Writing an FAC Solver

Brian Gunney

Introduction

The SAMRAI library provides high-level abstractions for writing linear equation solvers using using the fast adaptive composite (FAC) algorithm. The FAC algorithm and all operations common to it are provided in the class `FACPreconditioner`, while problem-specific operations are accessed through the interface of the abstract `FACOperatorStrategy` class.

Thus, `FACPreconditioner`

- provides the entry point for the solution process.
- controls the FAC cycles and parameters such as number of cycles, residual tolerance, and number of smoothing sweeps
- provide temporary storage for error and residual vectors.

The `FACOperatorStrategy` class provides interfaces to these problem-specific operations:

- restricting the solution
- restricting the residual
- prolonging the error and applying the correction
- solving the coarsest level
- computing the residual norm

These operations require the details of the equation and discretization methods and must be implemented in the the user's concrete child class.

To develop a solver using the FAC algorithm, one uses a combination of the preconditioner and a concrete implementation of the operator strategy abstract class. This document describes the methods contained in these two classes, how to implement `FACOperatorStrategy` and how to use the preconditioner. For an example of an FAC operator class, see `CellPoissonFACOps`. For an example of using the preconditioner, see the higher-level class `CellPoissonFACSolver`.

Using `FACPreconditioner`

This section covers the setup and usage of the preconditioner. For users of the solver, this is the most important part. The operator strategy methods are called through the FAC preconditioner class. Each preconditioner object requires one operator strategy implementation, which is registered through the preconditioner's constructor.

Setting up the preconditioner parameters involves these self-explanatory preconditioner methods and corresponding input database parameters:

Method	Input name	Default setting
`setPresmoothingSweeps(int num_pre_sweeps)`	`num_pre_sweeps`	1
`setPostsmoothingSweeps(int num_post_sweeps)`	`num_post_sweeps`	1
`setMaxCycles(int max_cycles)`	`max_cycles`	10
`setResidualTolerance(double residual_tol)`	`residual_tol`	1.00E-006
`enableLogging(bool enable)`	`enable_logging`	FALSE

The method

```
     solveSystem(SAMRAIVectorReal<double> &u,
                 SAMRAIVectorReal<double> &f);
```

performs the solve. The unknown u and right-hand-side f are described as vectors so that systems of equations are handled through the same interface. Vectors can wrap multiple patch data under a single object. The two vectors must have the same hierarchy, level range and number of components. Further requirements, such as ghost cell width, may be imposed by the operator object.

`solveSystem` initializes the solver state (set up temporary storage, etc.), performs the FAC cycling steps according to the above parameters, then deallocate the solver state. For multiple solves, the solver state can be set up and preserved across the solves, leading to significant time savings. The methods

```
     void initializeSolverState(
        const SAMRAIVectorReal<double> &solution ,
        const SAMRAIVectorReal<double> &rhs );
```

and

```
     void deallocateSolverState();
```

are used set up and remove the solver state manually. If `solveSystem` is entered with an initialized state, that state will be used but left undisturbed. Otherwise, the state is initialized using the vector arguments to `solveSystem`. Note that `initializeSolverState` calls the `FACOperator::initializeOperatorState` and `deallocateSolverState` calls `FACOperator::deallocateOperatorState` to keep that object in a matching state. The FAC operator's methods are described below.

After a solve, the number of FAC iterations, the residual norm and the convergence factors can be retrieved by the functions

```
     int getNumberIterations() const
     void getConvergenceFactors(double *avg_factor,
                                double *final_factor) const
     double getResidualNorm() const
```

The convergence factor is the factor by which the residual is reduced by one FAC iteration. The average factor is that which, when applied the number of iterations used gives the same overall reduction, while the final factor is that of the last iteration taken. The residual norm is the RMS norm of the residual.

Implementing `FACOperatorStrategy`

Critical methods that must be implemented in `FACOperatorStrategy` are the pure virtual functions performing essential operations for the FAC solver.

- `restrictSolution`
- `restrictResidual`
- `prolongErrorAndCorrect`
- `smoothError`
- `solveCoarsestLevel`
- `computeCompositeResidualOnLevel`
- `computeResidualNorm`

In addition, there are three non-pure virtual functions that are not related to the FAC algorithm but help in the implementation of the strategy class:

- `initializeOperatorState`
- `deallocateOperatorState`
- `postprocessOneCycle`

This section will make general comments about these methods. The source code documentation of this abstract class provides the full description, requirement and allowable assumptions when implementing the methods.

As with the `FACPreconditioner` class, patch data is passed to these methods through vectors. When implementing `FACOperatorStrategy`, you can assume that the vectors are those given to `initializeOperatorState`, those given to `FACPreconditioner::solveSystem` or clones of those vectors, leading to some consistency through out the FAC solve.

Each of the essential operations for the FAC solver, with the exception of `computeResidualNorm`, operates on just one level. The level number argument in these methods always refers to the level whose data is being changed. The restrict and prolong operations obviously require data from an adjacent level, but they modify only one level.

The computation of the residual norm by `computeResidualNorm` should only compute the norm of the data passed in. It should not recompute the residual, which is done by `computeCompositeResidualOnLevel`.

Acknowledgements:

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence-Livermore National Laboratory under contract No. W-7405-Eng-48..