Multi-core Programming Speedup

Based on slides from Intel Software College

and

Multi-Core Programming –

increasing performance through software multi-threading

by Shameem Akhter and Jason Roberts,

Agenda

- Concurrency vs. Parallelism
- Characterization
- Hardware & Parallel Computing
- Threading Tools Overview
- Thread Communication

Agenda

Concurrency vs. Parallelism

Characterization

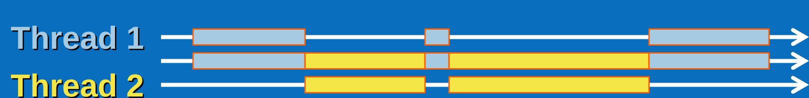
Hardware & Parallel Computing

Threading Tools Overview

Thread Communication

Concurrency vs. Parallelism

- Concurrency: two or more threads are <u>in progress</u> at the same time:



- Parallelism: two or more threads are executing at the same time

- Multiple cores needed

Agenda

Concurrency vs. Parallelism

Characterization

Hardware & Parallel Computing

Threading Tools Overview

Thread Communication

Speedup (Simple)

Measure of how much faster the computation executes versus the best serial code

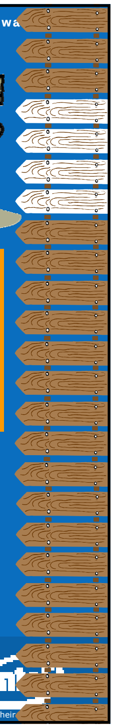
- Serial time divided by parallel time

Example: *Painting a picket fence*

- 30 minutes of preparation (serial)
- One minute to paint a single picket
- 30 minutes of cleanup (serial)

Thus, 300 pickets takes 360 minutes (serial time)

Computing Speedup



Number of painters	Time	Speedup
1	30 + 300 + 30 = 360	1.0X
2	30 + 150 + 30 = 210	1.7X
10	30 + 30 + 30 = 90	4.0X
100	30 + 3 + 30 = 63	5.7X
Infinite	30 + 0 + 30 = 60	6.0X

Illustrates Amdahl's Law

Potential speedup is restricted by serial portion

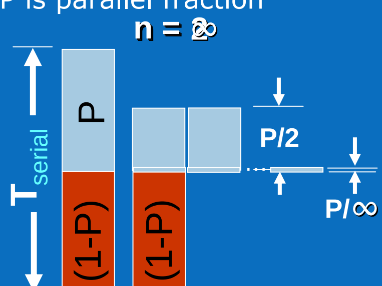
What if fence owner uses spray gun to paint 300 pickets in one hour?

- Better serial algorithm
- If no spray guns are available for multiple workers, what is maximum parallel speedup?

Amdahl's Law

Describes the upper bound of parallel execution speedup

P is parallel fraction



0.5 + 0.25

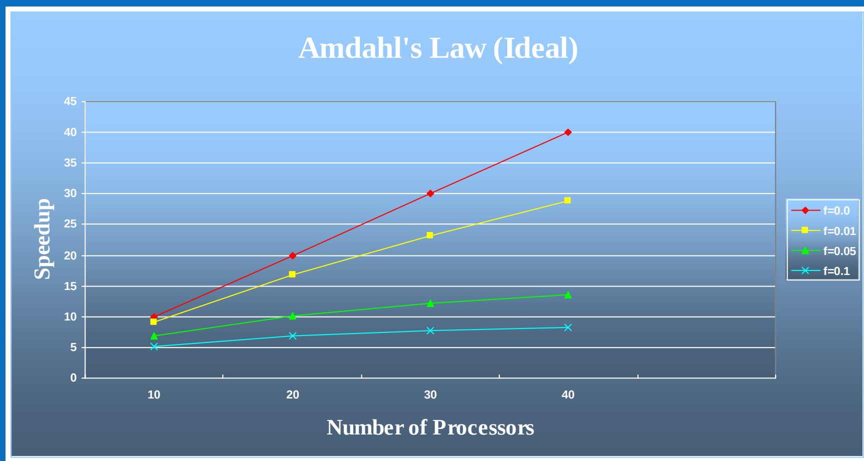
$$T_{parallel} = \{(1\text{-}P) + P/n\}\, T_{serial}$$

n = number of processors

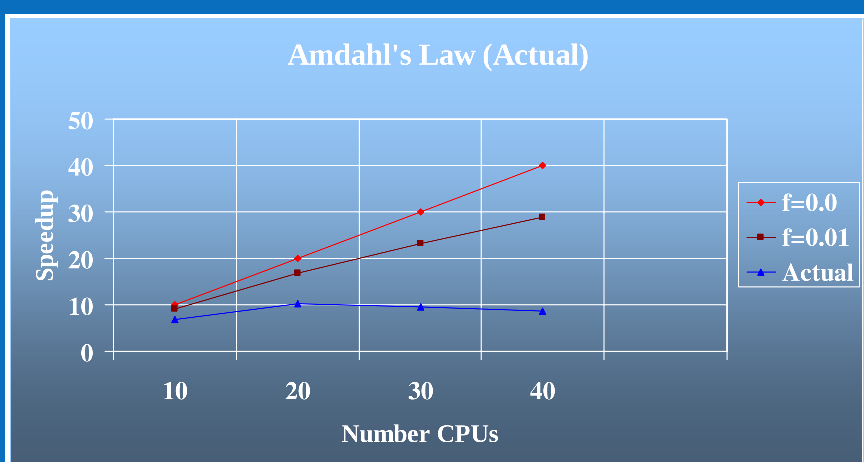
Speedup = T_{serial} / $T_{parallel}$ 1.0/0.55=2.033

Serial code limits speedup

Amdahl's Law



Amdahl's Law (Continued)



Consequences of Amdahl's Law

- Increasing the number of cores only affects the parallel portion
  - If code only 10% parallel, best one can do is run it in 90% of time
- It is more important to reduce the proportion of the code that is sequential than to increase the number of cores
  - P= 0.3 (30% parallel)
    - Running on dual-core S = (1-P) + P/n = .7 + .3/2 = .85
    - Running on quad-core S = (1-P) + P/n = .7 + .3/4 = .775
    - Running on dual-core with double amount of parallelized code P = 0.6 (60% parallel)
      - Running on dual core S = (1-P) + P/n = .4 + .6/2 = .7

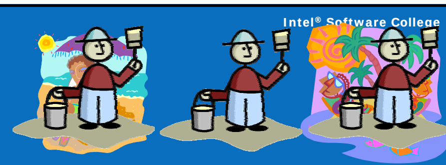
Modification of Amdahl's Law for thread overhead

- Need to consider overhead of adding threads

  Speedup = 1/[(1-P) + P/n + H(n)]

  where H(n) is thread overhead
  - Note that overhead is not linear on good parallel machine
- Source of overhead
  - Actual OS overhead
  - Inter-thread activities such as synchronization and communication
- If overhead large enough speedup can be less than 1
  - Threading can actually slow performance
- Important to design multi-threaded applications well

Efficiency



Measure of how effectively computation resources (threads) are kept busy

- Speedup divided by number of threads
- Expressed as average percentage of non-idle time

Number of painters	Time	Speedup	Efficiency
1	360	1.0X	100%
2	30 + 150 + 30 = 210	1.7X	85%
10	30 + 30 + 30 = 90	4.0X	40%
100	30 + 3 + 30 = 63	5.7X	5.7%
Infinite	30 + 0 + 30 = 60	6.0X	very low

Amdahl's Law and Hyper-Threading

- With HT get 30% performance gain
- A thread on HT enabled processor runs slowe

Repeal of Amdahl's Law

- Amdahl's work seemed to limit applicability of parallel computing
- Amdahl's Law based on assumptions
  - Best serial algorithm limited by CPU cycles available
    - In multi-core may have multiple caches, so more of data may be in cache reducing memory latency
  - Serial algorithm best
    - sometimes parallel solution requires less computational steps
  - Problem size fixed
    - Usually size increases with resources available

Gustafson's Law

Consider scaling problem size as processor count increased

Ts = serial execution time

Tp(N,W) = parallel execution time for same problem, size <u>W</u>, on <u>N</u> CPUs

S(N,W) = Speedup on problem size W, N CPUs

S(N,W) = (Ts + Tp(1,W))/(Ts + Tp(N,W))

Consider case where Tp(N,W) ~ W*W/N

S(N,W) -> (N*Ts + N*W*W)/(N*Ts + W*W)

If W-> ∞ as N -> ∞ then S(N,W) -> N

Gustafson's Law provides hope for parallel applications to deliver on their promise.