Gung-Ho Dynamics:

A primal-dual finite element based method for solution of the rotating shallow water equations on the sphere.

Some notes on optimization of the algorithm.

John Thuburn
17 December, 2012

These notes flag up some issues and questions related to optimization of the primal-dual finite element shallow water code. Some are algorithmic, related to convergence of the solver. Some are computational science issues related to parallel performance and scalability.

- **Solver.**
  - *Number of outer iterations.* All tests so far have used 4 outer iterations of the solver (compare the 2×2 used by ENDGame). Is it possible to use fewer? This will depend on the convergence rate obtained, which will be related to the other issues mentioned below.
  - *Inverses of J, M, and H operators.* Several operators need to be inverted at each time step: J once, M n times, and H $3n+1$ times, where n is the number of outer iterations. These are well conditioned, sparse matrices. Currently the inverse is computed by (i) making a diagonal approximation, and (ii) taking a number of underrelaxed Jacobi iterations. There is scope for optimization of the diagonal first guess (e.g. some kind of mass lumping), and for optimization of the underrelaxation parameter. Any optimization should be robust to the choice of grid (e.g. hexagons versus quadrilaterals). After the first outer iteration it should be possible to accelerate convergence by using the result of the previous outer iteration as the first guess.
  - *Approximate inverse of M.* In the current algorithm an approximate inverse of M, $\mathcal{M}^{-1}$, appears in the semi-implicit linearization and hence in the Helmholtz operator. Initial experiments using a diagonal approximation gave slow convergence of the outer iterations. In the current implementation $\mathcal{M}^{-1}$ is constructed by a diagonal approximation followed by a single Jacobi iteration. (Thus $\mathcal{M}^{-1}$ has the same stencil as M itself.) There is scope for improving the diagonal first guess (e.g. some kind of mass lumping, as above). Also, enlarging the stencil for $\mathcal{M}^{-1}$ enlarges the stencil for the Helmholtz operator, with possible performance implications; this trade-off could be investigated. Alternative methods for solving the nonlinear system that do not involve approximating M^{-1} might be worth investigating.
  - *Multigrid solver.* The relaxation subroutine in the multigrid solver uses underrelaxation. Experiments to date suggest that this performs satisfactorily, but there may be some benefits from fine tuning. However, any such fine tuning should be robust

to the choice of grid (e.g. hexagons versus quadrilaterals). The multigrid solver currently uses all the grids in the multigrid hierarchy. Theory and experiment suggest that it should only be necessary to coarsen until $\Delta x \sim c\Delta t$ where c is the speed of the fast waves. Moreover, coarsening only as far as necessary should be beneficial to parallel performance (reduced communications, and processors don't run out of work). It should be straightforward to automate a good choice of multigrid depth.

- **Re-use of stencils / connectivity arrays.**

  Currently, for generality, every operator has its own stencil info, resulting in a significant amount of duplication (and unnecessary data transfer etc.). For a specific discretization, some re-use of stencil information (e.g. via pointers) will be possible.

- **Global reduce / colouring.**

  Almost all operations (kernels) in the current code loop over target entities rather than source entities, avoiding the need for MPI global reduce operations and/or 'colouring' to ensure correctness of results and bit reproducibility. The only exception is the `prolong` kernel. It would be straightforward to store the stencil and weights in transpose form to allow a loop over target entities. (They are also needed in their current form for the `restrict` operator.)

- **Advection.**

  Currently the code loops over edges and builds a polynomial fit in the cell upstream of each edge in order to compute the flux. This means that the polynomial fit may be built several times in any given cell (especially on a hexagonal grid). Is it better to loop over cells first and build and save polynomial fits, then loop over edges to compute fluxes? Is it better to precompute and retrieve more quantities (e.g. position vectors of edge ends)? Is it possible to reduce data transfers by precomputing and retrieving fewer quantities?

- **Storage of multigrid data.**

  Currently multigrid information is stored in arrays that are the same size on all grids. This is obviously wasteful and may hit performance.

- **Operators on all grids.**

  Not all operators are needed on all grids in the multigrid hierarchy; only those operators involved in the Helmholtz operator and its inverse are needed on all grids. For the others, removing the `igrid` index from their coefficients and stencils could improve performance and save memory.

- **Diagonal operators.**

  Some operators (e.g. `massL`) are diagonal but are coded to allow for possible non-diagonal case. Exploiting the fact that they're diagonal should be more efficient.

- **Helmholtz operator.**

  The helmholtz operator is currently built 'on the fly' from four simpler operators. There may be some advantage in automatically building the stencil and weights for the Helmholtz operator from those of the four simpler operators, which would reduce some computation (and perhaps communications). But this would have to be done on all grids in the multigrid hierarchy at every step (because the Helmholtz operator coefficients change). Also,

building the Helmholtz stencil and coefficients would itself involve some computation and communications, so the trade-off is not clear.