MARS Code in Linux Environment

Moonkyu Hwang*, Sung-Won Bae, Jae-Joon Jung, Bub-Dong Chung
*Korea Atomic Energy Research Institute, P.O. box 150, Yusong-gu, Deajeon, Korea mkhwang@kaeri.re.kr**

1. Introduction

The two-phase system analysis code MARS [1] has been incorporated into Linux system. The MARS code was originally developed based on the RELAP5/MOD3.2 and COBRA-TF. The 1-D module which evolved from RELAP5 alone could be applied for the whole NSSS system analysis. The 3-D module developed based on the COBRA-TF, however, could be applied for the analysis of the reactor core region where 3-D phenomena would be better treated. The MARS code also has several other code units that could be incorporated for more detailed analysis. The separate code units include containment analysis modules and 3-D kinetics module. These code modules could be optionally invoked to be *coupled* with the main MARS code. The containment code modules (CONTAIN and CONTEMPT), for example, could be utilized for the analysis of the plant containment phenomena in a coupled manner with the nuclear reactor system. The mass and energy interaction during the hypothetical coolant leakage accident could, thereby, be analyzed in a more realistic manner. In a similar way, 3-D kinetics could be incorporated for simulating the three-dimensional reactor kinetic behavior, instead of using the built-in point kinetics model.

The MARS code system, developed initially for the MS Windows environment, however, would not be adequate enough for the PC cluster system where multiple CPUs are available. When parallelism is to be eventually incorporated into the MARS code, MS Windows environment is not considered as an optimum platform. Linux environment, on the other hand, is generally being adopted as a preferred platform for the multiple codes executions as well as for the parallel application.

In this study, MARS code has been modified for the adaptation of Linux platform. For the initial code modification, the Windows system specific features have been removed from the code. Since the coupling code module CONTAIN is originally in a form of dynamic load library (DLL) in the Windows system, a similar adaptation method called *shared library* in the Linux platform had to be incorporated

2. Methods and Comparisons

2.1 Code Modification

Two major steps, therefore, were necessary for the code modification. Initially, the code segments specifically targeted for the Windows platform had to be transformed into the compatible form in Linux system. The task is relatively straightforward, especially since one of the MARS code versions is BIN version which is intended for the console application. The BIN version therefore naturally served as base version for Linux platform.

The MARS code in Windows environment provides optional execution of the coupled code module. That is, for instance, the CONTAIN module could be executed in parallel with the main MARS code. The computer memory is, therefore, allocated for the CONTAIN module only when the users choose to perform the coupled analysis. The similar functionality had to be provided in the Linux system. The shared library along with so called *dynamic loading* has been used for this purpose. The calling program and the dynamic shared library are demonstrated in Figure 1 and 2.

```
program dl
!     demonstraion prgram for dynamic loading
!DEC$ ALIAS dlopen, "dlopen"
!DEC$ ALIAS dlsym, "dlsym"
!DEC$ ALIAS dlerror, "dlerror"
interface
   function dlsym(handle,name)
   !dec$ attributes value::handle
   integer(int_ptr_kind()) dlsym
   character(*), intent(in)::name
   integer(int_ptr_kind()),intent(in)::handle
   end function dlsym

   function dlopen(file, mode)
   integer(4)::dlopen
   character(*), intent(in)::file
   integer(4), intent(in)::mode
   !dec$ attributes value::mode
   end function dlopen
end interface

   integer(2) ::i,i2, ishared
   integer ::RTLD_LAZY=Z"00001"
   integer ::RTLD_NOW=Z"00002"
   character*132 :: messagex
   integer(int_ptr_kind()) handle
   pointer (addrFunc,subr)
   handle= dlopen("my_lib_s.so"C, RTLD_LAZY)
   addrFunc=dlsym(handle, "twotimes_")
   i=3
   call subr(i,i2)
   write(*,*) 'i2=', i2
   stop
   end program dl
```

Fig 1. Calling Program A

```
subroutine twotimes(i,i2)
   integer i,i2
   i2=i*2
   return
end subroutine twotimes
```

Fig 2. Subroutine B(Shared Library)

The calling program 'A' listed in Figure 1 is used to dynamically load the subroutine shared library during the program execution. The code segment 'B' listed in Figure 2 is made as a shared library beforehand. Without the added program segment in the program 'A', the shared library would be loaded into the memory during the initial loading of the program 'A'. The program 'A' successfully identifies a target subroutine in the shared library during the execution and loads the selected module in the memory. The method demonstrated could be, therefore, directly applied for the code modification of separate code modules developed as a DLL in the Windows environment.

2.1 Results Comparison

For the verification of the code porting, calculations have been performed using *typpwr.i* which is for the typical W/H 4-loop PWR of 3600 MWth. The problem is to simulated loss of coolant accident in the cold leg from the full power normal operating conditions. Three cases were tested for both 12-inch LBLOCA and 4-inch SBLOCA. The three cases are comprised of calculations with MARS compiled under COMPAQ Fortran compiler [2] for MS Windows, MARS from Intel Fortran compiler [3] also for MS windows, and finally MARS in the Linux system. The Intel Fortran compiler [4] was used in the Linux platform.

Figures 3 to 5 present the primary system pressure, the secondary system pressure, and core bottom void fraction during the LBLOCA. Figure 6 to 8 show the similar parameters for SBLOCA calculation. As seen from the Figure 3 to 5, very good agreements are observed for LBLOCA calculation. In case of SBLOCA, however, the deviations among the calculation results are observed, especially from around 600 seconds after the accident initiation. The discrepancy, however, is not believed as an error in the code modification. It rather appears to stem from the compiler differences. It is noted that the different results among the calculations performed in the various platforms are well known for other system analysis code as well, including RELAP5. Eventually, the discrepancy certainly needs to be eliminated for MARS code as well as other system analysis codes.

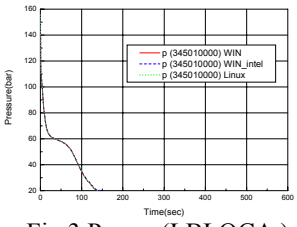

Fig.3 $P_{primary}$ (LBLOCA)

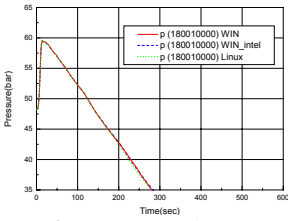

Fig. 4 $P_{secondary}$(LBLOCA)

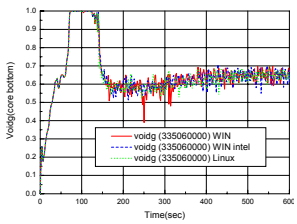

Fig. 5 α_g at core bottom (LBLOCA)

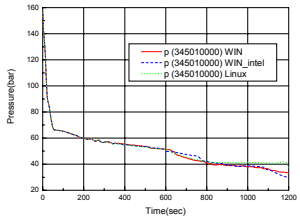

FiG. 6 $P_{primary}$ (SBLOCA)

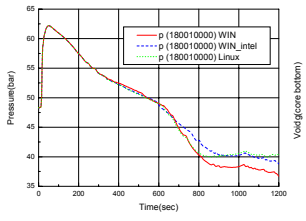

Fig. 7 $P_{secondary}$ (SBLOCA)

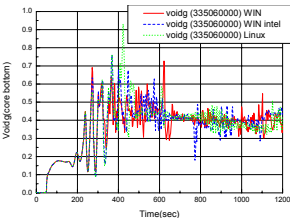

Fig. 8 α_g at core bottom (SBLOCA)

3. Summary

In this study, MARS code has been modified for the adaptation of Linux platform. For the initial code modification, the Windows system specific features have been removed from the code. Since the coupling code module such as CONTAIN was originally in a form of dynamic load library (DLL) in the Windows system, a similar adaptation method called shared library in the Linux platform had to be incorporated. When compared with results by two separate compilers at Windows system, MARS code results show successful code adaptation for the Linux platform.

REFERENCES

[1] 1. Jeong, J.-J., Ha, K. S., Chung, B. D., Lee, W. J., "Development of A Multi-dimensional Thermal-Hydraulic System Code, MARS 1.3.1,"*Annals of Nuclear Energy*26(18), 1161-1642, 1999.

[2] Compaq Fortran Language Reference Manual, Digital Equipment Corporation, 1999

[3] Intel Fortarn Language Reference Manual, Intel corporation, 2003-2004

[4] Intel Fortran Compiler for Linux Systems User's Guide, Intel Corporation, 2003-2004.