CSC630/CSC730
Parallel & Distributed Computing

Analytical Modeling of Parallel Programs
Chapter 5

Dr. Joe Zhang
PDC-13: Performance

Contents

• Sources of Parallel Overhead
• Performance Metrics
• Granularity and Data Mapping
• Scalability
Sources of Parallel Overhead

- Interprocessor Communication
- Extra Computation
  - The difference in computation performed by the parallel program and the best serial program
- Idling
  - Load imbalance
  - Synchronization
  - Sequential components

Execution Profile

Tools for analyzing parallel performance
- gprof
- upshot
- Speedshop
Jumpshot is a Java-based visualization tool for doing performance analysis [Link]
http://www.mcs.anl.gov/research/projects/perfvis/software/viewers

**Performance Metrics**

- **Execution time**  
  - The time that elapses from the moment a parallel computation starts to the moment the last PE finishes execution

- $T_s =$ serial run time

- $T_p =$ parallel run time

- **Total overhead:**  
  - The total time collectively spent by all the PEs over and above that required by the fastest known serial algorithm for solving the same problem on a single PE.

\[
T_0 = pT_p - T_s
\]
Performance Metrics - Speedup

- **Speedup**
  - Ratio of the serial runtime of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on a parallel computer with p identical PEs.
  
  \[ S = \frac{T_S}{T_P} \]

- **Theoretically, S < p**
- **Superlinear speedup, S>p**
  - more work by a serial algorithm
  - or result from hardware features (cache effects)

Superlinear Speedup

- Superlinear speedup (>p) can occur due to cache effects and algorithmic shortcuts
- Consider performing a simple version of matrix multiplication on 1 system vs 16
- The matrices might be too large for cache on the 1 system trial, but might fit quite well on the 16 system trial.
Superlinear Effects

Figure 5.3 Searching an unstructured tree for a node with a given label, ‘S’, on two processing elements using depth-first traversal. The two-processor version with processor 0 searching the left subtree and processor 1 searching the right subtree expands only the shaded nodes before the solution is found. The corresponding serial formulation expands the entire tree. It is clear that the serial algorithm does more work than the parallel algorithm.

Dr. Joe Zhang  PDC-13: Performance

Superlinear Speedup

• Exploratory decomposition:
  – Suppose there are 10 processes and the first 9 compute for 2 seconds before determining there is no solution
  – Assume the tenth process finds a solution in 1 second
• The serial program would take 19 seconds to find the solution, while the parallel takes only 1 second
• Speedup = 19
False Speedup

- Have we compared against the best serial algorithm?
  - It is often not possible to find a serial algorithms that is optimal for all instances

- Serial bubble sort takes 150 seconds
- Parallel version (odd-even sort) takes 40 seconds
- Is this a speedup of 150/40 = 3.75?
- No! You need to consider the best serial algorithm.
- Perhaps quicksort takes 30 seconds
- The real speedup is 30/40 = 0.75

Performance Metrics - Efficiency and Cost

- Efficiency
  \[ E = \frac{S}{p} \]
  - Efficiency is also the ratio of sequential cost to parallel cost
- Parallel cost (work or processor-time product)
  - The product of parallel runtime and the number of PEs
  \[ pT_p \]
- Cost-optimal
  - The parallel cost has the same asymptotic growth as a function of the input size as the fast-known sequential algorithm on a single PE.
  \[ E = \Theta(1) \]
Adding n Numbers in Parallel

Figure 5.2  Computing the global sum of 16 partial sums using 16 processing elements. $\sum_i^j$ denotes the sum of numbers with consecutive labels from $i$ to $j$. 

Dr. Joe Zhang  PDC-13: Performance

Speedup for Addition

- Using binary tree-based algorithm
- $\log n$ levels

$$T_p = \Theta(\log n)$$

$$S = \Theta\left(\frac{n}{\log n}\right)$$

$$E = \frac{\Theta\left(\frac{n}{\log n}\right)}{n} = \Theta\left(\frac{1}{\log n}\right)$$

Is the algorithm in Figure 5.2 cost optimal? No
Importance of Cost-Optimality

- Consider a parallel sort with n PEs
  \[ T_p = (\log n)^2 \]
- Serial cost:
  \[ T_s = n \log n \]
- Then speedup is
  \[ S = \frac{n}{\log n} \]
- If we implement with \( p < n \) processors
  \[ T_p = \frac{n(\log n)^2}{p} \quad S = \frac{p}{\log n} \quad E = \frac{1}{\log n} \]
- If \( n = 1000000000 \) and \( p = 32 \), \( S = 32/30 = 1.07 \)

Granularity and Data Mapping

- Scaling down
  - Using fewer PEs to execute a parallel algorithm
- Design for one input element per processor and use less PEs
- Let \( p \) physical PEs to simulate a large number of virtual PEs
  - Each of physical PE simulate \( n/p \) virtual PEs
- Scaling down preserves cost-optimality
  - But it may or may not create cost-optimality
Cost-Optimal Addition

- Each processor adds n/p numbers locally
- Then add the p partial sums on p PEs
- Cost is p times the parallel time

\[ T_p = \Theta\left(\frac{n}{p} + \log p\right) \]

- Its total cost is
  \[ \Theta\left(n + p \log p\right) \]
- As long as n dominates p log p the cost is optimal
  - With n=1000000 and p = 32, p log p = 160
  - E = ?

---

Figure 5.7 A cost-optimal way of computing the sum of 16 numbers using four processing elements.
Scalability

- **Amdahl's Law:**
  - Given a serial component of $W_s$ for a job of size $W$, speedup is limited by $W/W_s$, no matter how many PEs are used.
  - As the problem size grows the serial percentage tends to decrease for many algorithms.
- **Scalability**
  - The capability to increase speedup in proportion to the number of PEs.
- **Efficiency generally drops as $p$ increases**
- **A scalable algorithm allows maintaining a certain level of efficiency by increasing the problem size as $p$ increases.**

---

**FFT Speedup**

![FFT Speedup Graph](image)

**Figure 5.8** A comparison of the speedups obtained by the binary-exchange, 2-D transpose and 3-D transpose algorithms on 64 processing elements with $t_c = 2$, $t_w = 4$, $t_s = 25$, and $t_h = 2$ (see Chapter ?? for details).

---

Dr. Joe Zhang | PDC-13: Performance
---|---
19 | 20
FFT Speedup

- In the FFT speedup graph we see that two algorithms perform much better for smaller arrays, while the third scales better.
- It is typical to run into contradictory speedup results for different sizes.

Scaling Characteristics

- Efficiency can be written in several ways:
  \[ E = \frac{S}{p} = \frac{T_S}{pT_P} = \frac{T_S}{T_0 + T_S} = \frac{1}{1 + \frac{T_0}{T_S}} \]
- The overhead term is an increasing function of \( p \):
  - Since there is some serial component and \( p-1 \) processors are idle during that time.
- The overhead is also a function of problem size:
  - Many times the overhead percentage becomes smaller as the problem size grows.
  - As a result we can scale up to more processors effectively by increasing the problem size.
- A system which can maintain efficiency by scaling the problem size is called scalable.
**Efficiency of Addition**

- The efficiency can be maintained at 0.8 for more CPUs with larger size arrays

<table>
<thead>
<tr>
<th>N</th>
<th>P=1</th>
<th>P=4</th>
<th>P=8</th>
<th>P=16</th>
<th>P=32</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1</td>
<td>0.8</td>
<td>0.57</td>
<td>0.33</td>
<td>0.17</td>
</tr>
<tr>
<td>192</td>
<td>1</td>
<td>0.92</td>
<td>0.8</td>
<td>0.6</td>
<td>0.38</td>
</tr>
<tr>
<td>320</td>
<td>1</td>
<td>0.95</td>
<td>0.87</td>
<td>0.71</td>
<td>0.5</td>
</tr>
<tr>
<td>512</td>
<td>1</td>
<td>0.97</td>
<td>0.91</td>
<td>0.8</td>
<td>0.62</td>
</tr>
</tbody>
</table>
Isoefficiency

- Isoefficiency is a metric to relate problem size required to maintain efficiency with more CPUs

![Graph showing isoefficiency](image)

**Figure 5.10** Variation of efficiency: (a) as the number of processing elements is increased for a given problem size; and (b) as the problem size is increased for a given number of processing elements. The phenomenon illustrated in graph (b) is not common to all parallel systems.

Problem Size

- **Problem size**
  - The number of basic computation steps in the best sequential algorithm to solve the problem on a single PE.

- **Problem size for matrix multiplication** is $O(n^3)$

- **Problem size** is denoted by $W$ and $W = T_S$
  - Assumption: it takes unit time to perform one basic computation step of an algorithm
Isoefficiency Function

To maintain a given level of efficiency $E$, it can be shown that the problem size $W$ must satisfy

$$ E = \frac{W}{p} \frac{T_0}{p} = \frac{Wp}{W + T_0(W, p)} = \frac{W}{W + T_0(W, p)} = \frac{1}{1 + T_0(W, p)/W} $$

$$ W = \frac{E}{1 - E} T_0(W, p) = KT_0(W, p) $$

Where $T_0$ is the overhead function

Since $E$ is constant we simplify to get $W$ equation

- Sometimes the problem size equation can be solved as $p$ increases

These are scalable systems and $W$ is the isoefficiency function

CSC630/CSC730
Parallel & Distributed Computing

Questions?