Example: Laplace Equation in 1-D

Finite difference approximation

\[ u''(t_i) \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \]

yields tridiagonal system of algebraic equations

\[ u_{i+1} - 2u_i + u_{i-1} = 0, \quad i = 1, \ldots, n, \]

for \( u_i \), \( i = 1, \ldots, n \), where \( u_0 = \alpha \) and \( u_{n+1} = \beta \)

Starting from initial guess \( u^{(0)} \), compute Jacobi iterates

\[ u_i^{(k+1)} = \frac{u_i^{(k)} + u_{i+1}^{(k)}}{2}, \quad i = 1, \ldots, n, \]

for \( k = 1, \ldots \) until convergence

Example: Laplace Equation in 1-D

Map the problem onto a distributed-memory multicomputer using message passing.

Tasks must be assigned to physical processors for execution.

Tasks can be mapped to processors in various ways, including multiple tasks per processor.

Semantics of program should not depend on number of processors or particular mapping of tasks to processors.

Performance usually sensitive to assignment of tasks to processors due to concurrency, workload balance, communication patterns, etc.

Computational model maps naturally onto distributed memory multicomputer using message passing.

Tasks can be mapped to processors in various ways, including multiple tasks per processor.

Program should not depend on number of processors or particular mapping of tasks to processors.

Performance usually sensitive to assignment of tasks to processors due to concurrency, workload balance, communication patterns, etc.

Computational model maps naturally onto distributed memory multicomputer using message passing.
Other Models of Parallel Computation

- PRAM — Parallel Random Access Machine
- LogP — Latency/Overhead/Gap/Processors
- BSP — Bulk Synchronous Parallel
- CSP — Communicating Sequential Processes
- Linda — Tuple Space
- and many others

Four-Step Design Methodology

- **Partition**: Decompose problem into fine-grain tasks, maximizing number of tasks that can execute concurrently
- **Communicate**: Determine communication pattern among fine-grain tasks, yielding task graph with fine-grain tasks as nodes and communication channels as edges
- **Agglomerate**: Combine groups of fine-grain tasks to form fewer but larger coarse-grain tasks, thereby reducing communication requirements
- **Map**: Assign coarse-grain tasks to processors, subject to tradeoffs between communication costs and concurrency

Graph Embeddings

- Target network may be virtual network topology, with nodes usually called processes rather than processors
- Overall design methodology is composed of sequence of graph embeddings:
  - fine-grain task graph to coarse-grain task graph
  - coarse-grain task graph to virtual network graph
  - virtual network graph to physical network graph
- Depending on circumstances, one or more of these embeddings may be skipped
- Target system may automatically map processes of virtual network topology to processors of physical network

Desirable Properties of Partitioning

- Maximum possible concurrency in executing resulting tasks
- Many more tasks than processors
- Number of tasks, rather than size of each task, grows as overall problem size increases
- Tasks reasonably uniform in size
- Redundant computation or storage avoided

Communication Patterns

- Communication pattern determined by data dependences among tasks: because storage is local to each task, any data stored or produced by one task and needed by another must be communicated between them
- Communication pattern may be
  - local or global
  - structured or random
  - persistent or dynamically changing
  - synchronous or sporadic

Example: Domain Decomposition

3-D domain partitioned along one (left), two (center), or all three (right) of its dimensions
With 1-D or 2-D partitioning, minimum task size grows with problem size, but not with 3-D partitioning
Desirable Properties of Communication

- Frequency and volume minimized
- Highly localized (between neighboring tasks)
- Reasonably uniform across channels
- Network resources used concurrently
- Does not inhibit concurrency of tasks
- Overlapped with computation as much as possible

Example: Laplace Equation in 1-D

- Combine groups of consecutive mesh points \( i \) and corresponding solution values \( u_i \) into coarse-grain tasks, yielding \( p \) tasks, each with \( u_i/p \) of \( u_i \) values

Communication is greatly reduced, but \( u_i \) values within each coarse-grain task must be updated sequentially

Overlapping Communication and Computation

- Updating of solution values \( u \) is done only after all communication has been completed, but only two of those values actually depend on awaited data
- Since communication is often much slower than computation, initiate communication by sending all messages first, then update all “interior” values while awaiting values from neighboring tasks
- Much (possibly all) of updating can be done while task would otherwise be idle awaiting messages
- Performance can often be enhanced by overlapping communication and computation in this manner

Surface-to-Volume Effect

- For domain decomposition, computation is proportional to volume of subdomain
- Communication is (roughly) proportional to surface area of subdomain
- Higher-dimensional decompositions have more favorable surface-to-volume ratio
- Partitioning across more dimensions yields more neighboring subdomains but smaller total volume of communication than partitioning across fewer dimensions

Agglomeration

- Increasing task sizes can reduce communication but also reduces potential concurrency
- Subtasks that can’t be executed concurrently anyway are obvious candidates for combining into single task
- Maintaining balanced workload still important
- Replicating computation can eliminate communication and is advantageous if result is cheaper to compute than to communicate

Example: Laplace Equation in 1-D

initialize \( u_1, \ldots, u_r \)

for \( k = 1, \ldots \)

if \( j > 1 \), send \( u_j \) to task \( j - 1 \) \{ send to left neighbor \}

if \( j < p \), send \( u_j \) to task \( j + 1 \) \{ send to right neighbor \}

if \( j < p \), recv \( u_{r+1} \) from task \( j + 1 \) \{ receive from right neighbor \}

if \( j > 1 \), recv \( u_{r-1} \) from task \( j - 1 \) \{ receive from left neighbor \}

for \( i = l \) to \( r \)

\( \bar{u}_i = (u_{i-1} + u_{i+1})/2 \) \{ update local values \}

end

wait for sends to complete

\( u = \bar{u} \)

eend

Mapping

- As with agglomeration, mapping of coarse-grain tasks to processors should maximize concurrency, minimize communication, maintain good workload balance, etc
- But connectivity of coarse-grain task graph is inherited from that of fine-grain task graph, whereas connectivity of target interconnection network is independent of problem
- Communication channels between tasks may or may not correspond to physical connections in underlying interconnection network between processors
Two communicating tasks can be assigned to
- one processor, avoiding interprocessor communication but sacrificing concurrency
- two adjacent processors, so communication between the tasks is directly supported, or
- two nonadjacent processors, so message routing is required

In general, finding optimal solution to these tradeoffs is NP-complete, so heuristics are used to find effective compromise

For problems with regular structure, it is often possible to determine mapping in advance that yields reasonable load balance:

- Whenever executing task becomes idle awaiting data, another task is executed
- Dynamic load balancing usually based on local exchanges of workload information (and tasks, if necessary)

If communication is mainly global, then communication performance may not be sensitive to placement of tasks on processors, so random mapping may be as good as any

Random mappings sometimes used deliberately to avoid communication hot spots, where some communication links are oversubscribed with message traffic

With multiple tasks per processor, execution of those tasks must be scheduled over time:

- For shared-memory, any idle processor can simply select next ready task from common pool of tasks
- For distributed-memory, analogous approach is manager/worker paradigm, with manager dispatching tasks to workers
- Manager/worker scales poorly, as manager becomes bottleneck, so hierarchy of managers and workers becomes necessary, or more decentralized scheme

Example: Atmospheric Flow Model

- Fluid dynamics of atmosphere modeled by system of partial differential equations
- 3-D problem domain discretized by \( n_x \times n_y \times n_z \) mesh of points
- Vertical dimension (altitude) \( z \), much smaller than horizontal dimensions (latitude and longitude) \( x \) and \( y \), so \( n_z \ll n_x, n_y \)
- Derivatives in PDEs approximated by finite differences
- Simulation proceeds through successive discrete steps in time
**Example: Atmospheric Flow Model**

**Partition:**
- Each fine-grain task computes and stores data values (pressure, temperature, etc) for one mesh point
- Typical mesh size yields $10^5$ to $10^7$ fine-grain tasks

**Communicate:**
- Finite difference computations at each mesh point use 9-point horizontal stencil and 3-point vertical stencil
- Solar radiation computations require communication throughout each vertical column of mesh points
- Global communication to compute total mass of air over domain

**Agglomerate:**
- Combine horizontal mesh points in blocks of four into coarse-grain tasks to reduce communication for finite differences to exchanges between adjacent nodes
- Combine each vertical column of mesh points into single task to eliminate communication for solar computations
- Yields $n_x \times n_y / 4$ coarse-grain tasks, about $10^3$ to $10^5$ for typical mesh size

**Map:**
- Cyclic or random mapping reduces load imbalance due to solar computations

**Horizontal finite difference stencil for typical point (shaded black) in mesh for atmospheric flow model before (left) and after (right) agglomeration**

**References**
- I. T. Foster, *Designing and Building Parallel Programs*, Addison-Wesley, 1995
- T. G. Mattson and B. A. Sanders and B. L. Massingill, *Patterns for Parallel Programming*, Addison-Wesley, 2005