CS6560: Parallel Computer Architecture

Programming for Performance

Madhu Mutyam
PACE Laboratory
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Feb 13 - 20, 2018

Partitioning For Performance

$$Speedup(p) \leq \frac{Sequential\ Work}{max_p(Work + Synch\ Wait\ Time + Comm\ Cost + Extra\ Work)}$$

- Balance the workload
- Reduce communication
- Reduce the extra work done to determine and manage a good assignment

Load Balance and Synchronization

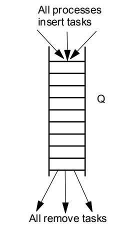
- Identify enough number of tasks
- Managing Tasks
- Reducing serialization and synchronization cost

Identifying Enough Number of Tasks

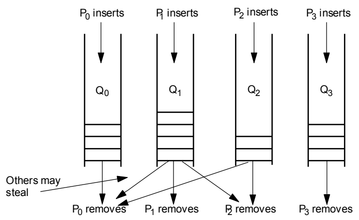
- Data parallelism
  - Similar functions executed on elements of a large data structure
  - Grows with data set size
- Function parallelism
  - Different functions executed concurrently on same or different data
  - Does not grow with the size of the problem

Managing Tasks

- Static assignment of tasks
  - Load may not be balanced
  - Low task management overhead
- Dynamic assignment of tasks
  - Load balancing at runtime
  - Semi-static assignment
  - Dynamic tasking
  - High task management overhead



(a) Centralized task queue



(b) Distributed task queues (one per process)

Reducing Serialization

- Careful about assignment, orchestration, and scheduling of tasks
- Event synchronization
  - Reduce the use of conservative synchronization
  - Fine-grained synchronization is more complex to program
    - Execution of more synchronization operations
- Mutual exclusion
  - Separate locks for separate data items
    - Ex:- lock per task in task queue, not per queue
    - fine grained $\Rightarrow$ low contension, but more space and less reuse
  - Stagger critical sections in time
  - Make critical sections smaller and less frequent
    - do not do reading/testing in critical section, only modifications

Architecture Implications of Load Balance

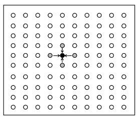
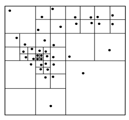
- Easy to name and access the logically shared data
- Efficient fine-grained communication
- Efficient point-to-point synchronization

Reducing Inherent Communication

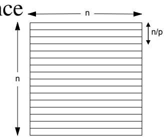
- Measure the impact of communication through $\frac{Communication}{Computation}$
- Inherent communication
  - One process produces data values that another process needs
  - Determined by the assignment of tasks to processes
- Assign tasks that access the same data or requiring frequent communication to the same process

Domain Decomposition

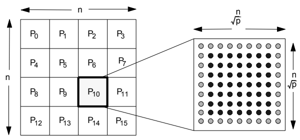
- Works well for load balancing and inherent communication
- Domain refers to the data set on which the application operates
- A point in the domain requires information from
  - only a small localized region, or
  - a longer region, but falls off with distance

- Give every process a contiguous region of the domain, while maintaining load balance



Strip decomposition



Block decomposition

- The comm.-to-computation ratio is a perimeter-to-area ratio in 2D
  - Decreases with n and increases with p

Finding Suitable Domain Decomposition

- Statically, by inspection
- Statically, by analysis
- Semi-Statically, with periodic repartioning
- Statically or Semi-Statically, with dynamic task stealing

Implications of Communication-to-Computation Ratio

- Architects examine the needs of applications to see where to spend effort
  - Bandwidth requirements (operations/second)
  - Latency requirements (seconds/operation)
- Need to keep communication balanced across processors

Reducing the Extra Work

- Common sources of extra work:
  - Computing a good partition
  - Using redundant computation to avoid communication
  - Aspects of orchestrating parallel programs
- Architecture can help reduce the need for extra work by making communication and task management more efficient

Data Access and Communication in Multi-Memory System

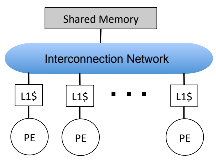
- Multiprocessor system can be viewed as
  - a collection of cooperating processors
    - Goals: balancing load, reducing inherent communication and extra work
  - a multi-cache, multi-memory system
    - Effect the performance irrespective of programming model

Performance in Uniprocessor Systems

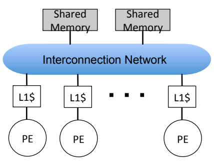
$$Time_{program}(1) = Busy(1) + Data\ Access(1)$$

- Reducing the data access cost
  - Optimize the program: data locality
  - Optimize the machine: bigger caches, latency tolerant techniques

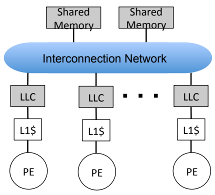
Multiprocessor As An Extended Memory Hierarchy



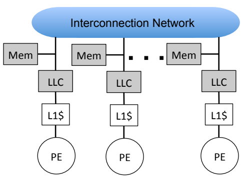
Ideal view



Multi-banked Shared Memory



Shared Caches on Other Processors



Distributed Shared Memory

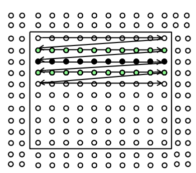
- Memory closer to the processor provides high bandwidth and low latency
- Data movement is transparent in shared memory model while it is explicit in message passing model
- Improve the architecture or the data locality to improve data access performance

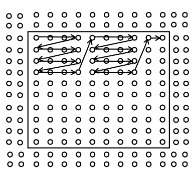
Artifactual Communication in the Extended Memory Hierarchy

- Sources of artifactual communication:
  - Poor allocation of data
  - Unnecessary data in a transfer
  - Unnecessary transfers of data
  - Redundant communication of data
  - Finite replication capacity
    - Cold, capacity, conflict, and communication misses at cache
    - Communication misses do not diminish with cache size

Exploit Temporal Locality to Reduce Artifactual Communication

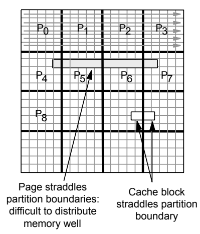
- Structure an algorithm to map working sets well
  - Programmer can keep the working sets small
  - Techniques that reduce inherent communication can reduce working sets
    - Assign tasks that tend to access same data to the same process
  - When multiple data structures are accessed in the same phase of a computation:
    - Prefer temporal locality on non-local data rather than local data
  - Solver example: *blocking* to exploit temporal locality


(a) Unblocked access pattern in a sweep

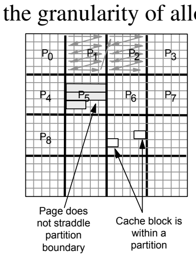

(b) Blocked access pattern with B = 4

Exploit Spatial Locality to Reduce Artifactual Communication

- Spatial locality is affected by
  - Granularity of data transfer
  - Granularity of allocation
  - Granularity of coherence
- Keep the data accessed by a processor close together (contiguous) in the address space and data accessed by different processors apart
- Restructure the data to interact better with the granularity of allocation



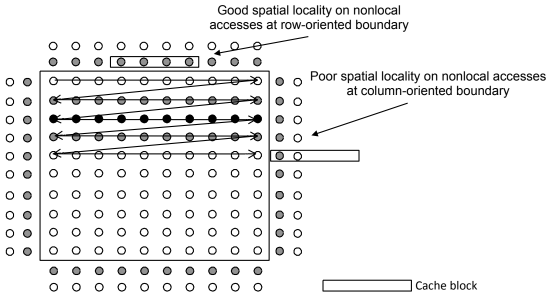
(a) Two-dimensional array



(b) Four-dimensional array

Trade-offs with Inherent Communication

- Block vs strip decomposition



Cost of Communication

$$C = Freq \times (Overhead + Delay + \frac{Length}{Bandwidth} + Contention - Overlap)$$

Reducing Overhead

- Fewer but larger messages
  - Easy in explicitly initiated communication
  - Coalesce reads and writes into larger messages in shared memory model
- Making larger messages is easy in applications that have regular data access and communication patterns
- Trade-off between extra work to determine which data to coalesce and savings in overhead

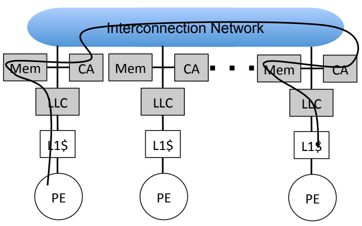
Reducing Delay

- Optimize the communication assist and network interface hardware components

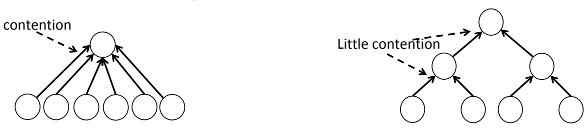
$$Network\ transit\ delay = freq \times (hop\ count) \times (hop\ latency)$$

- Map processes to processors in a way to exploit the network topology for reducing the hop count

Reducing Contention



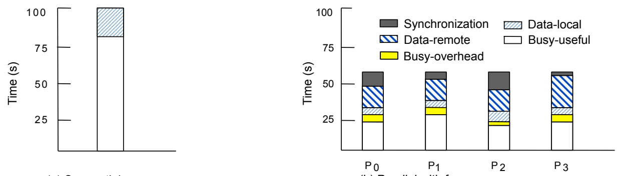
- Latency to serve a request increases with the number of contenders
- Stall time across processors can lead to synchronization wait times
- Use topology-aware process mapping and communication scheduling to reduce network contention
- Reduce contention at the processing elements



Overlapping Communication and Computation

- Prefetching – Initiate communication well before the data is actually needed
- Initiate some other useful computation or communication
- Multithreading – Switch to a different thread when a communication event is encountered

Performance Factors From the Processor's Perspective



(a) Sequential

(b) Parallel with four processors

$$Speedup(p) \quad = \quad \frac{Busy(1) + Data_{local}(1)}{Busy_{useful}(p) + Data_{local}(p) + Synch(p) + Data_{remote}(p) + Busy_{over}(p)}$$

Summary

- It is important to understand the parallel programs and workloads that we run on the systems
- Programming for performance is a process of successive refinement
- Key points for performance improvement:
  - Load balancing
  - Communication cost
  - Data locality and its interaction with replication capacity

Thank You