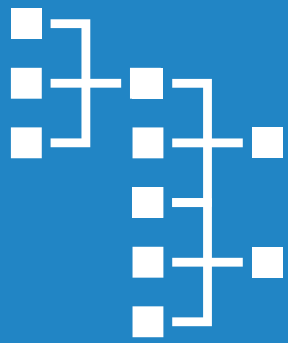
Yogesh Simmhan
simmhan@iisc.ac.in

Department of Computational and Data Sciences
Indian Institute of Science, Bangalore

DS256 (3:1)

Scalable Systems for Data Science



Distributed Systems

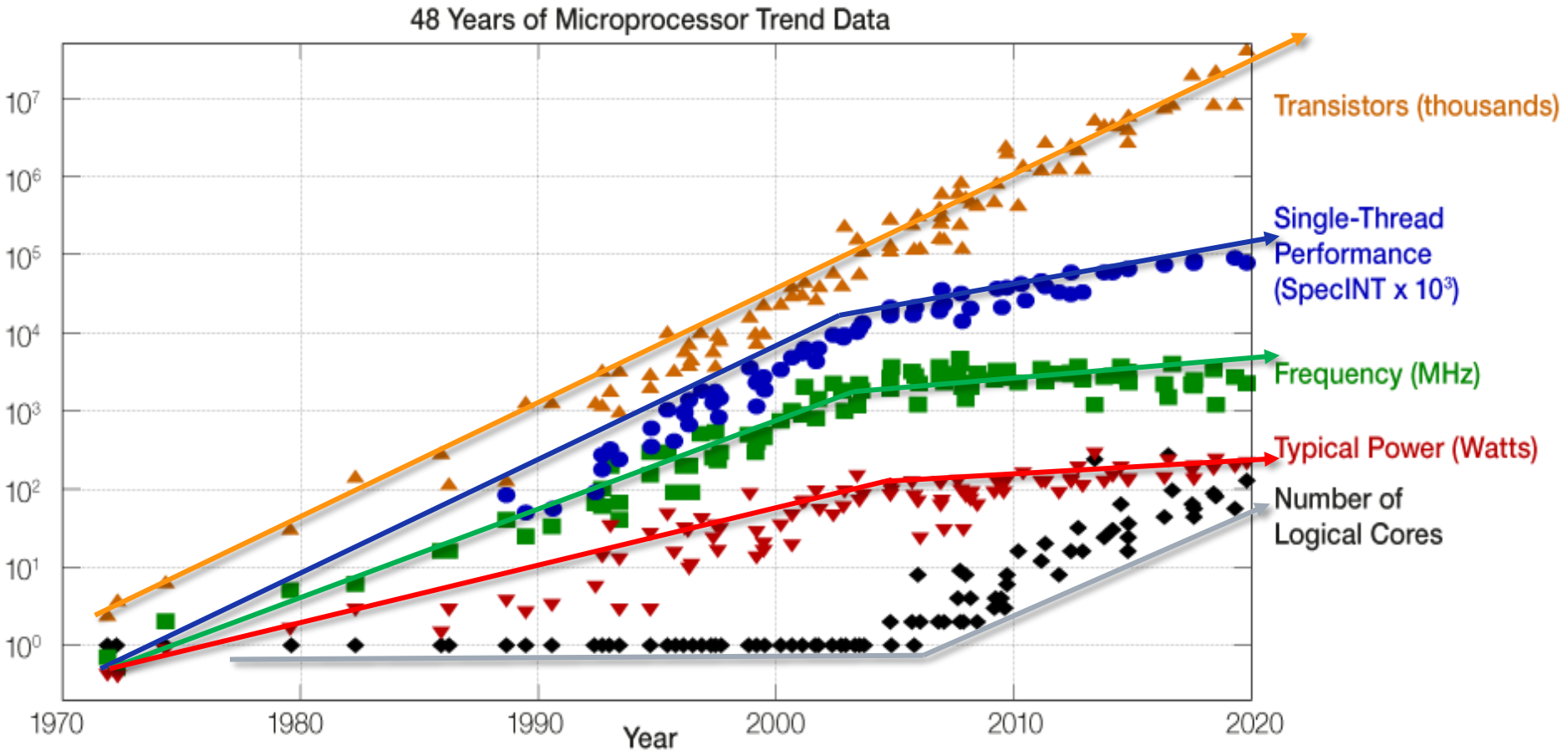
Helping Data Science Scale

Vertical and Horizontal Scaling

- *How do you get getter performance for an application?*
  - Better algorithm
  - Better programming
  - Better hardware
- **Vertical Scaling (Scale up)**
  - Adding more hardware resources to an existing machine
    - Faster CPU, more cores, more memory
  - Simpler. Little to no (multi-threaded or loosely coupled) application constraint
  - Upper bound of hardware limits
- **Horizontal Scaling (Scale Out)**
  - Adding more number of machines
  - Cumulatively more number of CPU cores, memory
  - Applications need to be designed to use multiple machines

Moore's Law (2x Trans. in 3y)
Dennard Scaling (2x Perf/Watt in 1.5y

Scale Up *(Faster CPUs/Clockspeed)*

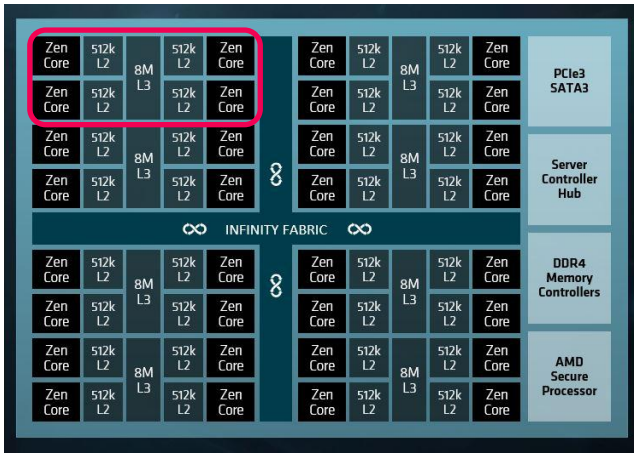


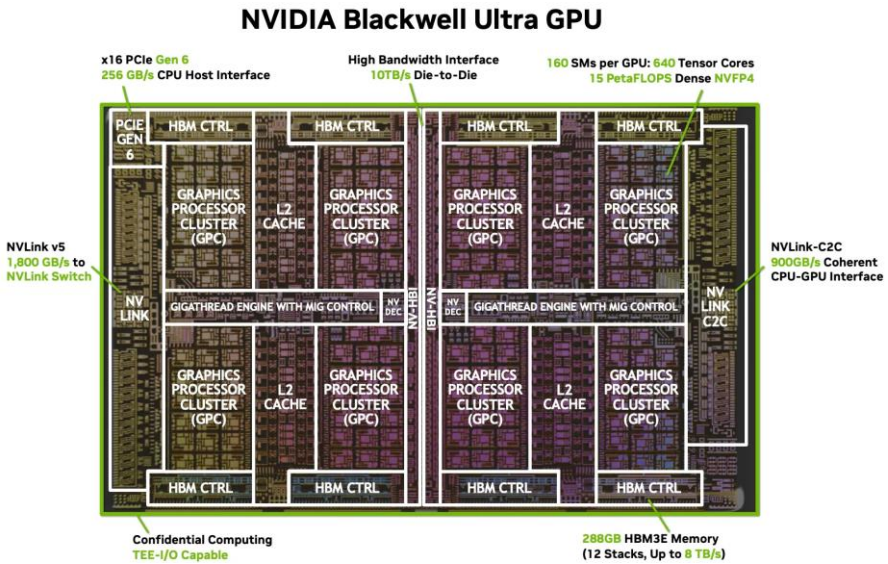
https://www.man.com/technology/single-core-stagnation-and-the-cloud

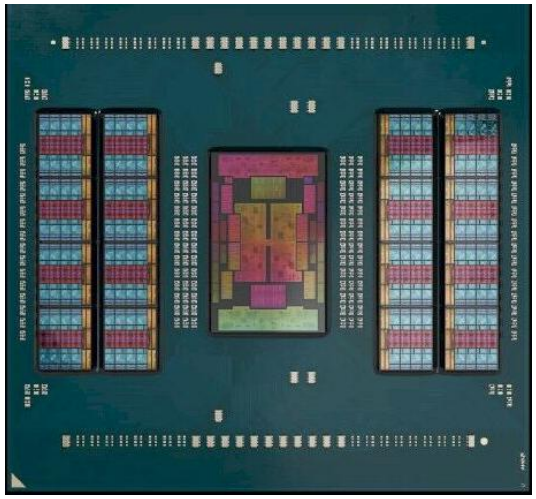
Scale Up *(More CPU cores)*

- **Multi/Many-core processing**
  - Multiple *cores* on single machine
    - Shared local memory space
    - Cores may be heterogeneous
  - Independent execution on each core
    - *A single thread/process does not run faster*
  - Multiple processes on a desktop/laptop
  - Multi-threaded applications on a server

AMD EPYC Zen





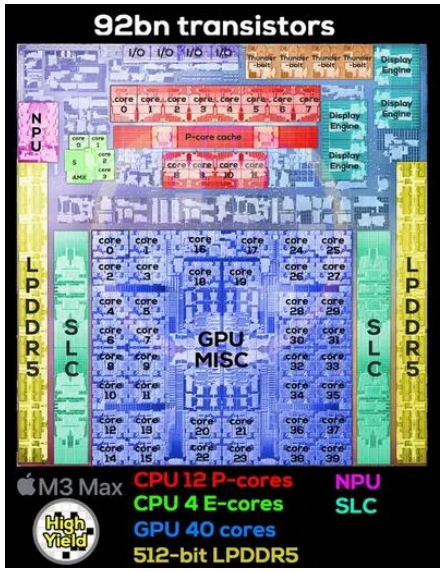


AMZ Epyc Zen5 Turin

https://www.nextplatform.com/2017/06/20/competition-returns-x86-servers-epyc-fashion/
https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/arm-dynamiq-technology-for-the-next-era-of-compute
https://www.tomshardware.com/news/annotated-apple-m3-processor-die-shots-bring-chip-designs-to-life
https://developer.nvidia.com/blog/inside-nvidia-blackwell-ultra-the-chip-powering-the-ai-factory-era/
https://www.nextplatform.com/2024/10/10/amd-turns-the-screws-with-turin-server-cpus/

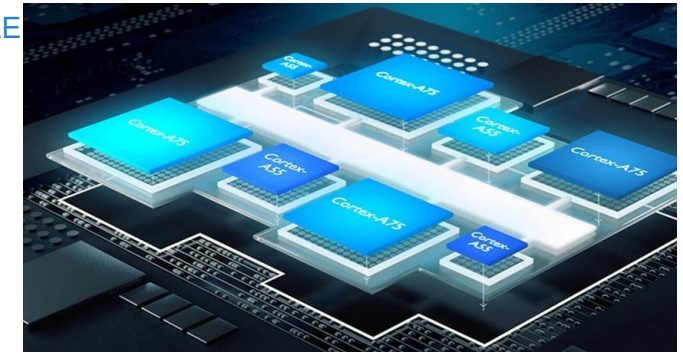
Scale Up (SoC Heterogeneous cores)

- Has a flavor of scale out
  - Applications may need to be redesigned to exploit parallelism
  - May be able to increase to 100s of cores



Apple M3

ARM big.LITTLE



https://www.nextplatform.com/2017/06/20/competition-returns-x86-servers-epyc-fashion/
https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/arm-dynamiq-technology-for-the-next-era-of-compute
https://www.tomshardware.com/news/annotated-apple-m3-processor-die-shots-bring-chip-designs-to-life
https://developer.nvidia.com/blog/inside-nvidia-blackwell-ultra-the-chip-powering-the-ai-factory-era/
https://www.nextplatform.com/2024/10/10/amd-turns-the-screws-with-turin-server-cpus/

Scale Out

- More processing units, working *collaboratively*

- **Distributed Systems**
  - Multiple servers, connected over the network
  - Distributed *Computing*
    - Clusters of machines, with CPU/GPU/Memory
    - Distributed memory space
  - Distributed *Storage*
    - Disks attached to clusters of machines
    - Network Attached Storage

- *How can we make effective use of multiple machines?*

Distributed Systems

Sirius cluster@DREAM:Lab
CDS 2nd Floor Data Center

- **Commodity** clusters & **Clouds**
  - Available off the shelf at large volumes, lower cost
  - Virtualized resources, modest performance
  - "Low speed" Ethernet: Higher latency, lower bandwidth
  - Loosely coupled applications

Param Pravega cluster@SERC
Ground Floor

- **HPC** clusters and **Supercomputers**
  - High-end CPUs
  - Fast network interconnect. Low latency, high bandwidth.
  - Tightly coupled applications

How many cores? PF?

30k Cores 3.5PF

- *How can we make effective use of many machines of modest capability?*

Cloud Computing

- Large Data Centers
  - 1000's of racks, 100k servers
- Virtualized infrastructure
  - On-demand: Rent VMs by the minute
  - 100 machines for 10mins costs same as 1 machine for 1000 minutes
- Makes use of economies of scale. Much cheaper than on-premises servers.
  - Reduce operations costs (energy, personnel)
  - Ensures capital costs (servers) are fully used

Ashburn, Virginia

https://www.ciodive.com/news/cloud-ai-data-center-aws-microsof-google-oracle/743218/

Designing applications for distributed systems

- Software is critical to enable use of distributed computing and storage
  - System software for resource management
  - Data management systems
  - Distributed programming models for designing apps
  - Runtime environments to execute these apps

- Question: How do you design a (C++/Java/Python) program to add a billion numbers, from scratch?

Scalability and Parallel Speedup

Jun and Ni, JPDC 1993

- **Speedup**: Performance measure for parallel processing
  - Ratio of execution time on a *uniprocessor* and on a *parallel processor*
- Different variations of Speedup, depending on the *assumptions* and *constraints* on application and system
  - Fixed problem size
  - Fixed time for processing
  - Fixed amount of memory
  - Effect of uneven workload
  - Effect of communication overhead

Scalable Multi Processor

- Scalable Multi Processor
  - As number of processors increase, the amount of *memory* and the *network bandwidth* also increases
  - Homogeneous processors
  - Shared or distributed memory
  - Different network topologies
  - Message passing or share memory programming paradigms

Speedup on *N* processors

- W is amount of **work** for application
- $T_i\ (W)$ is **time** to complete W amount of **work** on i **processors**
- Speedup on $\boldsymbol{N}$ processors with $\boldsymbol{W}$ amount of work is:

$$S_N(W) = \frac{T_1(W)}{T_N(W)}$$

Degree of Parallelism for an Application

- Maximum number of processors that can be actively processing at a given phase of the application
- Amount of parallel processing that can be exploited by the application during its execution lifetime
  - Load can be uneven over its lifetime
- Parallelism Profile and Shape
  - *For each degree of parallelism, what's the duration?*

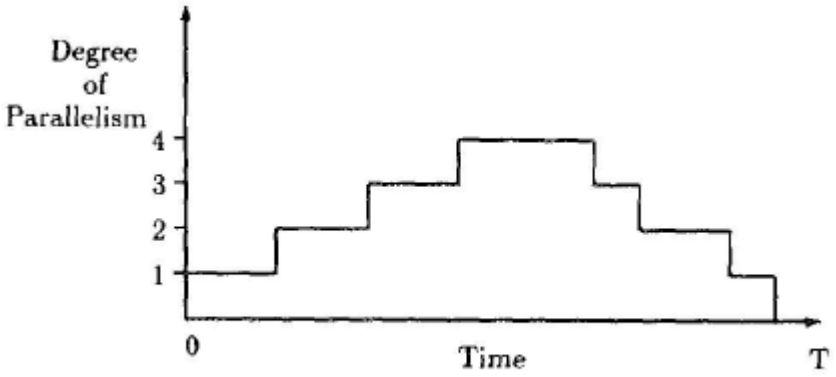


FIG. 1. Parallelism profile of an application.

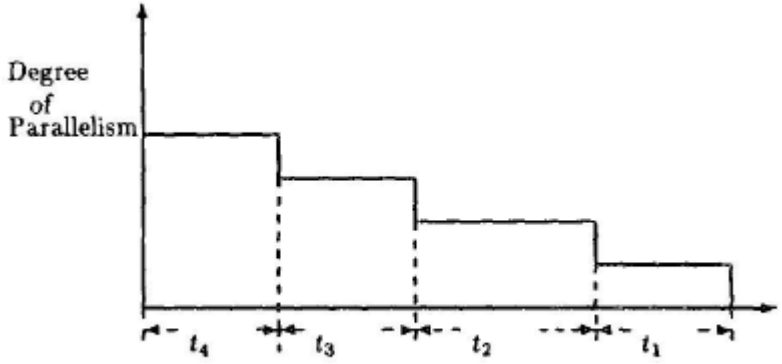
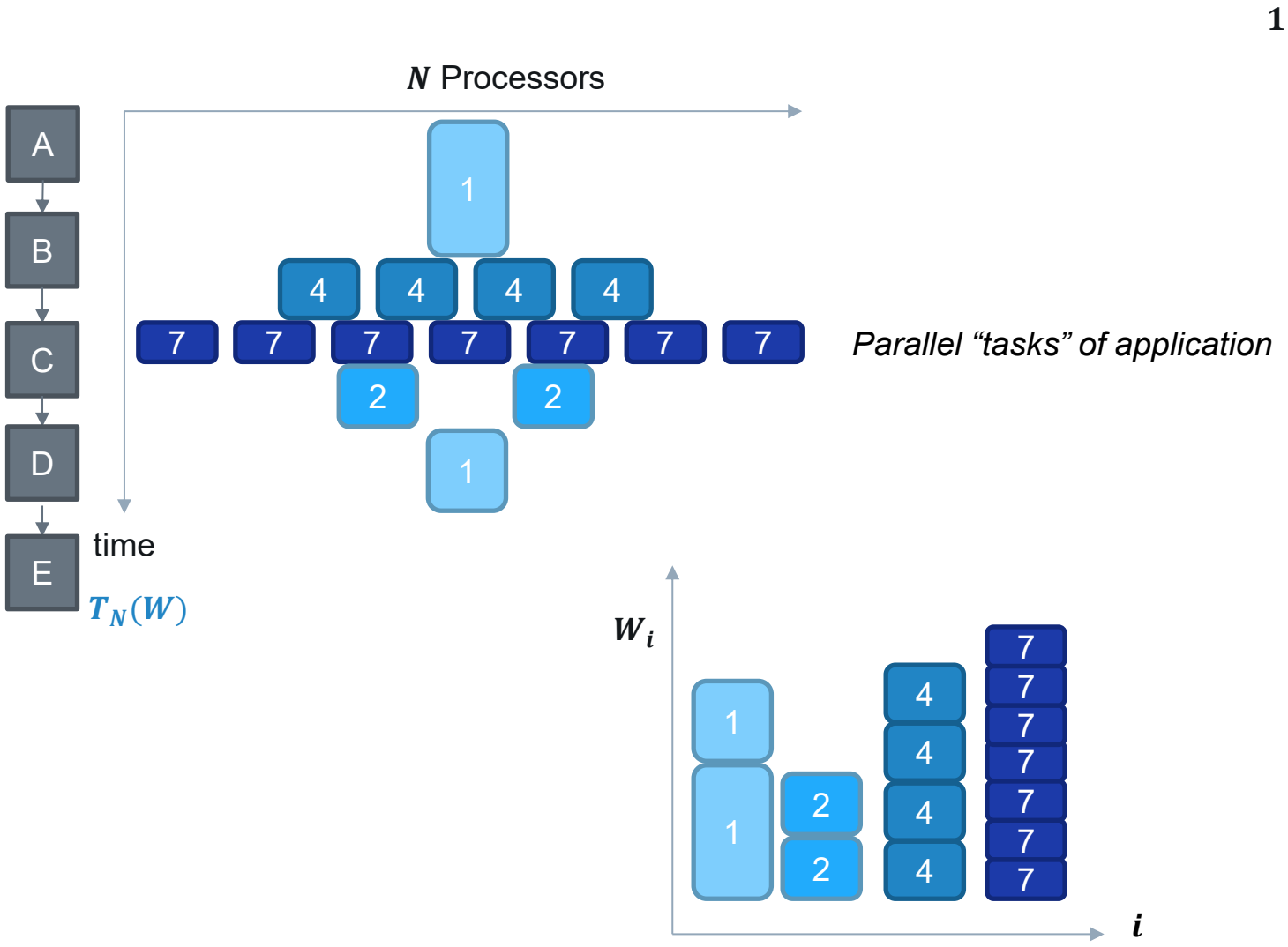


FIG. 2. Shape of the application.

Uneven Workloads: Different degrees of parallelism for different phases



Parallel Speedup: **One** processor with **uneven** load, no communication

- W_i is amount of work available with a degree of parallelism i
  - Across all tasks of the application
- m is maximum degree of parallelism for application
- Total work for application is:

$$W = \sum_{i=1\ to\ m} W_i$$

- Δ is computing capacity of one processor
- Execution time of W_i on a single processor is $t_1(W_i) = \frac{W_i}{\Delta}$
- Total execution time of W on a single processor is:

$$T_1(W) = \sum_{i=1\ to\ m} t_1(W_i) = \sum_{i=1\ to\ m} \frac{W_i}{\Delta}$$

Parallel Speedup: **Infinite** processors with **uneven** load, no communication

- With i processors, the execution time for W_i is $t_i(W_i) = \frac{W_i}{i.\Delta}$
- With ∞ processors, the execution time for W_i still remains:

$$t_\infty(W_i) = \frac{W_i}{i.\Delta}, 1 \leq i \leq m$$

  - *Parallelism benefits accrue only up to the degree of parallelism that is available for a task*

- Total execution time of W on ∞ processors is:

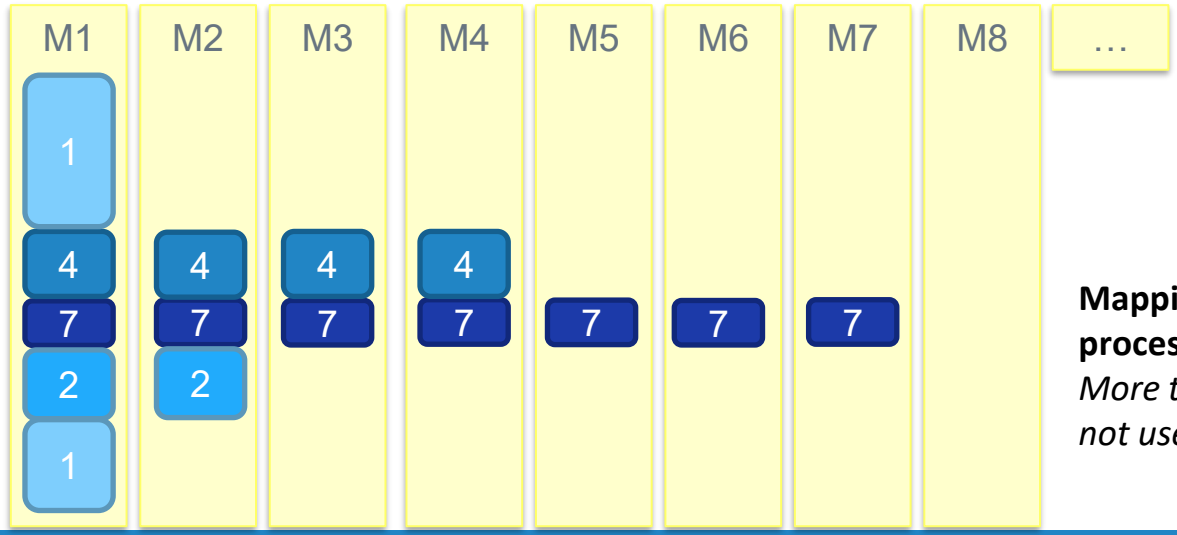
$$T_\infty(W) = \sum_{i=1\ to\ m} t_\infty(W_i) = \sum_{i=1\ to\ m} \frac{W_i}{i.\Delta}$$

Parallel Speedup: Infinite processors with **uneven** load, no communication

- Maximum speedup with work W and ∞ processors:

$$S_\infty(W) = \frac{T_1(W)}{T_\infty(W)} = \frac{\sum_{i=1\ to\ m} \frac{W_i}{\Delta}}{\sum_{i=1\ to\ m} \frac{W_i}{i.\Delta}} = \frac{\sum_{i=1\ to\ m} W_i}{\sum_{i=1\ to\ m} \frac{W_i}{i}}$$

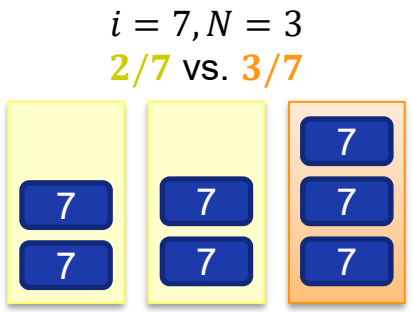
- S_∞ is the **best possible speedup** based on inherent parallelism of the application
  - Machine-dependent factors are ignored



Mapping "tasks" to infinite processors
More than 7 processors is not useful for m=7

Parallel Speedup: **N** processors with **uneven** load, no communication

- With N processors and i degree of parallelism, the load on each processor will be different if $N < i$
  - $W_i \cdot \frac{\lfloor \frac{i}{N} \rfloor}{i}$, for less loaded ones
  - $W_i \cdot \frac{\lceil \frac{i}{N} \rceil}{i}$, for more loaded ones



- Time elapsed for W_i will be limited by the **slowest** of these
- With N processors, the execution time for W_i is:

$$t_N(W_i) = \frac{W_i}{i.\Delta} \cdot \left\lceil \frac{i}{N} \right\rceil$$

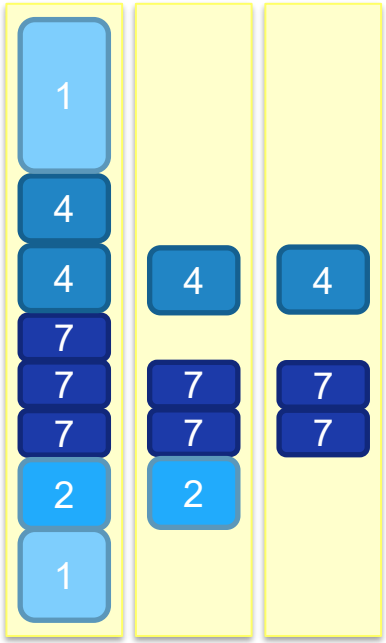
- Total execution time of W on N processors is:

$$T_N(W) = \sum_{i=1\ to\ m} \frac{W_i}{i.\Delta} \cdot \left\lceil \frac{i}{N} \right\rceil$$

Parallel Speedup: **N** processors with **uneven** load, no communication

- Speedup with work W and N processors is:

$$S_N(W) = \frac{T_1(W)}{T_N(W)} = \frac{\sum_{i=1\ to\ m} W_i}{\sum_{i=1\ to\ m} \frac{W_i}{i} \cdot \left\lceil \frac{i}{N} \right\rceil}$$



Mapping "tasks" to N=-3 processors

Next task phase can start only after previous phase has completed

***Note**: This model consolidates phases with the same degree of parallelism, i.e., W_i across all phases is consolidated and run in parallel. So its an approximation.*

Parallel Speedup: **N** processors with **uneven** load, **with** communication

- Communication latency is machine dependent, and application dependent
  - NW topology, routing, etc.
- If the communication cost for N processors to complete W work is $Q_N(W)$
- Assuming degree of parallelism is not affected by communication costs
- Speedup with work W and N processors is:

$$S_N(W) = \frac{T_1(W)}{T_N(W)} = \frac{\sum_{i=1\ to\ m} W_i}{\left(\sum_{i=1\ to\ m} \frac{W_i}{i} \cdot \left\lceil \frac{i}{N} \right\rceil\right) + Q_N(W)}$$

Parallel Speedup with Constraints: **Fixed Problem Size**

- Fixed problem size constraint
  - Size of the application is constant, W
    - Amount of work to be done is constant
    - E.g., number of iterations, size of the matrix, input file size
  - Need to complete the application as quickly as possible
  - *How does the **fixed-work speedup** vary with increasing the number of processors?*

- This is the default formulation, same as above:

$$S_N(W) = \frac{T_1(W)}{T_N(W)} = \frac{\sum_{i=1\ to\ m} W_i}{\left(\sum_{i=1\ to\ m} \frac{W_i}{i} \cdot \left\lceil \frac{i}{N} \right\rceil\right) + Q_N(W)}$$

Parallel Speedup with Constraints: **Fixed Time**

- Fixed time constraint
  - Time budget for processing the application is fixed
    - Usually set as the time taken for application of some size to complete on a single processor
  - Size of the application is increased with the increase in the number of processors
    - E.g., more iterations, larger matrix size, larger input file
  - Need to complete the scaled application within time budget
  - *How does the **fixed-time speedup** vary with increasing the number of processors?*

Parallel Speedup with Constraints: **Fixed Time**

- W' is the "scaled" amount of work for an application, when given N processors
- m' is the maximum degree of parallelism for the scaled problem
  - This can be different from m
- Total amount of scaled work is $W' = \sum_{i=1\ to\ m'} W_i'$
- For fixed-time, the constraint imposed is: $T_1(W) = T_N(W')$
  - *Time to run W on 1 processor is same as time to run W' on N processors*
- This is re-written as:

$$\sum_{i=1\ to\ m} W_i = \left(\sum_{i=1\ to\ m'} \frac{W_i'}{i} \cdot \left\lceil \frac{i}{N} \right\rceil \right) + Q_N(W')$$

Parallel Speedup with Constraints: **Fixed Time**

- Speedup with fixed-time is:

$$S_N(W') = \frac{T_1(W')}{T_N(W')} = \frac{T_1(W')}{T_1(W)} = \frac{\sum_{i=1\ to\ m'} W_i'}{\sum_{i=1\ to\ m} W_i}$$

- This is the time to complete the scaled problem on one processor divided by the non-scaled problem on one processor

Simplified Models of Speedup

- Negligible communication costs, $Q_N = 0$
- Workload only has a **sequential** and a **"perfectly parallel"** part, i.e., W_i is non-zero only for:
  - $i = 1$ (sequential)
  - $i = N$ (perfectly parallel)

- Sequential part is independent of system size
  - Remains constant even when problem is "scaled"
  - $W_1 = W_1'$

Simplified Models of Speedup: **Fixed Problem Size**

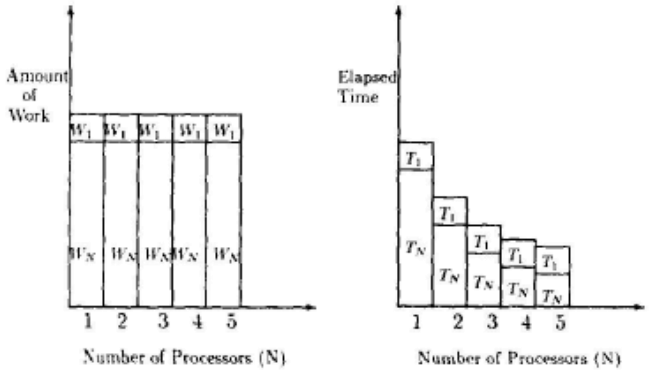


FIG. 3. Amdahl's law.

- Fixed-size speedup reduces to:

$$S_N(W) = \frac{\sum_{i=1\ to\ m} W_i}{\left(\sum_{i=1\ to\ m} \frac{W_i}{i} \cdot \left\lceil \frac{i}{N} \right\rceil\right) + Q_N(W)}$$

$$= \frac{W_1 + W_N}{W_1 + \frac{W_N}{N}}$$

- **Amdahl's Law of Parallel Speedup**
  - For large values of N, the sequential part will dominate the total time taken, $W_1 \gg \frac{W_N}{N}$
  - **Maximum Speedup** is bound by $\frac{W_1 + W_N}{W_1}$

Amdahl's Law of Strong Scaling

- Amdahl's Law for Application Scalability
  - Total problem size is fixed
  - *Speedup limited by sequential bottleneck*
- s is *serial* fraction of application, cannot be parallelized
- $(1 - s)$ is fraction of application that can be *parallelized*
- p is number of processors
- t is time taken to process input of size x on 1 processor
- Then the speedup is limited by the serial fraction

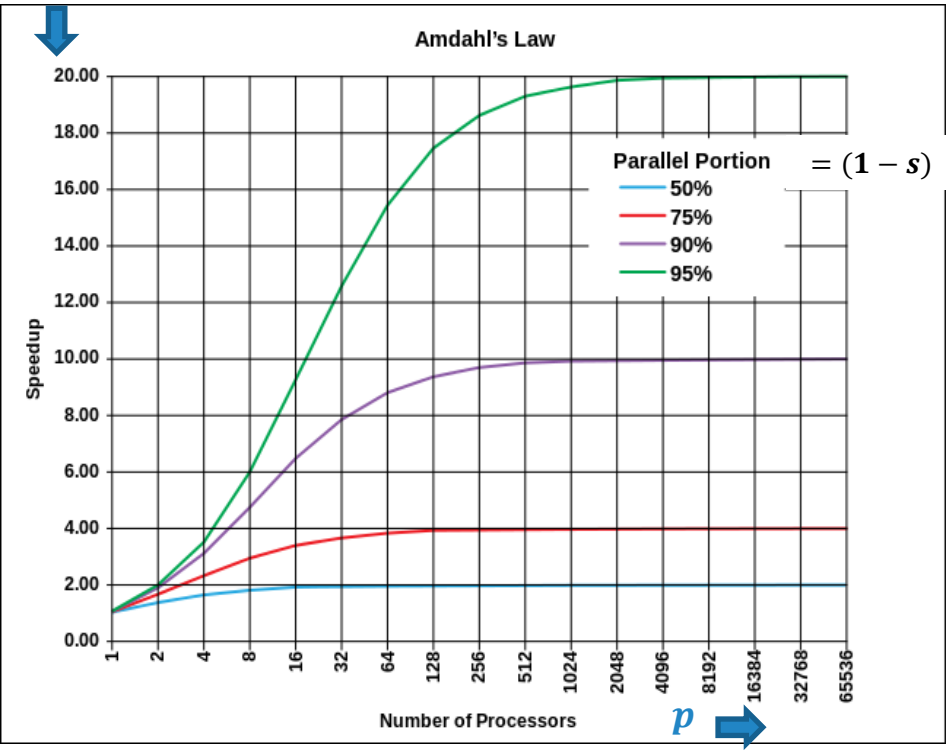
$$Speedup(p, x) = \frac{time(1,x)}{time(p,x)} = \frac{t}{s.t + \frac{(1-s)}{p}.t} = \frac{s.t + \frac{(1-s)}{1}.t}{s.t + \frac{(1-s)}{p}.t} = \frac{\mathbf{1}}{\boldsymbol{s + \frac{(1-s)}{p}}}$$

- For large values of N, the sequential part will dominate, and **maximum speed up** is $\frac{\mathbf{1}}{\boldsymbol{s + \frac{(1-s)}{p}}} \approx \frac{\mathbf{1}}{\boldsymbol{s}}$

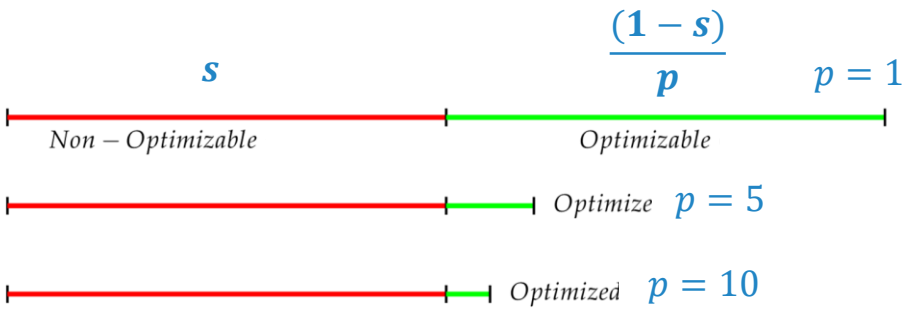
Amdahl's Law (Strong Scaling)

s is *serial* fraction of application, cannot be parallelized
$(1 - s)$ is fraction of application that can be *parallelized*

$$\frac{1}{s + \frac{(1-s)}{p}}$$



$= (1 - s)$



Meaningful only for small values of ***p***, small fractions of ***s***

Simplified Models of Speedup:
Fixed Time

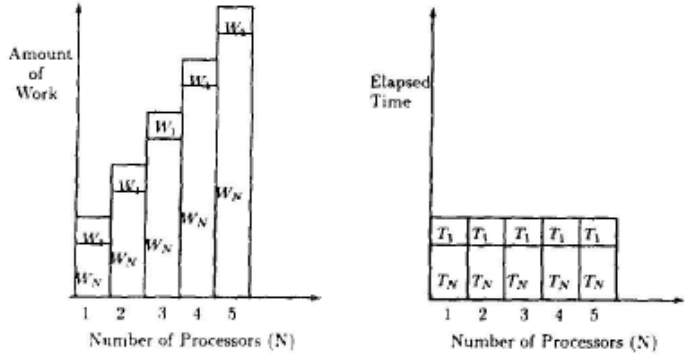


FIG. 4. Gustafson's scaled speedup.

- Fixed-time constraint becomes: $W_1 + W_N = W_1' + \frac{W_N'}{N}$
- Since sequential part is constant even with scaled problem $W_1 = W_1'$, we have $W_N = \frac{W_N'}{N}$, i.e., $W_N' = N \cdot W_N$
- Fixed-time speedup reduces to:

$$S_N(W') = \frac{W_1 + N \cdot W_N}{W_1 + W_N}$$

- **Gustafson's Law of Weak Scaling**

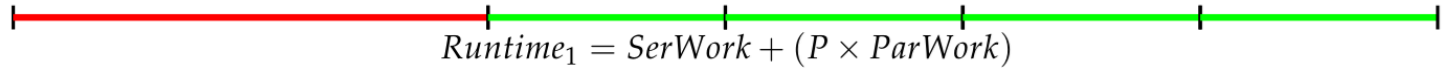
Gustafson's Law of weak scaling

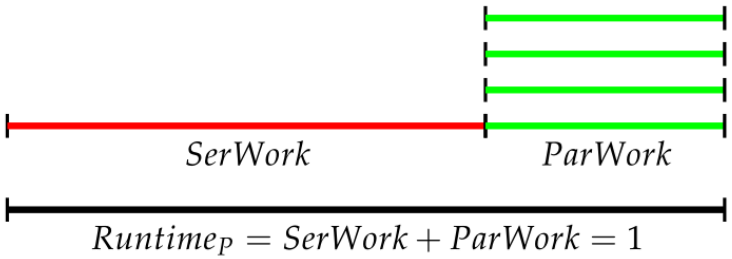
- Gustafson's Law of Application Scalability
  - Problem size (***p.x***) increases with number of processors (***p***)
  - "Scaled speedup"
- $Speedup(p, x.p) = \frac{time(1,p.x)}{time(p,p.x)} = \frac{s.p.t+(1-s).p.t}{s.p.t+\frac{(1-s).p.t}{p}}$
- If $s.p.t \approx s.t$ …i.e., only parallelizable fraction of input work increases with number of processors
- $Speedup(p, x.p) = \frac{s.t+(1-s).p.t}{s.t+\frac{(1-s).p.t}{p}} = \frac{s+(1-s).p}{s+(1-s)}$

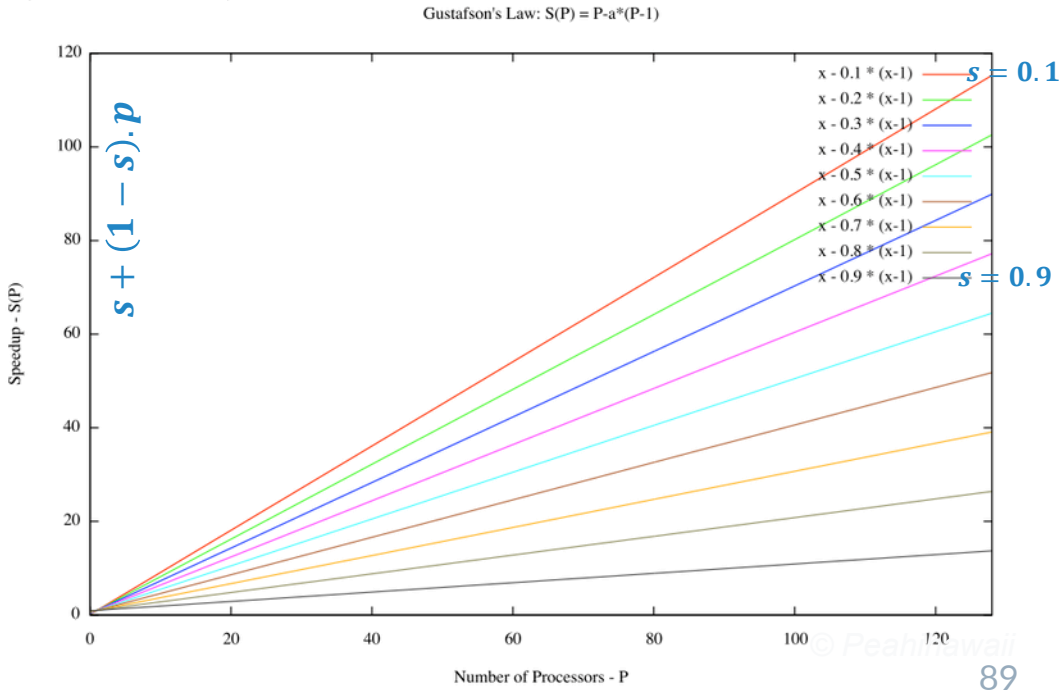
$$= \boldsymbol{s + (1 - s).p}$$

Gustafson's Law of weak scaling

- As input work increases, all incremental work can be parallelized. Total time remains constant as we increase number of processors.







Scalability

- **Speedup Efficiency**: *How good is the speedup relative to perfect linear speedup?*

$$Speedup\ Efficiency\ = \frac{Speedup(p, \_)}{p}$$

- **Strong Scaling**: How the performance varies with the # of processors for a *fixed total problem size*

- **Weak Scaling**: How the performance varies with the # of processors for a *fixed problem size per processor*
  - Big Data platforms are intended for "Weak Scaling"

Additional Reading

- Computer Architecture and Amdahl's Law, Gene M. Amdahl, *IEEE Computer*, 2013, doi:10.1109/MC.2013.418
- Reevaluating Amdahl's Law, Gustafson, John L., *Communications of the ACM*, 31 (5), 1988, doi:10.1145/42411.42415
- The evolution of distributed computing systems: from fundamental to new frontiers. Lindsay, D., Gill, S.S., Smirnova, D. et al. *Computing* (2021). https://doi.org/10.1007/s00607-020-00900-y
- Big Data computing and clouds: Trends and future directions, M. D. Assunção, et al., Journal of Parallel and Distributed Computing, Volumes 79–80, 2015, Pages 3-15, https://doi.org/10.1016/j.jpdc.2014.08.003

End of Lecture 2

- *REMINDER*
- *Read the GFS paper*