Multigrid Pattern

I. Problem

Problem domain is decomposed into a set of geometric grids, where each element participates in a local computation followed by data exchanges with adjacent neighbors. The grids represent a system of discretized equations that needs to be solved iteratively until a convergence (or solution above predefined level of accuracy) has been reached. The convergence rate depends on the granularity of the grids, however it is difficult to start with an optimal grid size in a static manner. An iterative solver is given for the system of grids to define the computational requirements that lead to the convergence. The solver performs differently with varying accuracy depending on the behavior of the solution space, making it more difficult to choose a fixed grid granularity that meets efficient convergence criteria. This leads to a suggestion of using multiple levels of grids for the computation on the system of equations.

II. Driving Forces

1. **Performance** in achieving the convergence must be higher than naïve approach (i.e. with one static grid size).
2. **Tradeoffs** between performance and accuracy must be balanced when applying varying level of grid sizes.
3. When going from finer grids to coarser grids (and vice versa), there must be an **algorithm** that defines how to derive different granularity of grids from current set of grids.
4. When shifting to coarser grids, it becomes more difficult for the processors have equal amount of work. **Load balancing** must be taken into account in the parallelization process at each level, while maintaining a stable **communication to computation ratio**.

III. Solution

1. Multilevel Grids

   There is a family of parallel iterative solvers for discretized finite element methods that gives equivalent results as the sequential versions(i.e. Jacobi and Red-Black Gauss-Seidel solvers). In these parallel solvers, the data is decomposed into a geometrical partitions distributed among the processors, each of them responsible for executing finite element computations with exchange of data between neighboring processors. The iterative solvers produce efficient performance because of the work distribution, however they are known to have slow asymptotic convergence rate. This limits their usefulness in practice.

   For each of the local grid computation, an error is defined to be the difference between the real solution and the estimated value in the current iteration.

Depending on the grid, the error components might converge in a smooth manner, or exhibit an oscillatory behavior as they come to the convergence. Most of the iterative solvers make rapid progress in reducing the error initially, and settle down to a slow asymptotic phase. Particularly, the error reduction is done rapidly for the oscillatory error components, but slowly for the smoother components of the error. Eventually, we want to solve the system of equation at a finest grid level. But this can turn out to be a slow process when the solvers fine grids settle down at smooth error components.

Multilevel grids can enhance the slow computation process at the smooth level by varying the size of the grids. Namely, the component that appears smooth on fine grid may appear oscillatory when sampled on coarser grid. Thus, if we apply the solver on coarser grid, then we can make rapid progress on the oscillatory error components. After a few iterations, the values are "interpolated" back to the finer grid, and the solution now contains more rigorously reduced errors than results that might have occurred by solving in one fixed level of grid. We can view the coarser grid as correction grids, accelerating convergence of the finest grid by efficiently making progress on smooth error components. This idea can be recursively applied to hierarchy of multiple levels of grid sizes where correction is made for one level by interacting with the higher grids. Note that unlike adaptive mesh refinement, a uniform mesh levels will be applied to all grids within the same level.

For problem $Ax = b$, and approximate solution $\hat{x}$, a residual function is defined to be: $r = b - A\hat{x}$, and the error $e = x - \hat{x}$ satisfies equation $Ae = r$.

Instead of applying the original problem and right-hand side, we can use the residual to improve the accuracy of the solution. Residual function is particularly helpful when guess a starting solution--zero is a reasonable choice.

We are now ready to describe the Multigrid algorithm.

1. On fine grid, use few iterations of the solver to compute approximate solution $\hat{x}$ for system $Ax = b$.
2. Compute the residual of from the solution. $r = b - A\hat{x}$.
3. Resample residual to coarser grid.
4. On coarse grid, use few iterations of the solver to obtain coarse-grid approximation.
5. Interpolate the results from coarse-grid back to the fine grid to obtain improved approximate solution.
6. Continue more iterations on the fine grid.

Algorithm 1. Multigrid Algorithm

Step 4 can be taken recursively, to take advantage of even coarser grid which would recursively sample at another coarser grid and so on. This forms a hierarchy of grids at different levels, and there are several way of interacting between the levels in practice. Commonly used interactions are shown in figure 1.

V-cycle start with finest grid and goes down through successive levels to coarsest grid and then back up again to finest grid. W-cycle zig-zags among lower level grids, where computations are cheaper, before moving back up to finest grid. Full Multigrid start at coarsest level, then alternates for a while, eventually reaching the finest grid. The choice of interaction cycle heavily depends on the nature of each problem.

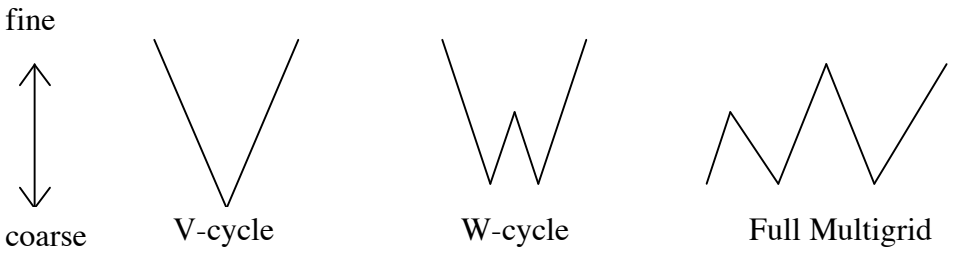

Figure 1: Multigrid Interaction Cycles

In short, Multigrid method avoids the weaknesses of the underlying iterative solvers and instead tries to exploit the strength of them by shifting the grid resolution to where it works best. Eventually at some point, all components of error will appear to be oscillatory, which will be reduced rapidly by the solver. Thus convergence rate of the entire Multigrid solver will be independent of a particular grid size.

2. Varying Grid Levels

When shifting from one grid level to another, the grid boundary need to be redefined. For structured (rectangular, cubical) data domain, this just means dividing the grid size at each dimension by a constant factor c. For programmability and efficiency, the maximum number of grids in the data domain is usually defined exactly to be a factor of c. This ensures no odd-sized grids will be left over in the boundary when switching between finer and coarser grids.

For unstructured grids, however, varying the level of grid size is not straightforward. Irregular grids have no easy way of deriving coarser grid from finer grids. There have been many approaches to achieve such derivation, and frequency used technique used in practice is an algorithm called *Maximal Indepedenct Set*.

An *independent set* is a set of vertices $I \subseteq V$, in graph $G = (V,E)$, in which no two members of I are adjacent. A *maximal independent set (MIS)* is an independent set where if any more vertex is brought into the current set, the property of independence is violated. MIS is NP-Hard, but we can use the following simple greedy algorithm as an approximation, as shown in Algorithm 2. Based on this algorithm, we devise a parallel MIS. Each processor will be given a partition of vertices to begin with. Each vertices will have a unique random number assigned to it. For all vertices in the partition each processor can pick a vertex with smallest

number and put it into the independent set. All the adjacent vertices will be discarded. Each processor continues process until all vertices have been examined.

```
forall v in V
      if v.state = = undone then
            v.state = selected
            forall v1 in v.adjacent
                  v1.state = deleted
I = all v in V such that v.state == selected
```

Algorithm 2. Greedy Maximal Independent Set

When MIS has been determined locally, we can connect the vertices of MIS to have a logical coarser grid. Since the MIS is distributed evenly throughout the graph the sample of data to follow is valid representation of the grid partition.

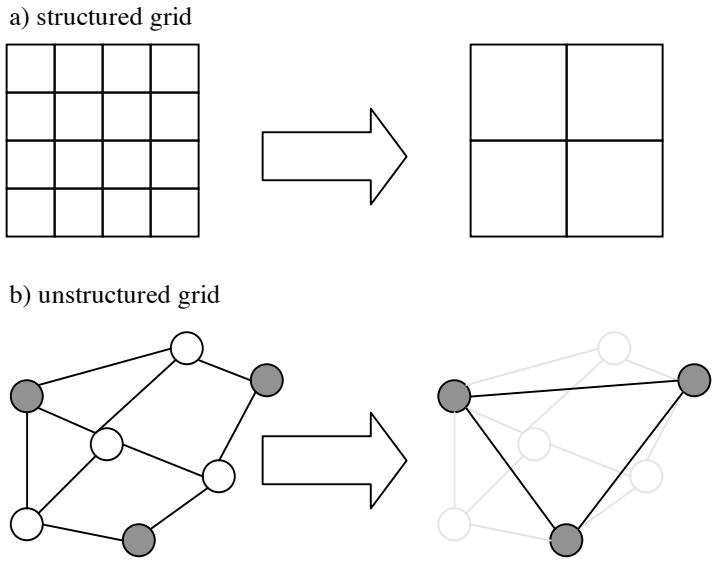

Figure 2. Deriving coarse grid from fine grid
a) structured grid using division in each dimension
b) unstructured grid using Maximal Independent Set Algorithm

3. Parallelization

Usually parallelization is done at each level partitioning the grid data spatially in geometric shapes. As the grid level gets coarser, the communication to computation ratio goes up, and the workload often becomes uneven between the processors. In practice, there are three ways to handle coarser grids in Multigrid.

a) Load Balancing
Using one of the load balancing technique described in LoadBalance Pattern, we can redistribute the work across the processors. But when data gets redistributed across other processors, locality of computation

will be damaged significantly. For example, when interpolating from coarser to finer grid, the computation might not be local any more.

b) Idle Processor
To lower the communication to computation ratio, we can lower the number of processors used. However, this incurs low processor utilization rate.

c) Additive Multigrid
This is one of the popular techniques used in practice, but require caution. Instead of parallelizing at only one level, we can parallelize at all levels simultaneously, relaxing the sequential requirements between levels. Processors can obtain enough workload without having to go through the load balancing process. This technique, however suffers in many cases from lack of convergence.

IV. Difficulty

1. Interaction cycle has to been determined by the designer to ensure best convergence rate.
2. For unstructured grids, graph partitioning and coarsening algorithm needs to be implemented.
3. Depth of the grid resolution needs to be carefully tuned after the implementation.
4. Boundary data and ghost cells might have to be implemented for programmability between processors.
5. Load balancing will introduce non-local computations between level interactions.

V. Related Patterns

Adaptive Mesh Refinement, LoadBalancing, GraphPartitioning

VI. Example

Finite Element Method for PDE

Behavior of physical objects can be explained and predicted with system of differential equations. Some of the basic equations that model common natural phenomenon include Maxwell's equations for electro magnetic field, Navier-Stokes equations for fluid dynamics, Linear elasticity equations vibrations in an elastic solid, Schrodinger's equations for quantum mechanics, and Einstein's equations for general relativity. Partial differential equations(PDE) have partial derivatives of an unknown function with respect to more than one independent variables.