Lessons learned in Developing a coupling interface between Kinetic PUI code (Fortran) and a Global MHD code (C++)

Laura Carrington, Ph.D.
Director of Performance Modeling and Characterization (PMaC) Lab
Team Objective:
To obtain a quantitative understanding of the dynamical heliosphere, from its solar origin to its interaction with the local interstellar medium (LISM), by creating a data-driven suite of models of the Sun-to-LISM connection.

• Use case: examining Voyager 1 data then using the new simulations to model galactic cosmic ray (GCR) trajectories starting at a point along the Voyager 1’s path. The show convincing evidence that GCR variations and the structure of the solar wind (SW)- (LISM) transition region are strongly correlated.
Kinetic PUI code (Fortran)

- kinetic PUI code models the nonthermal (pickup) ions (PUIs) created as new populations of neutral atoms are born in the SW and LISM. The PUIs generate turbulence that heats up the thermal ions. PUIs are further accelerated to create cosmic rays.
- Written in mix of Fortran77 and Fortran90 with both modules and common blocks.
- Serial code designed to compute the trajectory of single particle.
Global MHD code (C++)

• The Global MHD code simulates magnetic field in the SW and LISM using an AMR method where the data and computational work is distributed among the computational nodes via patches. Due to the different levels of refinement the amount and resolution of the data contained in the patches will vary.

• Written is C++ and Charm++.

• Parallel code.

The magnetic field data from the Global MHD simulation will be utilized by the Kinetic PUI code
Parallelization of Kinetic PUI code

- Parallelize the PUI code by creating a pool of trajectories to be computed (new data structures, random seeds, etc.)
- At each step in a trajectory magnetic field data is required from MHD code.
Parallelization of Kinetic PUI code

- Parallelize the PUI code by creating a pool of trajectories to be computed (new data structures, random seeds, etc.)
- At each step in a trajectory magnetic field data is required from MHD code.
- Trajectories take different number of steps – need to manage the completion of trajectories and the pool of trajectories.
Parallelization of Kinetic PUI code

• Parallelize the PUI code by creating a pool of trajectories to be computed (new data structures, random seeds, etc.)

• At each step in a trajectory magnetic field data is required from MHD code.

• Trajectories take different number of steps – need to manage the completion of trajectories and the pool of trajectories.

• Management of the output of each trajectory – some MPI ranks will complete multiple trajectories.
Sample of Data Layout of Two codes

MHD data layout

Original data

Decomposition into 16 patches

Distribution onto 16 MPI ranks
Sample of Data Layout of Two codes

**MHD data layout**

- **Original data**
- **Decomposition into 16 patches**
- **Distribution onto 16 MPI ranks**

**Trajectory computation**

Starting point of all trajectories

Each trajectory computed on separate MPI rank, 5000 total trajectories scheduled among the MPI ranks

Schedule the computation of each of the 5,000 trajectories in round robin on MPI ranks

Distribution onto MPI ranks
Extracting data from MHD code

• For a given point \((x,y,z)\) of a trajectory, magnetic field data is extracted from the MHD code.

• This extraction is done by each MPI rank searching its patch to determine if the point is within its patch.

• This search is done in parallel – every MHD rank is given a list of points of all the current trajectories.

• If there is a match for any of the points it returns the values of the data for those points.

• Data transfer is done using global tables.
Global table for trajectory coordinates

- **Step 1:** Create an X, Y, Z global coordinate table

To extract parameters three master tables (one for X, Y, Z) of the coordinates is filled in by each MPI rank then the final set of tables is distributed to each MPI rank.
Global table for trajectory coordinates

- **Step2: Example of creating a global X coordinate**

To create and distribute the table each MPI rank creates a table with NPROC rows. It fills every row with zeros except for the row number that matches its MPI rank (e.g. row id=MPI rank)

\[
\begin{array}{c|c|c|c}
\text{RANK #0} & X_0, Y_0, Z_0 & X_{-value} & 0 \\
& X_0 & 0 \\
& 0 & 0 \\
& \cdots & \cdots \\
& 0 & 0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{RANK #1} & X_1, Y_1, Z_1 & X_{-value} & 0 \\
& X_1 & 0 \\
& 0 & 0 \\
& \cdots & \cdots \\
& 0 & 0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{RANK #15} & X_{15}, Y_{15}, Z_{15} & X_{-value} & 0 \\
& 0 & 0 \\
& 0 & 0 \\
& \cdots & \cdots \\
& 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\text{X-value} & 0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & 0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & 0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & X_0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & X_1 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & X_{15} \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & X_0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & X_1 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & X_{15} \\
\end{array}
\]

\text{MPI\_Allreduce is used to sum the values in each row of the table to get a master table containing all the X-values}

\[
\begin{array}{c|c|c|c}
\text{X-value} & 0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & 0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & 0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & X_0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & X_1 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & X_{15} \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & X_0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & X_1 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
\text{X-value} & X_{15} \\
\end{array}
\]
Global table for MHD data

Step 1: Create an global MHD parameter table

The parameters for a given coordinate are extracted from the MHD data by identifying the patch that contains the specific coordinates.
Global table for MHD data

• Step 1: Create an global MHD parameter table

To create and distribute the table each MPI rank creates a table with NPROC rows. It fills every row with zeros. It loops through all the coordinates in the X, Y, Z tables to see if any are within its patch. If so then it extracts the corresponding parameters.

MPI_Allreduce is used to sum the individual parameter values in each row of the table to get a master table containing all the parameters. Each MPI rank’s coordinates and corresponding parameters are in the row that equates to its MPI rank (e.g., MPI rank=row #)
Lessons learned: C++ and Fortran routines

C++ code define as external function:
extern "C" {
    void ModCode_alloc();
}

Fortran code bind name:
subroutine ModCode_alloc() BIND(C,NAME='ModCode_alloc')
Lessons learned: Fortran common blocks in C++

C++ code define as external structure:

```cpp
extern "C" struct {
    double rp0[6];
    double rpb[6];
    double t;
} ming_walk;
```

Fortran code bind common block name:

```fortran
COMMON /ming_walk/ rp0,rpb,t
real*8:: rp0(0:5),rpb(0:5),t
BIND(C,NAME='ming_walk') :: /ming_walk/
```
Lessons learned: Vectorization on KNL & Skylake

- **KNL**: -xCORE-AVX512 gets 512-bit vectors
Lessons learned: Vectorization on KNL & Skylake

- **KNL**: -xCORE-AVX512 gets 512-bit vectors

- **Skylake**: -xCORE-AVX512 gets 256-bit vectors and -xCORE-AVX512 -qopt-zmm-usage=high gets 512-bit
  
  Default is -qopt-zmm-usage=low due to heat issues

  used [https://github.com/mlaurenzano/PEBIL](https://github.com/mlaurenzano/PEBIL) to determine instruction mix and lack of 512-bit instructions

- **-axCORE-AVX2** often faster than -axCORE-AVX512
Questions:
lcarring@ucsd.edu
www.sdsc.edu/pmac