KAPL-P-000063
(K-98012)
CONF-9804109--

DORT AND TORT WORKSHOP - OUTLINE FOR PRESENTATION FOR SPLICING WITH TORSED AND TORSET

A. Barnett

April, 1998

NOTICE

KAPL ATOMIC POWER LABORATORY SCHENECTADY, NEW YORK 12301

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

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

CONF-9804109--

Splicing with TORSED and TORSET

Allen Barnett

1.0 Problem Statement

1.1 You want to solve a problem which is larger than can be accommodated by the computer system at your disposal.

Large size can result from two constraints:

1.1.1 The available memory of the machine is too small to contain the problem (even with TORT's memory conservation features, c.f. Running Large TORT Problems).

1.1.2 Individual files may be too large to store on-line.

1.2 You want to alter only a subset of the solution space of a larger problem (as in a perturbation problem) and don't want to rerun the entire problem.

2.0 TORSED RZ to XYZ Splicing

2.1 If the basic shape of your problem is cylindrical and azimuthally uniform, with only a small region of three-dimensionality, then the best splicing method is the TORSED - DORT to TORT splice.

2.2 The ENTIRE volume of the problem is solved in 2D RZ geometry in DORT. Directional fluxes from DORT solution are then interpolated onto the six surfaces of the TORT geometry. The TORT geometry must be entirely contained within the (3D) volume of the DORT solution.

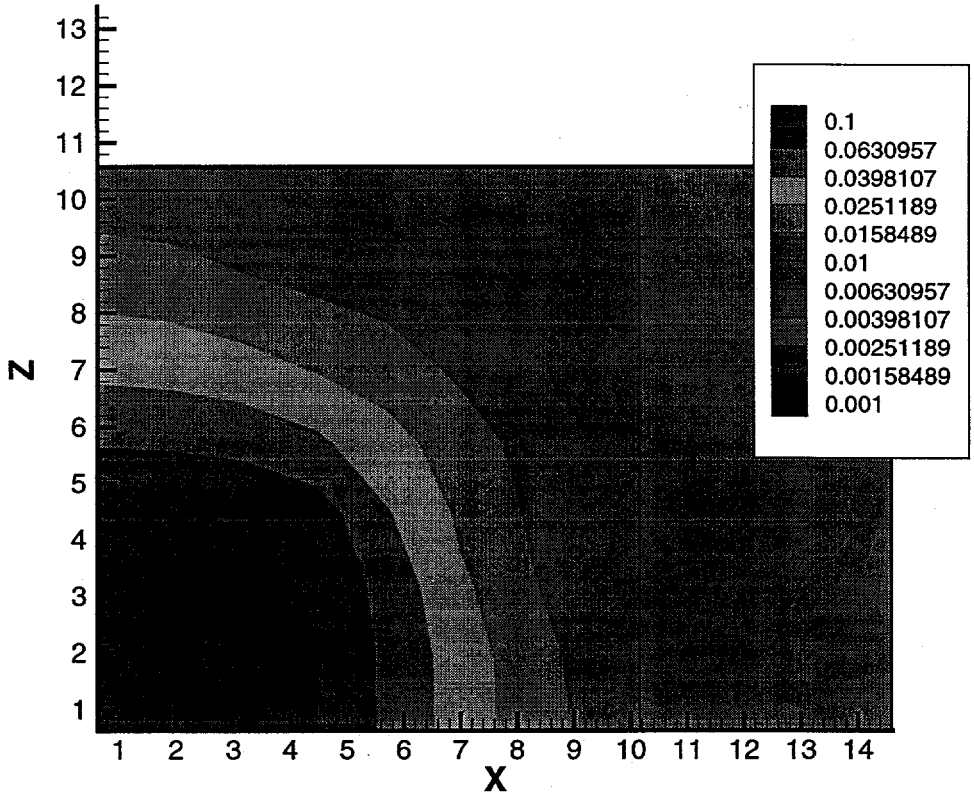
2.3 Besides setting up the geometry, there is only one (non-obvious) decision which must be made: How to interpolate from the DORT quadrature to the TORT quadrature. Two methods are available:

2.3.1 Look-backwards: each TORT ordinate is compared to each DORT ordinate; the directional flux from the "closest" DORT ordinate is copied to the TORT ordinate.

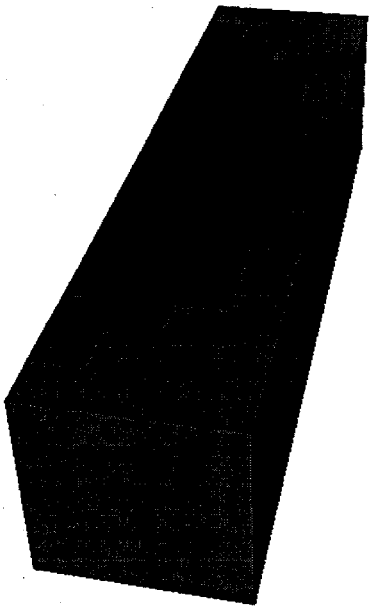
2.3.2 Look-forward: the directional flux from each DORT ordinate is apportioned among the "closest" TORT ordinates such that the scalar flux and current are approximately equivalent between the two quadrature representations.

Which method to choose is not really much of a decision: use look-forward. (The look-backwards method, while somewhat faster than look-forward, may miss DORT directions which have a strong directional component. A prime example of this would be to use GRTUNCL to generate an uncollided source from a point source: each cell in the DORT solution will have a strong source in ONE direction. If the TORT quadrature doesn't happen to have an ordinate in that direction, look-backward interpolation might miss it.)

2.4 Setting up a TORSED splice. This is the DORT geometry for the DOORS sample problem YSEDREAC:



It is a water cylinder 8" in diameter. Note that the model is extended with a 2" layer of air. The TORT geometry is shown as the dotted region. Why the air? Consider the



TORT geometry. It is only entirely inside the water along the centerline of the 2D geometry. Away from the XZ plane, the TORT geometry pokes out into the air surrounding the water cylinder.

2.4.1 Modify the DORT input to generate both the flux moments and the directional flux over the region of the secondary TORT problem. First, set `ntfog` to a unit number for the flux moments and `ntdir` to a unit number to contain the directional flux. Also set `idirf` to 2 to save the data (or 1 if you want a print-out too->WARNING: can produce large amounts of data!). As a disk space conservation feature, you must also set the axial range of the directional fluxes which are saved; use `jdirf` and `jdirl`. Note that DORT saves the CELL-CENTERED directional fluxes; be sure that the Z height of the mid-point of `jdirf` is strictly less than the bottom surface of the TORT geometry and that the Z height of the mid-point of `jdirl` is strictly greater than the top surface of the TORT geometry. DORT outputs the entire X row, so you don't have to be particularly careful here. Here is the relevant input for YSEDREAC:

```
61$$ 0 21 a10 22 e / input flux, output flux, directional flux
62$$ a40 2 4 9 e / output dir.flx(no print), z range 4 to 9
.......................
ntflx  =     0         flux guess input unit           if.gt.0
ntfog  =    21         flux output unit                if.gt.0
...
ntdir  =    22         directional flux output unit    if.gt.0
...
idirf  =     2         0=directional flux not saved; 1=saved and printed; 2=saved but not printed
jdirf  =     4         first axial interval for directional flux output  (0=no effect)
jdirl  =     9         last  axial interval for directional flux output  (0=no effect)
```

2.4.2 Now, the hard to understand part: DORT writes out the cell-centered directional fluxes after the completion of the inner iteration (since that is when the data is available: ALL discrete ordinates codes routinely throw this information away). However, iteration convergence is NOT tested until after the acceleration procedure has been applied, therefore, the scalar flux obtained by integrating the directional fluxes stored in the output file will be slightly different from the scalar flux used to determine convergence. Thus, you must use the VISA code to reconcile this difference. VISA takes the scalar flux from the flux moments file created by DORT (`ntfog`) and scales the directional fluxes in each cell such that the scalar flux integral of the directional flux is the same as the converged values.

The input to VISA for the YSEDREAC problem is:

```
1$$  9  4  9  000  0     0 21 22 0 23   5  6  4  9  0   1   e t
4$$  7i 4 12 t
.......................
nip    =     9 no. radial intervals output
jpl    =     4 first axial interval output
jpu    =     9 last  axial interval output
ned    =     0 edit ned source groups
norm   =     0 0: normalize to scalar flux; 1: do not
...
nflsv  =    21 logical no. scalar flux input      default=1 (old visa output if ntype.ge.10)
naft   =    22 logical no. directional flux input default=2 (may be 0 if ntype.ge.10)
nuncl  =     0 logical no. uncollided flux input  default=3 (may be 0)
ndata  =    23 logical no. source output          default=4
n5     =     5 logical no. standard input         default=5
```

```
n6    =     6 logical no. standard output        default=6
nj1   =     4 first axial interval input; 0 implies=1
njm   =     9 last  axial interval input; 0 implies=jm
```

Chiefly, note that the input units `nflsv` and `naft` correspond to `ntfog` and `ntdir` from the DORT input, and `nj1` and `njm` correspond to `jdirf` and `jdirl`. If you're folding in a GRTUNCL uncollided source, then `jpl` and `jpu` may be different from `jdirf` and `jdirl`, otherwise they should be the same as well. VISA writes the modified directional flux file to `ndata`.

2.4.3 TORSED prepares the boundary directional flux file for TORT. It must know both the mesh boundaries and the quadrature to be used in the TORT problem. The input is relatively straight-forward. `nvisa` is the directional flux output unit from VISA and `ntort` is the directional boundary flux file for TORT. `imti`, `jmt` and `kmt` are the `im`, `jm` and `km` values for TORT's mesh boundaries. `mmt` is the number of ordinates in the TORT quadrature. `ilook` is the look-forward(1)/look-backward(0) flag. The input for YSEDREAC is:

```
61$$  23 24 4 16 4  96 00  e t
 t
 /  s   8 full symmetric
  81**
  0 2r151235-7 0 4r113425-7 0 113425-7 11574-6 2r113425-7 11574-6
  113425-7 0 151235-7 2r113425-7 2r151235-7 2r113425-7 151235-7
    3q  24
  82**
  -30861-5 -21822-5 21822-5 -61721-5 -57735-5 -21822-5 21822-5 57735-5
  -8165-4 -7868-4 -57735-5 -21822-5 21822-5 57735-5 7868-4 -9759-4
  -95119-5 -7868-4 -57735-5 -21822-5 21822-5 57735-5 7868-4 95119-5
    3q  24
  83**
  3r-95119-5 5r-7868-4 7r-57735-5 9r-21822-5
    q  24 g  48
   t
 2** 3i5.08 10.16
 3** 15i-10.16 10.16
 4** 3i5.08 10.16
 t
.......................
nvisa =    23 visa input  unit number
ntort =    24 tort output unit number (0: direct access output; neg: both)
imti  =     4 no. tort i intervals; (neg=discont. mesh)
jmt   =    16 no. tort j intervals
kmt   =     4 no. tort k intervals
mmt   =    96 no. tort directions
locobj=     0 memory objective, words*1000  (0: ignored)
lcmobj =     0 file segment size, words*1000 (0 implies unlimited segment length)
ilook =     0 0=look backward; 1=look forward
nedit =     0 edit control (use 0)
rzero = 0.00000E+00 dort radius  of tort coordinate origin
zzero = 0.00000E+00 dort height  of tort coordinate origin
thzero= 0.00000E+00 ccw rotation of tort coordinates (deg)
flxmin= 0.00000E+00 minimum flux for log interpolation (0: use linear interpolation)
```

Although for this problem, the TORT geometry is located at the absolute position in XYZ space, it could just as easily be positioned at the origin. The `rzero` and `zzero` values (in the 62* array, which is in the same BLOCK as

the 61$ array) can be used to locate the origin of the TORT coordinate system in the DORT coordinate system. Similarly, if the TORT geometry is rotated with respect to the DORT geometry, `thzero` sets the counter-clockwise rotation of the TORT coordinates.

Finally, TORSED defaults to linear interpolation in space. If you set `flxmin` to a small positive value (usually use 1e-30), TORSED will do logarithmic spatial interpolation.

2.4.4 In order to use the boundary directional flux in TORT, set `ntbsi` to the output unit from TORSED (`ntort`). Ordinarily, this is the only source used. One can also set the boundary conditions to use whichever boundary values are desired. `ibl`, `ibr`, `ibi`, `ibo`, `ibb` and `ibt` control the boundary conditions. To use the boundary flux just defined, set each of these to 4. The sample problem input looks like:

```
61$$ a4 24 e  /source from torsed
63$$ a4 4 4 4 4 4 4 / bc = bndry source on all surfaces
.......................
ntbsi  =    24 boundary source input unit     if .gt. 0 (neg: input from 90+ arrays)
...
ibl    =     4 left    b.c.              (0=void         )
ibr    =     4 right   b.c.              (1=reflected    )
ibi    =     4 inside  b.c.              (2=periodic     )
ibo    =     4 outside b.c.
ibb    =     4 bottom  b.c.              (4=bndy source  )
ibt    =     4 top     b.c.
```

2.5 Additional notes:

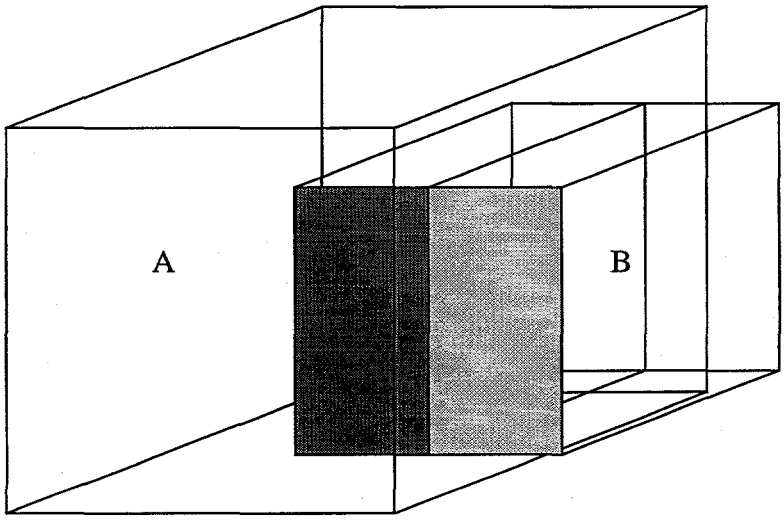
2.5.1 You are not obligated to use all of the boundary sources given in the input directional flux file. For instance, if the TORT problem stopped at the XZ plane, then one could use a reflecting boundary condition on the inner surface, `ibi=1`. If the TORT problem is near the top of the 2D geometry, then one could use a void boundary there, `ibt=0`.

2.5.2 TORSED is capable of outputting the direct access version of the boundary source file (`fort.95`). This is a much better solution than saving the sequential access file since TORT does not have to copy the boundary fluxes when it starts up. If you're saving the direct access flux moments file between dependent runs (see Running Large TORT Problems), then you need to save the `fort.95` file as well. One caveat is that TORSED needs to know the memory structure of TORT in order to build the proper direct access files; therefore, you may need to set `locobj` and `lcmobj` accordingly.

3.0 TORSET XYZ to XYZ Splicing

3.1 If there is no part of your problem which is azimuthally constant, then you might want to consider a TORT to TORT splice. In the TORT to TORT splice the secondary geometry does not have to be contained entirely inside the primary geometry. Those parts of the secondary geometry which are outside of the primary geometry

are assigned a boundary flux of zero. For example, B is only half inside A. The dark gray part of B's face will have non-zero directional flux surface values; the light gray part will be zero.

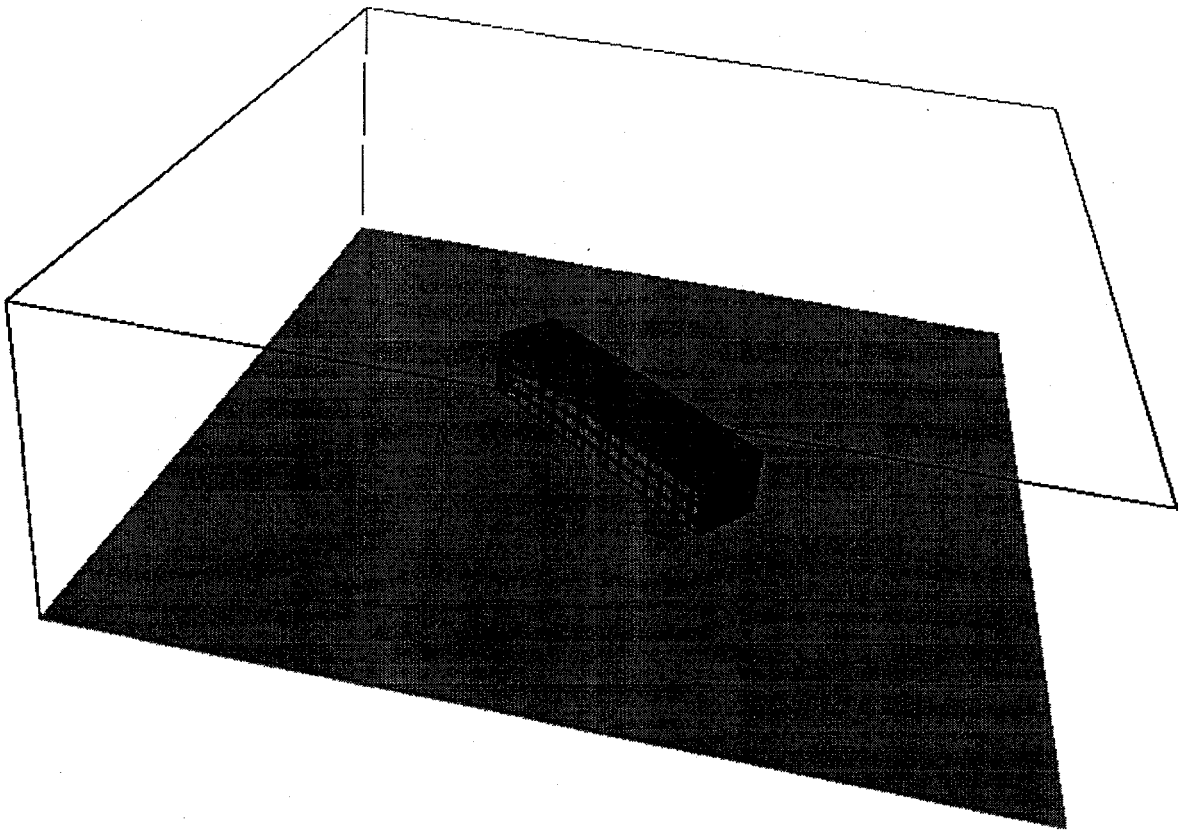


3.2 TORSET uses the look-forward method by default.

3.3 This example will follow the TORT sample problem, YSETMAX. The basic geometry is shown below. The source in the primary problem is on the left and top surfaces of the volume; it is uniform in space and isotropic in angle. (A good quiz question is how do you actually determine this from the input, since there is no explicit indication of a source of any kind!) Note that, in this case, the entire secondary geometry is inside the primary geometry.

3.3.1 In the primary TORT problem, one must save the directional fluxes over a "patch." This is defined first by setting `ntscl` to the scalar flux output unit and `ntdir` to the directional flux output unit, 61 and 62 for the sample problem. The volumetric range of directional fluxes is specified in the 34* (`pcnbn`) array:

```
61$$  a8 61 0 62 e /dir flux, scalar flux
...
34**  -50e2 50e2  q2  -15 3200 e    /output patch
.......................
ntscl  =    61 scalar flux output unit        if .gt. 0
ntzon  =     0 zone map input unit            if .gt. 0
ntdir  =    62 directional flux output unit   if .ne. 0 (neg: write reduced-size file)
...
0      k bndry     k midpoint  k width                  k cmesh   .  pchbn
     1-3.00000E+01-2.25000E+01 1.50000E+01                1    . -5.00000E+03
     2-1.50000E+01-1.00000E+01 1.00000E+01                2    .  5.00000E+03
     3-5.00000E+00-2.50000E+00 5.00000E+00                3    . -5.00000E+03
     4 0.00000E+00 3.00000E+01 6.00000E+01                4    .  5.00000E+03
     5 6.00000E+01 1.30000E+02 1.40000E+02                5    . -1.50000E+01
     6 2.00000E+02 3.00000E+02 2.00000E+02                6    .  3.20000E+03
```



The patch range is from -5000 cm to 5000 cm in X and Y and from -15 cm to 3200 cm in Z. Note that the dimensions of the secondary geometry are -3474.72 cm to 3474.72 cm in X, -838.2 cm to 838.2 cm in Y and -12.7 cm to 1493.52 cm in Z. Why is the patch so much larger in Y than the Y dimension of the secondary geometry? Because the secondary geometry is rotated with respect the primary geometry's mesh.

Ordinarily, one would want to limit the size of the overlap between the two parts of the problems because TORT must save ALL of the cell-centered directional fluxes in the patch volume. The size of this file may become the overall limiting factor in a TORT calculation! (Note: later editions of TORT can reduce the storage of directional fluxes if you limit the secondary geometry to be parallel to the primary geometry.)

3.3.2 Like TORSED, TORSET must be given the complete mesh and quadrature description of the secondary problem. TORSET also replaces VISA functionality in scaling the directional fluxes to correspond the final value of the scalar flux. TORSET is smart enough to be able to interrogate the primary TORT geometry from its input files. Thus, TORSET is somewhat easier to set up than TORSED. Here is the TORSED input and output from the YSET-MAX problem:

```
"large concrete building -- torset
61$$  62 61 0 0 00   0 2000 500 0 e
62$$  -117 33 27 140  e
66**  0 0 0 219 e
 t
... Secondary geometry description deleted ....
.....................
control parameter input ..........
ntdir =    62 directional flux input unit number  (neq: read reduced-size file)
ntscl =    61 scalar      flux input unit number
ntmap =     0 geometry input unit number (or 0)
ntunc =     0 uncollided  flux input unit number (or 0)#
ntort =     0 boundary source output unit number (0: direct access output; neg: both)
nedit =     0 use 0
locobj=  2000 memory objective, words*1000  (0: ignored)
lcmobj=   500 file segment size, words*1000 (0 implies unlimited segment length)
igmx  =     0 number of secondary groups (0: same as input)
--- directional flux read from unit  62 in dirraw   format
--- date----  user----  charge--  case----  time----
     01/08/98  barnett   gFSSH     55586     14:43:33
--- job title="air-ground environment -- tort
parameters from directional flux file ........
ipch1  =     3 left    patch boundary in iset=1
ipch2  =     7 right   patch boundary in iset=1
jpch1  =     3 inside  patch boundary in jset=1
jpch2  =     7 outside patch boundary in jset=1
kpch1  =     2 bottom  patch boundary
kpch2  =     9 top     patch boundary
ntbfo  =     0 boundary flux input unit
mmdnup =    30 maximum no. of down/up directions
mm     =    60 maximum no. of directions, largest m-set
mmsmsm =    60 total no. of directions, all m-sets
msm    =     1 no. of m-sets
igm    =    20 no. of energy groups
ntdir  =    62 directional flux input unit
--- flux guess read from unit  61 in varscl   format   (varscl=scalar flux; flxmom=flux moments)
--- date----  user----  charge--  case----  time----
     01/08/98  barnett   gFSSH     55586     14:45:00
 --- job title="air-ground environment -- tort
parameters from scalar flux file ........
im     =     9 no. of x intervals (neg: disc. mesh)
jm     =     9 no. of y intervals
km     =    10 no. of z intervals
igm    =    20 no. of energy groups
ism    =     1 no. of i-sets
jsm    =     1 no. of j-sets
imsism =     9 sum of i intervals over i-set
jmsjsm =     9 sum of j intervals over j-set
jmskm  =    90 sum of j intervals over k
geometric parameter input ..........
imt   =  -117 no. tort i intervals; (neg=discont. mesh)
jmt   =    33 no. tort j intervals
kmt   =    27 no. tort k intervals
mmt   =   140 no. tort secondary directions (0: use input quadrature)
real parameter input ..........
xzero = 0.00000E+00 x cordinate of secondary origin
yzero = 0.00000E+00 y cordinate of secondary origin
zzero = 0.00000E+00 z cordinate of secondary origin
thzero= 2.19000E+02 ccw rotation of secondary coord (deg)
flxmin= 0.00000E+00 minimum flux for log interpolation (0: use linear interpolation)
```

There are a couple of items worth noting:

3.3.2.1 If you have a binary TORT geometry file (also called the zone map, named `ntzon` in TORT input), then you can set `ntmap` to the unit and dispense with writing any TORT geometry input.

3.3.2.2 TORSET can also generate the direct access form of the boundary flux file and store it in the `fort.95` file (`ntort`=0). Again, if the secondary TORT problem uses memory blocking, you must specify the same memory size here (`locobj`). The same applies if you plan on using the split direct access file size (`lcmobj`).

3.3.3 The boundary fluxes are used in the secondary TORT problem in exactly the same way as with TORSED. Here, though, because the boundary flux is in direct access form, the input to TORT is slightly different. Instead of setting `ntdsi`, just set `ntfog` to a negative value and set `nifbnd` to 1. The boundary condition values, `ibl`, `ibr`, `ibi`, `ibo`, `ibb` and `ibt`, can be set in the same way as for a TORSED splice, namely to 4 to use the boundary fluxes in the input file, or to 0 or 1 if you want to force a void or reflected boundary condition. For the YSETMAX problem, for instance, the bottom boundary fluxes are probably not very meaningful so they are ignored:

```
61$$  00 -1  a4 00 a8 65 e   62$$  a36 1 e       /bndry source on dir access
63$$ a4 4 4  4 4 0 4 e
.......................
ntfog  =    -1 flux output unit              if .gt. 0
...
ibl    =     4 left    b.c.              (0=void         )
ibr    =     4 right   b.c.              (1=reflected    )
ibi    =     4 inside  b.c.              (2=periodic     )
ibo    =     4 outside b.c.
ibb    =     0 bottom  b.c.              (4=bndy source  )
ibt    =     4 top     b.c.
```