Colorado State University

CS 475 Parallel Processing Quinn Ch 7: Performance Modeling & Analysis

Sanjay Rajopadhye
Colorado State University

Outline

- Move away from asymptotic analysis
- Account for real machine behavior
- Communication time
- Idle time/load imbalance

Performance Analysis

- General formulas for speedup & efficiency
- Amdahl's Law
- Gustafson's Law (scaled speedup)
- Karp-Flatt Metric
- Isoefficiency
  - Design of scalable algorithms

Speedup & Efficiency

$$\text{Speedup } \psi = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}$$

$$\text{Efficiency } \varepsilon = \frac{\text{Sequential execution time}}{\text{Processors} \times \text{Parallel execution time}}$$

$$\varepsilon = \frac{\text{Speedup}}{\text{Processors}}$$

Execution Time Components

- Inherently sequential computations (c.f. depth of the computation graph) σ
- Perfectly parallelizable computations φ
- Overhead (may also depend on the number of processors, *p*) κ

$$\psi \le \frac{\sigma + \phi}{\sigma + \frac{\phi}{p} + \kappa}$$

Amdahl's Law

- Ignore κ for now (only makes speedup worse: Amdahl is optimistic)
- Bounds on speedup
- Inherently sequential fraction $f = \frac{\sigma}{\sigma + \phi}$

$$\psi \le \frac{1}{f + \frac{1-f}{p}}$$

Example 1

- 95% of a program's execution time is spent in a tight loop that can be parallelized with #omp pragma parallel for. What is the maximum speedup that can be achieved on a 16 core machine?

$$\psi_{16} \le \frac{1}{0.05 + \frac{0.95}{16}} = 9.14$$

- What is the max speedup possible on any machine?

$$\psi_{\infty} \le \frac{1}{0.05 + \frac{0.95}{\infty}} = 20$$

Limitations

- Why did we just not give up?
- Why did people (the supercomputing community) continue to write codes that run on very large number of processors?

Recap

- Inherently sequential computations (c.f. depth of the computation graph) σ(*n*)
- Perfectly parallelizable computations φ(*n*)
- Overhead (may also depend on the number of processors, *p*) **κ**(*n, p*)

$$\psi(n,p) \le \frac{\sigma(n)+\phi(n)}{\sigma(n)+\frac{\phi(n)}{p}+\kappa(n,p)}$$

Colorado State University 9

Amdahl's Law

- Inherently sequential fraction:

$$f(n)=\frac{\sigma(n)}{\sigma(n)+\phi(n)}$$

$$\psi(n,p) \le \frac{1}{f(n)+\frac{1-f(n)}{p}}$$

Colorado State University 10

Gustafson-Barsis' Law

- Let $$s(n,p) = \frac{\sigma(n)}{\sigma(n) + \frac{\phi(n)}{p}}$$
- Then we can show that

$$\psi(n,p) \le p - (p-1)s$$

Colorado State University 11

Example 2

- An application running on 10 processors spends 3% of its execution time doing serial work. What is its scaled speedup?

$$\psi(n,p) \le 10 - (10-1)0.03 = 10 - 0.27 = 9.73$$

Except that 9 don't do the serial fraction

Execution on 1 processor takes 10 times longer

Colorado State University 12

Karp Flatt Metric

- Both Amdahl and Gustafson-Barsis ignore the overhead term, $\kappa(n,p)$
- Overestimate the (scaled) speedup
- Karp & Flatt proposed a more realistic (and empirical) metric
- Allows to account for decreasing speedup

Colorado State University 13

Empirical Serial Fraction

- Start with Amdahl's law. $\psi(n,p) \le \dfrac{1}{f(n) + \dfrac{1-f(n)}{p}}$

multiply top & bottom by *p*,
collect the *f*(*n,p*) terms together and take them to lhs – solve for *f*(*n,p*), assuming you know ψ

$$e(n,p) = \frac{\dfrac{p}{\psi(n,p)} - 1}{p-1} = \frac{\dfrac{1}{\psi(n,p)} - \dfrac{1}{p}}{1 - \dfrac{1}{p}}$$

Colorado State University 14

Example 1

p	2	3	4	5	6	7	8
ψ	1.8	2.5	3.1	3.6	4.0	4.4	4.7

e	0.1	0.1	0.1	0.1	0.1	0.1	0.1

- Why is speedup only 4.7 on 8 processors?

Example 2

p	2	3	4	5	6	7	8
ψ	1.9	2,6	3.2	3.7	4.1	4.5	4.7

e	0.07	0.075	0.08	0.085	0.09	0.095	0.1

- Why is speedup only 4.7 on 8 processors?
- *e* is steadily increasing. Overhead is the culprit

Isoefficiency Metric

- Main goal is to quantify the relative scaling of problem size and number of processors
- And account for the overhead term quantitatively
  - To maintain "good" performance
  - What is good?
  - Maintain constant efficiency = linear speedup

Colorado State University 17

Isoefficiency Analysis

- Start with speedup formula
- Identify the total overhead
  - Non-essential work done by the parallel program
- Do algebra so that efficiency = constant
- Determine the relationship between the sequential execution time (work) and overhead

Colorado State University 18

Problem size

- Isoefficiency analysis studies
  - How problem size should increase
  - As #p is increased
  - To keep efficiency constant
- What is problem size?
  - Not a parameter like n as in most analyses
  - But rather the work of the best sequential algorithm, $W = T(n,1) = \sigma(n)+\varphi(n)$

Colorado State University 19

General Approach

- Express overhead as function of n and p.
- Isoefficiency relation:

$$W(n) = KT_o(n,p)$$

- Massage this to remove n from the rhs

$$W(n) = f(p)$$

- Function $f(p)$ is isoefficiency function

Colorado State University 20

Scalability

- The smallest growing isoefficiency function is the most scalable.
- Factors that impose a lower bound on *f*(p)
  - Communication costs, and load imbalance
  - Memory bounds
  - Degree of parallelism in the application itself

Colorado State University 21

Isoefficiency relation

- Remember, $T(n,p) = \sigma(n) + \frac{\phi(n)}{p} + \kappa(n,p)$
- Overhead = useless work:
  - Multiply *T*(*n*,*p*) by *p*, remove useful work:

$$T_o(n,p) = (p-1)\sigma(n) + p\kappa(n,p)$$

  - Modify speedup equation to use T_o rather than κ

$$\psi(n,p) = \frac{p}{1 + \frac{T_o(n,p)}{W}}$$

  - For efficiency = constant

$$T(n,1) = W = KT_o(n,p) = KT_o(W,p)$$

Colorado State University 22

Example 1: Reduction

- Sequential $T(n,1) = W = n$
- Parallel $T(n,p) = (n/p) + \log(p)$
- Overhead $T_o(W,p) = p\ \log(p)$
- Isoefficiency function: *p* log (*p*)
- How should work increase as *p* increases?

Example 2

- Complicated overhead function
- Overhead $T_o(W,p) = p^{3/2} + pW^{3/4}$
- Separate the different parts, analyze each one and take the worst case
  - First function: $W = \Theta(p^{3/2})$
  - Second $W = KpW^{3/4}$ i.e., $W = \Theta(p^4)$
  - Second one dominates. The work must grow as the 4th power of the number of processors, to maintain linear speedup.