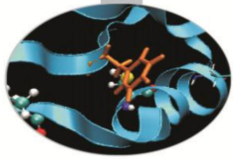
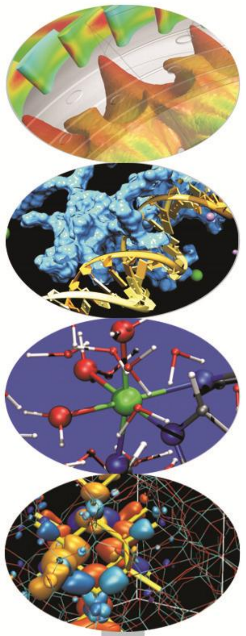
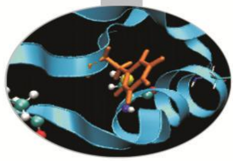
Profiling Exercise

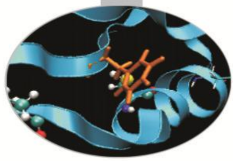



Matrix Multiplication



The program (molt_mat.f90, molt_mat.c), given the matrices A(L,M), B(M,N), C(L,N) computes C = A x B. Moreover if the value of the external loop counter is even, to the element (i,j) of the C matrix we add 4*i^i, otherwise 9*i^i. Check with gprof and gcov the hot spots inside the source and try to find out a solution to increase the efficiency.

```
// Matrix multiplication
for(i=0;i<a_r; i++) {
    for(j=0;j<b_c; j++) {
       for(k=0;k<a_c; k++){
          if (i%2==0) {
            inc=4*pow((double)i,(double)i);
            c[i][j]=c[i][j]+a[i][k]*b[k][j] +inc;
            }
          else {
            inc=9*pow((double)i,(double)i);
            c[i][j]=c[i][j]+a[i][k]*b[k][j] +inc;
          }
       }
   }}
```



Pi by quadrature

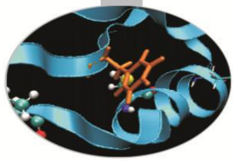
-

It is known that the mathematical constant π can be approximated by computing the following formula:

$$\pi = 4 \int_0^1 \frac{1}{1+x*x}\, dx$$

The value of the above integral can be approximated by numerical integration, i.e. by computing the mean value of the function f(x) = $\frac{1}{1+x*x}$ in a number of points and multiplying per the x range. This can be easily done in parallel by dividing the [0,1] range into a number of intervals.

Pi by quadrature



Using paraprof profiler execute the following steps:

•Compile the source `pi.c` or `pi.f90` using tau configuration scripts

–`tau_cc.sh -o pi pi.c`

•Run the executable with 4 MPI processes using the input file “iterations.inp” and visualize the results with the TAU profiler.

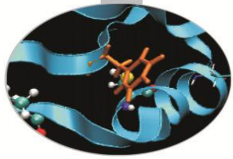
–`mpirun –np 4 pi.exe < iterations.inp`

–`paraprof`

•Analyze the results using the following windows:

–“Bar chart”, “Statistic Test”, “Statistic Table”, “Call graph”, “Call Path Relations”

Source code: *pi.c*



Pi by quadrature

•The timing information on the function “f()” is correct?

•Disable function throttle and run again the executable and TAU with “paraprof”

•Can you see any difference on the timing result and on the weight of the function “f()” respect to total execution time?

•Try to find a solution to optimize the code.

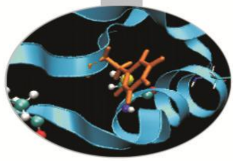
Source code: *pi.c*

c

Matrix Multiplication

- Use TAU compiler instrumentation to profile matrix multiplication program
- Analize result with paraprof
- Create a selective intrumentation file for pdb
- Perform pdt-based profile analysis and use paraprof to visualize result

Source code: *matrix.cc*

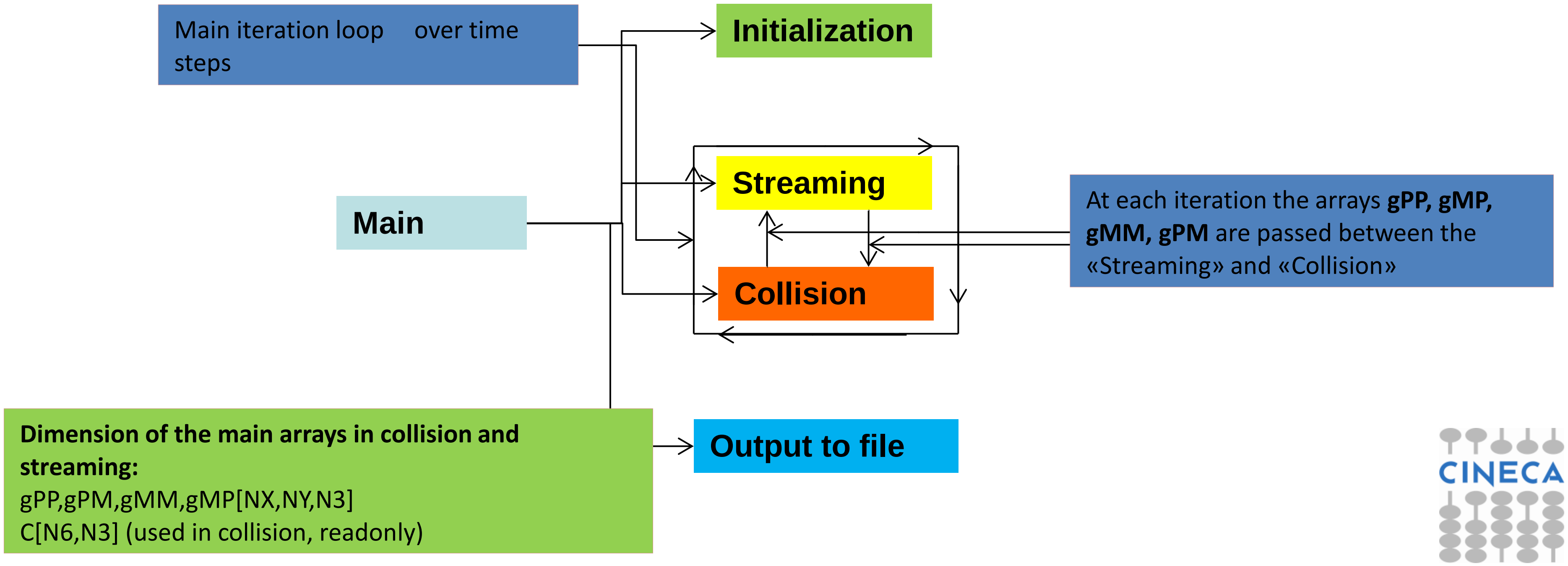


Rarefied Gas Dynamics Code

The serial code "Raref_gas_dyn.c" solves the semi-deterministic Boltzmann equation for studying turbulence and instability of rarefied gas dynamics.

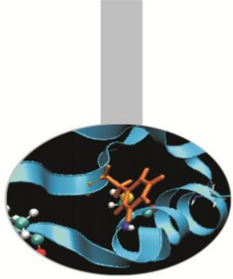
In the code we have two main functions "streaming()" and "collision()" while all the other ones are only utility functions.

•The flow chart of the code can be sketched this way



Main iteration loop over time steps
Initialization
Streaming
Main
At each iteration the arrays **gPP, gMP, gMM, gPM** are passed between the «Streaming» and «Collision»
Collision
Dimension of the main arrays in collision and streaming:
gPP,gPM,gMM,gMP[NX,NY,N3]
C[N6,N3] (used in collision, readonly)
Output to file

Rarefied Gas Dynamics Code



•Compile the code using the TAU scripts

•Analyze the results

–Use the windows “Bar chart”, “Statistic Table”

–Can you find any function where most of time is spent?

•Look at that function in the source code and analyze carefully the loops:

```
–for(int i=0; i<NX;i++){
–   for(int j=0; j<NY;j++){
```

•Can you find any parallelization strategy using OpenMP?

–Take care about shared and private variables inside the loops above

•Insert the OpenMP directives inside the function collision according to your strategy

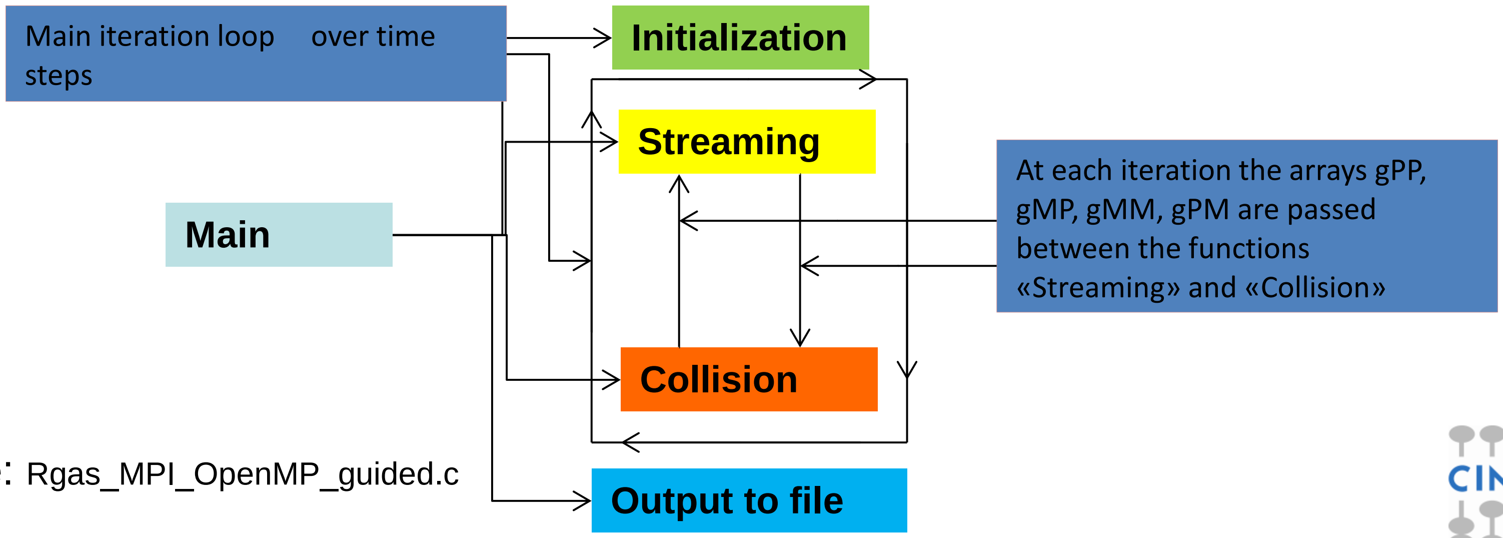
•Test the speedup obtained with OpenMP making a scalability test from 1 to 4 cores

–If parallelization is correct, you should obtain the following results

Rgas-OpenMP	Elapsed Time (sec)	Speed up	Efficiency %
1	270	1,00	100%
2	136	1,99	99%
4	83	3,25	81%

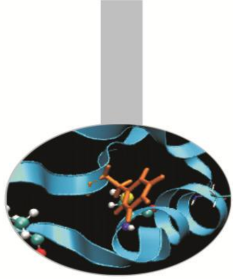
Rarefied Gas Dynamics Code

- Compile the code using the TAU scripts
- Analyze the results
  - Use the windows “Comparison Window”, “3D Visualization”
  - The load balancing among threads is good?
- OpenMP has the limit to work just on a single node.
- Look again at the function “collision” in the source code . In addition to the strategy implemented with OpenMP can you find a way to add a domain decomposition with the MPI directives thus to exploit the computing power of a cluster.
- Look at the following dependency graph for hints. Do you think it’s possible to decompose along the k dimension of the loop “for(int k=0;k<N3;k++)” contained in “collision”?


Main iteration loop over time steps
Initialization
Streaming
Main
At each iteration the arrays gPP, gMP, gMM, gPM are passed between the functions «Streaming» and «Collision»
Collision
Output to file

Source code: Rgas_MPI_OpenMP_guided.c

Rarefied Gas Dynamics Code



•Try to implement the domain decomposition following the guidelines in the source code : `Rgas_MPI_OpenMP_guided.c`

•Compile and run the code

–Use the script `compile_openmp_mpi_tau.sh`

–Run the executable : `mpirun -np 2 -x OMP_NUM_THREADS=2 executable`

•Analyze the results using TAU

–Use the windows “Comparison Window”, “3D Visualization” and “Statistic Table”

–The load balancing among threads remains good?

•Looking at the main function in the program you can notice that the “streaming” function is executed only by the master process which broadcasts the arrays gPP, gMM, gMP, gPM to the other processes.

•For large simulations the MPI_Bcast() time, which is done at each iteration, can increase significantly.

–Can you find a better solution to the use of MPI_Bcast()?

Source code: Rgas_MPI_OpenMP_guided.c

Source code: Rgas_MPI_OpenMP_optim.c