'2000춘계학술발표회 논문집
한국원자력학회

중대사고해석 코드 개발을 위한 개편시안

A Restructuring Proposal for MIDAS

박선희, 송용만, 김동하
한국원자력연구소
대전광역시 유성구 덕진동 150

요 약

중대사고해석 코드인 MIDAS의 코드 개편 시안을 개발하기 위하여 기존 MELCOR 코드의 데이터 저장 및 전달 방법을 파악하였으며, 기존 코드의 데이터 전달방법을 수정하여 SPR 팩키지에 적용할 모듈을 개발하였다. 기존의 MELCOR 코드는 데이터 타입별로 고정적인 크기의 저장장소를 확보하고, 그 한도내에서 가변적인 크기의 데이터를 처리하기 위해 팩키지 순서대로 데이터를 저장하고, 포인터를 사용하여 데이터를 추적한다. 이로 인해 메모리의 낭비는 물론 내용을 파악하기 어려운 단점이 있다. 하지만, FORTRAN90이 발표된 이후에는 저장장소의 동적 할당과 사용자가 정의한 data type을 사용할 수 있게 되었으며, 이 기능을 적용하여 메모리를 효과적으로 사용하고 코드의 내부를 쉽게 이해할 수 있도록 격납건물 살수계통이 관련된 SPR 팩키지에 대한 모듈을 구성하고, 시안을 개발하였다. 또한 개편시안의 검증을 위하여 개편이후의 계산 결과와 기존의 결과를 비교하였으며, 결과가 같음을 확인하였다. 본 논문에서 언급된 SPR 팩키지에 대한 개편시안은 검증 결과 코드 전체로 확장하여 사용할 수 있는 타당성을 제시하여 주며, 이 개편시안은 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

In order to develop a template for the MIDAS computer code, current data saving and transferring methods used in MELCOR are addressed first, and then a restructured module for the SPR package has been developed to be applied to MELCOR. The current MELCOR code ensures a fixed-size storage for four different data types, and manages the input-dependent data size by storing the data on the stacked packages. It uses pointer to identify the variables between the packages. This technique causes a difficult grasping of the meaning of the variables as well as waste of memory. New features of FORTRAN90, however, make it possible to allocate the storage dynamically and to use the user-defined data type, which lead to a restructured module development for the SPR package. An efficient memory treatment and an easy understanding of the code are the benefit from this developed module. The validation of the template has been done by comparing the results of the modified code with those from the existing code, and it is confirmed that the results are the same. The template for the SPR package hints the extension of the template to the entire code. It is expected that the template 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.

기존의 MELCOR 코드를 파악하기 위해서는 전체 데이터의 저장 구조 및 저장 방법을 정확하게 알아야 한다. 기존의 MELCOR 코드가 효율적인 데이터 전달 및 저장을 위하여 포인터 변수를 사용하고 있으나, 이로 인해 코드내부에 대한 전반적인 이해가 어려우며, 특히 다른 모델을 추가할 경우 연계 작업이 복잡하여 많은 어려움이 따르고 있다.

기존 코드의 데이터 저장 방법은 데이터 별로 고정적인 크기의 저장장소를 확보하고, 그 한도내에서 가변적인 크기의 데이타를 처리한다. 이로 인해 메모리의 낭비가 발생하며, 데이터의 분류가 내용 위주가 아닌 컴퓨터가 처리하는 데이터 타입 위주로 된 것이어서 내용을 파악하기가 어렵다. 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 모듈 구성시에 지침이 되는 변수 명명에 관한 연구를 하였으며, 구성된 모듈을 적용하여 subroutine 개편시 도출변수(derived type variables)로의 변경을 자동으로 수행하는 프로그램을 개발하여 사용하였다.[4]

코드의 전체적인 개편에 앞서 간단한 팩키지에 대한 개편 시안을 먼저 개발하여 타당성을 분석하고 해당 팩키지에 대하여 검증 작업을 거친 후 전체 코드로 확장하는 전략으로 수행하고 있다. 개편 시안 대상 팩키지는 26개의 팩키지 중 외부 팩키지와의 연계가 가장 적은 SPR (격납건물 살수계통 담당) 로 선정하여 코드 구조 개편 시안을 개발하였으며, 개편 이전의 결과와 이후의 결과를 비교하여 검증작업을 수행하였다.

2. 기존 구조 파악

MELCOR code는 압력자료 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. (그림 1.)

모든 수행부분은 COR, HT, SPR 등 24개의 팩키지로 구성되어 있으며, 각 팩키지간에 data를 공유하여 수행되며, 각 팩키지마다 수십개의 subroutine이 있고, message를 기록하는 subroutine, code 수행 흐름을 제어하는 subroutine 등 전체적으로 수백개의 subroutine으로 구성되어 있다.

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, argument 값에 따라 read 또는 write를 수행한다. Real, integer, logical, character의 4개 데이터 타입별 array에 대한 읽기와 쓰기, subroutine xyzPRS을 호출하여 1, 2단계의 포인터에 대한 읽기와 쓰기, subroutine xyzRIO을 호출하여 coefficient들에 대한 읽기와 쓰기를 수행한다.

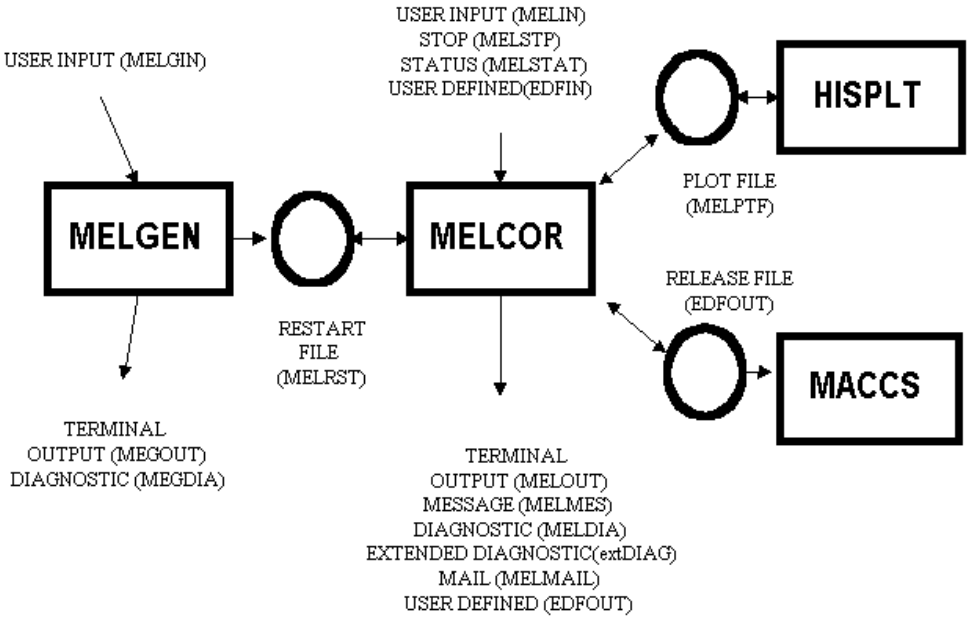


그림 1. MELCOR CODE 전체 구성도

2.2 Database

MELCOR 코드의 Data의 종류는 problem size에 따라 Dynamic data와 Fixed size data가 있고, time에 따라 Old/New 값과 Single value가 있다. 또한 모든 팩키지의 data를 저장하기 위하여 다음과 같은 4가지 타입의 저장 장소가 예약되어 있다.

- XREALX : floating point variables (500,000개의 array)
- INTEGE : integer variables (50,000개의 array)
- LOGICA : logical variables (5,000개의 array)
- CHARAC : character variables (30,000개의 array)

효율적인 데이터 전달 및 저장을 위해 네 개의 데이터 타입별 포인터를 사용하고 있다. 각 데이터 타입의 고정된 크기 내에서 사용자의 입력에 따라 저장 용량의 크기를 최적화하여 데이터를 저장한다. 이때 사용하는 데이터 저장 방법은 팩키지별 데이터를 차례 차례로 축적하는 방법으로(그림 2), 임의의 팩키지에서 사용하고 있는 변수를 이용하고자 할 경우 실제로 쓰이는 변수 이름 대신 위치를 나타내는 포인터를 이용하여야 하며, 또한 포인터 위치를 파악하기 위해서는 전체 데이터의 저장 구조 및 저장 방법을 정확하게 이해하고 있어야 한다.

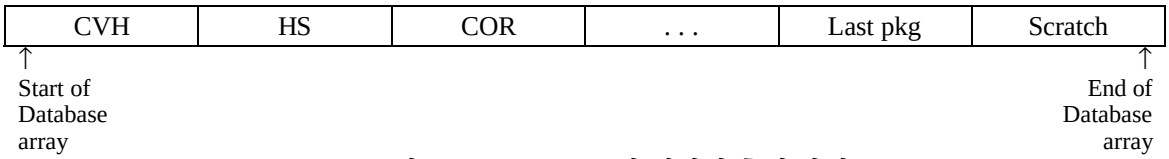
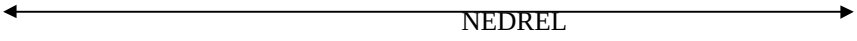
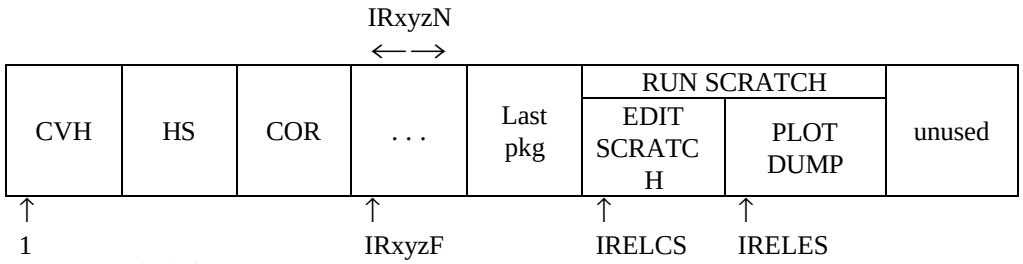


그림 2. Database array의 팩키지 축적 방식

포인터는 다음과 같이 2 단계로 구분하여 사용되고 있다.

- 팩키지와 scratch 영역의 포인터 (그림 3)
- 각각의 팩키지 array를 가리키는 팩키지영역내의 포인터 (그림 4)





* xyz : **팩키지** name

* XREALX array (**다른** 3**개의** array**인** INTEGE, LOGICA, CHARAC**도 비슷함**)

그림 3. **팩키지와** scratch**영역의 포인터**

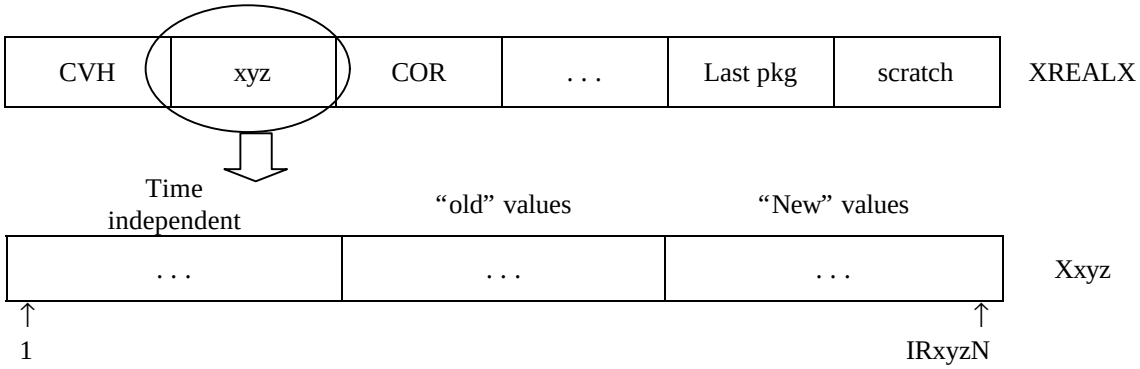


그림 4. **팩키지 영역내의 포인터**

Database **이동의** 1**단계에서는 데이터 타입별** array**의 개수와 해당 펙키지에서 사용하는** array**의 시작지점을** argument**로 전달하며**, **그** subroutine **예는 그림** 5**와 같다**.

```
      SUBROUTINE xyzDBC
*- INCLUDE BLANK
      COMMON /DBREAL/ NEDREL, IRELCS, IRELES
      COMMON /      / DREALX(NUMREL/2)
      DOUBLE PRECISION DREALX C
      DIMENSION XREALX(NUMREL)
      EQUIVALENCE (XREALX(1), DREALX(1))
      DOUBLE PRECISION VREALX(NUMREL/2)
      EQUIVALENCE (VREALX(1), DREALX(1))
*- INCLUDE xyzDB
      COMMON /xyzDB/  IRxyzF , IRxyzN , IIxyzF , IIxyzN ,
     1              ILxyzF , ILxyzN , ICxyzF , ICxyzN
       CALL xyzDBD(
     1 IRxyzN , XREALX(IRxyzF) ,
     2 IIxyzN , INTEGE(IIxyzF) ,
     3 ILxyzN,  LOGICA(ILxyzF),
     4 ICxyzN,  CHARAC(ICxyzF), . . . )
C
      RETURN
      END
```

그림 5. 1**단계** database **이동에 대한** subroutine **예**

Database **이동의** 2**단계에서는 더에타 타입별 팩키지내에서 사용되는 변수에 대한** array**내의 포인터를** argument**로 전달하며**, **그** subroutine **예는 그림** 6**과 같다**.

```
      SUBROUTINE xyzDBD (
     1 NxyzR , Rxyz ,
     2 NxyzI , Ixyz ,
     3 NxyzL,  Lxyz,
     4 NxyzC,  Cxyz, . . . )
C
       DIMENSION Rxyz(NxyzR), Ixyz(NxyzI), . . .
C
*- INCLUDE xyzPNT
      COMMON /xyzPNT/  Nvar1, Nvar2, . . .
C
      CALL xyzRUN (Nvar1, Rxyz(Ivar1), Nvar2,Ixyz(Ivar2), . . . )
C
      RETURN
      END
```

그림 6. 2**단계** database **이동에 대한** subroutine **예**

Database **이동의 작업단계에서는** argument**로 전달된 변수들을 지역변수로 사용하고 있고**, **그** subroutine **예는 그림** 7**과 같다**.

```
      SUBROUTINE xyzRUN(
     1  Nvar1, var1, Nvar2, var2, . . .)
C
       DIMENSION var1(Nvar1), . . .
C
C     FROM HERE ON, SIMPLY USE var1 AND var2.
C     FORGETTING ALL DETAILS OF WHERE AND HW THEY ARE STORED.
C
      CALL  xyz. . .
C
      RETURN
      END
```

그림 7. **작업단계의** database **이동에 대한** subroutine **예**

2.3 포인터 변수

포인터 변수를 분석하기 위하여 database**의 이동에 관련된** subroutine**인** xyzDBC subroutine**과** xyzDBD subroutines**에 있는** comment **및** argument pass**를 분석하였다**. **단계별 포인터 변수는 다음과 같다**.

(1) 1**단계의 포인터 변수**

Subroutine xyzDBC **내에** ‘xyzDB’ **라는** common block**으로 정의되어 있으며**, (**그림** 8) 4**개의** data type**별로** 2**개씩의 변수를 사용한다**. 1**단계의 포인터 변수는 모든 팩키지에 대해 동일하게 적용된다**. (**그림** 9)

```
*- INCLUDE xyzDB
C     POINTERS TO xyz PACKAGE DATABASE
      COMMON /xyzDB/ IRxyzF, IRxyzN, IIxyzF, IIxyzN,
     *               ILxyzF, ILxyzN, ICxyzF, ICxyzN
C
```

그림 8. 1**단계의 포인터 변수 정의**

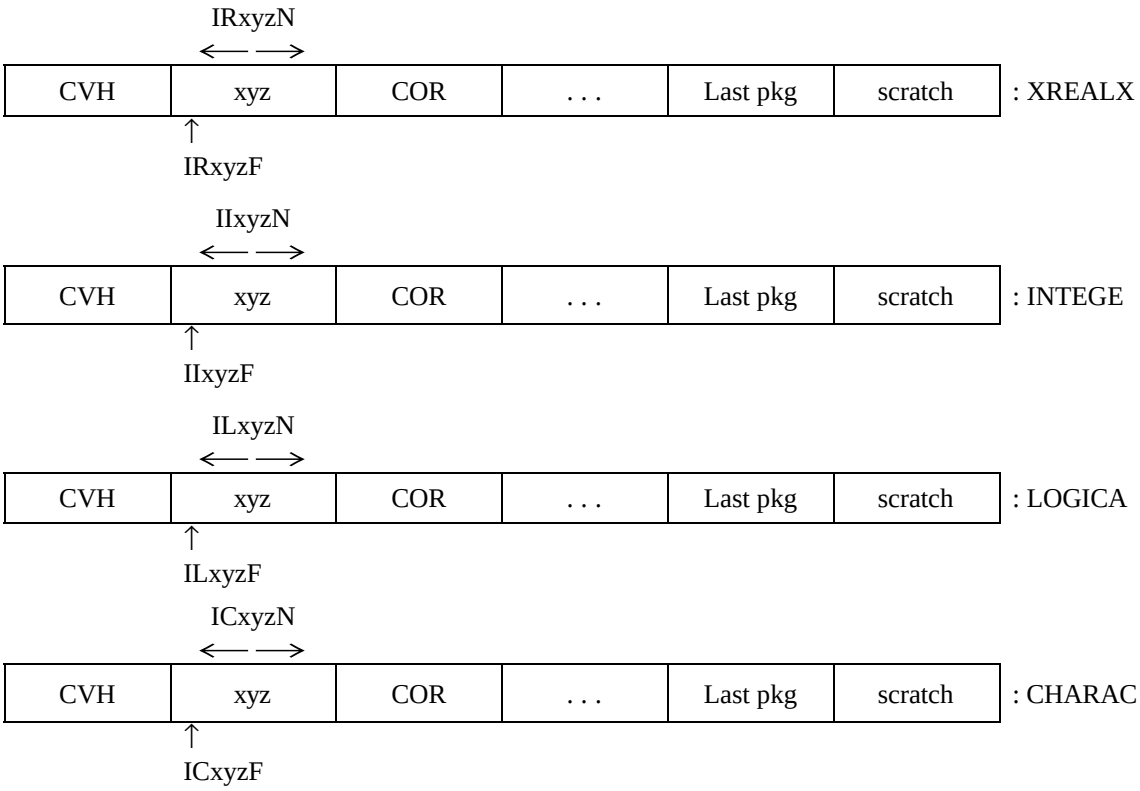


그림 9. 1**단계의 포인터 변수의 내용**

(2) 2**단계의 포인터 변수**

Subroutine xyzDBD **내에** ‘xyzPNT’**라는** common block**으로 정의되어 있으며**, 4**개의** data type**에 대한** database **중 어느 하나를 가리키도록 정의된다**. 2**단계의 포인터 변수는 팩키지에 따라 다르게 적용된다**. **전체의 팩키지중** SPR **팩키지에 대한 포인터 변수 정의는 그림** 10**과 같다**.

```
      COMMON /SPRPNT/  MSPJNM, MKCVFM, MKCVTO, MIFRSP, MFRSPT,
     1                 MIVOLF, MIVOLT,
     2                 MSPNUM, MSPNAM, MIVOL,
     3                 MSPCON, MONOFF, MSRCVL, MFALLS,
     4                 MTDROP, MTMPCF, MSPFLO, MFLOCF, MSPFLA,
     5                 MVWFRQ, MSRCJN, MSRCSZ,
     6                 MDIAMO, MDRFRQ, MFLORT, MFLRTA, MDRPMS,
     7                 MSPCVL, MTOTVL, MFALLH, MCAROV,
     8                 MHTTRN, MMSTRN,
     9                 MCVISM, MSPSMP, MCVSUM, MXSUMP,
     A                 MSMPHT, MSMPMS,
     B                 NDSPSR, NDSPVL, NDSPJN, NDSPSM, NDSUMP,
     C                 NSPSRC, NSPVOL, NSPJUN, NSPVSM, NSUMPS,
     D                 MXSPSZ, MXSPJN, MIFDRY, MLDRYO, MLDRYN,
     E                 MCVLSR, MCVXSR, MHEXCO, MHEXCN, IORSPR,
     F                 MELDRY, MELWET, IOLSPR
```

그림 10. SPR **팩키지의** 2**단계의 포인터 변수 정의**

SPR **팩키지에 대한 포인터 변수가 의미하는 내용은 팩키지내의** subroutine**에서 찾아볼 수 있으며**, **이를 근거로** SPR **팩키지에 대한 모듈을 구성하였다**.

(3) 다른 팩키지에서의 SPR data 사용여부

개편대상의 SPR data는 SPR 팩키지이외에 RN2 팩키지의 subroutine RN2LAM 에서만 참조하고 있으며, 그 subroutine을 분석한 결과 SPR data에 관련된 포인터 변수는 사용하지 않고 지역변수를 사용하여 참조하는 것을 확인하였다. 이것으로 SPR 팩키지를 개편하였을 때 기존의 MELCOR 와 접목하여 검증이 가능함을 확인하였다.

3. 개발 내용

코드의 전체적인 개편에 앞서 처음부터 전체적인 구조를 개편하기 보다는 우선 간단한 팩키지에 대한 개편 시안을 먼저 개발하여 타당성을 분석하고 일부 팩키지에 대하여 검증 작업을 거친 후 전체 코드로 확장하는 방법으로 진행하였다. 또한 기존의 restart file을 그대로 사용하기 위하여 restart file을 작성하는 MELGEN 부분은 그대로 유지하고, restart file 작성 이후의 계산 수행 부분인 MELCOR부분을 개편대상으로 하였다. 이 때, MELGEN과 MELCOR에서 모두 사용되는 subroutine의 경우에는 MELGEN 의 수행에 지장이 없도록 개편된 subroutine을 다른 이름으로 저장하여 사용하도록 하였다. Subroutine을 분석한 결과, 변경이 필요한 subroutine중에는 MELGEN과 MELCOR에서 동시에 사용되는 subroutine은 없었다.

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 type data type)로 바꾸어 팩키지별로 모듈화 하였다.[5]

개편 시안 대상의 팩키지는 26개의 팩키지중 외부 팩키지와의 연계가 가장 적은 SPR(격납건물 살수계통 담당)로 선정하였으며, SPR 팩키지에서 사용하는 변수를 모듈화 된 구조로 전환하고, 동적 저장 기법을 이용하여 계산 효율을 높였다. MELCOR 전체의 수행을 제어하는 subroutine과 SPR 팩키지에 속해 있는 70여 개의 subroutine을 분석하였으며, 개발된 자동 변환 프로그램(MELtoMID)의 사용으로 수작업을 최소화하여 관련 subroutine을 변환하였다.[4]

3.1 모듈 구성

Subroutine SPRDBD 에서 subroutine SPRRUN 을 호출하면서 포인터 변수를 사용하였으며, subroutine SPRRUN 및 그 이후에 호출되는 subroutine에서는 포인터변수와 같은 이름의 지역변수(local variable)를 사용하고 있다. 다른 몇 개의 subroutine은 SPRDBD에서와 같이 포인터 변수를 직접 사용하고 있다. 2단계의 포인터 변수 분석에 근거하여 구성된 SPR 팩키지의 모듈은 그림 11과 같다.

3.2 Subroutine 개편

Subroutine의 개편에 앞서 먼저 FORTRAN77로 쓰여진 기존 MELCOR 코드 전체를 FORTRAN90 언어로 변환하였다. 이때 사용한 프로그램은 이미 개발되어 성능이 입증된 F77290을 사용하였다. SPR 팩키지의 subroutine의 경우, argument가 전달되는 모든 단계를 확인한 결과, 포인터 변수를 직접 사용하는 subroutine은 8개였으며, 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. 예로, ‘ DRPIMS(ISIZE, ISPATH)’ 으로 선언된 array는 새로 구성된 모듈에서 ‘ SR_SRC(ISPATH)%SR_SZ(ISIZE)%DRPIMS’ 로 선언되었다. 또한 첨자의 위치가 바뀌는 경우가 있는데, 이것은 기존의 2차원 array구성에서는 첫째 첨자와 둘째 첨자의 위치가 중요한 의미가 없었지만, 새로 구성된 모듈에서는 포함관계의 의미가 있으므로 기존의 순서와 다르게 되는 경우가 발생한다. 이에 대한 내용을 표 1에, 개편 전후의 subroutine 비교를 표 2에 나타냈다.

```
      MODULE SPR_MDL
!
!          *****  SPRAY GLOBAL DATA   *****
!                (SPRAY JUNCTION DATA)
```

```
      INTEGER :: NDSPJN, NSPJUN
      REAL(8), ALLOCATABLE :: FRSPTR(:,:)
!               (SPRAY SOURCE DATA)
      INTEGER :: NDSPSR, NSPSRC, MXSPSZ, MXSPJN
!               (SPRAY VOLUME DATA)
      INTEGER :: NDSPVL, NSPVOL
!               (SUMP DATA)
      INTEGER :: NDSPSM, NSPVSM, NDSUMP, NSUMPS
!
!          *****  SPRAY JUNCTION DATA   *****
      TYPE SPR_J  ; SEQUENCE
         INTEGER :: ISPJNM,  KCVFM, KCVTO
         REAL(8) :: FRSPTI
      END TYPE
      TYPE (SPR_J), ALLOCATABLE :: SPR_JN(:)
!
!          *****  SPRAY SOURCE DATA   *****
      TYPE  SPR_S1 ; SEQUENCE
      .
      .
      .
      .
      .
      TYPE (SPR_S6), ALLOCATABLE :: SC_SPR(:)
    END MODULE SPR_MDL
```

그림 11. **구성된 모듈**

표 1. 2**차원** array**의 변경내용**

기존 방식에서의 사용	**신규 방식에서의 사용**	**변경 내용**
FRSPTR(NDSPVL, NDSPVL)	FRSPTR(NDSPVL, NDSPVL)	2**차원** array**를 그대로 사용함**
IVOLFR(MXSPJN, NDSPSR)	SPR_SR(NDSPSR)%JNC(MXSPJN)%IVOLFR	index **위치 변경됨**
IVOLTO(MXSPJN, NDSPSR)	SPR_SR(NDSPSR)%JNC(MXSPJN)%IVOLTO	index **위치 변경됨**
DIAMO(MXSPSZ, NDSPSR)	SPR_SR(NDSPSR)%SR_SZ(MXSPSZ)%DIAMO	index **위치 변경됨**
DRFREQ(MXSPSZ, NDSPSR)	SPR_SR(NDSPSR)%SR_SZ(MXSPSZ)%DRFREQ	index **위치 변경됨**
FLORTO(MXSPSZ, NDSPSR)	SPR_SR(NDSPSR)%SR_SZ(MXSPSZ)%FLORTO	index **위치 변경됨**
FLORTA(MXSPSZ, NDSPSR)	SPR_SR(NDSPSR)%SR_SZ(MXSPSZ)%FLORTA	index **위치 변경됨**
DRPIMS(MXSPSZ, NDSPSR)	SPR_SR(NDSPSR)%SR_SZ(MXSPSZ)%DRPIMS	index **위치 동일함**
TDFLO(MXSPSZ, NDSPVL)	SC_SPR(MXSPSZ)%VL(NDSPVL)%TDFLO	index **위치 동일함**
DIAMF(MXSPSZ, NDSPVL)	SC_SPR(MXSPSZ)%VL(NDSPVL)%DIAMF	index **위치 동일함**
TDFIN(MXSPSZ, NDSPVL)	SC_SPR(MXSPSZ)%VL(NDSPVL)%TDFIN	

표 2. Subroutine 개편전후의 비교

Subroutine SPRYVL (개편이전)

```
      SUBROUTINE SPRYVL (DT,NSRCSZ,CAROVR,TDINIT,DIAMI,TDFLO,DRPIMS,
     1  ZTOT,PTOT,PPSTM,TBULK,ENTSTM,CVVOL,DIAMF,TDFIN,
     2  TRHTSP,TRMSSP,VSPSMM,VSPSME,
     3  XMASS, NUMMAT, NSEC, XMSEC, IICLS, HPART, XLMDAV, XLMDAA)

      . . . . . .

            DO 10 ISPR = 1,NSRCSZ
        DIAMF(ISPR) = DIAMI(ISPR)
        TDFIN(ISPR) = TDINIT(ISPR)
10    CONTINUE

      . . . . . .

      RETURN
      END
```

Subroutine SPRYVL (개편이후)

```
      SUBROUTINE SPRYVL (DT, ISVOL, ISPATH, ZTOT, PTOT, PPSTM, TBULK20, &
      ENTSTM, CVVOL, TRHTSP, TRMSSP, VSPSMM, VSPSME, XMASS, NUMMAT,  &
      NSEC, XMSEC, IICLS,  HPART, XLMDAV, XLMDAA)

         . . . . . .

           USE SPR_MDL

         . . . . . .


!       INITIALIZE FINAL DROP DIAMETER AND TEMPERATURE
      DO 10 ISPR = 1, SPR_SR(ISPATH)%NSRCSZ
       SC_SPR(ISPR)%VL(ISVOL)%DIAMF = SC_SPR(ISPR)%DIAMI
       SC_SPR(ISPR)%VL(ISVOL)%TDFIN = SC_SPR(ISPR)%TDINIT
   10 END DO

      . . . . . .

      RETURN
      END SUBROUTINE SPRYVL
```

4. 개발결과 및 검증

포인터 변수의 사용을 없애고 FORTRAN90**의 특징인 동적 저장기법을 사용하여야 하므로**, SPR **팩키지의 개편에 앞서** FORTRAN77**로 쓰여진 기존** MELCOR **코드 전체를** FORTRAN90 **언어로 변환하였으며**, **이와 관련된** library**와** execution file**도 새로이 생성하였다**. **결과를 비교하기 위하여 울진발전소에서의** TLOFW**를 해석하였으며**, TLOFW **발생 전인** 100**초에** spray**가 작동하도록 가정하였다**.

FORTRAN77**로 쓰여진 기존의** MELCOR **수행 결과와** FORTRAN90 **언어로 변환한** MELCOR**의 수행 결과를 비교하였으며**, **결과가 완벽히 일치함을 확인하였다**.(**그림** 13.) **또한** FORTRAN90 **언어로 변환한** MELCOR **수행 결과와 개편된** SPRMELCOR**의 수행 결과를** SPR**팩키지와 관련된 주요 변수를 비교하였으며**, **그 결과 또한 완벽히 일치함을 확인하였다**. (**그림** 14.)

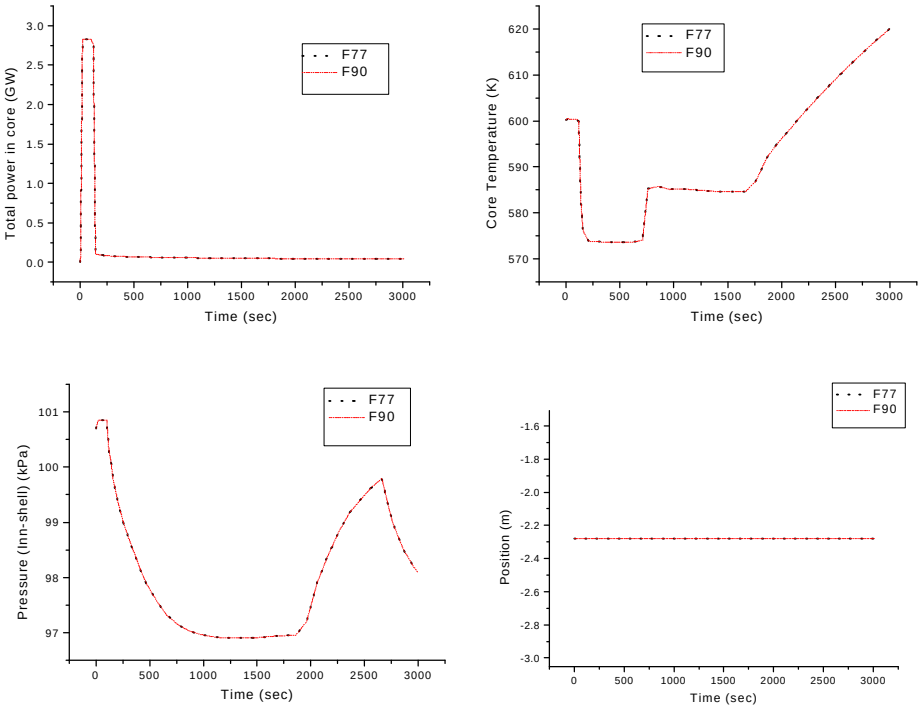


그림 13. FORTRAN90 변환 전후의 수행결과 비교

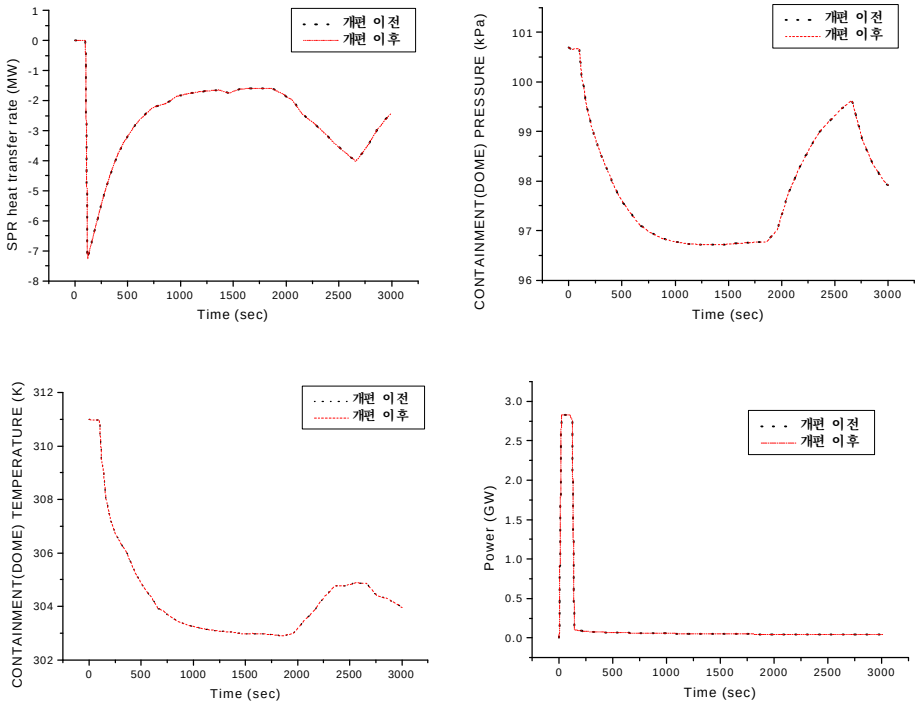


그림 14. SPR 팩키지 개편 전후의 수행결과 비교

5. 결론 및 추후 연구 계획

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

새로 구성된 모듈을 사용하여 SPR팩키지의 관련 subroutine을 개편하였으며, SPR팩키지와 관련된 주요 변수에 대하여 바교한 결과, 개편 전후의 변수값이 같음을 확인하여 SPR팩키지에 적용한 개편시안의 타당성을 검증하였다. 이로 인해, 코드의 어느 부분에서든지 용이한 변수의 파악과 기존 코드의 개선이나 새로운 모델의 접목이 가능한 코드 구조로 확장할 수 있는 기반이 조성되었다. 따라서 본 논문에서 제시된 개편시안을 동일한 방법에 의해 각 팩키지별로 적용할 수 있도록 확장하여 전체 코드를 변환할 계획이며, 우리 실정에 맞도록 코드를 국산화하기 위해서는 기존 모델의 개선에 대한 연구와 새로운 모델의 추가에 대한 연구가 지속적으로 요구된다.

감사의 글

본 연구는 과학기술부의 원자력연구개발 사업의 일환으로 수행되었습니다.

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] 중대사고해석 코드 개발을 위한 MELCOR 개편시안, KAERI/TR–1536/2000, March, 2000
[5] DIGITAL Fortran Language Reference Manual