

Εργαστήριο Φυσικής Της Ατμόσφαιρας
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Οδηγός χρήσης libRadtran.

Χρήση libRadtran σε περιβάλλον
GNU/Linux, Windows και PBS-Torque cluster,
με πολλαπλές/παράλληλες εκτελέσεις.

Νάτσης Αθανάσιος
natsisthanasis@gmail.com

Περιεχόμενα

	3
1 Δημιουργία εκτελέσμου (build/compile) της Libradtran 2.0 σε GNU/Linux.	4
1.1 Εγκατάσταση βιβλιοθηκών (libraries) συστήματος.	4
1.2 Εγκατάσταση του <code>uvspec</code> στο σύστημα.	4
1.3 Εκτέλεση του <code>uvspec</code>	4
2 Παράλληλη εκτέλεση του ‘<code>uvspec</code>’ σε Bash shell.	6
2.1 Παραλληλοποίηση με τη χρήση της εντολής <code>xargs</code>	6
2.2 Παραλληλοποίηση με τη χρήση subshells στο background και της εντολής <code>wait</code>	9
2.3 Παραλληλοποίηση με τη χρήση του GNU <code>parallel</code>	12
3 Εκτέλεση της libRadtran σε HPC cluster.	16
3.1 Εισαγωγή	16
3.2 Ανάθεση της εκτέλεσης στο PBS grid.	16
3.3 Επεξεργασία των αποτελεσμάτων.	24
3.4 Δημιουργία λίστα παραμέτρων προς εκτέλεση.	29
4 Διαχείριση αποτελεσμάτων.	32
5 Οδηγός Εγκατάστασης της LibRadtran σε περιβάλλον Windows.	34
5.1 Εγκατάσταση του λειτουργικού περιβάλλοντος ‘Cygwin’	34
5.2 Εγκατάσταση της LibRadtran.	45
6 Χρήσιμες πληροφορίες για τη χρήση της libRadtran.	49
6.1 Μονάδες μέτρησης ηλιακού φάσματος.	49
6.2 Έξοδος του <code>uvspec</code>	49
6.3 Τυπικά ατμοσφαιρικά προφίλ.	50
6.4 Προειδοποίηση για τη γωνία $SZA = 43.2^\circ$	50
A' Παράρτημα	51
A'.1 <code>man pbsnodes</code>	51
A'.2 <code>man qstat</code>	53
A'.3 <code>man qsub</code>	61
Αναφορές	79

Η εργασία αυτή διανέμεται υπό την άδεια:

Creative Commons - Αναφορά Δημιουργού - Μη Εμπορική Χρήση - Παρόμοια

Διανομή 4.0 Διεθνές. <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Η στοιχειοθεσία έγινε με: bookdown, Rmarkdown, Pandoc, X_ET_EX.

Τελευταία τροποποίηση: Νάτσης Αθανάσιος 2019-01-28

Build: 7e31c353

Για διορθώσεις και προσθήκες επικοινωνήστε με τον Νάτση Αθανάσιο.

Σημείωση: Ο πηγαίος κώδικας (source code) που παρατίθεται εδώ είναι ένα παράδειγμα εφαρμογής κάποιων τεχνικών. Σε περίπτωση που θέλετε να τον χρησιμοποιήσετε, προτείνουμε να δοκιμαστεί η καταλληλότητα του για την επιθυμητή χρήση, πρώτα σε μη σημαντικές εφαρμογές (non-critical). Τα παραδείγματα κώδικα εδώ, διαφέρουν από αυτά που χρησιμοποιούμε, λόγω της συνεχής εξέλιξης και βελτίωσης κατά την χρήση τους.

1 Δημιουργία εκτελέσιμου (build/compile) της Libradtran 2.0 σε GNU/Linux.

Το παράδειγμα αυτό έχει εφαρμοστεί σε λειτουργικό σύστημα Debian και βασίζεται στις οδηγίες που βρίσκονται στις ιστοσελίδες www.libradtran.org/doku.php?id=download και www.libradtran.org/doc/INSTALL.

1.1 Εγκατάσταση βιβλιοθηκών (libraries) συστήματος.

Εκτός από τα εργαλεία που περιγράφονται στις οδηγίες, είναι αναγκαίες και κάποιες άλλες βιβλιοθήκες για να μπορεί να γίνει compile του εκτελέσιμου uvspec. Η εγκατάστασή τους μπορεί να γίνει με την εντολή:

```
sudo apt-get install libgs10* netcdf-* libnetcdf*
```

1.2 Εγκατάσταση του uvspec στο σύστημα.

Το εκτελέσιμο που δημιουργήθηκε μπορεί να γίνει διαθέσιμο στο σύνολο του συστήματος βάζοντας ένα link της θέσης του στο φάκελο των αρχείων συστήματος.

```
sudo ln -s '/path/to/libRadtran-2.0-64/bin/uvspec' '/usr/bin/uvspec'
```

Το βήμα αυτό δεν είναι απαραίτητο, αλλά διευκολύνει την εκτέλεση του αρχείου από οποιοδήποτε σημείο του συστήματος χωρίς την ανάγκη να χρησιμοποιούμε ολόκληρη τη διαδρομή (full path) του αρχείου.

1.3 Εκτέλεση του uvspec.

Κατά την εκτέλεση του uvspec, το αρχείο χρειάζεται να διαβάσει μια σειρά από παραμετρικά αρχεία που βρίσκονται στον φάκελο της libRadtran. Η θέση των αρχείων αυτών μπορεί να δοθεί ως μία παράμετρος περιβάλλοντος (environmental variable). Αυτό γίνεται είτε ορίζοντας την πριν την εκτέλεση, είτε βάζοντας την στη ρυθμίσεις του shell του συστήματος.

```
export LIBRADTRAN_DATA_FILES='/location/of/libRadtran-2.0-32/data/'
```

Κατά την εκτέλεση του uvspec είναι πρακτικό να χρησιμοποιούμε ένα κατάλληλα προετοι-

μασμένο αρχείο εισόδου (.inp) και να αποθηκεύουμε τα αποτελέσματα σε ένα αρχείο εξόδου (.out). Το πετυχαίνουμε αυτό κάνοντας χρήση της δυνατότητας ‘redirect’ της εισόδου (stdin) και εξόδου (stdout) του εκτελέσιμου.

```
uvspec < '/input/location/clear_atm.inp' > '/output/location/clear_atm.out'
```

2 Παράλληλη εκτέλεση του ‘uvspec’ σε Bash shell.

Οι παρακάτω μέθοδοι έχουν δοκιμαστεί σε περιβάλλον GNU/Linux και Windows (Cygwin). Θεωρούμε ότι η εκτέλεση γίνεται σε έναν υπολογιστή του οποίου θέλουμε να αξιοποιήσουμε όλους ή μερικούς από τους πυρήνες του επεξεργαστή του. Με κάποιες τροποποιήσεις και σχεδίαση, οι λύσεις αυτές μπορούν να εφαρμοστούν ταυτόχρονα και σε περισσότερους υπολογιστές, δεν θα αναφερθούμε ιδιαίτερα σε αυτό διότι υπάρχουν εξειδικευμένα εργαλεία για αυτή την δουλειά, όπως: [GNU parallel](#) (shell), [PPSS](#) (shell), [HTCondor](#), [dispy](#) (python), [snow](#) (R) κ.λ.π.

Και στις δύο περιπτώσεις, πρέπει να υπάρχουν έτοιμα τα αρχεία εισόδου (.inp) της libRadtran. Ο τρόπος παραγωγής τους, δεν θα μας απασχολήσει, καθώς μπορεί να γίνει με τα προτιμώμενα εργαλεία του χρήστη.

2.1 Παραλληλοποίηση με τη χρήση της εντολής xargs.

Σε αυτήν την περίπτωση χρησιμοποιούμε δύο Bash script, το πρώτο (xargs_uvspec_worker.sh) είναι υπεύθυνο για την μία εκτέλεση του uvspec με τις κατάλληλες παραμέτρους (arguments). Και το άλλο (xargs_parallel.sh), έχει την οργάνωση της παρτίδας εργασιών που θα εκτελεστούν από την εντολή xargs. Η εντολή xargs έχει την δυνατότητα να παρακολουθεί, μεταξύ άλλων, την χρήση των πυρήνων του επεξεργαστή, καθώς και να ελέγχει την εκτέλεση πολλαπλών εντολών.

2.1.1 Αρχείο xargs_uvspec_worker.sh

```
#!/bin/bash
## Worker to run one instance of uvspec, this is to be used by another script

## get arguments
OUTDIR="${1}"
ERRDIR="${2}"
total="${3}"
Tic="${4}"
INPUTF="${5}"
cntt="${6}"
```

```

## file to log this run
logfile="/path/to/a/log/file/JOB_$(date + "%F").log"

## set libradtran executable path
UVSPEC="/path/to/uvspec"

## check how many arguments
if [ $# -ne 6 ] ; then echo " 6 arguments needed" ; exit 1 ; fi

## input base file name
fname=$(basename $INPUTF)
## out and error file names
OUTFIL="${OUTDIR}/${fname%.*}.out"
ERRFIL="${ERRDIR}/${fname%.*}.err"

## print some info while running
TOT=$(echo "scale=1; ($cntt*100/$total)" | bc)
ETA=$((($((($total-cntt))*$((($date +%s%N)-Tic))/6000000000))/cntt))
printf " %5s %5s/$total %5s%%  ETA: %4s min\n" $((total-cntt)) $cntt $TOT $ETA

## keep a log of what happened
echo "$(date + "%F %T") $fname $cntt" >> "${logfile}"

##### HERE IS THE HEAVY LOAD #####
#####TEST##### First try this to check what will run
echo "(( ${UVSPEC} < ${INPUTF} ) | gzip > ${OUTFIL}.gz" 2> ${ERRFIL}"
sleep $((RANDOM%5+2))

## Then use this to run the load
#(( ${UVSPEC} < ${INPUTF} ) | gzip > ${OUTFIL}.gz ) 2> ${ERRFIL}
exit 0

```

2.1.2 Αρχείο xargs_parallel.sh

```

#!/bin/bash
## Executioner of uvspec worker, this is used to run multiple script instances

```

```

## EDIT all paths to full paths

## this will run a uvspec
WORKER_sh="./xargs_uvspec_worker.sh"
## I/O folders
INPDIR="/path/to/files/for/INPUT/"
OUTDIR="/path/to/files/for/OUTPUT/"
ERRDIR="/path/to/files/for/error/"

## Cores to use
cores=8

#####TEST##### DELETE THIS TEST VARIABLE
INPDIR="/home/.../LibRadTranM/clear_H2O_LAP/DATA"

## initial files count
total=$(find "${INPDIR}" -type f -iname "*.inp" | wc -l)
## ask to continue
echo "" ; input=0
echo -n "Found $total input files continue (y/n)?:" "
read -n 1 input ; echo
if [ "$input" == "y" -o "$input" == "Y" ] ; then
    printf ""
else
    echo "exit now.."; exit 2
fi

## set some variables
Tic=$(date +%s%N)      ## keep time
Tac=$(date +%"F %T")   ## keep time
cntt=0

##### THIS IS THE PARALLEL TRICK #####
## run all input files through the WORKER_sh
find "${INPDIR}" -type f -iname "*.inp" | sort | while read line;do
    echo "$line" ${((++cntt))}"
done | xargs -n 2 -P "$cores" "$WORKER_sh" "${OUTDIR}" \
"${ERRDIR}" "$total" "$Tic"

## you are done, print end report

```

```

T=$(($(date +%s%N)-Tic))
S=$((T/1000000000))
M=$((T%1000000000/1000000))
echo ""
echo "      ____UVSPEC_runs_finished____"
printf "DONE in:          %02d %02d:%02d.%03d <\n" \
      "$((S/86400))" "$((S/3600%24))" "$((S/60%60))" "$((S%60))" "${M}"
echo "From : $Tac"
echo "Until : $(date +"%F %T")"
exit 0

```

Σημείωση: Τα προηγούμενα πιθανότατα θα τρέξουν στον mistral.

Προσοχή: Ακόμα και αν σταματήσει το κύριο script, οι workers θα συνεχίσουν να τρέχουν, οπότε πρέπει να ξέρετε πως μπορεί να σταματήσει ένα script που τρέχει στο background!

2.2 Παραλληλοποίηση με τη χρήση subshells στο background και της εντολής wait.

Εδώ κάνουμε χρήση μιας απλής τεχνικής ελέγχου του αριθμού των uvspec που εκτελούνται από τον υπολογιστή. Η λογική που εφαρμόζουμε είναι, να μετρούμε τις εκτελέσεις που έχουν γίνει και να περιμένουμε να τελειώσει κάποια από αυτές πριν ξεκινήσουμε την επόμενη.

2.2.1 Αρχείο execute_in_subshells.sh

```

#!/bin/bash
## Simple parallel execution in bash
## Test it before use. This can not be stopped with 'ctr+C'.
## All processes are send to the background.

## libradtran executable
UVSPEC="/path/to/uvspec"
## folders
INPDIR="/path/to/files/for/INPUT/"
OUTDIR="/path/to/files/for/OUTPUT/"
ERRDIR="/path/to/files/for/error/"

```

```

LBRDAT="/path/to/libradtran/data/folder/data"
## file to log this run
logfile="/path/to/a/log/file/JOB_$(date +"%F_%T").log"
## parameters
cores=4      ## cores to use
prs=1        ## counter of concurrent processes

## ensure folders exist
mkdir -p "${OUTDIR}"
mkdir -p "${ERRDIR}"
## export libradtran data files path this may be redundant
export LIBRADTRAN_DATA_FILES="${LBRDAT}"

## count files
total=$(find "${INPDIR}" -type f -iname "*.inp" | wc -l)
## ask to continue
echo "" ; input=0
echo -n "Found $total input files continue (y/n)?: "
read -n 1 input ; echo
if [ "$input" == "y" -o "$input" == "Y" ] ; then
    printf ""
else
    echo "exit now.."; exit 1
fi

cntt=0          ## count total runs
((cores--))     ## start from zero
Tic=$(date +%-s%N)      ## keep time
Tac=$(date +"%F %T") ## keep time

## list all input files
find "${INPDIR}" -type f -iname "*.inp" | while read line; do
    ((cntt++))           ## increase count
    fname=$(basename "$line")      ## input file-name
    INPUT="$line"          ## input file-name full path
    OUTPUT="${OUTDIR}/${fname%.*}.OUT" ## output file-name full path
    ERROR="${ERRDIR}/${fname%.*}.err" ## error file-name full path
    ## print some info
    TOT=$(echo "scale=1; ($cntt*100/$total)" | bc)
    ETA=$(((($((($total-cntt))*$((($date +%-s%N)-Tic))/60000000000))/cntt))
    printf " %5s %5s/$total %5s% prs: %2s    ETA: %s min\n" \

```

```

    $((total-cntt)) $cntt $TOT $prs $ETA
## keep a log of what happened
echo "$(date +"%F %T") $fname $cntt" >> "${logfile}"

##### uncomment to choose output with gzip compression
# ( ( ${UVSPEC} < ${INPUT} ) | gzip > ${OUTPUT}.gz" ) 2> $ERROR ) &

##### uncomment for output without compression
# ( ( ${UVSPEC} < ${INPUT} ) > ${OUTPUT} ) 2> $ERROR ) &

##### uncomment for output without compression and
##### export libradtran data path this may be redundant
# ( ( export LIBRADTRAN_DATA_FILES=${LBRDAT}
#       ${UVSPEC} < ${INPUT} ) > ${OUTPUT} ) 2> $ERROR ) &

## Throttle execution. This keeps script from running everything at once!
if (( ++prs > cores )); then
    wait
    ((prs--))
fi
done

## wait for the last of the runs after the loop ends
for i in $(seq 1 $cores); do; wait; done
## print end report
T=$(($(date +%s%N)-Tic))
S=$((T/1000000000))
M=$((T%1000000000/1000000))
echo ""
echo "      ____UVSPEC_runs_finished____"
printf "DONE in:          %02d %02d:%02d:%02d.%03d <\n" \
      "$((S/86400))" "$((S/3600%24))" "$((S/60%60))" "$((S%60))" "${M}"
echo "From : $Tac"
echo "Until : $(date +"%F %T")"
exit 0

```

Σημείωση: Αυτό πιθανότατα δεν θα τρέξει στον mistral λόγω διαφορών στις εκδόσεις της wait. Δουλεύει σε διανομές Debian.

Προσοχή: Για να το σταματήσετε πρέπει να τερματίσετε όλες τις διεργασίες που γίνονται στο *background* και όχι μόνο αυτό το *script* (*ctrl+C*). Γι' αυτό πρέπει να ξέρετε πώς τερματίζονται εργασίες στον *background*.

2.3 Παραλληλοποίηση με τη χρήση του GNU parallel.

Με το *parallel* (GNU *parallel*) μπορούμε να χρησιμοποιήσουμε πολλαπλούς πυρήνες ενός υπολογιστή, αλλά και περισσότερους από έναν υπολογιστή (cluster). Το *parallel* είναι στην ουσία ένας διαχειριστής εκτελέσεων (job scheduler). Τα επόμενα παραδείγματα είναι ίσως η πιο απλοποιημένη παραμετροποίηση που μπορούμε να κάνουμε. Για πιο σύνθετες περιπτώσεις δείτε τα εγχειρίδια των προγραμμάτων.

2.3.1 Παραμετροποίηση cluster.

Για να έχει πρόσβαση το *parallel* στους υπόλοιπους υπολογιστές, χρειάζεται πρώτα να ρυθμίσουμε το *ssh* ώστε να μπορεί να βρει τους υπόλοιπους υπολογιστές στο δίκτυο.

2.3.1.1 Configure network.

Οι υπολογιστές μπορεί να βρίσκονται στο τοπικό μας δίκτυο ή να έχουν σταθερή διεύθυνση IP, οπότε χρειάζεται ελάχιστη ρύθμιση. Είτε, να έχουμε ρυθμισμένο κάποιο VPN ώστε να μην χρειάζεται να μας απασχολεί η περίπτωση της μη σταθερής διεύθυνση IP ή της μετακίνησης του υπολογιστή σε άλλο δίκτυο (π.χ. laptop).

2.3.1.2 Configure ssh.

Πρέπει να έχουμε πρόσβαση σε κάθε υπολογιστή μέσω *ssh* και τη χρήση κλειδιού χωρίς την ανάγκη εισαγωγής κωδικού. Η διαδικασία ρύθμισης είναι εύκολο να βρεθεί στο *internet* ψάχνοντας π.χ. “passwordless authentication ssh keys”. Μετά από αυτό πρέπει να μπορείτε να έχετε πρόσβαση σε κάθε έναν από τους υπολογιστές εκτελώντας την εντολή π.χ. *ssh user@155.207.10.10 -i .ssh/libradtran_cl.pri* χωρίς να εισάγεται κωδικό. Όπου “155.207.10.10” είναι η θέση του υπολογιστή στο δίκτυο και “libradtran_cl.pri” το αρχείο του προσωπικού κλειδιού πρόσβασης (private key).

Για να οργανώσουμε το cluster χρειάζεται να συμπληρώσουμε ένα παραμετρικό αρχείο του

ssh όπου θα αναφέρονται όλοι οι υπολογιστές που θέλουμε να χρησιμοποιήσουμε (default file .ssh/config). Αν όλα γίνουν σωστά θα μπορούμε να συνδεόμαστε μέσω ssh δίνοντας μόνο το όνομα του υπολογιστή π.χ. ssh machine_1. Αυτό, επίσης βοηθάει πολύ, στη γενικότερη αντιμετώπιση των προβλημάτων που μπορεί να προκύψουν, αλλά και στη παραμετροποίηση του κάθε υπολογιστή για την δουλειά που σκοπεύουμε να τρέξουμε.

Παράδειγμα αρχείου .ssh/config

```
Host machine_1
    HostName 155.207.10.10
    User user
    IdentityFile ~/.ssh/machine1.pri
```

```
Host home_pc
    HostName 10.10.10.1
    User athan
    IdentityFile ~/.ssh/libradtran_cl.pri
```

```
Host laptop
    HostName 10.10.10.2
    User athan
    IdentityFile ~/.ssh/libradtran_cl.pri
```

2.3.2 Παραμετροποίηση του parallel

Αφού ο κεντρικός υπολογιστής έχει πρόσβαση σε όλους του άλλους μέσω ssh, αρκεί ένα απλό παραμετρικό αρχείο για το parallel, με του υπολογιστές που θέλουμε να χρησιμοποιήσουμε στο cluster.

Στο παρακάτω αρχείο, δηλώνουμε ότι το cluster θα έχει τέσσερις υπολογιστές. Με το σύμβολο : είναι ο κεντρικός υπολογιστής, όπου οι πυρήνες του αναγνωρίζονται αυτόματα. Ο αριθμός πριν το όνομα του υπολογιστή δηλώνει τους πυρήνες του επεξεργαστή που θα είναι διαθέσιμοι από τον κάθε υπολογιστή.

Παράδειγμα αρχείου .parallel/hosts

```
:
4/machine_1
2/home_pc
```

2.3.3 Χρήση parallel

Ένα παράδειγμα εκτέλεσης του parallel. Η χρήση των παραμέτρων αναλύεται στο manual της εντολής. Εδώ χρησιμοποιούμε έναν πυρήνα από κάθε υπολογιστή του cluster, όπου θα τρέξουμε τις εντολές που βρίσκονται στο αρχείο jobs.list. Όπου το αρχείο optimise_worker_v1.R παίρνει δύο παραμέτρους που τις χρησιμοποιεί για να τρέξει ένα εύρος παραμέτρων από μία άλλη λίστα εργασιών. Έτσι έχουμε χωρίσει την συνολική εργασία σε εκτελέσεις 100 μικρότερων εργασιών που αναθέτουμε σε κάθε υπολογιστή του cluster.

```
parallel      \
  --jobs 1      \
  --progress    \
  --eta         \
  --results   /home/athan/Aerosols_03/DATA/par.out      \
  --joblog    /home/athan/Aerosols_03/Libradtran_modeling/par.resume.file \
  --sshloginfile ~/.parallel/hosts \
  -a /home/athan/Aerosols_03/Libradtran_modeling/jobs.list
```

Αρχείο jobs.list.

```
/home/athan/Aerosols_03/Libradtran_modeling/optimise_worker_v1.R 1 100
/home/athan/Aerosols_03/Libradtran_modeling/optimise_worker_v1.R 101 200
/home/athan/Aerosols_03/Libradtran_modeling/optimise_worker_v1.R 201 300
/home/athan/Aerosols_03/Libradtran_modeling/optimise_worker_v1.R 301 400
/home/athan/Aerosols_03/Libradtran_modeling/optimise_worker_v1.R 401 500
/home/athan/Aerosols_03/Libradtran_modeling/optimise_worker_v1.R 501 600
/home/athan/Aerosols_03/Libradtran_modeling/optimise_worker_v1.R 601 700
/home/athan/Aerosols_03/Libradtran_modeling/optimise_worker_v1.R 701 800
/home/athan/Aerosols_03/Libradtran_modeling/optimise_worker_v1.R 801 900
/home/athan/Aerosols_03/Libradtran_modeling/optimise_worker_v1.R 901 1000
```

2.3.4 Συγχρονισμός/μεταφορά αρχείων.

Ανάλογα με τη λογική της εφαρμογής της παραλληλοποίησης, είναι λιγότερο η περισσότερο απαραίτητο να μετακινούνται αρχεία μεταξύ των υπολογιστών. Αυτά τα αρχεία μπορεί

να είναι αρχεία προς επεξεργασία, εκτελέσιμα ή αρχεία αποτελεσμάτων. Το parallel έχει κάποιες τέτοιες δυνατότητες, αλλά είναι ίσως πιο εύκολο να χρησιμοποιηθεί κάποια πιο εξειδικευμένη εφαρμογή. Δεν θα αναλύσουμε την εφαρμογή τους αλλά θα δώσουμε κάποιες ιδέες.

unison: Αμφίδρομα συγχρονίζει αρχεία μεταξύ υπολογιστών (όσα χρειάζονται), πολλές δυνατότητες παραμετροποίησης.

rsync: Συγχρονίζει αρχεία μεταξύ υπολογιστών (όσα χρειάζονται), βελτιστοποιημένο στη μεταφορά αρχείων.

owncloud: Cloud server απόθεσης αρχείων, μπορεί να εγκατασταθεί σε οποιονδήποτε υπολογιστή και οι υπόλοιποι να συγχρονίζονται από αυτόν.

nfs: Δίσκος δικτύου, στον οποίο συνδέονται όλοι οι υπολογιστές και τον χρησιμοποιούν ως τοπικό.

Προφανώς υπάρχουν και άλλοι τρόποι, και όποια επιλογή εξαρτάται από την διάρθρωση του δικτύου, τους διαθέσιμους πόρους και τον επιθυμητό στόχο.

3 Εκτέλεση της libRadtran σε HPC cluster.

3.1 Εισαγωγή

Εάν έχουμε την δυνατότητα, να χρησιμοποιήσουμε το HPC grid του πανεπιστημίου, μπορούμε να εκτελέσουμε πολύ μεγάλο όγκο προσομοιώσεων της ακτινοβολίας στην ατμόσφαιρα με την libRadtran, σε μικρό χρονικό διάστημα. Για τον σκοπό αυτό φτιάξαμε μερικά εργαλεία, τα οποία βοηθούν στη χρήση του συστήματος Portable Batch System (PBS/Torque) που ελέγχει την εκτέλεση και τον καταμερισμό των εργασιών στο grid.

Γενικά, θα αναφερόμαστε στην μία εκτέλεση του `uvspec` (εκτελέσμιο αρχείο της libRadtran) ως μία ανεξάρτητη ‘εργασία’ (job), αφού αυτή εξαρτάται μόνο από μία προκαθορισμένη ομάδα παραμέτρων. Αυτές οι ομάδες παραμέτρων αποτελούν μία λίστα πολλαπλών εργασιών τις οποίες αναθέτουμε στο PBS να εκτελέσει. Θα αναφερόμαστε σε αυτές ως παρτίδα εργασιών (batch job).

Μπορούμε να χωρίσουμε την όλη διαδικασία σε τρεις φάσεις.

1. Δημιουργία της λίστα παραμέτρων προς εκτέλεση.
2. Ανάθεση της εκτέλεσης στο PBS.
3. Επεξεργασία/διαχείριση των αποτελεσμάτων.

Ανάλογα με τις ανάγκες, οι φάσεις αυτές επαναλαμβάνονται κυκλικά μέχρι να έχουμε τα επιθυμητά δεδομένα. Γι' αυτό, είναι καλό να λάβουμε κάποια μέτρα κατά την προετοιμασία της ακολουθίας των φάσεων (workflow) ώστε να αποφύγουμε προβλήματα και να διευκολύνουμε την παραγωγή και διαχείριση των αποτελεσμάτων. Δεν θα αναφερθούμε ιδιαίτερα σε αυτό, αλλά έχουμε φροντίσει ώστε να υπάρχει επαρκές logging και παρακολούθηση σε όλα τα στάδια της διαδικασίας ώστε να μπορούν να εντοπιστούν εύκολα τα προβλήματα και να διορθωθούν.

3.2 Ανάθεση της εκτέλεσης στο PBS grid.

3.2.1 Εκτέλεση της εντολής `uvspec` (libRadtran) ως μία εργασία (job).

Χρησιμοποιούμε ένα script (`LBT_PBS.sh`) σε Bash το οποίο αναλαμβάνει να καλέσει το εκτελέσμιο `uvspec` με τις παραμέτρους που του δίνει το σύστημα PBS. Αυτό το αρχείο θα εκτελεστεί σε κάθε πυρήνα, από τους διαθέσιμους και είναι υπεύθυνο για την ολοκλήρωση μίας

προσομοίωσης ακτινοβολίας του μοντέλου.

Οι παράμετροι (arguments) που δέχεται αυτό το script είναι τέσσερις.

1. **Libradtran file path:** ο φάκελος όπου βρίσκονται τα αρχεία της libRadtran.
2. **Libradtran output folder:** ο φάκελος όπου θα αποθηκευτούν τα παραγόμενα αρχεία.
3. **unique id:** μοναδικός δείκτης για να χρησιμοποιηθεί στο όνομα των αρχείων κάθε εργασίας.
4. **serialized string with uvspec options:** μία λέξη με όλες τις παραμέτρους που χρειάζεται να ορίσουμε σε μία εκτέλεση του uvspec.

Ο τρόπος που έχουμε επιλέξει για την αποστολή των παραμέτρων στο uvspec είναι η γραμμικοποίησή τους (serialization). Αυτό επιτρέπει την χρήση οποιοδήποτε κατάλληλων παραμέτρων της libRadtran χωρίς καμία τροποποίηση αυτού του αρχείου. Αυτό το πετυχαίνουμε, έχοντας κωδικοποιήσει την ερμηνεία του συμβόλου @ ως νέα γραμμή και του @@ ως τον χαρακτήρα του κενού. Με αυτόν τον τρόπο κάθε αρχείο εισόδου (.inp) μπορεί να σταλεί ως μία λέξη. Για παράδειγμα, η λέξη:

```
atmosphere_file@@aattmmoo=afglms.dat@source@@solar@@ssoolla=kurudz_1.0nm.  
dat@mol_modify@@03@@290@@DU@albedo@@0.05@sza@@26@altitude@@0.062694168@rte  
_solver@@sdisort@number_of_streams@@6@wavelength@@250@@5025@pseudospherica  
l@quiet@
```

θα μετατραπεί στο αρχείο εισόδου (.inp):

```
atmosphere_file /mnt/.../libRadtran-2.0.1/data/atmmod/afglms.dat  
source solar /mnt/.../libRadtran-2.0.1/data/solar_flux/kurudz_1.0nm.dat  
mol_modify 03 290 DU  
albedo 0.05  
sza 26  
altitude 0.062694168  
rte_solver sdisort  
number_of_streams 6  
wavelength 250 5025  
pseudospherical  
quiet
```

3.2.1.1 Αρχείο LBT_PBS.sh.

Στο πρώτο κομμάτι του script ελέγχουμε αν όλες οι παράμετροι έχουν δοθεί κανονικά από το PBS και προετοιμάζουμε τα ονόματα των αρχείων με τη πλήρη τους διαδρομή στο δίσκο.

```
#!/bin/bash

libpath=$1    ## Libradtran file path .../libRadtran-2.0/
workdir=$2    ## Libradtran output folder for this model family
jobid=$3      ## unique id for this run
options=$4    ## serialized string with uvspec options

## expand file paths
UVSPEC="${libpath}/bin/uvspec"
DATA="${libpath}/data"
WPTEMP="${workdir}/clearwaterLAP"

## have libRadtran path?
if [ -z "$libpath" ]; then
    echo "Empty variable 'libpath'"
    exit 2
fi

## check executable location
if [ ! -f "$UVSPEC" ]; then
    echo "Can not find uvspec" ; exit 3
fi

## check data folder location
if [ ! -d "$DATA" ]; then
    echo "Can not find data folder $DATA" ; exit 4
fi

## have working dir path?
if [ -z "$workdir" ]; then
    echo "Empty variable 'workdir'" ; exit 5
fi

## check working folder location
if [ ! -d "$workdir" ]; then
    echo "Can not find working folder $workdir" ; exit 6
fi

## have jobid?
if [ -z "$jobid" ]; then
    echo "Empty variable 'jobid'" ; exit 7
fi

## have libratran options?
if [ -z "$options" ]; then
    echo "Empty variable 'options'" ; exit 8
```

```

fi
## create working folder
mkdir -p "${WPTEMP}"
## files for this job
INPUT="${WPTEMP}/LBT_${jobid}.inp"
OUPUT="${WPTEMP}/LBT_${jobid}.out"
ERPUT="${WPTEMP}/LBT_${jobid}.err"

```

Στη συνέχεια μορφοποιούμε τις παραμέτρους της libRadtran σε ένα κατάλληλο αρχείο “.inp”.

```

## create input file
( echo $options | sed 's/@@/ /g' | sed 's/@/\n/g' | while read line ;do
  echo $line | sed "s@aattmmoo=@${DATA}/atmmmod/@g" \
    | sed "s@ssoolla=@${DATA}/solar_flux/@g"
done ) > $INPUT

```

Εκτελούμε το uvspec παραπέμποντας τα κατάλληλα αρχεία (εισόδου, εξόδου και σφαλμάτων) και όταν τελειώσει η εκτέλεσή του συμπλέζουμε το αρχείο εξόδου. Η συμπίεση βοηθάει στο να μειωθεί το μέγεθος των αρχείων αποτελεσμάτων αλλά και στη μεταφορά τους από τον υπολογιστή που έκανε την εκτέλεση, στον υπολογιστή ελέγχου.

```

##### ready to run uvspec
tic=$(date +"%s")
loa=$(uptime | grep -o "load .*")
export LIBRADTRAN_DATA_FILES=${DATA}
( $UVSPEC < $INPUT > $OUPUT ) >& $ERPUT
wait; wait
gzip -f $OUPUT
tac=$(date +"%s")

```

Αποθηκεύουμε κάποιες επιπρόσθετες πληροφορίες σχετικά με τον χρόνο εκτέλεσης και το σύστημα στο οποίο έγινε, ώστε να τις χρησιμοποιήσουμε σε περίπτωση σφαλμάτων.

```

## helpful info for this run and uvspec error collector
( echo $loa
  uptime | grep -o "load .*"
  echo "hostname=$(hostname)"
  date +"%F %T"
  echo $tic
  echo $tac
) >> $ERPUT
exit 0

```

3.2.2 Τροφοδοσία παρτίδας εργασιών στο grid.

Για να αναθέσουμε μία παρτίδα εργασιών (batch job), χρησιμοποιούμε ένα παραμετρικό αρχείο (submit.pbs) το οποίο περιέχει και ορίζει τις παραμέτρους εκτέλεση της παρτίδας. Εδώ, καθορίζονται οι παράμετροι που ελέγχουν τον τρόπο εκτελεσης της λίστα εργασιών. Για παράδειγμα, το πώς θα γίνει η αναφορά κατά την εκτέλεση, η διαχείριση των λαθών (error handling) κλπ.

Το αρχείο αυτό έχει ταυτόχρονα την λειτουργία του παραμετρικού αρχείου για το PBS. Αλλά, και την λειτουργία Bash script που το εκτελεί το PBS προκειμένου να χρησιμοποιήσει το περιεχόμενό/αποτέλεσμα των εντολών του.

Το δείγμα αρχείου που παραθέτουμε μπορεί να χρησιμοποιηθεί, είτε για την μερική εκτέλεση/δοκιμή μιας παρτίδας εργασιών, ή για την πλήρη εκτέλεση μιας παρτίδας.

3.2.2.1 Αρχείο submit.pbs.

Οι σειρές που ξεκινούν με '#PBS' αφορούν παραμέτρους (arguments) που χρησιμοποιούνται από την εντολή qsub του PBS. Περισσότερες λεπτομέρειες στο manual της εντολής (man qsub).

```
#!/bin/bash
#PBS -N clearatm
#PBS -j oe
#PBS -q see
#PBS -M at...ys@gmail.com
#PBS -m abe
#PBS -l nodes=1:ppn=1
#PBS -o condorlog/run.log
#PBS -t 1-10
```

Οι μεταβλητές που ξεκινούν με PBS_* παίρνουν τιμή από το σύστημα PBS κατά την εκτέλεση των ανεξάρτητων εργασιών (jobs). Εδώ, ως εργασία νοείται η εκτέλεση του αρχείου LBT_PBS.sh με τις κατάλληλες παραμέτρους και φυσικά αυτό είναι που θα καλέσει τελικά το uvspec. Όλα τα υπόλοιπα καθορίζουν την ανάσυρση των παραμέτρων από την λίστα εργασιών και η καταγραφή της προόδου της παρτίδας σε αρχείο.

```
jobhomedir="/mnt/.../LibRadTranM/clear_water_pressure_ozone_LAP_meas/"
joblogfile="${jobhomedir}/condorlog/clearwatermeas_$(date +%F).log"
```

```

echo $PBS_O_WORKDIR >> "${joblogfile}"
cd "$jobhomedir"
## parse arguments from file
args=`sed -n "${PBS_ARRAYID} p" jobs_args.list`
arglist=($args)
arg1="/mnt/lapmg_a/.../libRadtran-2.0.1/"
arg2="/mnt/lapmg_a/.../LibRadTranM/clear_water_pressure_ozone_LAP_meas/"
arg3=${arglist[0]}
arg4=${arglist[1]}
(
    echo $(hostname) $PBS_O_HOST $PBS_SERVER $PBS_O_QUEUE
    echo ${PBS_O_WORKDIR}
    echo "JOBID =" ${PBS_ARRAYID}
    echo "arg1" ${arg1} ${arg2}
    echo "arg3"=${arg3}
    echo "arg4"=${arg4}
    echo "-----"
) >> "${joblogfile}"
## this is a job run
./LBT_PBS.sh $arg1 $arg2 $arg3 $arg4

```

Θεωρούμε ότι ο αναγνώστης, είτε είναι εξοικειωμένος με τη χρήση του Bash shell, είτε ότι μπορεί να αναζητήσει πληροφορίες για την χρήση και την ερμηνεία των παραπάνω εντολών στο διαδίκτυο.

3.2.3 Εκτέλεση μεγάλης παρτίδας

Προκειμένου να οργανώσουμε καλύτερα την κατάθεση (submission) κάποιας μεγάλης παρτίδας (1000+ jobs), και κατόπιν επικοινωνίας με τους διαχειριστές του grid. Για την καλύτερη εξυπηρέτηση των χρηστών του grid, γράψαμε ένα Bash script ώστε να κάνουμε την κατάθεση σε μικρότερα τμήματα και με κάποια χρονική απόσταση.

3.2.3.1 Αρχείο multisub.sh

Οι μεταβλητές ‘step’ και ‘watt’ καθορίζουν αντίστοιχα το μέγεθος των πακέτων εργασιών και τον χρόνου μεταξύ κάθε τμήματος. Οι τιμές τους σχετίζονται με τον χρόνο εκτέλεσης της libRadtran και τους διαθέσιμους πυρήνες. Οι τιμές εδώ, είναι ενδεικτικές για “σχετικά βαριές” εκτελέσεις της libRadtran. Σημειώνουμε ότι εδώ ο χρόνος αναμονής μεταξύ των πακέτων

ανεται γραμμικά.

```
#!/bin/bash
subfile="clearatm_submit.pbs"
total=4900
step=150
watt=40

## get total lines
total=$(cat jobs_args.list | wc -l)
echo
echo $subfile
echo Total: $total
echo step : $step
echo wait : $watt
echo "" ; input=0
echo -n "Submit (y/n)?:" "
read input
if [ "$input" == "y" -o "$input" == "Y" ] ; then
    echo "                Bombs Away...."
    echo
else
    echo
    echo "ABORT! ABORT! ABORT! ABORT!"
    echo
    exit 0
fi
echo
for ii in $(seq 0 $((total / step - 1))); do
    date -d"+$((watt*ii+1)) seconds" +"next at: %F %T "; echo
    sleep $((watt*ii+1))
    echo "running: qsub -t $((step*ii+1))-$((step*ii+step)) ${subfile}"
    qsub -t $((step*ii+1))-$((step*ii+step)) "${subfile}"
done
if [ $((total % step )) -gt 0 ];then
    date -d"+$((watt*ii+1)) seconds" +"next at: %F %T "; echo
    sleep $((watt*ii+1))
    echo "run: qsub -t $((step*ii+step+1))-$((step*ii+step+total%step)) ${subfile}"
    qsub -t $((step*ii+step+1))-$((step*ii+step+total%step)) "${subfile}"
fi
```

```
echo  
echo " * * *  submitting done  * * *"  
echo  
exit 0
```

3.2.4 Κοινές εντολές για την διαχείριση των παρτίδων εργασιών.

Εκτέλεση παρτίδας όπως καθορίζεται από τις παραμέτρους του αρχείου ‘submit.pbs’. Η σειρά '#PBS -t 1-10' στο αρχείο λέει στο PBS να εκτελέσει τις εργασίες 1 έως 10. Αυτός είναι ένας τρόπος να δοκιμάζονται μεγαλύτερες παρτίδες εργασιών ως προς την ορθότητα των ρυθμίσεών τους, πριν αφεθούν να τρέξουν στο σύνολό τους.

```
qsub submit.pbs
```

Ενας τρόπος να μπουν 100 εργασίες στη σειρά αναμονής προς εκτέλεση (queue).

```
qsub -t 301-400 submit.pbs
```

Λίστα με τις εργασίες στην ουρά (queue).

```
qstat -t
```

Λίστα με τις εργασίες που εκτελούνται.

```
qstat -e
```

Λίστα με τους υπολογιστές (nodes) όπου εκτελούνται οι εργασίες.

```
qstat -n1
```

Σταματάει όλες τις εργασίες της παρτίδας 4168020.

```
qdel 4168020
```

Σταμάτημα επιλεγμένων εργασιών της παρτίδας 4168020.

```
qdel 4168020[]
```

Συνεχείς τρόποι παρακολούθησης των εργασιών σε εκτέλεση/ουρά.

```
watch -10 ' qstat -t | grep " R " | wc -l ; \  
           qstat -t | grep " Q " | wc -l ; \  
           '
```

```

        qstat -t '
watch ' qstat -t | grep " R " | wc -l ; \
        qstat -t | grep " Q " | wc -l ; \
        qstat -t | grep " R " '

```

Διαθέσιμοι πόροι χωρίς τους εκτός λειτουργίας κόμβους.

```
pbsnodes -c
```

Λίστα όλων των κόμβων.

```
pbsnodes -l
```

Περισότερες πληροφορίες για τις παραπάνω εντολές μπορούν να βρεθούν στις αντίστοιχες σελίδες manual (`man pbsnodes`, `man qstat`, `man qsub`) ή στο Παράρτημα (pbsnodes: A'.1, qstat: A'.2, qsub: A'.3).

3.3 Επεξεργασία των αποτελεσμάτων.

Ο υπολογιστής από τον οποίο κάνουμε την ανάθεση των εργασιών έχει εγκατεστημένη την γλώσσα προγραμματισμού ‘R’. Θα την χρησιμοποιήσουμε για να συγκεντρώσουμε τα αποτελέσματα που μας ενδιαφέρουν.

Θα διαβάσουμε τα πρωτογενή δεδομένα από τα αρχεία που παράχθηκαν κατά την εκτέλεση της libRadtran και θα τα αποθηκεύσουμε σε ένα αρχείο δεδομένων της ‘R’. Με αυτόν τον τρόπο, αποφεύγουμε την μεταφορά μεγάλων ποσοτήτων δεδομένων, στις περιπτώσεις που μας ενδιαφέρουν μόνο κάποιες συγκεντρωτικές τιμές.

Παρακάτω παρατίθεται ένα παράδειγμα για το πώς μπορεί να επιτευχθεί αυτό. Δεν κρίνουμε σκόπιμο να αναλύσουμε τη λειτουργία του script διότι η διαδικασία αυτή μπορεί να γίνει με διαφορετικού τρόπους και χρησιμοποιώντας πολλές άλλες γλώσσες προγραμματισμού, ανάλογα με τις ανάγκες και τις δεξιότητες του χρήστη.

3.3.0.1 Αρχείο `parse_data.R`.

```

#!/usr/bin/env Rscript
##### Clear environment
#### -----

```

```

closeAllConnections()
rm(list = (ls()[ls() != ""]))
Sys.setenv(TZ = "UTC")
tic = Sys.time()

## folder for this model family
## setwd('/mnt/lapmg_a/.../LibRadTranM/lear_H2O_LAP') folder
## paths
jobsfolder = "clearwaterLAPmeas/"
datafolder = "DATA/"
archfolder = "DATA/ARCHIVED/"

if (file.exists(datafolder) & file.exists(archfolder) & file.exists(jobsfolder)) {
  cat("\nFolders exists\n")
} else {
  stop("Can not see all Folders")
}

## read libraries
source("../R_common/trapezUVSPEC.R")

## data output
datafile = "../clear_H2O_PR_03_meas_LAP.Rds"

## function to read inp files
input_parms <- function(inputfile) {
  if (!file.exists(inputfile))
    stop(paste("Missing file: ", inputfile))

  # read input file
  fcon = file(inputfile)
  lines = readLines(fcon)
  close(fcon)

  # 1
  atmosphere_file = basename(unlist(strsplit(grep("atmosphere_file",
    lines, value = TRUE), " +"))[2])
  atmosphere_file = unlist(strsplit(atmosphere_file, ".dat",
    fixed = T))[1]

  # 2
  source_solar = basename(unlist(strsplit(grep("source solar",
    lines, value = TRUE), " +"))[3])
  source_solar = unlist(strsplit(source_solar, ".dat", fixed = T))[1]

  # 3
  mol_modify_03 = as.numeric(unlist(strsplit(grep("mol_modify 03",

```

```

    lines, value = TRUE), " +"))[3])
# 4
albedo = as.numeric(unlist(strsplit(grep("albedo", lines,
    value = TRUE), " +"))[2])
# 5
sza = as.numeric(unlist(strsplit(grep("sza", lines, value = TRUE),
    " +"))[2])
# 6
altitude = as.numeric(unlist(strsplit(grep("altitude", lines,
    value = TRUE), " +"))[2])
# 7
rte_solver = unlist(strsplit(grep("rte_solver", lines, value = TRUE),
    " +"))[2]
# 8
number_of_streams = as.numeric(unlist(strsplit(grep("number_of_streams",
    lines, value = TRUE), " +"))[2])
# 9
wavelength = as.numeric(unlist(strsplit(grep("wavelength",
    lines, value = TRUE), " +"))[2:3])
wvlngth_min = wavelength[1]
wvlngth_max = wavelength[2]
# 10
pseudospherical = any(grepl("^ *pseudospherical", lines))
# 11
mol_modify_H2O = as.numeric(unlist(strsplit(grep("mol_modify H2O +[.0-9]+ +MM",
    lines, value = TRUE), " +"))[3])
# 12
pressure = as.numeric(unlist(strsplit(grep("pressure", lines,
    value = TRUE), " +"))[2])

aninput = data.frame(atmosphere_file = atmosphere_file, source_solar = source_solar,
    mol_modify_O3 = mol_modify_O3, mol_modify_H2O = mol_modify_H2O,
    albedo = albedo, sza = sza, altitude = altitude, pressure = pressure,
    rte_solver = rte_solver, number_of_streams = number_of_streams,
    wvlngth_min = wvlngth_min, wvlngth_max = wvlngth_max)
return(aninput)
}

## function to get integral of .out columns
output_read_trapz <- function(outputfile) {

```

```

if (!file.exists(outputfile))
  stop(paste("Missing file: ", outputfile))

## fmt: lambda edir edn eup uavgdir uavgdn uavgup
tempdata = read.table(outputfile)
get = trapezUVSPEC(tempdata)
data.frame(edir = get[1], edn = get[2], eup = get[3], eglo = get[1] +
  get[2])
}

## function to read .err files
error_param <- function(errfile) {
  if (!file.exists(errfile))
    stop(paste("Missing file: ", errfile))

  # read input file
  fcon = file(errfile)
  lines = readLines(fcon)
  close(fcon)

  ## start end
  minD = min(as.numeric(grep("^0-9]+$", lines, value = TRUE)))
  maxD = max(as.numeric(grep("^0-9]+$", lines, value = TRUE)))
  hosts = grep("hostname=.*", lines, value = TRUE)
  hosts = unlist(strsplit(hosts, "="))[2]
  if (hosts != "") {
    host = hosts
  } else {
    host = "unknown"
  }
  return(data.frame(hostname = host, ticTime = minD, tacTime = maxD))
}

##### Parsing starts here ##### read saved data or fail
saved_data <- readRDS(datafile)
# saved_data <- data.frame() # if there is no previous file
# read list of inp files
inpfiles = list.files(path =
# datafolder,
inpfiles = list.files(path = jobsfolder, pattern = "LBT_.*.inp",
  full.names = TRUE, recursive = FALSE)
if (length(inpfiles) < 1) stop("No input files found")

```

```

## check if all files matching
pause = FALSE
for (ii in inpfiles) {
  outputfile = paste0(unlist(strsplit(ii, split = ".inp"))[1],
    ".out.gz")
  errfile = paste0(unlist(strsplit(ii, split = ".inp"))[1],
    ".err")
  ## check output files
  if (!file.exists(outputfile)) {
    cat(paste("Missing file", outputfile), sep = "\n")
    pause = TRUE
  }
  ## check error files
  if (!file.exists(errfile)) {
    cat(paste("Missing file", errfile), sep = "\n")
    pause = TRUE
  }
}
outfiles = list.files(path = jobsfolder, pattern = "LBT_.*.out.gz",
  full.names = TRUE, recursive = FALSE)
for (ii in outfiles) {
  inpurfile = paste0(unlist(strsplit(ii, split = ".out.gz"))[1],
    ".inp")
  ## check input files
  if (!file.exists(inpurfile)) {
    cat(paste("Missing file", inpurfile), sep = "\n")
    pause = TRUE
  }
}
if (pause) {
  stop("Pause to manual clean files")
}
cat(paste(length(inpfiles), " files to read\n"))
## read data to a data frame
gather = data.frame()
ccc = 0
for (ii in inpfiles) {
  ccc = ccc + 1
  cat(paste(ccc, "/", length(inpfiles), " processed\n"))
}

```

```

outputfile = paste0(unlist(strsplit(ii, split = ".inp"))[1],
                   ".out.gz")
errfile = paste0(unlist(strsplit(ii, split = ".inp"))[1],
                 ".err")
record = cbind(input_parms(ii), output_read_trapz(outputfile),
                error_param(errfile))
gather = rbind(gather, record)
}

colall = c("atmosphere_file", "source_solar", "mol_modify_03",
          "albedo", "sza", "altitude", "rte_solver", "number_of_streams",
          "wvlngth_min", "wvlngth_max", "edir", "eglo", "edn", "eup",
          "mol_modify_H2O", "pressure")

colinp = c("atmosphere_file", "source_solar", "mol_modify_03",
          "albedo", "sza", "altitude", "rte_solver", "number_of_streams",
          "wvlngth_min", "wvlngth_max", "mol_modify_H2O", "pressure")

## combine old and new data
combined_data <- rbind(saved_data, gather)

## keep unique input combination
uniqueinqx <- !duplicated(combined_data[, colinp])
combined_data <- combined_data[uniqueinqx, ]

## check different results for same input
if (any(duplicated(combined_data[, colall]))) {
  stop("Possible different results for the same input")
}

# ## this will remove stored data!! saved_data = saved_data[
# saved_data$sza<50, ] saveRDS( saved_data, file = datafile,
# compress = 'xz') save this set of data as local repository
# combined_data <- data.frame() ## used to reset stored data
saveRDS(combined_data, file = datafile, compress = "xz")
cat(" now you can move files to ARCHIVED ")
# print(apply(combined_data[,c(1:10,17)], 2, unique))
str(combined_data)
summary(combined_data)

```

3.4 Δημιουργία λίστα παραμέτρων προς εκτέλεση.

Η λίστα παραμέτρων μίας παρτίδας εργασιών περιγράφει όλες τις συνθήκες με τις οποίες επιθυμούμε να γίνει η προσομοίωση της ακτινοβολίας στην ατμόσφαιρα.

Επιλέξαμε να δημιουργούμε αυτή τη λίστα σε δύο στάδια. Στην αρχή, καθορίζονται οι συνθήκες για τις οποίες επιθυμούμε να τρέξουμε την libRadtran και από αυτές κατασκευάζουμε όλα τα ενδεχόμενα. Στη συνέχεια, η λίστα των επιθυμητών ενδεχομένων συγκρίνεται με τα υπάρχοντα αποτελέσματα, ώστε να αποφευχθεί ο επαναυπολογισμός ήδη έτοιμων δεδομένων. Το αποτέλεσμα είναι μία νέα παρτίδα εργασιών που μπορεί να κατατεθεί στο grid για επεξεργασία.

Με αυτόν τον τρόπο, μπορούμε να ξεκινήσουμε με αρκετά αραιές επιθυμητές συνθήκες, και στη συνέχεια να αυξήσουμε την πυκνότητά τους ανάλογα με τις παραμέτρους για τις οποίες ενδιαφερόμαστε. Μοιράζοντας τον φόρτο εργασίας σε μικρότερα κομμάτια, έχοντας τη δυνατότητα να εκτιμήσουμε προοδευτικά τα αποτελέσματα που παίρνουμε από το μοντέλο και ελέγχοντας για την ύπαρξη προηγούμενων αποτελεσμάτων, μπορούμε να μειώσουμε τους άσκοπους υπολογισμούς.

Παρακάτω παραθέτουμε ένα script γραμμένο σε 'R' το οποίο προσπαθεί να δημιουργήσει έναν ορθογώνιο χώρο φάσεων. Εδώ, φαίνεται και ο τρόπος γραμμικοποίησης των παραμέτρων του `uvspec`. Δεν θα περιγράψουμε τη λειτουργία του καθώς το ίδιο αποτέλεσμα μπορεί να επιτευχθεί με οποιαδήποτε γλώσσα προγραμματισμού.

3.4.0.1 Αρχείο LBT_job_list_creationPBS.R

```
#!/usr/bin/env Rscript
##### Clear environment -----
closeAllConnections()
rm(list = (ls()[ls() != ""]))
Sys.setenv(TZ = "UTC")
tic = Sys.time()
library(bitops)
## folder for this model family
## setwd('/mnt/lapmg_a/.../LibRadTranM/clear_H2O_LAP')
datafile = ".../clear_H2O_LAP.Rds"
outfilelist = "jobs_args.list"
if (!file.exists(datafile)) {
  stop("No previous results found")
}
##### Options to create multiple jobs -----
wvlngth_min = 250
# wvlngth_max = 4000.0
wvlngth_max = 5025
```

```

# spline_min = 250 spline_max = 4000 spline_stp = 1
altitude = 62.694168/1000 ## for LAP 62.694168/1000.
number_of_streams = 6
rte_solver = "sdisort"
mol_modify_03 = seq(290, 370, 10) ## climatology or iterate
mol_modify_H2O = seq(2, 37, 10) ## MM : kg/m^2
## Min sza for thessaloniki ~17.20. At sza=100 something
## breaks !!!!
sza = unique(c(seq(15, 95, 10), seq(25, 90, 10), 0))
atmosphere_file = c("afglms", "afglmw")
source_solar = c("kurudz_1.0nm", "kurudz_0.1nm", "kurudz_full")
albedo = unique(c(seq(0.05, 0.15, 0.05), 0.07))

##### create combination of all options given -----
todolisting <- expand.grid(wvlngth_min = wvlngth_min, wvlngth_max = wvlngth_max,
                           atmosphere_file = atmosphere_file, source_solar = source_solar,
                           rte_solver = rte_solver, number_of_streams = number_of_streams,
                           sza = unique(sza), mol_modify_03 = unique(mol_modify_03),
                           mol_modify_H2O = unique(mol_modify_H2O), albedo = unique(albedo),
                           altitude = altitude, stringsAsFactors = FALSE)
todolisting <- data.frame(todolisting, stringsAsFactors = FALSE)
colanmes <- names(todolisting)
##### load saved results
saved_results <- readRDS(datafile)
saved_results <- data.frame(saved_results, stringsAsFactors = FALSE)
saved_results$hostname <- as.character(saved_results$hostname)

### Find jobs to do
temptodo = todolisting[, colanmes] ## will keep intact
tempsaved = saved_results[, colanmes] ## used to compare
## combine with index
temptodo$coder = "A"
tempsaved$coder = "B"
temp <- rbind(tempsaved, temptodo)
uniqu <- !duplicated(temp[, colanmes])
## keep unique
temp = temp[uniqu, ]
## dont include 'B' (saved results)
temp = temp[temp$coder == "A", ]

```

```

## drop coder column
todolist = subset(temp, select = -coder)
if (length(todolist[, 1]) < 1) stop("STOPED! No jobs to do")

## add job index to keep track
todolist$job_id = cksm(apply(todolist, 1, paste, collapse = ""))
## creat file with list of jobs to submit serialize
cat("", file = outfilelist)
for (ri in 1:nrow(todolist)) {
  OptVect = todolist[ri, ]
  cat(sprintf("", sprintf("%s ", OptVect$job_id), sprintf("atmosphere_file@@aattmmoo=%s.dat",
    OptVect$atmosphere_file), sprintf("source@@solar@@ssoolaa=%s.dat",
    OptVect$source_solar), sprintf("mol_modify@@03@@%s@@DU@",
    OptVect$mol_modify_03), sprintf("mol_modify@@H2O@@%s@@MM@",
    OptVect$mol_modify_H2O), sprintf("albedo@@%s@", OptVect$albedo),
    sprintf("sza@@%s@", OptVect$sza), sprintf("altitude@@%s@",
    OptVect$altitude), sprintf("rte_solver@@%s@", OptVect$rte_solver),
    sprintf("number_of_streams@@%s@", OptVect$number_of_streams),
    sprintf("wavelength@@%s@@%s@", OptVect$wvlngth_min, OptVect$wvlngth_max),
    sprintf("pseudospherical@"), sprintf("quiet@"), sprintf("\n"),
    sep = "", file = outfilelist, append = TRUE)
}
cat(paste("Jobs to do: ", nrow(todolist), "\n"))

```

4 Διαχείριση αποτελεσμάτων.

Η αποθήκευση και χρήση των αποτελεσμάτων μπορεί να γίνει αρκετά δύσκολη μετά από κάποιο όγκο προσομοιώσεων. Όπως για παράδειγμα, όταν έχουμε να κάνουμε με διερεύνηση ατμοσφαιρικών συνθηκών, οπού πολύ εύκολα μπορούμε να φτάσουμε τα εκατομμύρια εκτελέσεων. Αυτό μπορεί να θέσει όρια λόγω της μνήμης του υπολογιστή, της πολυπλοκότητας αποθήκευσης, του χρόνου αναζήτησης, του συστήματος αρχείων και άλλα. Για τον λόγο αυτό, προτείνεται η χρήση κάποιας βάσης δεδομένων του προτύπου SQL. Στην εφαρμογή μας χρησιμοποιήσαμε την SQLite.

Επιγραμματικά αναφέρουμε κάποια οφέλη:

- Πρακτικά, απεριόριστη δυνατότητα αποθήκευσης σε ενιαίο αρχείο/πίνακα.

- Ταχύτατη αναζήτηση και ανάσυρση δεδομένων.
- Δυνατότητα απόρριψης διπλότυπων κατά την εισαγωγή δεδομένων.
- Συμβατή με τις περισσότερες γλώσσες προγραμματισμού και υπολογιστικών συστημάτων.
- Δυνατότητα αποθήκευση περιληπτικών αποτελεσμάτων και αντιστοίχηση με τα φασματικά.

5 Οδηγός Εγκατάστασης της LibRadtran σε περιβάλλον Windows.

Αρχικός οδηγός από τον Κώστα Φράγκο, προσαρμογή από Θανάση Νάτση.

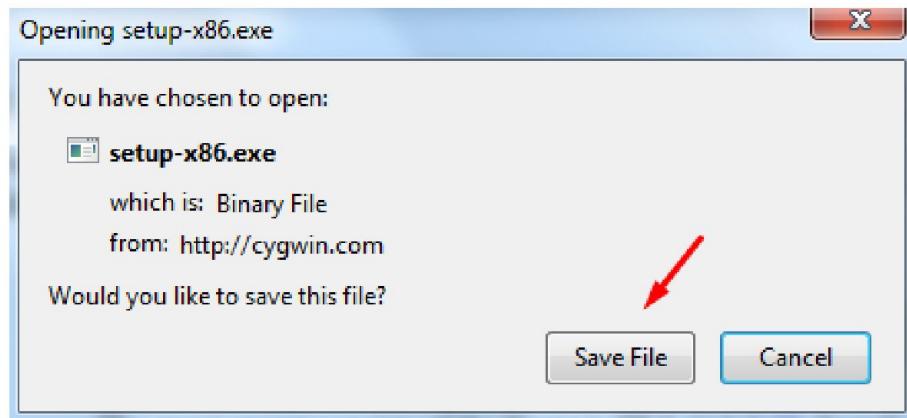
Η LibRadtran είναι ένα πακέτο επίλυσης της εξίσωσης της διάδοσης της ακτινοβολίας μιας διάστασης (1D). Αποτελείται από ξεχωριστούς κώδικες γραμμένους σε γλώσσα ‘C’ ή/και ‘Fortran’. Πριν να είναι κάποιος σε θέση να χρησιμοποιήσει τη LibRadtran θα πρέπει να “μεταγλωτίσει” και να “συνδέσει” (εγκατάσταση) τις ξεχωριστές ρουτίνες μεταξύ τους. Επειδή, το πακέτο είναι κατασκευασμένο να δουλεύει σε περιβάλλον Linux, για να το εγκαταστήσουμε σε περιβάλλον Windows απαιτούνται κάποιες επιπρόσθετες διαδικασίες, οι οποίες περιγράφονται βήμα-βήμα σε αυτόν τον οδηγό και περιγράφονται πιο συνοπτικά στις οδηγίες εγκατάστασης σε περιβάλλον Windows στη σελίδα της [LibRadtran](#).

5.1 Εγκατάσταση του λειτουργικού περιβάλλοντος ‘Cygwin’.

Αρχικά εγκαθιστούμε τη σουίτα ‘Cygwin’, που μας επιτρέπει να εκτελούμε εντολές Linux. Κατά τη διάρκεια της εγκατάστασης του ‘Cygwin’ πρέπει να σιγουρευτούμε ότι έχουμε περιλάβει όλες τις απαραίτητες βιβλιοθήκες που απαιτούνται για την ομαλή εγκατάσταση της LibRadtran. Αυτές είναι οι ακόλουθες:

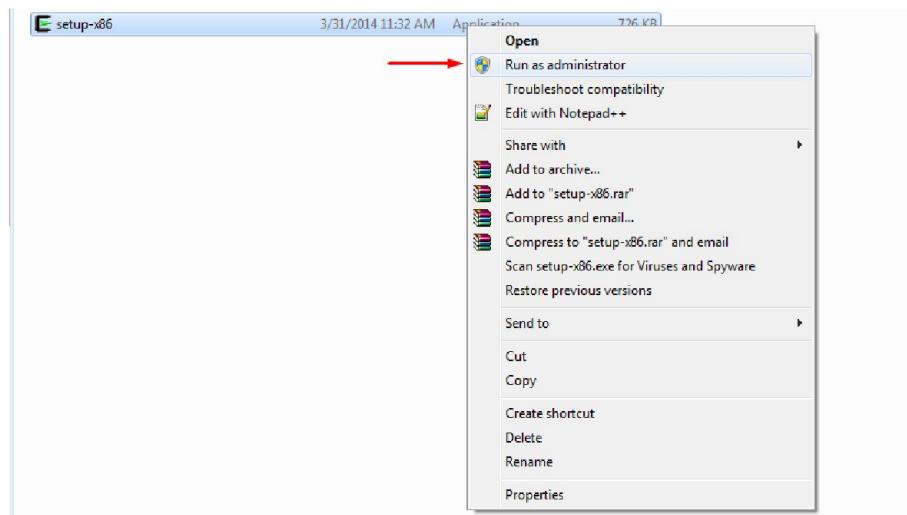
```
bash binutils cygwin flex gawk  
gcc gzip make tar
```

Από την ιστοσελίδα του [Cygwin](#) κατεβάζουμε το πρόγραμμα εγκατάστασης του Cygwin. **Σημείωση ακόμα και αν έχουμε 64bit έκδοση λειτουργικού, καλό θα ήταν να εγκαταστήσουμε την 32bit έκδοση του ‘Cygwin’, για να αποφύγουμε στη συνέχεια κάποιες επιπλέον ρυθμίσεις.**



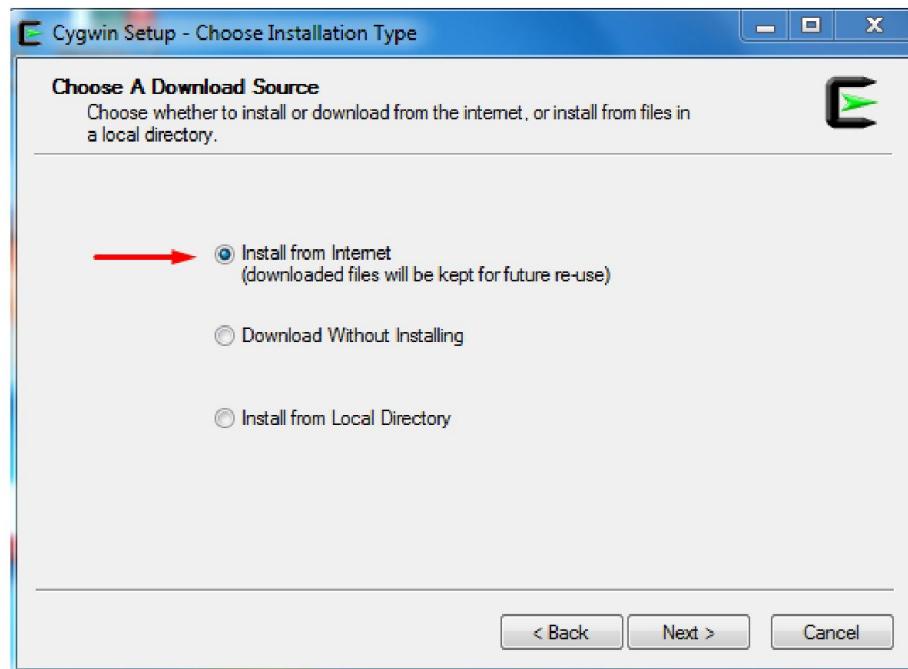
Εικόνα 5.1: Κατέβασμα και αποθήκευση του εκτελέσιμου προγράμματος εγκατάστασης του 'Cygwin'.

Αφού κατεβάσουμε το εκτελέσιμο (Εικόνα 5.1), πατάμε δεξί κλικ και επιλέγουμε 'Run as Administrator' (Εκτέλεση ως διαχειριστής) (Εικόνα 5.2).



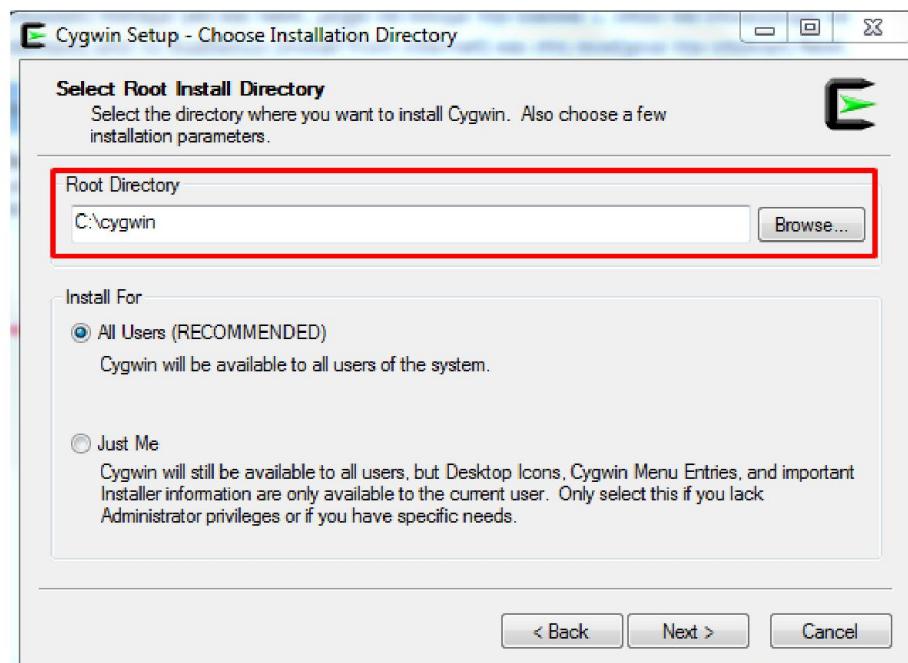
Εικόνα 5.2: Εκτέλεση του προγράμματος εγκατάστασης του 'Cygwin'.

Στις επόμενες επιλογές πατάμε 'yes' και 'Next', μέχρι να δούμε την Εικόνα 5.3, όπου θα επιλέξουμε να κάνουμε εγκατάσταση από το διαδίκτυο (install from internet) και στη συνέχεια την επιλογή 'Next'.



Εικόνα 5.3: Εγκατάσταση του Cygwin χρησμοποιώντας το Internet.

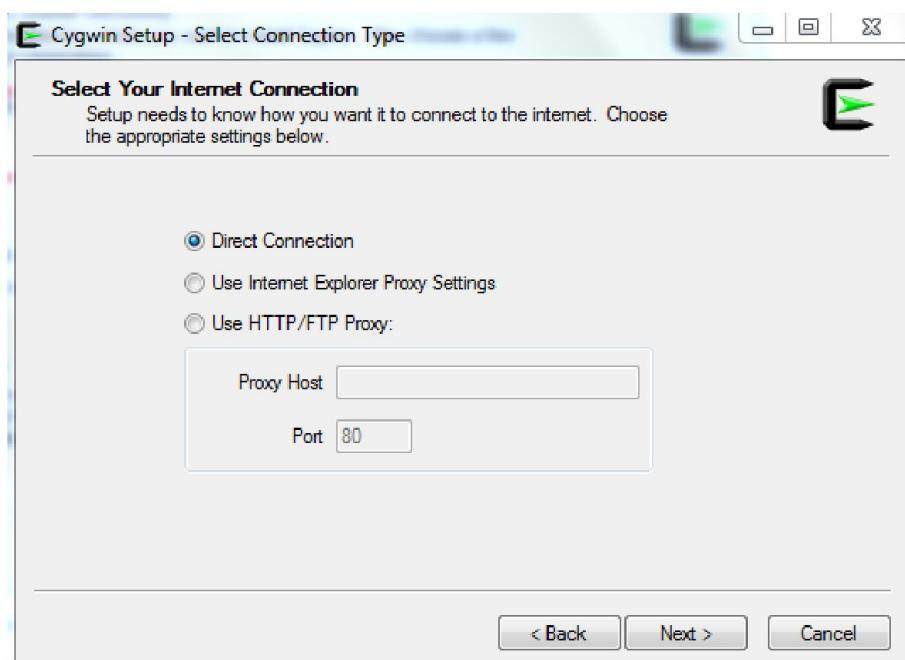
Επιλέγουμε να αφήσουμε το default directory εγκατάστασης (5.4) και συνεχίζουμε πατώντας ‘Next’. Είναι καλό ο φάκελος εγκατάστασης που θα επιλέξουμε να μην έχει κενά, να αποτελείται μόνο από λατινικούς χαρακτήρες και είναι δυνατόν να βρίσκεται στο c:\ (root των Windows).



Εικόνα 5.4: Επιλογή εγκατάστασης του Cygwin στο default directory.

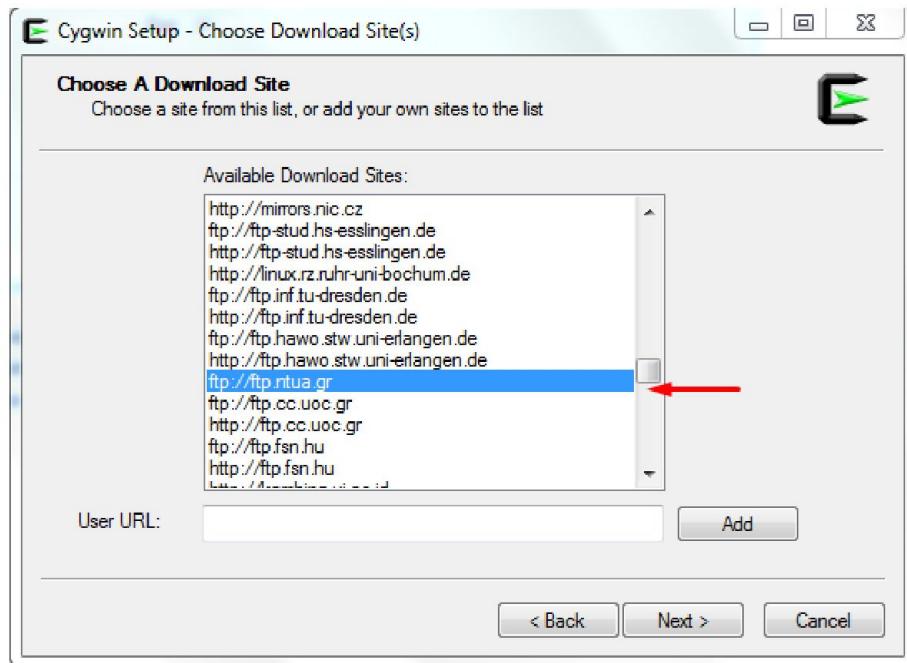
Προσπερνάμε την επόμενη σελίδα που θα μας βγάλει, πατώντας ‘Next’ και στη συνέχεια

επιλέγουμε τον τύπο σύνδεσής μας στο Internet (συνήθως είναι direct connection) και επιλέγουμε το ‘Next’ (Εικόνα 5.5).



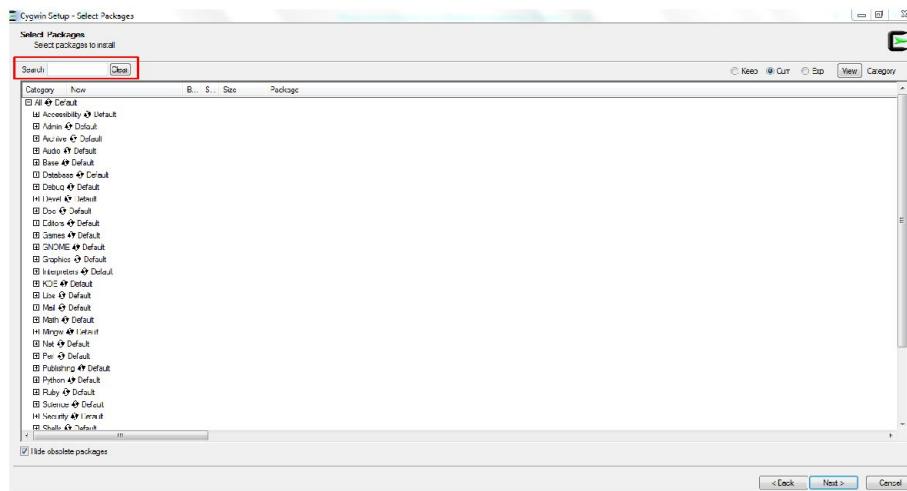
Εικόνα 5.5: Επιλογή σύνδεσης στο Internet.

Στην επόμενη καρτέλα διαλέγουμε από που θέλουμε να κατεβάσουμε τα πακέτα που είναι απαραίτητα για την εγκατάσταση του ‘Cygwin’. Κάνοντας scroll down με το ποντίκι μπορούμε να βρούμε κάποια τοποθεσία στην Ελλάδα (π.χ. στην Εικόνα 5.6 έχει γίνει επιλογή του ftp του Μετσόβιου Πολυτεχνείου. Κάποιος μπορεί να επιλέξει οποιαδήποτε τοποθεσία θέλει, επιλέγοντας κάποια τοποθεσία στην Ελλάδα ελαχιστοποιεί το χρόνο που χρειάζεται για να κατέβουν τα πακέτα εγκατάστασης).



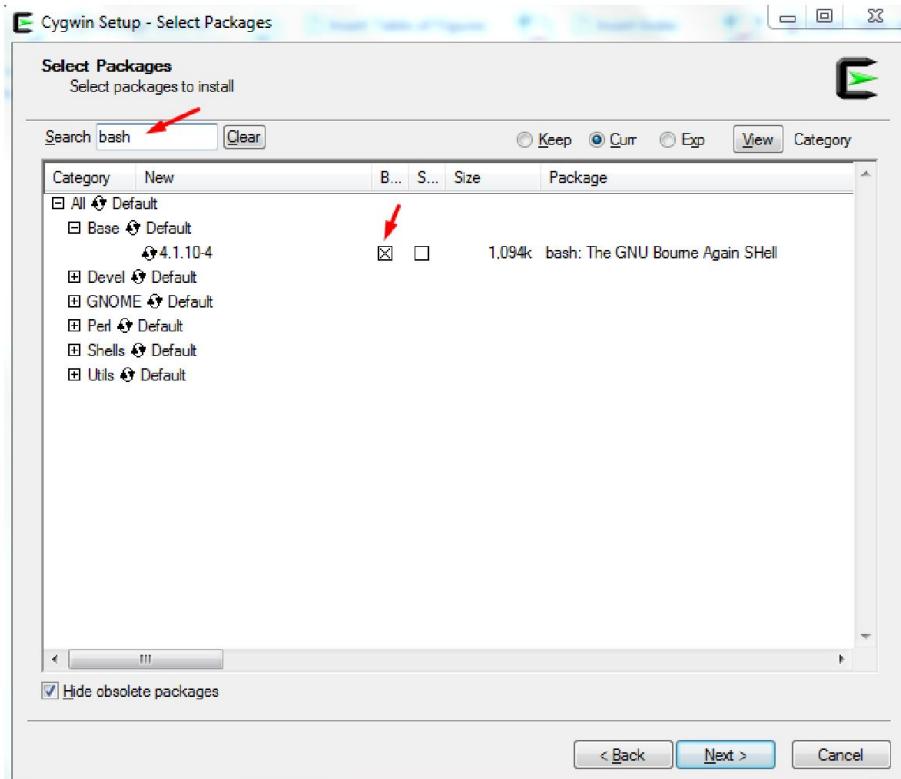
Εικόνα 5.6: Επιλογή τοποθεσίας για το κατέβασμα του 'Cygwin'.

Στην επόμενη σελίδα θα διαλέξουμε τα πακέτα που χρειάζονται για την εγκατάσταση της LibRadtran τα οποια όπως είδαμε πιο πάνω είναι: bash, binutils, cygwin, flex, gawk, gcc, gzip, make, tar. Στο πλαίσιο που έχει για αναζήτηση (Εικόνα 5.7) τοποθετούμε καθένα πακέτο ξεχωριστά και κάνουμε αναζήτηση για να δούμε αν είναι προεγκατεστημένο ή χρειάζεται να το επιλέξουμε εμείς. Συνήθως, τα πακέτα βρίσκονται στις κατηγορίες 'Base', 'Devel' και 'Utils'.



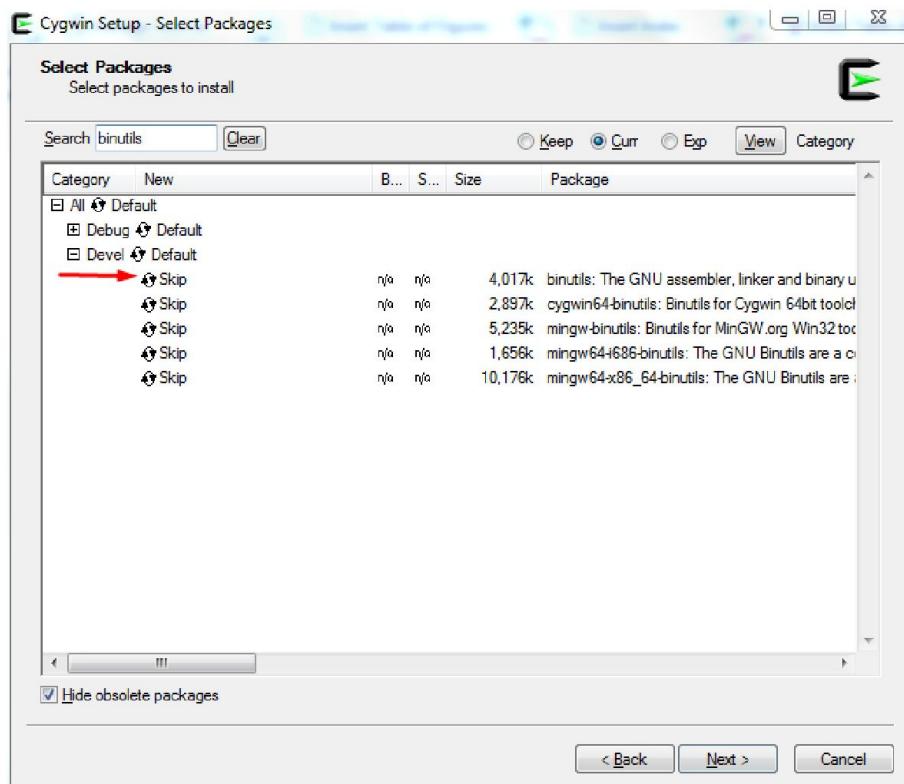
Εικόνα 5.7: Αναζήτηση των πακέτων που είναι απαραίτητα να εγκατασταθούν.

Ξεκινούμε αναζητώντας το πακέτο bash (Εικόνα 5.8) όπου παρατηρούμε ότι είναι ήδη στα πακέτα που γίνονται εγκατάσταση.

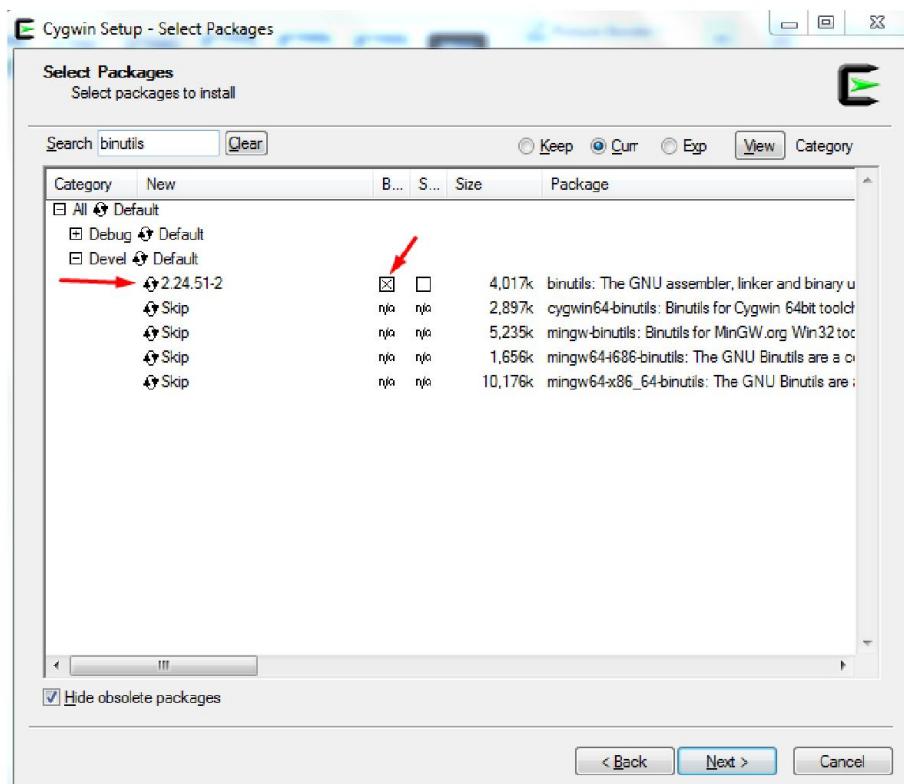


Εικόνα 5.8: Αναζήτηση του πακέτου 'bash', παρατηρούμε ότι είναι προεγκατεστημένα πακέτα του Cygwin.

Στη συνέχεια αναζητούμε το binutils, το οποίο είναι στο 'Devel' (Εικόνα 5.9) παρατηρούμε ότι έχει την επιλογή Skip, πατώντας μια φορά με το ποντίκι πάνω στο 'Skip', αλλάζουμε την κατάσταση ώστε να εγκατασταθεί, όπως φαίνεται στην Εικόνα 5.10. Την ίδια διαδικασία ακολουθούμε και για όλα τα υπόλοιπα πακέτα που παρατηρούμε ότι δεν περιέχονται από default (όπως το bash) στην εγκατάσταση του 'Cygwin'.



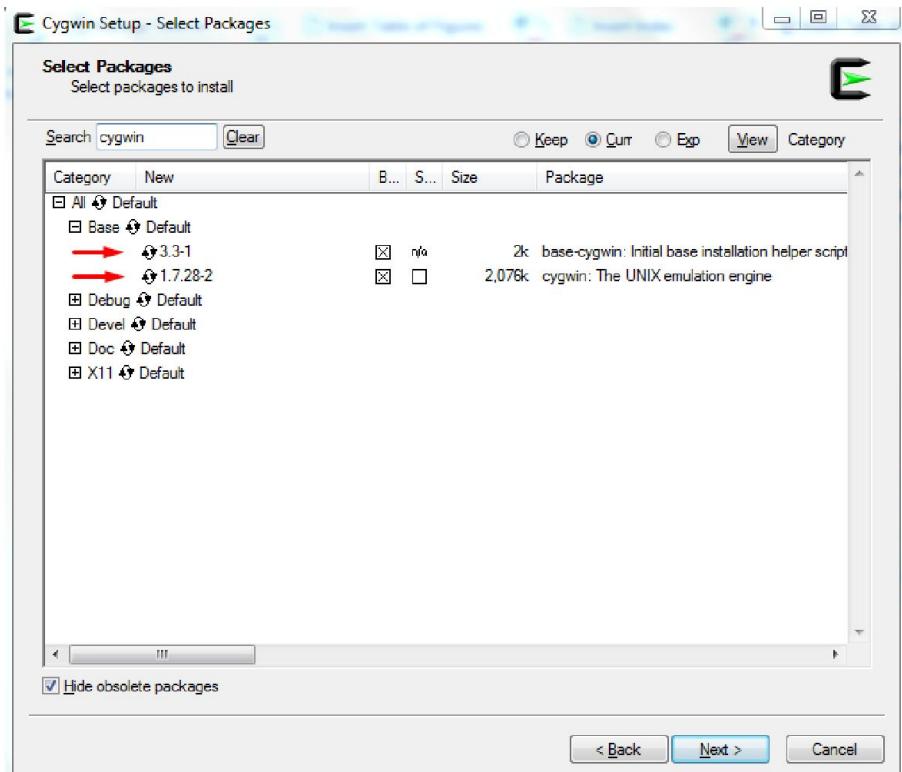
Εικόνα 5.9: Αναζήτηση της επιλογής 'binutils'.



Εικόνα 5.10: Επιλογή εγκατάστασης του 'binutils'.

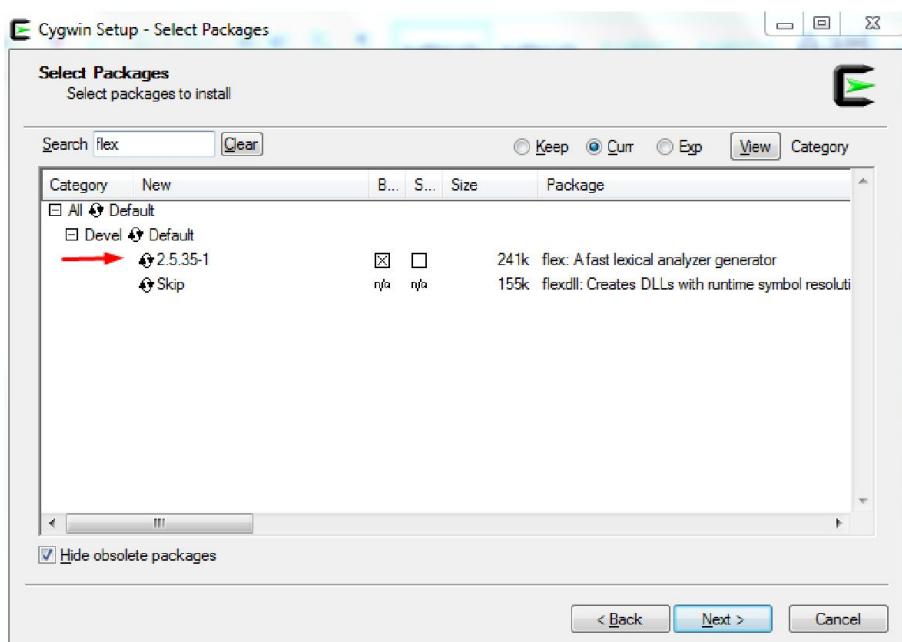
Εν συνεχεία αναζητούμε το 'Cygwin', το οποίο όπως παρατηρούμε στην Εικόνα 5.11, εγκα-

θίσταται ως default επιλογή.



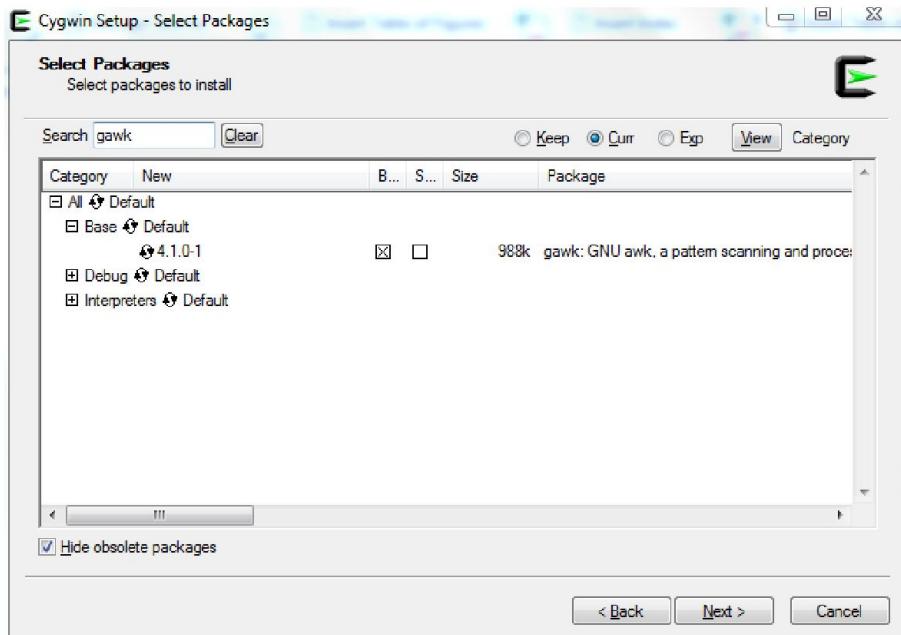
Εικόνα 5.11: Αναζήτηση του 'Cygwin', δεν χρειάζεται κάποια μεταβολή.

Το πακέτο flex δεν περιέχεται στα default πακέτα εγκατάστασης, οπότε κάνουμε ότι για το πακέτο binutils, έτσι ώστε να δούμε την Εικόνα 5.12.



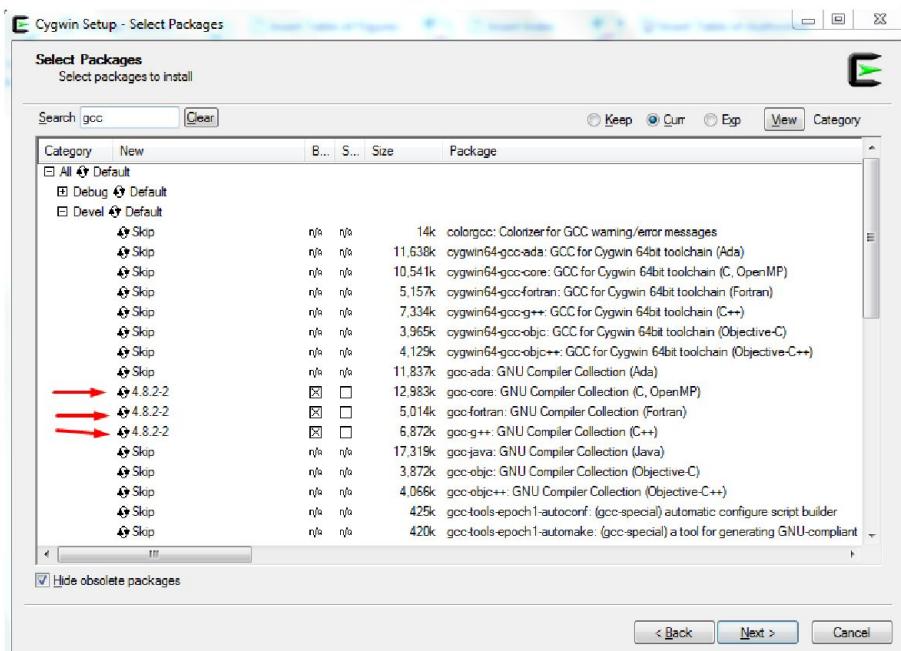
Εικόνα 5.12: Επιλογή εγκατάστασης του πακέτου 'flex'.

To πακέτο gawk, βρίσκεται ανάμεσα σε αυτά που εγκαθίστανται από default (Εικόνα 5.13).



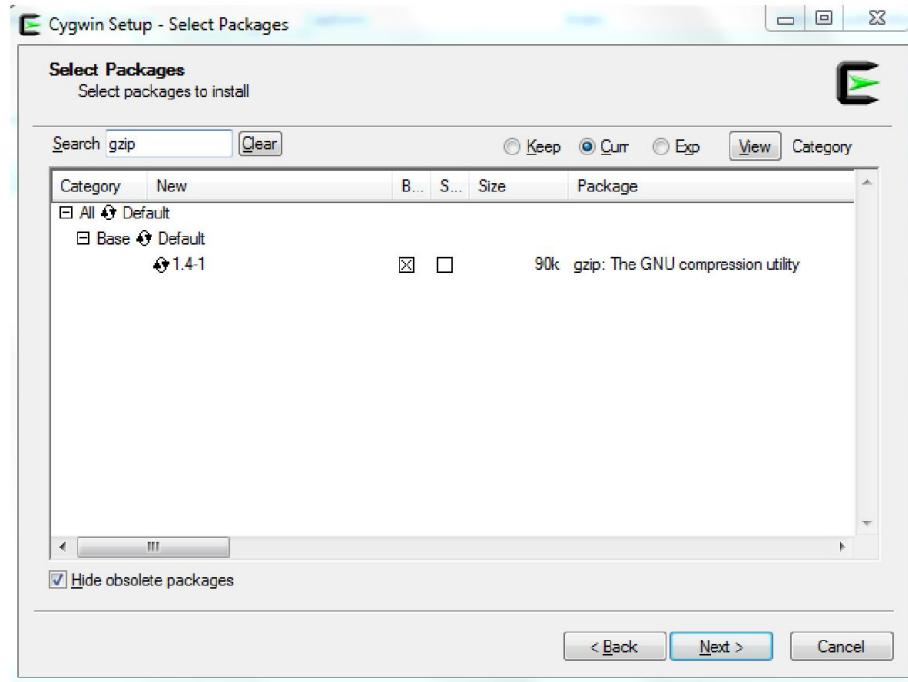
Εικόνα 5.13: Αναζήτηση του πακέτου 'gawk'.

Ο gcc είναι ο compiler που μεταγλωττίζει κώδικα γλώσσας 'C' και 'Fortran'. Επιλέγετε για εγκατάσταση αυτούς που φαίνονται στην Εικόνα 5.14.



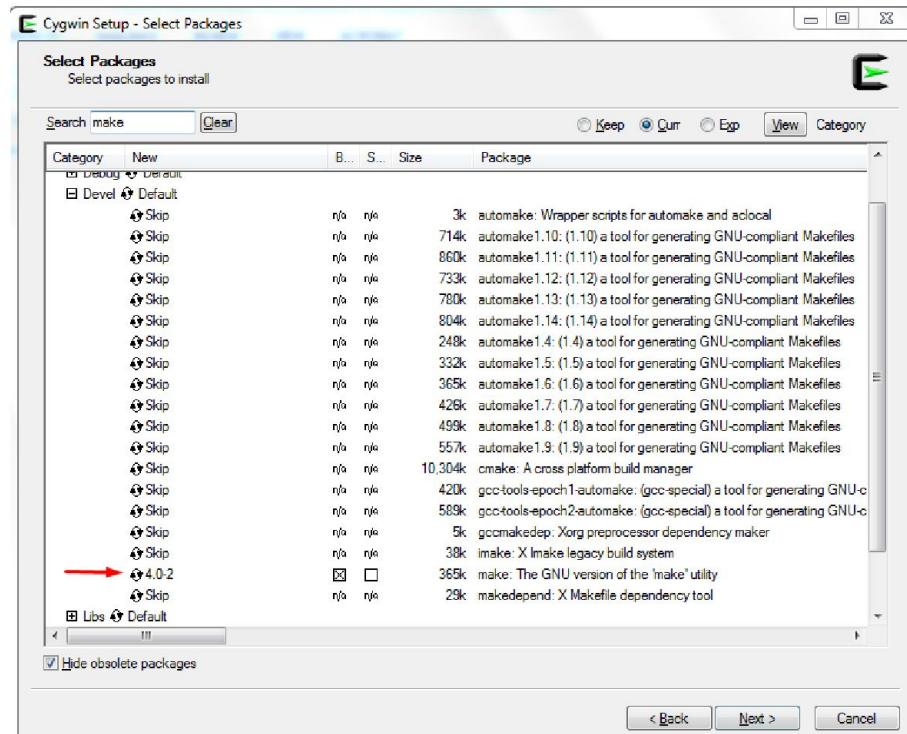
Εικόνα 5.14: Εγκατάσταση του compilers 'gcc' για μεταγλώττιση του κώδικα C και Fortran.

To gzip είναι από τα default πακέτα του 'Cygwin' (Εικόνα 5.15).



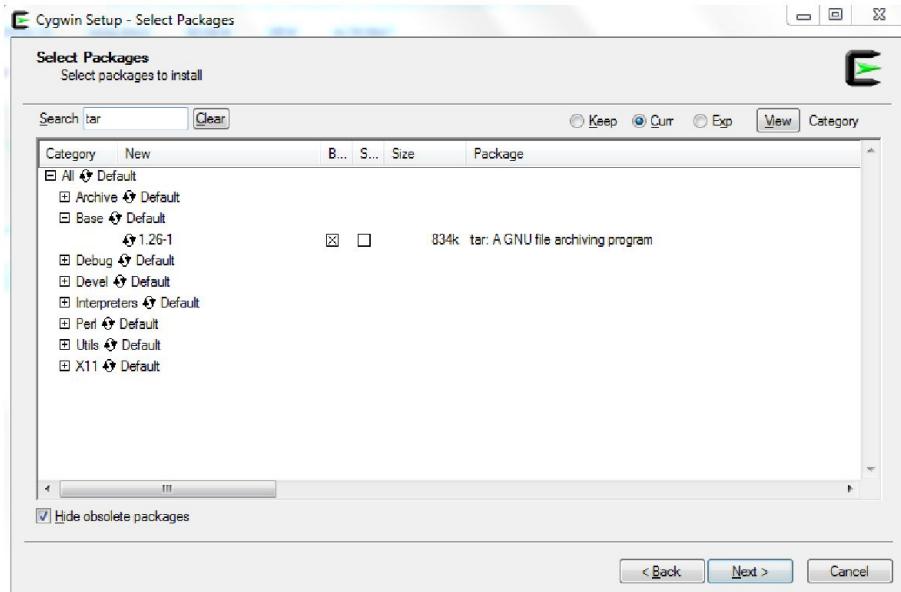
Εικόνα 5.15: Αναζήτηση του πακέτου 'gzip'.

To `make` δεν είναι στα default οπότε χρειάζεται να αλλάξουμε την κατάστασή του για να εγκατασταθεί (Εικόνα 5.16).



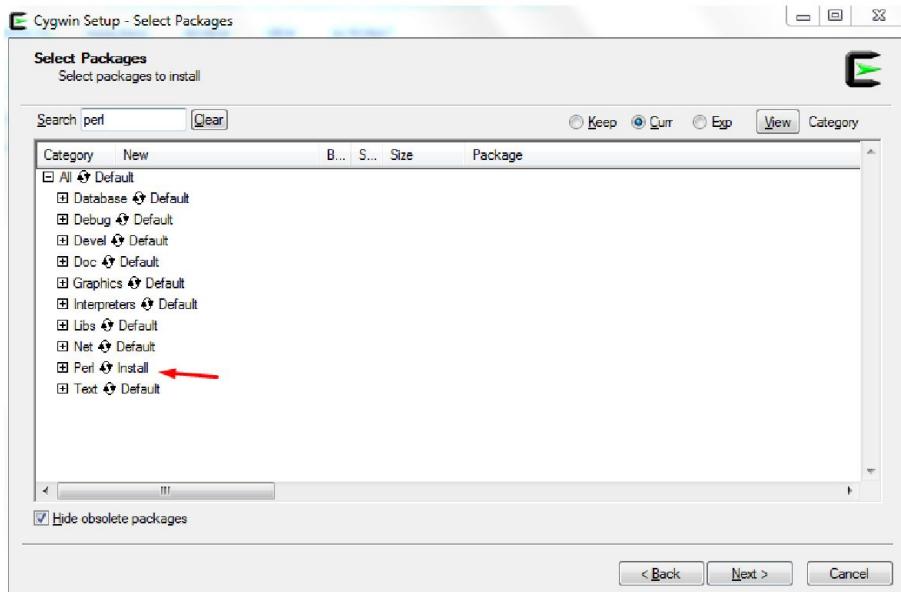
Εικόνα 5.16: Εγκατάσταση του πακέτου 'make'.

To `tar` εγκαθίσταται από default (Εικόνα 5.17).



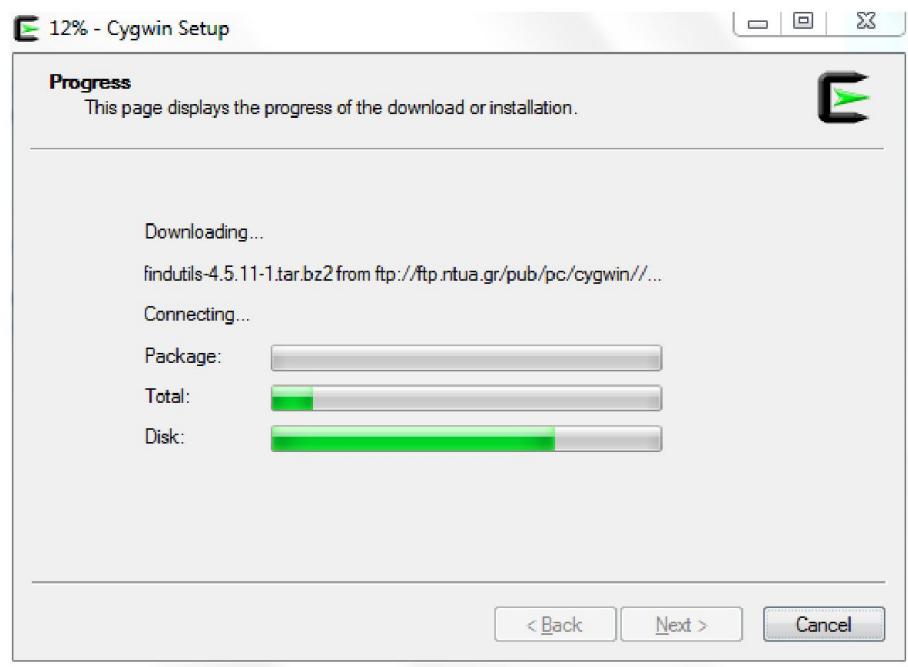
Εικόνα 5.17: Αναζήτηση του πακέτου 'tar'.

Τέλος εκτός από τα παραπάνω που είναι απαραίτητα για την εγκατάσταση της LibRadtran, καλό θα ήταν να εγκαταστήσουμε μαζί με το 'Cygwin' και τη γλώσσα 'Perl', αλλάζοντας το status από default σε install (Εικόνα 5.18).

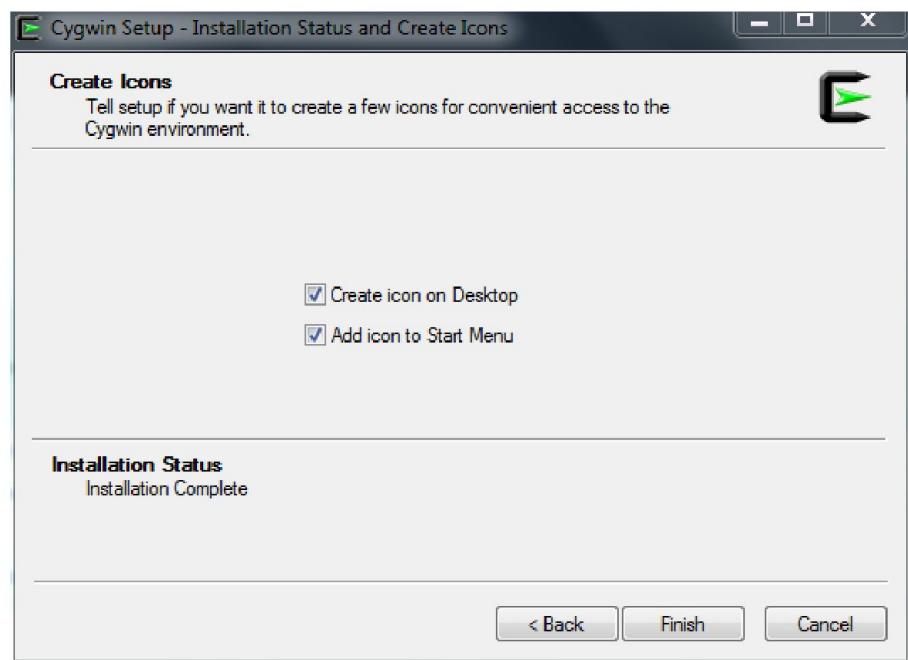


Εικόνα 5.18: Εγκατάσταση της γλώσσας 'Perl' σε περιβάλλον 'Cygwin'.

Αφού ολοκληρώσουμε τον έλεγχο για τα πακέτα που θέλουμε να εγκαταστήσουμε πατάμε 'Next', όπου μετά θα δούμε όλα τα πακέτα που εγκαθίστανται και πατάμε 'Next', οπότε ξεκινάει το κατέβασμα και η εγκατάσταση του 'Cygwin' και των πακέτων που επιλέξαμε (Εικόνα 5.19).



Εικόνα 5.19: Εγκατάσταση του 'Cygwin'.

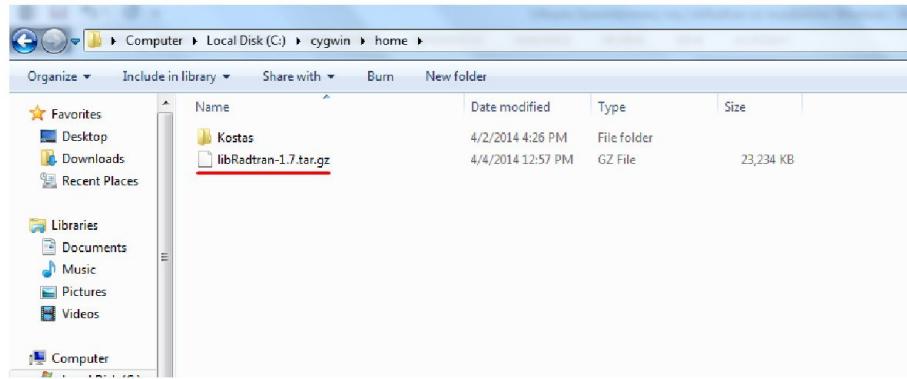


Εικόνα 5.20: Ολοκλήρωση εγκατάστασης του 'Cygwin'.

5.2 Εγκατάσταση της LibRadtran.

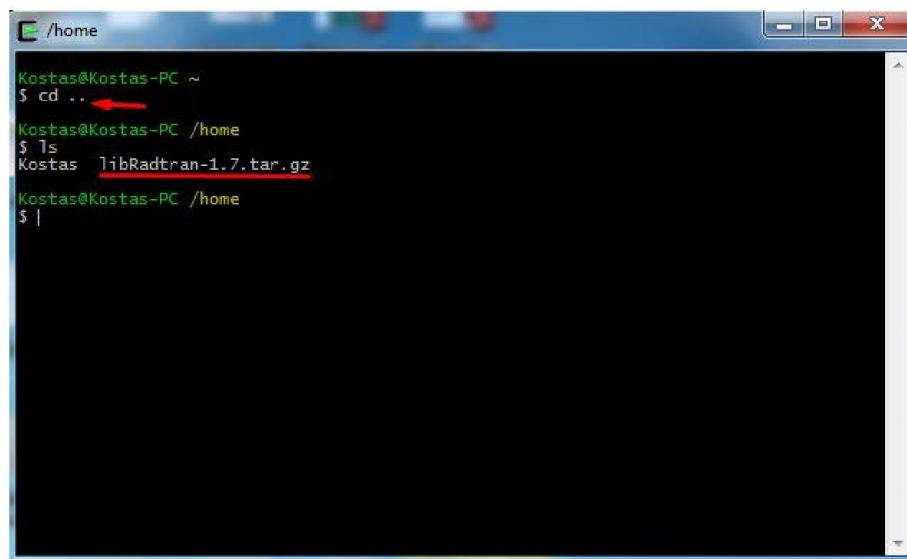
Κατεβάζουμε τη LibRadtran από το [site](#). Παλιότερες εκδόσεις της LibRadtran μπορούν να βρεθούν [εδώ](#).

Αφού ολοκληρωθεί η λήψη του αρχείου, το τοποθετούμε στο φάκελο c:\cygwin\home (Εικόνα 5.21).



Εικόνα 5.21: Κατάλογος στον οποίο θα γίνει η εγκατάσταση της LibRadtran.

Ανοίγουμε το τερματικό του 'Cygwin' από την επιφάνεια εργασίας, εκτελούμε την εντολή cd .. για να βρεθούμε στο 'home' και στη συνέχεια την εντολή ls για να σιγουρευτούμε ότι έχουμε το πακέτο της libRadtran στον κατάλογο που βρισκόμαστε (Εικόνα 5.22).



Εικόνα 5.22: Κατάλογος εργασίας στο περιβάλλον 'Cygwin'.

Αφού σιγουρευτούμε ότι έχουμε το πακέτο της LibRadtran στον κατάλογο που βρισκόμαστε εκτελούμε τις ακόλουθες εντολές, για την αποσυμπίεση των αρχείων.

```
gzip -d libRadtran-1.7.tar.gz  
tar -xvf libRadtran-1.7.tar
```

στη συνέχεια με την εντολή ls επαληθεύουμε ότι έχει δημιουργηθεί ένας νέος φάκελος (Εικόνα 5.23).

```

/home
libRadtran-1.7/libsrc_f/avhrr22.f
libRadtran-1.7/libsrc_f/avhrr23.f
libRadtran-1.7/libsrc_f/setout.f
libRadtran-1.7/libsrc_f/de1.f
libRadtran-1.7/libsrc_f/radscat3.f
libRadtran-1.7/libsrc_f/DISORT.doc
libRadtran-1.7/libsrc_f/initds.f
libRadtran-1.7/libsrc_f/xerhlt.f
libRadtran-1.7/libsrc_f/dcsevl.f
libRadtran-1.7/libsrc_F/MIEVO.f
libRadtran-1.7/libsrc_f/avhrr12.f
libRadtran-1.7/libsrc_f/xgetua.f
libRadtran-1.7/libsrc_f/avhrr33.f
libRadtran-1.7/libsrc_F/ErrPack.f
libRadtran-1.7/libsrc_f/spsmisc.f
libRadtran-1.7/flexstor/
libRadtran-1.7/flexstor/TABLE.pm
libRadtran-1.7/flexstor/Makefile.in
libRadtran-1.7/flexstor/SGL.pm
libRadtran-1.7/flexstor/Spectrum.pm
libRadtran-1.7/flexstor/Column.pm
libRadtran-1.7/flexstor/Flexstor.pm

Kostas@Kostas-PC /home
$ ls
Kostas libRadtran-1.7 libRadtran-1.7.tar
Kostas@Kostas-PC /home
$
```

Εικόνα 5.23: Αποσυμπίεση του αρχείου Libradtran.tar.

Γράφουμε `cd libRadtran-1.7`, για να μεταφερθούμε στον φάκελο της Libradtran, με `ls` μπορούμε να δούμε τα περιεχόμενα του φακέλου και στη συνέχεια γράφουμε την ακόλουθη εντολή, για να επιλέξουμε τον compiler για την μεταγλώττιση του κώδικα ‘Fortran’ (Εικόνα 5.24).

```
export F77=gfortran
```

```

/home/libRadtran-1.7
Kostas@Kostas-PC /home
$ cd libRadtran-1.7
Kostas@Kostas-PC /home/libRadtran-1.7
$ ls
bin      configure   doc      INSTALL    libsrc_f    README      sdoc
ChangeLog  configure.in examples  install-sh Makeconf.in README.macosx  src
config.guess  COPYING    flexstor  lib       Makefile.in README.unpacking test
config.sub   data      GUI      libsrc_c  python    README.windows TODO
Kostas@Kostas-PC /home/libRadtran-1.7
$ export F77=gfortran
```

Εικόνα 5.24: Επιλογή του compiler για την αποσφαλμάτωση του κώδικα Fortran.

Τώρα εκτελούμε την εντολή `./configure`. Στο τέλος, θα πρέπει να δούμε στην οθόνη του τερματικού τα ακόλουθα όπως φαίνεται στην Εικόνα 5.25.

```
home@kostas-OptiPlex-5090:~
```

```
config-status: creating data/iccpe/Makefile
config-status: creating data/rhume_hufft/Makefile
config-status: creating data/fft/Makefile
config-status: creating data/correlated/kabs/Makefile
config-status: creating data/correlated/kabs2/Makefile
config-status: creating data/correlated/k/fu/Makefile
config-status: creating data/correlated/k/avx/Makefile
config-status: creating test/test_p
config-status: creating lib/resource/Makefile
config-status: creating python/Makefile

libKastran is now configured for i686-pc-cygwin

Source directory:
 �abelization prefix: /usr/local
  C compiler:          gcc -std=gnu99 -O2 -Wall
  C++ compiler:        g++ -std=c++0x
  Fortran compiler:
  Fortran linker:
  Fortran libraries:
  On the fly linking:  /usr/lib/i686-pc-cygwin/4.2.2/lib/cygwin-i686-pc-cy
  On the fly libraries: -lgcc -lgcc_s -lstdc++ -lstdc++fs -lncurses -ladvapi32 -lshlwapi -luser32 -lkernel32 -luser32 -lshell32
  ARIE:                gade
  VIM:                 /usr/bin/vim
  PYTHON:              /usr/bin/python
  MPIFFP: library:
    GSI:                gsi
    QOI:                qoi

  PolRadtrans:         yes
  LAMMPS:              no
  Fu and Lin:          yes
  Kratz (AVX):        yes
  SIS:                 yes
  O2AC:                no
  Kostas et al. (2002): yes
  Yang/Key/Mayer:      yes
  (Yang/Key/Mayer):    no
  Bona et al. (2005):  no
  HBY (Ice shant):     no
  SOURCE CONTRAINT:    yes
  Full T-matrix:       no
  BDFP:                yes
  NVSTFC:              yes

  VROOM:               yes
```

Εικόνα 5.25: Ολοκλήρωση της διαδικασίας *configure*.

Η εγκατάσταση (compile) τώρα μπορεί να ολοκληρωθεί εκτελώντας την εντολή `make` και για σιγουρευτούμε ότι όλα είναι εντάξει κατά τη διάρκεια της εγκατάστασης, αφού τελειώσει η εντολή `make`, εκτελούμε την εντολή `check`, όπου θα δούμε στην οθόνη του τερματικού μας μια σειρά από `test`, όπως στην Εικόνα 5.26.

```
make[1]: Leaving directory '/home/libRadtran-1.7/libsr_f'
make[1]: Entering directory '/home/libRadtran-1.7/src'
make[1]: Leaving directory '/home/libRadtran-1.7/src'
make[1]: Entering directory '/home/libRadtran-1.7/GUI'
for dir in resources; do make -C $dir all; done
make[2]: Entering directory '/home/libRadtran-1.7/GUI/resources'
echo tar -xzf html_doc.tar.gz
tar -xzf html_doc.tar.gz
make[2]: Leaving directory '/home/libRadtran-1.7/GUI/resources'
make[1]: Leaving directory '/home/libRadtran-1.7/GUI'
make -e -C test
make[1]: Entering directory '/home/libRadtran-1.7/test'
/usr/bin/perl test.pl
Running various libRadtran tests. This may take some time....
```

The numbers in the parenthesis behind the name of the tests are:
1st number: The lower absolute limit of values included in the test.
Values in the output less than limit are ignored.
2nd number: The maximum difference allowed between local test
results and the standard results (in percentage).

If this is still unclear, check the source in test/test.pl.in.

```
make_slitfunction test
make_slitfunction (0.00001, 0.1)..... ok.
All make_slitfunction tests succeeded.
Some uvspec tests
uvspec simple (0.00001, 0.1)..... ok.
disort clear sky (0.00001, 0.1)..... |
```

Εικόνα 5.26: Έλεγχος της ορθής εγκατάστασης της Libradtran.

Εάν όλα έχουν ολοκληρωθεί επιτυχώς, στον φάκελο libRadtran-1.7/bin υπάρχουν μια σειρά από εκτελέσιμα αρχεία. Για την χρήση τους μπορείτε να ανατρέξετε στο manual της libRadtran ή στα αντίστοιχα αρχεία του πηγαίου κώδικα (source code) που πολλές φορές, μπορεί να είναι πολύ πιο κατατοπιστικά για τη λειτουργία που εκτελούν. Το κύριο εκτελέσιμο αρχείο του μοντέλου είναι το uvspec.

6 Χρήσιμες πληροφορίες για τη χρήση της libRadtran.

6.1 Μονάδες μέτρησης ηλιακού φάσματος.

Οι μονάδες μέτρησης των αποτελεσμάτων είναι ίδιες με του παραμετρικού αρχείου εισόδου του ηλιακού φάσματος. Για παράδειγμα για το αρχείο ‘kurudz_0.1nm.dat’ αναφέρεται ότι:

The original Kurudz [1992] data were converted to $mW/(m^2 nm)$ and averaged over $0.1nm$ intervals centered around the given wavelength.

6.2 Έξοδος του uvspec.

Η τυπική διαμόρφωση (default output format) της εξόδου του uvspec είναι:

```
lambda edir edn eup uavgdir uavgdn uavgup
```

Οι μεταβλητές εξόδου που μπορούν να δοθούν από το uvspec, ανάλογα με τα στοιχεία εισόδου, περιγράφονται στον Πίνακα 6.1.

Πίνακας 6.1: Μεταβλητές εξόδου της εντολής uvspec.

Symbol	Description
cmu	Computational polar angles from polradtran.
down_flux, up_flux	The total (direct+diffuse) downward (down_flux) and up-ward (up_flux) irradiances. Same units as extraterrestrial irradiance (e.g $mW/(m^2 nm)$ if using the atlas3 spectrum in the data/solar_flux directory.)
edir	Direct beam irradiance w.r.t. horizontal plane (same unit as extraterrestrial irradiance).
edn	Diffuse down irradiance, i.e. total minus direct beam (same unit as edir).
eup	Diffuse up irradiance (same unit as edir).
lambda	Wavelength (nm)
u0u	The azimuthally averaged intensity at numu user specified angles umu (units of e.g. $mW/(m^2 nmsr)$ if using the atlas3 spectrum in the data/solar_flux directory.) Note that the intensity correction included in the disort solver is not applied to u0u, thus u0u can deviate from the azimuthally-averaged intensity-corrected uu.
uavg	The mean intensity. Proportional to the actinic flux: To obtain the actinic flux, multiply the mean intensity by 4π (same unit as edir).
uavgdir	Direct beam contribution to the mean intensity (same unit as edir).
uavgdn	Diffuse downward radiation contribution to the mean intensity (same unit as edir).

Symbol	Description
uavgup	Diffuse upward radiation contribution to the mean intensity (same unit as edir).
uu	The radiance (intensity) at umu and phi user specified angles (unit e.g. $mW/(m^2 nmsr)$ if using the atlas3 spectrum in the data/solar_flux directory.)
uu_down, uu_up	The downwelling and upwelling radiances (intensity) at cmu and phi angles (unit e.g. $mW/(m^2 nmsr)$ if using the atlas3 spectrum in the data/solar_flux directory.)

6.3 Τυπικά ατμοσφαιρικά προφίλ.

Στον Πίνακα 6.2 παραθέτουμε στοιχεία από το ‘AFGL Atmospheric Constituent Profiles’ (Anderson et al., 1986).

Πίνακας 6.2: Παράμετροι πρώτυπων ατμοσφαιρικών προφίλ.

Model	Name	Lat	Time
1	Tropic	15N	Annual Average
2	Mid-Latitude Summer	45N	July
3	Mid-Latitude Winter	45N	January
4	Sub Arctic Summer	60N	July
5	Sub Arctic Winter	60N	January
6	U.S. Standard		1976

6.4 Προειδοποίηση για τη γωνία $SZA = 43.2^\circ$.

Η ζενίθια γωνία των 43.2° προκαλεί μία προειδοποίηση (warning). Το οποίο σχετίζεται με την δυνατότητα υπολογισμού τριγωνομετρικών συναρτήσεων μικρών γωνιών.

```
***** WARNING >>>>
SETDIS--beam angle=computational angle;
***** changing cosine of solar zenith angle, umu0, from 0.728969 to 0.728928
```

A' Παράρτημα

```
$ zcat /usr/share/man/man8/pbsnodes.8.gz | groff -mandoc -T html > man_qsub.html  
$ zcat /usr/share/man/man1/qsub.1.gz | groff -mandoc -T html > man_qsub.html  
$ zcat /usr/share/man/man8/pbsnodes.8.gz | groff -mandoc -T html > man_pbsnodes.html
```

A'.1 man pbsnodes

NAME

pbsnodes – pbs node manipulation

SYNOPSIS

```
pbsnodes [-{a|x}] [-q] [-s server] [node[:property]]  
pbsnodes -l [-q] [-s server] [state] [nodename[:property] ...]  
pbsnodes [-{c|d|o|r}] [-q] [-s server] [-n] [-N "note"] [node[:property]]
```

DESCRIPTION

The **pbsnodes** command is used to mark nodes down, free or offline. It can also be used to list nodes and their state. Node information is obtained by sending a request to the PBS job server. Sets of nodes can be operated on at once by specifying a node property prefixed by a colon. Nodes do not exist in a single state, but actually have a set of states. For example, a node can be simultaneously “busy” and “offline”. The “free” state is the absence of all other states and so is never combined with other states.

In order to execute **pbsnodes** with other than the **-a** or **-l** options, the user must have PBS Manager or Operator privilege.

OPTIONS

-a

All attributes of a node or all nodes are listed. This is the default if no flag is given.

-x

Same as -a, but the output has an XML-like format.

-c

Clear OFFLINE from listed nodes.

-d

Print MOM diagnosis on the listed nodes. Not yet implemented. Use momctl instead.

-o

Add the OFFLINE state. This is different from being marked DOWN. OFFLINE prevents new jobs from running on the specified nodes. This gives the administrator a tool to hold a node out of service without changing anything else. The OFFLINE state will never be set or cleared automatically by pbs_server; it is purely for the manager or operator.

-p

Purge the node record from pbs_server. Not yet implemented.

-r

Reset the listed nodes by clearing OFFLINE and adding DOWN state. pbs_server will ping the node and, if they communicate correctly, free the node.

-l

List node names and their state. If no state is specified, only nodes in the DOWN, OFFLINE, or UNKNOWN states are listed. Specifying a state string acts as an output filter. Valid state strings are “free”, “offline”, “down”, “reserve”, “job-exclusive”, “job-sharing”, “busy”, “time-shared”, or “state-unknown”.

-N

Specify a “note” attribute. This allows an administrator to add an arbitrary annotation to the listed nodes. To clear a note, use -N ”” or -N n.

-n

Show the “note” attribute for nodes that are DOWN, OFFLINE, or UNKNOWN. This option requires -l.

-q

Supress all error messages.

-s

Specify the PBS server’s hostname or IP address.

SEE ALSO

`pbs_server(8B)` and the PBS External Reference Specification

A'.2 man qstat

NAME

`qstat` – show status of pbs batch jobs

SYNOPSIS

```
qstat [-f [-1]] [-l] [-W site_specific] [-x] [job_identifier... | destination...]
qstat [-a|-i|-r|-e] [-l] [-n [-1]] [-s] [-G|-M] [-R] [-u user_list] [job_identifier...
| destination...]
qstat -Q [-f [-1]] [-W site_specific] [-l] [destination...]
qstat -q [-G|-M] [-l] [destination...]
qstat -B [-f [-1]] [-W site_specific] [-l] [server_name...]
qstat -t
```

DESCRIPTION

The `qstat` command is used to request the status of jobs, queues, or a batch server. The requested status is written to standard out.

When requesting job status, synopsis format 1 or 2, `qstat` will output information about each *job_identifier* or all jobs at each *destination*. Jobs for which the user does not have status

privilege are not displayed.

When requesting queue or server status, synopsis format 3 through 5, qstat will output information about each *destination*.

OPTIONS

-f

Specifies that a full status display be written to standard out.

-a

“All” jobs are displayed in the alternative format, see the Standard Output section. If the operand is a destination id, all jobs at that destination are displayed. If the operand is a job id, information about that job is displayed.

-e

If the operand is a job id or not specified, only jobs in executable queues are displayed. Setting the PBS_QSTAT_EXEONLY environment variable will also enable this option.

-i

Job status is displayed in the alternative format. For a destination id operand, status for jobs at that destination which are not running are displayed. This includes jobs which are queued, held or waiting. If an operand is a job id, status for that job is displayed regardless of its state.

-r

If an operand is a job id, status for that job is displayed. For a destination id operand, status for jobs at that destination which are running are displayed, this includes jobs which are suspended.

-n

In addition to the basic information, nodes allocated to a job are listed.

-1

In combination with **-n**, the **-1** option puts all of the nodes on the same line as the job ID. In combination with **-f**, attributes are not folded to fit in a terminal window. This is intended to ease the parsing of the qstat output.

-s

In addition to the basic information, any comment provided by the batch administrator or scheduler is shown.

-G

Show size information in giga-bytes.

-M

Show size information, disk or memory in mega-words. A word is considered to be 8 bytes.

-R

In addition to other information, disk reservation information is shown. Not applicable to all systems.

-t

Normal qstat output displays a summary of the array instead of the entire array, job for job. qstat **-t** expands the output to display the entire array. Note that arrays are now named with brackets following the array name; for example:

```
echo sleep 20 | qsub -t 0-299 189[] .pali
```

Individual jobs in the array are now also noted using square brackets instead of dashes; for example, here is part of the output of qstat **-t** for the preceding array:

```
189[299] .napali STDIN[299] dbeer 0 Q batch
```

-u

Job status is displayed in the alternative format. If an operand is a job id, status for that job is displayed. For a destination id operand, status for jobs at that destination which are owned by the user(s) listed in *user_list* are displayed. The syntax of the *user_list* is:

user_name[@host] [,user_name[@host],...]

Host names may be wild carded on the left end, e.g. “*.nasa.gov”. User_name without a “?” is equivalent to “user_name@*”, that is at any host.

-Q

Specifies that the request is for queue status and that the operands are destination identifiers.

-q

Specifies that the request is for queue status which should be shown in the alternative format.

-B

Specifies that the request is for batch server status and that the operands are the names of servers.

-x

Specifies that the output is to be displayed in XML form. This option is only valid with the -f option or by itself, which will also specify the -f full status display.

-l

Specifies that the long name of the job (or the job name appended with the suffix alias) should be displayed.

OPERANDS

If neither the **-Q** nor the **-B** option is given, the operands on the qstat command must be either job identifiers or destinations identifiers.

If the operand is a job identifier, it must be in the following form:

sequence_number [.server_name] [@server]

where **sequence_number.server_name** is the job identifier assigned at submittal time, see **qsub**. If the **.server_name** is omitted, the name of the default server will be used. If **@server** is supplied, the request will be for the job identifier currently at that Server.

If the operand is a destination identifier, it is one of the following three forms:

queue

@server
queue@server

If **queue** is specified, the request is for status of all jobs in that queue at the default server. If the **@server** form is given, the request is for status of all jobs at that server. If a full destination identifier, **queue@server**, is given, the request is for status of all jobs in the named queue at the named server.

If the **-Q** option is given, the operands are destination identifiers as specified above. If **queue** is specified, the status of that queue at the default server will be given. If **queue@server** is specified, the status of the named queue at the named server will be given. If **@server** is specified, the status of all queues at the named server will be given. If no destination is specified, the status of all queues at the default server will be given.

If the **-B** option is given, the operand is the name of a server.

STANDARD OUTPUT

Displaying Job Status

If job status is being displayed in the default format and the **-f** option is not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the job identifier assigned by PBS.
- the job name given by the submitter.
- the job owner
- the CPU time used
- the job state:
 - C - Job is completed after having run/
 - E - Job is exiting after having run.
 - H - Job is held.
 - Q - job is queued, eligible to run or routed.
 - R - job is running.
 - T - job is being moved to new location.
 - W - job is waiting for its execution time
(-a option) to be reached.
 - S - (Unicos only) job is suspend.
- the queue in which the job resides

If job status is being displayed and the **-f** option is specified, the output will depend on

whether qstat was compiled to use a Tcl interpreter. See the configuration section for details.

If Tcl is not being used, full display for each job consists of the header line:

Job Id: job identifier

Followed by one line per job attribute of the form:

attribute_name = value

If any of the options -a, -i, -r, -u, -n, -s, -G or -M are provided, the alternative display format for jobs is used. The following items are displayed on a single line, in the specified order, separated by white space:

- the job identifier assigned by PBS.
- the job owner.
- The queue in which the job currently resides.
- The job name given by the submitter.
- The session id (if the job is running).
- The number of nodes requested by the job.
- The number of cpus or tasks requested by the job.
- The amount of memory requested by the job.
- Either the cpu time, if specified, or wall time requested by the job, (hh:mm).
- The job's current state.
- The amount of cpu time or wall time used by the job (hh:mm).

If the -R option is provided, the line contains:

- the job identifier assigned by PBS.
- the job owner.
- The queue in which the job currently resides.
- The number of nodes requested by the job.
- The number of cpus or tasks requested by the job.
- The amount of memory requested by the job.
- Either the cpu time or wall time requested by the job.
- The job's current state.
- The amount of cpu time or wall time used by the job.
- The amount of SRFS space requested on the big file system.
- The amount of SRFS space requested on the fast file system.
- The amount of space requested on the parallel I/O file system.

The last three fields may not contain useful information at all sites or on all systems.

Note: Remaining walltime does not account for walltime multiplication factors.

Displaying Queue Status

If queue status is being displayed and the *-f* option was not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the queue name
- the maximum number of jobs that may be run in the queue concurrently
- the total number of jobs in the queue
- the enable or disabled status of the queue
- the started or stopped status of the queue
- for each job state, the name of the state and the number of jobs in the queue in that state.
- the type of queue, execution or routing.

If queue status is being displayed and the *-f* option is specified, the output will depend on whether qstat was compiled to use a Tcl interpreter. See the configuration section for details. If Tcl is not being used, the full display for each queue consists of the header line:

Queue: queue_name

Followed by one line per queue attribute of the form:

attribute_name = value

If the *-q* option is specified, queue information is displayed in the alternative format: The following information is displayed on a single line:

- the queue name
- the maximum amount of memory a job in the queue may request
- the maximum amount of cpu time a job in the queue may request
- the maximum amount of wall time a job in the queue may request
- the maximum amount of nodes a job in the queue may request
- the number of jobs in the queue in the running state
- the number of jobs in the queue in the queued state
- the maximum number (limit) of jobs that may be run in the queue concurrently
- the state of the queue given by a pair of letters:
 - either the letter E if the queue is Enabled or D if Disabled, and
 - either the letter R if the queue is Running (started) or S if Stopped.

Displaying Server Status

If batch server status is being displayed and the *-f* option is not specified, the following items

are displayed on a single line, in the specified order, separated by white space:

- the server name
- the maximum number of jobs that the server may run concurrently
- the total number of jobs currently managed by the server
- the status of the server
- for each job state, the name of the state and the number of jobs in the server in that state

If server status is being displayed and the *-f* option is specified, the output will depend on whether qstat was compiled to use a Tcl interpreter. See the configuration section for details. If Tcl is not being used, the full display for the server consist of the header line:

Server: server name

Followed by one line per server attribute of the form:

attribute_name = value

STANDARD ERROR

The qstat command will write a diagnostic message to standard error for each error occurrence.

CONFIGURATION

If qstat is compiled with an option to include a Tcl interpreter, using the *-f* flag to get a full display causes a check to be made for a script file to use to output the requested information. The first location checked is \$HOME/.qstatrc. If this does not exist, the next location checked is administrator configured. If one of these is found, a Tcl interpreter is started and the script file is passed to it along with three global variables. The command line arguments are split into two variable named **flags** and **operands**. The status information is passed in a variable named **objects**. All of these variables are Tcl lists. The **flags** list contains the name of the command (usually “qstat”) as its first element. Any other elements are command line option flags with any options they use, presented in the order given on the command line. They are broken up individually so that if two flags are given together on the command line, they are separated in the list. For example, if the user typed

```
qstat -QfWbigdisplay
```

the **flags** list would contain

```
qstat -Q -f -W bigdisplay
```

The **operands** list contains all other command line arguments following the flags. There will always be at least one element in **operands** because if no operands are typed by the user,

the default destination or server name is used. The **objects** list contains all the information retrieved from the server(s) so the Tcl interpreter can run once to format the entire output. This list has the same number of elements as the **operands** list. Each element is another list with two elements. The first element is a string giving the type of objects to be found in the second. The string can take the values “server”, “queue”, “job” or “error”. The second element will be a list in which each element is a single batch status object of the type given by the string discussed above. In the case of “error”, the list will be empty. Each object is again a list. The first element is the name of the object. The second is a list of attributes. The third element will be the object text. All three of these object elements correspond with fields in the structure **batch_status** which is described in detail for each type of object by the man pages for **pbs_statjob(3)**, **pbs_statque(3)**, and **pbs_statserver(3)**. Each attribute in the second element list whose elements correspond with the **attrl** structure. Each will be a list with two elements. The first will be the attribute name and the second will be the attribute value.

EXIT STATUS

Upon successful processing of all the operands presented to the qstat command, the exit status will be a value of zero.

If the qstat command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

qalter(1B), **qsub(1B)**, **pbs_alterjob(3B)**, **pbs_statjob(3B)**, **pbs_statque(3B)**, **pbs_statserver(3B)**, **pbs_submit(3B)**, **pbs_job_attributes(7B)**, **pbs_queue_attributes(7B)**, **pbs_server_attributes(7B)**, **pbs_resources_*(7B)** where * is system type, and the PBS ERS.

A'.3 man qsub

NAME

qsub – submit pbs job

SYNOPSIS

```
qsub [-a date_time] [-A account_string] [-b secs] [-c checkpoint_options]
[-C directive_prefix] [-d path] [-D path] [-e path] [-f] [-h] [-I] [-j join]
```

```
[-k keep] [-l resource_list] [-m mail_options] [-M user_list] [-N name] [-o
path] [-p priority] [-P proxy_username[:group]] [-q destination] [-r c] [-S
path_list] [-t array_request] [-T prologue/epilogue script_name] [-u user_list]
[-v variable_list] [-V] [-w] path [-W additional_attributes] [-x] [-X] [-z]
[script]
```

DESCRIPTION

To create a job is to submit an executable script to a batch server. The batch server will be the default server unless the **-q** option is specified. See discussion of PBS_DEFAULT under Environment Variables below. Typically, the script is a shell script which will be executed by a command shell such as sh or csh.

Options on the qsub command allow the specification of attributes which affect the behavior of the job.

The qsub command will pass certain environment variables in the *Variable_List* attribute of the job. These variables will be available to the job. The value for the following variables will be taken from the environment of the qsub command: **HOME**, **LANG**, **LOGNAME**, **PATH**, **MAIL**, **SHELL**, and **TZ**. These values will be assigned to a new name which is the current name prefixed with the string “PBS_O_”. For example, the job will have access to an environment variable named **PBS_O_HOME** which have the value of the variable **HOME** in the qsub command environment.

In addition to the above, the following environment variables will be available to the batch job.

PBS_O_HOST the name of the host upon which the qsub command is running.

PBS_SERVER the hostname of the pbs_server which qsub submits the job to.

PBS_O_QUEUE the name of the original queue to which the job was submitted.

PBS_O_WORKDIR the absolute path of the current working directory of the qsub command.

PBS_ARRAYID each member of a job array is assigned a unique identifier (see **-t**)

PBS_ENVIRONMENT set to **PBS_BATCH** to indicate the job is a batch job, or to **PBS_INTERACTIVE** to indicate the job is a PBS interactive job, see **-I** option.

PBS_JOBID the job identifier assigned to the job by the batch system.

PBS_JOBNAME the job name supplied by the user.

PBS_NODEFILE the name of the file contain the list of nodes assigned to the job (for parallel and cluster systems).

PBS_QUEUE the name of the queue from which the job is executed.

OPTIONS

-a date_time Declares the time after which the job is eligible for execution.

The *date_time* argument is in the form: [[[CC]YY]MM]DD]hhmm[.ss]

Where CC is the first two digits of the year (the century), YY is the second two digits of the year, MM is the two digits for the month, DD is the day of the month, hh is the hour, mm is the minute, and the optional SS is the seconds.

If the month, MM, is not specified, it will default to the current month if the specified day DD, is in the future. Otherwise, the month will be set to next month. Likewise, if the day, DD, is not specified, it will default to today if the time hhmm is in the future. Otherwise, the day will be set to tomorrow. For example, if you submit a job at 11:15am with a time of **-a 1110**, the job will be eligible to run at 11:10am tomorrow.

-A account_string Defines the account string associated with the job. The *account_string* is an undefined string of characters and is interpreted by the server which executes the job. See section 2.7.1 of the PBS ERS.

-b seconds Defines the maximum number of seconds qsub will block attempting to contact pbs_server. If pbs_server is down, or for a variety of communication failures, qsub will continually retry connecting to pbs_server for job submission. This value overrides the CLIENTRETRY parameter in torque.cfg. This is a non-portable TORQUE extension. Portability-minded users can use the PBS_CLIENTRETRY environmental variable. A negative value is interpreted as infinity. The default is 0.

-c checkpoint_options Defines the options that will apply to the job. If the job executes upon a host which does not support checkpoint, these options will be ignored.

Valid checkpoint options are:

none No checkpointing is to be performed.

enabled Specify that checkpointing is allowed but must be explicitly invoked by either the qhold or qchkpt commands.

shutdown Specify that checkpointing is to be done on a job at pbs_mom shutdown.

periodic Specify that periodic checkpointing is enabled. The default interval is 10 minutes and can be changed by the \$checkpoint_interval option in the mom config file or by specifying an interval when the job is submitted

interval=minutes Checkpointing is to be performed at an interval of *minutes*, which

is the integer number of minutes of wall time used by the job. This value must be greater than zero.

depth=number Specify a number (depth) of checkpoint images to be kept in the checkpoint directory.

dir=path Specify a checkpoint directory (default is /var/spool/torque/checkpoint).

-C directive_prefix Defines the prefix that declares a directive to the qsub command within the script file. See the paragraph on script directives in the Extended Description section.

If the **-C** option is presented with a *directive_prefix* argument that is the null string, qsub will not scan the script file for directives.

-d path Defines the working directory path to be used for the job. If the **-d** option is not specified, the default working directory is the home directory. This option sets the environment variable PBS_O_INITDIR.

-D path Defines the root directory to be used for the job. This option sets the environment variable PBS_O_ROOTDIR.

-e path Defines the path to be used for the standard error stream of the batch job. The *path* argument is of the form:

[hostname:] [path_name] where **hostname** is the name of a host to which the file will be returned and **path_name** is the path name on that host in the syntax recognized by POSIX. The argument will be interpreted as follows:

path_name Where **path_name** is not an absolute path name, then the qsub command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the *hostname* component.

hostname: path_name Where **path_name** is not an absolute path name, then the qsub command will not expand the path name relative to the current working directory of the command. On delivery of the standard error, the path name will be expanded relative to the user's home directory on the **hostname** system.

path_name Where **path_name** specifies an absolute path name, then the qsub will supply the name of the host on which it is executing for the *hostname*.

hostname: path_name Where **path_name** specifies an absolute path name, the path will be used as specified. *hostname*.

hostname: Where **hostname** specifies the name of the host that the file should be returned to. The path will be the default file name.

If the **-e** option is not specified or the **path_name** is not specified or is specified and is a directory, the default file name for the standard error stream will be used. The default name has the following form:

job_name.esequence_number

where **job_name** is the name of the job, see **-N** option, and **sequence_number** is the job number assigned when the job is submitted.

-f Specifies that the job is fault tolerant. The *fault_tolerant* attribute will be set to true, which indicates that the job can survive the loss of a mom other than the “mother superior” mom (the first node in the exec hosts)

-h Specifies that a user hold be applied to the job at submission time.

-I Declares that the job is to be run “interactively”. The job will be queued and scheduled as any PBS batch job, but when executed, the standard input, output, and error streams of the job are connected through qsub to the terminal session in which qsub is running. Interactive jobs are forced to not rerunable. See the “Extended Description” paragraph for addition information of interactive jobs.

-j join Declares if the standard error stream of the job will be merged with the standard output stream of the job.

An option argument value of **oe** directs that the two streams will be merged, intermixed, as standard output. An option argument value of **eo** directs that the two streams will be merged, intermixed, as standard error.

If the *join* argument is **n** or the option is not specified, the two streams will be two separate files.

-k keep Defines which (if either) of standard output or standard error will be retained on the execution host. If set for a stream, this option overrides the path name for that stream. If not set, neither stream is retained on the execution host.

The argument is either the single letter “e” or “o”, or the letters “e” and “o” combined in either order. Or the argument is the letter “n”.

e The standard error stream is to retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by: **job_name.esequence** where **job_name** is the name specified for the job, and **sequence** is the sequence number

component of the job identifier.

- o The standard output stream is to be retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by: **job_name.osequence** where **job_name** is the name specified for the job, and **sequence** is the sequence number component of the job identifier.
- eo Both the standard output and standard error streams will be retained.
- oe Both the standard output and standard error streams will be retained.
- n Neither stream is retained.

-l resource_list Defines the resources that are required by the job and establishes a limit to the amount of resource that can be consumed. If not set for a generally available resource, such as CPU time, the limit is infinite. The *resource_list* argument is of the form:
resource_name[=[value]][,resource_name[=[value]],...]

-m mail_options Defines the set of conditions under which the execution server will send a mail message about the job. The *mail_options* argument is a string which consists of either the single character “n”, or one or more of the characters “a”, “b”, and “e”.

If the character “n” is specified, no mail will be sent.

For the letters “a”, “b”, and “e”:

- a mail is sent when the job is aborted by the batch system.
- b mail is sent when the job begins execution.
- e mail is sent when the job terminates.

If the *-m* option is not specified, mail will be sent if the job is aborted.

-M user_list Declares the list of users to whom mail is sent by the execution server when it sends mail about the job.

The *user_list* argument is of the form:

user[@host][,user[@host],...]

If unset, the list defaults to the submitting user at the qsub host, i.e. the job owner.

-N name Declares a name for the job. The name specified may be up to and including 15 characters in length. It must consist of printable, non white space characters with the first character alphabetic.

If the **-N** option is not specified, the job name will be the base name of the job script file specified on the command line. If no script file name was specified and the script was read from the standard input, then the job name will be set to STDIN.

-o path Defines the path to be used for the standard output stream of the batch job. The *path* argument is of the form:

[hostname:]path_name

where **hostname** is the name of a host to which the file will be returned and **path_name** is the path name on that host in the syntax recognized by POSIX. The argument will be interpreted as follows:

path_name Where **path_name** is not an absolute path name, then the qsub command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the *hostname* component.

hostname:path_name Where **path_name** is not an absolute path name, then the qsub command will not expand the path name relative to the current working directory of the command. On delivery of the standard output, the path name will be expanded relative to the user's home directory on the **hostname** system.

path_name Where **path_name** specifies an absolute path name, then the qsub will supply the name of the host on which it is executing for the *hostname*

hostname:path_name

Where **path_name** specifies an absolute path name, the path will be used as specified. *hostname*.

hostname: Where **hostname** specifies the name of the host that the file should be returned to. The path will be the default file name.

If the **-o** option is not specified or the **path_name** is not specified or is specified and is a directory, the default file name for the standard output stream will be used. The default name has the following form:

job_name.osequence_number

where **job_name** is the name of the job, see **-N** option, and **sequence_number** is the job number assigned when the job is submitted.

-p priority Defines the priority of the job. The *priority* argument must be an integer between -1024 and +1023 inclusive. The default is no priority which is equivalent to a priority of zero.

-P proxy_user[:group] Proxy user for whom the job should be submitted. This option is only available for the super user.

-q destination Defines the destination of the job. The *destination* names a queue, a server, or a queue at a server.

The qsub command will submit the script to the server defined by the *destination* argument. If the destination is a *routing queue*, the job may be routed by the server to a new destination.

If the **-q** option is not specified, the qsub command will submit the script to the default server. See PBS_DEFAULT under the Environment Variables section on this man page and the PBS ERS section 2.7.4, “Default Server”.

If the **-q** option is specified, it is in one of the following three forms:

queue

@server

queue@server

If the *destination* argument names a queue and does not name a server, the job will be submitted to the named queue at the default server.

If the *destination* argument names a server and does not name a queue, the job will be submitted to the default queue at the named server.

If the *destination* argument names both a queue and a server, the job will be submitted to the named queue at the named server.

-r y|n Declares whether the job is rerunable. See the **qrerun** command. The option argument is a single character, either **y** or **n**.

If the argument is “y”, the job is rerunable. If the argument is “n”, the job is not rerunable. The default value is ‘y’, rerunable.

-S path_list Declares the shell that interprets the job script.

The option argument *path_list* is in the form:

*path[@host][,path[@host],...]**

Only one path may be specified for any host named. Only one path may be specified without the corresponding host name. The path selected will be the one with the host

name that matched the name of the execution host. If no matching host is found, then the path specified without a host will be selected, if present.

If the **-S** option is not specified, the option argument is the null string, or no entry from the *path_list* is selected, the execution will use the user's login shell on the execution host.

-t array_request Specifies the task ids of a job array. Single task arrays are allowed.

The *array_request* argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimited list. Examples : -t 1-100 or -t 1,10,50-100

An optional slot limit can be specified to limit the amount of jobs that can run concurrently in the job array. The default value is unlimited. The slot limit must be the last thing specified in the array_request and is delimited from the array by a percent sign (%).

```
qsub script.sh -t 0-299%5
```

This sets the slot limit to 5. Only 5 jobs from this array can run at the same time.

Note: You can use qalter to modify slot limits on an array. The server parameter max_slot_limit can be used to set a global slot limit policy.

-T script_name Allows for per job prologue and epilogue scripts. The full script name will be prologue.[name] or epilogue.[name]. For the job submission, only request the name of the prologue or epilogue script.

Example: `qsub -T prescript`

Specifies to use the script prologue.prescript

-u user_list Defines the user name under which the job is to run on the execution system.

The *user_list* argument is of the form:

`user[@host][,user[@host],...]`

Only one user name may be given per specified host. Only one of the **user** specifications may be supplied without the corresponding **host** specification. That user name will be used for execution on any host not named in the argument list. If unset, the user list defaults to the user who is running qsub.

-v variable_list Expands the list of environment variables that are exported to the job.

In addition to the variables described in the “Description” section above, *variable_list* names environment variables from the qsub command environment which are made available to the job when it executes. The *variable_list* is a comma separated list of strings of the form **variable** or **variable=value**. These variables and their values are passed to the job.

-V Declares that all environment variables in the qsub command’s environment are to be exported to the batch job.

-w path Defines the working directory path to be used for the job. If the **-w** option is not specified, the default working directory is the current directory. This option sets the environment variable PBS_O_WORKDIR.

-W additional_attributes The **-W** option allows for the specification of additional job attributes. The general syntax of the **-W** is in the form:

-W attr_name=attr_value[attr_name=attr_value...]

Note if white space occurs anywhere within the option argument string or the equal sign, “=”, occurs within an *attribute_value* string, then the string must be enclosed with either single or double quote marks.

PBS currently supports the following attributes within the **-W** option.

depend=dependency_list Defines the dependency between this and other jobs. The *dependency_list* is in the form: *type[:argument[:argument...]][,type:argument...]*.

The *argument* is either a numeric count or a PBS job id according to *type*. If *argument* is a count, it must be greater than 0. If it is a job id and not fully specified in the form **seq_number.server.name**, it will be expanded according to the default server rules which apply to job IDs on most commands. If *argument* is null (the preceding colon need not be specified), the dependency of the corresponding type is cleared (unset).

synccount:count This job is the first in a set of jobs to be executed at the same time. *Count* is the number of additional jobs in the set.

syncwith:jobid This job is an additional member of a set of jobs to be executed at the same time. In the above and following dependency types, *jobid* is the job identifier of the first job in the set.

after:jobid[:jobid...] This job may be scheduled for execution at any point after jobs *jobid* have started execution.

afterok:jobid[:jobid...] This job may be scheduled for execution only after jobs *jobid* have terminated with no errors. See the csh warning under “Extended Description”.

afternotok:jobid[:jobid...] This job may be scheduled for execution only after jobs *jobid* have terminated with errors. See the csh warning under “Extended Description”.

afterany:jobid[:jobid...] This job may be scheduled for execution after jobs *jobid* have terminated, with or without errors.

on:count This job may be scheduled for execution after **count** dependencies on other jobs have been satisfied. This form is used in conjunction with one of the **before** forms, see below.

before:jobid[:jobid...] When this job has begun execution, then jobs **jobid...** may begin.

beforeok:jobid[:jobid...] If this job terminates execution without errors, then jobs **jobid...** may begin. See the csh warning under “Extended Description”.

beforenotok:jobid[:jobid...] If this job terminates execution with errors, then jobs **jobid...** may begin. See the csh warning under “Extended Description”.

beforeany:jobid[:jobid...] When this job terminates execution, jobs **jobid...** may begin.

If any of the **before** forms are used, the jobs referenced by **jobid** must have been submitted with a dependency type of **on**.

Array Dependencies It is now possible to have a job depend on an array. These dependencies are in the form **depend=arraydep:arrayid[num]**. If [num] is not present, then the dependencies applies to the entire array. If [num] is present, then num means the number of jobs that must meet the condition for the dependency to be satisfied.

afterstartarray:arrayid[count] This job may be scheduled for execution only after jobs in *arrayid* have started execution.

afterokarray:arrayid[count] This job may be scheduled for execution only after jobs in *arrayid* have terminated with no errors.

afternotok:arrayid[count] This job may be scheduled for execution only after jobs in *arrayid* have terminated with errors.

afteranyarray:arrayid[count] This job may be scheduled for execution after jobs in *array id* have terminated, with or without errors.

beforerestartarray:arrayid[count] This job may be scheduled for execution only before jobs in *arrayid* have started execution.

beforeokarray:arrayid[count] This job may be scheduled for execution only before jobs in *arrayid* have terminated with no errors.

beforeenotok:arrayid[count] This job may be scheduled for execution only before jobs in *arrayid* have terminated with errors.

beforeanyarray:arrayid[count] This job may be scheduled for execution before jobs in *array id* have terminated, with or without errors.

If any of the **before** forms are used, the jobs referenced by **jobid** must have the same owner as the job being submitted. Otherwise, the dependency is ignored.

Error processing of the existence, state, or condition of the job on which the newly submitted job is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the new job will be deleted by the server. Mail will be sent to the job submitter stating the error.

Dependency examples:

```
qsub -W depend=afterok:123.big.iron.com /tmp/script  
qsub -W depend=before:234.hunk1.com:235.hunk1.com /tmp/script  
qsub -W depend=afterokarray:21.tom.com[] /tmp/script  
qsub -W depend=beforeenotokarray:22.tom.com[] [5] /tmp/script
```

group_list=g_list Defines the group name under which the job is to run on the execution system. The *g_list* argument is of the form:

group[@host][,group[@host],...]

Only one group name may be given per specified host. Only one of the **group** specifications may be supplied without the corresponding **host** specification. That group name will be used for execution on any host not named in the argument list. If not set, the *group_list* defaults to the primary group of the user under which the job will be run.

interactive=true If the interactive attribute is specified, the job is an interactive job. The **-I** option is an alternative method of specifying this attribute.

stagein=file_list, stageout=file_list Specifies which files are staged (copied) in before job start or staged out after the job completes execution. On completion of the job, all staged-in and staged-out files are removed from the execution system. The *file_list* is in the form

local_file@hostname:remote_file[,...]

regardless of the direction of the copy. The name **local_file** is the name of the file on the system where the job executed. It may be an absolute path or relative to the home directory of the user. The name **remote_file** is the destination name on the host specified by **hostname**. The name may be absolute or relative to the user's home directory on the destination host. The use of wildcards in the file name is not recommended. The file names map to a remote copy program (rcp) call on the execution system in the following manner:

For stagein: rcp **hostname:remote_file local_file**

For stageout: rcp **local_file hostname:remote_file**

Data staging examples:

-W stagein=/tmp/input.txt@headnode:/home/user/input.txt

-W stageout=/tmp/output.txt@headnode:/home/user/output.txt

If TORQUE has been compiled with wordexp support, then variables can be used in the specified paths. Currently only \$PBS_JOBID, \$HOME, and \$TMPDIR are supported for stagein.

umask=XXX Sets umask used to create stdout and stderr spool files in pbs_mom spool directory. Values starting with 0 are treated as octal values, otherwise the value is treated as a decimal umask value.

-x When running an interactive job, the **-x** flag makes it so that the script won't be parsed for PBS directives, but instead will be a command that is launched once the interactive job has started. The job will terminate at the completion of this command.

-X Enables X11 forwarding. The DISPLAY environment variable must be set.

-z Directs that the qsub command is not to write the job identifier assigned to the job to the command's standard output.

OPERANDS

The qsub command accepts a *script* operand that is the path to the script of the job. If the path is relative, it will be expanded relative to the working directory of the qsub command.

If the *script* operand is not provided or the operand is the single character “–”, the qsub command reads the script from standard input. When the script is being read from Standard Input, qsub will copy the file to a temporary file. This temporary file is passed to the library interface routine pbs_submit. The temporary file is removed by qsub after pbs_submit returns or upon the receipt of a signal which would cause qsub to terminate.

STANDARD INPUT

The qsub command reads the script for the job from standard input if the *script* operand is missing or is the single character “–”.

INPUT FILES

The *script* file is read by the qsub command. Qsub acts upon any directives found in the script.

When the job is created, a copy of the script file is made and that copy cannot be modified.

STANDARD OUTPUT

Unless the *-z* option is set, the job identifier assigned to the job will be written to standard output if the job is successfully created.

STANDARD ERROR

The qsub command will write a diagnostic message to standard error for each error occurrence.

ENVIRONMENT VARIABLES

The values of some or all of the variables in the qsub command’s environment are exported with the job, see the *-v* and *-V* options.

The environment variable **PBS_DEFAULT** defines the name of the default server. Typically, it corresponds to the system name of the host on which the server is running. If **PBS_DEFAULT** is not set, the default is defined by an administrator established file.

The environment variable **PBS_DPREFIX** determines the prefix string which identifies directives in the script.

The environment variable **PBS_CLIENTRETRY** defines the maximum number of seconds qsub will block. See the *-b* option above. Despite the name, currently qsub is the only client that

supports this option.

TORQUE.CFG

The torque.cfg file, located in PBS_SERVER_HOME (/var/spool/torque by default) controls the behavior of the qsub command. This file contains a list of parameters and values separated by whitespace

QSUSLEEP takes an integer operand which specifies time to sleep when running qsub command. Used to prevent users from overwhelming the scheduler.

SUBMITFILTER specifies the path to the submit filter used to pre-process job submission. The default path is \$(libexecdir)/qsub_filter, which falls back to /usr/local/sbin/torque_submitfilter for backwards compatibility. This torque.cfg parameter overrides this default.

SERVERHOST specifies the value for the PBS_SERVER environment variable

QSUBHOST specifies the hostname for the jobs QSUB_O_HOST variable

QSUBSENDUID specifies a uid to use for the jobs PBS_O_UID variable

XAUTHPATH specifies the path to xauth

CLIENTRETRY specifies the integer seconds between retry attempts to communicate with pbs_server

VALIDATEGROUP set this parameter to force qsub to verify the submitter's group id

DEFAULTCKPT specifies the default value for the jobs checkpoint attribute. The user overrides this with the -c qsub option.

VALIDATEPATH set this parameter to force qsub to validate local existence of a “-d” working directory

RERUNNABLEBYDEFAULT this parameter specifies if a job is rerunnable by default. The default is true, setting this to false causes the rerunnable attribute value to be false unless the user specifies otherwise with the -r option

FAULT_TOLERANT_BY_DEFAULT this parameter specifies if a job is fault tolerant by default. The default value for the fault_tolerant job attribute is false, setting this parameter to

true causes the default value of the attribute to be true. The user can specify their preference with the -f qsub option.

For example:

```
QSUBSLEEP 2
RERUNNABLEBYDEFAULT false
```

EXTENDED DESCRIPTION

Script Processing:

A job script may consist of PBS directives, comments and executable statements. A PBS directive provides a way of specifying job attributes in addition to the command line options.

For example:

```
:
#PBS -N Job_name
#PBS -l walltime=10:30,mem=320kb
#PBS -m be
#
step1 arg1 arg2
step2 arg3 arg4
```

The qsub command scans the lines of the script file for directives. An initial line in the script that begins with the characters “#!” or the character “:” will be ignored and scanning will start with the next line. Scanning will continue until the first executable line, that is a line that is not blank, not a directive line, nor a line whose first non white space character is “#”. If directives occur on subsequent lines, they will be ignored.

A line in the script file will be processed as a directive to qsub if and only if the string of characters starting with the first non white space character on the line and of the same length as the directive prefix matches the directive prefix.

The remainder of the directive line consists of the options to qsub in the same syntax as they appear on the command line. The option character is to be preceded with the “-” character.

If an option is present in both a directive and on the command line, that option and its argument, if any, will be ignored in the directive. The command line takes precedence.

If an option is present in a directive and not on the command line, that option and its argument, if any, will be processed as if it had occurred on the command line.

The directive prefix string will be determined in order of preference from:

The value of the **-C** option argument if the option is specified on the command line.

The value of the environment variable **PBS_DPREFIX** if it is defined.

The four character string **#PBS**.

If the **-C** option is found in a directive in the script file, it will be ignored.

User Authorization:

When the user submits a job from a system other than the one on which the PBS Server is running, the name under which the job is to be executed is selected according to the rules listed under the **-u** option. The user submitting the job must be authorized to run the job under the execution user name. This authorization is provided if

- (1) : The host on which qsub is run is trusted by the execution host (see /etc/hosts.equiv),
- (2) : The execution user has an .rhosts file naming the submitting user on the submitting host.

C-Shell .logout File:

The following warning applies for users of the c-shell, csh. If the job is executed under the csh and a *.logout* file exists in the home directory in which the job executes, the exit status of the job is that of the .logout script, not the job script. This may impact any inter-job dependencies. To preserve the job exit status, either remove the .logout file or place the following line as the first line in the .logout file

```
set EXITVAL = \$status
and the following line as the last executable line in .logout
exit \$EXITVAL
```

Interactive Jobs:

If the **-I** option is specified on the command line or in a script directive, or if the “interactive”

job attribute declared true via the **-W** option, **-W interactive=true**, either on the command line or in a script directive, the job is an interactive job. The script will be processed for directives, but will not be included with the job. When the job begins execution, all input to the job is from the terminal session in which qsub is running.

When an interactive job is submitted, the qsub command will not terminate when the job is submitted. Qsub will remain running until the job terminates, is aborted, or the user interrupts qsub with an SIGINT (the control-C key). If qsub is interrupted prior to job start, it will query if the user wishes to exit. If the user response “yes”, qsub exits and the job is aborted.

Once the interactive job has started execution, input to and output from the job pass through qsub. Keyboard generated interrupts are passed to the job. Lines entered that begin with the tilde (~) character and contain special sequences are escaped by qsub. The recognized escape sequences are:

- ~. Qsub terminates execution. The batch job is also terminated.
- ~susp** Suspend the qsub program if running under the C shell. “susp” is the suspend character, usually CNTL-Z.
- ~asusp** Suspend the input half of qsub (terminal to job), but allow output to continue to be displayed. Only works under the C shell. “asusp” is the auxiliary suspend character, usually CNTL-Y.

EXIT STATUS

Upon successful processing, the qsub exit status will be a value of zero.

If the qsub command fails, the command exits with a value greater than zero.

SEE ALSO

qalter(1B), qdel(1B), qhold(1B), qmove(1B), qmsg(1B), qrerun(1B), qrls(1B), qselect(1B), qsig(1B), qstat(1B), pbs_connect(3B), pbs_job_attributes(7B), pbs_queue_attributes(7B), pbs_resources_irix5(7B), pbs_resources_sp2(7B), pbs_resources_sunos4(7B), pbs_resources_unicos8(7B), pbs_server_attributes(7B) and pbs_server(8B)

Αναφορές

Anderson, G., Clough, S., Kneizys, F., Chetwynd, J., and Shettle, E. (1986). AFGL Atmospheric Constituent Profiles (0-120km). Technical Report AFGL-TR-86-0110, Air Force Geophysics Laboratory, Hanscom Air Force Base, Bedford, MA.