Partitioning and Partitioning Tools

Tim Barth
NASA Ames Research Center
Moffett Field, California 94035-1000 USA
Graph/Mesh Partitioning

- Why do it?
- The graph bisection problem
- What are the standard heuristic algorithms?
- What tools are available?
Why do it?

Efficient utilization of distributed computational resources

- Equidistribution of workload among processors (load balancing)
  - Minimized time spend in interprocessor communication
    - Communication takes time and it's not always possible to hide this latency in data transfer
  - Cost of communication is often modeled by the linear relationship for $n$ messages:
    \[\text{Cost} = \text{P}n + \text{m}/n\]

Figure 1: (a) Mesh partitioning with minimized number of messages, (b) Mesh with minimized message length.
Why do it?

- Efficient utilization of distributed computational resources
Why do it?

- Efficient utilization of distributed computational resources
  - Equidistribution of workload among processors (load balancing)

![Figure 1: (a) Mesh partitioning with minimized number of messages, (b) Mesh with minimized message length.](3-b)
Why do it?

- Efficient utilization of distributed computational resources
  - Equidistribution of workload among processors (load balancing)
  - Minimized time spend in interprocessor communication
    * Communication takes time and it’s not always possible to hide this latency in data transfer

\[
\text{Cost} = P n^a (1 + m n^b)
\]
Why do it?

- Efficient utilization of distributed computational resources
  - Equidistribution of workload among processors (load balancing)
  - Minimized time spend in interprocessor communication
    * Communication takes time and it’s not always possible to hide this latency in data transfer
    * Cost of communication is often modeled by the linear relationship for $n$ messages: $Cost = \sum_n (\alpha + \beta m_n)$
Why do it?

- Efficient utilization of distributed computational resources
  - Equidistribution of workload among processors (load balancing)
  - Minimized time spend in interprocessor communication
    * Communication takes time and it’s not always possible to hide this latency in data transfer
    * Cost of communication is often modeled by the linear relationship for $n$ messages: $Cost = \sum_n (\alpha + \beta m_n)$

Figure 1: (a) Mesh partitioning with minimized number of messages, (b) Mesh with minimized message length.
Why do it?

As a strategy for reducing the overall arithmetic complexity of an algorithm:

- Overlapping Schwarz methods
  - "Divide and Conquer" methods, e.g. nested dissection of matrix, Schur complement substructuring

- Multiscale methods, e.g. agglomeration multigrid
Why do it?

- As a strategy for reducing the overall arithmetic complexity of an algorithm
Why do it?

- As a strategy for reducing the overall arithmetic complexity of an algorithm
  - Overlapping Schwarz methods
Why do it?

- As a strategy for reducing the overall arithmetic complexity of an algorithm
  - Overlapping Schwarz methods
  - “Divide and Conquer” methods, e.g. nested dissection of matrix, Schur complement substructuring
Why do it?

- As a strategy for reducing the overall arithmetic complexity of an algorithm
  - Overlapping Schwarz methods
  - “Divide and Conquer” methods, e.g. nested dissection of matrix, Schur complement substructuring
  - Multiscale methods, e.g. agglomeration multigrid
Why do it?

- Overlapping Schwarz methods
- Overlapping Schwarz methods with subdomain size $H$, mesh cell size $h$ and overlap $\delta$

Let $A$ be the discretization matrix and $M_{as}$ the additive Schwarz preconditioner. There exists a constant $C$ independent of $H$ and $h$ such that the condition number $\kappa$

$$\kappa(M_{as}^{-1}A) \leq CH^{-2} \left(1 + \left(\frac{H}{\delta}\right)^2\right).$$  \hspace{1cm} (1)

with 2-level coarse space correction

There exists a constant $C$ independent of $H$ and $h$ such that

$$\kappa(M_{as}^{-1}A) \leq C \left(1 + \left(\frac{H}{\delta}\right)^2\right).$$ \hspace{1cm} (2)
Why do it?

- Substructuring

\[
\begin{pmatrix}
A_1 & A_2 \\
A_3 & A_4
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
b_2
\end{pmatrix},
\quad \quad A^{-1} =
\begin{pmatrix}
C_1 & C_2 \\
C_3 & C_4
\end{pmatrix}
\]

with
\[
S = A_4 - A_3 A_1^{-1} A_2, \quad C_1 = A_1^{-1} + A_1^{-1} A_2 S^{-1} A_3 A_1^{-1},
\]
\[
C_2 = -A_1^{-1} A_2 S^{-1}, \quad C_3 = -S^{-1} A_3 A_1^{-1}, \quad C_4 = S^{-1}.
\]

\[
\kappa (M^{-1}_{\text{Schur}} A) = C (1 + \log (H/\delta))
\]
Define a partitioning vector $p \in \mathbb{Z}^n$ which 2-colors the vertices of a graph

$$p = [+1, -1, -1, +1, +1, \ldots, +1, -1]^T$$

(3)

- Minimize the cut-weight of the weighted graph
- Produce balanced partitions
Heuristic Graph Partitioning

Three commonly used partitioning techniques

- Recursive coordinate bisection
- Recursive Cuthill-McKee
- Recursive Spectral bisection
Recursive Coordinate Bisection

- Spatial coordinates are sorted along alternating horizontal and vertical directions
- Divisors are found to balance partitions
Graph Ordering Cuthill-McKee

Algorithm: Graph ordering, Cuthill-McKee.

Step 1. Find vertex with lowest degree. This is the root vertex.

Step 2. Find all neighboring vertices connecting to the root by incident edges. Order them by increasing vertex degree. This forms level 1.

Step 3. Form level $k$ by finding all neighboring vertices of level $k - 1$ which have not been previously ordered. Order these new vertices by increasing vertex degree.

Step 4. If vertices remain, go to step 3.
Graph Ordering Cuthill-McKee

Matrix nonzero pattern

Figure 2: Natural Ordering (left) and Cuthill-McKee ordering (right)
• The level structure computed in Cuthill-McKee ordering is utilized

• Divisors are found to balance partitions
Motivated by the observation that the cut-weight of a graph is precisely
\[ W_c = \frac{1}{4} p^T \mathcal{L} p \]

**Algorithm:** Spectral Graph Bisection.

*Step 1.* Calculate the matrix \( \mathcal{L} \) associated with the Laplacian of the graph.

*Step 2.* Calculate the eigenvalues and eigenvectors of \( \mathcal{L} \).

*Step 3.* Order the eigenvalues by magnitude, \( \lambda_1 \leq \lambda_2 \leq \lambda_3 \ldots \lambda_n \).

*Step 4.* Determine the smallest nonzero eigenvalue, \( \lambda_f \) and its associated eigenvector \( \mathbf{x}_f \) (the Fiedler vector).

*Step 5.* Sort elements of the Fiedler vector.

*Step 6.* Choose a divisor at the median of the sorted list and 2-color vertices of the graph which correspond to elements of the Fielder vector less than or greater than the median value.
Recursive Spectral Bisection
Multilevel $k$-way Partitioning

- Utilized successive $k$-way graph contraction to coarsen graph
- Perform high quality partitioning on coarsened graph
- Prolongate to finer graphs with local interface optimization to improve cut-weight
ParMETIS is an METIS-based parallel library that implements a variety of algorithms for partitioning unstructured graphs, meshes, and for computing fill-reducing orderings of sparse matrices. ParMETIS extends the functionality provided by METIS and includes routines that are especially suited for parallel AMR computations and large scale numerical simulations. The algorithms implemented in ParMETIS are based on the parallel multilevel k-way graph-partitioning algorithms described in [KK95a], [KK96], [KK97], the adaptive repartitioning algorithms described in [SU96], and the parallel multi-constrained algorithms described in [SK98].

ParMETIS provides the following five major functions:

**Graph Partitioning**
- Computes high quality partitionings of very large graphs quickly.
- Takes advantage of geometry information (when available) to reduce the partitioning time without loss in quality.
- Can partition graphs for multi-phase and multi-physics computations.

**Mesh Partitioning**
- Computes high quality partitionings of very large meshes directly, without requiring the application to create the underlying graph.
- Provides highly efficient parallel routines for generating the dual graph of a mesh.

**Graph Repartitioning**
- Computes high quality repartitions of adaptively refined meshes quickly.
- Optimizes both the number of vertices that are moved as well as the edge-cut of the resulting partitioning.

**Partitioning Refinement**
- Improves the quality of partitions produced by other partitioning algorithms.

**Matrix Reordering**
- Computes fill-reducing orderings of sparse matrices.
- Uses a node-based nested dissection algorithm that has been shown to significantly outperform other popular reordering algorithms.
Metis, ParMetis

- Extremely fast
- Parallel implementation (requires some initial partitioning)
- Supports weighted graphs by vertices or edges
- Supports incremental load balancing (repartitioning) with minimized data migration
JOSTLE - mesh partitioning software

JOSTLE is a software package designed to partition unstructured meshes (for example, finite element or finite volume meshes) for use on distributed memory parallel computers. It can also be used to repartition existing partitions (such as those deriving from adaptive refined meshes).

The code is extremely fast and provides high quality multilevel partitioning and diffusive load-balancing in both serial & parallel.

serial jostle executable

Executables of the software are freely available for academic and research purposes, but interested users are required to sign a licence agreement.

- download licence for standalone serial jostle (version 3.0 - July 2002)

To apply for a copy of the software please post or fax the form to the address below.

The software is in principle available on any platform with a C compiler. Currently supported platforms include:

- Silicon Graphics Workstations
- DEC Alpha Workstations
- Sun/Solaris Workstations
- Linux PCs

The purpose of the licence is to prevent resale or commercial exploitation but we strongly wish to encourage the use of the software and would be keen to work on collaborations, etc. We are also interested in supporting commercial exploitation, although under different licencing arrangements. Send the completed licence to me and I will mail you a copy of the package.

jostle serial & parallel libraries

The libraries for the serial & parallel versions are also available on an evaluation basis for academic and research purposes:

- download licence for jostle serial & parallel library (version 3.0 - July 2002)

To apply for a copy of the software please post or fax the form to the address below.
Zoltan

- Relatively new package under development at Sandia under GPL
- Interfaces with Metis or Jostle
- Documentation suggests that the package will contain most of the commonly needed services for parallel scientific codes: partitioning, repartitioning, data migration, etc.
Partitioning Tools for SSS?

- Domain specific languages?
  - Language for finite element methods
  - Language for molecular dynamics
  - <Insert your favorite problem domain here>

- Partial or full data dependency specification (analogous to scene graph specification in Java3d).

- Automatic tools for performance enhancement
  - Use hardware performance statistics (memory access patterns) of previous executions in subsequence compilations
  - Runtime data migration