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
Concurrency vs. Parallelism

Characterization

Hardware & Parallel Computing

Threading Tools Overview

Thread Communication

• Concurrency: two or more threads are in progress at the same time:

  Thread 1
  Thread 2

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

  Thread 1
  Thread 2

• 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 take 360 minutes (serial time)
Computing Speedup

<table>
<thead>
<tr>
<th>Number of painters</th>
<th>Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30 + 300 + 30 = 360</td>
<td>1.0X</td>
</tr>
<tr>
<td>2</td>
<td>30 + 150 + 30 = 210</td>
<td>1.7X</td>
</tr>
<tr>
<td>10</td>
<td>30 + 30 + 30 = 90</td>
<td>4.0X</td>
</tr>
<tr>
<td>100</td>
<td>30 + 3 + 30 = 63</td>
<td>5.7X</td>
</tr>
<tr>
<td>Infinite</td>
<td>30 + 0 + 30 = 60</td>
<td>6.0X</td>
</tr>
</tbody>
</table>

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?

Illustrates Amdahl’s Law

Potential speedup is restricted by serial portion

Amdahl’s Law

Describes the upper bound of parallel execution speedup

\[ P \text{ is parallel fraction} \]

\[ n = 2 \]

\[ T_{parallel} = \frac{(1-P) + P/n}{T_{serial}} \]

\[ n = \text{number of processors} \]

\[ \text{Speedup} = \frac{T_{serial}}{T_{parallel}} \]

Serial code limits speedup

\[ 0.5 + 0.5 + 0.25 + 0.25 = \frac{1.0}{0.75} = \frac{1.0}{0.5} = 2.0 \]

\[ n = \infty \]

\[ \frac{0.5 + 0.5}{0.5} = 2.0 \]

\[ \frac{1.0}{0.35} = 2.857 \]
Amdahl’s Law

Amdahl’s Law (Ideal)

Amdahl’s Law (Actual)

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: \[ S = (1-P) + P/n = 0.7 + 0.3/2 = 0.85 \]
    - Running on quad-core: \[ S = (1-P) + P/n = 0.7 + 0.3/4 = 0.775 \]
    - Running on dual-core with double amount of parallelized code
      - \[ P = 0.6 \] (60% parallel)
        - Running on dual core: \[ S = (1-P) + P/n = 0.4 + 0.6/2 = 0.7 \]

Modification of Amdahl’s Law for thread overhead

- Need to consider overhead of adding threads
  - \[ \text{Speedup} = 1/[ (1-P) + P/n + H(n)] \]
    - \[ H(n) \] 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

<table>
<thead>
<tr>
<th>Number of painters</th>
<th>Time</th>
<th>Speedup</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>360</td>
<td>1.0X</td>
<td>100%</td>
</tr>
<tr>
<td>2</td>
<td>30 + 150 + 30 = 210</td>
<td>1.7X</td>
<td>85%</td>
</tr>
<tr>
<td>10</td>
<td>30 + 30 + 30 = 90</td>
<td>4.0X</td>
<td>40%</td>
</tr>
<tr>
<td>100</td>
<td>30 + 3 + 30 = 63</td>
<td>5.7X</td>
<td>5.7%</td>
</tr>
<tr>
<td>Infinite</td>
<td>30 + 0 + 30 = 60</td>
<td>6.0X</td>
<td>very low</td>
</tr>
</tbody>
</table>

Amdahl’s Law and Hyper-Threading

- With HT get 30% performance gain
- A thread on HT enabled processor runs slower
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 W, on N 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^2/N
S(N,W) -> (N*Ts + N*W^2)/(N^2*Ts + W^2)
If W->∞ as N->∞ then S(N,W) -> N

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