Adaptive Mesh Refinement Full Approximation Scheme (AMRFAS)

Chris L. Gebhart

March 4, 2019

1 Variables, Notation, and Syntax

In general, variables which refer to tokens in `Proto` are written using a `monospaced font`. Vector-like objects are written in **bold**, and sets use black-board font (e.g. $\mathbb{R}$). Occasionally words are typed in bold-face simply for **emphasis**.

Variable Name	Variable Definition
ϕ	Independent variable, generally a potential
R	Right hand side of Poisson's Equation
r	Residual
dx	Spacing in ALL spatial directions
λ	Relaxation parameter
F	Flux or Flux Register
N or `DIM`	Number of Dimensions
Ω	Subset of Space, usually a Proto::Box
Ω_F^C	Coarsened Fine domain (e.g. invalid region of coarse domain)
Γ	Bitbox

2 Data Structures

In the current implementation of AMRFAS there are 3 levels of data structure: the AMR level (`AMRFAS*.H`), the Multigrid level (`Multigrid*.H`), and the operator level (`TestOp.H`).

`TestOp.H` contains the `TestOp<DATA>` class. The template parameter `DATA` corresponds to the patch-level data holder used by the algorithm. At the time of writing, `DATA` must be a valid template parameter of CHOMBO's

LevelData class, however it is likely that in the future DATA will be a PROTO object: either Proto::BoxData or something derived from it (e.g. for embedded boundary methods).

2.1 Operator Level

TestOp contains the following data members:

- All Stencil objects needed to compute a cell-to-face flux
- All Stencil objects needed to compute a face-to-cell divergence
- An interpolation InterpStencil for course-to-fine prolongation
- An interpolation InterpStencil for boundary interpolation in the case of AMR (not used for vanilla Multigrid operations)
- An averaging Stencil for fine-to-coarse restriction
- An integer defining the refinement ratio between the current and next coarser level.
- A Real representing the grid spacing of this level
- A Real representing the relaxation parameter λ
- A LevelData<DATA> used as a temporary for some operations (this might be moved into Multigrid later to improve encapsulation)

The operations specifically used by TestOp are:

- $flux_i$ computes the flux into each cell from the low face (in direction i):
$$flux_i(\phi_i) = \frac{\phi_i - \phi_{i-1}}{dx}$$
- div computes the divergence:
$$div_i(flux[i]) = \frac{flux[i+1] - flux[i]}{dx}$$
- L is the operator itself and is equivalent to a $2 * DIM + 1$ Point Laplacian:
$$L(\phi) = \sum_{i=1}^{DIM} div_i(flux_i(\phi))$$

- *avg* is a conservative, linear average:

$$\langle \phi \rangle = \frac{1}{N} \sum_{i=1}^{N} (\phi_i)$$

- *interp* is a piecewise-constant interpolation
- *interpBC* is a 3rd-order accurate quadratic interpolation

Most low-level subroutines are currently managed at the operator level. This will most likely change in the final AMRFAS API to improve encapsulation and user work flow:

Algorithm 1 Residual

procedure RESIDUAL(r, ϕ, R) ▷ inputs are `LevelData<DATA>&`
  $exchange(\phi)$
  $r \leftarrow R - L(\phi)$
  return $absMax(r)$ ▷ output is usually unused

Algorithm 2 Relax (Multigrid Version)

procedure RELAX(r, ϕ, R, n) ▷ r, ϕ, and R are `LevelData<DATA>&`
  for i **in** $range(0, n)$ **do**
    $exchange(\phi)$
    $residual(r, \phi, R)$
    $\phi \leftarrow \phi + \lambda * r$

Algorithm 3 Coarsen

procedure COARSEN(ϕ_C, ϕ) ▷ inputs are `LevelData<DATA>&`
  $temp_C \leftarrow avg(\phi)$
  $copyTo(temp_C \rightarrow \phi_C)$

Algorithm 4 CoarseRhs (Multigrid Version)

procedure $\textsc{CoarseRhs}(R_C, \phi_C, \phi, R)$ ▷ inputs are LevelData<DATA>&
  $exchange(\phi)$
  $temp_C \leftarrow avg(R - L(\phi))$
  $copyTo(temp_C \rightarrow R_C)$
  $exchange(\phi_C)$
  $R_C \leftarrow R_C + L(\phi_C)$

Algorithm 5 FineCorrection

procedure $\textsc{FineCorrect}(\phi, \phi_C, \phi_{C0})$ ▷ inputs are LevelData<DATA>&
  $\phi_{C0} \leftarrow \phi_C - \phi_{C0}$
  $copyTo(\phi_{C0} \rightarrow temp_C)$
  $\phi \leftarrow \phi + interp(temp_C)$

Algorithm 6 interpBoundary

procedure $\textsc{bitPoint}(\Omega, bitRatio)$
  return $low(\Omega)/bitRatio$
procedure $\textsc{getCoarseEdge}(p, n, \Omega_p^C, bitRatio)$
  $\Omega_n \leftarrow Box(n, n)$
  $\Omega_n^C \leftarrow refine(\Omega_n, bitRatio/refRatio)$
  $\partial\Omega_{p,n}^C \leftarrow \Omega_n^C \cap \Omega_p^C$
  return $\partial\Omega_{p,n}^C$
procedure $\textsc{interpBoundary}(\phi, \phi_C)$ ▷ inputs are LevelData<DATA>&
  $copyTo(\phi_C \rightarrow temp_C)$
  $\Omega^F \leftarrow domainBox(\phi)$ ▷ bounding box of refined area
  $\Gamma^F \leftarrow \Omega^F/patchSize$
  for each patch $\phi_i, temp_{C,i}$ **in** $\phi, temp_C$ **do**
    $p_i \leftarrow bitPoint(box(\phi_i))$ ▷ compute the bit point of this patch
    for each neighor n_j of p_i **do**
      if $n_j \ni \Gamma^F$ **then**
        $\partial\Omega_{p,n}^C \leftarrow getCoarseEdge(p_i, n_j, box(temp_{C,i}), bitRatio)$
        $\phi_i \leftarrow interpBC(temp_{C,i}) \mid \partial\Omega_{p,n}^C$

Algorithm 7 computeRhs

procedure REFLUX(R_C, ϕ_C, ϕ, F)
    $F \leftarrow 0$ ▷ Initialize
    $exchange(\phi)$
    $exchange(\phi_C)$
    $F \leftarrow incrementFine(flux(\phi)) \mid \Omega_F$
    $F \leftarrow incrementCoarse(flux(\phi_C)) \mid \Omega_C$
    $R_C \leftarrow R_C + \frac{-1}{dx_C} * F$ ▷ e.g. `F.reflux(RC, -1/(refRatio*dx))`
procedure COMPUTERHS (AMR VERSION)$(R_C, \phi_C, \phi, R, \rho_C, F)$
    $exchange(\phi)$
    $exchange(\phi_C)$
    $copyTo(\rho_C \rightarrow R_C)$ ▷ initialize R_C
    $reflux(R_C, \phi_C, \phi, F)$
    $temp_C \leftarrow \langle R - L(\phi) \rangle$
    $copyTo(temp_C \rightarrow R_C)$ ▷ overwrites Ω_F^C including reflux garbage
    $R_C \leftarrow R_C + L(\phi_C) \mid \Omega_F^C$

2.2 Multigrid Level

The code in `Multigrid*.H` is very minimal at the time of writing, and contains the operations needed to compute a Multigrid V-Cycle with or without the interpolation of boundary conditions (needed in the AMR case). After refactoring, some of the subroutines present in `OP` may be moved here to mitigate code duplication.

Member data of `Multigrid` include:

- `m_level` where 0 is the coarsest
- `m_op` an instance of the operator upon which `Multigrid` is templated
- `m_phiC, m_phiC0, m_RC` coarse level quantities computed on this level. Not used on (or allocated for) level 0.
- `m_coarser` a recursive `Multigrid` instance. Each `Multigrid` object controls a single level.
- `m_amrInterp` an `InterpStencil` used to interpolate boundary conditions to this level when embedded in an AMR hierarchy.
- `m_phiCAMR` the next coarser AMR level from which we interpolate boundary conditions.

Algorithm 8 VCycle (Non-AMR version)

procedure VCYCLE(ϕ, R)
    if `level == 0` **then**
        $relax(\phi, R, BOTTOM\_RELAX)$
    else
        $relax(\phi, R, PRE\_RELAX)$
        $coarsen(\phi_C, \phi)$
        $copyTo(\phi_C \rightarrow \phi_{C0})$
        $coarseRhs(R_C, \phi_C, \phi, R)$
        $vcycle(\phi_C, R_C)$ ▷ Call `vcycle` on next coarser `Multigrid`
        $fineCorrection(\phi, \phi_C, \phi_{C0}$
        $relax(\phi, R, POST\_RELAX)$

Algorithm 9 VCycle (AMR Version)

procedure VCYCLE(ϕ, ϕ_C^{AMR}, R)
    if `level == 0` **then**
        $relax(\phi, R, BOTTOM\_RELAX)$
    else
        $copyTo(\phi_C^{AMR} \rightarrow \phi_{C,temp}^{AMR})$
        $interpBoundary(\phi_C, \phi_{C,temp}^{AMR}, amrInterp)$
        $relax(\phi, R, PRE\_RELAX)$
        $coarsen(\phi_C, \phi)$
        $copyTo(\phi_C \rightarrow \phi_{C0})$
        $coarseRhs(R_C, \phi_C, \phi, R)$
        $vcycle(\phi_C, R_C)$ ▷ Call `vcycle` on next coarser `Multigrid`
        $fineCorrection(\phi, \phi_C, \phi_{C0}$
        $relax(\phi, R, POST\_RELAX)$

2.3 AMRFAS Level

The structure of the `AMRFAS` object mirrors `Multigrid`. Again, it likely makes sense to move some of the functionality out of `OP` into this `AMRFAS` once the code is refactored. `AMRFAS` is templated on an operator `AMR_OP` which is effectively the same as the `OP` parameter of `Multigrid`.

The members of `AMRFAS` are as follows:

- `level`, an integer for the AMR level of this object. Level 0 is the coarsest.
- `mg` a `Multigrid` object
- `op` an instance of `AMR_OP` with the flags for AMR turned on
- `phi_C0` temporary storage for ϕ.
- `coarser` the next coarser instance of `AMRFAS`.
- `reflux` an instance of `LevelFluxRegister` used for refluxing.

The only real code in `AMRFAS` is the V-Cycle algorithm. In the following description, a superscript "AMR" denotes a full AMR hierarchy. The analogous variables without this superscript represent data on the current level (or the next coarser level if there is a subscript "C").

Algorithm 10 AMRVCycle

procedure AMRVCYCLE($\phi^{AMR}, \rho^{AMR}, r^{AMR}, R$)
    if `level == 0` **then**
        $vcycle(\phi, R)$ ▷ normal MG V-Cycle
    else
        $interpBoundary(\phi, \phi_C)$
        $vcycle(\phi, \phi_C, R)$ ▷ MG V-Cycle with BC interp
        $coarsen(\phi_C, \phi)$
        $copyTo(\phi_C \to \phi_{C0})$
        if `level > 1` **then** ▷ At least 2 coarser levels exist
            $coarsen(\phi_{CC}, \phi_C)$
            $interpBoundary(\phi_C, \phi_{CC})$
        $computeRhs(R_C, \phi_C, \phi, R, \rho_C, F)$
        $AMRVcycle(\phi^{AMR}, \rho^{AMR}, r^{AMR}, R_C)$ ▷ Recursive call
        $fineCorrection(\phi, \phi_C, \phi_{C0})$
        $interpBoundary(\phi, \phi_C)$
        $vcycle(\phi, \phi_C, R)$ ▷ MG V-Cycle with BC interp
    $residual(r, \phi, R)$