Mesh Generation and Load Balancing

Stan Tomov
Innovative Computing Laboratory
Computer Science Department
The University of Tennessee

April 4, 2018
Outline

• Motivation
  – Reliable & efficient PDE simulations for high end computing systems

• Background
  – PDE simulation concept: approximation is over a mesh

• Error Analysis
  – Simulation error: related to “local mesh size”

• Adaptive Mesh Generation
  – Support parallel refinement/derefinement and “element migration”

• Load Balancing
  – Scalability of the computation on modern architectures

• Data structures
  – Algorithmically motivated: multigrid, domain decomposition, etc.
  – For performance optimization: architecture aware computing

• Numerical Example

• Conclusions
Motivation

• PDE simulations have errors stemming from the numerical approximation (related to the mesh, ...)

• The need for
  • **Reliable**: “error” to be less than desirable tolerance
  and
  • **Efficient**: do not do “overkill” computation

PDE simulations for
  • **High end computing systems.**
Background

- In general: “Error” from the discretization is proportional to the mesh size
- A problem: localized physical phenomena deteriorate the approximation properties of classical PDE approximations
- How can we find a “good” mesh, i.e. yielding small and reliable error and efficient computation

For example flows near
- wells;
- faults;
- moving fronts, etc.
Background

- **Solution:** (1) determine (automatically) the regions of singular behaviour, and
  (2) refine them in a “balanced” manner

Example: **Efficiency** of locally adapted *vs* uniform approximation of
\[ r^{1/2} \sin(\theta/2) \] on an L-shaped domain
Background

• Computational framework of the Adaptive methods:

1. Solve PDE
2. Evaluate the approximation “error”
3. Is “error” acceptable?
   - yes: done
   - no: Improve the approximation (h/p refinement)

• i.e. a process of continuous feedback from the computation to find a reliable and efficient numerical PDE approximation
Error Analysis

• The numerical solution of PDE (e.g. FEM)
  – Boundary value problem: \( Au = f \), subject to boundary conditions
  – Get a “weak” formulation: \( (Au, \phi) = (f, \phi) \) - multiply by test function \( \phi \) and integrate over the domain

\[
a( u, \phi) = <f, \phi> \quad \text{for } \forall \phi \in S
\]

  – Galerkin (FEM) problem: Find \( u_h \in S_h \subset S \) s.t.

\[
a( u_h, \phi_h) = <f, \phi_h> \quad \text{for } \forall \phi_h \in S_h
\]

• The error \( e \equiv u - u_h \)
  – The Error problem: \( a( e, \phi) = a( u - u_h, \phi) = <A(u - u_h), \phi> \)

\[
= <f - Au_h, \phi> \equiv <R_h, \phi> \quad \text{for } \forall \phi \in S
\]
  – Various error estimators: depend on how we “solve” the Error problem
Adaptive Mesh Generation

• “good” mesh ~ “good” approximation
• Huge area of research and software development
  – See Steven Owen’s (Sandia) survey
    http://www.andrew.cmu.edu/user/sowen/mesh.html

• Which one to choose?
  – Classification: structured/unstructured, element type, support/or no adaptivity, sequential/parallel, etc.
  – Algorithm requirements: conforming/non-conforming, problem size, etc

• For HPC on 100s of 1000s of processors: parallel adaptive
  – Software design: framework (application is embedded) or toolkits (CCA interface compliant)
  – Important algorithmic issues to consider
    • “low bookkeeping” and storage overhead, easy “data transfer” between meshes, load balancing
Adaptive Mesh Generation

- **Mesh generation techniques**
  - Regenerate the mesh
    - locally or globally;
    - Appealing only for steady state problems;
    - produce meshes with particular properties (Delaunay/Voronoi, etc.).
  - **Hierarchical refinement** (common method of choice in AMR; used in *ParaGrid*)
    - keep hierarchy of meshes;
    - good for both steady & transient problems;
    - algorithms to maintain mesh quality.
  - Various hybrid methods
    - h-refinement with various local node movements (r-refinement/mesh smoothing);
    - various patch-grid refinement strategies;
    - techniques for coupling various grids, etc.
Adaptive Mesh Generation

• Hierarchical mesh generation
  – Element subdivision (e.g. tetrahedral edge bisection)
  – Hierarchy is usually stored in tree (e.g. quad/octrees in 2/3D)
    • Facilitate coarsening
    • Natural creation of multilevel data structures for multilevel solvers
    • Research on various formats/tricks to reduce storage overhead
      – Exploring the “deterministic” nature of refinement
        (relation of parent-child elements)
Load Balancing

• The need for load balance throughout the adaptive solution process
  – Minimize idle time + interprocessor comm. ~ scalability

• Partitioning for
  – Load balance, and
  – minimal interface

is NP-complete but there are many heuristics discovered,

See the survey

History of partitioning algorithms


- Cartesian nested dissection
- Multilevel partitioning (available in Metis, Jostle, Party)
- Space filling curve

Partitioning Quality

Computational Requirements

- Spectral (1992)
- Multilevel Spectral (1993)
- Kernighan-Lin (1970)
- Fiduccia-Mattheyses (1982)
- Space-filling Curves (1995)
- Coordinate/Inertial Bisection (1993)
- Levelized Nested Dissection (1973)
- Multilevel k-way Partitioning (1995, Karypis-Kumar)

Multilevel partitioning

- Initial Partitioning Phase: A bisection of the smallest graph is computed.
- Refinement Phase: The bisection is successively projected to the next finer graph. At each level a KL-FM type refinement algorithm is applied.
- Fast!
- High Quality!
Dynamic Load Balancing

• Issues to consider:
  – Load balance (what about DD with ≠ conditioned subdomain matrices?)
  – Minimize edge cut
  – Minimize data redistribution cost (most expensive)
    • Rebuild internal and shared data structures
    • What about balance for multilevel data structures?

• Two main techniques (ParMETIS supports both):
  – Diffusive (“diffuse” load among neighbors)
  – Global  (global repartition + smart remapping to minimize redistribution cost)

• Which one to choose?
  – See for example: R. Biswas, S. Das, D. Harvey, L. Oliker, Parallel Dynamic Load Balancing Strategies For Adaptive Irregular Applications
Data Structures

• Adaptive methods run at a fraction of the performance peak of cache-based machines:
  – This is due to irregular memory access patterns
    • because of their dynamic nature and unstructured sparse matrices produced
    • Can be improved but efficient parallel programming is difficult for this class of problems
  – A lot of current work in the field is on data structures
    • Improve memory access patterns for better cache reuse

(to be discussed further in Lecture #3 ... )
Data Structures

• In particular: for adaptive mesh generation
  – Need distributed tree (quadtree/octree) for efficient derefinement
  – Contiguous storage space + hash table access (performance)
  – Use the “deterministic” nature of the refinement/coarsening (minimize data storage)
Data Structures

• Algorithmically motivated
  – Multigrid
  – Domain decomposition

• For performance of parallel matrix-vector product
  – Pre-compute & block inter-processor communication patterns
  – Index ordering (SAW, space filling curves, Cuthill McKee, etc)

  and structures for sparse matrix storage for
  • register blocking, and
  • cache blocking

  see the Sparsity & BeBOP projects at Berkeley;
  Techniques: similar to blocking for dense matrices;
  architecture dependant (need processor-specific tuning)
• When does register/cache blocking work? (see R. Nishtala et.al., 2006; R. Vuduc 2003; Berkeley optimization group)

• Discontinuous Galerkin FEM is of great current interest
  – Mesh and nonzero matrix structures for approximation of order 1 and 3 (pictures from D.Darmofal, 2004; MIT; aerospace applications)
  – Naturally occurring dense blocks: open possibilities for various register and multiple level of cache blocking techniques

• Tuning:
  – Machine dependant
  – Performance can be surprising
  – Need for automatic machine-specific search (Vuduc, 2003)
    • register blocking for sparse 3-diagonal matrix consisting of 8x8 dense blocks (on Intel Itanium 2)
    • explored by storing them as 8x8 blocks
    • what about r x c block storage?
    • shown is the speedup relative to unblocked 1 x 1 code
Numerical Example

An example of contaminant flow in porous media:

http://www.cs.utk.edu/~tomov/cflow/
Conclusions

Adaptive methods:

- A computational methodology for **reliable** and **efficient** numerical solution of PDE problems
- Multidisciplinary field
  - CS, math, engineering
  - Need multidisciplinary effort for their successful development
  - Overview of the CS aspects
- The goal: develop the methodology and build it into an
  - intelligent
  - adaptable
  - reconfigurable system

for current and next generation supercomputers