Proposal for the Use of High Level Support Team resources

Abstract

KInetic code for Plasma Periphery (KIPP), developed at IPP/Garching, was created as a tool to study kinetic effects of parallel plasma transport in the scrape-off layer and divertor. Experience gained in the code runs should form a scientific basis for implementation of kinetic effects (such as heat flux limiters/enhancements) in the present day 2D fluid codes such as SOLPS. Test runs of KIPP demonstrated the code's stability and extreme accuracy. Benchmarks carried out in the limit of strong collisionality demonstrated good match to analytical expressions with accuracy within 1% for most important transport coefficients: plasma electric conductivity, ion-electron energy equipartition rate for realistic ratio of deuteron to electron mass, parallel heat conduction and ion-electron thermoforce. The code's application to modelling real experimental profiles is presently hampered by its insufficient speed. Code optimisation for the speed of execution has been an integral part of its development right from the beginning. The critical issue for the code's speed up: MPI parallelisation enabling the most CPU consuming operations of Coulomb collisions to be split among large number of processors, each dedicated to a separate spatial location, has been successfully implemented. It is believed however that involvement of computational experts in the code optimisation can result in a further considerable speedup of this code.

Project Title	*Speeding up KInetic code for Plasma Periphery (KIPP)*
Project Acronym (up to 8 characters)	*KIPPADV*

Project coordinator:

Name of Coordinator:	Alex Chankin
Institution:	Max-Planck-Institute for Plasma Physics
Street Address:	Boltzmannstr. 2
City:	Garching
Country:	Germany
Email:	Alex.Chankin@ipp.mpg.de
Phone:	+49 89 32991844

Principal Investigator(s) [other than coordinator]:

Institution:	
Names of Investigators:	
Street Address:	
City:	
Country:	
Email:	
Phone:	

Please duplicate the table above if Principal Investigators from more than one Research Unit

Requirements for the present largest run of the code

Total amount of CPU hours	1000
Architecture(s) where application is already used	TOK-P Cluster
Number of CPUs	256
Memory requirements	1x256=256 Gb
Storage requirements	insignificant
Pure MPI or mixed communication (OpenMP+MPI)	Pure MPI
Own code / 3rd party code	Own code
Code publicly available (yes/no)?	no
Library requirements	MKL, MUMPS sparse matrix solver
Special requirements	none
Site name(s) where application is already used	IPP/Garching
Expected usage of the IFERC computer (yes/no)?	yes

Technical Improvement or adaptation work done so far

1. Do you apply in parallel for similar support from other institutions?
2. Has your code/project already received support (especially as part of a previous HLST call) related to improvement of its computational capabilities?

1. No
2. No

Request for work

a) Indicate nature (type) of HLST support being requested

b) Indicative level of support (in ppm[1])

a) Implementation of methods, algorithms and solver capable of speeding up the code.
b) 6 ppm

Involvement of the project proponents

Indicate the effort (in ppm[1]) of the projects proponents to be given (in parallel to the HLST work) to the execution of the project

6 ppm

Potential Impact

Indicate the estimated benefits that the HLST support activity will have on the software and physics modelling capabilities

The present version of KIPP was created by physicists, with limited support from computation experts available in the TOK division. Primary goals were: a) accuracy of the code in solving known problems, b) stability, c) speed, so that test code runs could complete in reasonable amount of time to allow further code development. The code is presently parallelised, with the most time consuming operations split over ~ 256 processors using MPI parallelisation.

It is believed that the execution time can be reduced by factor ~ 10 due to improvements in: parallelisation, including implementation of OpenMP parallelisation; choice of more efficient solvers; reorganisation of data flow including inter-processor communication.

[1] Note that 1ppy=12ppm

Detailed Project Description (max. 1-2 pages)

KIPP is a continuum Vlasov-Fokker-Planck (VFP) code used to describe parallel plasma propagation (along magnetic field lines, 'B-field') in the scrape-off layer (SOL) and divertor. The code is 1d2v: one spatial coordinate and two velocity coordinates: parallel velocity and gyro-averaged perpendicular velocity. It is written in Fortran 90. The motivation for creating this code comes from the well known fact, despite the edge plasma in many respects being strongly collisional, some transport coeffecients, in particular parallel heat conduction coefficient, require kinetic analysis, since they are determined by a minority of supra-thermal charged particles which are poorly collisional. The widely used 2D edge fluid codes such as SOLPS don't include kinetic effects, instead, they only use very simplified kinetic corrections.

For numerical solution of the VFP equation for the (presently, only) electron distribution function $f_e(v_{||},v_{\perp},s_{||})$, KIPP uses an operator splitting scheme, by alternating steps with electron-electron (e-e) plus electron (e-i) Coulomb collisions, and parallel propagation ('free-streaming'). Also present are effects of the parallel electric field (E-field) and the Debye potential sheath drop at the divertor target. KIPP assumes plasma quasi-neutrality, with ambipolarity of the plasma flow along B-field achieved by adjustment of the E-field.

1. Coulomb collisions

One of the most CPU consuming operations in KIPP is calculation of the effect of Coulomb collisions. For each spatial position, one processor (core) is dedicated, and calculations are done on a large number of processors simultaneously using MPI parallelisation. A fully implicit scheme to solve the Fokker-Planck (FP) equation is used. The effect of the E-field is part of the same implicit scheme. The FP equation is solved using the 9-point stencil to describe effects (in velocity space) of convection, diffusion and pitch angle scattering, represented by various derivatives of f_e. A Multifrontal Massively Parallel Solver (MUMPS) in a one processor mode is used to solve the FP equation. The typical velocity grid size is 256x512 and it takes ~ 0.1 s to solve the FP equation. Before the FP equation can be solved numerically, a number of other parameters and arrays need to be calculated as described below.

On each individual processor (dedicated to certain position $s_{||}$ in space) the solution of the FP equation at each time step requires prior sequential calculations of the two Rosenbluth potentials $H(v_{||},v_{\perp})$ and $G(v_{||},v_{\perp})$. Derivatives of these potentials determine transport coefficients in velocity space caused by Coulomb collisions. Rosenbluth potentials are calculated by solving the two Poisson equations in sequence, on a 5-point stencil again using the MUMPS sparse matrix solver.

Solutions of the above mentioned Poisson equations in turn require boundary conditions, different for H and G arrays. The boundary conditions, being 1D arrays in velocity space (against v_b - the array of velocities surrounding the velocity grid along its perimeter), are calculated as a product of the two Green functions (3D arrays in velocity space, e.g. $GRH(v_{||},v_{\perp},v_b)$) and 2D arrays in $(v_{||},v_{\perp})$ space with the subsequent summation over its elements using the vectorised *matmul* procedure.

Calculation of the two Green functions is the most CPU consuming operation. They however only depend on the velocity grid which doesn't change during the run. For this reason Green functions are calculated before the KIPP run and are stored in files. Reading them from the files is one of the first operations in KIPP. They have to be stored in the memory of each processor throughout the whole run. This to a large degree determines memory requirements of KIPP.

2. Parallel free-streaming

These calculations are also parallelised using MPI. Similar to Coulomb collisions, changes in f_e due to the free-streaming are calculated by each processor attached to a given spatial position. The 2^{nd} order high resolution explicit scheme used requires for each spatial cell knowledge of f_e from the current cell and 4 of its neighbours, 2 from each side. This requires an exchange of f_e's between individual processors presently achieved by four MPI_SENDRECV operations.

3. Splitting the job between host and other processors

Out of the total No. of processors, one (host processor) is dedicated to common tasks while all others – to Coulomb collisions and free-streaming operations for each spatial position. The host processor collects 1D arrays or macroscopic parameters (such as plasma density and temperature), calculates sources, sends them to individual processors, outputs results to the screen and writes them to files. The structure of the code was elaborated with the aim of splitting calculations among individual processors and minimising exchange between them.

4. Possible ways of speeding up

Speed up may potentially come from the following improvements:

- Using OpenMP parallelisation in addition to the MPI parallelisation for calculations on individual processors.
- Optimising data exchange among processors.
- For the solution of the FP equation, using more efficient sparse matrix solvers than MUMPS which can benefit from parallelisation (calculations involving MUMPS are done on one processor).
- For the solution of Poisson equations for Rosenbluth potentials, other schemes, apart from the MUMPS solver, e.g. based on iterative procedures, might be used.