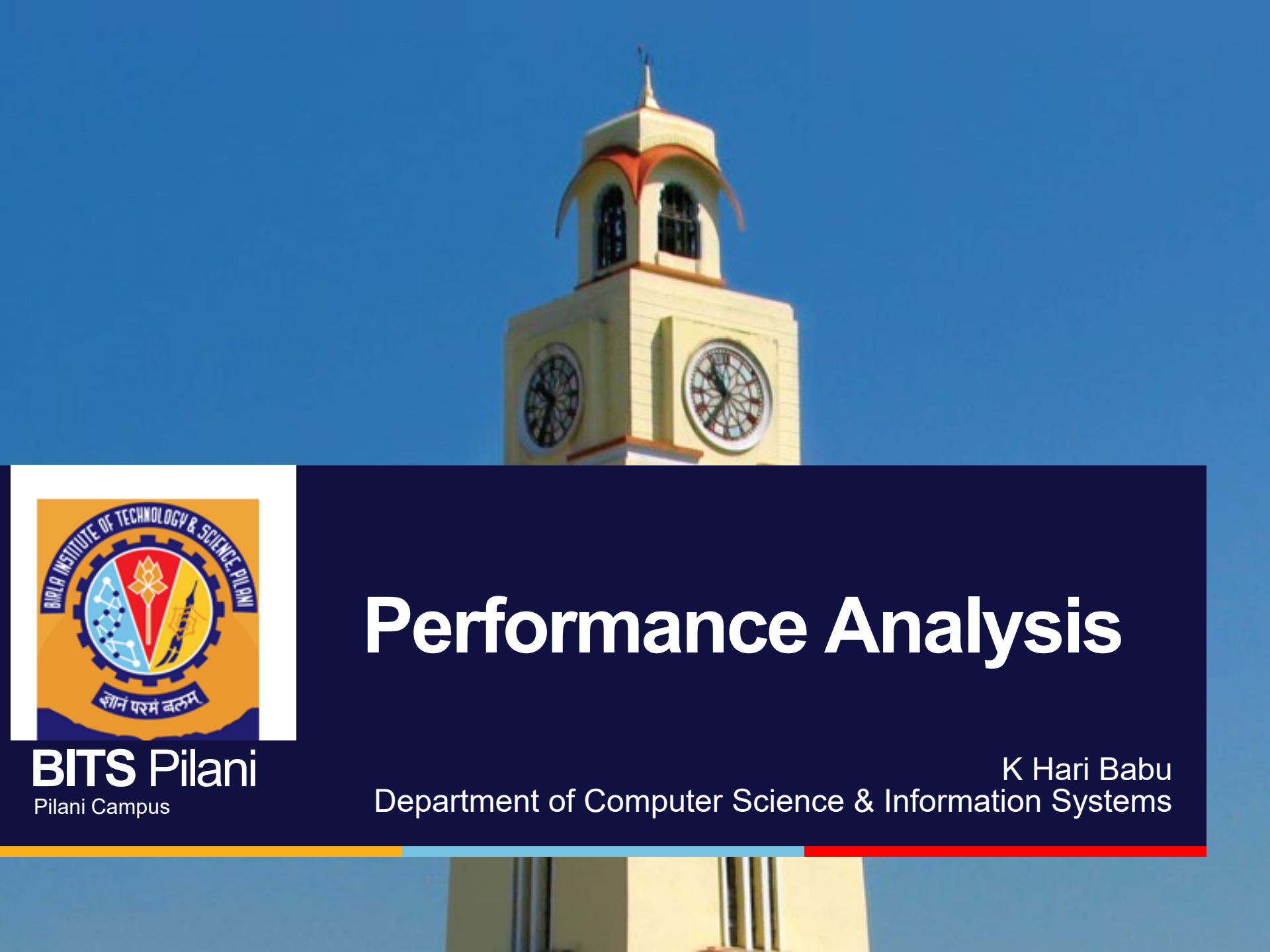


Performance Analysis

BITS Pilani
Pilani Campus

K Hari Babu
Department of Computer Science & Information Systems

innovate achieve lead

BITS Pilani
Pilani Campus

Amdahl's Law & Gustafson-Barsis' Law

Performance Model

- (System level) Performance relates to Processor Utilization:
  - If processor(s) is (or are) 100% utilized then you cannot hope to do better:
    - Assumption: Processor is the fastest and most critical component in the system
- This model can be extended to parallel systems
  - **Speedup** = T_{serial} / $T_{parallel}$
  - Ideal speedup with ***p*** processors:
    - ***p***, if the program can be fully parallelized (e.g. data parallel execution)
  - **Parallel efficiency =** $(performance\ on\ n\ processors)/(n * performance\ on\ 1\ CPU$
  - $= \frac{Speedup}{n}$
    - how effectively a given resource (e.g. CPU) is used

Performance Model

- **Scaling**
  - How well algorithms work for large problems requiring lots of time?
- Strong scaling:
  - How does the algorithm perform as the number of processors P increases for a fixed problem size n?
    - Any algorithm will eventually break down (consider P >n)
- Weak scaling:
  - How does the algorithm perform when the problem size increases with the number of processors?
    - E.g. If we double the number of processors can we solve a problem "twice as large" in the same time

Factors Limiting To Sublinear Speed-Up

- Load imbalance
  - Not all processes might execute their tasks in the same amount of time
    - because the problem was not (or could not) be partitioned into pieces with equal complexity
  - This load imbalance hampers performance because some resources are underutilized
- Shared Resources
  - shared resources like, shared paths to memory, I/O devices are needed by all processes
    - This will effectively serialize part of the concurrent execution
- Communication Overhead
  - The parallel algo may require some communication between processes
    - adding overhead that would not be present in the serial case

Super Linear Speed Up

- A program will exhibit superlinear speedup
- Scaled-up cache memory in p processes
  - If a program's performance is strongly dependent on having a sufficient amount of cache memory
    - if multiple workers bring that cache memory which is not possible by a single worker
- A parallel algorithm may be *more* efficient than the equivalent serial algorithm
  - since it may be able to avoid work that its serialization would be forced to do
    - For example, in search tree problems, searching multiple branches in parallel sometimes permits chopping off branches (by using results computed in sibling branches) sooner than would occur in the serial code.

Amdahl's Law

- *f* = fraction of computation (algorithm) that is serial and *cannot be parallelized*
  - Data setup
  - Reading/writing to a single disk file
- $T_s = f * Ts + (1 - f) * Ts$
  = serial portion + parallelizable portion
- $T_p = f * Ts + (1 - f) * Ts/p$
  - *p* is the number of processors
- Speedup= $S(p) = \frac{T_s}{f*Ts+(1-f)*\frac{T_s}{p}}$
- $$S(p) = \frac{1}{f+\frac{1-f}{p}}$$
- Limit as p → ∞, S(p) → 1/f

Amdahl's Law - Example

BITS Pilani

- Suppose that a program has a 14% serial code, what is the speedup if 24 processors are used? What is the maximum speedup that could be achieved?
- $S(p) = \frac{1}{f + \frac{1-f}{p}}$
  - =1/(0.14 + (1-0.14)/24) = 5.68
- What is the maximum speedup?
  - 1/0.14 = 7.14
- Parallelization efficiency decreases as the amount of resources increases
  - E(12)=4.72/12=.394
  - E(24)=5.68/24=.236
    - For this reason, parallel computing with many processors is useful only for highly parallelized programs

Amdahl's Law - Example

- Suppose we nave implemented a parallel version of a sequential program with time complexity $\theta(n^2)$, where n is the size of the dataset. Assume the time needed to input the dataset and output the result is (18000 + n) μsec. This constitutes the sequential portion of the program. The computational portion of the program can be executed in parallel; it has execution time (n^2 /100) μsec. What is the maximum speedup achievable by this parallel program on a problem of size 10000?
- Solution: $S(p) = \frac{1}{f + \frac{1-f}{p}}$
  - f= (18000+n) / (18000+n)+ (n^2 /100) = $28000/10^6$ = 0.028
  - Maximum speedup=1/f= 35.7
  - For p=16 → speedup=11.27

Amdahl's Law - Limitations

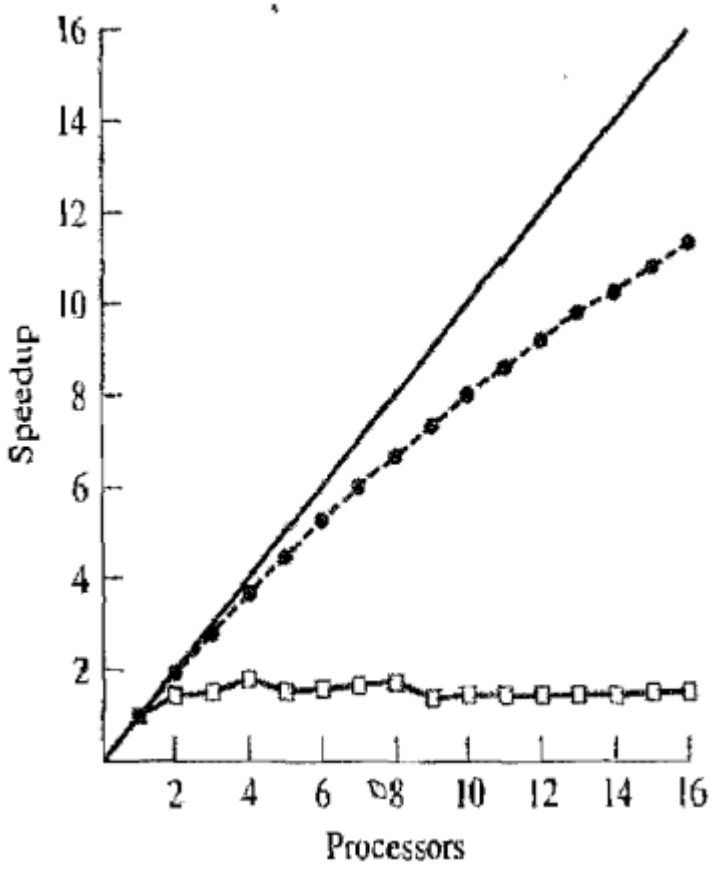
- Amdahl's Law ignores overhead associated with the introduction of parallelism
- Example
  - Let's return to our previous example. Suppose the parallel version of the program has $\lceil \log n \rceil$ communication points. At each of these points, the communication time is $10000\lceil \log p \rceil + \left(\frac{n}{10}\right)\mu sec$.

For a problem size of 10000, the total communication time is $14(10000\lceil \log p \rceil + 1000)\mu sec$.

$$speedup = \frac{sequentail\ portion + parallizable\ portion}{sequential\ portin + \frac{parallizable}{p} + parallel\ overhead}$$

$$= \frac{(28000 + 1000000)\mu sec}{28000 + \frac{1000000}{p} + 140000\lceil \log p \rceil + 14000}$$

Amdahl's Law - Limitations

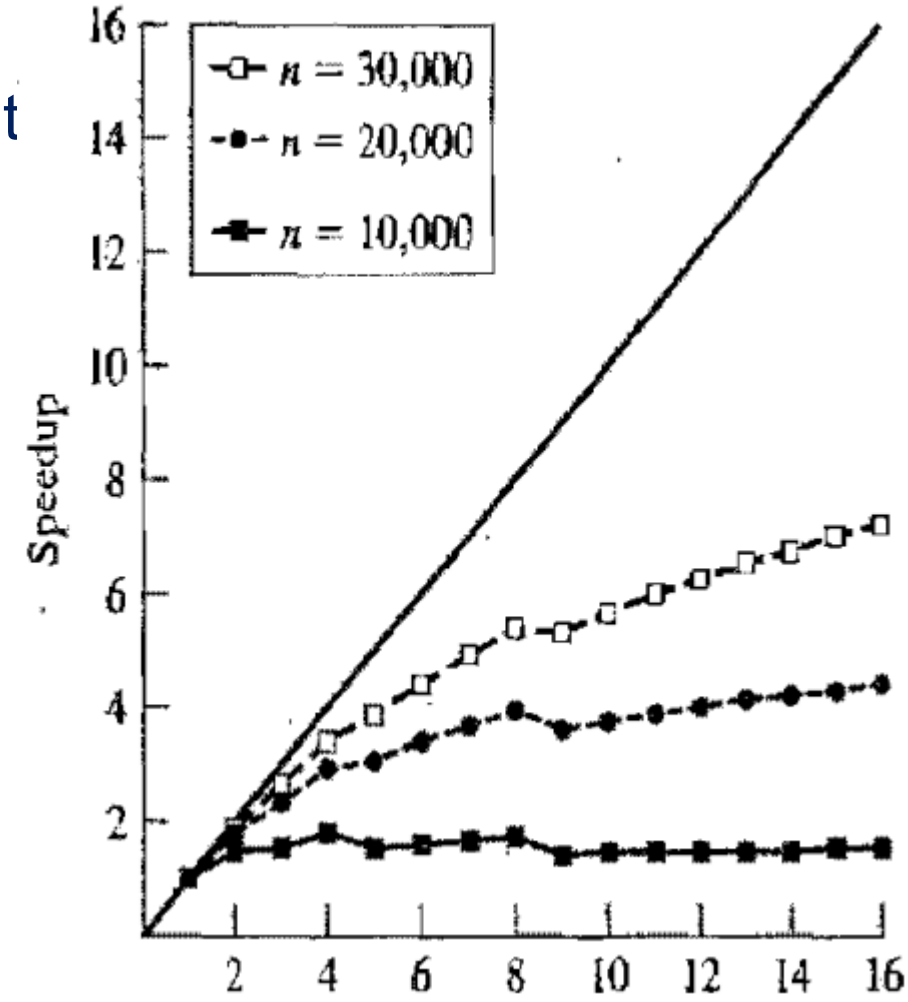


Speedup predicted by Amdahl's Law (dashed line) is higher than speedup prediction that takes communication overhead into account (solid line with boxes).

The Amdahl Effect

BITS Pilani

- Increasing the size of the problem increases the computation time faster than it increases the communication time. Hence for a fixed number of processors speedup is usually an increasing function of the problem size.
- Example
  - Communication overhead: $\theta(nlogn + nlogp)$
  - Parallizable portion=$\theta(n^2)$



Gustafson's Law

BITS Pilani

- Fix execution of a program on a single processor as
  - s + p = serial part + parallelizable part = 1
- Amdahl's law
  - $S(n) = (s + p)/(s + \frac{p}{n})$
    - n is the number of processors
  - $S(n) = \frac{1}{s+\frac{1-s}{n}} = \frac{n}{n*s+(1-s)} = \frac{n}{1+s*(n-1)}$
- Gustafson's Law
  - Now let, $s + \pi = 1$ = execution time on a parallel computer, with π = parallel part.
  - Scaled Speed up=
  - $Ss(n) = \frac{s+\pi*n}{s+\pi}$
  - $Ss(n) = s + (1 - s) * n = s + n - n * s = n + (1 - n) * s$

Gustafson's Law - Example

- Suppose that a program has a 14% serial code, what is the speedup if 24 processors are used? What is the maximum speedup that could be achieved?
- $Ss(n) = s + (1 - s) * n$
  - =0.14+(1-0.14)*24= 20.78 (5.68 in Amdahl's law)
- Parallelization efficiency remains the same
  - E(12)=10.46/12=.87
  - E(24)=20.78/24=.86
  - E(100)=86.14/100=.86
  - E(1000)=860.14/1000=.86

Amdahl's Law vs Gustafson's Law

- The scaled speedup is calculated based on the amount of work done for a scaled problem size
  - in contrast to Amdahl's law which focuses on fixed problem size
- Gustafson observed that
  - the serial part decreases with when problem size increases
  - in practice the sizes of problems scale with the amount of available resources
  - the parallel part scales linearly with the amount of resources
    - Example: Applications when they find more processing power, increase their parallelizable part. Browsers make more connections in parallel.

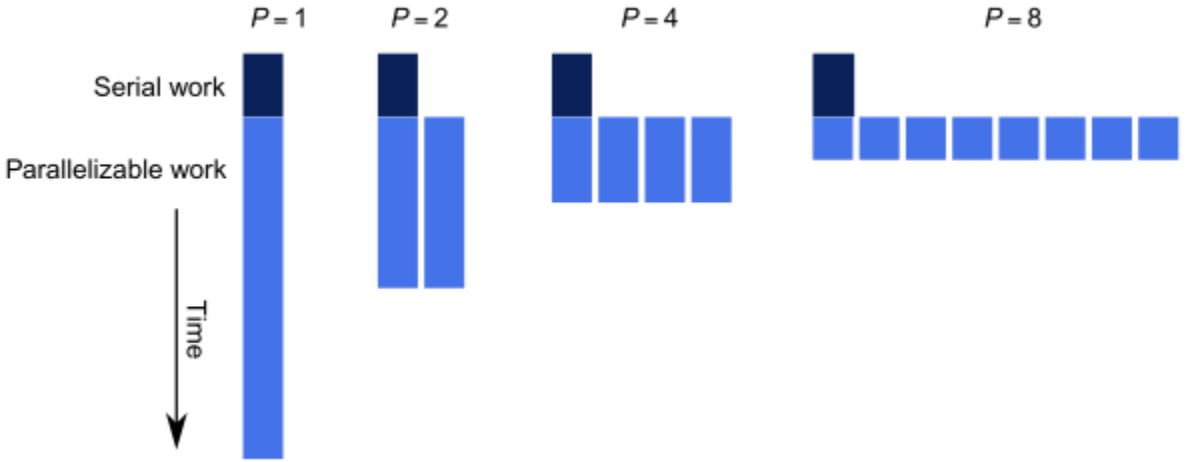
Amdahl's Law vs Gustafson's Law

- $S(n) = \frac{1}{s+\frac{1-s}{n}}$
  - Amdahl's law states that, for a fixed problem, the upper limit of speedup is determined by the serial fraction of the code
    - This is called strong scaling
- $Ss(n) = s + (1 - s) * n$
  - With Gustafson's law the scaled speedup increases linearly with respect to the number of processors, and there is no upper limit for the scaled speedup
    - This is called weak scaling

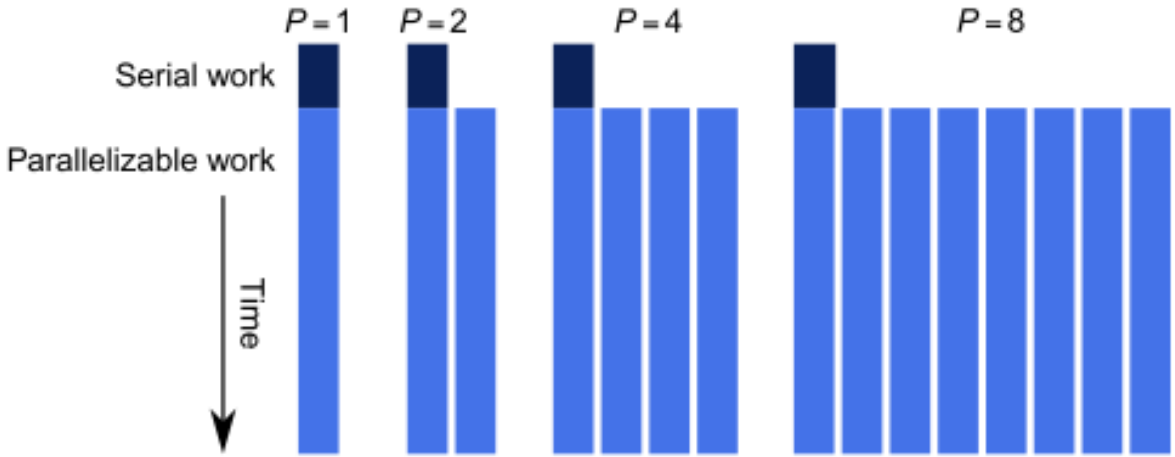
Amdahl's Law vs Gustafson's Law

BITS Pilani

- Amdahl's Law. Speedup is limited by the non-parallelizable serial portion of the work



- Gustafson-Barsis' Law. If the problem size increases with P while the serial portion grows slowly or remains fixed, speedup grows as workers are added



Amdahl's Law vs Gustafson's Law

- Both Amdahl's and Gustafson-Barsis' Laws are correct
  - It is a matter of "glass half empty" or "glass half full."
  - The difference lies in whether you want to make a program run faster with the same workload or run in the same time with a larger workload
- History clearly favors programs getting more complex and solving larger problems, so Gustafson's observations fit the historical trend
- Amdahl's Law still haunts you when you need to make an application run faster on the same workload to meet some latency target

The Karp-Flatt Metric

- Because Amdahl's Law and Gustafson-l3arsis's Law ignore the parallel overhead term, they can overestimate speedup or scaled speedup
- Karp and Flatt have proposed another metric, called the experimentally determined serial fraction, which can provide valuable performance insights
- Execution lime of a parallel program executing on p processors is defined as
  - $T(n,p) = \sigma(n) + \frac{\varphi(n)}{p} + k(n,p)$
  - where $\sigma(n)$ is the inherently serial component of the computation, $\varphi(n)$ is the portion of the computation that may be executed in parallel, and $k(n,p)$ is overhead resulting from processor communication and synchronization, and redundant computations.

The Karp-Flatt Metric

- The serial program does not have any interprocessor communication or synchronization overhead is
  - $T(n,1) = \sigma(n) + \varphi(n)$
- Experimentally determined serial fraction e of the parallel computation is
  - $e = \frac{\sigma(n)+k(n,p)}{T(n,1)}$. Hence $\sigma(n) + k(n,p) = T(n,1)e \rightarrow T(n,p) = T(n,p)e + \frac{T(n,1)(1-e)}{p}$
  - Let us use ψ as speedup. $\psi = \frac{T(n,1)}{T(n,p)} \rightarrow T(n,p) = T(n,p)\psi e + \frac{T(n,p)\psi(1-e)}{p}$
  - $\rightarrow 1 = \psi e + \frac{\psi(1-e)}{p} \Rightarrow \frac{1}{\psi} = e + \frac{1-e}{p} \Rightarrow \frac{1}{\psi} = e + \frac{1}{p} - \frac{e}{p} \Rightarrow \frac{1}{\psi} = e\left(1 - \frac{1}{p}\right) + \frac{1}{p} \Rightarrow \boldsymbol{e = \frac{\frac{1}{\psi} - \frac{1}{p}}{1 - \frac{1}{p}}}$

The Karp-Flatt Metric

- The experimentally determined serial fraction is a useful metric for two reasons
  - First, it rakes into account parallel overhead that Amdahl's Law and Gustafson-Barsis's Law ignore
  - Second, it can help us detect other sources of overhead or inefficiency that are ignored in our simple model of parallel execution time
- For a problem of fixed size, the efficiency of a parallel computation decreases as the number of processors increases
- By using the experimentally determined serial fraction, we can determine whether this efficiency decrease is due to
  - limited opportunities for parallelism or
  - increases in algorithmic or architectural overhead.

The Karp-Flatt Metric

BITS Pilani

- Considering speedup for various values of p, determine e and judge whether overhead indicates inherently sequential computation or communication or algorithmic overhead

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

p	2	3	4	5	6	7	8
ψ	1.82	2.50	3.08	3.57	4.00	4.38	4.71
e	0.10	0.10	0.10	0.10	0.10	0.10	0.10

- Since the experimentally determined serial fraction is not increasing with the number of processors, the primary reason for the poor speedup is the limited opportunity for parallelism-that is, the large fraction of the computation that is inherently sequential

The Karp-Flatt Metric

- Considering speedup for various values of p, determine e and judge whether overhead indicates inherently sequential computation or communication or algorithmic overhead

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

p	2	3	4	5	6	7	8
ψ	1.87	2.61	3.23	3.73	4.14	4.46	4.71
e	0.070	0.075	0.080	0.085	0.090	0.095	0.10

- Since the experimentally determined serial fraction is steadily increasing as the number of processors increases, the principal reason for the poor speedup is parallel overhead. This could be time spent in process startup, communication, or synchronization.

References

- Chapter 7 of “Parallel Programming in C with MPI and OpenMP” by Michael J. Quinn, McGraw Hill Education (India) dition 2003
- http://www.johngustafson.net/pubs/pub13/amdahl.htm

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You