EGU22-13438
https://doi.org/10.5194/egusphere-egu22-13438
EGU General Assembly 2022

Loop blocking ("tiling") in NEMO

Daley Calvert[1], Mike Bell[1], Francesca Mele[2], Italo Epicoco[2], Sebastien Masson[3], Maff Glover[1], and Gurvan Madec[3]

[1]Met Office, Exeter, UK
[2]CMCC Foundation, Lecce, Italy
[3]Sorbonne Universités (UPMC, Univ Paris 06)-CNRS-IRD-MNHN, LOCEAN Laboratory, Paris, France

NEMO is an example of a memory-bound code, where computational performance is limited by memory bandwidth due to intensive main memory access. This can be mitigated by more efficient use of the memory hierarchy, ensuring that data is held in fast low-level cache for as long as possible. Loop blocking or "tiling" is one way of achieving this, by partitioning loops so that data access is divided into smaller blocks that fit into cache.

This optimisation technique has been applied to several numerical ocean models, including MITgcm, ROMS and CROCO. We describe an implementation of tiling in the NEMO 4.2 release candidate and its performance in realistic global configurations.

The MPI domain is partitioned horizontally (over latitude and longitude) by modifying DO loop bounds and local working array declarations. A DO loop at the time-stepping level iterates over the tiles asynchronously. Lateral boundaries use an extended halo (two points wide instead of one) so that MPI communications can be suppressed when the tiling is active. Tiling has so far been implemented in the active tracer, vertical diffusion and dynamical code sections of NEMO.

Tiling of the longitudinal axis always results in a loss of performance while tiling of the latitudinal axis can perform very well, reducing the execution time of individual code sections by up to 40% in ORCA2 simulations. However, this is offset against the cost of additional calculations due to the two-point halo, which increases the execution time of individual code sections by up to 15%. Additionally, the number of calculations is proportional to the number of tiles, further reducing potential performance gains from the tiling. This can be mitigated by removing halo calculations where possible.

Overall tiling performance in ORCA025 is significantly worse compared to ORCA2. This is because a greater proportion of time is spent in code that is not tiled (75% in ORCA025 vs 45% in ORCA2) and because the performance of some tiled code is worse. Tiling in the vertical, as well as the horizontal, not only recovers the performance of this tiled code but improves it beyond that in the ORCA2 simulations.