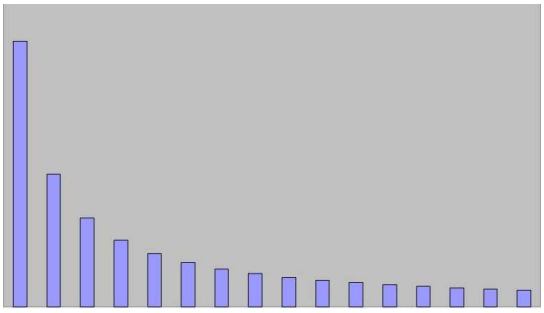
Performance Analysis

Introduction

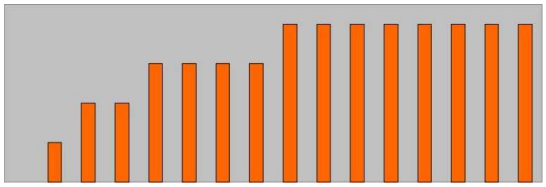
- Analysis of execution time for parallel algorithm to dertmine if it is worth the effort to code and debug in parallel
- Understanding barriers to high performance and predict improvement
- Goal: to figure out whether a program merits parallelization

Speedup and efficiency

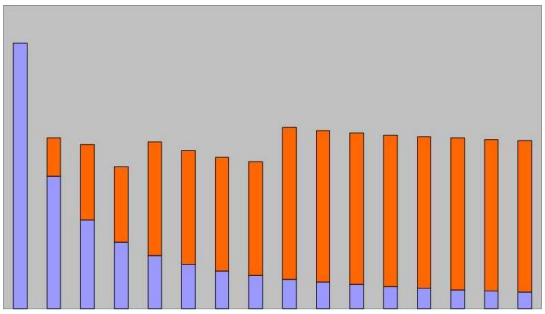
- Speedup
  - Expectation is for the parallel code to run faster than the sequential counterpart
  - Ratio of sequential execution time to parallel execution time
- Execution time components
  - Inherently sequential computations: $\sigma(n)$
  - Potentially parallel computations: $\varphi(n)$
  - Communication operations and other repeat computations: $\kappa(n,p)$
- Speedup $\psi(n,p)$ solving a problem of size n on p processors
  - On sequential computer
    - Only one computation at a time
    - No interprocess communication and other overheads
    - Time required is given by
    $$\sigma(n) + \varphi(n)$$
  - On parallel computer
    - Cannot do anything about inherently sequential computation
    - In best case, computation is equally divided between p PEs
    - Also need to add time for interprocess communication among PEs
    - Time required is given by
    $$\sigma(n) + \frac{\varphi(n)}{p} + \kappa(n,p)$$
    - Parallel execution time will be larger if it cannot be perfectly divided among p processors, leading to smaller speedup
  - Actual speedup will be limited by
  $$\psi(n,p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \frac{\varphi(n)}{p} + \kappa(n,p)}$$
  - Adding processors reduces computation time but increases communication time
    - At some point, communication time increase is larger than decrease in computation time
    - *Elbow out point*
      - The speedup begins to decline with additional PEs



Computation time



Communication time



- Efficiency
  - Measure of processor utilization
  - Ratio of speedup to number of processors used
  - Efficiency $\varepsilon(n,p)$ for a problem of size n on p processors is given by

$$\begin{aligned} & \varepsilon(n,p) & \leq & \frac{\sigma(n)+\varphi(n)}{p\times\left(\sigma(n)+\frac{\varphi(n)}{p}+\kappa(n,p)\right)} \\ \Rightarrow\ & \varepsilon(n,p) & \leq & \frac{\sigma(n)+\varphi(n)}{p\sigma(n)+\varphi(n)+p\kappa(n,p))} \end{aligned}$$

  - It is easy to see that

$$0 \leq \varepsilon(n,p) \leq 1$$

Amdahl's law

- Consider the expression for speedup $\psi(n,p)$ above
  - Since $\kappa(n,p) > 0$

$$\begin{aligned} \psi(n,p) & \leq & \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\frac{\varphi(n)}{p}+\kappa(n,p)} \\ & \leq & \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\frac{\varphi(n)}{p}} \end{aligned}$$

    This gives us the maximum possible speedup in the absence of communication overhead
  - Let f denote the inherently sequential fraction of the computation

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

  - Then, we have

$$\begin{aligned}\psi(n,p) &\leq \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\frac{\varphi(n)}{p}} \\ &\leq \frac{\sigma(n)/f}{\sigma(n)+\sigma(n)(1/f-1)/p} \\ &\leq \frac{1/f}{1+(1/f-1)/p} \\ &\leq \frac{1}{f+(1-f)/p}\end{aligned}$$

Definition 1 *Let f be the fraction of operations in a computation that must be performed sequentially, where $0 \leq f \leq 1$. The maximum speedup ψ achievable by a parallel computer with p processors performing the computation is*

$$\psi \leq \frac{1}{f+(1-f)/p}$$

- Based on the assumption that we are trying to solve a problem of fixed size as quickly as possible
  - Provides an upper bound on the speedup achievable with a given number of processors, trying to solve the problem in parallel
  - Also useful to determine the asymptotic speedup as the number of processors increases
- Example 1
  - Determine whether it is worthwhile to develop a parallel version of a program to solve a particular problem
  - 95% of program's execution occurs inside a loop that can be executed in parallel
  - Maximum speedup expected from a parallel version of program executing on 8 CPUs

$$\begin{aligned}\psi &\leq \frac{1}{0.05+(1-0.05)/8} \\ &\approx 5.9\end{aligned}$$

  - Expect a speedup of 5.9 or less
- Example 2
  - 20% of a program's execution time is spent within inherently sequential code
  - Limit to the speedup achieveable by a parallel version of the program

$$\begin{aligned}\psi &= \lim_{p\to\infty}\frac{1}{0.2+(1-0.2)/p} \\ &= \frac{1}{0.2} \\ &= 5\end{aligned}$$

- Example 3
  - Parallel version of a sequential program with time complexity $\Theta(n^2)$, n being the size of dataset
  - Time needed to input dataset and output result (sequential portion of code): $(18000+n)$ μs
  - Computational portion can be executed in parallel in time $(n^2/100)$ μs
  - Maximum speedup on a problem of size 10,000

$$\psi \leq \frac{28000+1000000}{28000+1000000/p}$$

- Limitations of Amdahl's law
  - Ignores communication overhead $\kappa(n,p)$
    - Assume that the parallel version of program in Example 3 has $\lceil \log n \rceil$ communication points
    - Communication time at each of these points: $10000\lceil \log p \rceil + (n/10)$ μs
    - For a problem with $n = 10000$, the total communication time is

      $$14(10000\lceil \log p \rceil + 1000)\mu\text{s}$$

      The communication time takes into account $\lceil \lg 10000 \rceil = 14$ for transmission across the net
    - Now, the speedup is given by

      $$\psi \leq \frac{28000 + 1000000}{42000 + 1000000/p + 140000\lceil \log p \rceil}$$
  - Ignoring communication time results in overestimation of speedup
- The Amdahl effect
  - Typically, $\kappa(n,p)$ has lower complexity than $\varphi(n)$
    - In the hypothetical case of Example 3

      $$\begin{aligned} \kappa(n,p) &= \Theta(n \log n + n \log p) \\ \varphi(n) &= \Theta(n^2) \end{aligned}$$
  - As n increases, $\varphi(n)/p$ dominates $\kappa(n,p)$
  - For a fixed p, as n increases, speedup increases
- Review of Amdahl's law
  - Treats problem size as a constant
  - Shows how execution time decreases as number of processors increase
- Another perspective
  - We often use faster computers to solve larger problem instances
  - Treat time as a constant and allow problem size to increase with number of CPUs

Gustafson-Barsis's law

- Assumption in Amdahl's Law: Minimizing execution time is the focus of parallel computing
  - Assumes constant problem size and demonstrates how increasing PEs can reduce time
- User may want to use parallelism to improve the accuracy of the result
- Treat time as a constant and let the problem size increase with number of processors
  - Solve a problem to higher accuracy within the available time by increasing the number of PEs
  - Sequential fraction of computation may decrease with increase in problem size (Amdahl effect)
    - Increasing number of PEs allows for increase in problem size, decreasing inherently sequential fraction of a computation, and increasing the quotient between serial and parallel execution times
  - Consider speedup expression again ($\kappa(n,p) \geq 0$)

    $$\psi(n,p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \frac{\varphi(n)}{p}}$$

  - Let s be the fraction spent in parallel computation performing inherently sequential operations
  - Pure parallel computations can be given as $1 - s$

$$\begin{aligned} s &= \frac{\sigma(n)}{\sigma(n) + \frac{\varphi(n)}{p}} \\ 1 - s &= \frac{\varphi(n)/p}{\sigma(n) + \frac{\varphi(n)}{p}} \end{aligned}$$

  - This leads to

$$\begin{aligned} \sigma(n) &= \left(\sigma(n) + \frac{\varphi(n)}{p}\right) s \\ \varphi(n) &= \left(\sigma(n) + \frac{\varphi(n)}{p}\right) (1 - s)p \end{aligned}$$

  - Now, speedup is given by

$$\begin{aligned} \psi(n, p) &\leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \frac{\varphi(n)}{p}} \\ &\leq \frac{\left(\sigma(n) + \frac{\varphi(n)}{p}\right)(s + (1 - s)p)}{\sigma(n) + \frac{\varphi(n)}{p}} \\ &\leq s + (1 - s)p \\ &\leq p + (1 - p)s \end{aligned}$$

Definition 2 *Given a parallel program solving a problem of size n using p processors, let s denote the fraction of total execution time spent in serial code. The maximum speedup ψ achievable by this program is*

$$\psi \leq p + (1 - p)s$$

- Predicts scaled speedup
  - Amdahl's law looks at serial computation and predicts how much faster it will be on multiple processors
    - It does not scale the availability of computing power as the number of PEs increase
  - Gustafson-Barsis's law begins with parallel computation and estimates the speedup compared to a single processor
- In many cases,assuming a single processor is only p times slower than p processors is overly optimistic
  - Solve a problem on parallel computer with 16 PEs, each with 1GB of local memory
  - Let the dataset occupy 15GB, and aggregate memory of parallel computer is barely large enough to hold the dataset and multiple copies of program
  - On a single processor machine, the entire dataset may not fit in the primary memory
  - If working set size of process exceeded 1GB, it may begin to thrash, taking more than 16 times as long to execute the parallel portion of the code as the group of 16 PEs
  - Effectively, in Gustafson-Bassis's Law, speedup is the time required by parallel program divided into the time that *would* be required to solve the same problem on a single CPU, *if* it had sufficient memory
  - Speedup predicted by Gustafson-Barsis's Law is called *scaled speedup*
    - Using the parallel computation as the starting point, rather than the sequential computation, it allows the problem size to be an increasing function of the number of PEs
- A driving metaphor

  - Amdahl's Law: Suppose a car is traveling between two cities 60 miles apart, and has already spent one hour traveling half the distance at 30 mph; no matter how fast you drive the last half, it is impossible to achieve 90 mph average before reaching the second city. Since it has already taken you one hour and you have a distance of 60 miles total, going infinitely fast you would only achieve 60 mph.
  - Gustafson's Law: Suppose a car has already beeen traveling for some time at less than 90 mph. Given enough time and distance to travel, the car's average speed can always eventually reach 90 mph, no matter how long and how slowly it has already traveled. For example, if the car spent one hour at 30 mph, it could achieve this by driving at 120 mph for two additional hours, or at 150 mph for an hour, and so on.

- Example 1
  - An application running on 10 processors spends 3% of its time in serial code
  - Scaled speedup of this application

$$\begin{aligned} \psi &= 10 + (1 - 10)(0.03) \\ &= 10 - 0.27 \\ &= 9.73 \end{aligned}$$

- Example 2
  - Desired scaled speedup of 7 on 8 processors
  - Maximum fraction of program's parallel execution time that can be spent in serial code

$$7 = 8 + (1 - 8)s \Rightarrow s \approx 0.14$$

- Example 3
  - Vicki plans to justify her purchase of a $30 million Gadzooks supercomputer by demonstrating its 16,384 processors can achieve a scaled speedup of 15,000 on a problem of great importance to her employer. What is the maximum fraction of the parallel execution time that can be devoted to inherently sequential operations if her application is to achieve this goal?

$$\begin{aligned} 15000 &\leq 16384 + (1 - 16384)s \\ s &\leq 1384/16383 \\ &\leq 0.084 \end{aligned}$$

- Application in research
  - Amdahl's Law assumes that the computing requirements will stay the same, given increased processing power
    - Analysis of the same data will take less time with more computing power
  - Gustafson's Law argues that more computing power will cause the data to be more carefully and fully analyzed at a finer scale (pixel by pixel or unit by unit) than on a larger scale
    - Simulating impact of nuclear detonation on every building, car, and their contents

Karp-Flatt metric

- Amdahl's law and Gustafson-Barsis's law ignore communication overhead
  - Results in overestimation of speedup or scaled speedup
- Experimentally determined serial fraction to get performance insight
  - Execution time of parallel program on p processors

$$T(n, p) = \sigma(n) + \frac{\varphi(n)}{p} + \kappa(n, p)$$

- No communication overhead for serial program

$$T(n,1) = \sigma(n) + \varphi(n)$$

- *Experimentally determined serial fraction* e of parallel computation is the total amount of idle and overhead time scaled by two factors: the number of PEs (less one) and the sequential execution time

$$e = \frac{(p-1)\sigma(n) + p\kappa(n,p)}{(p-1)T(n,1)}$$

- Now, we have

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

- Rewrite the parallel execution time as

$$T(n,p) = (T(n,1)e + T(n,1)(1-e)/p$$

- Use ψ as a shorthand for $\psi(n,p)$

$$\begin{aligned} \psi &= \frac{T(n,1)}{T(n,p)} \\ T(n,1) &= T(n,p)\psi \end{aligned}$$

- Now, compute time for p processors as

$$\begin{aligned} T(n,p) &= T(n,p)\psi e + T(n,p)\psi(1-e)/p \\ 1 &= \psi e + \psi(1-e)/p \\ \frac{1}{\psi} &= e + \frac{1-e}{p} \\ &= e + \frac{1}{p} - \frac{e}{p} \\ &= e\left(1 - \frac{1}{p}\right) + \frac{1}{p} \\ e &= \frac{\frac{1}{\psi} - \frac{1}{p}}{1 - \frac{1}{p}} \end{aligned}$$

Definition 3 *Given a parallel computation exhibiting speedup* ψ *on* p *processors, where* $p > 1$*, the experimentally determined serial fraction* e *is defined to be*

$$e = \frac{\frac{1}{\psi} - \frac{1}{p}}{1 - \frac{1}{p}}$$

- Useful metric for two reasons
  1. Takes into account parallel overhead ($\kappa(n,p)$)
  2. Detects other sources of overhead or inefficiency ignored in speedup model
     - Process startup time
     - Process synchronization
     - Imbalanced workload
     - Architectural overhead
- Helps to determine whether the efficiency decrease in parallel computation is due to limited opportunities for parallelism or the increase in algorithmic/architectural overhead

- Example 1
  - Look at the following speedup table with p processors

p	2	3	4	5	6	7	8
ψ	1.82	2.50	3.08	3.57	4.00	4.38	4.71

  - What is the primary reason for speedup of only 4.71 on 8 CPUs?
  - Compute e

e	0.1	0.1	0.1	0.1	0.1	0.1	0.1

  - Since e is constant, the primary reason is large serial fraction in the code; limited opportunity for parallelism
- Example 2
  - Look at the following speedup table with p processors

p	2	3	4	5	6	7	8
ψ	1.87	2.61	3.23	3.73	4.14	4.46	4.71

  - What is the primary reason for speedup of only 4.71 on 8 CPUs?
  - Compute e

e	0.070	0.075	0.080	0.085	0.090	0.095	0.100

  - Since e is steadily increasing, the primary reason is parallel overhead; time spent in process startup, communication, or syncronization, or architectural constraint

Isoefficiency metric

- Parallel system: A parallel program executing on a parallel computer
- Scalability: A measure of parallel system's ability to increase performance as number of processors increase
  - A scalable system maintains efficiency as processors are added
- Amdahl effect
  - Speedup and efficiency are typically an increasing function of problem size because of communication complexity being lower than computational complexity
  - The problem size can be increased to maintain the same level of efficiency when processors are added
- Isoefficiency: way to measure scalability
- Isoefficiency derivation steps
  - Begin with speedup formula
  - Compute total amount of overhead
  - Assume efficiency remains constant
  - Determine relation between sequential execution time and overhead
- Deriving isoefficiency relation

$$\begin{aligned}
\psi(n,p) &\leq \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\frac{\varphi(n)}{p}+\kappa(n,p)} \\
&\leq \frac{p(\sigma(n)+\varphi(n))}{p\sigma(n)+\varphi(n)+p\kappa(n,p)} \\
&\leq \frac{p(\sigma(n)+\varphi(n))}{\sigma(n)+\varphi(n)+(p-1)\sigma(n)+p\kappa(n,p)}
\end{aligned}$$

- Let $T_0(n,p)$ be the total time spent by all processors doing work not done by sequential algorithm; it includes
  - Time spent by $p-1$ processors executing inherently sequential code
  - Time spent by all p processors performing communication and redundant computations

$$T_0(n,p) = (p-1)\sigma(n) + p\kappa(n,p)$$

- Substituting for $T_0(n,p)$ in speedup equation

$$\psi(n,p) \leq \frac{p(\sigma(n)+\varphi(n))}{\sigma(n)+\varphi(n)+T_0(n,p)}$$

- Efficiency equals speedup divided by p

$$\begin{aligned} \varepsilon(n,p) &\leq \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)+T_0(n,p)} \\ &\leq \frac{1}{1+\frac{T_0(n,p)}{\sigma(n)+\varphi(n)}} \end{aligned}$$

- Substituting $T(n,1) = \sigma(n)+\varphi(n)$, we have

$$\begin{aligned} \varepsilon(n,p) &\leq \frac{1}{1+\frac{T_0(n,p)}{T(n,1)}} \\ \frac{T_0(n,p)}{T(n,1)} &\leq \frac{1-\varepsilon(n,p)}{\varepsilon(n,p)} \\ T(n,1) &\geq \frac{\varepsilon(n,p)}{1-\varepsilon(n,p)} T_0(n,p) \end{aligned}$$

- For constant level of efficiency, the fraction $\frac{\varepsilon(n,p)}{1-\varepsilon(n,p)}$ is a constant
- This yields the isoefficiency relation as

$$T(n,1) \geq CT_0(n,p)$$

Definition 4 *Suppose a parallel system exhibits efficiency $\varepsilon(n,p)$. Define $C = \frac{\varepsilon(n,p)}{1-\varepsilon(n,p)}$ and $T_0(n,p) = (p-1)\sigma(n) + p\kappa(n,p)$. In order to maintain the same level of efficiency as the number of processors increases, n must be increased so that the following inequality is satisfied:*

$$T(n,1) \geq CT_0(n,p)$$

- Isoefficiency relation
  - Used to determine the range of processors for which a given level of efficiency can be maintained
  - Parallel overhead increases with number of processors
    - Efficiency can be maintained by increasing the size of the problem being solved
- Scalability function
  - All algorithms assume that the maximum problem size is limited by the amount of primary memory available
  - Treat space as the limiting factor in the analysis
  - Assume that the parallel system has isoefficiency relation as $n \geq f(p)$
    - $M(n)$ denotes the amount of memory needed to store a problem of size n
    - Relation $M^{-1}(n) \geq f(p)$ indicates how the amount of memory used must increase as a function of p to maintain a constant level of efficiency
    - Rewrite the relation as $n \geq M(f(p))$
    - Total amount of memory available is a linear function of the number of processors

    - Amount of memory *per processor* must increase by $M(f(p))/p$ to maintain the same level of efficiency
  - Scalability function is: $M(f(p))/p$
  - Complexity of scalability function determines the range of processors for which a constant level of efficiency can be maintained
    - If $M(f(p))/p = \Theta(1)$, memory requirement per processor is constant and the system is perfectly scalable
    - If $M(f(p))/p = \Theta(p)$, memory requirement per processor increases linearly with the number of processors p
      - While memory is available, it is possible to maintain same level of ε by increasing the problem size
      - But memory used per processor increases linearly with p; at some point it may reach the memory capacity of the system
      - Limits efficiency when the number of processors increase to a level
    - Similar arguments for the cases when $M(f(p))/p = \Theta(\log p)$ and $M(f(p))/p = \Theta(p \log p)$
  - In general, lower the complexity of $M(f(p))/p$, higher the scalability of the parallel system

- Example – Reduction
  - Computational complexity of sequential reduction algorithm – $\Theta(n)$
  - Reduction step time complexity in parallel algorithm – $\Theta(\log p)$
  - Since every PE participates in this step, $T_0(n, p) = \Theta(p \log p)$
  - Isoefficiency relation for reduction algorithm (with a constant C)

$$n \geq Cp \log p$$

  - Since sequential algorithm reduces n values, $M(n) = n$, leading to

$$\begin{aligned} M(Cp \log p)/p &= Cp \log p/p \\ &= C \log p \end{aligned}$$

  - Reduction of n values on p PEs
    - Each PE adds about n/p values and participates in the reduction that has $\lceil \log p \rceil$ steps
    - If we double the values of n and p, each PE still adds about n/p values, but reduction time is increased to $\lceil \log(2p) \rceil$; this drops efficiency
    - To maintain the same level of efficiency, n has to be greater than double with twice the number of PEs

- Example – Floyd's algorithm
  - Sequential algorithm has time complexity $\Theta(n^3)$
  - Each of p PEs spends $\Theta(n^2 \log p)$ time in communications
  - Isoefficiency relation

$$n^3 \geq C(pn^2 \log p) \Rightarrow n \geq Cp \log p$$

  - Consider the memory requirements along with problem size n
  - Amount of storage needed to represent a problem of size n is n^2, or $M(n) = n^2$
  - The scalability function is

$$\begin{aligned} M(Cp \log p)/p &= C^2 p^2 \log^2 p/p \\ &= C^2 p \log^2 p \end{aligned}$$

  - Poor scalability compared to parallel reduction