8.6 MIGRATION OF LEGACY WSR-88D ALGORITHMS AND PRODUCT GENERATORS TO THE OPEN SYSTEM RPG

by Zhongqi Jing[1,3], Steve Smith[2], Michael Jain[3] and Allen Zahrai[3]
[1]Cooperative Institute for Mesoscale Meteorological Studies,
University of Oklahoma, Norman, Oklahoma
[2]NOAA/NWS/WSR-88D Operational Support Facility
[3]NOAA/ERL/National Severe Storms Laboratory

1. INTRODUCTION

As part of the NEXRAD product improvement effort [Saffle and Johnson, 1977], the current WSR-88D Radar Product Generator (RPG) software is being migrated to the open system computing environment. A substantial part of the RPG software consists of the product generation and meteorological algorithms, which was implemented through more than 50 tasks. These tasks, written in FORTRAN with CONCURRENT extensions and running under the CONCURRENT OS/32 operating system (OS), cannot be directly compiled and run under a UNIX OS. Inter-task communication is implemented through the use of shared memory segments, requiring all tasks to run on the same computer hardware. Tasks must fit into a predefined data flow network and are controlled by a central control module. This design makes the system sensitive to task failure and difficult to be dynamically reconfigured (redistributing the tasks among multiple hosts, adding new tasks, and so on).

The WSR-88D product generation and meteorological algorithm tasks contain sophisticated algorithms and lengthy code. Rewriting these tasks would require a substantial programming and verification effort and cause maintenance problems in the transition period when both current and the open system RPG versions have to be maintained.

On the other hand, these tasks use a similar coding structure and application programming interface (API) and require minimal system support, which makes direct porting feasible. After some path-finding work, it was decided to take a direct porting approach to migrate these tasks to the open system RPG. In this paper we describe the porting approach and discuss some of the related issues.

In the following we designate the existing RPG and the Open System RPG under development by RPG and ORPG respectively.

2. THE PORTING METHODOLOGY

The porting approach we adopted was designed to meet the following goals:

- The modification to the existing code should be minimized such that the amount of verification work for the ported code can be significantly reduced.
- The ported software should run in a distributed environment and fully support dynamic system reconfiguration. The ported software must be able to run together with any new ORPG product/algorithm tasks.

Corresponding author address: Zhongqi Jing, National Severe Storms Laboratory, 1313 Halley Circle, Norman, OK 73069-8480; e-mail <jing@nsslsun.nssl.uoknor.edu>

- Only a single version of the RPG source code needs to be maintained, which means that the ported code can still be compiled and run on the CONCURRENT OS/32 operating system.
- The ported source code can be compiled by any ANSI FORTRAN compiler and run on any POSIX compliant OS environment without substantial modification.
- The ported software should run efficiently in terms of the use of hardware resource.

To meet these requirements, it was decided to preserve the current RPG task partition. That is each current RPG task will be individually ported and compiled to generate a corresponding ORPG task. The inputs and outputs of each ported task will be identical to those of their RPG version.

The RPG tasks are controlled by a central control module. In the distributed ORPG, however, product/algorithm tasks are running in a data-driven environment [Jain, et. al., 1997]. This will require changes in the task control part of the code. We thus must write a new task control routine to replace the old one in the main function. With this exception, the remaining RPG modules are directly ported without or with minor modifications. By direct porting, we mean that the modules are simply copied over and compiled on an ORPG host.

The RPG FORTRAN code contains non-standard extensions that may not be supported by the FORTRAN compiler on an ORPG machine. A FORTRAN preprocessor can be developed to convert the existing code to an ANSI FORTRAN version which can be compiled by any FORTRAN compiler that supports the standard. The conversion is conducted at compile time and the converted version is discarded after compilation. Only the original RPG source code is maintained. The preprocessor can be ported to different platforms while the RPG code does not need to be modified.

To guarantee that the ported code can still be compiled by the CONCURRENT OS/32 FORTRAN compiler, any modification to the ported code must be transparent to the CONCURRENT compiler. This is done by defining special preprocessor directives that make the modifications appear as comment lines to the CONCURRENT compiler.

The RPG uses data buffers, allocated in shared memory and managed by a buffer control module, for passing radar data and products among tasks. The ORPG relies on similar data buffers, implemented by a data management tool called Linear Buffer (LB), to perform this task [Jain, et. al., 1997]. Functionally the LB is used in the same way as the RPG data buffers are used, which makes the porting easier. One of the differences however is that the LB supports distributed applications while the RPG data buffer mechanism does not. An LB is a self-controlled data store and communication object that does not require a control module to support it. This characteristics improves the robustness of the system. Moreover, each LB stores a sequence of products for a given product type. This provides the potential of dynamically relocating tasks and buffers without loosing data. For more

discussions on the Linear Buffer, refer to [Jain, et. al., 1997] .

Shared data, such as adaptation data, scan summary information, and product generation control information, are stored in shared common blocks in the RPG. In the ORPG, those are stored in corresponding LBs and can be accessed by all ORPG tasks. For the ported tasks, each of them will have their own common blocks allocated. A mechanism can be built to automatically update the local common blocks by reading data from the LBs. This mechanism can be transparent to the RPG tasks, requiring no modification to the ported code.

RPG tasks also use shared common blocks for passing data between tasks. These common blocks are identified as Inter-Task Common blocks (ITCs). This kind of interprocess communication must also be replaced by explicit message passing. Additional LBs are created for implementing the ITCs. Message passing via these LBs, when activated by product generation or data flow events, can be automatically processed without requiring any change to the RPG code. In other cases, function calls may be inserted into the RPG code to activate ITC reading and writing.

A library of functions, emulating the current RPG API, will be developed to provide data flow and processing control support as well as automatic update of local shared common blocks. By linking with these functions, changes to the ported code can be minimized and in most cases it can be eliminated. Most of the supporting functions may be conveniently written in C. However, calling conventions between C and FORTRAN functions are non-standard. To maximize portability, we must keep the set of necessary conventions to its minimum.

In addition, CONCURRENT OS/32 specific system functions must be implemented to support the ported code.

3. THE PORTING PROCEDURE

Porting an RPG task involves the following steps:

- Get a copy of all RPG modules required for building the task.
- Identify all input and output buffer types, adaptation blocks and other shared data blocks, all ITCs and the data flow timing of the task.
- Write a new main function which contains statements providing the information identified in the previous step to the supporting modules, followed by a main processing loop.
- Modify remaining RPG modules if necessary.
- Compile the code using the preprocessor and the local FORTRAN compiler and generate the executable by linking the compiled object files and the supporting libraries.

As an example, the following is the new main function of the 'hail" task that implements the hail algorithm:

```
      IMPLICIT NONE

$INCLUDE rpg_port.inc,**rpg_port

C adaptation common blocks
$INCLUDE a309adpt.inc,**A3CD70CA              ;SITEADP
$INCLUDE a309adpt.inc,**A3CD70C2              ;CP15ALG

$INCLUDE A309.INC/G,**A3PM00
$INCLUDE itc.inc,**A315TRND

      integer param                  ;return value from A31519 call

      ;** specify inputs and outputs
      call input_data (CENTATTR, VOLUME_DATA)
      call input_data (TRFRCATR, VOLUME_DATA)
      call output_data (TRENDATR, VOLUME_DATA)
      call output_data (HAILATTR, VOLUME_DATA)

      ;** register adaptation blocks
      call register_adpt (SITEADP, SITEADP_FIRST, BEGIN_VOLUME)
      call register_adpt (CP15ALG, CP15ALG_FIRST, BEGIN_VOLUME)

      ;register ITC inputs
      call itc_input (A315TRND, A315TRND_FIRST, A315TRND_LAST(2),
     1        TRFRCATR)

      call task_initialize (VOLUME_BASED)

      ;** The main loop, which will never end
10    call wait_for_activation (WAIT_ALL)
      call A31519__BUFFER_CONTROL(param)
      goto 10

      stop
      end
```

In the above code listing we can see that some of the CONCURRENT FORTRAN features are used: The inline comment and the $INCLUDE directive. Note that the RPG include files are used and the RPG functions are called. New include files are defined and new library function calls are added. The first part of the code initializes the supporting modules. It declares that this task has two inputs, CENTATTR and TRFRCATR, which are expected to be available on a volume basis. Calling subroutine input_data will cause the supporting modules to monitor these two input products and activate this task for generating its outputs when the inputs are ready. This task will generate two products called TRENDATR and HAILATTR. Both are volume based products. Because this task needs two adaptation data blocks, SITEADP and CP15ALG, subroutine register_adpt is called to tell the supporting modules that the two adaptation data common blocks need to be automatically updated at the beginning of every volume. The hail algorithm needs data provided in a common block called A315TRND. Since the data in this common block are generated by another task, we must define this common block as an ITC. By calling itc_input we tell the supporting modules that ITC A315TRND is used as an input and its contents need to be updated when input TRFRCATR is read in. The initialization is then completed by calling "task_initialize", which also specifies that this task processes data by volume.

Following the initialization is the main processing loop. In this loop, we first call wait_for_activation to pass program control to the supporting modules. When this task needs to be activated for processing (the outputs are scheduled and the inputs are available), wait_for_activation will return and we then call A31519__BUFFER_CONTROL to perform the processing tasks and generate the outputs. After A31519__BUFFER_CONTROL finishes, the control is passed to the supporting modules again through calling wait_for_activation. This loop will last until the task is terminated.

The new main function, called hail.ftn, is compiled, for example on an HP workstation, by commands

```
ftnpp -DHPUX -I/users/jing/rpg/include hail.ftn
fort77 -c  -K +U77 hail.f
```

where "ftnpp" is the preprocessor, "fort77" is the HP FORTRAN compiler and "hail.f" is the ANSI FORTRAN file generated by "ftnpp". For this task, there is no change needed for the remaining 9 ported RPG modules (a31509.ftn, a31519.ftn, etc.). These modules are then compiled in a similar fashion. And finally the executable "hail" is created by linking the object files and the

supporting libraries:

```
fort77 -o hail -K+U77 hail.o a31509.o a31519.o a31529.o a31539.o a31549.o
a31559.o a31569.o a31579.o a31599.o -L/users/jing/lib/hpux -lrpgcm -lrpg -llb
-lmisc -lm
```

where "rpgcm" is a library containing all RPG shared functions (A3CMnn.FTN), "rpg" is the library containing all RPG supporting modules, "lb" is the LB library and "misc" is a library containing some general ORPG utility routines, which are invoked by "rpg" and "lb".

4. THE PREPROCESSOR

The RPG FORTRAN code contains non-standard CONCURRENT extensions, which can not be processed by an ANSI FORTRAN compiler. Some of the extensions are useful for improving the code readability and maintainability. For example the $INCLUDE directive is very important in defining global constants and variables. We would like to keep taking advantage of these features. Moreover, we prefer, if possible, to maintain only a single version of the RPG code, which can be compiled on both an ORPG machine and the CONCURRENT machine. This will make the verification and future RPG version upgrades easier. Maintaining the same look and feel of the FORTRAN RPG code is also desirable for algorithm developers and maintainers. To reach these goals, a preprocessor has been developed for converting the CONCURRENT FORTRAN code to ANSI FORTRAN code at compile time.

The preprocessor reads the CONCURRENT FORTRAN code and generates the ANSI FORTRAN code for the ORPG compiler. The ANSI FORTRAN code is discarded after the code is successfully compiled. The preprocessor processes the following non-standard features:

- $INCLUDE files.
- CONCURRENT specific directives such as $INLINE, $INSKIP and $TCOM.
- The debugging print statement led by the "X" character.
- Non-standard hexadecimal constants of format Y'n' and Zn, where “n” is a hexadecimal constant.
- In-line comment proceeded by ";".

The preprocessor resolves $INCLUDE directives recursively and eliminates any duplicated inclusion. CONCURRENT specific directives such as $INLINE, $INSKIP and $TCOM are simply ignored. The block data files are processed as include files because the block data usage is not explicitly specified in the code, which can be a source of error. Other undocumented features that are not accepted by an ORPG compiler must also be processed.

The preprocessor supports special directives for making necessary changes in the ported code. They allow the modifications in the ported code to be transparent to the CONCURRENT compiler.

In certain cases an identifier needs to be replaced by a different one. For example, the RPG code may use an intrinsic function called "INT2" which is not supported by an ORPG compiler. We may then replace it by "INT". The preprocessor supports identifier replacement. Any preprocessor features that involve code changes, however, are used with caution because they can introduce inconsistencies between the ported and the original versions. They are only used when they are absolutely necessary.

The preprocessor also supports directives that allow compile time code segment selection similar to #indef ... #else ... #endif in the C preprocessor. This increases the portability of the source code.

5. THE SUPPORTING LIBRARY

A ported RPG task must be able to run in the ORPG environment. Specifically any interaction with the RPG monitor and control module and the buffer control module must be replaced by an interaction with the ORPG infrastructure. Note that in the ORPG, which uses an INTERNET/WEB model, a task can and must work without direct interaction with other ORPG tasks. For example, if a task needs adaptation data, it must explicitly read it from the LB storing these data. For a ported RPG product/algorithm task, it must read in the necessary adaptation data, the product generation control information and other system configuration and control information. It must also find the availability information of the products it wants to use as inputs. When the inputs are available, it must then read them in for processing. It must determine what to do when exceptional situations occur. For example, it must abort the unfinished processing if the volume scan is restarted or the task generating its input fails.

Fortunately all of these ORPG “burdens”, which are tradeoffs for distributed processing, greater flexibility and expand ability, as well as fault tolerance, can be performed automatically and hidden in supporting library functions.

To ease the porting effort and minimize changes to the RPG code, most of the functions, which the RPG product/algorithm tasks use to interact with the RPG control and monitoring module and the buffer control module, are emulated in the ORPG environment with identical API. This is essentially an implementation of the existing RPG API in the ORPG system.

Necessary CONCURRENT OS/32 specific system functions are emulated to support porting RPG product/algorithm tasks. The RPG API functions and the OS/32 functions are parts of the supporting library.

The following is a brief description of the functions in the supporting library.

- Input/output data type registration

The input and output data types of a task must be registered in the supporting modules. The information is used in processing control and input data synchronization. The ORPG uses a data driven model for product/algorithm task control. Each task is activated by a unique input called the driving input. Other inputs are used in synchronization with the driving input. The driving input is processed sequentially. If a discontinuity in the driving input is detected, a processing abort may result.

In ORPG, all RPG products as well as intermediate products are treated as ORPG products. They are identified by their RPG buffer type number. Each ORPG product is stored in an individual LB. The product time is used as the message ID in the LB for convenient product retrieval. Each ORPG product has an ORPG product header for additional control and data information.

- Adaptation data and the scan summary table

The RPG adaptation data is organized into several common

blocks, e.g. COLRTBL, RDACNT, CP13ALG and so on. The ORPG uses an LB to distribute the RPG adaptation data to tasks. The RPG scan summary table is implemented via an ITC for supporting ported RPG tasks. The supporting modules automatically update selected adaptation data blocks and the scan summary table for the ported tasks.

- Task initialization

Every task that uses the RPG supporting library must call task_initialize to initialize the supporting environment. This function also establishes the task type. The concept of task type is used in processing control. For example, if a task is of VOLUME_BASED type, it can not start processing data in the middle of a volume scan after task start up or a processing abort.

- Product generation control

The ORPG uses a product request LB to store product generation control information. For each product, a message containing a list of requests may be stored in the LB. Product request messages are identified by the product ID numbers. A request may specify an elevation at which the product is requested. It may also specify a user defined window if the product is window based. All products' generation can be controlled through requests and each product is controlled individually.

Subroutine wait_for_activation is designed for processing scheduling. wait_for_activation suspends the processing and keeps track of the input data flow and the user product requests. If an output is requested or scheduled to be generated and the required input data are ready, wait_for_activation will return and the product/algorithm task's main processing routine will then be activated. The processing routine reads the input by calling get_inbuf and processes the data until all output products are generated.

- Buffer control support

RPG tasks rely on buffer control functions to get inputs, generate outputs and allocate scratch working areas. The RPG buffer control functions, A31212__REL_INBUF, A31211__GET_INBUF, A31215__GET_OUTBUF and A31216__REL_OUTBUF are supported in the supporting library. These functions emulate their original buffer control functions with enhanced abort processing support.

- Inter-Task Common block (ITC) support

The RPG tasks use common blocks in global shared memory, called ITCs, to exchange data among tasks. In ORPG, ITC data are exchanged through LBs. When data in an ITC are ready, they are written out as a message to an LB and they are then read by other tasks that need the data. Each ITC is assigned a unique ITC ID number, which is used for identifying the LB that stores the message and the particular message in the LB. Multiple ITCs can be implemented with a single LB.

Functions itc_input and itc_output are used for informing the supporting modules that certain ITCs are used by the task and they need to be automatically updated or written out at scheduled times. Functions itc_read and itc_write provide explicit accesses to the ITC LB. In cases the scheduled automatic read-in/write-out is not sufficient, one can customize ITC reading/writing by inserting the itc_read/itc_write calls into the RPG code.

- RPG control and monitor function support

A product/algorithm processing procedure has to be aborted in several circumstances, which include elevation/volume scan restart, load shed and task failure. When the supporting modules detect an abort situation, get_inbuf will return with STATUS set to TERMINATE. This STATUS return will cause the task to terminate the current unfinished processing procedure, free allocated resources, clean up and set up the next processing resumption time.

- OS32 FORTRAN extension and system functions support

OS32 FORTRAN extension and OS32 system call functions must be emulated. Because the number of these functions is quite large and some of them may be difficult to emulate in a different OS, the approach we took is to implement those that are found to be necessary. Examples of supported functions include:

btest, bclr, bset, ilbyte and isbyte
lokon and lokoff
date, iclock and wait
sndmsg and queue
deflst, atl, abl, rtl, rbl and lstfun

6. SUMMARY

A direct porting approach to migrate the legacy WSR-88D algorithm and product generators to the Open Systems RPG is described. The approach implements a "true" porting of the RPG FORTRAN code to the Open Systems environment in the sense that most of the code is simply copied over, compiled and ready to run. Only a single version of the RPG code, which works on both the current legacy hardware and the distributed open system environment, needs to be maintained. Since the basic processing routines are not modified in the porting, a full scale verification of the ported code is not necessary. The ported product/algorithm tasks offer full advantage of the distributed processing and are able to run together with new ORPG product generators and meteorological algorithms.

7. ACKNOWLEDGMENTS

The authors would like to acknowledge the Office of Systems Development of the National Weather Service for funding this project.

8. REFERENCES

All references are to papers included in this preprint volume. Preprints 13th International Conference on Interactive Information and Processing Systems (IIPS) for Meteor., Ocean., and Hydro., Long Beach, CA, Amer. Meteor. Soc.