Parallelisation of a Staggered Grid solver

Heisig, Hammer, Ernst

FAU

February 2, 2014

Outline

Parallelization basics

- Why parallelize your code?

Pro

- more compute power
- more memory
- parallel computing is the future

Con

- added code complexity
- communication overhead
- Increased power consumption

Don't parallelize without profiling and performance modelling!

- MPI in a nutshell
  The **Message Passing Interface**
  - call your program with `mpirun -np <N> <NAME> <ARGS>`
  - spawns `<N>` identical processes
  - only `MPI_MPI_Comm_rank(...)` gives different results

Typical usage:

Implementation

- Implementation

  The following steps must be parallelized
  - SOR::solve()
  - SOR::residual()
  - SOR::normalize()
  - determineNextDT()
  - refreshBoundaries()
  - computeFG()
  - composeRHS()
  - updateVelocities()

  Most of the time is spent in the SORSolver, so this is the focus.
- Domain partitioning
  - Usually domain is split in roughly quadratic tiles
  - We chose the simpler approach: Split in horizontal stripes
  - Pro
    - easier to implement
    - fast access patterns along the cachelines
  - Con
    - bad surface / size ratio for large number of processes

Results

- Results

  Was it worth the effort?

  Explanation:
  - SOR or Jacobi solver does not scale well

  Use a better algorithm before writing parallel code!

Possible improvements for numerical codes

- Possible improvements for numerical codes

  Use LISP