Modification of EIRENE neutral particle Monte Carlo Code for improved Computing performance and application to ITER Edge Plasma Modelling: Final Report on Contract FU07-CT-2007-00007 (EFDA/07-1577)

Vladislav Kotov, Petra Börner, Detlev Reiter
Institut für Energieforschung - Plasma Physik,
Forschungszentrum Jülich GmbH

April 28, 2008

Contents

1 Overview

1.1 Parallelization Status of EIRENE prior to this project

In linear Monte Carlo transport applications the performance is not measured by the number of test particles (histories) in a given CPU-time, but instead by the variance per cpu time [1, 2], which is the true ***figure of merit*** of a Monte Carlo code.

Improving efficiency of Monte Carlo Codes hence requires both programming and algorithmic optimization.

Beyond this, and in order to gain efficiency in terms of ***turn around time*** on a specific computing system, as it is e.g. essential for high density ITER SOL simulations, parallelization of Monte Carlo transport solvers is an alternative, and very powerful option.

The Monte Carlo transport algorithm is inherent suitable for parallelization [3, 4]. The Monte Carlo particle trajectories are independent of each other, as long as the real (physical) particles do not interact with each other but only with their background host medium.

Interaction between test particles (such as neutral-neutral or neutral-photon interactions in EIRENE) are treated in EIRENE by successive linearization and iteration [5] , hence the above statements fully apply to the EIRENE code.

A complication arises because of the special feature in EIRENE to “stratify sources”, i.e. to de-compose the primary source for test particles into a sum of sub-sources (= “strata”) and to obtain the final result by linear superposition of results from such individual strata (see [1], under “stratified sources sampling”)

This option has severe consequences for the parallization strategy. On the one hand it provides further flexibility for optimization but on the other hand it also leads to load balancing issues which have to be dealt with. Most of the material (except for the purely technical descriptions in Section 5 and 6) in the report below is related to this issue.

The status of parallelization of stand alone EIRENE prior to the present project, which was mainly developed in 2000–2003 for stand alone applications and for the (single stratum-) interface to the 3D edge code EMC3, is described in detail in diploma-thesis [6].

In the present project this material is used, implemented into the ITER code version of B2-EIRENE and extended from the present single stratum EMC3-EIRENE (3D edge code) concepts towards multi-strata B2-EIRENE (2D edge code) applications.

1.1.1 Stratified Sampling in EIRENE

As the “stratified source sampling concept” of EIRENE is essential for understanding the rest of this report, a brief description of this concept is given next:

The splitting up to the total primary source for neutral particles (including photons) into different strata can be done according to various criteria, but in particular also according to the different physics involved for different primary source particles, such as their species (hydrogen, atoms, molecules, impurities, etc. and/or according to their birth location(e.g. inner target, wall, outer target, gas puff, volume recombination).

This means that the primary source distribution can be decomposed into a sum of independent sources: $Q = \sum_i Q_i$ and the solution is obtained by linear superposition of the solutions for each sub-source ("stratum") Q_i.

The stratified sampling technique is a well known statistical procedure, see any textbook on statistics or Monte-Carlo methods.

It sometimes can significantly affect the efficiency of a Monte-Carlo run. This can go both ways, depending upon the particular problem and the particular stratification used. Therefore, as with the non-analog options in EIRENE too, no general recommendation can be made. "Optimal stratification" is case dependent.

The stratification with "proportional allocation" provided as one of the default option in EIRENE guarantees, however, that a case with stratified sampling is always at least not worse than without (and in most real cases better in terms of code efficiency).

The effects of stratification, however, are usually more easy to assess (predict) than with general non-analog methods.

In order to make connection with the textbooks we need to note here only the following:

EIRENE provides estimates of linear functionals ("moments", "responses")

$$< g, \phi_Q > = < Q, \phi_g^* > \tag{1}$$

As described in [1], g is the detector function ("weighting function") ϕ_Q is the solution to the kinetic transfer equation (Boltzmann eq.) in which Q was used as (inhomogeneous) source term. ϕ_g^* is the solution of the corresponding adjoint equation, with g as source term.

The above equation means that EIRENE estimates can simply be regarded as Monte-Carlo estimates of multi-dimensional integrals, namely of the adjoint function ϕ_g^* integrated over phase space with Q defining a distribution in phase space.

It is then obvious that one can split that integral into a sum of integrals by arbitrarily decomposing the domain of integration, i.e., by decomposing the source Q into sub-sources ("strata") Q_i with $Q = \sum_i Q_i$.

This is the concept of stratified sampling, which, for Monte Carlo particle transfer procedures in particular actually turns out to be a "stratified source sampling".

Further details about this in our particular context are given in the description of input block 7 of the EIRENE online manual [1].

1.2 Work done under present contract FU07-CT-2007-00007

B2-EIRENE [7, 8] was originally developed under NET contracts in the late eighties and early nineties of the last century at FZJ, in close collaboration with ERM Brussels and UKAEA Culham.

In the frame of the present contract the ITER version of the B2-EIRENE plasma edge modelling code, which was derived and has evolved from that original B2-EIRENE code package (and is referred to as SOLPS4.2 now) has been upgraded in order to allow parallel calculations on multiprocessor systems. Only the EIRENE module, which provides volumetric sources and sinks for the plasma fluid components treated in B2, is parallelized.

The B2 module, a 2D finite volume solver for highly un-isotropic and strongly nonlinear flow problems remains serial.

This feature: parallel kinetic (often Monte Carlo) modules linked to serial macroscopic fluid solvers is also encountered in other applications in simulation sciences, such as micro macro (polymer) models [9], radiation transfer in astrophysics and many more. This is because a) Monte Carlo solvers allow easy parallelization (farming technique) and it is usually these microscopic sub-modules which limit convergence of the overall code systems, due to the inherent statistical noise involved in Monte Carlo terms.

In what follows only the EIRENE code part, including its interface module EIRCOP to link EIRENE to edge fluid codes is described, because predominantly this part has been modified in the present project.

The following steps have been achieved:

- MPI-based technology is used (MPI=Message Passing Interface).
- Full backward compatibility of the calculations made with various different numbers of processors on various computer systems with the previous serial cases has been proved.
- The code has been installed and tested on several computer systems (a “self-made” cluster of ITER Team at Cadarache, the JET Analysis Cluster at JET, Culham, and the IBM Cluster JUMP in FZJ).
- The new version of the code has been delivered to ITER, first results have been presented jointly with the ITER team at the PSI and EPS conferences 2008. The upgraded code package is referred to as SOLPS4.3 (the old, serial code was “SOLPS4.2”) and it is currently in use both at FZ Jülich and at ITER-Cadarache for design optimization studies.
- The technical changes carried out to achieve this parallel performance of the B2-EIRENE code are described below in this document.

2 General code revisions and upgrades

The EIRENE version maintained at ITER-Cadarache has a number of specific features. It is only occasionally (once per year) synchronized and updated with the EIRENE master maintained at FZJ Jülich, to the extend that the code developments by ITER allow such a version management without loosing the ITER team’s own features.

As a first stage of the upgrade under the present contract such a ”partial synchronization” was carried out again: the variable set of the ITER version of EIRENE was made equal to that of the current reference version maintained at FZJ under CVS and SUBVERSION systems as master. A number of new technical features of that **EIRENE master version 2007** (see www.eirene.de) have been transferred into the ITER code system simultaneously with the current upgrading for parallelization.

Some parts of the B2-EIRENE (SOLPS4.2) code have been cleaned up during the present upgrade, rudimentary features (diagnostics, balance correction, C-preprocessor) have been removed. For the time being, the new definition of the reaction rate data and new atomic-molecular data structures of EIRENE (e.g., for ADAS and hydrocarbon databases) are not yet supported by this ITER version.

The present version of EIRENE does not use any external libraries any more (except MPI).

2.1 Parallelization based on MPI

The main objective of the present task was to enable full parallelization of the kinetic (EIRENE) part of SOLPS4.xxx. The EIRENE stand alone code was already parallelized with MPI. In the present project this option was extended for use in B2-EIRENE. Since the Monte Carlo particles ("histories") in EIRENE are independent, they can be relatively easy distributed over many processors (farming technique). The performance gain depends on the quality of synchronization between processors and relative importance of the remaining serial part:

a) Overhead: pre- and post processing;

b) Communications: data transfer between processors.

These two parts are not yet parallelized in the present project. For further optimization of the overall code package B2-EIRENE those parts will also have to be parallelized (for example by domain decomposition), and iteration strategies with longer Monte Carlo loops (more test-particles per iteration) have to be developed. Such options are now fully available and will be explored in the future. In the next subsections technical aspects of MPI parallelization are described.

2.1.1 MPI: version and its installation

Current realization of Message Passing in EIRENE and its code interface module EIRCOP is based on MPI-2 standard. The related subroutines are located mainly in EIRENE directory `broadcast`.

One of the best known open source implementation of the MPI-2 standard is the library MPICH (MPI-CHameleon) developed by Argonne National Laboratory. This was chosen for installation at the ITER computer system. At the time when this project started (July 2007) the most recent stable version was 1.0.5p4 (downloaded from the developer's site). The library and supplemented process manager (MPD) have been successfully installed and tested both at FZJ and on the computers of ITER edge modelling group in Cadarache. The installation instruction can be found in Section 6.1.

Available MPI distributions allow, in principle, parallelized runs on any set of machines connected in a network (e.g. Ethernet). No special hardware is required, but fast (Gbit/s) network connection is very desirable.

2.1.2 Parallel (test flight generation) part, and stratified source sampling

The parallel Monte Carlo sampling procedure has remained quite similar to the one already in plasma for stand-alone EIRENE runs and for parallelized EMC3-EIRENE runs (see section 1.1).

The various strata (see 1.1.1) can be distributed between processors in a more flexible way than before: e.g. one can sample all strata on each processor or allocate several processors only to one stratum or a subset of strata, etc. The corresponding controlling variables (like strata-processor correspondence table `PROCFORSTRA`) are defined in module CPES: either automatically, see Section 7), or read from the input file (Block `INFORMATION_FOR_MPI`).

The input, geometry and atomic data is broadcasted from the root to all processors. The results are gathered (reduced) separately for each stratum. This allows to apply exactly the same way of the particle balance control and re-scaling as in serial runs. See for more details of the implementation in Section 5.2. Performance diagnostic data such as the sampled number of particles and consumed time on each processor is printed into an extra file `mpistat.dat`.

2.1.3 serial (root) process: B2, pre-and post processing, and interface

B2, and EIRENE pre-and post processing:

B2 and initializing routines of EIRENE (`INPUT` etc.) as well as its post-processing routines (`OUTPUT, MODUSR etc.`) are executed only on the (serial) root process. No changes as compared to the earlier stand alone parallel version of EIRENE have been implemented.

Interface: B2-EIRENE

There are, in principle, two versions of the EIRENE interface module INFCOP to run with the B2 plasma code for transferring data from B2 (SOLPS4.XX) to EIRENE and vice versa: for the standard grid option (EIRENE directory `couple_B2`) and for the triangular grid option (EIRENE directory `couple_Tria`). This latter version (EIRENE geometry level: LEVGEO=4) is the most flexible and also most frequently used option in SOLPS4.XX now, therefore, only this interface has been upgraded and tested with MPI. The historically older “Standard grid” option (LEVGEO=3) in which EIRENE directly uses the B2-grid without any further processing, has not been used for ITER modelling since 2004.

Besides the interface, two extra calls have to be added to B2: `B2_MPI_INTERFACE` at the beginning of the program and `FINALIZE_EIRSRT` by the end of the main loop. See also Section 6.3 for more details.

Some more technical notes:

It is quite likely that the user needs permission to modify the file `/etc/hosts` to be able to change the systems hostname (if it was `localhost`). The need for this can be queried by command `mpdcheck -l`. The Process Manager distributed with MPICH is called MPD. An MPI-ring of machines which can be used for parallel runs can be created by command `mpdboot` using prescribed list of hosts. This command requires SSH to work without password. If this is not possible, then one can create the ring manually, by calling `mpd` on each machine to be added. A multiprocessor run is started through script `mpiexec`. Since this script has problems to read long files from standard input, EIRENE expects input file in `fort.1` when running on more than one processor. Section 6.2 gives some examples of using MPD and running B2-EIRENE on it. Without `mpiexec` the code compiled with MPI library can be run in exactly the same way as the serial version.

3 Backward compatibility tests

3.1 Code verification, backward compatibility

To verify the program changes of this upgrade, the results of the old ITER version (referred below as "old version") and the present MPI-version (running either on one or on several processors) have been benchmarked. The compared numerical data of the different versions have been the EIRENE sources passed to B2 (particle sources SNI, momentum sources SMO, energy sources SEI and SEE) and data stored on file fort.44. All data have been compared as text files with 8 decimal digits.

The chosen test cases were ITER 1055 and 1080. They are characterized by: divertor structure F12, full carbon wall, 9 fluids in plasma (D, He, C), full package for molecular kinetics (MAR, elastic collisions), neutral-neutral collisions.

The new version was run on one, two and four processors. In the latter case all kinds of communications were tested (including collecting data for one stratum from several processors, CALSTR).

Sequential and multiprocessor runs had the same sequences of random numbers, therefore it was possible to match single time-step B2-EIRENE runs exactly, hence proving the **full backward compatibility**.

For two and more time steps the exact match (up to 8 digits) could not be achieved: the solutions on different numbers of processors were different due to numerical round-off errors. This did not change even if the accuracy of floating point operations prescribed by IEEE standard was enforced: options `-O0 -Kieee -pc 64 -Mnobuiltin -i4 -r8` for Portland compiler and `-O0 -fp-model strict -pc64 -fp -mp -fltconsistency -i4 -r8` for INTEL compiler. Therefore, for the coupled B2-EIRENE runs the time-traces of some key parameters have been compared: control variables printed by B2 on each time step (total sources and fluxes, balances, temperature and density in peculiar points). It was shown that those time traces do not contain any anomal features (such as "discontinuities") when the code is restarted (continued) with the different number of processors. By continuing the steady time traces obtained with the old serial version after convergence both short ($\sim$10 time-steps) as well as long (5000 time steps) runs have been compared in this way on 1, 2, 4 and 8 processors, **again proving full backward compatibility of the new version also in iterative mode with B2.**

Initially the long run tests revealed a serious memory bug in library MPICH (see below in Section : Bugs and Bug-fixes, **??**), which has been fixed meanwhile.

More details on the performed tests can be found below in sub-section 3.2. At present the standard case chosen for tests of EIRENE stand-alone is ITER 1055p6_be: similar to ITER 1055 but with Be wall, Be ions in plasma and photon transport.

It has been also found useful to carry technical test runs with all possible compiler checks switched on (control of array bounds etc.): for Portland compiler `-C -Mbound - -Ktrap=fp -Mchkstk`, for INTEL compiler `i4 -r8 -debug all -traceback -check all -ftrapuv -debug-parameters all -g -inline-debug-info -fpstkchk`.

3.2 Description of the code verification tests

To prove the correctness of the code upgrade the results of the old version of B2-EIRENE (so called "photon version" from 2005-2006, or: SOLPS4.2) have been compared with the new version, both run on a single and on several processors. On the debugging stage the modelling case ITER 1055 was taken for the tests. It is a case with divertor structure F12, full carbon wall, 9 plasma species (D, He, C), complete package for molecular kinetics and neutral-neutral collisions (actual standard model for ITER).

3.2.1 EIRENE stand alone

For stand-alone EIRENE runs the arrays of sources passed to B2 and file `fort.44` were compared. The sources (sum over all strata) are printed by subroutine `PRINT_EIRBRA` called after `INF3COP` (from `PRTUSR`): `sniX.eir` is the particle source, `smoX.eir` is the momentum source, `sei.eir` and `see.eir` are the energy sources for electrons and ions. Here `X` is the index of species. File `fort.44` printed from `WNEUTRALS` contains neutral data to be passed to B2PLOT (neutral density, temperature, fluxes etc.). Files `*.eir` and `fort.44` are ASCII files and keep 8 decimal digits. To compare different runs two drivers have been written: `CFL` compares all numerical data in two text files, `CMP44` compares two files `fort.44`. They print maximum absolute and relative difference, difference of maximums and sums, number of non-equal elements, `CMP44` does it for each array.

The first test case can be found in `ipp247:/home/kotov/cases/iter1055.mpi.test` (see file `dirinfo`). The tests were made with "virtual parallelization": emulating several processors on one physical machine. The total number of particles: 5500. All data files produced by the "photon version" and the upgraded version on one processor are exactly the same. Serial runs have been compared with runs on 2 and 4 processes (subsequently, without and with CALSTR): the results are exactly the same.

The results of 2 and 1 processor runs are exactly the same, for 4 processor run a small difference has been found for variable `srcml` from `fort.44`: the absolute difference is 5.5E-014. This can be attributed to round off error during communications (see **??**).

3.2.2 B2-EIRENE, one single time-step

The single time-step runs of B2-EIRENE can be compared in the same (exact way) as stand alone EIRENE runs: they differ from EIRENE stand-alone cases only because some input for EIRENE is still provided via B2 in SOLPS4.xxx. (This risky coding has been completely eliminated in the current FZJ master of EIRENE and B2-EIRENE). Again perfect matching of results (down to machine precision) from the original and the upgraded code has been achieved.

3.2.3 B2-EIRENE, multiple time-steps

Two time step runs on 1 and 2 processors could be compared in exact way if the code was compiled with option `-Kieee -pc 64` (floating point arithmetic strictly in accordance with IEEE standard). In case of 4 CPU run the exact comparison did not work anymore: for some sources calculated by EIRENE (those with high statistical noise: momentum source and ion energy source) the difference reaches locally 30 %. Similar features can be seen for 2 CPU tests with 3 time-steps: the difference can reach 60 % (momentum sources for He). An investigation showed, that neutral-neutral collisions render the coupled system very sensitive to round-off errors during communications. An investigation showed, that neutral-neutral collisions makes the system very sensitive: due to round-off errors during communications the neutral background for the next time step becomes slightly different (the difference in 15th decimal digit) on the different number of processors. And subsequently correlation between the two test cases is lost.

The conclusion was that it is reasonable to apply the exact comparison (all digits) only for single time-step runs and to compare (visually or by post processing tools) B2 time-traces of important physical quantities for the longer runs. The "time-traces" chosen here are the neutral particle source terms, total plasma fluxes to targets and some plasma parameters (upstream density and temperature, maximum density and temperature in both divertors), printed on each time step to directory `tracing`. These tests have been fully successful and hence have **verified the upgraded code** (see next subsection).

3.2.4 Installation of upgrades at ITER Cadarache

After having passed all these tests in FZJ the code and the MPI software were installed on the computer system of ITER Team (server `eskimo.iter.org`). The test case was ITER 1080 (same geometry and plasma content as 1055).

The first technical problem was that the results obtained with INTEL Compiler (used by ITER Team) differ slightly from those obtained with Portland Group Compiler (used in IPP FZJ), see ??. Applying compiler options `-O0 -Kieee -pc 64 -Mnobuiltin -i4 -r8` for Portland and `-O0 -fp-model strict -pc64 -fp -mp -fltconsistency -i4 -r8` for INTEL (enforce IEEE-prescribed accuracy for floating point operations) did not help to reduce the difference. An investigation showed that the discrepancy can be reduced (down to maximum relative difference 1e-4) only if the subroutine GAUMEH (Gauss-Mehler integration for scattering angle integral in elastic collisions) is commented out, proving that the numerical round off errors in Monte Carlo particle trajectories are mainly originating from this routine. Further investigation was not carried out, since the resulting difference is of no practical importance anyway (smaller than the level of statistical noise).

The test runs for ITER 1080 are saved in directory
`iter.eskimo.org:/home/kotovv/cases/iter1055.mpi.test`.

The results of the stand-alone EIRENE runs of the old ITER version of EIRENE and the actual version were also different. The reason was a minor inconsistency in the actual Jülich version of EIRENE. Sputtering of carbon was not taken into account in recalculating the scaling factors for global particle balance for each atomic species. Globally this source is usually small compared to recycling sources, therefore, this bug had little or no influence on the results. After this was fixed, the results of the that-time ITER version of EIRENE and of the current Jülich version of EIRENE could again be matched exactly.

In those tests two physical machines were used, in 4 processor runs the subroutine CALSTR was called. For 4 processor runs a small difference was found for variables `srcml` and `edissml` from `fort.44` (maximum relative difference <1e-2) and flux `wldna` (zero and non-zero value). With 2 and more time-steps the solutions become different and can not be matched exactly. However, when compiled with options `-fp-model strict -pc64` the maximum relative difference (for the neutral particle sources transferred to B2) for 2 time-steps on 2 processors did not exceed 1e-6.

Comparison of the time traces of the short coupled run was done in the following way: 20 time steps (dt=1e-6 sec) of the reference ITER version, then 10 time steps of the new version on one processor, then 10 steps on 2 processors, then 10 steps on 4 processors and then 10 steps on 10 processors (all strata on each processor). No anomalies larger than statistical noise could be seen on the time traces. After that, a long run test was performed: the run ITER 1080 was continued for 5000 time steps (dt=1e-6 sec) on 4 processors, then for 5000 steps on 4 processors and then 5000 steps of the old version. Also the long run test was successfully completed.

Apart from runs in FZJ and Cadarache the code was technically tested on the IBM cluster JUMP in FZJ and on JET Analysis Cluster (uses OpenMPI) on up to 32 physical processors. On JAC the time tracings for the case iter.D10 on 1 and 4 processors were compared in addition: this is a reduced ITER case with deuterium only. Technical stability was proved by compiling the code with extra checks (array bounds check etc.). Corresponding options for Portland compiler: `-C -Mbound -Ktrap=fp`, for INTEL compiler: `i4 -r8 -debug all -traceback -check all -ftrapuv -debug-parameters all -g -inline-debug-info -fpstkchk`.

The "maximum" case which is suggested for stand alone EIRENE tests is the moment reference case ITER 1005p6_be. It is a case with divertor structure F12, beryllium wall (and Be ions) and photon transport. In addition to neutral sources and `fort.44` the particle sources and sinks for Ly_α and Ly_β photons are compared. The results of the actual version of EIRENE on 1 and 2 processors match exactly the "photon" version for this case.

4 Parallel code performance tests

Modelling case ITER 1055 was chosen for the performance tests. The number of particles was fixed: 2000 for each target, 1000 for volume recombination, 2000 for the gas puff, 500 for each PFR boundary (entrance into dome region) and 10000 for main chamber wall recycling (18000 in total). Those numbers are larger than the numbers usually used for this case: it was done in order to emulate a case with slightly longer EIRENE runs, as it is anticipated for routine applications with parallelized EIRENE in SOLPS4.3.

"Standard" ITER grid size: B2 grid 28×76, 5589 cells in triangular grid. The measurements were performed on the second time-step of two-steps B2-EIRENE runs (starting each time from the same solution). The time is measured by Fortran subroutine `cpu_time`.

4.1 Profiling of the first parallelized version

The tests were performed in Cadarache (machines edge29,30), in Jülich on Jülich Multi-Processor - JUMP (an IBM cluster of symmetric nodes) and at Culham on JET Analysis Cluster JAC (a Beowulf cluster consisting of off-shell workstations, tests were performed at the end of 2007). The results are shown in Table 1. In all cases, except "JAC (opt)." all strata were running on all processors: the most inefficient option in terms of communications. In Table 1 "B2-EIRENE" is the total time of the time-step, "Sampling" is the time spent by EIRENE for the particle sampling, "B2" is the time consumed by B2, "Communications" is the time spent for data transfer between processors and "Overhead" is the time consumed in EIRENE not for sampling. "Sampling", "Communications" and "Overhead" are shown for root process. "Overhead" is mainly related to integration of tallies and correction of particle balance for each stratum (rescaling of tallies).

From Table 1 one can see, first of all, that influence of multiprocessing on performance strongly depends on the hardware. The runs in Cadarache were twice as fast as those in Jülich partly because of faster processors and partly because of better compiler optimization (with INTEL compiler). It is very important to have fast network: in this case (ITER machines and JUMP) one can reduce the total run time by a factor 3 on 4 processors, but on 8 processors the performance gain saturates. In this case the saturation comes not due to communications but due to EIRENE's overhead. On the cluster JAC which has a slow network the gain hardly reaches a factor of 2 on 8 processors. However, applying more clever distribution of strata between processors can improve the situation. The results shown in the rows "JAC (opt)" were obtained with the following distributions (same fixed number of particles):

```
NPRS=2, NSTRA=8
TTTFFFFF
FFFTTTTT

NPRS= 4, NSTRA= 8
TFFFFFFF
FTFFFFFF
```

Table 1: Run-time for the reference case ITER 1055 (one time-step)

Machine	B2-EIRENE	Sampling	B2	Communications	Overhead
ipp754	106.2	95.0	3.7	0.0	7.4
edge29	54.6	47.4	2.3	0.0	4.8
edge29,30	31.2	23.4	2.3	0.4	4.9
JAC 1 CPU	92.0	81.6	1.8	0.0	8.5
... 2 CPU	62.5	35.4	2.6	14.3	5.5
... 4 CPU	63.7	21.1	2.8	24.3	8.2
... 8 CPU	41.9	8.8	2.3	18.6	5.6
JAC (opt) 2 CPU	48.8	36.7	2.4	2.0	2.9
... 4 CPU	41.1	21.3	2.7	7.8	3.1
... 8 CPU	44.8	18.6	4.3	9.7	3.7
JUMP 1 CPU	156.2	133.1	10.4	0.0	12.7
... 2 CPU	84.7	61.2	10.6	0.5	12.3
... 4 CPU	52.8	30.1	10.1	0.8	10.8
... 8 CPU	41.1	15.0	10.5	1.1	14.3

```
FFTTTTFF
FFFFFFTF

NPRS= 8, NSTRA= 8
TFFFFFFF
TFFFFFFF
FTFFFFFF
FTFFFFFF
FFTTFFFF
FFFFTTFF
FFFFFFTF
FFFFFFTF
FFFFFFTF
```

In this case a factor of 2 improvement can be achieved already on 2-4 CPU's, but this almost immediately saturates at this point when moving to more processors. Note that the JAC results are scattered because of inhomogeneity of the system: in different runs the task manager could send the task to different processors. In the "optimized" runs in most of the cases one strata was calculated on one processor: this leads to significant reduction of required communications. Besides that, the overhead reduced as well because integrating and rescaling was performed for different strata in parallel. The overhead increases when the number of strata and the grid size is increased. Therefore, applying optimized distribution of strata between processors can be useful even for systems with fast communications.

4.2 Parallel performance of the current version

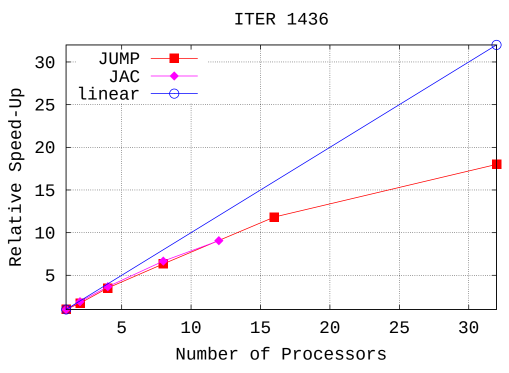

Figure 1: Relative speedup factor vs. number of processors, measured by time spent on one time step of B2-EIRENE (case ITER 1436, very high divertor density) on different systems.

Based on the results of the study described in the previous section, the code was further optimized: the overhead was reduced, run-time synchronization of strata was implemented (see Section 5.3), automatic distribution of strata between processors was added, Section 7. Performance tests of this partly optimized code have been performed on the same three systems as in the previous section for the modelling case ITER 1436. This model has 9 ion species and is characterized by extremely dense divertor (peak heat power only 2 MW/m^2). It normally takes 10. . . 30 min for one time-step depending on the machine. EIRENE run-time was fixed at 1800 sec on JAC and at 1000 sec on JUMP. In addition, the code was given the following minimum number of particles: 400 for strata 1 and 2 (targets), 100 for strata 3 and 4 (PFR), 5000 and 2000 for 5 and 5 (SOL boundary and gas puff, fastest strata) and 200 for 7 (volume recombination, slowest strata). The specified run-time ensured that ≈40000 particles (±30 %) were sampled in each run.

The results are shown in Figure 1. The speed up is linear only up until 4 processors, the maximum gain is a factor of 18 on 32 processors. In absolute numbers that means the following. On JUMP one time-step takes 60 sec on 32 processors instead of 1000 sec on one processor. B2 takes 10 sec, overhead of EIRENE: 2 sec, communications: 4 sec. On JAC: 200 sec on 12 processors, instead of 1800 sec on one processor, 5 sec are taken by B2, 2 sec: overhead in EIRENE, 80 sec: communications (JAC has a relatively slow network).

The saturation of speed-up takes place not only due to serial part and communications. There is also an issue with synchronization. The cpu-time, spent for each single trajectory can be very different. Even if time synchronization is applied, it can still happen that a very long trajectory occurs on one processor, and other processors are then waiting (idle) until it finishes. In case of high density divertors very long trajectories may appear because of Monte Carlo splitting. For the reference test case the lost time due to this effect can reach 10 % on 8 processors and 30 % on 32 processors. This problem will have to be resolved in the future either by avoiding Monte Carlo splitting techniques, or by applying dynamic redistribution of particles between processors.

In general one can say that for regular ITER cases or for reduced cases on finer grids the typical robust speed-up is at least a factor of 3 on 4 processors and at least a factor of 6 on 8 processors (typical processor numbers currently used at ITER Cadarache). Parallelization of EIRENE overhead (re-scaling, MODUSR) and improved synchronization for the cases with long histories can further improve the scalability.

5 Technical description I: Modification of the code

5.1 General upgrade and cleaning-up

The ITER version of the code was upgraded from the CVS repository master version maintained at FZJ. The main point of this upgrade was to synchronize the variable set of the FZJ repository version with the ITER version as well as to transfer all MP-related code. Consequently, the following entire modules were replaced with the FZJ master: `PARMMOD, CESTIM, COMUSR, CUPD, COUTAU, CSPEI, CTRCEI, CLOGAU, CADGEO, CGEOM, CGRID, CLGIN, CTETRA, CESTIM, CSTEP` and subroutine `SETPRM` (assigning indexing of the tallies). Module `COMPRT` was modified accordingly. In the repository (master) version the definition of reaction rate data was completely re-written (`REACDAT` instead of `CREAC`) to adopt it to further atomic database formats and the FZJ-hydrocarbon databases. This modification was not yet taken over to the ITER version for the time being. Therefore, module `COMXS` was left unchanged.

The upgraded variable set has some new tallies, like momentum flux (`VGENA, VGENM, VGENI, VGENPH`), primary particle (`PPAT, PPML, PPPHT`) and energy (`EPAT, EPML, EPIO, EPPHT`) sources. Not all of them are updated in the current version, but all their infrastructure (allocation-deallocation etc.) works. In the new version each tally has a logical switch, e.g. `LPDENA` for `PDENA` etc. which allows to switch it off if necessary (consequently, not to spend memory for this variable etc.). This storage saving option is not completely cleaned up in the ITER version of EIRENE: it still can be that extra `IF` statements are missing in some places.

To allow one more possibility of memory optimization in the new version variable `NPLS` (number of plasma species) was complemented by two over variables: `NPLSTI` (number of different background ion temperatures) and `NPLSV` (number of different background ion velocities). If in the input `INDPRO(2)`$\leq$`0` is specified (normal case for ITER applications), then `NPLSTI=NPLS`, otherwise (`INDPRO(2)<0`) `NPLSTI=1`. The same is with `NPLSV` and flag `INDPRO(4)`. Note, that this option has until now been tested in the ITER version only for the default case `NPLS=NPLSTI=NPLSV`. Variables `NVOLTL` and `NSRFTL` (number of volume and surface tallies) are defined now in `SETPRM` instead of `PARMMOD`.

The subroutine `MCARLO` has been cleaned up. Some parts of its code were condensed into subroutines

`scal_volav_tallies.f`	normalizing the volume tallies with the total fluxes and cell's volumes;
`integrate_tallies.f`	calculating integrals over the volume tallies;
`scal_surf_tallies.f`	scaling total surface fluxes with the sources strength;
`sum_average.f`	sum of the total fluxes over all strata;
`scale_tallies.f`	calculating the scaling factors (particle balance, subroutine `GETSCL`) and rescaling the all tallies;
`sumostra.f`	updates the sum of integrals over all strata;
`stat_sumostra.f`	sum over all strata for standard deviation;
`scale_deviation.f`	rescaling the standard deviation;

Treatment of recombination sources in IF3COP was slightly modified. The particle source due to recombination is added to PAPL, the energy sources for electrons and ions: to EAEL and EAPL respectively. Only the array CPPVR (momentum sources due to recombination) is left locally in INFCOP.

Some obsolete or rudimentary parts have been removed from the code:

- C preprocessor (subsequently, all extension changed to f);
- EIRENE does not use any external libraries (except MPI) any more;
- subroutine `eirpbls` (extra correction of balances);
- subroutine `srfprvsl` (some graphics diagnostic);
- special subroutines for output (`textoutput, out_transrates`);
- specification of chemical sputtering data in `INFCOP`;
- input for all removed features (Blocks 9..11) and corresponding variables in `EIRDIAG`.

Subroutine `FINALIZE_EIRENE(NLLAST)` which cleans all the EIRENE allocated memory is an entry of subroutine `EIRENE` now.

New user features are described in Appendix.

5.2 MPI-related code

The MPI library is not linked to the code directly: it has to be compiled through the script `mpif90` (for Fortran 90) instead of regular compiler's name.

The code related to the Message Passing is mainly concentrated in directory `BROADCAST` and module CPES. The code is taken from the repository version of EIRENE and then modified (in some cases significantly). The corresponding parts of the code are described below:

`broadcast.f`	broadcasts all variables from root process;
`allocate_modules.f`	allocates modules on non-root processes;
`broadsput.f`	broadcasts some variables from `SPUTER`;
`broadref.f`	broadcasts some variables from `REFLEC`;
`broad_eirsrt.f`	broadcasts variables passed from B2 in `EIRSRT`;
`broad_pedist.f`	broadcasts MPI-related variables from `PEDISTS`;
`broad_mpistat.f`	broadcasts run-time distribution on each processor;
`calstr.f`	collects data for one strata;
`collect_coutau.f`	collects data for sum over strata;
`collect_infcop.f`	collects sources for B2 (SNI, SMO, SEE, SEI);
`collect_census.f`	collects census array;
`b2_mpi_interface.f`	an interface to be called at the beginning of B2;

Variables related to MPI are defined in module `CPES.f`:

`MY_PE`	index of actual process;
`NPRS`	number of processes;
`PROCFORSTRA(NSTRA,0:NPRS-1)`	logical table, marks for each strata processors on which it is run;
`NPESTR(0:NSTRA)`	for each strata: number of processes on which it is calculated;
`NPESTA(0:NSTRA)`	index of processor on which the sum for this stratum to be calculated;
`NSTRPE(0:NRPES-1)`	number of strata for each process;

(`NSTRA`) is the number of strata).

A flow chart of the MPI implementation in EIRENE is shown in Figure 2.

Table `PROCFORSTRA` allows a flexible way of distributing the strata between processors. Only if `PROCFORSTRA(I,J)=.TRUE.` stratum `I` will be launched on processor `J`. This array is set up in subroutine `PEDIST` (module CPES). It is either read from the input file (subroutine `READ_PROCFORSTRA`), see section 6.2, or automatic distribution, see Section 7, is used. After obtaining `PROCFORSTRA` subroutine `PEDIST` corrects the cumulative time allocated to sample each strata (array `XTIM`) to match the total allocated time on each processor.

In EIRENE all numerical input and definition of the atomic data (subroutines `INPUT, SETCON, SETPRM`) is performed only on the root process (`MY_PE==0`). After that it is broadcasted (by subroutine `BROADCAST`, `MPI_BCAST`) to other processors (the subroutine `ALLOCATE_MODULES` allocates memory on the non-root processors).

In subroutine `MCARLO` the sampling for the stratum is launched if the corresponding `PROCFORSTRA(ISTRA,MY_PE).EQ..TRUE.` (here `ISTRA` is the index of a current stratum). If one stratum is sampled on several processors, then the results are collected on the process `NPESTA(ISTRA)` by subroutine `CALSTR`. This subroutine collects (reduces, `MPI_REDUCE`) data over intra-communicators created in advance in entry `CALSTR_INIT` (uses `MPI_COMM_SPLIT`). After that, for the processors on which the stratum data were collected (`MY_PE.EQ.NPESTA(ISTRA)`) rescaling is performed and sum over strata is updated:

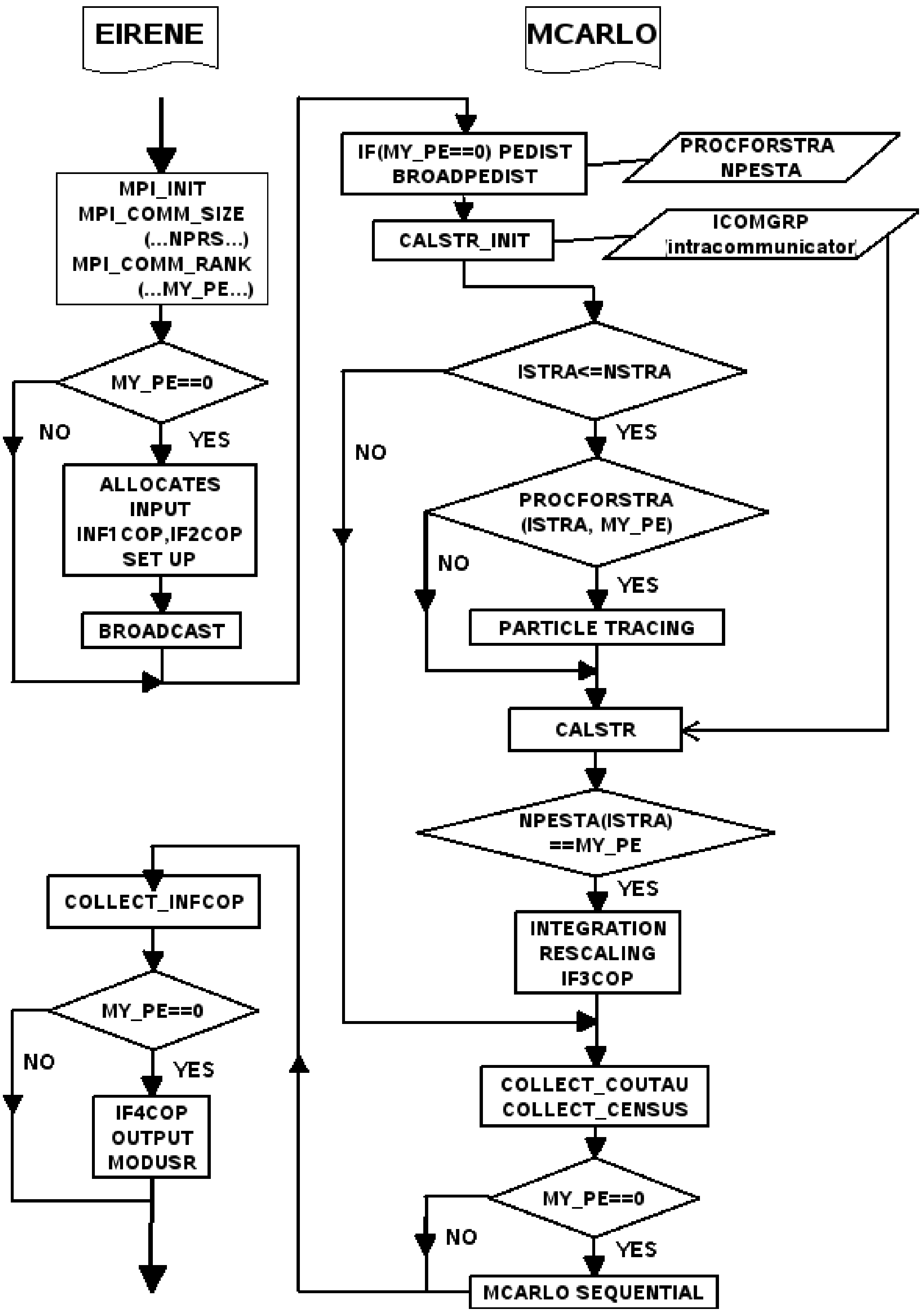

Figure 2: Flow-chart of the subroutines EIRENE and MCARLO running with MPI interfaces (NPRS>1).

subroutines STATS2, SCAL_VOLAV_TALLIES, INTEGRATE_TALLIES, SCAL_SURF_TALLIES, SUM_AVERAGE, SCALE_TALLIES, ALGTAL, SCALE_DEVIATION, IF3COP, SUMOSTRA. After finishing the strata loop (1000 CONTINUE) data for sum over strata are collected (subroutine COLLECT_COUTAU) as well as census array (COLLECT_CENSUS). COLLECT_COUTAU gathers data over intra-communicator containing only processes listed in NPESTA(ISTRA).

This operation is followed by the serial part, which is performed only on MY_PE==0: moving sum over strata back to ESTIMS, ESTIMV etc., writing files fort.10 (WRSTRT) and fort.11, printing routines (OUTPLA, OUTEIR, DIAGNO etc.) and MODUSR.

Arrays of source terms for B2 (SNI, SMO, SEE, SEI) are collected on the root processor in subroutine COLLECT_INFCOP using point-to-point communications (MPI_SEND and MPI_RECV). Subroutine WNEUTRALS was modified to read the volume parameters of the neutrals (dab2, tab2, etc.) only in the entry WNEUSAVE: called in IF4COP in the serial part. Flux variables (wldnek, wldnep, etc) are calculated individually for each stratum and collected on the root by COLLECT_INFCOP.

The applied technique of parallelization allows exactly the same way of rescaling (to enforce perfect particle balance despite statistical errors) as in the serial EIRENE runs.

EIRENE requires a library which supports standard MPI-2. It uses two features of this standard: statement MPI_IN_PLACE in subroutine MPI_REDUCE and subroutine MPI_FINALIZED. MPI_IN_PLACE is needed to avoid aliasing: using the same variable as input and output. It is prohibited by Fortran standards, although some compilers allow this (e.g. IBM compiler on JUMP).

Some other changes, already in place on the stand alone FZJ EIRENE master, had to be implemented into the ITER version in order to adopt the code for parallel runs.

- The index of output stream was changed everywhere to IUNOUT (using driver chgun). IUNOUT=6 on the root (standard output) and 7 on all other processors.
- The index of input stream STANDARD_INPUT is now a variable which is equal to 5 (standard input) on one processor and 1 for NPRS>1.
- Subroutine EXIT was replaced by EXIT_OWN which calls MPI_ABORT to stop all parallel processes.
- The initialization procedure scheme was somewhat changed. CALL ALLOC_CESTIM(1) was moved to stay in front of ALLOC_COMUSR(1). CALL ALLOC_CSPEI was moved from INPUT to EIRENE. ALLOC_CESTIM(2) is now called from SETPRM instead of EIRENE.

In subroutine EIRSRT (the interface program which is called from B2) variables transferred through dummy arguments are broadcasted by BROAD_EIRSRT. To pass around B2 on all processors except root, subroutine B2_MPI_INTERFACE has to be called at the beginning of a B2 run. This subroutine makes a call to EIRSRT for MY_PE.GT.0. Subsequently, this process is waiting then for the root to call BROAD_EIRSRT which synchronizes them. Subroutine BROAD_FLUX is used to broadcast the strength of B2 sources (EIRSRT's argument FLUXES).

New performance diagnostics which measures CPU time spent in different parts of the code has been added. The corresponding code is located in module CPES. Detector functions UPDATE_MPISTATDATA, TIME_OF_EIRENE_START and TIME_OF_EIRENE_END. Subroutine PRINT_MPISTATDATA prints a text file mpistat.dat which shows on each processor the time spent for sampling of each strata, number of particles, time for spent for overhead subroutines (INPUT, rescaling, MODUSR etc.) and for communications.

The procedure of automatic distribution of strata between processors is described in Section 7.

5.3 Further modifications and bug fixes

General modifications and bug fixes:

1. Module ADSP added: AK's option for calculating the incident spectra. All related code (including broadcast) is hidden in the module, access through subroutines with prefix ADSP_.

2. A bug with NLMPGS has been fixed (see below).

3. Switch NLTRIMESH to use EIRENE only for grid generation.

4. Bug with input in block 5 (FIND_PARAM) has been fixed.

5. WRPLAM can now read fort.13 in both old (2004) and actual formats, distinguishing between them automatically.

6. Subroutine WNEUTRAL_FLUXES which calculates EIRENE fluxes mapped to the edge of B2 grid has been added (called in WNEUTRAL).

The fix of the bug with NLMPGS has to be described in more detail. If spatially resolved surface tallies are applied, then the size of corresponding arrays (variable NLMPGS) depends of the number of triangles belonging to B2 surfaces. In turn, the code can learn this number only in IFOCOP. In the old version of the code those variables were not corrected properly after this place. Such a correction has been added, all related arrays can be allocated only after this point. Module CESTIM was already allocated at the right place. In modules CGEOM and COMUSR this had to be changed.

In CGEOM array AREAG (surfaces of wall elements, now perhaps obsolete) made a separate array (not a part of RCGM1). ALLOC_CGEOM was splitted into two parts, ALLOC_CGEOM(2) is used to allocate AREAG. In the module COMUSR entry ALLOC_COMUSR(3) was added to allocate arrays SAREA (surfaces of wall elements) and FLXOUT (array of incident flux for Roth formula, not yet used). In subroutine EIRENE, CALL ALLOC_CGEOM replaced by CALL ALLOC_CGEOM(2).

Now the procedure work as follows. Number of triangle sides belonging to B2 surfaces is calculated in IFOCOP, variables NGITT and NLMPGS are re-initialized where, if previously estimated values (calculated in if0prm) were too small. After that, ALLOC_CGEOM(2) and

`CALL ALLOC_COMUSR(3)` are called in `INPUT`. Local variable `SAREA0` is introduced in `INPUT` to temporary replace `SAREA` when it is not yet initialized. Besides that, extra checks added to modules `CESTIM, CSDEI, CSTEP, PARMMOD`.

Optimization of memory usage:

1. Unused tallies switched off in `SETPRT` (`EAPHT...,VXDENA,...,MAPL...`);
2. For tallies, not related to BGK, the size is now defined by variable `NPLS_TAL`, not `NPLS`. `NPLS_TAL` counts only real ion species, excluding BGK species;

For the test cases ITER `1055` (9 ions + 9 BGK species) and ITER `1055p6_be` (13 ions species + 9 BGK species + 5 photons) this optimization allowed to reduce EIRENE overhead (excluding `MODUSR`) by a factor of 1.5.

To add Message Passing operations to the modules, wrappers around MPI library subroutines are used (file `mpi_wrappers.f`). This allow replacing them with dummies when compiling without MPI. Subroutines `PEDIST`, `BROADCAST_MPISTAT`, `BROADCAST_PEDIST` moved to module `CPES`. Strata running in parallel on more than one processor are now explicitly synchronized in time when running with prescribed minimum number of histories. On each processor the startum runs for a fixed amount of time `XTIM(ISTRA)`. When this time is over, the stratum stops and calls subroutine `CPES_SYNCHRONIZE` (in `MCARLO`). This subroutine adds up the number of particles, which have been sampled on all processors on which this stratum is running. It checks, is this number is larger than the prescribed minimum. If not, then the estimated time required to sample the left trajectories is calculated, and the run is continued. `CPES_SYNCHRONIZE` is also called if the maximin specified number of particles is achieved, to avoid communication dead-lock.

6 Technical description II: Installation and usage

6.1 Installing MPICH

A well known free implementation of the MPI-2 standard library MPICH (MPI-CHameleon), Argonne National Laboratory, was chosen for installation to the ITER system. The most recent (at the time when the project started) version was 1.0.5p4. The instruction below explains step-by-step how to install it.

1. To download MPICH2:
   `http://www-unix.mcs.anl.gov/mpi/mpich/downloads/mpich2-1.0.5p4.tar.gz`
2. To unpack it: `tar -zxf mpich2-1.0.5p4.tar.gz`
3. To create directories for building and installing, e.g.:

   ```
   mkdir mpich2-install
   mkdir mpich2-build
   ```

4. to set up the name of fortran compiler (e.g. Intel Fortran in bash):

   ```
   export F77=ifort
   export F90=ifort
   ```

5. To go to build directory and to run configuration script, e.g.:

   ```
   cd mpich2-build
   ../mpich2-1.0.5p4/configure --prefix=/home/kotov/MPI/mpich2-install ...
   ... > configure.log
   ```

   Here `/home/kotov/MPI/mpich2-install` is the full path to installation directory

6. To run make (compiling MPI requires GCC):

   ```
   make > make.log
   make install > make.install.log
   ```

7. To add MPI `bin` directory to the path, e.g.:
   `export PATH=/home/kotov/MPI/mpich2-install/bin:\$PATH`
   (this line has to be added to `.profile`)

8. To create "secretword" file for MPD (for some security reasons) one has to go to the home directory and to type:

   ```
   touch .mpd.conf
   chmod 600 .mpd.conf
   ```

   The file `.mpd.conf` has to contain the following line:
   `secretword=<some combination of letters and numbers>`

After this procedure the MPI library is installed. The default process management system delivered with MPICH is MPD. One can check if MPI library and MPD have been installed correctly by running the command `mpich2version`. It will print something like this:

```
Version:           1.0.5
Device:            ch3:sock
Configure Options: '--prefix=/home/kotovv/MPI/mpich2-install' 'F77=ifort' ...
... 'F90FLAGS=' 'F90=ifort'
CC:  gcc
CXX: c++
F77: ifort
F90: ifort
```

Note that it is necessary to install MPI only once in home, but not on each machine to be used in the MPI ring.

AN IMPORTANT NOTE. In the used version of the MPICH library a bug was discovered (see sub-section ??) which caused a "memory leak" in the code (the code gradually increased its size in memory). Memory profiler VALGRIND detected a memory leak in the MPICH library. The bug (a missing call of "free" for a pointer) was found and fixed. The bug has been confirmed by the developer. To fix it one has to add one line to the source file
`<MPI directory>/src/mpi/coll/helper_fns.c before compiling`.

In function `MPIC_Send` the construction:

```
    if (request_ptr) {
        mpi_errno = MPIC_Wait(request_ptr);
    }
```

must read:

```
    if (request_ptr) {
        mpi_errno = MPIC_Wait(request_ptr);
    if (mpi_errno) { MPIU_ERR_POP(mpi_errno); }
            MPID_Request_release(request_ptr);
        }
```

6.2 Compiling and running the code with MPI

The recent EIRENE distribution adopted for parallel runs within SOLPS4.xxx is called `Eirene.mpi`. To compile it without MPI, one has to link the `*.f` files from directory `../Eirene.mpi/broadcast_dummy/` and include `mpif.h` from this directory. For compilation with MPI one instead has to use directory `../Eirene.mpi/broadcast/` and include file `<MPIdirectory>/mpich2-install/include/mpif.h`. Command `mpif90` has to be used instead of the name of the Fortran compiler. This script links all the necessary library files automatically. One should not use the regular compiler's name and also not link the libraries explicitly, because this does not work! Note that the executable compiled with MPI can be used on one CPU in exactly the same way as without MPI (as before). Note also that to run the code compiled with the INTEL compiler one may need to increase the stack size: BASH command `ulimit -s 262144`.

Note, that apart from MPI this distribution of EIRENE does not require any other external library. File `eirene_main.f` is required only for compiling the stand alone version of EIRENE (without B2). For coupling with B2 one can choose between two sets of subroutines, located in directories `couple_B2` and `couple_Tria`, respectively. They are related to different geometry options in EIRENE: the 2D regular polygonal standard grid (geometry level LEVGEO=3), in which the cells are quadrangles, and the more flexible triangular grid (LEVGEO=4), respectively. Presently within SOLPS4.xxx the triangular grid is the stable option and thus recommended for use, whereas the older standard polygonal grid has not been used and tested since 3 years. The modifications needed in B2 described in section 6.3.

For making multiprocessor runs using MPICH, one first has to start MPD on the host machine: `mpd &`. To find out whether parallel runs are possible, one has to use command `mpdcheck -l`. In case of computers in Jülich it prints the following message:

```
**********
Your unqualified hostname resolves to 127.0.0.1, which is
the IP address reserved for localhost. This likely means that
you have a line similar to this one in your /etc/hosts file:
127.0.0.1   \$uqhn
This should perhaps be changed to the following:
127.0.0.1   localhost.localdomain localhost
**********
```

According to the policy provided by the Jülich Supercomputing Center (JSC) users at FZJ are not allowed to change this parameter. Therefore, real parallel runs on regular LINUX cluster machines are not possible, since they can not tell the other machine their unique host address. The command `mpdallexit` stops MPD on the whole MPI-ring (see below).

On the ITER computer system in Cadarache this problem did not appear. Package MPICH has a tool `mpdboot` which automatically creates a MPI ring on the specified number of computers taken from the list of hosts. Using this tool requires `SSH` to work without password. If this is not possible (as it was the case during the project described here), then one has to create the MPI-ring manually. An example is shown below:

1. Starting MPD on the host (root):
   `mpd&`

2. Typing on the host:
   `mpdtrace -l`
   It will print something like this:
   `edge29.itereu.de_38804 (192.168.0.39)`
   Those are the host name, the port number and the IP-address.

3. Go to the next machine (to be added to the ring) and type:
   `mpd -h <host name> -p <port number> &`
   e.g. in the case in question:
   `mpd -h edge29.itereu.de -p 38804 &`

4. Go back to the host, and check the created ring:
   `mpdtrace -l`
   then you will see something like this:

```
              edge29.itereu.de_38804 (192.168.0.39)
              edge30.itereu.de_33756 (192.168.0.40)
```

... and so on.

After starting the MPI ring one can run the code on several machines in parallel. In EIRENE input file one can specify the distribution of strata between processors: array PROCFORSTRA, see section 5.2. This can be done after block 14, e.g. in this way:

```
*** INFORMATION_FOR_MPI
NPRS= 2, NSTRA= 8
TFTTTTFF
FTFFFFTF
NPRS =4, NSTRA = 8
TTTTTTTF
TTTTTTTF
TTTTTTTF
TTTTTTTF
```

The code looks for the key word INFORMATION_FOR_MPI everywhere through the input file. In each entry: NPRS is the number of processors and NSTRA is the number of strata (the time census is taken into account: that's why it is 8 instead of 7 for ITER cases). The code looks for the combination which corresponds to the current run. If it has been found, then the code reads logical table which follows immediately after NPRS, NSTRA. Each line corresponds to one processor. .TRUE. means that this strata has to be run on this processor, .FALSE.: not to run it. E.g. the entry for NPRS=2 says that strata 1 and 7 will be run on processor 0 and all the rest on processor 1. Entry for NPRS=4 says that every processor has to sample all strata. The code looks for the MPI data until it reaches the end of file or the key word END_OF_INFORMATION_FOR_MPI. If it can not find the appropriate combination of NPRS and NSTRA in the input file, then the default (all strata on each processor) is used. If a key word AUTOMATIC_DISTRIBUTION is put to this block, then automatic distribution, Section 7 is applied.

One important note. Lines in PROCFORSTRA which switch on the same strata have to represent the same pattern. E.g. in case of distribution:

```
FFFTTTTF
TTTTFFFF
```

processor 1 will start from the 4th strata and will be waiting then for processor 0 where this stratum is the last one. Only after that processor 1 will be able to finish its work while processor 0 will be waiting for it.

To start a parallel run one has to use the script mpiexec from MPICH package. An example of the corresponding command:

```
nohup mpiexec -l -n 2 ./linux.Intel_EM/b2eirmpi > b2.log&
```

Here -n is the flag, which specifies the number of processors (2 in this case),

`./linux.Intel_EM/b2eirmpi` is the name of executable. Option `-l` enforces printing the number of process in front of each line passed to standard output. The number for `-n` can exceed the number of physical machines in the ring: in this case several processes are emulated on one physical processor. `mpiexec` has problem to read long files from standard input. Therefore, for multiprocessor runs the input is expected in file `fort.1` (or it has to be linked to the input file). For a single processor run the code still expects input in the standard input stream. Note, that this prohibits from using command `mpiexec -n 1`: single processor run through `mpiexec`, which however does not make sense anyway. By default MPD places the root process on the local host (the machine there `mpiexec` has been started).

The process running on the host pipes its printing to the standard output (e.g. `b2.log` for the shown example, as usual). For other processes the output is piped to the files `output.0001`, `output.0002`, ..., etc. On each time-step EIRENE prints the extra file `mpistat.dat` with run-time data for different strata and different parts of the code. Those data can be used to optimize the parallelization (distribution of strata between processes etc.) It is recommended to set EIRENE variable ALLOC (Block 7, 3rd line) to zero. In this case only the specified number of particles (not the source strength) affects the time allocated for each stratum.

6.3 Modifications required in B2

To use B2.4 with MPI version of EIRENE one has to add two lines to the code:

1. File `driver/b2main.F`. Line:
   `CALL B2_MPI_INTERFACE`
   has to be added at the beginning of the subroutine.

2. File `driver/b2driv.F`. Line:
   `CALL FINALIZE_EIRSRT(LAST)`
   has to be added right before operator `GOTO 50`.

The first command skips B2 on all but root process.

Besides that, a bug fix related to neutral radiation in B2 has to be added (see subsection 9.1.

7 Automatic load balancing

7.1 Outlook

In case of stratified sampling parallelization of a Monte-Carlo algorithm can become a non-trivial task. The most obvious procedure would be to sample all strata on each processor, but in this case the amount of communications is maximal. It is also obvious that if the number of processors exceeds the number of strata, then one might use one processor for only one stratum, thus, reducing significantly the number of communications.

Therefore, optimizing the distribution of strata between processors can increase scalability of the code. In order to do this, a set of subroutines which perform such a distribution automatically was added to the code.

In general, the algorithm first distributes the processors into multi-processor bands (a single processor is a special case of such a band) and then distributes strata between those bands depending of the estimated run-time. The main steps of the algorithm are:

1. collecting the run-time information;
2. estimating the expected run-time for the next EIRENE iteration;
3. distributing processors into bands;
4. distributing strata between bands;

It is not a general limitation, but in many places the algorithm assumes homogeneous system (processors with equal performance).

The run-time information is collected by the subroutine `UPDATE_MPISTATDATA` (module `CPES.f`). At the end of an EIRENE run subroutine `NEXT_ITERATION_DATA` processes this information: it calculates the average time spent for one particle at each stratum, stratum overhead and communication time. A library of subroutines which distribute strata between processors is located in `mpi_balance.f`. Those subroutines are called in EIRENE through interface `AUTOMATIC_DISTRIBUTION2`.

Subroutine `EIRMPI_ESTIMATE_RUNTIME` estimates the cpu-time, which will be used to sample each stratum. After that, subroutine `EIRENE_BAND_BALANCE` yields the required distribution. This subroutine checks different variants of distributing processors into bands, some of them are calculated by subroutine `EIRMPI_DISTRIBUTE_BANDS`. For each of those variants subroutine `EIRMPI_BAND_BALANCE` distributes strata between bands. The variant with the most optimal distribution (with smallest estimated run-time) is returned.

The consistency of the result is checked finally in `EIRENE_BAND_BALANCE` (e.g., that no strata is missing in the resulting table `PROCFORSTRA`). It is also checked how much is the gain compared to a uniform distribution (all strata on each processor). If the gain with the "optimized" distribution is too small, then the uniform distribution is used. The total time allocated in EIRENE for sampling can be increased automatically, if necessary.

Interface `AUTOMATIC_DISTRIBUTION2` is called from subroutine `PEDIST` if the key word `AUTOMATIC_DISTRIBUTION` is placed in the block `*** INFORMATION_FOR_MPI` of the input file. The automatic distribution is applied only starting from the second EIRENE iteration: on the first iteration either prescribed or default (uniform) distribution is used.

The subsections below contain more detailed description of the individual steps of the algorithm.

7.2 Collecting the run-time information

Information about the run-time of the different parts of the code is collected by subroutine `UPDATE_MPISTATDATA`.

Time required for overhead of one stratum O_s is calculated as:

$$O_s = \sum_p \text{STATIS2...SUMOSTRA} + \frac{1}{P_s}\sum_p \text{INIT\_STRATA}$$

Here s is the index of stratum, p is the index of processor, P_s is the number of processors, on which stratum s is calculated, $\sum_p$ is the sum over such processors. Time for `INIT_STRATA` is divided by the number of processors because this part is identical on all processors.

Communication time C of strata s scaled to 2 processors is calculated as:

$$C_s = \frac{1}{\tau(P_s)P_s}\sum_p \text{CALSTR}$$

Here function $\tau(P)$ takes into account, how the communication time scales between P processors. In this case it is assumed that $\tau(P) = P$. Note, that `COUTAU` and `COLLECT_INFCOP` are not taken into account here, since they are related to global (not further optimizable) overhead (together with INPUT, OUTPUT, MODUSR etc.).

If no statistic on communication time is collected for stratum s, then the maximum communication time estimated from other strata is used.

Average time spent for one particle by the stratum s:

$$t_s = \frac{\sum_p T_{ps}}{\sum_p N_{ps}}$$

Here T_{ps} is the time, spent for sampling of stratum s on processor p, N_{ps} is the number of particles, sampled for stratum s on processor p.

7.3 Estimating the run-time of each stratum

The estimate of the expected run-time is based on the: i) time, allocated in MCARLO (variable XTIM); ii) prescribed minimum number of histories (NMINPTS, referred below as M_s); iii) t_s from the previous subsection. First, weights w of each stratum are calculated (subroutine `EIRMPI_ESTIMATE_RUNTIME`):

$$w_s = \frac{\text{XTIM}_s - \text{XTIM}_{s-1}}{\text{XTIM}_S - \text{XTIM}_0}$$

Here S is the number of strata.

The total time, available on all processors:

$$T_{tot} = (\text{XTIM}_S - \text{XTIM}_0)\,P$$

Here XTIM is taken on the root, P is the number of processors. Note that $\text{XTIM}_S - \text{XTIM}_0$ is equal to NTCPU/NPRS but not NTCPU, because this variable is corrected in routine MCARLO.

T_{tot}, w_s, t_s, and M_s are then sent to subroutine `EIRMPI_ESTIMATE_RUNTIME` which returns the expected run-time of each stratum T_s. This subroutine works as follows. For each stratum s it compares two times:

$$T_1 = T^*_{tot}\frac{w_s}{W} \qquad \text{and} \qquad T_2 = M_s \cdot t_s$$

At the beginning $T^*_{tot}= T_{tot}$ and $W = \sum_{s=1}^{S} w_s$. If $T_2 > T_1$, then time $T_s := T_2$, $T^*_{tot} := T^*_{tot} - T_2$ and $W := W - w_s$. Otherwise $T_s := T_1$. The cycle is continued for other strata. After the last stratum number S has been processed, the routine is repeated for strata, for which $T_s \neq T_2$. If no further assignments $T_s := T_2$ have been made during this cycle, then the iterations stop. This algorithm can lead to the result, that $\sum_{s=1}^{S} T_s > T_{tot}$. One also has to be careful: NMINPTS should be specified for all strata or not specified at all, otherwise some strata can get zero cpu-time and hence be turned off by EIRENE.

7.4 Distributing processors into bands

Estimated T_s, t_s, O_s and C_s are sent to subroutine `EIRENE_BAND_BALANCE`. In this subroutine, first, a relative time is calculated:

$$R_s = T_s/T_{cpu}, \qquad T_{cpu} = \frac{1}{P}\sum_{s=1}^{S}(T_s + O_s)$$

C_s is not taken into account at this stage.

The strata are divided into two groups: with $R_s > 1$ (multi-processor strata) and $R_s < 1$ (single-processor strata). The bands for multi-processor strata are allocated in subroutine `EIRMPI_DISTRIBUTE_BANDS`. At this stage, a homogeneous system is assumed. The number of processors allocated for multi-processor bands is $P_m = [\sum_{R_s>1} R_s]$, where brackets [...] denote closest smaller integer. The algorithm works as follows. Initially, each stratum receives $B_s = [R_s] + 1$ processors. The initial number of bands is equal to the number of multi-processor strata. After that loading of processors in each band is estimated: $L_s = R_s/B_s$, and the stratum with the smallest L_s is taken. For this chosen stratum, B_s is reduced: $B_s := B_s - 1$. If B_s stays > 0, then L_s is recalculated for new B_s, otherwise, this stratum is not checked any more. The routine is continued until $\sum_s B_s$ does not reach P_m.

Since this algorithm is very primitive and does not take into account many other factors (e.g. communication time, statistical variance per particle, etc.), several distributions are checked and the most optimal of them is returned. These distributions are the following:

1. Allocating P_m processors for multiprocessor bands, run `EIRMPI_BAND_BALANCE` to divide them into bands, the rest $P - P_m$ processors are treated as single-processor bands.

2. Same as previous, but the rest $P - P_m$ processors are combined into one multiprocessor band.

3. Allocate P_μ=min($P_m + 1$,P) processor for multi-processor bands, the rest $P - P_\mu$ processors are treated as single-processor bands.
4. Same as previous, but the rest $P - P_\mu$ processors are combined into one multi-processor band.
5. All available processors are combined into one multi-processor band.

In general, distributing processors into bands is the weakest point of deriving a load balancing procedure.

7.5 Distributing strata between bands

Strata are distributed between bands in subroutine `EIRMPI_BAND_BALANCE`. This is the most universal subroutine in `mpi_balance.f`. In principle, it can be used even on non-homogeneous systems. The input parameters: R_s, B_b and H_{sb}. Here B_b is the relative time, available on the band b, and H_{sb} is the relative overhead (including communications) as a function of stratum s and band b. B_b and H_{sb} are calculated in `EIRENE_BAND_BALANCE`, internal subroutine `PREPARE_DATA`. B_b is simply equal to the number of processors in the band, H_{sb}=$B_b \cdot O_s/T_{cpu}$+$P_s \cdot C_s/T_{cpu} \cdot \tau(P)$.

The algorithm used in `EIRMPI_BAND_BALANCE` works as follows. It looks for the stratum with largest R_s. After that, it looks for the band with largest available time $B_b - H_{sb}$ and puts this stratum there. The time, available on the band is recalculated: $B_b := B_b - R_s - H_{sb}$. The procedure is repeated for the remaining strata.

In the trivial case of strata with equal $R_s = S/P$ this algorithm leads to optimal distribution. However, it is easy to show that it does not always lead to mathematical optimum. E.g., P=2 and $R_s = (0.6, 0.5, 0.4, 0.3, 0.2)$. The resulting distribution will be: 1st CPU, $(0.6, 0.3)$; 2nd CPU, $(0.5, 0.4, 0.2)$. Whereas the optimal distribution would be: $(0.6, 0.4)$ and $(0.5, 0.3, 0.2)$.

Subroutine `EIRMPI_BAND_BALANCE` also returns the relative cpu-time, allocated on each band:

$$A_b = \sum_s (R_s + H_{sb})$$

where $\sum_s$ is calculated over all strata, placed on this band. A_b/B_b is then used in `EIRENE_BAND_BALANCE` to evaluate, which distribution is the most optimal.

8 Conclusions, Outlook

A fully backward compatible upgrade of B2-EIRENE (ITER version: SOLPS4.2) with parallelization of EIRENE itself and the EIRENE-B2 interface EIRCOP has been achieved, tested, and delivered to the ITER team. First results with the revised code have been presented both at PSI and EPS conferences 2008.

Algorithms of automatic nearly optimal distribution of strata (sources) between processors and synchronization of runs between processors have been developed. A factor of

6 speed-up on 8 processors can be achieved for regular ITER cases. Further optimization and parallelization of the serial (non Monte-Carlo) part of the code is necessary to further increase scalability.

Applying parallelization in B2-EIRENE code allows a more flexible way of using the computational resources and broadens the scope of applicability of the code. However, this also increases technical complexity and requires additional user support. Even for well-parallelizable (in theory) problems, like linear Monte-Carlo transport solvers, modifying a large existing code represents a big technical and numerical challenge. Its testing and commissioning requires large amount of time, because of inevitable occurrence of unpredictable problems in unpredictable places.

A key issue is now to identify optimal iteration pathways towards convergence, balancing iteration errors vs. statistical noise on the route to converged solutions. This work will be carried out under a special HPC project funded from FZJ and granted for this particular optimization study (July 2008 – June-2009) for SOLPS4.3.

References

[1] D. Reiter, http://www.eirene.de, “Manual”, Chapter 1

[2] Reiter et al., Fus. Sci. Technol. 47 (2005) 172

[3] B. Wilkinson, M. Allen, “Parallel Programming”, 2nd ed., Pearson Education Inc, Upper Saddle River, NJ (2005)

[4] W.R. Martin, T.-Ch Wan, T.S. Abdel-Rahmann, and T.N. Mudge, The International Journal of Supercomputer Applications, Vol. 1, No. 3, (1987) p57-74

[5] Reiter, D. May, Chr. et al.,, J.Nucl.Mat. Vol.220, p987 (1994)

[6] Börner, P, “Coupling of statistical “Monte-Carlo” solvers with numerical fluid solvers”, Diploma thesis, Univ. Hagen, Germany (2005) (in german), http://www.eirene.de, “Recent reports”

[7] Reiter, D., “Progress in 2-Dimensional Plasma Edge Modelling”, J. Nucl. Mat., Vol.196-198, p241 (1992).

[8] Maddison, G.P., Reiter, D. “Recycling Source Terms for Edge Plasma Fluid Models and Impact on Convergence Behaviour in the BRAAMS B2 Code”, KFA Jülich report, Jül-2872 March 1994

[9] Keunings, R., “Micro-Macro methods for the multi-scale simulation of viscoelastic flow using molecular models of kinetic theory”, Rheology Reviews 2004, 67-98

9 Appendix

9.1 Patch for neutral radiation in B2

An old bug in SOLPS.xxx (distinct from original B2-EIRENE as still maintained at FZJ) with respect to calculation of the neutral radiated power in B2 was fixed. The bug was related to calculation of the neutral radiation losses on the B2 side of the code system (the old treatment has been left as a "feature" for backward compatibility). The old way of calculating the radiated power `radarr` (B2, `neutrals.F`) is:

$$\texttt{radarr} = \sum_{\texttt{is}} \texttt{sni(is)} \cdot \texttt{pot(is)} - \texttt{see} - \texttt{edissml} \ .$$

The first error in this expression is the assumption that plasma fluid species "is" is identical (one-to one) to EIRENE atom species "IA". This may be true in all ITER applications since about 1995, but certainly not generally in B2-EIRENE, where the species relations between B2 and EIRENE have been kept more general already from the original design onwards.

In the above expression the sum is taken over all (assumed) atom species (strictly: ion species), `sni(is)` is the total particle sink of atom "`is`", `see` is the total energy loss of electrons, `pot(is)` is the ionization potential of atom "`is`", `edissml` is the dissociation energy of molecules, calculated in `WNEUTRAL` as `edissml` = `PMML` · 4.48 eV with `PMML`, apparently, the total loss rate of (hydrogenic) molecules. This treatment overestimates the total hydrogenic radiation, because it assumes that all the energy spent for dissociation which exceeds the potential energy of molecules (4.48 eV) goes into radiation, whereas in reality it goes mainly into kinetic energy of resulting atoms.

To correct this, one more array has been added to `EIRDIAG`: `eneutrad`, total neutral radiation. This array is filled in `WNEUTRAL` as:

$$\texttt{eneutrad} = \texttt{EAEL} - \sum_{\texttt{IA}} \mathrm{PAAT(IA)} \cdot \mathrm{POT(IA)}$$

It is thus the atomic radiation, calculated directly from EIRENE. `PAAT` and `EAEL` are default EIRENE tallies: particle sink for atom `IA` and electron energy loss due to atoms respectively. The radiation due to molecules and molecular ions is currently not taken into account. For this similar expressions, involving EIRENE tallies `EMEL` and `PMML`, etc... would have to be added.

In B2, `neutrals.F` the treatment of neutral radiation depends now on the value of switch `l_neutrad`: if it is ≤0, then the old implementation is used, otherwise `eneutrad` is taken. Note, that the variable `eneutrad` has to be also added to `EIRDIAGUSE` and to B2PLOT (file `ngread.F`). Extra printing from the version of A. Kukushkin was added to `WNEUTRAL` as well.

The following construction in `neutrals/neutrals.F` has been added:

```
    if(l_neutrad.le.0) then !{
```

```
c
c*** correct radiation losses for the molecule dissociation
c
        do iy=1,ny !{
      do ix=1,nx !{
        k=region(ix,iy,0)
        radarr(ix,iy)=radarr(ix,iy)-edissml(ix,iy,1,1)
        rad_neutr=rad_neutr-edissml(ix,iy,1,1)
        rdneureg(k)=rdneureg(k)-edissml(ix,iy,1,1)
        recycereg(k)=recycereg(k)-edissml(ix,iy,1,1)
      end do !}
        end do !}
      else !}{
c
c*** calculate radiation losses directly (at the moment only atoms)
c
        rad_neutr=0.
        rdneureg=0.
        recycereg=0.
        do iy=1,ny !{
      do ix=1,nx !{
        k=region(ix,iy,0)
        radarr(ix,iy)=eneutrad(ix,iy,1)
        rad_neutr=rad_neutr+eneutrad(ix,iy,1)
        rdneureg(k)=rdneureg(k)+eneutrad(ix,iy,1)
        recycereg(k)=recycereg(k)+eneutrad(ix,iy,1)
      end do !}
        end do !}
      end if !}
```

For consistency, one line has to be added to B2PLOT, file `double/ngread.F`:

```
       if(jvft44.ge.960511) then
        do is=1,nmoli
          call gfsub2(44,nxdd,nydd,nnx,nny,srcml)
          call gfsub2(44,nxdd,nydd,nnx,nny,edissml)
        end do
        if(jvft44.ge.20070912)
     .     call gfsub2(44,nxdd,nydd,nnx,nny,eneutrad) !VK
        read(44,'(2i6)') nlimi,nstsi
```

A bug with calculating the incident heat flux density on the outer target in B2PLOT has to be fixed as well.

(Note added after submitted draft version, in final version of this report (March 08):

This bug in B2PLOT has been found and fixed. Revised version of B2PLOT has been send to ITER Cadarache)

9.2 User features, added to EIRENE

Some minor user option were added during the current project.

- Variable `MPTS` in Block 7 (3rd line, second entry): a multiplication factor for the specified number of particles for all strata.
- An option which allows exact comparison of single- and multi-processor runs: reset of the random number generator after each `NPTSDEL` histories and initialize it in the same way, as it is initialized if one strata is running on several processors.

  Variable `NPTSDEL` is assigned for each stratum in EIRENE input block 7 (6th entry in the same line as `NPTS` - the number of histories). The corresponding code is added to `MCARLO`.
- In EIRENE input block 11, switches for diagnostic output, which passes through the buffer (printed using `WRITE(0,*)`):

```
XXXXX XXXXX XXXXX XXXXX XXXXX ...
TRCDBG2,TRCDBGE,TRCDBGM,TRCDBGF,TRCDBGL ...
TRCDBGS,TRCDBGG,TRCDBGMPI,TRCDBGC
```

`TRCDBG2`	more detailed level of output;
`TRCDBGE`	tracing from `EIRENE`;
`TRCDBGM`	... `MCARLO`;
`TRCDBGF`	... `FOLNEUT`;
`TRCDBGL`	... `LOCATE`;
`TRCDBGS`	... routines for surface interaction;
`TRCDBGG`	... geometry routines;
`TRCDBGMPI`	... MPI routines;
`TRCDBGC`	... coupling routines;

EIRENE now also prints a text file `mpistat.dat` containing run-time information for strata sampling and other parts of the code on each processor.