$\text{Speedup} \leq \frac{\text{Sequential Work}}{\max(\text{Work on any processor})}$

$\text{Speedup} \leq \frac{\text{Sequential Work}}{\max(\text{Work + Synch Wait + Communication})}$
Reducing Extra Work

- **Redundant Computation**
  - If node would be idle anyway, compute data to avoid communication
  - Creating processes (high cost)

Speedup ≤

\[
\frac{\text{Sequential Work}}{\max(\text{Work} + \text{Synch Wait} + \text{Communication} + \text{work})}
\]
Inherent vs. Artifactual Communication

• Potential causes of artifactual communication
  – Poor allocation of data
  – Unnecessary data in transfer
  – Unnecessary data transfer because of system granularity
  – Redundant communication
  – Limited capacity for replication
Cache Memory 101

- Locality + smaller HW is faster = memory hierarchy
  - Levels: each smaller, faster, more expensive/byte than level below
  - Inclusive: data found in top also found in the bottom

- Definitions
  - Upper is closer to processor
  - Block: minimum unit of data present or not in upper level
  - Frame: HW (physical) place to put block (same size as block)
  - Address = Block address + block offset address
  - Hit time: time to access upper level, including hit determination

- 3C Model (Cold/Compulsory, Capacity, Conflict)
- Add communication/coherence misses
Cache Coherent Shared Memory

Interconnection Network

Main Memory

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

Time

P1

Interconnection Network

P2

Main Memory

ld r2, x

x
Cache Coherent Shared Memory

Time

P1

ld r2, x

Interconnection Network

P2

ld r2, x

Main Memory

x
Cache Coherent Shared Memory

Time

ld r2, x
add r1, r2, r4
st x, r1

Interconnection Network

Main Memory

ld r2, x

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

ECE 259 / CPS 221
Orchestration for Performance

• Exploit Temporal and Spatial Locality
  – Temporal locality affects replication
  – Touch too much data $\rightarrow$ capacity misses

• Computation Blocking

Naïve Computation Order

Blocked Computation order
Spatial Locality

• Granularities
  – Communication grain
  – Allocation grain
  – Coherence grain (for cache coherent shared memory)

• What benefits do you get from larger block size?

• Potential disadvantage is false sharing
  – Two or more processors accessing same cache block but don’t share any of the data
Poor Data Allocation

Elements on Same Page

Elements on Same Cache Block

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

Elements on Same Page

Elements on Same Cache Block
Review: Programming for Performance

• Partitioning for Performance
  – Identify concurrency
  – Managing concurrency
    » Static
    » Dynamic
  – Granularity of concurrency
  – Serialization and synchronization costs
  – Communication

• Orchestration for Performance
  – Exploit Locality
  – Data and Computation Blocking
  – Match system (page size, cache block size)
Outline

- Applications
- Creating Parallel Programs
- Programming for Performance
- Scaling
- Synchronization Basics
Scaling: Why Talk About it?

• **Speedup**: change in performance as system parameter is scaled (e.g., number of processors, P)

• **New problems on new machines**
  – Problem scaling
  – Data set size
  – Algorithmic complexity

• **Scaling is natural when simulating physical phenomena**
  – Space is grid
  – Refine grid size
  – Larger grid
Questions in Scaling

• Fundamental question:

  What do real users actually do when they get access to larger parallel machines?

• Constant problem size
  – Just add more processors to speed up execution

• Memory constrained scaling
  – Scale data size linearly with # of processors
  – Can significantly increase execution time

• Time constrained scaling
  – Keep same wall clock time as processors are added
  – Solve largest problem in same time (i.e., before hurricane arrives)
How to scale?

• Not just data

• Must consider application constraints
  – E.g., error scaling

• Equal error scaling
  – Scale all sources of error so they have equal contribution to total error
Example: Barnes-Hut Galaxy Simulation

- Different parameters govern different sources of error
  - Number of bodies \((n)\)
  - Time-step resolution \((dt)\)
  - Force calculation accuracy \((fa)\)

- Scaling Rule
  All components of simulation error should scale at the same rate

- Result: If \(n\) scales by a factor of \(s\)
  - \(dt\) must scale by \(s^{1/4}\)
  - \(fa\) must scale by \(s^{1/4}\)
Demonstrating Scaling Problems

- Small & big Ocean problems on SGI Origin2000

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

- User wants to solve same problem, only faster
  - E.g., Video compression & VLSI routing

\[ \text{Speedup}_{PC}(p) = \frac{\text{Time}(1)}{\text{Time}(p)} \]

- Assessment
  - Good: easy to do & explain
  - May not be realistic
  - Doesn’t work well for much larger machine (c.f., Amdahl’s Law)
Time Constrained Scaling

- Execution time is kept fixed as system scales
  - User has fixed time to use machine or wait for result
- Performance = Work/Time as usual, and time is fixed, so
  \[ \text{Speedup}_{TC}(p) = \frac{\text{Work}(p)}{\text{Work}(1)} \]

- Assessment
  - Often realistic (e.g., best weather forecast overnight)
  - Must understand application to scale meaningfully (would scientist scale grid, time step, error bound, or combination?)
  - Execution time on a single processor can be hard to get (no uniprocessor may have enough memory)
Memory Constrained Scaling

- Scale so memory usage per processor stays fixed
- Scaled Speedup: Is Time(1) / Time(p)?

\[
\text{Speedup}_{MC}(p) = \frac{\text{Work}(p)}{\text{Time}(p)} \times \frac{\text{Time}(1)}{\text{Work}(1)} = \frac{\text{Increase in Work}}{\text{Increase in Time}}
\]

• Assessment
  - Realistic for memory-constrained programs (e.g., grid size)
  - Can lead to large increases in execution time if work grows faster than linearly in memory usage
  - E.g., matrix factorization
    » 10,000-by 10,000 matrix takes 800MB and 1 hour on uniprocessor
    » With 1,000 processors, can run 320K-by-320K matrix
    » But ideal parallel time grows to 32 hours!
Scaling Down

- Scale down to shorten evaluation time on hardware and especially on simulators

- “Scale up” issues apply in reverse

- Must watch out if problem size gets too small
  - Communication dominates computation (e.g., all boundary elements)
  - Problem size gets too small for realistic caches, yielding too many cache hits
    » Scale caches down considering application working sets
    » E.g., if a on a realistic problem a realistic cache could hold a matrix row but not whole matrix
    » Scale cache so it hold only row or scaled problem’s matrix
Outline

• Applications

• Creating Parallel Programs

• Programming for Performance

• Scaling

• Synchronization Basics
A Hierarchy of Synchronization

• Application programmer uses high-level library

• Library programmer uses hardware instructions

• Hardware implements atomic primitives
What the Application Programmer Sees

- Application programmer uses synch libraries

- Machine-independent (i.e., portable) interfaces

- E.g., pthreads provides synch methods
  - Barriers, locks

- Barrier
  - All processors wait at barrier until all others have reached it

- Lock
  - Lock restricts access to shared data to enforce mutual exclusion
What the Library Programmer Sees

• Libraries implement high-level synch interface
  – Can implement locks with different algorithms
  – E.g., can try to acquire with “test & set” or “test and test & set”

• Synch libraries must deal with hardware specifics
  – E.g., CM-5 has hardware support for barriers
  – All machines have atomic operations, but they’re different
  – Synch implementation might depend on system
What the Hardware Does

• All systems implement atomic operations
  – SPARC: Compare & Swap
  – Alpha: Load linked / Store conditional

• Libraries use these primitives to implement synch
  – “Test and Test & Set” algorithm could use Compare & Swap
Transactional Memory (TM)

• Alternate model of synchronization
  – Very hot (overheated?) topic in architecture community

• Concept of transaction
  – Atomic chunk of code
  – Either completely executes or doesn’t execute at all
  – Effects of transaction (writes to shared memory) are either all seen (by other processors) or not seen at all

• Goal
  – Simplify programming \(\rightarrow\) transactions are “easier” than locks
  – But can’t just naively replace locks with transactions

• Much more about this topic, but mostly how to support transactions in hardware and/or software
Outline

- Applications
- Creating Parallel Programs
- Programming for Performance
- Scaling
- Synchronization Basics