Outline

• Applications

• Creating Parallel Programs

• Programming for Performance

• Scaling

• Synchronization Basics
Programming for Performance

- Partitioning, Granularity, Communication, etc.

- Caches and Their Effects
Aside on Cost-Effective Computing

- Isn’t Speedup(P) < P inefficient?
- If only throughput matters, use P computers instead?

- But much of a computer’s cost is NOT in the processor [Wood & Hill, *IEEE Computer*, Feb 95]
- Let Costup(P) = Cost(P)/Cost(1)
- Parallel computing is cost-effective if
  \[ \text{Speedup(P)} > \text{Costup(P)} \]
- E.g., for SGI PowerChallenge w/ 500MB:
  \[ \text{Costup(32)} = 8.6 \]
Where Do Programs Spend Time?

• Sequential
  – Busy computing
  – Memory system stalls

• Parallel
  – Busy computing
  – Stalled for local memory
  – Stalled for remote memory (communication)
  – Synchronizing (load imbalance and operations)
  – Overhead

• Speedup \( (p) = \frac{\text{time}(1)}{\text{time}(p)} \)
  – Amdahl’s Law
  – Could even be superlinear → how??
Partitioning for Performance

- **Balance workload**
  - Reduce time spent at synchronization

- **Reduce communication**

- **Reduce extra work**
  - Determining and managing good assignment

- **These goals are at odds with each other**
Programming for Performance

- Identifying concurrency

- Managing concurrency
  - Static
  - Dynamic

- Granularity of concurrency

- Serialization and synchronization costs
Identifying Concurrency

- **Data parallelism**
  - Same ops on different data items

- **Functional (control, task) parallelism**
  - Pipeline

- Impact on load balancing?

- Functional is more difficult
  - Longer running tasks
Managing Concurrency

• **Static**
  – Cannot adapt to changes

• **Dynamic**
  – Can adapt
  – Cost of management increases
  – Self-scheduling (guided self-scheduling)
  – Centralized task queue
    » **Contention**
  – Distributed task queue
    » Can steal from other queues
    » **Architecture: Name data associated with stolen task**
Granularity of Concurrency

- Granularity = Amount of work associated with task

- Large tasks
  - Worse load balancing
  + Lower overhead
  + Less contention
  + Less communication

- Small tasks
  + Better load balancing
  - More synchronization
  - More management overhead
  - Might have too much communication (affinity scheduling)
Impact of Synchronization and Serialization

- Too coarse synchronization
  - Barriers instead of point-to-point synch
  - Poor load balancing

- Too many synchronization operations
  - Lock each element of array
  - Costly operations

- Coarse grain locking
  - Lock entire array
  - Serialize access to array

- Architectural aspects
  - Cost of synchronization operation
  - Synchronization name space
Architectural Support for Dynamic Task Stealing

• How can architecture help?
  • Communication
    – Support for transfer of small amount of data and mutual exclusion
    – Can make tasks smaller
    – Better load balance
  • Naming
    – Make it easy to name data associated with stolen task
  • Synchronization
    – Support point-to-point synchronization
    – Better load balancing
Reducing Inherent Communication

- Communication required for parallel program
- Communication to Computation Ratio
  - \(\text{Bytes} / \text{time}\) or \(\text{Bytes} / \text{instruction}\)
- Affected by assignment (task → process)
- Domain decomposition
  - Interact with neighbors in space
  - Good for simulation of physical

(C) 2010 Daniel J. Sorin from Adve, Falsafi, Hill, Lebeck, Reinhardt, Singh