Portage of AL26T1_op4 on IBM Regatta p690 machine

.

François Thomas ft@fr.ibm.com
Revised and Approved by National Institute of Meteorology Tunisie

April, 5th, 2004

Part 7: Data assimilation, Variational computations

1. Summary

This document outlines the work performed at INM from February, 22nd to March, 2nd. The tasks that were performed during this stay are:

- **system tasks :** software upgrade (AIX, LoadLeveler, Parallel Environment, compilers), disk space management, installation of a web server, etc.
- **compilation and link of AL26T1_OP4**
- **unsuccessful attempt to compile ODB**
- **implementation** of a workload management strategy using LoadLeveler, WLM and vsrac, a tool developped by IBM Montpellier
- **test and implementation** of an asynchronous method for running a forecast with slightly outdated coupling files from Meteo France
- **start a port of Metview/Magics**

We will describe all these tasks and give four appendices regarding :

- **software management under AIX**
- **AIX system backup**
- **users management with AIX**
- **vsrac : task placement tool**

2. System tasks

2.1 Software upgrade

The first step of the work was to upgrade the system software to the latest levels. The table below summarizes the levels currently installed on the p690 Text (Body)

Software name	*Software name*
AIX	5.1 ML5 : Maintenance Level 5
Parallel Environment	3.2.0.17
Load Leveler	3.1.0.22
xlf compiler	8.1.1.4
ESSL library	3.3.0.6

The best way to get system software fixes is to use the IBM support web site : http://www-1.ibm.com/servers/eserver/support/pseries/fixes/. This service is free of charge. See Appendix A for details about the software management under AIX.

2.2 Disk space management

There are 3 different disk space classes at INM and 4 physical disks as seen by AIX.

- hdisk0+hdisk1 form the rootvg volume group that holds the operating system data. This volume group is mirrored, meaning every disk partition is doubled, providing maximum security to the system data.
- hdisk2 is made up of 4 physical disks in the FastT storage unit. This disk is the support for the uservg volume group that holds the home directories of the users as well as some fast storage for running forecast jobs. This disk has no redundancy and it is the users responsibility to backup the data until INM implements an automated backup strategy.

- hdisk3 is made up of 6 physical disks in the FastT storage unit. These disks are arranged in RAID5 mode which makes hdisk3 resilient to disk failures. This disk holds the chnvg volume group which contains all the operational data.

2.3 Installation of a web server

We have installed, configured and started a web server on the p690 to host the internal weather forecast web pages as well as the documentation for the major software products : LoadLeveler, Parallel Environment and the compilers. We have used the Apache software as found on the Linux Toolbox for AIX CDROM.

2.4 System startup scripts

We have added two stanzas in the /etc/inittab file to start LoadLeveler and our newly installed web server when the p690 boots. This is done by adding these two lines at the end of the file.

apache:2:once:/usr/local/bin/apachectl start > /dev/console 2>&1

loadleveler:2:once:/usr/bin/startll > /dev/console 2>&1

Where startll is a script containing the following line :

su -loadl -c llctl -g start

3. Compilation of AL26T1_OP4

3.1 Starting point

We used as a starting point the directory named cy26t1_op4_main.O1.AIX5.1 which was saved as .cy6t1_op4_main.O1.AIX5.1 for comparison. We set the OBJECT_MODE=64 environment variable in the user's profile so that all programs compiled will be 64bit programs.

The next step is to create the top make files : Makefile, Makefile.setp and Makefile.setptun for the tuned files. We also create the bin and lib directories. Then for each package, we create a Makefile and the expand and module directories. We use the Makefile structure from Josef Vivoda from SHMU. The addition we have made to the original Makefiles is to separate the compilation of the modules from that of the objects. This way we can take advantage of the parallel feature of the GNU make (gmake) for building the libraries.

We end up with this structure.

cy26t1_op4_main.O1.AIX5.1

Makefile

Makefile.setp

Makefile.setptun

bin/

lib/

xrd/

Makefile

tfl/

Makefile

arp/

Makefile

tal/

Makefile

ald/

Makefile

tun/

Makefile

Here is the top Makefile.

```
#-----------------------------------
#        Top level Makefile
#-----------------------------------
#     ALADIN benchmark ITT2003
#-----------------------------------
#    Author: JV, SHMU, 2003 (C)
#-----------------------------------
# include makefile with definitions of variables include Makefile.setp
# Ce Makefile permet de construire 2 cibles : tuned et untuned
# name of executable
# exe.t is tuned
# exe is asis
GMAKE   = gmake -j
EXEO =   bin/ALADIN.odb
EXET =   bin/ALADIN.exe.t
EXE   =   bin/ALADIN.exe
BATODB =       bin/BATODB.exe
all:$(EXE)
$(EXE) : lib/libxrd.a lib/libtfl.a lib/libarp.a lib/libtal.a lib/libald.a
  ar x lib/libarp.a master.o
  mpxlf90_r -o $@ master.o -Llib -lald -larp -ltal -ltfl -lsig -lxrd -llapack -lessl -lmassv -lmass
-brename:.flush,.flush_ -brename:.hostnm,.hostnm_ -qnoextchk -bloadmap:map
  rm -f master.o
$(EXET) : lib/libxrd.a lib/libtfl.a lib/libarp.a lib/libtal.a lib/libald.a lib/libtun.a
  ar x lib/libarp.a master.o
   mpxlf90_r -o $@ master.o -Llib -ltun -lald -larp -ltal -ltfl -lxrd -llapack -lessl -lmassv
-lmass -lsig -brename:.flush,.flush_ -brename:.hostnm,.hostnm_
  rm -f master.o
$(EXEO) : lib/libxrd.a lib/libtfl.a lib/libarp.a lib/libtal.a lib/libald.a lib/libodb.a lib/libcoh.a
lib/libost.a lib/libsat.a
  ar x lib/libarp.a master.o
   mpxlf90_r -o $@ master.o -Llib -lald -larp -ltal -ltfl -lodb -lcoh -lost -lsat -lxrd -llapack
-lessl -lmassv -lmass -lsig -brename:.flush,.flush_ -brename:.hostnm,.hostnm_ -bloadmap:map
  rm -f master.o
$(BATODB) : lib/libxrd.a lib/libtfl.a lib/libarp.a lib/libtal.a lib/libald.a lib/libodb.a
lib/libcoh.a lib/libost.a lib/libsat.a lib/libuti.a
  ar x lib/libuti.a batodb.o
   mpxlf90_r -o $@ batodb.o -Llib -luti -lald -larp -ltal -ltfl -lodb -lodbport -lcoh -lost -lsat
-lxrd -llapack -lessl -lmassv -lmass -lsig -brename:.flush,.flush_
    rm -f batodb.o
# here the EXECUTABLE shall be leaded
```

```
#
lib/libtun.a :
  cd tun ; $(MAKE) modules ; $(GMAKE) objects ; $(MAKE)
lib/libxrd.a :
  cd xrd ; $(MAKE) modules ; $(GMAKE) objects ; $(MAKE)
lib/libtfl.a :
  cd tfl ; $(MAKE) modules ; $(GMAKE) objects ; $(MAKE)
lib/libarp.a :
  cd arp ; $(MAKE) modules ; $(GMAKE) objects ; $(MAKE)
lib/libtal.a :
  cd tal ; $(MAKE) modules ; $(GMAKE) objects ; $(MAKE)
lib/libald.a :
  cd ald ; $(MAKE) modules ; $(GMAKE) objects ; $(MAKE)
lib/libodb.a :
  cd odb ; $(MAKE) modules ; $(GMAKE) objects ; $(MAKE)
lib/libcoh.a :
  cd coh ; $(MAKE) modules ; $(GMAKE) objects ; $(MAKE)
lib/libost.a :
    cd ost ; $(MAKE) modules ; $(GMAKE) objects ; $(MAKE)
clean:
  rm bin/ALADIN.exe
cleanall:
   (cd lib ; rm -f libald.a libtal.a libarp.a libtfl.a libtun.a libxrd.a libodb.a libcoh.a libost.a
libsat.o)
  (cd ald ; $(MAKE) cleanall)
  (cd arp ; $(MAKE) cleanall)
  (cd tal ; $(MAKE) cleanall)
  (cd tfl ; $(MAKE) cleanall)
  (cd xrd ; $(MAKE) cleanall)
  (cd odb ; $(MAKE) cleanall)
  (cd coh ; $(MAKE) cleanall)
  (cd ost ; $(MAKE) cleanall)
  (cd sat ; $(MAKE) cleanall)
  (cd tun ; $(MAKE) cleanall)
```

This is the basic Makefile.setp that sets the regular compilation flags

```
#-----------------------------------
# Makefiles with compilation config
#-----------------------------------
#    ALADIN benchmark ITT2003
#-----------------------------------
#   Author: JV, SHMU, 2003 (C)
#-----------------------------------
```

```
# path to ALADIN sources under which are xrd, tfl, arp, tal and ald directories
SRCPATH   = /chaine/cycles/cy26t1_op4_main.01.AIX5.1
# include directories (headers) for preprocessor
CPPINCLUDE =    \
-I$(SRCPATH)/tal/interface \
-I$(SRCPATH)/arp/namelist \
-I$(SRCPATH)/arp/interface \
-I$(SRCPATH)/arp/ald_inc/namelist \
-I$(SRCPATH)/arp/ald_inc/interface \
-I$(SRCPATH)/arp/ald_inc/function \
-I$(SRCPATH)/arp/odb-include \
-I$(SRCPATH)/arp/function \
-I$(SRCPATH)/arp/common \
-I$(SRCPATH)/tfl/interface \
-I$(SRCPATH)/xrd/utilities/include \
-I$(SRCPATH)/xrd/include \
-I$(SRCPATH)/xrd/grib_io/include \
-I$(SRCPATH)/xrd/model_io/include \
-I$(SRCPATH)/xrd/mpe/include \
-I$(SRCPATH)/xrd/lfi \
-I$(SRCPATH)/xrd/fa \
-I$(SRCPATH)/odb/include \
-I$(SRCPATH)/odb/interface \
-I$(SRCPATH)/odb/misc \
-I$(SRCPATH)/coh/include \
-I$(SRCPATH)/coh/common \
-I$(SRCPATH)/ost/include \
-I$(SRCPATH)/ost/interface \
-I$(SRCPATH)/ost/namelist \
-I$(SRCPATH)/ost/common \
-I$(SRCPATH)/sat/include \
-I$(SRCPATH)/coh/namelist
# include directories (modules + headers)
INCLUDE =       \
-I$(SRCPATH)/ald/module \
-I$(SRCPATH)/tal/module \
-I$(SRCPATH)/arp/module \
-I$(SRCPATH)/tfl/module \
-I$(SRCPATH)/xrd/module \
-I$(SRCPATH)/odb/module \
-I$(SRCPATH)/odb/misc \
-I$(SRCPATH)/coh/module \
```

```
-I$(SRCPATH)/ost/module \
-I$(SRCPATH)/sat/module $(CPPINCLUDE)
# preprocessor directives
CPPFLAGS  =  -DRS6K -DALLDBL -DADDRESS64 -DMPI -DBLAS
FCPPFLAGS  =  -DRS6K,-DALLDBL,-DADDRESS64,-DMPI,-DBLAS
QSTRICT=
QSTRICT=-qstrict
QSAVE=-qsave
QSAVE=
OPT=-qmaxmem=-1 -qspillsize=32768
OPT=-O3
# f90 compiler (system dependent compiler switches)
# first precprocessor is called (-Ep) with CPPFLAGS and then
# compilation is performed using preprocessed source
F77  =  mpxlf_r
F90  =  mpxlf90_r
FFLAGS  =  -WF,$(FCPPFLAGS)  $(INCLUDE)  $(OPT)  $(QSTRICT)  $(QSAVE)
-qarch=auto -qalias=noaryovrlp -qnoextchk -g
# options without amutomatic promotion
# tfl ,arp ,tal and ald libraries are compiled using these switches
FSPFLAG = $(OPT) $(QSTRICT) $(QSAVE) -qarch=auto -qalias=noaryovrlp -qnoextchk -g
# options with automatic promotion R4 -> R8 for xrd library (-A dbl4)
# xrd library is compiled using these switches
FDPFLAG  =  -qautodbl=dbl4  $(O3)  $(QSTRICT)  $(QSAVE)  -qarch=auto
-qalias=noaryovrlp -qnoextchk -g
# c/c++ compiler
CC  =  mpcc_r
CFLAGS  =  $(CPPFLAGS) $(CPPINCLUDE) -O3 -qstrict -qarch=auto
# shared libraries loading
BUILD =  ar
BDFLAGS  =  -ruv
This is the Makefile.setptun for tuned routines with higher optimization flags.
#-----------------------------------
# Makefiles with compilation config
#-----------------------------------
#    ALADIN benchmark ITT2003
#-----------------------------------
#   Author: JV, SHMU, 2003 (C)
#-----------------------------------
# path to ALADIN sources under which are xrd, tfl, arp, tal and ald directories
SRCPATH  = /chaine/cycles/cy26t1_op4_main.01.AIX5.1
# include directories (headers) for preprocessor
```

```
CPPINCLUDE =    \
-I$(SRCPATH)/tal/interface \
-I$(SRCPATH)/arp/namelist \
-I$(SRCPATH)/arp/interface \
-I$(SRCPATH)/arp/ald_inc/namelist \
-I$(SRCPATH)/arp/ald_inc/interface \
-I$(SRCPATH)/arp/ald_inc/function \
-I$(SRCPATH)/arp/odb-include \
-I$(SRCPATH)/arp/function \
-I$(SRCPATH)/tfl/interface \
-I$(SRCPATH)/xrd/utilities/include \
-I$(SRCPATH)/xrd/include \
-I$(SRCPATH)/xrd/grib_io/include \
-I$(SRCPATH)/xrd/model_io/include \
-I$(SRCPATH)/xrd/mpe/include \
-I$(SRCPATH)/xrd/lfi \
-I$(SRCPATH)/xrd/fa \
-I$(SRCPATH)/odb/include \
-I$(SRCPATH)/odb/interface \
-I$(SRCPATH)/odb/misc \
-I$(SRCPATH)/coh/include \
-I$(SRCPATH)/coh/common \
-I$(SRCPATH)/ost/include \
-I$(SRCPATH)/ost/interface \
-I$(SRCPATH)/ost/namelist \
-I$(SRCPATH)/ost/common \
-I$(SRCPATH)/sat/include \
-I$(SRCPATH)/coh/namelist
# include directories (modules + headers)
INCLUDE =       \
-I$(SRCPATH)/ald/module \
-I$(SRCPATH)/tal/module \
-I$(SRCPATH)/arp/module \
-I$(SRCPATH)/tfl/module \
-I$(SRCPATH)/xrd/module \
-I$(SRCPATH)/odb/module \
-I$(SRCPATH)/odb/misc \
-I$(SRCPATH)/coh/module \
-I$(SRCPATH)/ost/module \
-I$(SRCPATH)/sat/module $(CPPINCLUDE)
# preprocessor directives
CPPFLAGS   =   -DRS6K -DALLDBL -DADDRESS64 -DMPI -DBLAS
```

```
FCPPFLAGS  = -DRS6K,-DALLDBL,-DADDRESS64,-DMPI,-DBLAS
# f90 compiler (system dependent compiler switches)
# first precprocessor is called (-Ep) with CPPFLAGS and then
# compilation is performed using preprocessed source
F90  =  mpxlf90_r
FFLAGS = -WF,$(FCPPFLAGS) $(INCLUDE) -O3 -qhot -qarch=auto -qalias=noaryovrlp -qnoextchk
# options without amutomatic promotion
# tfl ,arp ,tal and ald libraries are compiled using these switches
FSPFLAG = -O3 -qarch=auto -qalias=noaryovrlp -qnoextchk
# options with automatic promotion R4 -> R8 for xrd library (-A dbl4)
# xrd library is compiled using these switches
FDPFLAG =      -qautodbl=dbl4 -O3 -qhot -qarch=auto -qalias=noaryovrlp -qnoextchk
# c/c++ compiler
CC =  mpcc_r
CFLAGS  =  $(CPPFLAGS) $(CPPINCLUDE) -O3 -qstrict -qarch=auto
# shared libraries loading
BUILD =  ar
BDFLAGS    =    -ruv
```

3.2 xrd compilation

We edit the xrd/not_used/second.F file to fix the call to rtc() whose return value should not be multiplied by the 4.167E3 factor.

One of the difficulties with xrd compilation is the inter language convention between C and Fortran. By default, and unlike many other Fortran compilers, the xlf compiler generates symbol names without a trailing underscore so that a Fortran program can call a C routine with no change. However, in case the C routine has a trailing underscore in its name, the -qextname option can be used. To support the -qextname option, most libraries have both the original and the underscore symbol. This is the case for the Fortran library, the MPI libraries and the MASS library. This is not the case for the LAPACK library that is installed in INM (and in SHMU, and OMS, and Meteo Marocco). It appears that, although the IBM convention of keeping the same symbol names between C and Fortran is simpler, ALADIN in many places assumes that the Fortran compiler adds trailing underscores. This forces us to alter the C files and add sections like.

```
#ifdef RS6K
#else if defined(CRAY)
#define symbol SYMBOL
#else
#define symbol symbol_
#endif
```

This has become so much common place that ECMWF decided to use the -qextname option to compile Fortran programs and treats therefore the Fortran compiler on IBM pSeries as ordinary underscore type compiler. The only library that is missing this feature is the LAPACK library which I have now rebuilt to include both symbols. I tried to resist the underscore trend but this now requires too many changes.

The version of AL26T1 is compiled with the no trailing underscore convention but this is probably the last one. I will now use the -qextname option and the new LAPACK library.

To diagnose differences between the argument list in the caller routine and the called routine, we used the -qextchk compilation and link flag. This option can be turned on to list all routines which need attention but must be switched off after as a single difference in the argument list will prevent the link from being performed, even though the difference is “safe”, like passing an array when only a scalar value is expected for example.

Our first attempt at compiling XRD failed due to a lack of space for temporary files. This is classical and usually means the TMPDIR environment variable is not set properly. The compiler writes files to /tmp unless TMPDIR is set in which case it must point to a writable directory.

IMPORTANT : make sure you remove the xrd/mpe/include/mpif.h file as it contains MPI definitions and constants for Fujitsu ! They have different values on IBM and as we use of the mpxlf_r, mpxlf90_r and mpcc_r compilers in the Makefile.setp files, we are sure to pick the right include file for mpif.h.

We found some duplicate sources for some files. For example, xrd/utilities/{expand21,pack21}.F and xrd/utilities/{expand21,pack21}.s exist and the .s version should be removed to pick up the Fortran version. Xrd/grib_mf/gsbite_mf.F and gsbite_mf.c exist. We use the Fortran version again and rename the C version to something that the Makefile will ignore.

3.3 tfl compilation

Nothing to say

3.4 arp compilation

We found one problem with arp/phys_dmn/suphy2.F90 where a statement

```
WRITE(UNIT=KULOUT,*) 'XMULAF SHOULD BE NEGATIVE'
```

needs to be changed to

```
WRITE(UNIT=KULOUT,FMT=*) 'XMULAF SHOULD BE NEGATIVE'
```

We also found when first compiling that some include files, brought by the ost package where needed. (int_setparam_obsort.h and boot_setparam_obsort.h). This is now fixed by adding the ost/interface directory in the include search path.

We also found in arp/control/cprep1.F90 that the min intrinsic routine is called with arguments of conflicting types. This is the case for statements like this.

```
XX=MIN(YY,1.0)
```

where XX and YY are REAL*8 and 1.0 is implicitely a REAL*4 value. We solve this using the following notation:

```
XX=MIN(YY,1.0_JPRB)
```

3.5 suspeca bug ! Beware !

This bug keeps creeping in the ALADIN versions. We found it once again in AL26T1 by comparing the spectral norms between Fujitsu and IBM. To get rid of this bug, we must declare the PSPVOR, PSPVB etc arrays as PSPVOR(:,:) in arp/interface/suspec.h, arp/interface/suspeca.h and arp/setup/suspeca.F90.

3.6 tal compilation

Nothing to say.

3.7 ald compilation

ald/pp_obs/sufpmove.F90 contains some doubtful calls to BROADCINT where we think they should be calls to BROADCREAL.

3.8 libsig compilation

Our first attempt to link the application shows that we are missing the SIG library. We build it under ~/util/sig by editing the fortopts file, included by the compile script, to set the path, the

compiler and compiler flags. We then run the compile and buildlib scripts.

3.9 Dummies

We create under xrd/support a file called dummy.F90 and one called dummy_odb.F90 to hold unsatisfied references.

In arp/c9xx/incli0.F90, we comment out the call to EINCLI8 and write a message to standard output in case we reach this section.

3.10 argument mismatch

Using the -qextchk argument, we found a lot of places where routines are called with varying number and types of arguments. Sometimes, this is harmless and sometimes this could create problems. For all cases where the number of argument would differ we print a warning message on standard error.

We make a list here of the mismatches. To keep a record of these, use the -bloadmap: themapfile option to create a link map file called themapfile.

- esc2rad is called with varying number of arguments in ald/coupling/elsto1ad.F90, ald/adiab/espayad.F90 and esc2rad.F90.
- esc2r is called with varying number of arguments in ald/inidata/esc2r.F90, ald/adiab/espay.F90, arp/adiab/gptenc.F90, ald/var/ewrlsgrad.F90.
- eggx_n defined in ald/utility/eggx_n.F90 is called from arp/setup/sufpd.F90, arp/utility/echien.F90 and arp/setup/sufpg2.F90 with minor differences in the constants which are REAL*8 and should be coded as 0._JPRB for correctness.
- abor1 is called in many places without arguments when its definition contains one argument.
- arp/pp_obs/fpachmt.F90 calls ACCLPH with 18 arguments instead of 16. We issue a message on standard error.
- arp/var/sujbwavgen.F90 calls BALNONLIN with 3 arguments instead of 2. We issue a message on standard error.
- arp/sinvect/opk.F90 calls CVARU3I and CVARU3IAD without arguments whereas one is required. We issue a message on standard error.
- arp/ocean/wrcodm.F90 calls DICOMOUT with 6 arguments instead of 7. We issue a message on standard error.
- ald/coupling/ebipaux.F90 should pass 0._JPRB as the PALFA argument to ESPLIN instead of '0'.
- arp/utility/fspglhlag.F90 calls INI2SCAN2M with 19 arguments instead of 21.
- ald/control/cnt4ad.F90 has a funny call to PPREQ with arguments of types (integer,logical,REAL_B) when we expect (integer,logical,CHAR*120)...
- arp/phys_dmn/acradin.F90 calls RADDIAG with 35 arguments instead of 40 and RADINT with 42 arguments instead of 50.

arp/phys_dmn/aplpa.F90 calls RADHEAT with 28 arguments instead of 33.

4. Execution of ALADIN

4.1 NPRTRV and NPRTRW are mandatory

It took us very long time to figure out why a test case would run on 1, 2, 3 processors and would fail on 4 and 8. We changed a lot of software (compilers, MPI library, AIX), tried hard to debug to finally find out that this was SUEMP that was creating the problem as it was creating a processor decomposition where NPRTRV would be something else than 1. The best way to solve the problem is to force NPRTRV=1 and NPRTRW=NPROC in the NAMPAR0 namelist.

4.2 Namelist changes

The namelist values that affect the performance of the ALADIN code are given below.

```
&NAMPAR0
  LMPOFF=.FALSE.,
  NOUTPUT=1,
  NPROC=NPROCG,
  NPROCA=NPROCG,
  NPROCB=1,
  NPRTRW=NPROCG,
  NPRTRV=1,
  MP_TYPE=2,
  MBX_SIZE=64000000,
  LIMP=.TRUE.,
  LIMP_NOOLAP=.TRUE.,
/
&NAMPAR1
  LSPLIT=.FALSE.,
  NSTRIN=NPROCG,
  NSTROUT=NPROCG,
  NCOMBFLEN=64000000,
  LSLONDEM=.TRUE.,
/
&NAMDIM
  NPROMA=-17,
```

5. Unsuccessful attempt at compiling ODB

As the title suggest we tried to compile ODB but with no success. This was not the first priority so we did not pursue too long. The good think about ODB is that it forced us to compile the COH, OST, SAT and UTI libraries that revealed some portability problems.

6. Workload management strategy

More important was the design and implementation of a workload management strategy. The solution that we have used in based on LoadLeveler, WLM (WorkLoad Management from AIX) and vsrac, a tool developped by IBM Montpellier. It is worth describing WLM here. Vsrac is described in great details in Appendix D.WLM Text (Body)

6.1 WLM description

WLM is a component of AIX that can be used (it is not activated by default) to manage access to resources (CPU, memory and I/O bandwidth) by the different processes running on the system. The processes are not managed one by one but rather grouped in classes that are defined by the AIX system administrator. A typical WLM configuration may classify the processes in less than 10 classes. A class can be for example : Interactive, Database, WebServing, etc, corresponding to the various possible uses of the system resources. In our case, the classes will rather by System, Development, Operational, ClimateModelling, Research, etc. The purpose of WLM is to make sure each usage of the system, each WLM class, gets the right share of the system resources.

WLM has two parts : a classification part and a resource entitlement part.

6.2 WLM classification

Every running process is classified by WLM according to rules. What is discriminant for WLM can be the process owner's userid or groupid, the name (path in fact) of the application or a tag. For example, we can decide that all processes belonging to the aladin userid will be classified in the "Operational" WLM class. There are default classes in which processes will fall if no other rule has been satisfied. To verify which WLM class a process belongs to, use the following ps command.

ps -e -o class,pid,command,user Text (Body)

6.3 Resource entitlement

Once WLM is started, all the processes are classified according to the rules set by the AIX administrator. Now, WLM will make sure that each class gets the share of resources that it is entitled to. There are 3 ways of managing the resources between classes:

- – shares
- – limits
- tiers

6.4 Tiers

This is a very strong way to assign resources to WLM classes. Each class is assigned a Tier from 0 to 9. All the processes of a lower tier have priority over processes of higher tiers. In other word, if there are enough processes of a lower tier to fill up a system, then, no process of a class with a higher tier will ever get the CPU. We have tried this first, assigning the "Operational" WLM class to Tier0 and the "Interactive" class to Tier9. However, this is too "strong" in the sense that no single shell command will run while an "Operational" job is running.

6.5 Limits

We can set hard limits to the resources usage. For example, we can specify that the "Interactive" class will not get more than 5% of the total resources and that the "Operational" class will not get more than 95%. However, if no interactive work is running while the Operational job runs, the 95% limit will still be in effect, leading to some resources waste.

6.6 Shares

The current configuration uses shares to make sure the operational job gets almost all system resources when it starts. We assign the following shares:

- Interactive : 50
- Development : 100
- Climate modelling : 50
- Operational: 2000

The way this works is : we sum up the number of shares of each class that is running and the resource are assigned to the each WLM class based on its individual share. For example, suppose we have development and interactive work running at time t=t0. The total number of shares is 50+100 = 150 shares. In this case the "Development" class will get 67% (100/150) of the total resources and the "Interactive" class will get 33% (50/150). Now, an "Operational" job starts. The total number of shares becomes 2000+100+50=2150. This time, and for the duration of the "Operational" job, the interactive workload will get 2% (50/2150), the development one 5%, leaving 93% to the operational job. In effect, this almost freezes all the workloads except the Operational one.

6.7 WLM adjustment

Of course, getting the right WLM configuration is an iterative process. There are many other

possibilities. For example, the WLM shares can be changed depending on the time of the day : favoring interactive work during work hours and favoring the development work during the night time and week-ends.

6.8 WLM tutorial

There are a few commands to control WLM. Here they are.

Action	*Command*
Start WLM	# wlmcntrl -a -d vsrac (to activate the vsrac WLM config)
Stop WLM	# wlmcntrl -o
Query the WLM status	# wlmcntrl -q
Update the WLM config	# wlmcntrl -u (does not stop WLM)
Configuration files	# /etc/wlm/vsrac/ files classes, shares, limits and rules

7. CHADA : asynchonous transfer of Meteo France files

We have ported the CHADA programs that is used to change the date of coupling files so that slightly obsolete coupling files can still be used for the forecast if the newest files have not been downloaded.

8. Appendix A : Software installation under AIX

We give here a few tips for querying software levels, installing new software or upgrading existing software under AIX version 5.

AIX version 5 recognizes three different types of packages :

- bff or installp : the traditional AIX format, bff stands for binary format file.
- rpm : RedHat package management, well known under Linux. Most of the Open Source utilities for AIX are distributed as rpms on the “Linux Toolbox for AIX” CD.
- ismp : Windows type install format.

Only the first two types are described below.

9. Installp format

Here are the main installp commands.

Action	*Command*
Query all installed software	# lslpp -
Query the operating system level	# oslevel -r
List all files belonging to a software	# lslpp -f <softwarename>
Which software does this installed file belong to ?	# lslpp -w /full/path/to/file
Which files does this installable contain ?	# restore -Tvf <bfffile>

Software numbering under AIX obeys specific rules. A software version number has 4 fields called V-R-M-F : Version-Release-Maintenance-Fix. For example in this display :

```
node8:root}/tmp ->lslpp -l xlfcmp
  Fileset                  Level  State      Description
  ----------------------------------------------------------------------
```

Path: /usr/lib/objrepos

xlfcmp 8.1.1.2 COMMITTED XL Fortran Compiler

Path: /etc/objrepos

xlfcmp 8.1.1.2 COMMITTED XL Fortran Compiler

we see that the Fortran compiler that is currently installed is 8.1.1.2. Version 8.1.1.0 is called the base level. In simple terms, the customer pays for the base level and can obtain the fixes (F>0) with no charge from IBM support web sites. Under AIX, applying fixes is often called applying PTFs, for Program Temporary Fixes. AIX provides much flexibility for fixes (PTFs) management. A fix can be applied (the new files are in effect but the other ones are still there) or committed (the new files are in effect and the old ones are removed). It's always a good practice to apply but not commit the fixes so that we can always revert to previous levels in case something goes wrong.

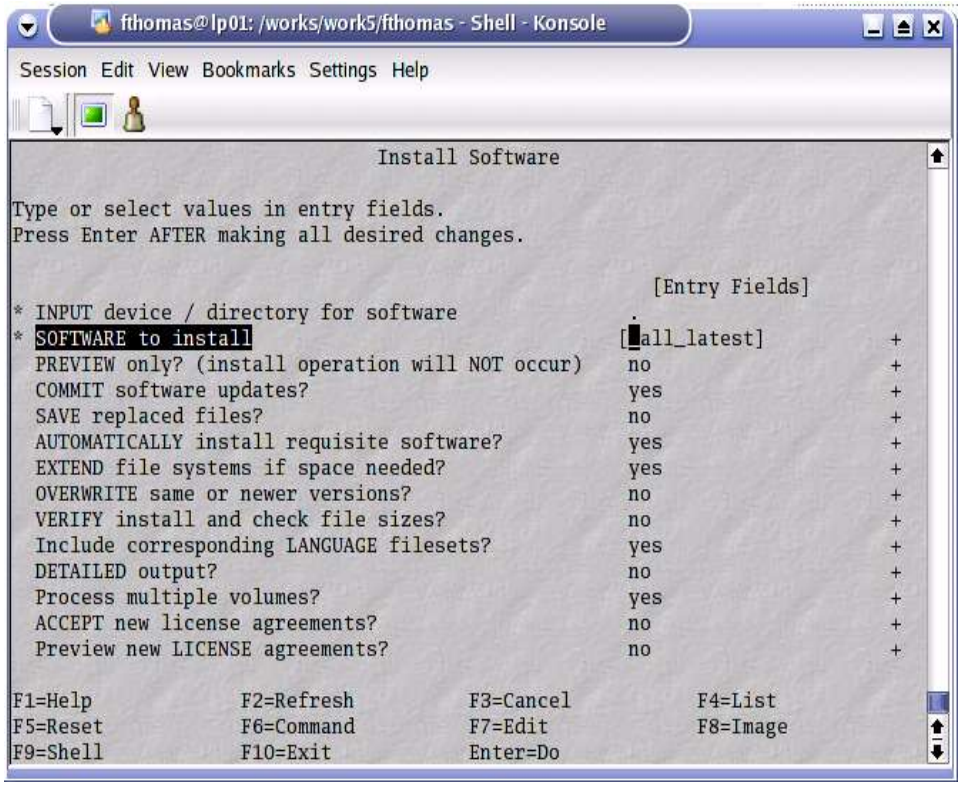
The preferred way of managing software is through smitty. A common situation is : the desired fixes have been downloaded in a directory ready to be installed. The first step is to create a table of contents of the software products in this directory. This is done using the inutoc command:

inutoc .

This command reads all the files in the directory, decides if some are of bff format and then builds a .toc file describing the software products. The smitty fastpath to software management can then be invoked :

smitty installp, then Install Software, then specify '.' as the input device/directory.

You will then choose which of the available software you wish to install, whether or not you are going to apply or commit the fixes and a few other choices.



10. RPM format

Here are the main RPM commands.

Action	*Command*
Install a RPM	# rpm -i <package>-<version>-aix4.3.ppc.rpm
Upgrade a RPM (installs if not already installed)	# rpm -Uvh <package>-<version>-aix4.3.ppc.rpm
List a package requirements	# rpm -q –requires <package> if installed # rpm -q –requires <package>-<version>-aix4.3.ppc.rpm if not
Force install a package, ignoring dependencies	# rpm -i –force –nodeps <package>-<version>-aix4.3.ppc.rpm
Uninstall a RPM	# rpm -e <package> (use the package name, not the name of a file)
Query all packages	# rpm -qa
Query a package	# rpm -q <package>
What does this package do ?	# rpm -qi <package>
What does this package, not yet installed, do ?	# rpm -qip <package>-<version>-aix4.3.ppc.rpm
What package does this file belong to ?	# rpm -qf /full/path/to/file
Which files are brought by this package ?	# rpm -ql <package>
Which files are brought by this not yet installed package	# rpm -qlp <package>-<version>-aix4.3.ppc.rpm
Help about rpm	# rpm # rpm --help

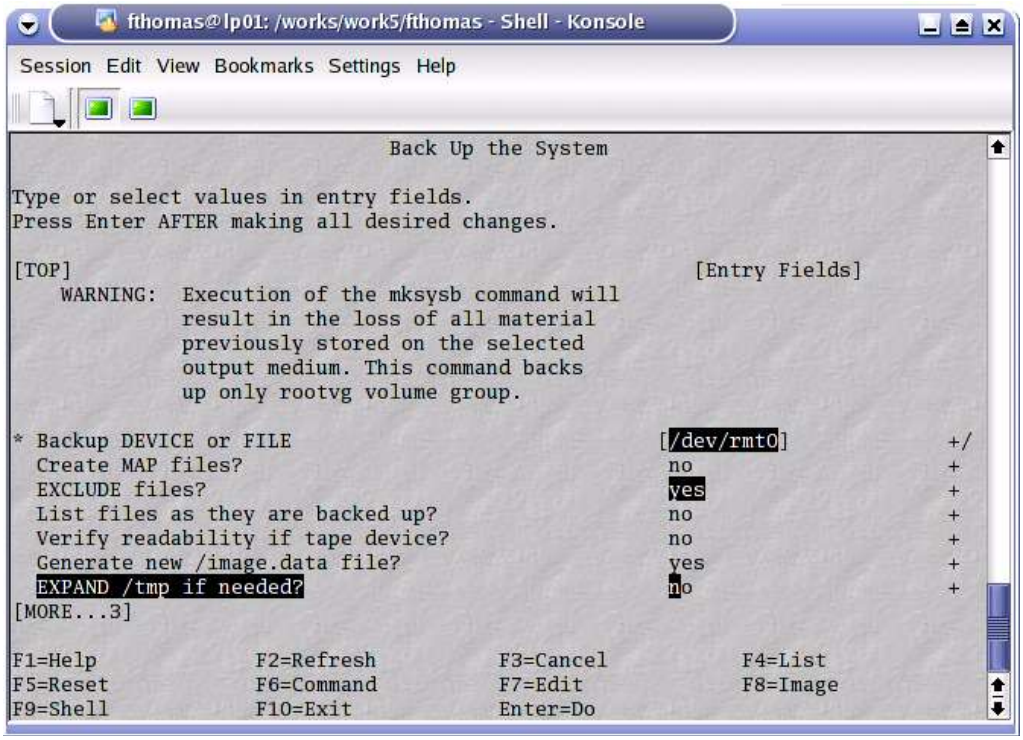
11. Appendix B : System backup

AIX provides a way to create a full system backup that can be reinstalled if needed. This type of backup is often referred to as a mksysb, named after the command mksysb : make system backup. A mksysb can be used for reinstalling AIX either on the originating machine or on another system, which will become a “clone” of the system where the mksysb was taken. A backup can reside on a tape, a DVD or on disk. A tape or DVD system backup is bootable and can be used directly to restore a system. A disk mksysb does not contain all the information and requires a form of network installation called NIM (Network Installation Manager). A full mksysb consists in four files

- a kernel
- bosinst.data : Basic Operating System installation data. This file describes what to do upon installation : overwrite or preserve existing installation, whether to install software bundles, interactive or batch installation, etc.
- image.data : describes the layout of the file systems when restoring the system.
- backup : contains all the installed file in a backup format.

To create a bootable system backup, enter :

smitty mksysb



Choosing a tape device will make the backup bootable. You can also choose a plain file but be careful, not to save the backup file in the backup itself. If you choose to save to a file, use a file in a filesystem that does not belong to the rootvg volume group or use the EXCLUDE files? option below. mksysb only saves rootvg, not the other volume groups. To list the filesystems that belong to all your volume groups, use:

$ for vg in $(lsvg)

do

- echo $vg
- lsvg -l $vg

done

You may wish to exclude some unnecessary files from the backup to make it smaller. This is the purpose of the option “EXCLUDE files?”. If you say yes, which is not at all necessary, you must create a file called /etc/exclude.rootvg that describes which files should not be backup up. Be careful that the syntax of this exclude.rootvg file is such that it is used in a grep command. For example, if you have a /tmp/data directory that contains huge files that you do not wish to have in the mksysb file, and if you are saving the mksysb file under /tmp/mksysb.040404, use the following syntax for /etc/exclude.rootvg.

cat /etc/exclude.rootvg

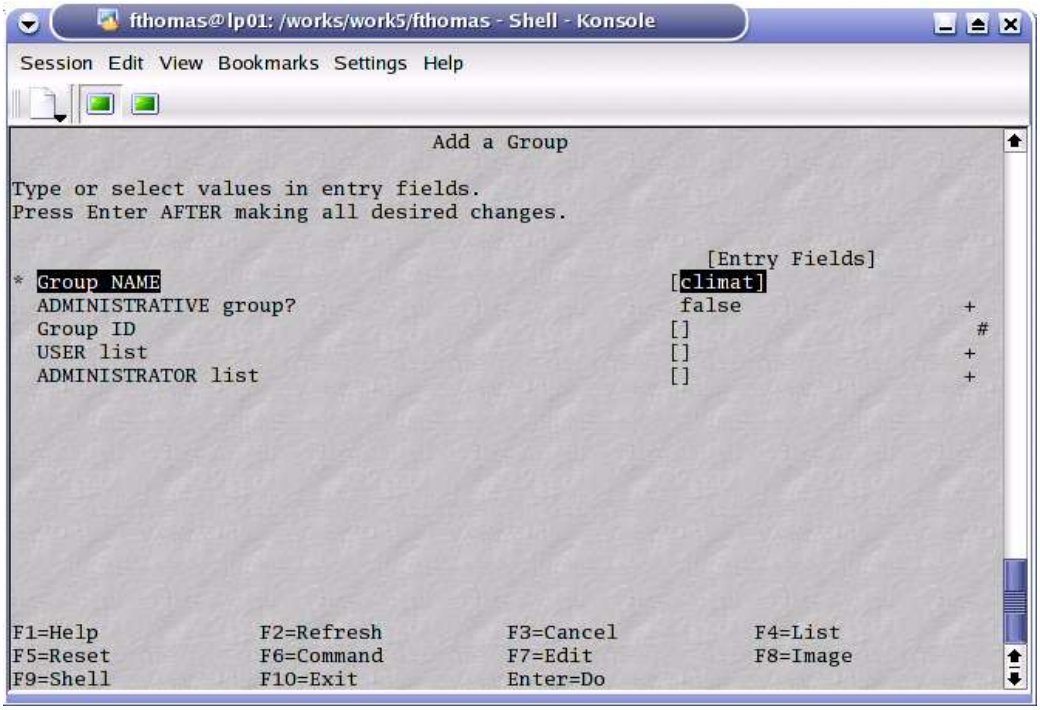
^.\/tmp\/data\/

^.\/tmp/mksysb.040404

12. Appendix C : Users management

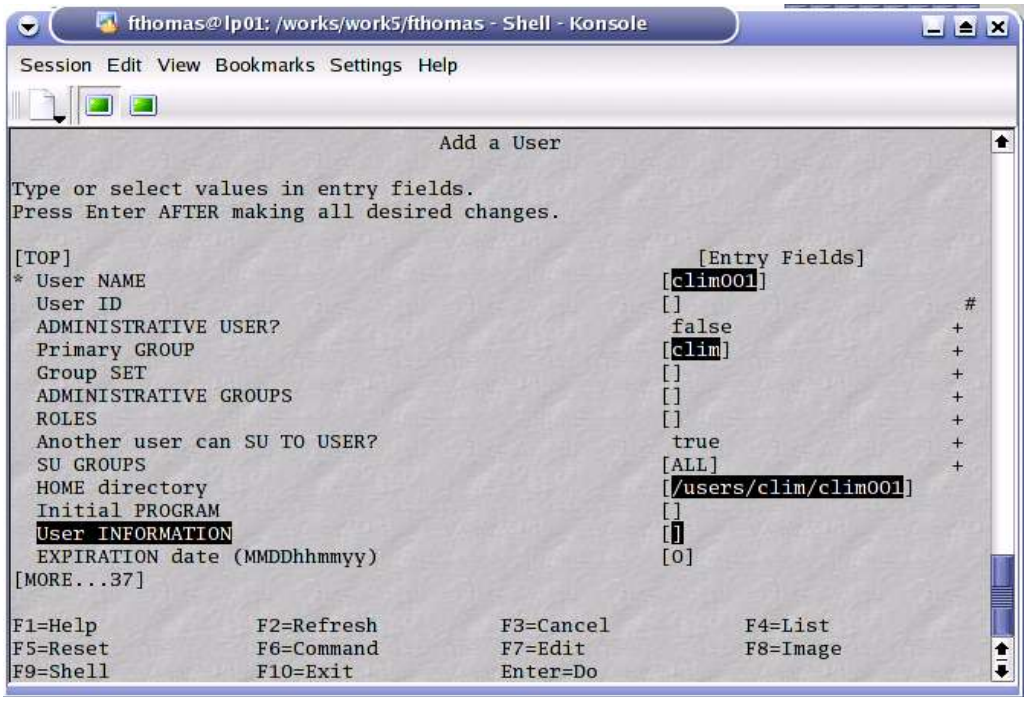
We give here a few tips for managing the groups and users under AIX. If you wish to create a new user, you must make sure that the group to which you want this user to belong exists. If not, first create the group with:

smitty group (then Add a group)

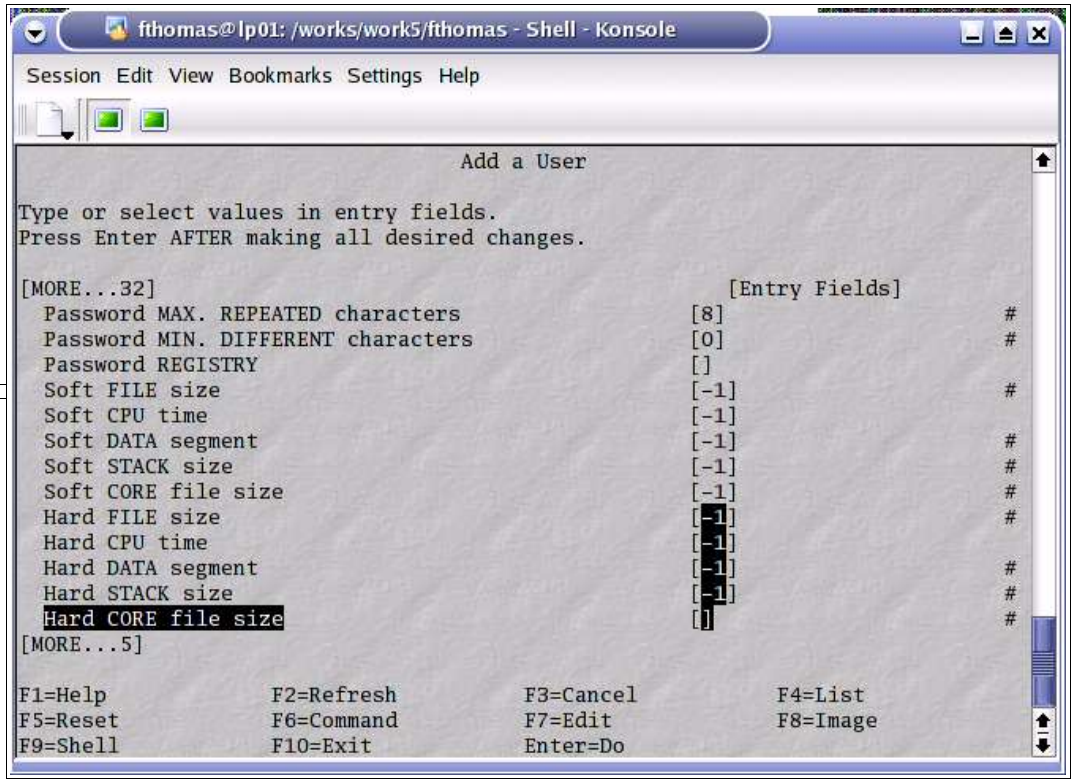


Then create the user with :

smitty users (Add a user)



Fill at least the userid (clim001), the primary group (clim), the home directory (/users/clim/clim001) and also the limits for this userid. It is advisable to set the memory limits to unlimited and use LoadLeveler to do the limits management later on. Scroll down the screen until you reach the limits section as shown below.



Once the user is created, you must set a password either with the passwd command or through smitty passwd. Once the password has been set, if you do not require the user to change his password when he first logins, you must run the following command:

pwdadm -c clim001

This command clears the ADMCHG flag in /etc/security/passwd which would otherwise force the user to change his password during the initial login.

A contrario, if your security policy requires that users change their password, you could issue:

pwdadm clim001

Allowing alternate shells for users

The default shell for AIX is ksh : Korn shell. If you wish to enable another login shell, for example /bin/bash for your users, you must have the required shell listed in the shells section of the /etc/security/login.cfg when you create the user. The shell is the “Initial program” field of the smitty users panel.

13. Appendix D : vsrac

Vsrac is a software tool developped by IBM Montpellier which interfaces between AIX, LoadLeveler and Parallel Environment and makes it easy to manage task placement in clusters of pSeries systems. It helps implement various job management policies to maximize the system performance. The following pages describe vsrac in great details.

Installation and usage guidelines for VSRAC (Versatile System Resource Allocation and Control)

Contact:

Pascal Vezolle:ezolle@fr.ibm.com: (33) 4 67 34 45 41

Jean-Armand Broyelle: abroyelle@fr.ibm.com: (33) 4 67 34 63 75

Francois Thomas: ft@fr.ibm.com: (33) 4 67 34 40 61

14. Overview

VSRAC is a package that provides means of controlling task placement on clusters of multi MCM/SCM pSeries AIX systems used in large computing centers. It sits between the IBM HPC software components (LoadLeveler and Parallel Environment) and the AIX resource management libraries and tools (CPU Resource Set, WorkLoad Manager). This tool streamlines and simplifies the usage of AIX Affinity services while providing workload management capability.

VSRAC manages user jobs requesting some amount of resources (number of processes and/or threads) according to policies that lead to the allocation of target resources. The tool provides a wide range of resource allocation policies to fit the maximum of customer production configurations. Some VSRAC capabilities are not implemented yet and might be depending on customer needs. The current version does not manage threads placement.

It is an open source (no fee, capability to change the source). Other features might be added.

The two main features are to guarantee MCM-affinity or to control resources for both interactive and batch jobs. To benefit of the memory affinity the processes should be constrained to run in a single MCM.

The coming versions of IBM parallel environment and LoadLeveler will bring some features provides by VSRAC. Parallel environment 4.1 will provide an environment variable, MP_AFFINITY, which will attach each parallel task to a single MCM. This new capability will be useful for running a large MPI application on dedicated nodes. In configurations where the workload is made up of sequential, multi threaded and mpi jobs, installation of such a vsrac tool is necessary.

In 2005 LoadLeveler will bring resource set reservation features.

The VSRAC package consists in a library, driver commands ***vsrac***, ***mpp***, configuration files and the ***vsem*** control command. At installation time (rpm package), VSRAC creates a modified version of the pmdv4 IBM POE daemon (***pmdv4vsrac***) as well as entries in /etc/services, /etc/inetd.conf, /etc/inittab and /etc/security/user. VSRAC creates a RSET configuration.

Besides the configuration files, several environment variables should be set to activate the tool, specify job types, resource allocation policies or target resources.

15. Installation

VSRAC comes in a RPM format. We describe the binary RPM installation as well as the source based installation.

15.1 Installation of the binary RPM

Download the appropriate rpm file and install it with:

rpm -i --nodeps vsrac-1.0.0-1.aix5.2.ppc.rpm

It is important to use the -- nodeps option when installing the RPM. This limitation will disappear in the future.

To uninstall VSRAC, use the following:

rpm -e vsrac

VSRAC installs itself under /opt/vsrac. It also affects the following system files:

15.1.1 /usr/lpp/ppe.poe/bin/pmdv[3|4]vsrac
a modified version of the IBM Parallel Environment partition manager daemon (pmd). Removed upon uninstallation.

15.1.2 /etc/inetd.conf
an entry for the modified pmd daemon is added. Removed upon uninstallation.

15.1.3 /etc/services
port numbers (6136 and/or 6137) are added for the new pmd daemon. Removed upon uninstallation.

15.1.4 /etc/security/user
the default stanza is modified to allow all users the necessary capabilities to attach processes to resource sets as shown below :
capabilities = CAP_NUMA_ATTACH,CAP_PROPAGATE

15.1.5 /var/adm/vsrac
this directory contains the logs file when VSRAC is used in verbose mode. Removed upon uninstallation. The access right of this file must be 666.

15.1.6 . rsets database
upon installation, VSRAC creates and loads a resource set database that matches the MCM/SCM topology of the target system. The namespace used for this database is “mcm”. Removed upon uninstallation.

15.2 Installation from source code

In case you wish to rebuild a binary RPM from source code, proceed as follows:

15.2.1 Install the source RPM

rpm -i vsrac-1.0.0-1.src.rpm

This creates two files: the tarball and the spec file. The tarball contains the source code. The spec file is the file used by the rpm command to build the binary RPM. Both files are installed under /usr/src/packages/ : the former in SOURCES/vsrac-1.0.0.tar.gz and latter in SPECS/vsrac.spec.

15.2.2 Modify the source code if needed

After untarring the /usr/src/packages/SOURCES/vsrac-1.0.0.tar.gz file, you may apply the desired code changes in the vsrac-1.0.0 directory. Once you are ready to rebuild, use the make rpm target to rebuild the tarball under /usr/src/packages/SOURCES/.

15.2.3 Rebuild the binary RPM

This is accomplished with the following command:

rpm -ba /usr/src/packages/SPECS/vsrac.spec

The resulting RPM will be under /usr/src/packages/RPMS/ppc, ready for installation.

15.2.4 Description

VSRAC tool allows to attach processes to Resource Set, WorkLoad Manager Classes or to bind them to processors. The tool provides configuration files, environment variables and LoadLeveler interfaces. Configuration files and environment variables are described in detail below.

The available policies are classified in 3 classes based on AIX Affinity services: ***CPU Resource Set***, ***WorkLoad Manager*** and ***binding***. For each policy there are 3 kinds of placement schemes that deal with how the processes are located on processors: default AIX placement, populating each MCM sequentially in order to minimize the number of MCMs per job, and round robin placement on the MCM number (from 0 to number of MCMs -1).

For more details about AIX APIs take a look on http://publib16.boulder.ibm.com/cgi-

bin/ds_form?lang=en_US&viewset=AIX.

15.3 Resource Set Policies: rset_mcm, rset_mcm_r, “ret_excl”

The main idea based on CPU Resource Set is to attach a process to a set of processors to guarantee MCM affinity and then keep Memory affinity. The displacement from an MCM to another of a process leads to a loss of memory affinity and a decrease of the memory bandwidth. The displacement of the processes might also generate significant fluctuations of the executable times. To fit the production requirements for High Performance Computing application, VSRAC provides two policies for attaching processes to CPU resource set: rset_mcm and rset_mcm_r:

15.3.1 rset_mcm (target: smooth execution time fluctuations): the processes are placed contiguously, minimizing the number of MCMs used

15.3.2 rset_mcm_r (target: optimize memory bandwidth): the processes are placed on the MCM in a round robin way. For MPI jobs, task 0 on the first MCM, task 1 on the second MCM,

In the next release a new rset_excl policy will be implemented. The processes will be placed contiguously on a CPU resource set while excluding any other jobs to run on this rset. This feature will be available in LoadLeveler in 2005.

The CPU resource set configuration is built at VSRAC installation. The new Rset are called mcm/mcmX, where X corresponds to the MCM number: mcm/mcm0 for MCM 0, mcm/mcm1 for MCM1, …, mcm/mcm01 for MCM0 and MCM1, etc. The AIX 5.2 command ‘lsrset –av” displays the resource set configuration. The number of Rset depends on the number of MCM or CSM available on the system. At the installation VSRAC sets up an rset for each MCM combination. The aim is to reduce the displacement of the processes across the MCMs. Depending on the number of CPUs available per MCM and the number of threads of the process, a process can be attached to several MCM.

To balance the workload on system and to assign processes to the less loaded MCM, the occupancy rate of the MCM or the processor must be known at any time. VSRAC manages the workload from a system semaphore. This system semaphore is created at installation or at each reboot of the system from /etc/inittab. This workload management semaphore shows two kinds of information: the number of threads running per MCM and the number of threads bound per processor. The command “vsem” (described below) displays the current workload and manages the semaphore.

VSRAC internal scheduler relies on the current workload from the semaphore to compute the process location and update the semaphore versus the policy. This scheduler can be disable by an environment variable “WORKLOAD_RAC=off”. If this variable is set to off, VSRAC scheduler ignores the current workload to compute process allocation and does not updated the system semaphore. When the variable is set to on, the process location is calculated depending on the workload on each MCM.

The current version of VSRAC exclusively assigns processes not thread. Nevertheless the number of threads per process is used to compute the process location and the workload.

A user can specify a list of MCMs or processors on which the processes will be attached through the environment variable “TARGET_RAC”.

The environment variable “OVERBOOKING= %” defines the limit beyond that VSRAC does no longer compute process location and attachment. This variable specifies the CPU over commitment under which VSRAC will manage the resource allocation and task placement. If the CPU workload becomes higher, VSRAC will not attempt to place the tasks on the rsets or processors and let the AIX scheduler manage. It is expressed in percentage. For example, setting it to 200 will allow VSRAC to manage the task placement until the cpu workload reaches 300% (3 tasks). To balance the workload across MCM, VSRAC scheduler assigns a new process to the less loaded MCM.

The scheme below shortly outlines the scheduler algorithm.

```
Sort current MCM workload on the ascending number of threads already running
    for i=1, number of processes
     {
    1. try to assign all the threads of the process to a single MCM
    2. if (1  failed )
            for j=1, number of threads of process(i)
            {
                    if ( VSRAC over commitment is not reached )
                            Adding the thread to the less loaded MCM
                            Updating the workload
                    else
                            Set the thread as no attachable
                    endif
                    }
             endif
             if ( round robin scheme ) current MCM++;
    }
```

Only the jobs launched through VSRAC are recorded in the system semaphore. In a production mode environment, a partial usage of the tool can generate a degradation of the performance, some MCM being able to be overloaded.

15.4 Workload Manager Policies: ll_wlm, ll_wlm_rset, ll_wlm_rset_r

It is often difficult to manage a system running both modes: interactively or through a batch scheduler. WLM classes are meant to deal with this case using WLM features (shares, tiers, rules and limits). This solution combines LoadLeveler and WLM classes, the interactive jobs being usually attached to WLM “default” class. The common configuration is to define WLM classes with inheritance settings and then to attach the LL “starter” daemon to the matching WLM class. The attachment of the “starter” takes place in LL prolog.

Moreover, the coupling of LL and WLM classes allows to use the capabilities of pre-emption of WLM, the WLM pre-emption being able to be more flexible than the LL pre-emption. This pre-emption is used to release hardware resources for urgent applications. Usually, the WLM shares and limits are more suitable to create priority classes than WLM tiers.

The VSRAC ll_wlm policies have be implemented to attach LL jobs to WLM classes. The WLM class attachment is realized by LL prolog for serial jobs and “pmdv[3|4]vsrac” process for MPI jobs. Integrating ll_wlm policies in Parallel Environment allows to put WLM class into general use, LL prolog is launched on a unique node and can not attached the ‘starter’ process on each node for a parallel job.

The matching between LL classes and WLM classes is set in the /opt/vsrac/etc/local.llwlm.cfg file on each node of the cluster. Such a implementation based on a local configuration allows to tune the policy regarding the local resources. This file is described in detail below.

On top of a simple attachment to a WLM class (ll_wlm policy), process can be assigned to CPU resource set in a contiguously way (ll_wlm_rset policy) or in a round robin way versus the MCMs (ll_wlm_rset_r).

In the case of ll_wlm policy, VSRAC scheduler is disabled and the workload is controled by WLM shares and limits.

Using VSRAC ll-wlm policies has 2 constraints:

- It is not compatible with LoadLeveler ConsumableCpus. When ConsumableCpus option is enabled by LL, the current WLM configuration is rubbed out.
- d.2. WLM classes must have a tag rule.

15.5 Binding Policies: bind_pr, bind_pr_r, "bind_th, bind_th_r"

These policies are essentially used for performance issues when running benchmarks. The processes are bound to processors. These processors are given by the user through "TARGET_RAC" variable or calculated by the scheduler. The system semaphore maintains the list of the processors running bound processes.

The capability to bind thread will be implemented later.

15.6 e.1. VSRAC implementation details

- VSRAC is based on AIX APIs attaching process to Rset and WLM class or binding to processor. MPI jobs are directly controlled by a new 'pmdv' daemon.
- Serial, OpenMP and multi-threads interactive jobs are managed by '***vsrac***' command while batch jobs are managed through LoadLeveler *user prolog*. The user *prolog* program increases the system semaphore and creates a temporary file on /tmp containing the value used to increase the system semaphore. This file is read and removed by the LL *user epilog* program while decreasing the system semaphore.
- MPI job*s* are controlled by a new Parallel Environment "pmdv" daemon. In order to not interfere with POE product VSRAC builds a new daemon pmdv[3|4]vsrac on / usr/lpp/ppe.poe/bin.

This new daemon is called by setting the poe environment variable

MP_PMDSUFFIX=vsrac.

The AIX ***fork*** and ***execvp*** subroutines in poe pmdv have be replaced by ***__bdf_fork*** and ***__bdf_execvp*** subroutines from VSRAC library /usr/bin/libbd_fork.a.

The system semaphore is updated by each MPI process in __bdf_fork routine. The flag "SEMDO' in the sembuf structure guarantees that the system semaphore will be updated when the process stops.

The paragraph below gives a description of the AIX API routines used in VSRAC implementation

15.7 How to attach a rset with API routines and VSRAC usage

A resource set is an opaque structure that identifies physical resources. The physical resources supported by the AIX are CPUs and memory pools; An rset parameter is used in many of the AIX resource set APIs to either get information from the system regarding resources or to pass information about requested resources to the system. Addition function are provided to examine or manipulate rsets. Application and job schedulers may attach an rset to a process. Attaching an rset to a process limits the process to only using the resources contained in the rset. The goal of this part is not to present in details all the Rset API routines but just to outline those used in VSRAC implementation. VSRAC Rset configuration is built at installation or reboot which limits the number of API routines. Only 6 rset API routines are needed to manage the VSRAC rset policies: ***rs_alloc***, ***rs_getnamedrset***, ***rs_getinfo***, ***ra_execvp***, ***ra_attachrset*** (AIX 5.2) or ***ra_attach*** (AIX 5.1)

In VSRAC the CPU Resource set are identified by a number: 0 for MCM0, 1 for MCM1, 01 for the merger of MCM0 and MCM1, etc … In API routines a rset is identified by a *namespace* and an *rsname*. The *rsname* (mcm0, mcm1, … in VSRAC) corresponds to the previously registered name of a resource set. The *namespace* (mcm in VSRAC configuration) corresponds to the name space within which rsname is found.

An rset is handled by a structure *rsethandle_t* defined in /usr/lib/include/sys/rest.h. This structure must be allocated by the ***rs_alloc*** routine.

The rs_alloc subroutine allocates a resource set and initializes it according to the information specified by the parameter *flags*. The value of the flags parameter determines how the new resource set is initialized. In VSRAC the flag is set to RS_EMPTY (or 0 value); rset is initialized to contain no resources.

The rs_getnamedrset subroutine retrieves a resource set definition from the system registry. The namespace and rsname parameters identify the resource set to be retrieved. The rset parameter identifies where the retrieved resource set should be returned. The rset parameter must be allocated (using the rs_alloc subroutine) prior to calling the rs_getnamedrset subroutine.

The number of CPU per MCM is get from ***rs_getinfo*** routine. The rs_getinfo subroutine retrieves the resource set information from the rset parameter. Depending on the value of the info_type parameter, the rs_getinfo subroutine returns information related to the number of available processors, the number of available memory pools, or the amount of available memory contained in the resource rset.

Syntax example of how to get the number of processors on MCM 0 in VSRAC

```
#include <sys/rset>
rsethandle_t rs_alloc (unsigned int flags);
int rs_getnamedrset (char *namespace, char *rsname, rsethandle_t rset);
int rs_getinfo(rsethandle_t rset, rsinfo_t info_type,unsigned int flags);
rsid_t rseth;
int error;
/* allocation rset structure */
  rseth.at_rset = rs_alloc(RS_EMPTY);
/* attach the rset structure to the mcm/mcm0 rset defined on MCM 0 */
  name_space="mcm";
  name="mcm0";
  error = rs_getnamedrset(name_space,name,rseth.at_rset);
/* number of processors on MCM 0 */
  error = rs_getinfo(rseth.at_rset,R_NUMPROCS,0);
/* size of  memory on MCM 0 */
  error = rs_getinfo(rseth.at_rset,R_MAXMEMPS,0);
  …
```

The Parallel Environment pmdv program launchs MPI task from the kernel ***fork*** and ***execvp*** routines. The new program pmdv[3|4]vsrac from VSRAC tool replaces these calls by internal routines __bdf_fork and __bdf_execvp. Attaching a rset to a process is realized in the __bdf_execvp using rset API ***ra_execvp*** subroutine.

The standard execvp routine is simply substituted by ra_execvp routine. The example below describes how to attach the mcm/mcm0 rset to a process

```
/* allocation rset structure */
  rseth.at_rset = rs_alloc(RS_EMPTY);
/* attach the rset structure to the mcm/mcm0 rset defined on MCM 0 */
  name_space="mcm";
  name="mcm0";
  error = rs_getnamedrset(name_space,name,rseth.at_rset);
```

ra_execvp(R_RSET, rseth, 0, executable name, arguments)

Jobs that are not MPI are assigned to a rset through VSRAC command or LL prolog using rset API ***ra_attachrset*** (AIX 5.2) or ***ra_attach*** (AIX 5.1). The example below shows how to attach the mcm/mcm0 rset to a process.

```
/* allocation rset structure */
  rseth.at_rset = rs_alloc(RS_EMPTY);
/* attach the rset structure to the mcm/mcm0 rset defined on MCM 0 */
  name_space="mcm";
  name="mcm0";
  error = rs_getnamedrset(name_space,name,rseth.at_rset);
/* get process pid */
  …
/* attach the rset in AIX 5.2 */
  error = ra_attachrset(R_PROCESS, pid, rseth.at_rset, 0);
/* attach the rset in AIX 5.1 */
  error = ra_attach(R_PROCESS, pid, R_RSET , rseth, 0);
```

15.8 How to attach a WLM class with API routines and VSRAC usage

The ***wlm_initialize*** routine initializes the WLM API for use with an application program. It is mandatory to call ***wlm_initialize*** prior to using the WLM API. Otherwise, all other WLM API function calls will return an error. If ***wlm_initialize*** is used in a multi-threaded application, the routine should be called by the main thread before additional threads are started.

Two methods are implemented in VSRAC to attach a process to WLM class:

- One process attaches by itself to a WLM class by setting the tag of the class. This method is used for MPI task. Each MPI task sets the WLM tag in the ‘fork’ routine called by ‘pmdv” daemon. API name: ***wlm_set_tag***
- *wlm_set_tag* description: The tag is a new attribute of a process that can be set using the WLM function wlm_set_tag. This tag is a character string with a maximum length of WLM_TAG_LENGTH (not including the null terminator). Process tags can be displayed using the ps command. The tag is also one of the process attributes used in the assignment rules to automatically assign a process to a given class. When an application sets its tag using wlm_set_tag, it is automatically reclassified according to the current assignment rules, and the new tag is taken into account when doing this reclassification. In addition to the tag itself, the application can also specify flags indicating to WLM whether a child process should inherit the tag from its parent after a fork and/or an exec system call. A process does not require any special privileges to set its tag.
- How to assign a process by itself in a WLM class using the tag of the class

```
#include <sys/wlm.h>
#include <sys/user.h>
int wlm_initialize (flags);
int wlm_set_tag (tag, &flags);
int flags, error;
char *tag;    /* The address of a character string. An error
                 will be returned if his tag is too long */
/* WLM API initialization */
```

```
error = wlm_initialize(WLM_VERSION);
/* tag setting */
        flags=                          WLM_VERSION|SWLMTAGINHERITFORK|
  SWLMTAGINHERITEXEC;
error = wlm_set_tag( tag, &flags);
```

- A process is assigned to a WLM class using the pid. This method is implemented in the LL 'prolog' program to attach the LL 'starter' process? API name: ***wlm_assign***
- *wlm_assign* description: The wlm_assign function is used to:

g.6. Assign a set of processes specified by their process identifiers (pids) or process group identifiers (pgids) to a specified superclass or subclass, thus overriding the automatic class assignment or a prior manual assignment.

- Cancel a previous manual assignment, allowing the processes to be subjected to the automatic assignment rules again.
- Implementation LL prolog to assign LL 'starter' process

```
#include <sys/wlm.h>
        int wlm_initialize (flags);
        int wlm_assign ( & args);
        struct wlm_assign args;
        int error;
        /* WLM API initialization */
        error = wlm_initialize(WLM_VERSION);
        /* starter process assignment */
        args.wa_pid_u._pids  = &_starter_pid;          /* LL starter pid */
        args.wa_pid_count    = 1;              /* number of pids */
        args.wa_pgid_count  = 0;                        /* number of gpids */
        args.wa_versflags     = WLM_VERSION|WLM_ASSIGN_SUPER;
        strcpy( args.wa_classname , wlm_class_name );
error = wlm_assign( &args );
```

15.9 How to bind a process with bindprocessor API routine and VSRAC usage

The *bindprocessor* subroutine binds a single kernel thread, or all kernel threads in a process, to a processor, forcing the bound threads to be scheduled to run on that processor. It is important to understand that a process itself is not bound, but rather its kernel threads are bound. Once kernel threads are bound, they are always scheduled to run on the chosen processor, unless they are later unbound. When a new thread is created, it has the same bind properties as its creator. This applies to the initial thread in the new process created by the *fork* subroutine: the new thread inherits the bind properties of the thread which called *fork*. When the *exec* subroutine is called, thread properties are left unchanged.

In VSRAC, each MPI task is bound in ***fork*** routine using the option ***BINDTHREAD*** of ***bindprocessor*** subroutine. The array 'cpu_allocation[i]' gives to processor number for the MPI task i. The following lines are includes in VSRAC __bdf_execvp routine of pmdv[3|4]vsrac program:

```
 #include <sys/processor.h>
  int error;
  error =bindprocessor(BINDTHREAD, thread_self, cpu_allocation[__bdf_ind]);
```

For none MPI jobs the bind of the LL starter program takes place in LL prolog.

```
#include <sys/processor.h>
int error;
error =bindprocessor(BINDPROCESS, (int) "starter pid", (int) "processor
                    number");
```

16. Current usage and limitations

16.1 Limitations

- AIX 5.1 ML4, LoadLeveler 3.1, Parallel Environment 3.2 and standalone system. The tool does not work on cluster. The issue has been identified and might be fixed in a next release.
- AIX 5.2 LoadLeveler 3.2, Parallel Environment 4.1
- VSRAC tool has not been tested and validated with the LoadLeveler ***llmodify*** command, especially with the option –C specifying a new LL class.

16.2 Interactive Jobs

Interactive jobs can be managed by VSRAC under the following conditions:

- Serial, OpenMP or multithreaded jobs must be started with the ***vsrac*** driver command
- MPI jobs must be started with the ***mpp*** command. ***mpp*** is a wrapper script around the ***poe*** command that accepts all options of standard poe command.

16.3 LoadLeveler jobs

- Loadleveler configuration

To enable VSRAC under LoadLeveler, the system administrator must activate a user job *prolog* and *epilog* in the LoadL_config file. The following lines must be added:

- JOB_USER_PROLOG = /opt/vsrac/bin/prolog_vsrac.sh
- JOB_USER_EPILOG = /opt/vsrac/bin/epilog_vsrac.sh

If user *prolog* and *epilog* are already being used at your site, make sure you call these two scripts at the end of you current *prolog* and *epilog*.

17. LoadLeveler usage

LoadLeveler job command files need to be changed to activate VSRAC control. The first variable to check is MCM_AFFINITY, which can be set globally to "on" in /opt/vsrac/etc/local.cfg) or locally in the "# @ environment" section of a LoadLeveler job.

Other VSRAC environment variables must be set in the environment field of LoadLeveler jobs command files. If they are not set properly, the job *prolog* will exit with a bad return code, preventing the job from running.As VSRAC is an open source tool, this default behaviour can be easily changed by the customer to fit his production requirements or contact the developers to realize the needed modifications.

Typical "# @ environment" stanzas of VSRAC enabled LoadLeveler job command files are listed below. The POLICY_RAC possible values are listed further down.

VSRAC environment variables in bold are mandatory in #@environment field, the other can be set in/opt/vsrac/etc/ local.cfg file.

17.1 Serial job

 # @ job_type = serial

 # @ environment = COPY_ALL ; MCM_AFFINITY=on ; \ JOBTYPE_RAC=serial ; POLICY_RAC=rset_mcm

17.2 OpenMP job

@ job_type = serial

@ environment = COPY_ALL ; MCM_AFFINITY=on ;\ ***JOBTYPE_RAC=openmp*** ; POLICY_RAC=rset_mcm ; ***OMP_NUM_THREADS***=8

17.3 Threaded job

@ job_type = serial

@ environment = COPY_ALL ; MCM_AFFINITY=on ; \ **JOBTYPE_RAC=*threads*** ; POLICY_RAC=rset_mcm_r ; \ ***THREADS_TASK_RAC***=4

17.4 MPI job

@ job_type = parallel

@ total_tasks = 4

@ environment = COPY_ALL ; MCM_AFFINITY=on ; \ ***JOBTYPE_RAC=mpi*** ; POLICY_RAC=rset_mcm

MPI-OpenMP job (mixed)

@ job_type = parallel

@ total_tasks = 4

@ environment = COPY_ALL ; MCM_AFFINITY=on ; \

JOBTYPE_RAC=mpi_openmp ; POLICY_RAC=rset_mcm ; ***OMP_NUM_THREADS***=4

MPI-threaded job (mixed)

@ job_type = parallel

@ total_tasks = 4

@ environment = COPY_ALL ; MCM_AFFINITY=on ; \

JOBTYPE_RAC=mpi_openmp ; POLICY_RAC=rset_mcm ;\

THREADS_TASK_RAC=4

18. VSRAC commands

There are three main VSRAC commands. Two: - ***vsrac*** and ***mpp*** -, are intended for general users and the last one: - ***vsem*** -, is for administrators only. They have a corresponding man page.

18.1 vsrac : vsrac program [args ...]

VSRAC is a driver program that exploits the task allocation and placement features of VSRAC. It can be used (or must be used in a global approach) to start serial or multithreaded (pthreads, OpenMP) programs. The behaviour is driven by environment variables.

18.2 mpp : mpp [poe args] program [program args]

mpp is a driver program used to start interactive IBM POE jobs. It exploits the task allocation and placement feature. It has the same function as the VSRAC command for serial or multithreaded jobs (pthreads or OpenMP).

18.3 vsem: vsem [-i|-k-o|-w]

vsem is an utility program that is used to initialize the workload management semaphore, display the current workload or, - in expert mode only -, kill the semaphore structure or fix the workload information if needed. Without an option, the default action is -o, display the current VSRACworkload.

- Initialize the semaphore structure (experts only)
- Kills the semaphore structure (experts only)
- Outputs the semaphore structure contents
- Writes into the semaphore structure (experts only)

Here is a sample output. The system is made up of 4 NUMA blocks, each with 2 cpus. The

workload per CPU is only reported if processes/threads are bound to the cpus. Otherwise, the load only appears on a per NUMA block (MCM) basis.

```
#MCM=4 #cpus/MCM=2
#threads on MCM 0 is 2
#threads on MCM 1 is 2
#threads on MCM 2 is 4
#threads on MCM 3 is 2
#threads bound to CPU 0 is 0
#threads bound to CPU 1 is 2
#threads bound to CPU 2 is 0
#threads bound to CPU 3 is 0
#threads bound to CPU 4 is 1
#threads bound to CPU 5 is 0
#threads bound to CPU 6 is 0
#threads bound to CPU 7 is 0
```

19. Environment variables

20. Configuration files

20.1 local.cfg

/opt/vsrac/etc/local.cfg is the general configuration file for VSRAC. It is read by the VSRAC libraries and utilities. It might look as follows:

```
# This line is a comment
MCM_AFFINITY=off
OVERBOOKING=0
NAME_SPACE_RSET=mcm
WORKLOAD=on
NB_MCM=4
NB_CPU_MCM=2
POLICY=no
```

This configuration file specifies default values for VSRAC. It is created at boot time and usually does not need to be changed later on. The variables are described below. MCM_AFFINITY, WORKSPACE, OVERBOOKING and POLICY can be overwritten by environment variables (MCM_AFFINITY, WORKLOAD_RAC, OVERBOOKING_RAC, POLICY_RAC).

Upon boot up, the VSRAC init script will save the previous configuration file in /opt/vsrac/etc/local.cfg.last.

20.2 local.llwlm.cfg

/*opt/vsrac/etc/local.llwlm.cfg* is required to implement the ll_wlm policies. It gives the mapping between LoadLeveler classes and WLM classes. A sample file is given in /*opt/vsrac/etc/local.llwlm.cfg.sample*.

This file gives the WLM class name and tag as well as a VSRAC scheduler option (called *vsrac_workload*) for each LoadLeveler class. It must be created by the system administrator before using the ll_wlm policies.

If LoadL_class field is set to ‘none’ there is no LL class and jobs are directly assigned to the corresponding WLM class.

The latest field *vsrac_workload* is an option in the internal scheduler for ll_wlm_rset and

ll_wlm_rset_r policies. If this field is set to “yes” the real workload of the system is not used to compute the process location on the MCM. When a job is launched in a high priority WLM (low tier) the other jobs running on the system might be pre-empted to free resources. In this configuration, setting *vsrac_workload* to “yes” allows to optimize the placement of processes through MCMs.

Local.llwlm.cfg file example:

#LoadL_class	*WLM_class_name*	*WLM_tag*	*vsrac_workload*
prod	prod	tag1	no
test	test	tag2	yes
night	night	night	no
none	day	tag3	yes

21. Examples

21.1 Example 1: Optimizing Memory bandwidth for MPI job

Some Computing Center does not need to put in place a general strategy of resource allocation. When the workload is dominated by a single MPI application running on a large number of systems, the priority is to constraint MPI task to a single MCM to guarantee memory affinity. On single multi MCM system, the MPI task can be attached to a MCM in a sequential way or in a round robin way. If the number of threads per MPI task is not handled, the only way to avoiding MCM overloaded is the round robin way. Such a policy is available in Parallel environment 4.1 and activated by the environment variable MP_AFFINITY.

This case can also be managed by VSRAC with the simple configuration:

1. Manually by setting the environment variable MCM_AFFINITY=on

1.1. Example of /opt/vsrac/etc/local.cfg file on a p690 32 way:

1.1.1. MCM_AFFINITY=off

1.1.2. OVERBOOKING=0

1.1.3. NAME_SPACE_RSET=mcm

1.1.4. WORKLOAD=on

1.1.5. POLICY=rset_mcm_r

1.1.6. NB_MCM=4

1.1.7. NB_CPU_MCM=2

1.2. Set MP_PMDSUFFIX=vsrac in /etc/environment or copy / usr/lpp/ppe.poe/bin/pmdv3.vsarc in /usr/lpp/ppe.poe/bin/pmdv3 with the same rights

2. Automatically

2.1. Example of /opt/vsrac/etc/local.cfg fileon a p690 32 way:

2.1.1. MCM_AFFINITY=on

2.1.2. OVERBOOKING=0

2.1.3. NAME_SPACE_RSET=mcm

2.1.4. WORKLOAD=on

2.1.5. POLICY=rset_mcm_r

2.1.6. NB_MCM=4

2.1.7. NB_CPU_MCM=2

Set MP_PMDSUFFIX=vsrac in /etc/environment or copy /usr/lpp/ppe.poe/bin/pmdv3.vsarc in /usr/lpp/ppe.poe/bin/pmdv3 with the same rights

21.2 Example 2: Managing interactive and batch activities on a single system

(More on this later…)

21.3 Example 3: Reducing execution time fluctuations for parallel applications

(More on this later…)

Field	value	Description
MCM_AFFINITY=	on|off	Set to "on" enables VSRAC usage and to "off disables it, letting AIX manage the resource allocation and control. "off" is the default. The value read in the configuration file is overwritten by the MCM_AFFINITY environment variable if set.
OVERBOOKING=	integer	Specify the CPU over commitment under which VSRAC will manage the resource allocation and task placement. If the CPU workload becomes higher, VSRAC will not attempt to place the tasks on the rsets or cpus and let the AIX scheduler manage. It is expressed in percentage. For example, setting it to 200 will allow VSRAC to manage the task placement until the cpu workload reaches 300% (3 tasks). The value read in the configuration file is overwritten by the OVERBOOKING_RAC environment variable if set.
WORKLOAD=	mcm	Name space used for defining the AIX resource sets. This is set by default to "mcm" and should not be changed.
NB_MCM=	integer	Number of SCM/MCM in the system. Although the variable is named NB_MCM, it accounts for the number of NUMA blocks in the system: SCM or MCM. This value is computed at boot time and should not be hand edited.
NB_CPU_MCM=	integer	Number of CPUs of each block (SCM or MCM) This value is computed at boot time and should not be hand edited.

Field	value	Description
POLICY=	no rset_mcm rset_mcm_r ll_wlm ll_wlm_rset ll_wlm_rset_r bind_pr bind_pr_r bind_th bind_th_r	Used as the default allocation policy. It is set to "no" by default in the configuration file which reverts to using the default AIX scheduler. The value read in the configuration file is overwritten by the POLICY_RAC environment variable if set. The available policies are listed below. They can be classified depending on the target resource that they will allocate to the job. For more details about policy read POLICY_RAC variable description
LOG_LEVEL	1-4	This variable set the log verbosity level in /var/adm/vsrac/logs file: 1 (default): only vsrac errors 2: 1 + warnings 3: 2 + info messages 4: 3 + debugging messages
TARGET_RAC	List of integers	Give a list of MCM/SCM or processors on which the processes are placed. If this variable is set, the VSRAC scheduler does not take into account the current workload of each NUMA block to place the tasks. It will however update the workload consistently. - for wlm and rset policies: TARGET_RAC specifies a list of MCM or SCM (from 0 to number of MCM-1) - bind policies: TARGET_RAC specifies a list of processors (from 0 to the maximum number of processors)

WLMCLASS_RAC	wlm class name	Specifies a wlm class name for ll_wlm, ll_wlm_rset and ll_wlm_rset_r policies. If this variable is not set vsrac reads the /opt/vsrac/etc/local.llwlm.cfg file.

OVERBOOKING_RAC	an integer (in percent)	specify the CPU overcommitment under which VSRAC will manage the resource allocation and task placement. If the CPU workload becomes higher, VSRAC will not attempt to place the tasks on the rsets or cpus and let the AIX scheduler manage. It is expressed in percentage. For example, setting it to 200 will allow VSRAC to manage the task placement until the cpu workload reaches 300% (3 tasks). Default value in /opt/vsrac/etc/local.cfg

WORKLOAD_RAC	On | off	This variable activates VSRAC scheduler. The processes are placed regarding the currently workload managed by VSRAC. The command vsem displays the current workloadl. If the variable is set to off the processes are placed versus the policy with the current workload at 0. Default value in /opt/vsrac/etc/local.cfg

THREADS_TASK_RAC	value	Number of threads per process. This variable must be set with JOBTYPE_RAC=threads or mpi_threads. Default is 1 thread per process

OMP_NUM_THREADS	value	Number of threads per process. This variable must be set with JOBTYPE_RAC=openmp or mpi_openmp. Default is 1 thread per process

MP_PMDSUFFIX (poe environment variable)	vsrac	Determines a string to be appended to the Partition Manager daemon service, or executable (when using LoadLeveler). The PMD service in /etc/services is named pmv4. By setting MP_PMDSUFFIX, you can append a string to pmv4. If MP_PMDSUFFIX is set to VSRAC, the service requested in /etc/services is pmv4vsrac. When using LoadLeveler, the string is appended to the partition manager daemon executable name, /etc/pmdv4.

JOBTYPE_RAC	serial openmp threads mpi mpi_openmp mpi_threads	The type of the job

POLICY_RAC	No rset_mcm rset_mcm_r rset_excl ll_wlm ll_wlm_rset ll_wlm_rset_r bind_pr bind_pr_r bind_th bind_th_r	Set VSRAC allocation policy. It is set to "no" by default in the configuration file which reverts to using the default AIX scheduler. The default is set in /opt/vsrac/etc/local.cfg. If not listed in the local.cfg, the dedault is no. The available policies can be classified depending on the target resource that they will allocate to the job. There are 3 policies classes based on AIX Affinity services: rset, WLM and binding. For each policies there are 3 types placement schemes specifying how the processes are located on processors: default AIX placement, continuous and round robin placement on the MCMs .
	No	default policy, AIX scheduler

RESOURCE SET Policies (MCM/Memory Afinity)

	rset_mcm	This policy attaches the processes/threads of the job to an AIX resource set defined on the MCMs, the processes are placed in order to populate each MCM sequentially, minimizing the number of MCMs. The resource sets are created at boot time by the VSRAC initialization script. This target is to try to guarantee MCM affinity and no losing memory affinity during execution. Compatible with LoadLeveler ConsumableCpus.
	rset_mcm_r	this policy is similar to rset_mcm, the only difference is that the process are located on MCM in round robin allocation way
	rset_excl	Resource rset exclusive policy. The processes are placed contiguously on a CPU resource set while excluding any other jobs to run on this rset. Not yet implemented in VSRAC. This feature will be available natively in the next LoadLeveler version.

WorkLoad Manager Policies (Workload control + MCM/Memory affinity)

	ll_wlm	this policy is used to assign a LoadLeveler class to a WLM class. With this policy, the VSRAC internal scheduler is disabled. The LL/WLM configuration is defined in the local.llwlm.cfg file under / opt/vsrac/etc/. Not compatible with LoadLeveler ConsumableCpus.
	ll_wlm_rset	This policy is the same as ll_wlm adding process placement with VSRAC scheduler. As a supplement to a simple attachment to WLM class, process can be assigned to CPU resource set. The processes are allocated continuously on MCMs, minimizing the number of MCMs per job. Not compatible with LoadLeveler ConsumableCpus.
	ll_wlm_rset_r	Same as ll_wlm_rset expect round robin placement on MCMs Not compatible with LoadLeveler ConsumableCpus.

Binding (CPU affinity)

	bind_pr	this policy binds the processes to individual processors. The processes are attached to processors continuously.
	bind_pr_r	this policy is similar to the previous one, the only difference is that we use a round robin allocation scheme, maximizing the number of MCM blocks used.
	bind_th	the policy is to bind the job's threads to individual processors. The threads are placed on contiguous processors. Not yet implemented
	bind_th_r	Idem bind_th policy, the only difference is that we use a round robin allocation scheme Not yet implemented

MCM_AFFINITY	On | off	set it to "on" to enable VSRAC globally and to "off to disable it, letting AIX manage the resource allocation and control. The default is set in / opt/vsrac/etc/local.cfg. If not listed in the local.cfg, the dedault is off.

VSRAC Environnent variables
MCM_AFFINITY
POLICY_RAC
JOBTYPE_RAC
THREADS_TASK_RAC
WORKLOAD_RAC
TARGET_RAC
WLMCLASS_RAC
OVERBOOKING_RAC
MP_PMDSUFFIX=vsrac
LOG_LEVEL

Key to obtaining the desired performance, I installed vsrac : a task placement tool developed by IBM Montpellier.

The disk layout was ready too with three main partitions:

- /TMP, a fast RAID0 file system for use by programs requiring fast IO on temporary files.
- /users, a safe RAID5 file system for the HOME directories of the users.
- /oper, a safe RAID5 file system whose projected use is to hold the data produced by the operational forecast over an extended period of time.

We then installed an archive of the work performed in our benchmark centers, our starting point for the acceptance test.

Prior to running the acceptance tests, we created a basic LoadLeveler configuration and started the daemons. In the event of a system restart, these daemons are not automatically started and as root, the following commands would be necessary:

su – loadl

$ llctl -g start

Acceptance test results and log files

The acceptance test comprises 4 parts:

- benchmark runs of lancelot in asis and tuned mode
- benchmark runs of morgane in asis and tuned mode compilation and linking time
- switchover test between an operational job and a research job, both running 32 way

We present here the results we obrained as well as the results we got in our benchmark centers last year.

22. Benchmark runs

Note: no tuning was applied to the lancelot runs so we only ran the asis test.

23. Performance data

Benchmark	Contractual time (s)	Time obtained in Bratislava (s)
lancelot asis	77	77
morgane asis	1471	1488
morgane tuned	1293	1304

Part of the slight difference between the two timings (1%) could be reduced by using a faster file system than the one we used. This was not judged critical for the acceptance test but this will be applied to the operational suite.

Benchmark	Log file location
lancelot asis	~aladin/shmu/listing/lancelot-bratislava
morgane asis	~aladin/shmu/listing/morgane-bratislava-untuned-affinity-rac
morgane tuned	~aladin/shmu/listing/morgane-bratislava-tuned-affinity-rac

24. Compilation and link

These timings were obtained prior to the parallelization of the building process described later on. Due to some problems in the initial Makefile, the compilation proceeds in two steps. To look for the timings for each step : compile (phase1), compile (phase2) and link, use the grep command and look for the "real<TAB>" string in the log file.

Test name	Contractual time (s)	Time obtained in Bratislava (s)
compilation	2575	2498
link	1	1

Test name	Log file location
compilation and link	~aladin/shmu/src/compile.log

25. Switchover

The research job is started first. The operational job preempts the research job using the LoadLeveler preemption mechanism. We used the asis version of ALADIN for both jobs. The timings are listed below. "Total time" is the time from submission of the job till completion. In the case of the research job, this is of course much larger than the "Real time" as the research job is preempted by the operational job.

Test name	Total time (s)	Real time (s)
research	3080	1498
Operational	1566	1566

Benchmark	Log file location
operational run	~aladin/shmu/oper/listing/morgane-untuned-gang
research run	~aladin/shmu/research/listing/morgane-untuned-gang

Note 1 : the preemption mechanism used is that of LoadLeveler. This is used at the Hungarian

Meteorological Service (HMS). Until very recently, this did not prove to be entirely satisfactory in production. However, we may have discovered the reason for the LoadLeveler preemption instability and solved the problem which was apparently not related to LoadLeveler itself. It is still too early to state that the problems encountered at HMS have disappeared as we need to observe the behavior for a longer period of time.

The preemption can be implemented with a different scheme, using a combination of LoadLeveler and AIX Workload Management (WLM) with possibly some benefits regarding the global throughput of the system. We are experimenting extensively these aspects and we will be able to propose the "best" solution for managing the workload and satisfy the customer requirements:

- absolute priority for the operational job
- ensure the compute time needed to run the operational forecast is constant, whichever the current workload of the system when the operational job is started
- enable interactive usage of the system during the day
- enable batch jobs to run over extended periods of time
- maximize the usage of the system

Note 2 : as seen in the timings table, the time to run the operational forecast is slightly bigger than the time to run the research job. This is being investigated but is apparently related to the difference in filling of the 4 memory pools (memory attached to each MCM).

26. Code tuning

As already detailed in the initial benchmark report, we have applied some tuning to AL25T2. The main changes are related to the use of the vectorized version of the MASS library (http://techsupport.services.ibm.com/server/mass/). This library implements mathematical functions (sin, cos, atan, log, exp, etc) in a particularly fast way. Even faster is the use of the vector functions where the operands are vectors instead of scalars. This was applied to the accvimp and accvimpd routines.

Also, we have made changes to fft992 so that we call, whenever possible the ESSL version of the Fast Fourier Transforms. Unfortunately, ESSL does not support every vector length for the FFTs and in the past the code would fail if the length was not supported by ESSL. We have improved this by checking whether the FFT length is acceptable for ESSL and if not, we revert to the classical fft992 routine. This tuning resides under the ~aladin/shmu/src/tun/xrd or ~nwp001/al26t1_op5/tun/xrd directories.

There is a scope for potential tuning in the accvimp and accvimpd routines that has not been explored yet. The idea is to concatenate multiple one dimensional temporary arrays into fewer bi dimensional arrays. This is very likely to help the memory accesses by limiting the number of streams between the L1 caches and the physical memory. If time permits, I will try to implement this in Tunisia and measure the impact.

27. Compilation of some graphics packages

We have successfully compiled PALADIN, GMT and GRIBEX on the system. The latter required the use of an additional flag for pbio to specify that the default size for a Fortran integer is the same as a C int (-DINTEGER_IS_INT). If this is not specified, pbio uses C long integers for Fortran integers. In 32bit compilation mode, this is correct as a long is 32bit long, just like a Fortran INTEGER but in 64bit compilation mode, a C long integer becomes 64bit long. This is summarized in the table below.

Type	Size in 32bit compilation mode (OBJECT_MODE=32)	Size in 64bit compilation mode (OBJECT_MODE=64)
C int	32bit	32bit

Type	Size in 32bit compilation mode (OBJECT_MODE=32)	Size in 64bit compilation mode (OBJECT_MODE=64)
C long	32bit	64bit
C long long	64bit	64bit
Fortran INTEGER	32bit	32bit

NCAR graphics still needs to be installed. This could be done, either using binaries compiled for a previous version of AIX or by compiling from source code. NCAR graphics has already been compiled under AIX version 5 at various meteos : Tunisia, Morocco, Turkey. I will send a copy of what was done in Tunisia, or at least give the instructions on how to recompile the package.

28. Attempt to compile AL26T1_OP5

We tried to compile the latest version of ALADIN. Unfortunately, there seems to be issues when running the morgane configuration. We discovered some inconsistencies between the definition of some routines and the way they are called. To check all occurrences of this problem, one could use the -qextchk compiler option at compile and link time.

However, this is during the compilation of AL26T1_OP5 that we improved the building process by enabling the use of the parallel feature of the GNU make (gmake -j). Each package is now built using two targets : first the modules and then the regular object files. The first target is built with the regular, sequential invocation of gmake but the second one is performed in parallel. The level of parallelism is controlled by the optional argument to the -j switch of the gmake command. If no argument is specified, gmake will build in parallel as many targets as there are processors.

I will have a chance to compile and link AL26T1_OP4 or OP5 for the Tunisian meteo. The lessons learnt there will be transferred to SHMU.

29. Getting the right performance

As described during informal discussions, the p690 system is made of 4 blocks, each with 8 processors (Multi Chip Module or MCM) and 8 G bytes of memory, spread over 2 memory cards. The 4 blocks are linked with a very high speed interconnection which ensures fast memory accesses from processes which are executing from distant blocks (MCM). Nevertheless, local memory accesses are always faster and should be preferred in all cases. We will learn in the following discussion how to ensure that this happens for the 32 MPI tasks that make up a morgane run.

Our first attempt to run morgane gave a total time of 1626s for the untuned version, compared to the 1471s that we obtained last year in our benchmark center, a difference of 10.5%.

30. Check the configuration

The first action was to check the configuration. In our benchmark center, we had 128GB of memory in 8 memory cards of 16GB each compared to the 8 memory cards of 4GB each in SHMU. The difference between 4GB and 16GB memory cards is that the 4GB cards have a single port whereas the 16GB are double ported, leading to higher memory bandwidth. We have observed up to 15% difference for applications doing only memory transfers and usually around 5% for real applications where not all floating point operations lead to memory transfers as is the case for ALADIN.

Here are a few commands for checking the configuration.

$ bindprocessor -q (lists the on line processors)

$ lscfg -vp |grep memory-controller (gives the number of memory cards)

$ lsattr -El sys0|grep realmem (gives the total amount of memory)

31. Memory affinity

Memory affinity is an AIX feature by which, when a process touches a page for the first time, this page is allocated in the memory local to the process. This must be enabled in the kernel with the following command.

vmo -y 1

However, with this scheme, if a process moves from one MCM to the other, it may allocate memory in a few different MCMs. This setting was already in effect on the system.

32. Memory affinity – improved

To ensure all the pages allocated by a process reside on one single MCM, the MEMORY_AFFINITY environment must be set to MCM.

$ export MEMORY_AFFINITY=MCM

Using this, we ran in 1567s, now 6.5% off the target

33. Memory affinity and task placement

Now that the memory is allocated on one single MCM, we have to ensure that the processes do not move from one MCM to the other. By default, the AIX scheduler tries to reschedule the processes on the processors where they were running before but this is not guarantied. Indeed, when looking at the error file from the morgane run, we see that the time per iteration increases. During the run, when system daemons start for example, our MPI tasks may be scheduled on other processors and even other MCMs loosing the benefits of memory affinity. The tool developed by IBM Montpellier : vsrac is here to constrain the tasks to remain on the same MCM. This is accomplished simply with the setting of some environment variables in the mpirun script. We need to set:

$ export MEMORY_AFFINITY=MCM

$ export MCM_AFFINITY=on (activate our vsrac tool)

$ export JOBTYPE_RAC=mpi

$ export POLICY_RAC=rset_mcm

Using these settings, the timing was down to 1514s, 3% off the target.

34. Using the latest compiler

After recompiling the code with the latest compiler (xlf 8.1.1.3), the time was down to 1488s, 1% off the target.

35. Job preemption strategies and tests

There are currently two possible strategies for implementing the job preemption required for regular production forecasts. One uses the native preemption capability of LoadLeveler. The other one uses a combination of LoadLeveler and Workload Manager (WLM).

36. LoadLeveler-only preemption

LoadLeveler can suspend already running jobs when a high priority job is submitted. The high priority job will run immediately till completion. When the job finishes, LoadLeveler will resume the suspended jobs. This scheme is used in production at HMS. As stated before, we need to collect information from HMS to see if the problems encountered in the past have disappeared completely.

The main drawback of this scheme is that it is not immune to the interactive workload that is running on the system when the high priority job starts. To protect ourselves from the interactive users, i.e. from processes started outside of LoadLeveler jobs, we limit any interactive shell to a small amount of CPU time, discouraging people from starting long running interactive commands.

To allow for such commands (compilations, post-processing, etc) that would exceed the CPU time limit, a simple shell script called i has been written. This script will create a LoadLeveler job command file to run the command passed as an argument. For example, to run a compilation that could take a long time, we use:

$ i make ALADIN.exe

The same command invoked without the i script would probably fail as it would exceed the CPU time limit.

37. LoadLeveler-WLM preemption

37.1 WLM principles

WLM is a very powerful workload control mechanism used by AIX. Simply stated, WLM assigns processes to classes according to customizable rules. WLM ensures that the various classes use resources according to predefined shares and limits.

37.2 Possible LoadLeveler–WLM configuration

The simplest configuration for job preemption would work as follows. We would define three WLM classes : “prod” for the operational jobs submitted by LoadLeveler, “other” for all the other batch jobs submitted by LoadLeveler and “inter” for interactive jobs, namely any other user process not submitted via LoadLeveler.

Now, by configuring properly the relative priorities of the “prod”, “other” and “inter” WLM classes, we can implement the job preemption as well as arbitrate the amount of CPU power for regular batch jobs (class “other”) and interactive usage (class “inter”). The partition between “other” and “inter” can even be changed dynamically, allocating more resources for the interactive usage during the day and less during the night.

Our first tests in SHMU were quite successful, leading to a maximum thruput of the system (the suspended job still gets some CPU cycles when the high priority job pauses for example for IO). The slight drawback is that, for the same reason, the preemption is not as sharp as with the LoadLeveler-only case. More tests need to be performed to decide on the best scheme to use, but certainly the LoadLeveler-WLM way appeals by its simplicity and the fact that interactive users need not worry about the commands they launch.

38. Additional system tasks

38.1 Virtual memory tuning

For large memory jobs like ALADIN, it is important to favor process pages over file system buffer pages in memory. AIX will attempt to cache in memory as much file data as possible, sometimes leading to processes paging in and out because the memory is full with file pages. We suggest running the following command as root:

```
# vmo -p -o minperm%=1 -o maxperm%=5 -o minclient%=1 -o maxclient%=5
# ioo -p -o maxpgahead=512 -o j2_maxPageReadAhead=512
```

38.2 bos.pmapi

This AIX package contains the kernel extension, libraries and header files needed for some hardware performance monitoring tools that will be described and used during the parallel programming class to be held in March. It would be useful to install this package on the system before the class runs.

38.3 Remote access to the system

As mentioned during my stay, it could be useful for me to access the system remotely, possibly using ssh or through a modem connexion.

39. Hints and tips

39.1 Soft limits

We found (the hard way) that the Fortran runtime library now enforces the soft limit for the stack. This is a new behavior that needs to be taken into account. Instead of requiring the system administrator to change the soft limits for every newly created user, we added default limits in /etc/security/limits.

39.2 Core files

To limit the size of the core files when ALADIN crashes, one may use the following environment variable:

$ export MP_COREFILE_FORMAT=core

This is implemented in the ~nwp001/bin/mpirun script. To recover the regular behavior, just unset this variable.

40. Parallel make command

To speed up the compilation of complex packages, we can make use of a nice feature of the GNU make command. From the man gmake command, we get:

> -j jobs
>
> Specifies the number of jobs (commands) to run simultaneously. If there is more than one -j option, the last one is effective. If the -j option is given without an argument, make will not limit the number of jobs that can run simultaneously.

For example, to simultaneously build four targets, use:

$ gmake -j 4

CONTENTS