Adaptive Mesh Applications

Sathish Vadhiyar

Sources:
- Schloegel, Karypis, Kumar. Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes. JPDC 1997 (**Taken verbatim**)

Adaptive Applications

- Highly adaptive and irregular applications
- Amount of work per task can vary drastically throughout the execution – similar to earlier applications, but..
- Has notions of "interesting" regions
- Computations in the "interesting" regions of the domain larger than for other regions
- It is difficult to predict which regions will become interesting

AMR Applications

- An example of such applications is Parallel Adaptive Mesh Refinement (AMR) for multi-scale applications
- Adaptive Mesh – Mesh or grid size is not fixed as in Laplace/Jacobi, but interesting regions are refined to form finer level grids/mesh
- E.g.: to study crack growth through a macroscopic structure under stress

AMR Applications – Crack propagation

- Such a system is subject to the laws of plasticity and elasticity and can be solved using finite element method
- Crack growth forces the geometry of the domain to change
- This in turn necessitates localized remeshing

AMR Applications- Adaptivity

- Adaptivity arises when advances crosses from one subdomain to another
- It is unknown in advance when or where the crack growth will take place and which subdomains will be affected
- The computational complexity of a subdomain can increase dramatically due to greater levels of mesh refinement
- Difficult to predict future workloads

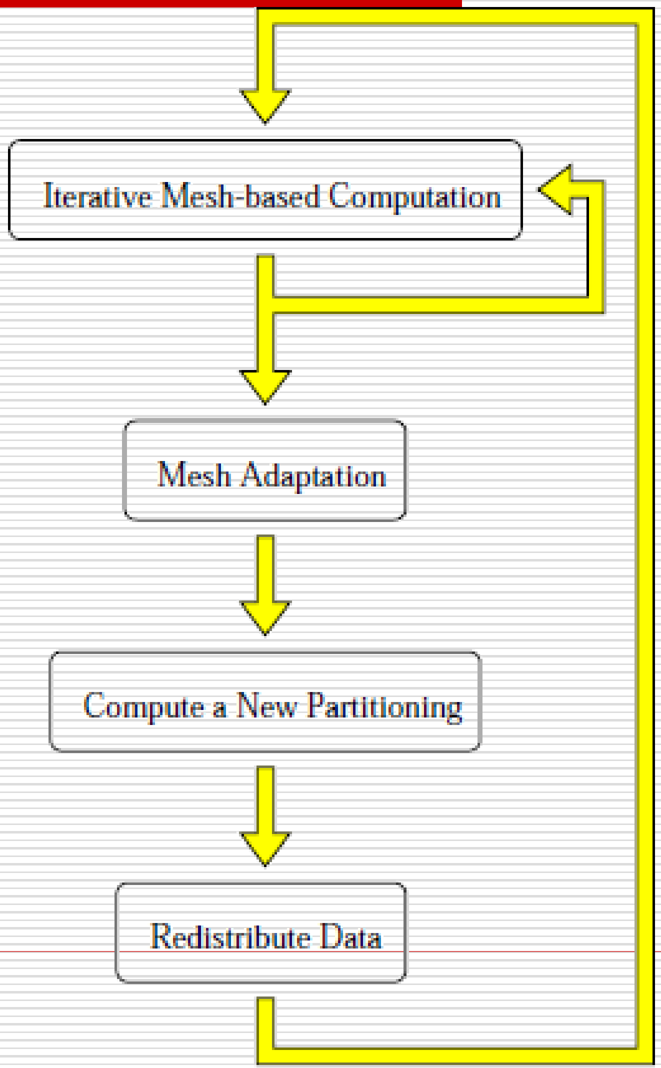
Repartitioning

- In adaptive meshes computation, areas of the mesh are selectively refined or derefined in order to accurately model the dynamic computation
- Hence, repartitioning and redistributing the adapted mesh across processors is necessary

Repartitioning

- The challenge is to keep the repartitioning cost to minimum limits
- Similar problems to MD, GoL
- The primary difference in AMR is that loads can drastically change; cannot predict; will have to wait for refinement, then repartition

Structure of Parallel AMR



Repartitioning

- 2 methods for creating a new partitioning from an already distributed mesh that has become load imbalanced due to mesh refinement and coarsening
- **Scratch-remap** schemes create an entirely new partition
- **Diffusive schemes** attempt to tweak the existing partition to achieve better load balance, often minimizing migration costs

Graph Representation of Mesh

- For irregular mesh applications, the computations associated with a mesh can be represented as a graph
- Vertices represent the grid cells; vertex weights represent the amount of computations associated with the grid cells
- Edges represent the communication between the grid cells; edge weights represent the amount of interactions

Graph Representation of Mesh

- The objective is to partition across P processors
  - Each partition has equal amount of vertex weight
  - Total weight of the edges cut by the partition is minimized

Scratch-map Method

- Partitioning from scratch will result in high vertex migration since the partitioning does not take the initial location of the vertices into account
- Hence a partitioning method should incrementally construct a new partition as simply a modification of the input partition

Notations

- Let B(q) be the set of vertices with partition q
- Weight of any partition q can be defined as: $W(q) = \sum_{v_i \in B(q)} w_i$
- Average partition weight: $\overline{W} = \frac{\sum_{i=1}^{p} W(i)}{p}$
- A graph is imbalanced if it is partitioned, and: $\exists q \mid W(q) > \overline{W} \times (1 + \epsilon)$

Terms

- A partition is over-balanced if its weight is greater than the average partition weight times $(1+\epsilon)$
- If less, under-balanced
- The graph is balanced when no partition is over-balanced
- Repartitioning – existing partition used as an input to form a new partition

Terms

- A vertex is clean if its current partition is its initial partition; else dirty
- Border vertex – adjacent vertex in another partition; those partitions are neighbor partitions
- TotalV – sum of the sizes of the vertices which change partitions; i.e., sum of the sizes of the dirty vertices

3 Objectives

- Maintain balance between partitions
- Minimize edge cuts
- Minimize TotalV

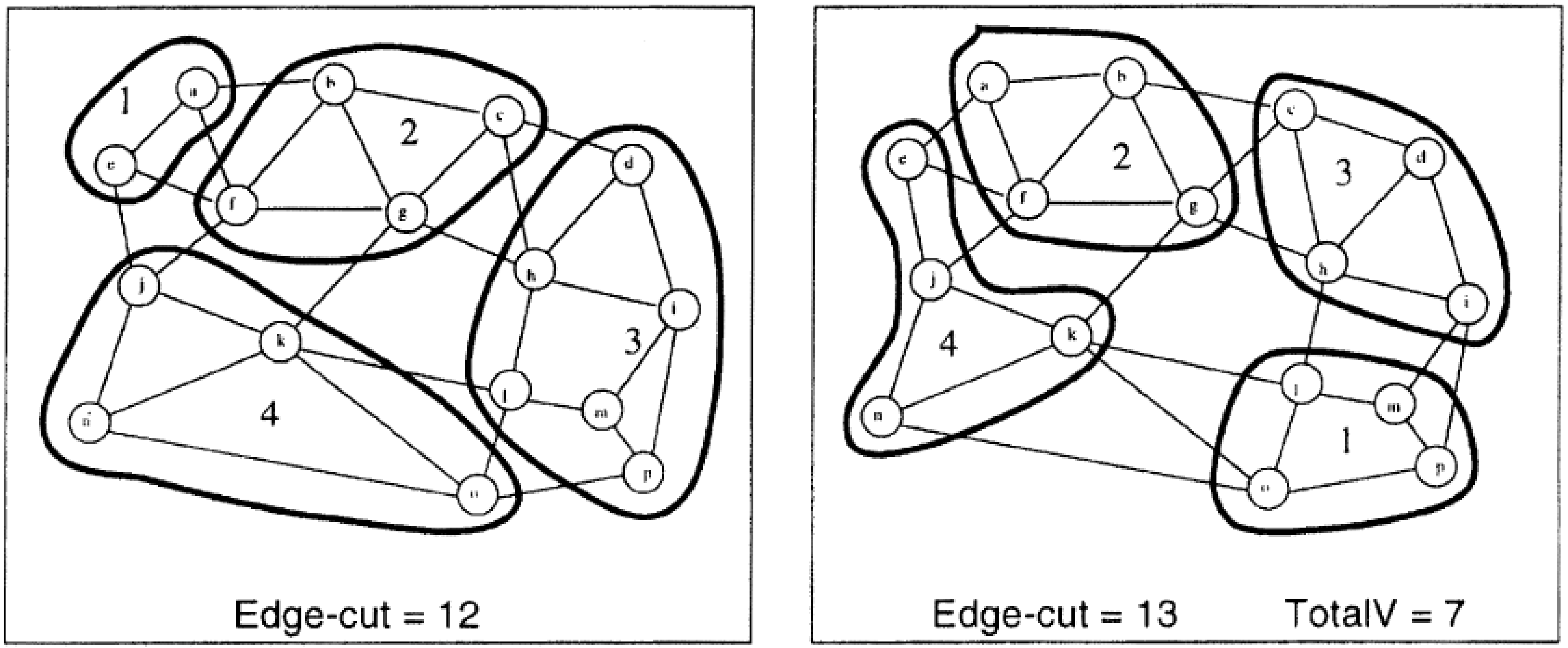
Different Schemes

- Repartitioning from **scratch**
- **Cut-and-paste** repartitioning: excess vertices in an overbalanced partition are simply swapped into one or more underbalanced partitions in order to bring these partitions up to balance
- The method can optimize TOTALV, but can have a negative effect on the edge-cut

Different Schemes

- Another method is analogous to **diffusion**
- Concept is for vertices to move from overbalanced to neighboring underbalanced partitions

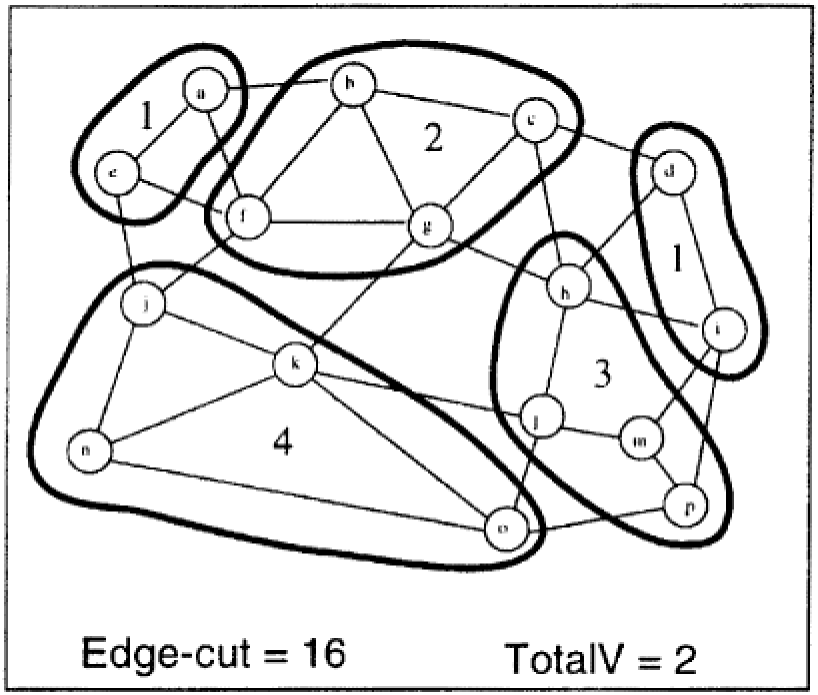
Example
(Assuming edge and vertex weights as equal to 1)



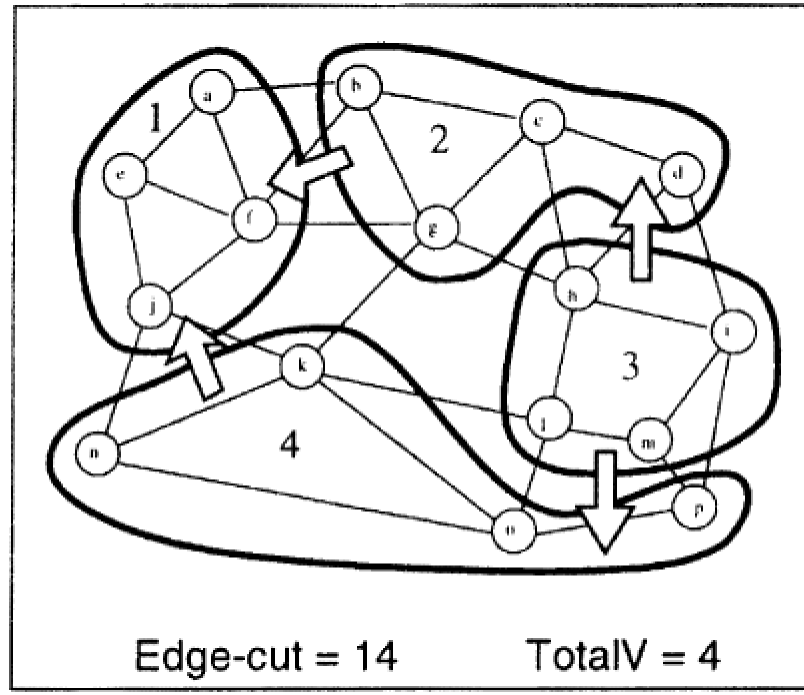
(a) Original Graph

(b) Partitioning from Scratch

Example (contd..)



(c) Cut-and-Paste Repartitioning



(d) Diffusion Repartitioning

Analysis of the 3 schemes

- Thus, cut-and-paste repartitioning minimizes TotalV, while completely ignoring edge-cut
- Partitioning the graph from the scratch minimizes edge-cut, while resulting in high TotalV
- Diffusion attempts to keep both TotalV and edge-cut low

Space Filling Curves for Partitioning and Load Balancing

Space Filling Curves

- The underlying idea is to map a multidimensional space to one dimension where the partitioning is trivial
- There are many different ways
- But a mapping for partitioning algorithms should preserve the proximity information present in the multidimensional space to minimize communication costs

Space Filling Curve

- Space filling curves are quick to run, can be implemented in parallel, and produce good load balancing with locality
- A space-filling curve is formed over grid/mesh cells by using the centroid of the cells to represent them
- The SFC produces a linear ordering of the cells such that cells that are close together in a linear ordering are also close together in the higher dimensional space

Space Filling Curve

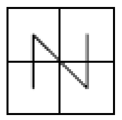
- The curve is then broken into segments based on the weights of the cells (weights computed using size and number of particles)
- The segments are distributed to processors; thus cells that are close together in space are assigned to the same processor
- This reduces overall amount of communication that occur, i.e., increases locality
- Repartitioning for load balancing also involves less communications only between neighbors

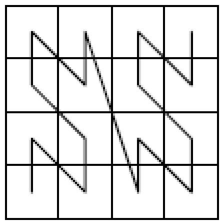
SFC representation

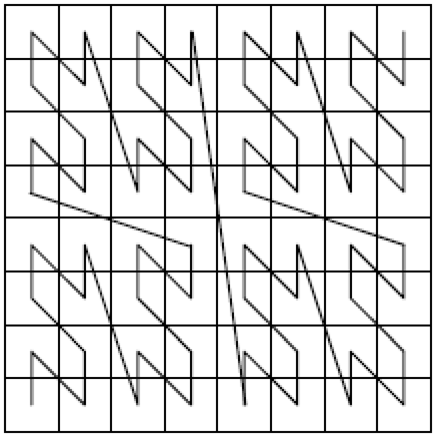
- One method is to define recursively – curve for a 2^k x 2^k grid composed of four 2^(k-1) x 2^(k-1) curves
- Another method is using bit interleaving – the position of a grid point along the curve can be specified by interleaving bits of the coordinates of the point
- The interleaving function is a characteristic of the curve

Z-curve or Morton ordering

- The curve for a 2^k x 2^k grid composed of four 2^(k-1) x 2^(k-1) curves, one in each quadrant of the 2^k x 2^k grid
- Order in which the curves are connected is the same as the order for 2x2 grid

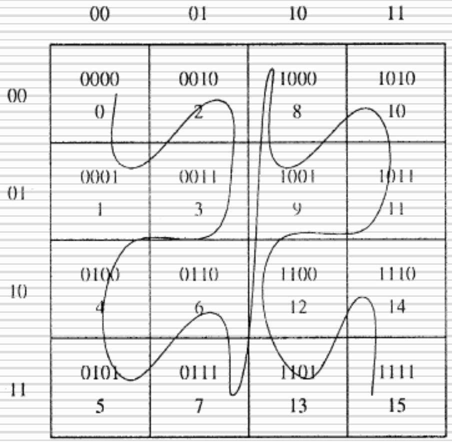






Morton Ordering

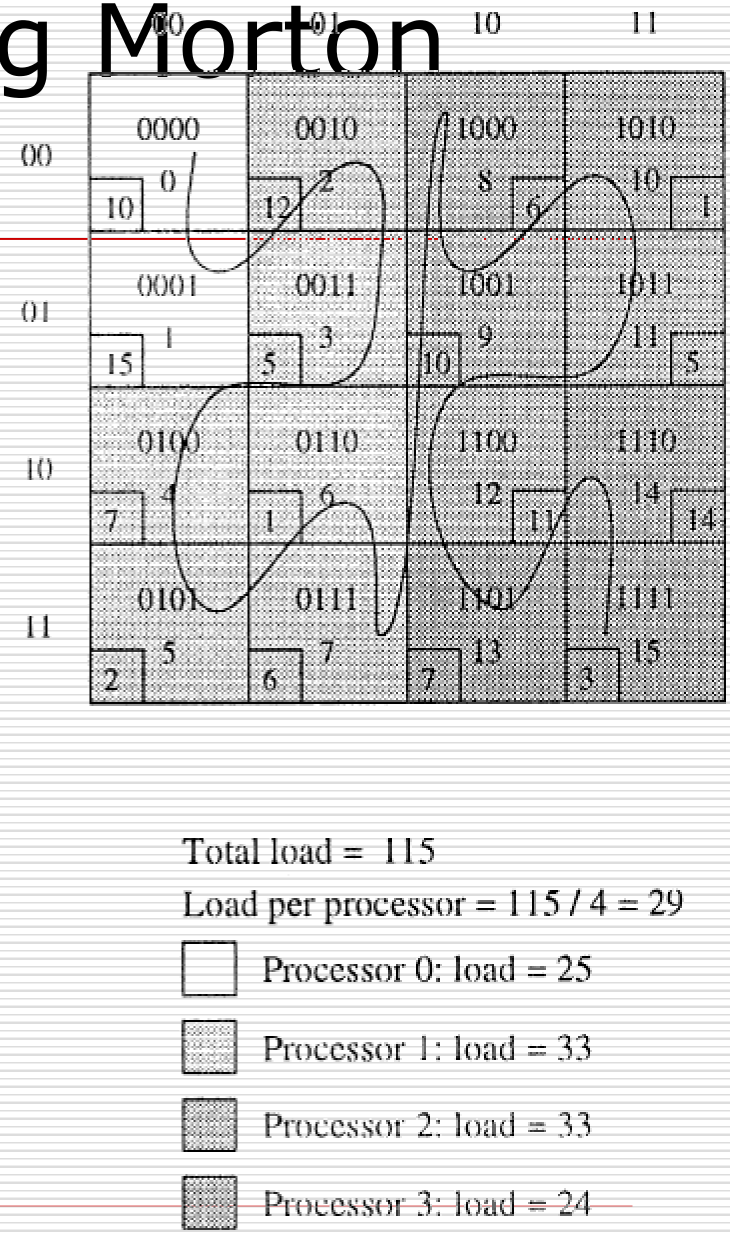
- Morton ordering is an ordering/numbering of the subdomains: the bits of the row and column are interleaved and the subdomains/clusters are labeled by Morton number



- The Morton ordered subdomains are located nearby each other mostly; when this ordered subdomains is partitioned across processors, nearby interacting particles/nodes are mapped to a single processor, thus optimizing communication

Load Balancing using Morton Ordering

- The ordered subdomains are stored in a sorted list
- After an iteration, a processor computes the load in each of its clusters; the load is entered into sorted list
- The load at each processor is added to form a global sum
- The global sum is divided by the number of processors, to form equal load
- The list is traversed and divided such that the loads in each division (processor) is approximately equal



Load Balancing using Morton Ordering

- Each processor compares the load in current iteration with the desired load in the next iteration
- If current load < desired load, clusters/subdomains are imported from next processor in Morton ordering
- If not, excess load clusters exported from the end of current list to next processor
- Done at end of each iteration

Graycode Curve

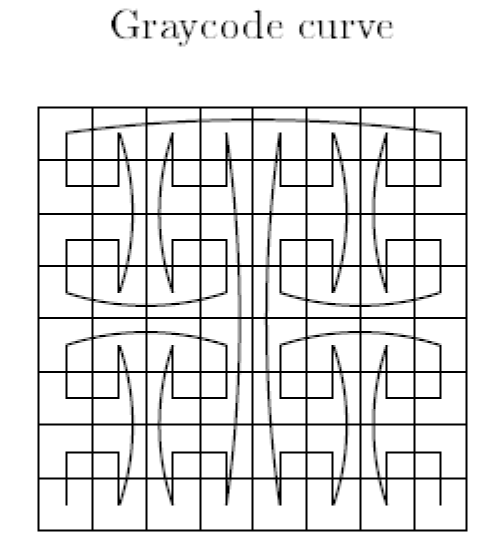
- Uses same interleaving function as Z-curve
- But visits points in the graycode order
- Graycode – two successive values differ in only one bit
- The one-bit gray code is (0,1)

Graycode Curve

- The gray code list for n bits can be generated recursively using n-1 bits
  - By reflecting the list (reversing the list) [1,0]
  - Concatenating original with the reflected [0,1,1,0]
  - Prefixing entries in the original list with 0, and prefixing entries in the reflected list with 1 [00,01,11,10]

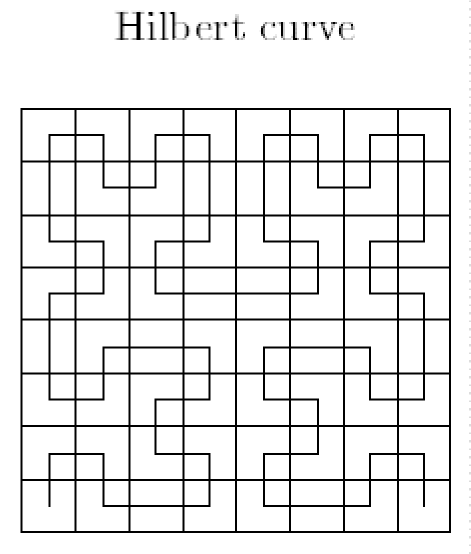
Graycode Curve

- 3-bit gray code: 000,001,011,010,110,111,101,100



Hilbert Curve

- Hilbert curve is a smooth curve that avoids the sudden jumps in Z-curve and graycode curve
- Curve composed of four curves of previous resolution in four quadrants
- Curve in the lower left quadrant rotated clockwise by 90 degree, and curve in lower right quadrant rotated anticlockwise by 90 degree



SFCs for AMR

- All these curve based partitioning techniques can also be applied for adaptive mesh by forming hierarchical SFCs