Experience from the EIRENE OpenMP parallelization and refactoring

and some general thougths

Y. Marandet, P. Genesio

H. Leggate (HLST/ACH)

TSVV#5 Regular VC, July 9th 2021

Objectives of EIRENE ‘core segregation’

➢ **Ring-fence parts of the code (as small as possible) which needs to be optimized, increasing readability & maintainability, clarity in order to**:

❑ Help improve the efficiency per core (identify and reduce main hotspots, allow better optimization by the compiler by e.g. reducing branching, improving structure, possibly changing tally structure ….)

❑ Enable large grids by domain decomposition (possibly on a reduced set of geometries)

Note : keep in mind that some aspect of pre-processing (Collisional Radiative model in each cell may become substantial in terms of computing time)

First attempt to define a ‘core’

- **The part which performs particle tracing and tally scoring**
- Located into eirmod_mcarlo.f, ~ 50 lines with 5 gotos (see next slide, would benefit from a - very careful - rewrite)

On the left – follow a particle, on the right follow any secondary particle (sputtered atoms, …)

```
C.................................................................
C  NEXT MONTE CARLO HISTORY
c
c   LAUNCH A NEW PARTICLE NOW
c.................................................................
          XMCP(ISTRA)=XMCP(ISTRA)+1.
          NPANU=NPANU+1

          IPANU=IPANU+1
          ITRJ = NCHORI + MOD(IPANU,NTRJ) + 1
          NLEVEL=0
          CALL EIRENE_LOCAT1(IPANU)


C  IS BIRTH PROCESS SURVIVED?
          IF (.NOT.LGPART) GOTO 110
C
 102      CONTINUE
C  FOLLOW NEUTRAL PARTICLE
          IF (ITYP.EQ.0.OR.ITYP.EQ.1.OR.ITYP.EQ.2) THEN
            CALL EIRENE_FOLNEUT
C  FOLLOW TEST ION
          ELSEIF (ITYP.EQ.3) THEN
            CALL EIRENE_FOLION
          ENDIF
C  NEXT GENERATION ?
          IF (LGPART) GOTO 102
C
 110      CONTINUE

          IF (NLRAY(ISTRA)) THEN
            CALL EIRENE_CLEAR_TRAJECTORY (ITRJ)
          END IF
```

```
C  NUMBER OF REMAINING NODES AND NUMBER OF LEVELS AT NEXT
NODE
         IF (NLEVEL.GT.0) THEN
 104        INODES=NODES(NLEVEL)-1
          NODES(NLEVEL)=INODES
          IF(INODES.LE.0) GO TO 103
C  RESTORE VARIABLES AND START NEW TRACK
          DO 105 J=1,NPARTC
           RPST(J)=RSPLST(J,NLEVEL)
 105        CONTINUE
          DO 106 J=1,MPARTC
           IPST(J)=ISPLST(J,NLEVEL)
 106        CONTINUE
          ITYP=ISPEZI(ISPZ,-1)
          IPHOT=ISPEZI(ISPZ,0)
          IATM=ISPEZI(ISPZ,1)
          IMOL=ISPEZI(ISPZ,2)
          IION=ISPEZI(ISPZ,3)
          IPLS=ISPEZI(ISPZ,4)
          CALL EIRENE_NCELLN(NCELL,NRCELL,NPCELL,NTCELL,NACELL,
     .     NBLOCK,NR1ST,NP2ND,NT3RD,NBMLT,NLRAD,NLPOL,NLTOR)
          NBLCKA=NSTRD*(NBLOCK-1)+NACELL
 NLSRFX=MRSURF.GT.0
          NLSRFY=MPSURF.GT.0
          NLSRFZ=MTSURF.GT.0
          NLSRFA=MASURF.GT.0
          IF (NLTRC) CALL EIRENE_CHCTRC(X0,Y0,Z0,0,12)


!  PARTICLE TYPE AND SPECIES MAY HAVE CHANGED
!  PREPARE POINTER FOR UNIFIED SUBROUTINES FPATH, UPDATE,
ETC.
          CALL EIRENE_SWITCH_PARTINFO

          IC_NEUT=0
          IC_ION=0
          GOTO 102
C  RETURN TO PREVIOUS LEVEL
 103        CONTINUE
          NLEVEL=NLEVEL-1
          IF(NLEVEL.GT.0) GOTO 104
         ENDIF
C  HISTORY HAS ENDED
```

First attempts to define a ‘core’

- **However these 50 lines have calls to routines in particle-tracing/,**

- which then call routines in surface-processes/, volume_processes/, scoring/ folders; with several associated modules in modules/
- In the 4 folders above : 18594+12718+12718 + 4339 = 48369 lines ~ 33% (Total lines in the code (develop_openmp) = 136996 lines)
- order of magnitude estimate but shows the current complexity/size of the routines performing the core functions …

How to reduce the amount of code to deal with ?

Limit strictly to moving particles within the grid : fol*.f, eirmod_time*.f ??

Possible ways towards simplification

- In fol*, time* routines, several *(5-10) select case* or *if/elseif* or *gotos* structures on the geometry option (LEVGEO)

  LEVGEO=1-5 ; 10 (7 options, 2 used in coupled cases 4 = triangles and 10 for EMC3)

  find an intelligent way to disentangle this (+ focus on the most relevant options for optimizing ?)
(NB : the code is specific to each options)

  use an OOP approach as nicely illustrated by Jorge ?

- in fpath.f, lots of branching on MODCOL array (defining how various rates are calculted to obtain mean free paths) – **same thing, can we cleverly disetangle this ? OOP ?**
- update.f : lots of 'if' to check if tallies active or not at every call

- 'Streamlining' these routines

  - should make the code more readable

  - may also help enable further compiler optimization of the code

(is this really true with OOP ? (Strongly dependant on array of structures vs structures of array performances for instance ?)

Experience from OpenMP parallelization

- ➢ Use the routines where $!OMP pragmas are used as an alternative indication of how many routines may actually be called during particle tracing – depends of course on input parameters

  - > 55 routines (mostly in the folders discussed previously + some others scattered around, some re-ordering might be in line)

- ➢ Most of the time is spent in routine update.f, to score tallies.

  Scoring after trajectories may improve things substantially (for OpenMP but also more more generally)

  Area where optimization could play a large role on single core performance also

Some general thoughts in conclusion

- Given the time frame and resources of the project an **incremental approach** is the only way to reach deliverables – this rules out
  - changing coding langage (C++)
  - full rewriting of the 'core' – the later, based on particle tracing + scoring, is still laaaarge

  (in spite of the fact that a full rewrite *could* bring large benefits on the long term).

- **OOP fortran features** (modules, derived types & interfaces) already there in bits and pieces, build gradually on these good programming practices. Jorge's work nicely shows how this can be used in practice.

- 'User routines' are where users will first be exposed to fortran and inner workings to implement specific features (standalone or coupled)

  no straightforward way this can be easily replaced by scripts.

  The **preprocessor is already written in fortran**, doing it again with scripts will take a lot of resources.