The modern problem of computational radiative transfer

Team `Magritte`

Abstract

The aim is to formulate the modern problem of computational radiative transfer and develop the solution strategy that is used in `Magritte`: a new multidimensional accelerated general-purpose radiative transfer code.

1 Problem definition

We want to model the physical state at each point of a region in space at a certain time. To do this, we are given a set of

We want to

Consider a multidimensional space with a scalar field $I(\mathbf{x}, \hat{\mathbf{n}})$, depending on both the position $\mathbf{x}$ and the viewing direction $\hat{\mathbf{n}}$ in that space. The geometric nature of the problem can be deduced from the the transfer equation by explicitly writing all geometric (i.e. $\mathbf{x}$ and $\hat{\mathbf{n}}$) dependencies

$$\hat{\mathbf{n}} \cdot \nabla I(\mathbf{x}, \hat{\mathbf{n}}) \;=\; \eta(\mathbf{x}, \hat{\mathbf{n}}) - \chi(\mathbf{x}) I(\mathbf{x}, \hat{\mathbf{n}}) \tag{1}$$

1.1 Method of rays

1.1.1 Ray tracing algorithms

Trace structured rays through an unstructered grid.

1.2 What is a good input for `Magritte`?

We choose to solve the problem by directly integrating the transfer equation along each ray. The result of this calculation is the intensity at a certain point in a certain direction.
`Magritte`'s input consists of an unstructered set of N points in 3D space. For each of these points we have three position coordinates, three components of the velocity and a density.

We need to determine which point configurations are well-sampled by the ray tracer in `Magritte`.
Angular versus radial resolution

2 Data structure

In order to obtain optimal perfomance one needs to start with an optimal memory layout for all data structures.

C (and C++) are row-major in their memory lay-out.

Array of structures (AoS) versus Structure of Arrays (SoA). Currently we use an array of structures called CELL. How to go to structure of arrays? Straightforwarrd for the location and data members.

3 Input/output strategy

Since we want to develop a general-purpose code, we need to make sure that it can handle many different types of input. The input can be user generated or come from the output of a hydrodynamics code. For both cases we will consider the best way to handle the conversion.

3.1 Model input

`Magritte`needs as input a velocity and density distribution.

3.2 Hydro output as input

Fast conversion from the input grid to the grid used by `Magritte` *and back.*

3.2.1 AMR grid

The simplest way to handle an AMR grid input is to use the centers of the grid cells as set input grid points G. However, this tends to produce an oversampled grid.

3.2.2 SPH data

A truely general-purpose code should be able to tackle problems of any size i.e. the only restriction on the problem size should come from the amount of available CPU time. In order to achieve this one needs a way to devide the problem into smaller pieces which can be solved individually and communicated efficiently to form the solution.

4 Parallellisation strategy

4.1 Questions

- How are we going to devide the grid? Are there smart ways to cut over high opacities?
- At which point in the calculation do we want to communicate?

4.2 Strategies

Work with ghost cells. Devide the total of cells up in subsets, one subset for each CPU. For each subset add the first neighbor not in the subset.

5 Self-consistent temperature determination

The physical parameters at each point in space strongly depend on the temperature. In dedetmining the temperature we assume that this must be such that at each point in space there is no net heating or cooling, i.e. that at each point the heating is equal to the cooling.

Strategy: determine guess temperature on coarser grid.

Question:
How to obtain minimum and maximum temperatures from guess?

6 Technical

NOTE: The `next_cell` DOES NOT terminate properly!

Reduction: When we say that the grid is reduced around a certain cell this means that we remove the neighbouring cells. We only do this when a certain criterion is satisfied that assures us that the cell that we keep is representative for its neighbours.

To keep track of the cells after reduction, each cell has an identifier `cell[n].id`. This identifier will help us map the data form the reduced grid back to the original one.

$$\texttt{CELL cell [NCELLS]} \xrightarrow{\text{REDUCE}} \left\{ \begin{array}{l} \texttt{CELL cell\_ori [NCELLS]} \\ \texttt{CELL cell\_red [MCELLS]} \end{array} \right\} \xrightarrow{\text{APPEND}} \texttt{CELL cell\_app [NCELLS]}$$

We initialize the identifier to be the cell number `cell[n].id = n`. After reduction there are two grids. In the original grid, `cell_ori`, there are two possibilities for the identifier. If the cell is kept after reduction the identifier will still be equal to the cell number. If t Either the identifier is still equal to the number of the cell which means the the cell is kept after reduction, or the identifier is equal to the

References