CONF-9804109--

KAPL-P-000062
(K-98011)

DORT AND TORT WORKSHOP - OUTLINE FOR PRESENTATION FOR PERFORMANCE ISSUES FOR LARGE PROBLEMS

A. Barnett

April, 1998

KAPL ATOMIC POWER LABORATORY SCHENECTADY, NEW YORK 12301

Operated for the U. S. Department of Energy
by KAPL, Inc. a Lockheed Martin company

[illegible]

Running Large TORT Problems

Allen Barnett

1.0 Problem Statement

1.1 Givens: A limited amount of time per job and limited amount of memory and disk space.

1.2 Solution: Must break up TORT run in time and space.

2.0 Time - Run multiple, sequential, dependent jobs

2.1 Method A

2.1.1 Set the flux iteration flag `nfxmx`=-1:

```
62$$ a2 1 -1 e / outer, inner
......................
nfxmx  =   -1 flux iteration max. per group per itn. (default=20; neg: limit by group in 21$$)
```

2.1.2 Set 21$$ (`itnbg`) to have a non-zero only in group 1 and fill with zeros:

```
21$$ 20  f0 /itns
......................
0      region/zn   matl/zn     dens/zn     impt/zn               itn/gp ...............
    1      1           1       1.00000E+00 1.00000E-02               10
    2      2           3       1.00000E+00 1.00000E+00                0
```

2.1.3 For job 1, set `ntflx`=0, `ntfog=#_ntfog`:

```
61$$ 0 1 e / input, output
......................
ntflx  =     0 flux guess input unit          if .gt. 0
ntfog  =     1 flux output unit               if .gt. 0
```

2.1.4 Run TORT and save the `fort.#_ntfog` file.

2.1.5 For subsequent jobs, examine iteration monitor of previous job and set 21$$ to zero for groups which have converged.

2.1.6 Retrieve the `fort.#_ntfog` file, call it `fort.#_ntflx` and set `ntflx=#_ntflux`, `ntfog=#_ntfog`:

```
61$$ 2 1 e / input, output
......................
ntflx  =     2 flux guess input unit          if .gt. 0
ntfog  =     1 flux output unit               if .gt. 0
```

2.1.7 Goto 2.1.4

Advantages: Each run ends at a group boundary so you can safely post-process the flux data.
Disadvantages: Requires that each run's input deck be modified by hand. Problem may run into time limit since each group may take a different number of iterations to converge.

2.2 Method B

2.2.1 Set flux iteration flag `nfxmx`=large value (sufficient to converge each group):

```
62$$ a2 1 20 e / outer, inner
......................
nfxmx  =    20 flux iteration max. per group per itn. (default=20; neg: limit by group in 21$$)
```

2.2.2 Set `tmax` to a value less than the job's time limit:

```
67** 1.7-20 1000. e / source mult., time
......................
tmax    = 1.00000E+03 maximum cpu min. for this problem   (0 ignored)
```

2.2.3 For job 1, set `ntflx`=0, `ntfog=#_ntfog`.

2.2.4 Run TORT and save the `fort.#_ntfog` file.

2.2.5 For job 2 and subsequent, retrieve the `fort.#_ntfog` file, call it `fort.#_ntflx` and set `ntflx=#_ntflux`, `ntfog=#_ntfog`.

2.2.6 Goto 2.2.4

Advantages: Straight forward, requires modifications to 2nd input deck, but 2nd and subsequent jobs all identical. Less likely to run into time limit.
Disadvantages: Does an extra iteration in each converged group on each restart (can drive problem out of convergence, converges higher energy groups to a tighter tolerance-> difficult to reproduce results).

2.3 Method C

2.3.1 Set flux iteration flag `nfxmx`=large value (sufficient to converge each group).

2.3.2 Set `tmax` to a value less than the job's time limit.

2.3.3 For job 1, set `ntflx`=0, `ntfog=#_ntfog`.

2.3.4 Run TORT and save the `fort.#_ntfog` file.

2.3.5 For subsequent jobs, set `nifcnv`=1 -> skip converged groups in `fort.#_ntflx:`

```
62$$ a35 1 e / skip converged
......................
nifcnv =      1 0: no effect; 1: skip cvgd grps on ntflx; +10: use rebal damping from ntflx
```

2.3.6 Retrieve the `fort.#_ntfog` file, call it `fort.#_ntflx` and set `ntflx=#_ntflux`, `ntfog=#_ntfog`.

2.3.7 Goto 2.3.4

Advantages: Straight forward, requires modifications to 2nd and subsequent input decks (but all the same), does not do extra iterations in converged groups.
Disadvantages: If TORT stops iterating in the middle of a group, then the restart will begin with the original values of PCR damping factors. This can occasionally drive the solution out of convergence (solution: set `nifcnv`=11, use rebal damping from `ntflx`!).

2.4 Notes

2.4.1 If using the time limit, `tmax`, note that TORT stops iterating only when `tmax` is exceeded at the end of an inner iteration. Therefore, allow at least one inner iteration's duration between `tmax` and the job's time limit. (And probably allow some more time for wrap up overhead.)

2.4.2 TORT restarts can be dramatically sped up by saving the direct access rather than the sequential file.

2.4.2.1 Set `ntflx` and `ntfog` to negative values:

```
61$$ -2 -1 e / input, output
......................
ntflx  =    -2 flux guess input unit          if .gt. 0
ntfog  =    -1 flux output unit               if .gt. 0
```

2.4.2.2 When a job is done, save both the `fort.#_ntfog` file AND the `fort.91` and `fort.95` files (flux moments and boundary fluxes).

2.4.2.3 For restarts, retrieve all three files and name `fort.#_ntfog` to `fort.#_ntflx`.

The disadvantage to this method is that the direct access file requires internal TORT routines to interpret. So, save the scalar flux file for postprocessing, too. Also, you cannot modify TORT's memory model and restart from an old direct access file (see below).

3.0 Space - use TORT internal memory conservation features

3.1 Memory

3.1.1 Memory usage is controlled by `locobj`. Specified in 1000 word blocks:

```
62$$ a17 1000 e / memory
......................
locobj =  1000 initial memory allocation, words*1000 (0 implies use default)
```

TORT tries to fit a problem into the available memory by:

3.1.1.1 Storing the entire problem (i.e., parameters, acceleration matrix, flux solution).

3.1.1.2 Storing a single flux group.

3.1.1.3 Partitioning the acceleration matrix.

3.1.1.4 Storing a limited number of planes of the flux solution.

Here's an example:

```
0end of primary input arrays              =        930
 end of secondary input arrays            =       1304
 end of general input arrays              =       4264
 end of indexing arrays                   =       5221
 end of work, geom, sweep, rebal, source  =    3606295 17979 3606295 2574650 2542788
 end of buffers, source-io, output arrays =    3606295 2554130 2406279
 end of user buffers                      =    3606296
0groups in memory=     2, PCR= 0, blocks per group=     1
0end of primary input arrays              =        930
 end of secondary input arrays            =       1304
 end of general input arrays              =       4264
 end of indexing arrays                   =       5221
 end of work, geom, sweep, rebal, source  =     689840 17979 689840 387105 355243
 end of buffers, source-io, output arrays =    2879027 366585 218734
 end of user buffers                      =    2879028
0groups in memory=     1, PCR= 0, blocks per group=     1
```

```
0end of primary input arrays             =         930
 end of secondary input arrays           =        1304
 end of general input arrays             =        4264
 end of indexing arrays                  =        5221
 end of work, geom, sweep, rebal, source  =      529463 17979 529463 203628 355411
 end of buffers, source-io, output arrays =     2718650 366753 218902
 end of user buffers                     =     2718651
0groups in memory=     1, PCR= 1, blocks per group=     1
0end of primary input arrays             =         930
 end of secondary input arrays           =        1304
 end of general input arrays             =        4264
 end of indexing arrays                  =        5221
 end of work, geom, sweep, rebal, source  =      429092 17979 429092 103257 255040
 end of buffers, source-io, output arrays =      996659 266382 118531
 end of user buffers                     =      996660
0groups in memory=     1, PCR= 1, blocks per group=     4
0final   memory requirement    =      996660
```

3.1.2 Finer control over the "blocking" can be achieved with `minblk` and `maxblk`. If you require that a certain number of k planes be in memory (for performance reasons), set `maxblk` to a number less than `km`. To enforce blocking, set `minblk` to the least number of planes. Normally these values are 0 which just lets TORT choose:

```
62$$ a19 1 0 e / enforce group-wise storage
......................
minblk = 1 minimum no. of k blocks per group (0: all groups stored in memory if possible)
maxblk = 0 maximum no. of k blocks per group (0: maximum is km)
```

3.2 Disk space

3.2.1 There is not much you can do to reduce disk space usage, except reduce the size of the problem. The largest file will be the flux solution, `(igm)(im)(jm)(km)(lm+1)^2` words. For example, TORT Test Problem 6 is `igm=2, im=117, jm=33, km=27, lm=1:`

```
logical unit 82 requires       832 bytes
logical unit 83 requires   1667952 bytes
logical unit 84 requires    833976 bytes
logical unit 91 requires  11675664 bytes
logical unit 92 requires   5837832 bytes
logical unit 93 requires   5837832 bytes
logical unit 94 requires   5741296 bytes
logical unit 95 requires    833976 bytes
```

3.2.2 However, if your off-line storage can only store finite sized files (such as a tape system which doesn't support tape spanning), you can use the lcmobj parameter to limit the size of any single file which TORT creates. Again, `lcmobj` is in 1000's words (note that these files may be slightly larger than requested due to additional OS control words):

```
62$$ a18 1000 e / file size
......................
lcmobj =  1000 file segment size, words*1000 (0 implies unlimited segment length)
```

4.0 Key Values

4.1 Besides simply getting the problem to run, postprocessing large TORT problems can occasionally present difficulties. TORT can help you out with a couple of easy to use features for editing computed values:

4.2 Key Fluxes - by default, TORT prints the scalar flux in the first cell (i=1, j=1, k=1) in the iteration monitor line:

```
0grp itn imfd jmfd kmfd*mx fx dv*mx dv fx*rebl*rebl err*max rebl*rb dv fx*grp rebl*key flux*neg fix
   1   1    8    8   10 1.00E+00 3.14E-01    0 0.00E+00 0.00E+00 0.00E+00-1.00E+00 1.58E-01 1.00E+00
```

If this is the only value of interest in the problem and three significant digits are enough, then you are done.

Otherwise, set `nkeyfx` to the number of positions in space where you would like the scalar flux printed. The positions themselves are given in the arrays, 22*, 23*, 24* (`ceyai`, `ceyaj`, `ceyak`). TORT locates the CELL in which these positions occur and prints the cell average scalar flux after each iteration. If `nkeyfx` is negative, then the key fluxes are only printed after the group converges. Here is an example from the primary TORT problem in the TORSET test problem YSETMAX:

```
62$$ a6 -10 e / key fluxs
...
22**  0 -2814 2814 q2  q5          /flux keys
23**  0 2r-692 2r692   q5
24**  5r130 5r480

......................
nkeyfx =   -10 length of key flux array (neg: print key fluxes last iteration only)
...
0       i key posn  j key posn  k key posn  i key indx  j key indx  k key indx
     1 0.00000E+00 0.00000E+00 1.30000E+02       5           5           5
     2-2.81400E+03-6.92000E+02 1.30000E+02       4           5           5
     3 2.81400E+03-6.92000E+02 1.30000E+02       6           5           5
     4-2.81400E+03 6.92000E+02 1.30000E+02       4           5           5
     5 2.81400E+03 6.92000E+02 1.30000E+02       6           5           5
     6 0.00000E+00 0.00000E+00 4.80000E+02       5           5           7
     7-2.81400E+03-6.92000E+02 4.80000E+02       4           5           7
     8 2.81400E+03-6.92000E+02 4.80000E+02       6           5           7
     9-2.81400E+03 6.92000E+02 4.80000E+02       4           5           7
    10 2.81400E+03 6.92000E+02 4.80000E+02       6           5           7
...
0grp itn imfd jmfd kmfd*mx fx dv*mx dv fx*rebl*rebl err*max rebl*rb dv fx*grp rebl*key flux*neg fix
   1   1    8    8   10 1.00E+00 3.14E-01    0 0.00E+00 0.00E+00 0.00E+00-1.00E+00 1.58E-01 1.00E+00
   1   2    2    9    1 6.61E-01 6.48E-03    9 6.99E-03 7.32E-01 6.96E-01 1.44E-01 2.49E-01 1.00E+00
   1   3    2    5    1 2.44E-01 2.58E-02    4-1.86E-03 7.21E-02 7.03E-02-1.68E-03 2.39E-01 1.00E+00
   1   4    9    1    2-4.14E-02 2.70E-02    3-1.89E-03-2.00E-02-1.89E-02-5.51E-03 2.38E-01 1.00E+00
   1   5    5    9    1-2.39E-02 1.35E-02    2-1.07E-03-6.28E-03-6.27E-03-8.95E-04 2.38E-01 1.00E+00
   1   6    2    1    1-5.44E-03 1.50E-02    3 6.83E-05-8.29E-04-7.72E-04 8.79E-05 2.38E-01 1.00E+00

 kfx 2.38339E-01 2.60313E-01 2.16980E-01 2.60313E-01 2.16980E-01   2.51230E-01 2.70373E-01
2.27926E-01 2.70373E-01 2.27926E-01
```

Note that all key fluxes are group-wise values.

4.3 Response Functions - another built-in editing capability of TORT is the computation of response functions. Here, a group-wise constant is multiplied by each group-wise flux and the product summed over all groups. Responses may be computed on a cell-wise or a region-wise basis.

nresp controls the number of different response functions to be edited. If nresp is less than zero, then only the region-wise values are printed. Two arrays are used to fold into the flux solution, 26* (dnres) and 27* (cnres). dnres is the response function per region; there must be (nresp)(izm) values in this array. The default is all 1's. cnres is the response function per energy group; there must be (nresp)(igm) values in this array. Again, the default is all 1's.

For the secondary TORT in the TORSET sample problem, YSETMAX, nresp is set to -3, dnres is unity and cnres is the free-in-air tissue kerma:

```
62$$ a8 -3 e / nresp
...
26**  f 1        27**  / n, gamma, then total  resp
 / 13n-7g free-in-air tissue kerma/ dna 37n-g set -- jvp 03jul84
 / n, g, total fia tis ker (rad*cm2/n,g)
   643617-14 +517832-14 f0
.......................
0region response integrals -- volume in last column, total in last row ...........
0 regn    resp  1       resp  2       resp  3       resp  4
     1  2.89418E-01  0.00000E+00  0.00000E+00  1.00146E+09
     2  7.76911E+00  0.00000E+00  0.00000E+00  1.57071E+10
...
region response averages -- volume in last column, overall in last row ..........
0 regn    resp  1       resp  2       resp  3       resp  4
     1  2.88996E-10  0.00000E+00  0.00000E+00  1.00146E+09
     2  4.94624E-10  0.00000E+00  0.00000E+00  1.57071E+10
...
0key responses ............................
0 key     resp  1       resp  2       resp  3
     1  5.89278E-11  0.00000E+00  0.00000E+00
     2  2.38074E-10  0.00000E+00  0.00000E+00
...
```

Notes:

4.3.1 Only the first response function has non-zero values for cnres, so the other two response functions are zero.

4.3.2 The first group of values are the region integrals. The second group of values are the region average responses. The region volume is given as the nresp+1 response.

4.3.3 If key fluxes are specified, then the corresponding key responses are also printed.

4.3.4 TORT cannot handle material dependent response fluxes (called activity edits in DORT).