, CLASS AND CV
!
        CALL CRACHR (CHARG (IX), CHDEL, 2, 1, N)
!
        . . . . . .

!
        INARGA (IX) = 0
!
!       *** SET POINTER TO AEROSOL SECTION IN GAS PHASE   ***
!
!! NOTICE: Modified by manually (2003. 8. 6. S.H. Park)
!!-------TOTAL AEROSOL
!        IF (INTKRA (5) .EQ.1) IRORT = IT1AGO
!!-------RADIOACTIVE AEROSOL
!        IF (INTKRA (5) .EQ.2) IRORT = IR1AGO
!
```

```
!       INARGB (IX) = IRRN1F - 1 + IRORT - 1 + NUMSEC * NUMCLS *      &
!                     (IVOL-1) + NUMSEC * (INTKRA(4)-1) + INTKRA(3)
        INARGB (IX) = 0
!       INARGC (IX) = IORRN1
        INARGC (IX) = 0
        IF ( PNAME .EQ. 'CF' ) THEN
!                       PNAME is 'CF'
           CF_ARG1(IX)%CFN1 = IVOL
           CF_ARG1(IX)%CFN2 = INTKRA(4)
           CF_ARG1(IX)%CFN3 = INTKRA(3)
           CF_ARG1(IX)%CFPNAM = 'RN1 '
           CF_ARG1(IX)%CFGNAM = 'CV1 '
           IF (INTKRA (5) .EQ.1) CF_ARG1(IX)%CFMNAM = 'AER1G   '
           IF (INTKRA (5) .EQ.2) CF_ARG1(IX)%CFMNAM = 'RDA1G   '
        ELSE
!                       PNAME is 'EDF'
           EDF_NCH1(IX)%EDFN1 = IVOL
           EDF_NCH1(IX)%EDFN2 = INTKRA(4)
           EDF_NCH1(IX)%EDFN3 = INTKRA(3)
           EDF_NCH1(IX)%EDFPNAM = 'RN1 '
           EDF_NCH1(IX)%EDFGNAM = 'CV1 '
           IF (INTKRA (5) .EQ.1) EDF_NCH1(IX)%EDFMNAM = 'AER1G   '
           IF (INTKRA (5) .EQ.2) EDF_NCH1(IX)%EDFMNAM = 'RDA1G   '
        ENDIF
!-                                      end S.H. Park
!
        GOTO 100
!
  305   CONTINUE
!
      RETURN
      END SUBROUTINE RN1DB5
```

그림 6. Subroutine RN1DB5의 개편 전후 비교

<> subroutine CFDBV의 개편 이전

```
      SUBROUTINE CFDBV (
C     INPUT
     1  MUXARG , NNARG , CHARG ,
     2  INARGA , INARGB, INARGC , IJK,
C      OUTPUT
     4  ARVALU , LARVAL , NNN  , IGOOD )
C
       . . . . . .

      DO 1000 N = 1 , NNARG
        IF ( INARGA(N) .EQ. 0 ) THEN
C         REAL VALUED IN BLANK COMMON
          ARVALU(N) = XREALX(INARGB(N)+IJK*INARGC(N))
        ELSE IF ( INARGA(N) .EQ. 1000 ) THEN
C         LOGICAL VALUED IN DBLOGC
          LARVAL(N) = LOGICA(INARGB(N)+IJK*INARGC(N))
          NNN = NNN + 2**(N-1)
        ELSE
       . . . . . .

      ENDIF
```

```
 1000 CONTINUE
      RETURN
      END
```

<> subroutine CFDBV의 개편 이후

```
!! NOTICE: Modified by manually (2003.11.11. S.H. Park)
!                              : last argument MMM inserted
      SUBROUTINE CFDBV (MUXARG, NNARG, CHARG, INARGA, INARGB, INARGC,   &
      IJK, ARVALU, LARVAL, NNN, IGOOD, MMM)
!-                                        end S.H. Park
!! NOTICE: Modified by manually (2003. 8. 7. S.H. Park) : Insert CF_MDL
      USE CF_MDL
!-                                        end S.H. Park
!
       . . . . . .
!
       DO 1000 N = 1, NNARG
        IF (INARGA (N) .EQ.0) THEN
!! NOTICE: Modified by manually (2003. 8. 7. S.H. Park)
!         REAL VALUED IN BLANK COMMON
!         ARVALU (N) = XREALX (INARGB (N) + IJK * INARGC (N) )
          IF ( INARGB(N) .NE. 0 ) THEN
          ARVALU (N) = XREALX (INARGB (N) + IJK * INARGC (N) )
          ELSE
       ARVALU(N) = GETCF(CF_ARG1(MMM-1+N)%CFN1, CF_ARG1(MMM-1+N)%CFN2, &
                   CF_ARG1(MMM-1+N)%CFN3, IJK, CF_ARG1(MMM-1+N)%CFPNAM,&
                   CF_ARG1(MMM-1+N)%CFGNAM,CF_ARG1(MMM-1+N)%CFMNAM )
          ENDIF
!-                                        end S.H. Park
        ELSEIF (INARGA (N) .EQ.1000) THEN
!         LOGICAL VALUED IN DBLOGC
          LARVAL (N) = LOGICA (INARGB (N) + IJK * INARGC (N) )
          NNN = NNN + 2** (N - 1)
        ELSE
       . . . . . .

      ENDIF
 1000 END DO
      RETURN
      END SUBROUTINE CFDBV
```

그림 7. Subroutine CFDBV 의 개편 전후 비교

<> 추가된 function GETCF

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
     REAL FUNCTION GETCF (NUM1, NUM2, NUM3, IJK, PNAM, GNAM, MNAM)
!
       . . . . . .
!
       USE RN1_MDL
       USE CF_MDL
       . . . . . .
```

```
      SELECT CASE ( GNAM )
          CASE ( 'CV0 ' )
              SELECT CASE ( MNAM )
              CASE ( 'XMRLS   ' )
                 GETCF = RN1_CV0(NUM3,NUM2)%XMRLS0(NUM1)
              CASE ( 'VAP1G   ' )
                 GETCF = RN1_CV0(NUM2,NUM1)%VAP1G0
              CASE ( 'RDV1G   ' )
                 GETCF = RN1_CV0(NUM2,NUM1)%RDV1G0
              CASE ( 'AER1L   ' )
                 GETCF = RN1_CV0(NUM2,NUM1)%AER1L0
              CASE ( 'RDA1L   ' )
                 GETCF = RN1_CV0(NUM2,NUM1)%RDA1L0
              CASE ( 'VAP1L   ' )
                 GETCF = RN1_CV0(NUM2,NUM1)%VAP1L0
              CASE ( 'RDV1L   ' )
                 GETCF = RN1_CV0(NUM2,NUM1)%RDV1L0
              END SELECT
!
          CASE ( 'CV1 ' )
              SELECT CASE ( MNAM )
              CASE ( 'AER1G   ' )
                 GETCF = RN1_CV1(NUM3,NUM2,NUM1)%AER1G0
              CASE ( 'RDA1G   ' )
                 GETCF = RN1_CV1(NUM3,NUM2,NUM1)%RDA1G0
              END SELECT
!
          CASE ( 'NC1 ' )
              SELECT CASE ( MNAM )
              CASE ( 'ADEP    ' )
                 GETCF = RN1_NC1(NUM2,NUM1)%ADEP0
              CASE ( 'RADEP   ' )
                 GETCF = RN1_NC1(NUM2,NUM1)%RADEP0
              CASE ( 'VDEP    ' )
                 GETCF = RN1_NC1(NUM2,NUM1)%VDEP0
              CASE ( 'RVDEP   ' )
                 GETCF = RN1_NC1(NUM2,NUM1)%RVDEP0
              END SELECT
!
          CASE ( 'VL0 ' )
              SELECT CASE ( MNAM )
              CASE ( 'VCND    ' )
                 GETCF = RN1_VL0(NUM1)%VCND0
              END SELECT
          END SELECT
!
        . . . . . .

     END FUNCTION
```

그림 8. 추가된 function GETCF

개편대상의 CF data는 CF 팩키지 이외에 RN1을 비롯한 대부분의 팩키지에서 사용되고 있으며, 그 중 RN1 팩키지의 subroutine들을 분석하여 개편하였다.

4. 개발결과 및 검증

포인터 변수의 사용을 없애고 FORTRAN90의 특징인 동적 저장기법을 사용하여야 하므로, CF 팩키지의 개편에 앞서 FORTRAN77로 쓰여진 기존 MELCOR 코드 전체를 FORTRAN90 언어로 변환하였으며, 이와 관련된 library와 실행파일도 새로이 생성하여 비교한 결과도 같음을 확인하였다.

또한 FORTRAN90 언어로 변환한 MELCOR 수행 결과와 개편된 MELCOR의 수행 결과를 CF 팩키지와 관련된 주요 변수를 비교하였다. 결과를 비교하기 위하여 총출력 4300MW_t의 PWR type 원자로에 대한 가상사고 SBO를 모사하였으며, 그외 몇 개의 경우에 대해서도 15,000초까지 수행되도록 모사하였다.

작업을 실행한 결과와 관련하여, FORTRAN90 언어로 변환한 MELCOR 수행 결과와 개편된 CFMELCOR의 수행 결과를 CF 팩키지와 관련된 주요 변수에 대해 비교하였다. MELGEN 및 MELCOR의 여러 subroutine에 대한 개편이 제대로 되었는지의 점검을 위해 data file(restart file)에의 read/write 부분에 대해 확인하였다. MELGEN 실행 이후 restart file 에의 write가 수행되고, MELCOR 실행의 초기 부분에서 data file(restart file)의 read 직후, CF 팩키지내의 각 변수 값에 대해 개편 이전과 개편 이후 같은지를 확인하였다. Unformatted print문을 추가한 subroutine을 작성하여 CF 값이 결정되는 subroutine RN1DB5와 CF 값을 사용한 subroutine CFDBV에서 CF의 모든 변수값들을 확인한 결과, 개편 이전과 개편 이후의 값이 같음을 확인하였다. 이때의 방법은 text file에의 write 및 UNIX diff command 를 사용한 text file의 비교이다. 이로써 적어도 data file(restart file)에의 read/write 부분에 대한 개편이 제대로 되었음을 확인하였다.

구체적인 비교는 그림 9 ~ 그림 10에 나타내었다. 그림 9는 모든 핵종에서 발생되는 총붕괴열이며 6000초부터 미세한 차이를 보이기 시작했다. 그림 10은 코드에서 사용하는 미분방정식의 적분시간간격(delta time)을 나타낸 것으로 약 2300초 부근과 약 8000초부근 및 10000초 이후부터 차이를 보이는 것으로 보아 그 시점에서 급격한 현상변화가 예측되며, 정확한 값은 다르지만 경향은 근사하게 나타남을 알 수 있다.

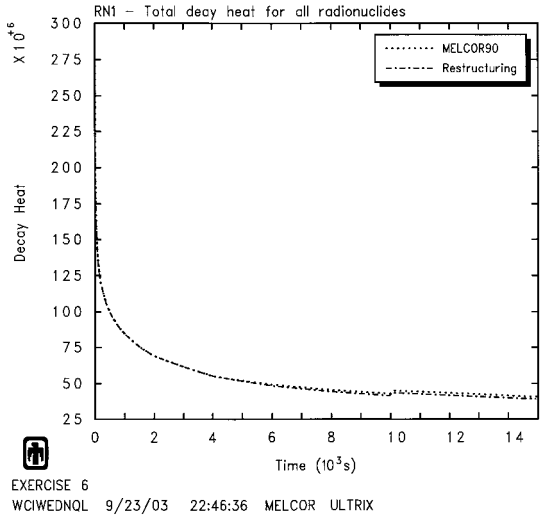

그림 9. 모든 핵종에서 발생되는 총붕괴열

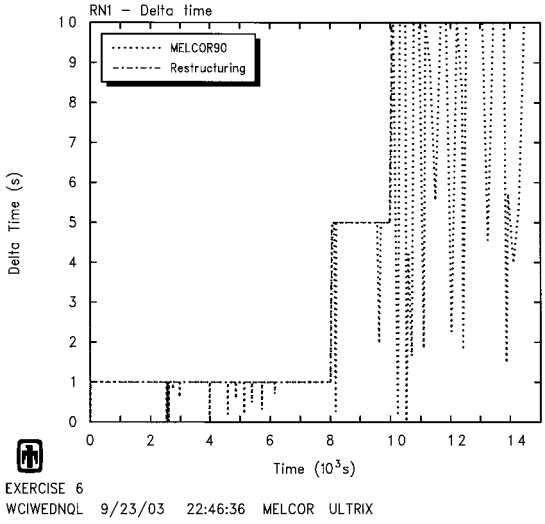

그림 10. 프로그램 적분 시간 간격(dt)

비교 결과로 보아, CF 팩키지의 변수값은 일치하였으며, 다른 팩키지의 변수값은 graph 의 추이는 비슷하게 나타났지만, event 에 대해 시간 차이를 보이고 있다. 이러한 차이는 같은 입력이지만 개편 전후 실수 형태의 변수들이 COR, ESF, HS 등의 팩키지에서 사용한 double precision 과는 달리 single precision 으로 정의된 변수로 인해 유효숫자 자릿수의 한계에서 초래된 차이로 보여지고, 코드 내부에서 자동으로 조절되는 적분시간간격(dt)의 차이에도 원인이 있을 것으로 보여지며(dt 는 현상이 복잡할수록 급격히 감소하며, 이로 인해 지극히 작은 값의 차이에 따라 계산되는 길의 선택이 달라질 수 있음), 현재 그 외의 다른 원인의 가능성에 대해서도 점검하고 있다.

이 외에도 두 경우에서의 CF 팩키지의 실행에 소요된 CPU time 을 비교해 보면 다른 팩키지 개편의 경우 결과와 비슷하게 차이를 보였으며. 이는 부프로그램 호출시 array 로의 전환/환원 및 even/odd cycle 에 대한 달라진 데이터 처리로 인해 생길 수 있는 결과로 판단된다.

5. 결론 및 추후 연구 계획

MIDAS에서의 데이터 저장 및 전달방법을 재구성하기 위하여 기존 MELCOR 코드의 전체적인 데이타 흐름을 분석하고, 특히 CF 팩키지를 사용한 RN1 팩키지에 대한 데이터 정보 및 포인터 변수를 분석하여 CF 팩키지에 적용할 모듈을 구성하였다. 이 과정에서 포인터 변수를 없애고, 대신 직접변수를 사용하도록 데이터구조를 재구성하였다.

새로 구성된 모듈을 사용하여 CF 팩키지의 관련 subroutine을 개편하였으며, CF 팩키지의 개편결과로 인해, 코드의 전반에 걸쳐 용이한 변수 파악과 기존 코드의 개선이나 새로운 모델의 접목이 가능한 코드 구조로 확장할 수 있음을 확인하였다. 비슷한 graph의 추이, 여러 가지의 실행 결과 등을 종합해 볼 때, 개편은 제대로 수행되었다고 볼 수 있으며, 결과 차이를 규명하는 것과 benchmarking을 위한 problem set 마련이 필요하다고 보여지며, 이에 대한 상세한 작업은 MELCOR code 전반에 관한 비교/검증 작업을 진행할 계획으로 있다.

따라서 본 논문에서 제시된 CF 팩키지의 개편 내용을 동일한 방법에 의해 각 팩키지별로 적용할 수 있도록 지속적으로 확장하여 전체 코드를 변환하고 있으며[11], 우리 실정에 맞도록 코드를 국산화하기 위해서는 기존 모델의 개선에 대한 연구와 새로운 모델의 추가에 대한 연구가 지속적으로 요구된다.

감사의 글

본 연구는 과학기술부의 원자력연구개발 중장기계획사업의 일환으로 수행되었다.

참고문헌

[1] A Multi-Dimensional Thermal-Hydraulic System Analysis Code, MARS 1.3.1, Vol.31, Number 3, pp.344-363, June 1998.
[2] Realistic Thermal-Hydraulic System Code Development Workshop, '98 Fall KNS Conference, 1998.
[3] 3rd MARS Users Group Meeting, '99 Dec. 2 KAIST Conference, 1999.
[4] 송용만, 박선희, 김동하, Development of a Computer Program for Automatic Variable Conversion in MELCOR Code, 한국원자력학회, 추계학술발표회 논문집, 2000
[5] 박선희, 송용만, 수작업을 이용한 MIDAS 중대사고해석 코드 개편, KAERI/TR-2220/2002, Jun, 2002.
[6] 박선희, 송용만, 김동하, A Restructuring Proposal for MIDAS, 한국원자력학회 춘계학술발표회 논문집, 2000.

[7] S.H.Park, H.D.Kim, D.H.Kim, Y.M.Song, B.D.Chung, Development of Restructuring Template for MELCOR, 5th PSAM Conference, Japan, 2000.11.26 – 2000.12.2
[8] 박선희, 송용만, 김동하, A Restructuring of TF package for MIDAS Computer Code, 한국원자력학회 추계학술발표회 논문집, 2002.
[9] 박선희, 김동하, 김고려, 중대사고해석 코드(MIDAS)개발을 위한 RN1 팩키지 개편, 한국원자력학회 추계학술발표회 논문집, 2003.
[10] 김동하, 박선희, MELCOR 코드 구조 분석, KAERI/TR-1543/00, June, 2000.
[11] 김동하, 박선희, 송용만, 중대사고해석 코드 개발을 위한 MELCOR 개편시안, KAERI/TR-1536/2000, March, 2000
[12] 김동하, 외 12인, 중대사고 위해도 완화전략분석 종합전산코드 개발, KAERI/RR-2216/2001, pp 302-321, May, 2002.