INAF HPC course 2024

hands-on session

Jacobi solver
OpenMP implementation

Core algorithm analysis

The Jacobi algorithm with a serial code has:

- time complexity of the order $O(I \cdot N^2)$, where I is the number of iterations to achieve convergence (which is a function of the tolerance, usually ~10^{-5}) and N is the grid size;
- memory complexity of the order $O(N^2)$.

So we expect:

- long times to process matrix of big size and/or great amount of iterations (cache misses rate severely impacts on performance when the matrix size exceeds cache size);
- size of the matrix is limited by the platform (i.e. size of the memory available on the node). On modern platforms, this may be a minor issue.

OpenMP implementation

We remind that the main loop of the Jacobi iterator takes the form:

```
#define TOL  1.e-5
while (err > TOL)
{
     for (int j=0; j<Dimension ; j++)
          for (int i=0 ; i<Dimension ; i++)
               {...}
     err = DIFF(...);
} /* while loop */
```

How to parallelize?

The while loop cannot be parallelized using explicitly the **`#pragma omp parallel for`** OpenMP directive, because the iterations of the while loop are not specified.

OpenMP implementation

```
int i, j;
while (err > TOL)
{
  #pragma omp parallel default(none) shared(...) private(...) \
                                     num_threads(NThreads) ...
   {
      #pragma omp for schedule(...) ...
      for (j=0; j<Dimension ; j++) {
            for (i=0 ; i<Dimension ; i++) {
              {...}
            } /* loop over i */
      } /* loop over j */
   } /* omp parallel */
} /* while loop */
```

How to parallelize with OpenMP?

- the outer for loop can be easily parallelized;
- using the clause `schedule(static)` the loop over the `j` index is parallelized across the threads with chunk size equal to `Dimension/NThreads`;
- NThreads is the number of OpenMP threads working within the parallel region, specified through the clause `num_threads(integer-expression)`;
- number of threads and schedule type can be set at runtime using `OMP_NUM_THREADS` and `OMP_SCHEDULE` environmental variables.

OpenMP implementation

- the loops are (i) perfectly nested, i.e. there is no intervening code between the loops, (ii) they form a rectangular iteration space and the bounds of each loop are invariant over all the loops, (iii) the instructions of the innermost associated loop do not contain any break statement nor continue statement. This allows code improvements;
- the parallel section is created and destroyed (fork-join model) for every iteration. This might lead to a performance penalty;
- with the introduction of the cancellation constructs in OpenMP 4.0, an elegant way to terminate the execution of a parallel construct is available. This feature is useful for specific parallel methods with dynamic behavior, such as, for instance, the `while` worksharing loop;
- a more sophisticated way to support unstructured parallelism, such as unbound loops and recursive functions, is offered by the tasking execution model, first introduced in OpenMP 3.0, and refined in later versions.