Parallel Programming for Performance

- What limits the performance of parallel programs
- What factors must be addressed in software?
- What factors can be addressed by the architecture?

Performance Goal => Speedup

- Architect Goal
  - observe how program uses machine and improve the design to enhance performance
- Programmer Goal
  - observe how the program uses the machine and improve the implementation to enhance performance
- What do you observe?
- Who fixes what?
What limits Performance?

- Available Parallelism
- Load Balance
  - Some processors do more work than others
  - Some work while others wait
  - Remote resource contention
- Communication
- Extra Work
  - Management of parallelism
  - Redundant computation

Parallelism in LU

```
procedure LU(A[n, n])
begin
    int i, j, k, p, d;
    int rows, mymin, mymax;
    rows ← n / procs;
    mymin ← pid * rows; mymax ← mymin + rows - 1;
    for k ← 0 to n-1 do
        if (mymin ≤ k ≤ mymax)
            for j ← k+1 to n-1 do
            endif
        BARRIER(bar1, nprocs);
    for i ← max(k+1, mymin) to mymax do
        for j ← k+1 to n-1 do
        endfor
    endfor
end
```
Parallelism with Row-wise Decomp.

- Step K
  - Serial: n-k to scale pivot row
  - Parallel: n-k rows of n-k steps

Speedup due to parallel computing

- Suppose only part of an application seems parallel
- Amdahl’s law
  - Let f be the fraction of work done sequentially, so (1-f) is fraction parallelizable.
  - n = number of processors.

\[
\text{Speedup}(n) = \frac{\text{Time}(1)}{\text{Time}(n)} \\
\leq \frac{1}{f + (1-f)/n} \\
\leq \frac{1}{f}
\]

Efficiency = Speedup/n

Even if the parallel part speeds up perfectly, we may be limited by the sequential portion of code.
Amdahl’s Law Implications

Generalizing Amdahl’s Law

- Parallelism Profile: \( PP(t) = \text{number of ops at step } t \)
- \( \text{Speedup}(p) = \frac{T(1)}{T(p)} \leq \frac{\sum pp(t)}{\sum [pp(t)/p]} \)
Gustafson’s Law

- Parallel processing is to solve larger programs in a fixed time
- Let $s$ be the serial execution time, and $p$ the execution time in parallel
- Scaled Speedup Factor:
  \[ S_s(n) = \frac{s + np}{s + p} = s + np = n + (1 - n)s \]

Suppose a serial section of 5% and 20 processors
  According to Amdahl’s law, the speedup is 10.26
  According to Gustafson’s law, the speedup is 19.05

Avoiding Serialization

- Careful about assignment and orchestration
- Event synchronization
  - Reduce use of conservative synchronization
    • e.g. point-to-point instead of barriers, or granularity of pt-to-pt
  - But fine-grained synch more difficult to program, more synch ops.
- Mutual exclusion
  • Separate locks for separate data
    - e.g. locking records in a database: lock per process, record, or field
    - lock per task in task queue, not per queue
    - finer grain => less contention/serialization, more space, less reuse
  • Smaller, less frequent critical sections
    - don’t do reading/testing in critical section, only modification
  - Stagger critical sections in time
Load Balance

- Each processor should do same amount of work

\[
\text{Speedup}_{\text{problem}}(p) \leq \frac{\text{Sequential Work}}{\text{Max Work on any Processor}}
\]

- Instantaneous load imbalance revealed at wait time
  - At completion; At barriers; At receive; At flags, even at mutex
- Cost of load imbalance = wait time

![Diagram showing load balance](image)

Sequential Work
\[
\text{Max (Work + Synch Wait Time)}
\]

Improving Load Balance

- Decompose into more smaller tasks (>>P)
- Distribute uniformly
  - variable sized task
  - randomize
  - bin packing
  - dynamic assignment
- Schedule more carefully
  - avoid serialization
  - estimate work
  - etc
- Ocean Simulation??
Load imbalance in LU

- Block row vs Cyclic row

Example: Barnes-Hut

- Divide space into roughly equal # particles
- Particles close together in space should be on same processor
- Nonuniform, dynamically changing
Dynamic Scheduling with Task Queues

- Centralized versus distributed queues
- Task stealing with distributed queues
  - Can compromise comm and locality, and increase synchronization
  - Whom to steal from, how many tasks to steal, ...
  - Termination detection
  - Maximum imbalance related to size of task

(a) Centralized task queue
(b) Distributed task queues (one per process)

Impact of Dynamic Assignment

- Barnes-Hut on SGI Origin 2000 (cache-coherent shared memory):
Communication

- Comm operation is expensive!!
  - Measure: communication to computation ratio

\[
\text{Speedup} \leq \frac{\text{Sequential Work}}{\max(\text{Work} + \text{Synch Wait Time} + \text{Comm Cost})}
\]

- Two Basic Models

  ![Extended Storage Hierarchy](image1)
  ![Distributed Address Space](image2)

Domain Decomposition

- Works well for scientific, engineering, graphics, ... applications
- Exploits local-biased nature of physical problems
  - Information requirements often short-range
  - Or long-range but fall off with distance
- Simple example: nearest-neighbor grid computation

Perimeter to Area comm-to-comp ratio (area to volume in 3-d)
- Depends on \( n,p \): decreases with \( n \), increases with \( p \)
Domain Decomposition (contd)

Best domain decomposition depends on information requirements

Nearest neighbor example: block versus strip decomposition:

- Comm to comp: \( \frac{n}{\sqrt{p}} \) for block, \( \frac{2\sqrt{n}}{p} \) for strip
- Application dependent: strip may be better in other cases
  - E.g. particle flow in tunnel

Comm. in an Extended Hierarchy

- All shared accesses are expensive
- First access since last update is expensive

```plaintext
procedure LU( A[n,n] )
begin
  for k <- 0 to n - 1 do
    for all j <- k+1 to n-1 do
    BARRIER(bar1, nprocs);
    for all i <- k+1 to n-1 by rows
      for j <- k+1 to n-1 do
  endfor
endfor
```

What is the total communication volume?
Comm. in a Distrib. Address Space

- Remote shared accesses are expensive

```plaintext
procedure LU( A[n,n] distribution [cyclic, *])
begin
  for k <- 0 to n-1 do
    for all j <- k+1 to n-1 do
      BARRIER(bar1, nprocs);
      for j <- k+1 to n-1 do
        Broadcast
        rowK[j] <- A[k,j]
        for all i <- k+1 to n-1 by rows
          for j <- k+1 to n-1 do
          endfor
  endfor

What is the total communication volume?
```

Cost of Communication

\[
Cost = frequency \times \left( Overhead + Latency + \frac{Transfer\ Size}{Bandwidth} - Overlap \right)
\]

- Total Volume
- Number of Comm. Operations
  - Long messages or bulk transfers
  - Long cache lines
- Contention (balance)
  - E.g. all read pivot row
- Overlap
Reducing Contention

- All resources have nonzero occupancy
  - Memory, communication controller, network link, etc.
  - Can only handle so many transactions per unit time

- Effects of contention:
  - Increased end-to-end cost for messages
  - Reduced available bandwidth for individual messages
  - Causes imbalances across processors

- Particularly insidious performance problem
  - Easy to ignore when programming
  - Slow down messages that don’t even need that resource
    - by causing other dependent resources to also congest
  - Effect can be devastating: *Don’t flood a resource!*

---

Types of Contention

- Network contention and end-point contention (hot-spots)

- Location and Module Hot-spots
  - Location: e.g. accumulating into global variable, barrier
    - solution: tree-structured communication

-Module: all-to-all personalized comm. in matrix transpose
  - solution: stagger access by different processors to same node temporally

- In general, reduce burstiness; may conflict with making messages larger
Overlapping Communication

- Cannot afford to stall for high latencies
  - even on unprocessors!
- Overlap with computation or communication to hide latency
- Requires extra concurrency (*slackness*), higher bandwidth
- Techniques:
  - Prefetching
  - Block data transfer
  - Proceeding past communication
  - Multithreading

Replication

- Memory Hierarchy in Parallel Computers
- Remote shared accesses are expensive
- Buffer data for for future reuse
  - Cache in Shared address space
  - Buffer messages in Distributed address space
- Temporal locality
- Spatial locality
- False sharing --- false sharing occurs when a block is invalidated because some word in the block, other than the one being read, is written into.
- Inherent vs Artificial communication
Exploiting Temporal Locality

- Structure algorithm so working sets map well to hierarchy
  - often techniques to reduce inherent communication do well here
  - schedule tasks for data reuse once assigned
- Multiple data structures in same phase
  - e.g. database records: local versus remote
- Solver example: blocking

- More useful when $O(n^{k+1})$ computation on $O(n^k)$ data
  - many linear algebra computations (factorization, matrix multiply)

Spatial Locality Example

- Repeated sweeps over 2-d grid, each time adding 1 to elements
- Natural 2-d versus higher-dimensional array representation

Contiguity in memory layout

(a) Two-dimensional array
(b) Four-dimensional array

Page straddles partition boundaries: difficult to distribute memory well

Cache block straddles partition boundary

Page does not straddle partition boundary

Cache block is within a partition
Extra Work

- Management of parallelism
  - Loop distribution
  - Task queues
  - Synchronization
- Redundant computation
- Inefficiencies
  - Address calculations
  - Less Locality

\[
\text{Speedup} \leq \frac{\text{Sequential Work}}{\max (\text{Work} + \text{Synch Wait Time} + \text{Comm Cost} + \text{Extra Work})}
\]
Trade-offs

- Load Balance
  - fine grain tasks
  - random or dynamic assignment
- Parallelism Overhead
  - coarse grain tasks
  - simple assignment
- Communication
  - decompose to obtain locality
  - recompute from local data
  - big transfers – amortize overhead and latency
  - small transfers – reduce overhead and contention