




058165 - PARALLEL COMPUTING

Fabrizio Ferrandi

a.a. 2025-2026



The background of the slide is a light gray gradient. It is decorated with several realistic water droplets of various sizes. Some droplets are large and prominent, while others are small and scattered. They are rendered with soft shadows and highlights, giving them a three-dimensional appearance. The droplets are primarily located in the upper right and lower right areas of the slide, leaving the left side relatively clear for the text.

PRAM ARCHITECTURES, ALGORITHMS, PERFORMANCE EVALUATION

ACKNOWLEDGE

- MATERIAL FROM:
 - PARALLEL COMPUTING LECTURES FROM PROF. RAN GINOSAR - TECHNION - ISRAEL INSTITUTE OF TECHNOLOGY
 - DAVID RODRIGUEZ-VELAZQUEZ - SPRING -09 CS-6260 - DR. ELISE DE DONCKER
 - JOSEPH F. JAJA, INTRODUCTION TO PARALLEL ALGORITHMS, 1992
WWW.UMIACS.UMD.EDU/~JOSEPH/
 - UZI VISHKIN, PRAM CONCEPTS (1981-TODAY)
WWW.UMIACS.UMD.EDU/~VISHKIN

OVERVIEW

- WHAT IS A MACHINE MODEL?
- WHY DO WE NEED A MODEL?
- RAM
- PRAM
 - STEPS IN COMPUTATION
 - WRITE CONFLICT
 - EXAMPLES

A PARALLEL MACHINE MODEL



What is a machine model?

Describes a “machine”

Puts a value to the operations on the machine



Why do we need a model?

Makes it easy to reason algorithms

Achieve complexity bounds

Analyzes maximum parallelism

RAM (RANDOM ACCESS MACHINE)

- **UNBOUNDED** NUMBER OF LOCAL MEMORY CELLS
- EACH MEMORY CELL CAN HOLD AN INTEGER OF **UNBOUNDED** SIZE
- INSTRUCTION SET INCLUDES SIMPLE OPERATIONS, DATA OPERATIONS, COMPARATOR, BRANCHES
- ALL OPERATIONS TAKE **UNIT TIME**
- **TIME COMPLEXITY** = NUMBER OF INSTRUCTIONS EXECUTED
- **SPACE COMPLEXITY** = NUMBER OF MEMORY CELLS USED

PRAM (PARALLEL RANDOM ACCESS MACHINE)

- DEFINITION:
 - IS AN ABSTRACT MACHINE FOR DESIGNING THE ALGORITHMS APPLICABLE TO PARALLEL COMPUTERS
 - M' IS A SYSTEM $\langle M, X, Y, A \rangle$ OF INFINITELY MANY
 - RAM'S M_1, M_2, \dots , EACH M_i IS CALLED A PROCESSOR OF M' . ALL THE PROCESSORS ARE ASSUMED TO BE IDENTICAL. EACH HAS ABILITY TO RECOGNIZE ITS OWN INDEX i
 - INPUT CELLS $X(1), X(2), \dots$,
 - OUTPUT CELLS $Y(1), Y(2), \dots$,
 - SHARED MEMORY CELLS $A(1), A(2), \dots$,

PRAM (PARALLEL RAM)

- UNBOUNDED COLLECTION OF RAM PROCESSORS $P_0, P_1, \dots,$
- PROCESSORS DON'T HAVE TAPE
- EACH PROCESSOR HAS UNBOUNDED REGISTERS
- UNBOUNDED COLLECTION OF SHARE MEMORY CELLS
- ALL PROCESSORS CAN ACCESS ALL MEMORY CELLS IN UNIT TIME
- ALL COMMUNICATION VIA SHARED MEMORY

PRAM (STEPS IN A COMPUTATION)

- CONSIST OF 5 PHASES (CARRIED IN PARALLEL BY ALL THE PROCESSORS) EACH PROCESSOR:
 - READS A VALUE FROM ONE OF THE CELLS $X(1), \dots, X(N)$
 - READS ONE OF THE SHARED MEMORY CELLS $A(1), A(2), \dots$
 - PERFORMS SOME INTERNAL COMPUTATION
 - MAY WRITE INTO ONE OF THE OUTPUT CELLS $Y(1), Y(2), \dots$
 - MAY WRITE INTO ONE OF THE SHARED MEMORY CELLS $A(1), A(2), \dots$

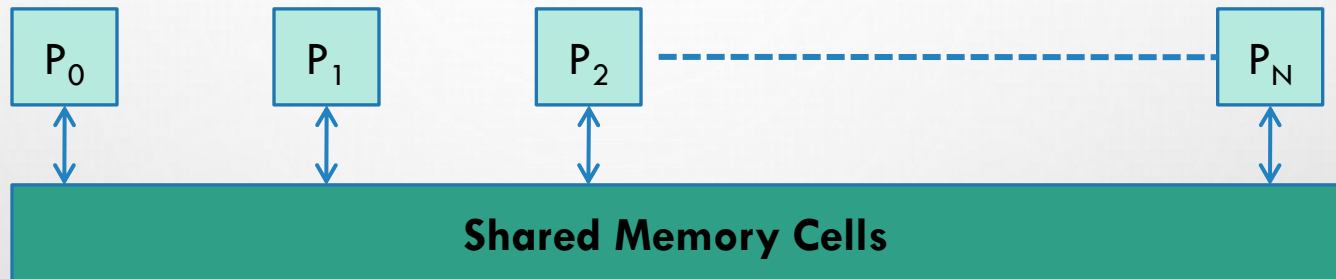
E.G. FOR ALL I , DO $A[I] = A[I-1] + 1$;

READ $A[I-1]$, COMPUTE ADD 1, WRITE $A[I]$

HAPPENED SYNCHRONOUSLY

PRAM (PARALLEL RAM)

- SOME SUBSET OF THE PROCESSORS CAN REMAIN IDLE



- Two or more processors may read simultaneously from the same cell
- A **write conflict** occurs when two or more processors try to write simultaneously into the same cell



SHARE MEMORY ACCESS CONFLICTS

- PRAM ARE CLASSIFIED BASED ON THEIR READ/WRITE ABILITIES (REALISTIC AND USEFUL)
 - EXCLUSIVE READ(**ER**) : ALL PROCESSORS CAN SIMULTANEOUSLY READ FROM DISTINCT MEMORY LOCATIONS
 - EXCLUSIVE WRITE(**EW**) : ALL PROCESSORS CAN SIMULTANEOUSLY WRITE TO DISTINCT MEMORY LOCATIONS
 - CONCURRENT READ(**CR**) : ALL PROCESSORS CAN SIMULTANEOUSLY READ FROM ANY MEMORY LOCATION
 - CONCURRENT WRITE(**CW**) : ALL PROCESSORS CAN WRITE TO ANY MEMORY LOCATION
 - EREW, CREW, CRCW

CONCURRENT WRITE (CW)

- WHAT VALUE GETS WRITTEN FINALLY?
 - **PRIORITY CW**: PROCESSORS HAVE PRIORITY BASED ON WHICH VALUE IS DECIDED, THE HIGHEST PRIORITY IS ALLOWED TO COMPLETE WRITE
 - **COMMON CW**: ALL PROCESSORS ARE ALLOWED TO COMPLETE WRITE **IFF** ALL THE VALUES TO BE WRITTEN ARE EQUAL. ANY ALGORITHM FOR THIS MODEL HAS TO MAKE SURE THAT THIS CONDITION IS SATISFIED. IF NOT, THE ALGORITHM IS **ILLEGAL** AND THE MACHINE STATE WILL BE UNDEFINED.
 - **ARBITRARY/RANDOM CW**: ONE RANDOMLY CHOSEN PROCESSOR IS ALLOWED TO COMPLETE WRITE

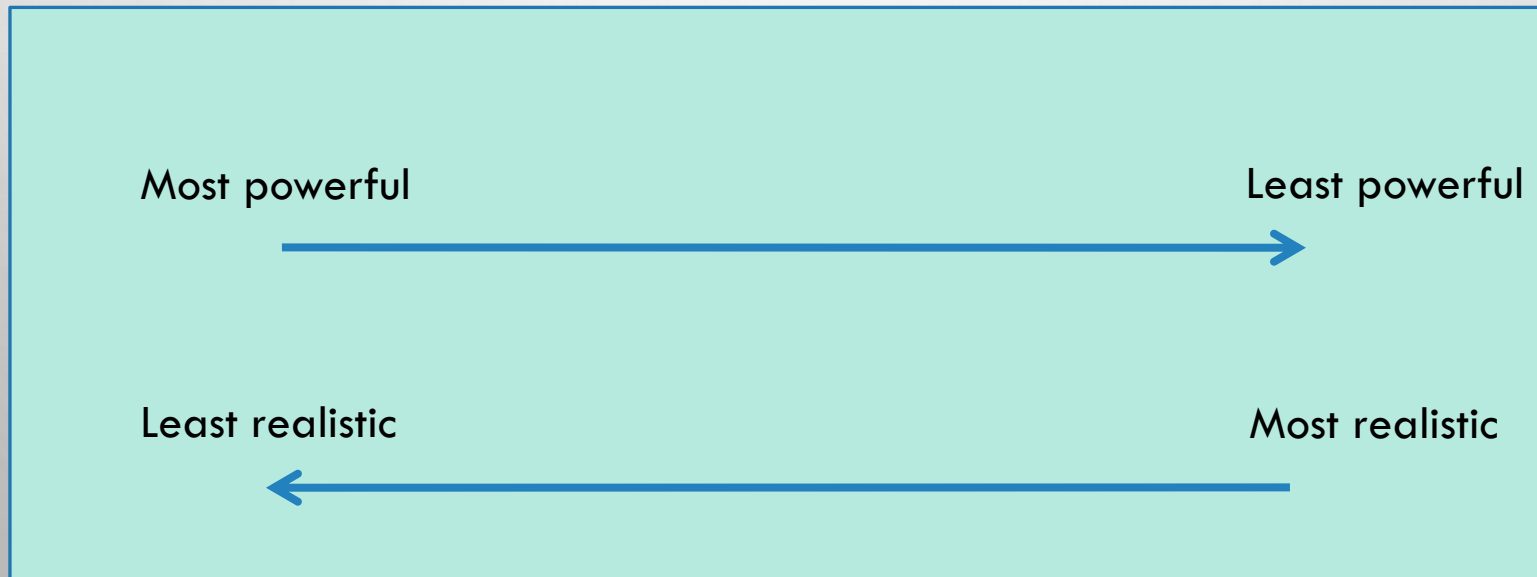
STRENGTHS OF PRAM

- PRAM IS ATTRACTIVE AND IMPORTANT MODEL FOR DESIGNERS OF PARALLEL ALGORITHMS WHY?
 - IT IS **NATURAL**: THE NUMBER OF OPERATIONS EXECUTED PER ONE CYCLE ON P PROCESSORS IS AT MOST P
 - IT IS **STRONG**: ANY PROCESSOR CAN READ/WRITE ANY SHARED MEMORY CELL IN UNIT TIME
 - IT IS **SIMPLE**: IT ABSTRACTS FROM ANY COMMUNICATION OR SYNCHRONIZATION OVERHEAD, WHICH MAKES THE COMPLEXITY AND CORRECTNESS OF PRAM ALGORITHM EASIER
 - IT CAN BE USED AS A **BENCHMARK**: IF A PROBLEM HAS NO FEASIBLE/EFFICIENT SOLUTION ON PRAM, IT HAS NO FEASIBLE/EFFICIENT SOLUTION FOR ANY PARALLEL MACHINE

COMPUTATIONAL POWER

- MODEL A IS COMPUTATIONALLY STRONGER THAN MODEL B ($A \geq B$) **IFF** ANY ALGORITHM WRITTEN FOR B WILL RUN UNCHANGED ON A IN THE SAME PARALLEL TIME AND SAME BASIC PROPERTIES.

PRIORITY \geq ARBITRARY \geq COMMON \geq CREW \geq EREW



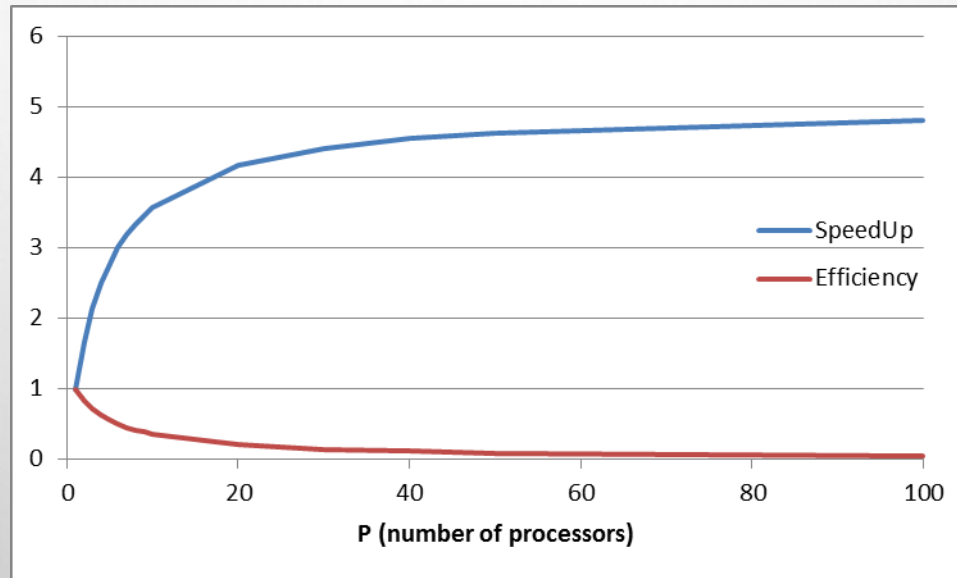
DEFINITIONS

$T^*(n)$	Time to solve problem of input size n on <u>one</u> processor, using best <u>sequential</u> algorithm
$T_p(n)$	Time to solve on p processors
$SU_p(n) = \frac{T^*(n)}{T_p(n)}$	Speedup on p processors
$E_p(n) = \frac{T_1(n)}{pT_p(n)}$	Efficiency (work on 1 / work that could be done on p)
$T_\infty(n)$	Shortest run time on any p
$C(n) = P(n) \cdot T(n)$	Cost (processors and time)
$W(n)$	Work = total number of operations

- $T^* \neq T_1$
- $SU_p \leq P$
- $SU_p \leq \frac{T_1}{T_\infty}$
- $E_p \leq 1$
- $T_1 \geq T^* \geq T_p \geq T_\infty$
- IF $T^* \approx T_1$, $E_p \approx \frac{T^*}{pT_p} = \frac{SU_p}{p}$
- $E_p = \frac{T_1}{pT_p} \leq \frac{T_1}{pT_\infty}$
 - NO USE MAKING P LARGER THAN MAX SU:
 - $E \rightarrow 0$, EXECUTION NOT FASTER
- $T_1 \in O(C)$, $T_p \in O(C/p)$
- $W \leq C$
- $p \approx \text{AREA}$, $W \approx \text{ENERGY}$,

$$\frac{W}{T_p} \approx \text{POWER}$$

SPEEDUP AND EFFICIENCY



Warning: This is only a (bad) example: An 80% parallel Amdahl's law chart.

We'll see why it's bad when we analyze (and refute) Amdahl's law. Meanwhile, consider only the trend.

EXAMPLE 1: MATRIX-VECTOR MULTIPLY

- $Y := AX \quad (n \times n, n) \quad A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{bmatrix}, \quad A_i \quad (r \times n)$
- $p \leq n, \quad r = n/p$
- EXAMPLE: $(256 \times 256, 256) \quad A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{32} \end{bmatrix}, \quad A_i \quad (8 \times 256)$
 - 32 PROCESSORS, EACH A_i BLOCK IS 8 ROWS
- PROCESSOR P_i READS A_i AND X , COMPUTES AND WRITES Y_i .
 - “EMBARRASSINGLY PARALLEL” – NO CROSS-DEPENDENCE

MVM ALGORITHM

i IS THE PROCESSOR INDEX

BEGIN

1. GLOBAL READ ($Z \leftarrow X$)
2. GLOBAL READ($B \leftarrow A_i$)
3. COMPUTE $W := BZ$
4. GLOBAL WRITE($W \rightarrow y_i$)

END

- STEP 1: CONCURRENT READ OF $X(1:N)$
 - BEST SUPPORTED BY B-CAST?
 - TRANSFER N ELEMENTS
- STEP 2: SIMULTANEOUS READS OF DIFFERENT SECTIONS OF A
 - TRANSFER n^2/p ELEMENTS TO EACH PROCESSOR
- STEP 3: COMPUTE
 - COMPUTE n^2/p OPS PER PROCESSOR
- STEP 4: SIMULTANEOUS WRITES
 - TRANSFER n/p ELEMENTS FROM EACH PROCESSOR
 - NO WRITE CONFLICTS

MVM ALGORITHM

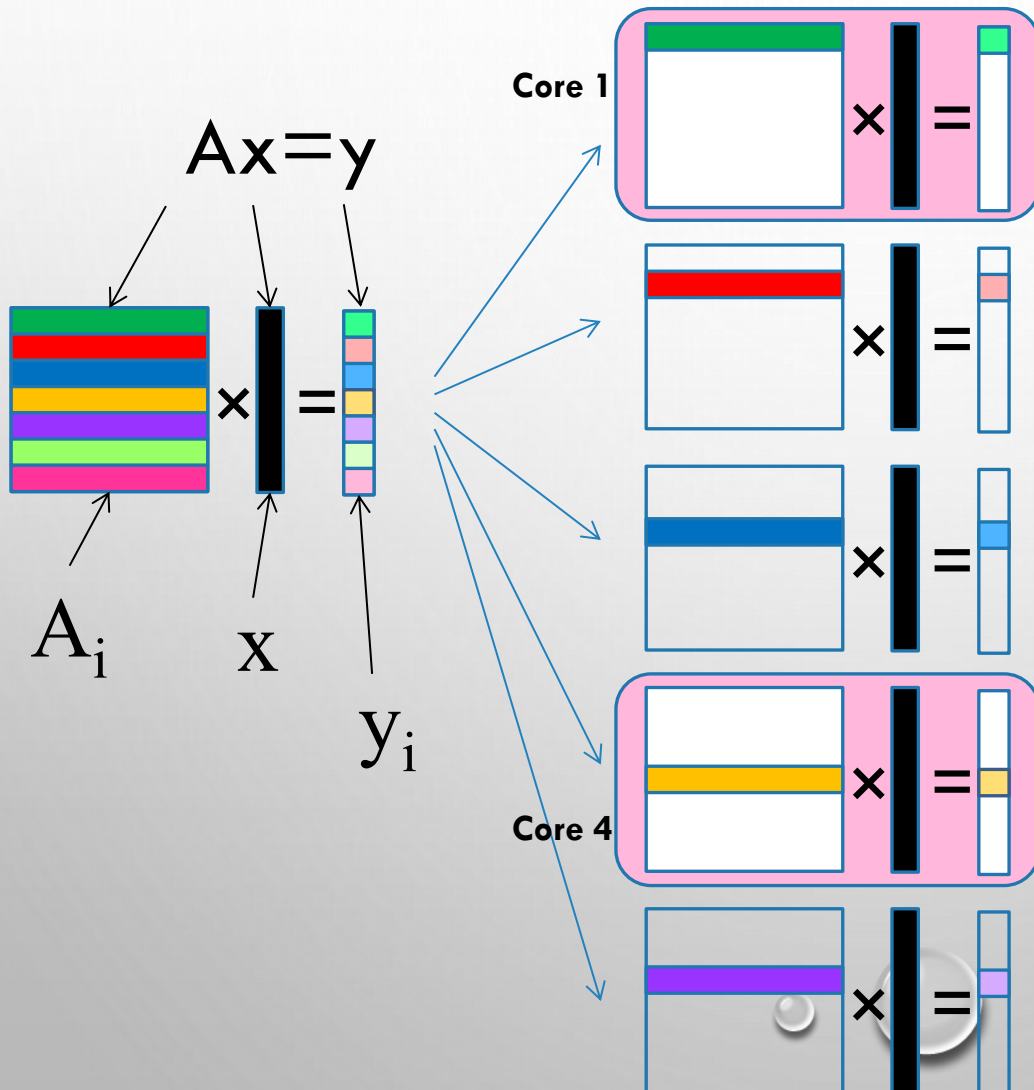
i IS THE PROCESSOR INDEX

BEGIN

1. GLOBAL READ ($Z \leftarrow X$)
2. GLOBAL READ($B \leftarrow A_i$)
3. COMPUTE $W := BZ$
4. GLOBAL WRITE($W \rightarrow y_i$)

END

MATRIX-VECTOR MULTIPLY



The PRAM algorithm

i is core index
AND slice index

Begin

$$y_i = A_i x$$

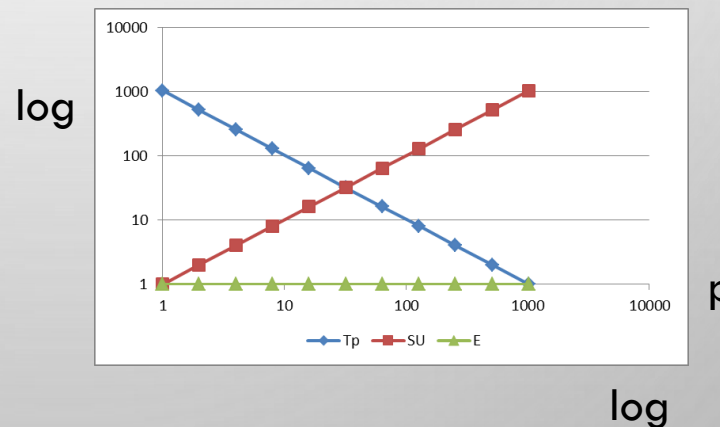
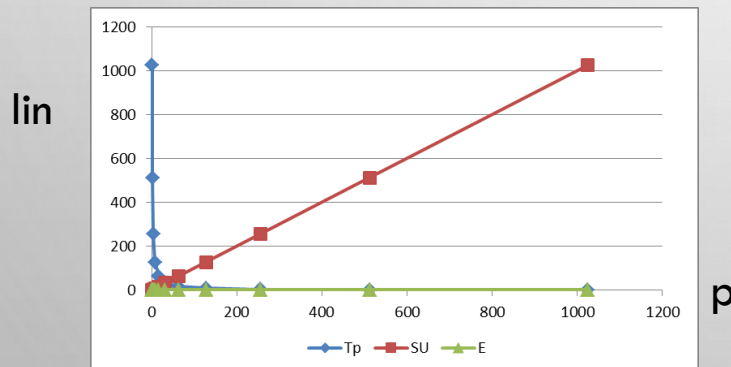
End

A, x, y in shared memory
(Concurrent Read of x)

Temp are in private memories ²⁰

PERFORMANCE OF MVM

- $T_1(N^2)=O(N^2)$
- $T_p(N^2)=O(N^2/P)$ --- LINEAR SPEEDUP, $SU=P$
- $COST=O(P \cdot N^2/P)=O(N^2)$,
- $W=C$, $W/T_p=P$ --- LINEAR POWER
- $E_p = \frac{T_1}{pT_p} = \frac{n^2}{pn^2/p} = 1$ --- PERFECT EFFICIENCY



EXAMPLE 2: SPMD SUM $A(1:N)$ ON PRAM

(GIVEN $n = 2^k$)

BEGIN

1. GLOBAL READ ($A \leftarrow A(I)$)
2. GLOBAL WRITE($A \rightarrow B(I)$)
3. FOR $H=1:K$
 - IF $i \leq n/2^h$ THEN BEGIN
 - GLOBAL READ($X \leftarrow B(2I-1)$)
 - GLOBAL READ($Y \leftarrow B(2I)$)
 - $Z := X + Y$
 - GLOBAL WRITE($Z \rightarrow B(I)$)
 - END
4. IF $I=1$ THEN GLOBAL WRITE($Z \rightarrow S$)

END

h	i	adding
1	1	1,2
	2	3,4
	3	5,6
	4	7,8
2	1	1,2
	2	3,4
3	1	1,2

LOGARITHMIC SUM

THE PRAM ALGORITHM

// SUM VECTOR A(*)

BEGIN

$B(i) := A(i)$

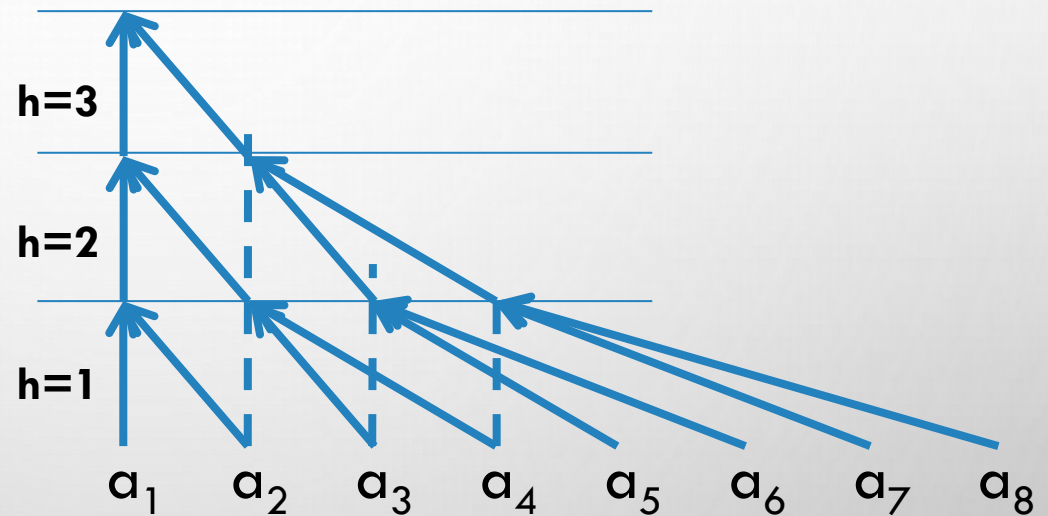
FOR $H=1:\text{LOG}(N)$

IF $i \leq n/2^h$ THEN

$B(i) = B(2i-1) + B(2i)$

END

// B(1) HOLDS THE SUM



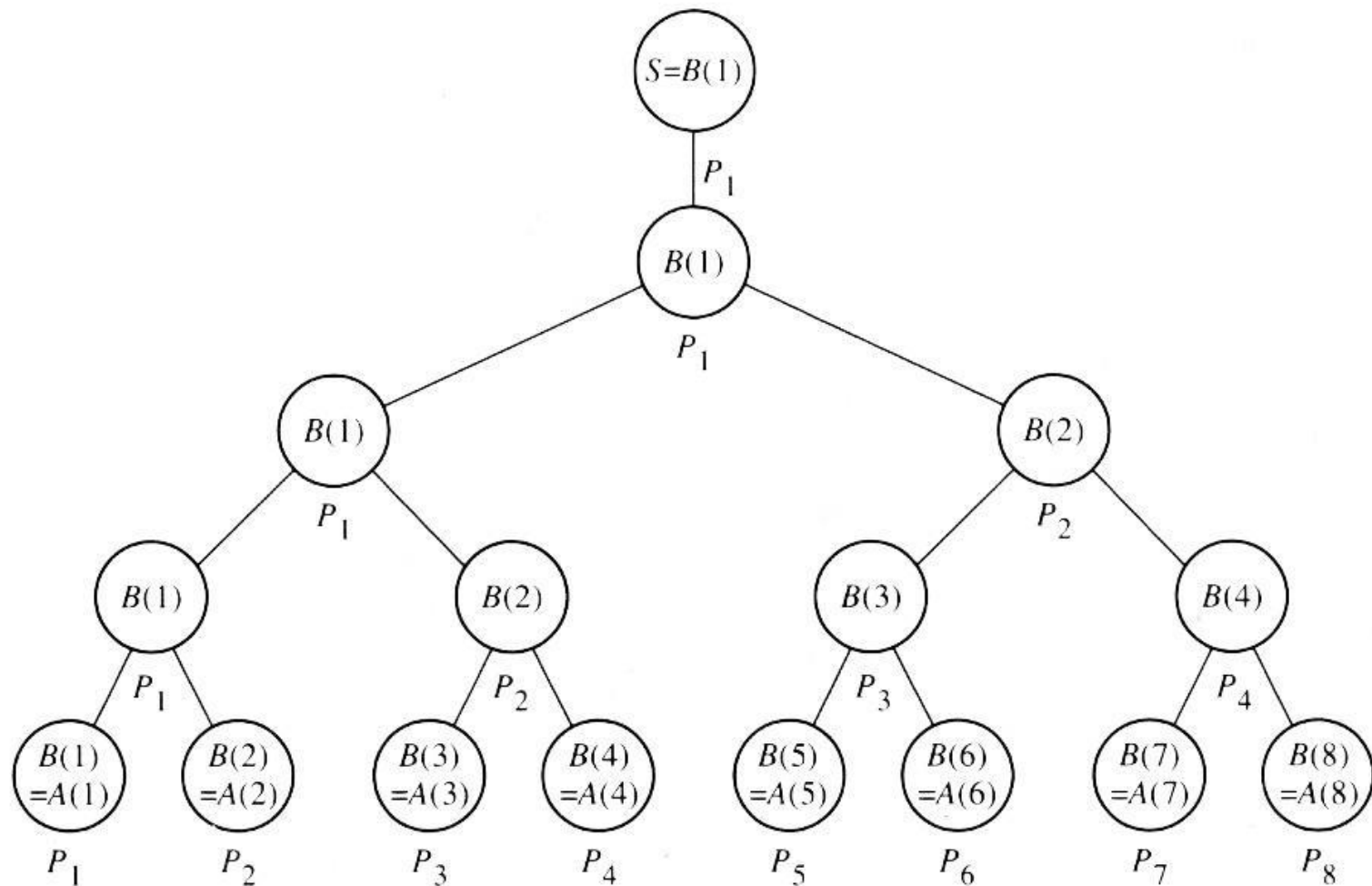


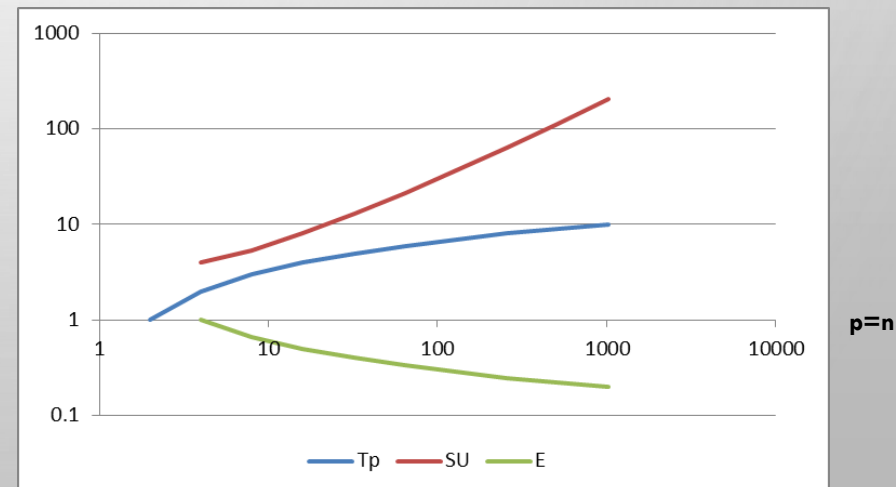
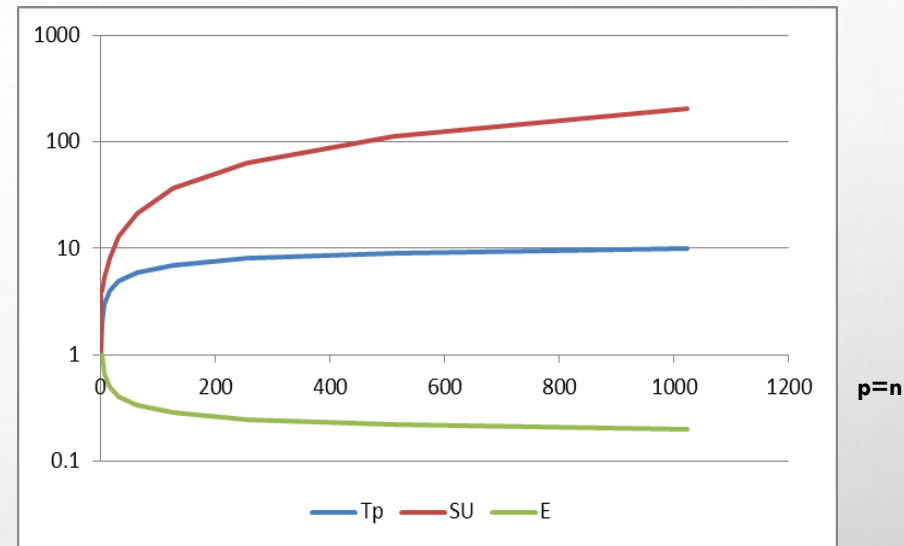
FIGURE 1.4

Computation of the sum of eight elements on a PRAM with eight processors. Each internal node represents a sum operation. The specific processor executing the operation is indicated below each node.

PERFORMANCE OF SUM (P=N)

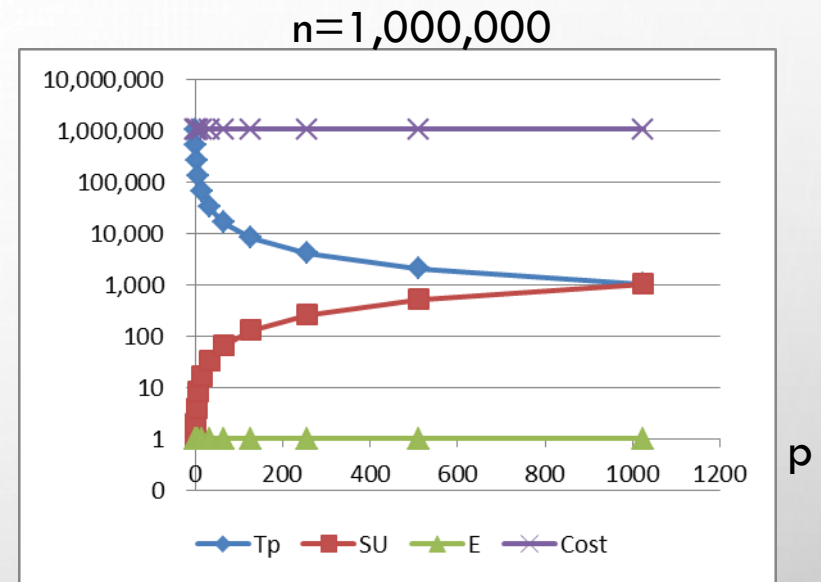
- $T^*(N) = T_1(N) = N$
- $T_{P=N}(N) = \text{LOG } N$
- $SU_P = \frac{n}{\log n}$
- $\text{COST} = P \cdot (2 + \text{LOG } N) \approx N \text{ LOG } N$
- $E_p = \frac{T_1}{pT_p} = \frac{n}{n \log n} = \frac{1}{\log n}$

Speedup and efficiency decrease



PERFORMANCE OF SUM ($N \gg P$)

- $T^*(N) = T_1(N) = N$
- $T_p(n) = \frac{n}{p} + \text{LOG } p$
- $SU_p = \frac{n}{\frac{n}{p} + \log p} \approx P$
- $\text{COST} = p \left(\frac{n}{p} + \text{LOG } p \right) \approx N$
- $\text{WORK} = N + P \approx N$
- $E_p = \frac{T_1}{pT_p} = \frac{n}{p \left(\frac{n}{p} + \text{LOG } p \right)} \approx 1$



Speedup & power are linear
 Cost is fixed
 Efficiency is 1 (max)

SIMPLIFYING PSEUDO-CODE

- REPLACE

GLOBAL READ($X \leftarrow B$)

GLOBAL READ($Y \leftarrow C$)

$Z := X + Y$

GLOBAL WRITE($Z \rightarrow A$)

- BY

$A := B + C$ ---A,B,C SHARED VARIABLES

EXAMPLE 3: MATRIX MULTIPLY ON PRAM



- $C := AB \quad (n \times n), \quad n = 2^k$
- RECALL MM: $C_{i,j} = \sum_{l=1}^n A_{i,l} B_{l,j}$
- $p = n^3$
- STEPS
 - PROCESSOR $P_{i,j,l}$ COMPUTES $A_{i,l} B_{l,j}$
 - THE n PROCESSORS $P_{i,j,1:n}$ COMPUTE SUM $\sum_{l=1}^n A_{i,l} B_{l,j}$

MM ALGORITHM

(EACH PROCESSOR KNOWS ITS I, J, L INDICES)

BEGIN

1. $T_{i,j,l} = A_{i,l}B_{l,j}$

2. FOR $H=1:K$

 IF $l \leq n/2^h$ THEN

$$T_{i,j,l} = T_{i,j,2l-1} + T_{i,j,2l}$$

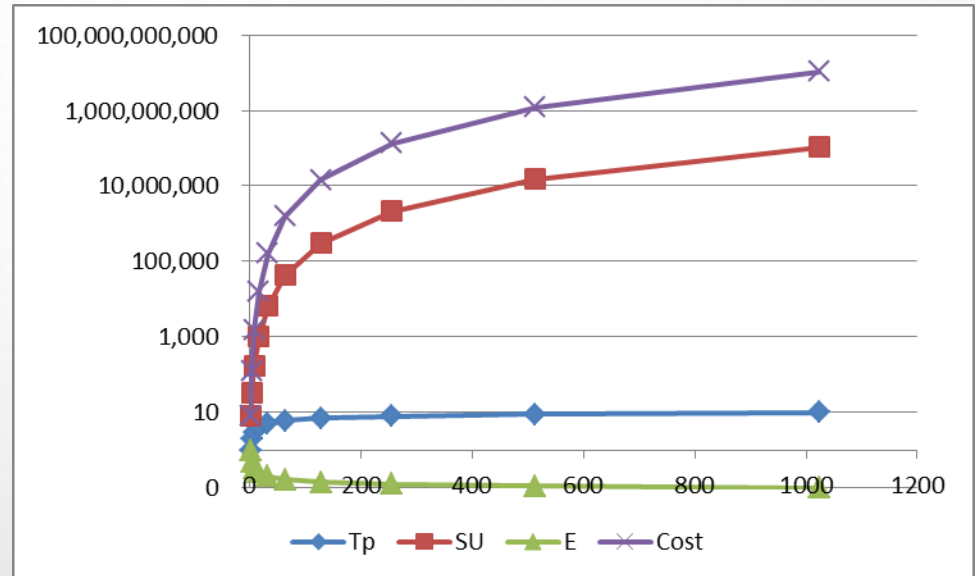
3. IF $l = 1$ THEN $C_{i,j} = T_{i,j,1}$

END

- STEP 1: COMPUTE $A_{i,l}B_{l,j}$
 - CONCURRENT READ
- STEP 2: SUM
- STEP 3: STORE
 - EXCLUSIVE WRITE
- RUNS ON CREW PRAM
- WHAT IS THE PURPOSE OF “IF $l = 1$ ” IN STEP 3?
WHAT HAPPENS IF ELIMINATED?

PERFORMANCE OF MM

- $T_1 = n^3$
- $T_{p=n^3} = \text{LOG } n$
- $SU = \frac{n^3}{\text{LOG } n}$
- $\text{Cost} = n^3 \text{ LOG } n$
- $E_p = \frac{T_1}{pT_p} = \frac{1}{\text{LOG } n}$



PREFIX SUM

- TAKE ADVANTAGE OF IDLE PROCESSORS IN SUM
- COMPUTE ALL PREFIX SUMS $S_i = \sum_1^i a_j$
 - $a_1, a_1 + a_2, a_1 + a_2 + a_3, \dots$

The diagram illustrates a 2D lattice structure with vertical bonds labeled s_1 through s_8 and diagonal bonds labeled a_1 through a_8 . Three green circles labeled "CR" are placed on vertical bonds s_2 , s_4 , and s_6 . Red arrows point from these circles to the next vertical bond (s_3 , s_5 , and s_7 respectively). Blue arrows point from diagonal bonds a_1 through a_8 to the next vertical bond (s_2 through s_8 respectively).

COMMON CW EXAMPLE 1: DNF

- BOOLEAN DNF (SUM OF PRODUCTS)
 - $X = A_1B_1 + A_2B_2 + A_3B_3 + \dots$ (AND, OR OPERATIONS)
 - PRAM CODE (X INITIALIZED TO 0, TASK INDEX=\$) :
$$\text{IF } (A_{\$}B_{\$}) \text{ } X=1;$$
 - COMMON OUTPUT:
 - NOT ALL PROCESSORS WRITE X.
 - THOSE THAT DO, WRITE 1.
 - TIME $O(1)$
 - GREAT FOR OTHER ASSOCIATIVE OPERATORS
 - E.G. $(A_1+B_1)(A_2+B_2) \dots$ OR/AND (CNF):
$$\text{INIT } X=1, \text{ IF NOT } (A_{\$}+B_{\$}) \text{ } X=0;$$
 - WORKS ON COMMON / PRIORITY / ARBITRARY CRCW/ERCW

PRAM SOP: CONCURRENT WRITE

- BOOLEAN $X = A_1B_1 + A_2B_2 + \dots$
- THE PRAM ALGORITHM

```
BEGIN
```

```
  IF ( $A_iB_i$ )  $X=1$ 
```

```
END
```

This is an example of ERCW.

ALL CORES WHICH WRITE INTO X, WRITE THE SAME VALUE

CRCW can be obtained by allowing the read of any variable
along the X computation

SOME VARIANTS OF PRAM

- **BOUNDED NUMBER OF SHARED MEMORY CELLS.** SMALL MEMORY PRAM (INPUT DATA SET EXCEEDS CAPACITY OF THE SHARE MEMORY I/O VALUES CAN BE DISTRIBUTED EVENLY AMONG THE PROCESSORS)
- **BOUNDED NUMBER OF PROCESSOR SMALL PRAM.** IF # OF THREADS OF EXECUTION IS HIGHER, PROCESSORS MAY INTERLEAVE SEVERAL THREADS.
- **BOUNDED SIZE OF A MACHINE WORD.** WORD SIZE OF PRAM
- **HANDLING ACCESS CONFLICTS.** CONSTRAINTS ON SIMULTANEOUS ACCESS TO SHARE MEMORY CELLS

LEMMA

- ASSUME $P' < P$. ANY PROBLEM THAT CAN BE SOLVED FOR A P PROCESSOR PRAM IN T STEPS CAN BE SOLVED IN A P' PROCESSOR PRAM IN $T' = O(TP/P')$ STEPS (ASSUMING SAME SIZE OF SHARED MEMORY)

PROOF:

- PARTITION P IS SIMULATED PROCESSORS INTO P' GROUPS OF SIZE P/P' EACH
- ASSOCIATE EACH OF THE P' SIMULATING PROCESSORS WITH ONE OF THESE GROUPS
- EACH OF THE SIMULATING PROCESSORS SIMULATES ONE STEP OF ITS GROUP OF PROCESSORS BY:
 - EXECUTING ALL THEIR READ AND LOCAL COMPUTATION SUBSTEPS FIRST
 - EXECUTING THEIR WRITE SUBSTEPS THEN

LEMMA

- **ASSUME $M' < M$.** ANY PROBLEM THAT CAN BE SOLVED FOR A P PROCESSOR AND M -CELL PRAM IN T STEPS CAN BE SOLVED ON A $\text{MAX}(P, M')$ -PROCESSORS M' -CELL PRAM IN $O(TM/M')$ STEPS

PROOF:

- PARTITION M SIMULATED SHARED MEMORY CELLS INTO M' CONTINUOUS SEGMENTS S_i OF SIZE M/M' EACH
- EACH SIMULATING PROCESSOR P'_i , $1 \leq i \leq P$, WILL SIMULATE PROCESSOR P_i OF THE ORIGINAL PRAM
- EACH SIMULATING PROCESSOR P'_i , $1 \leq i \leq M'$, STORES THE INITIAL CONTENTS OF S_i INTO ITS LOCAL MEMORY AND WILL USE $M'[i]$ AS AN AUXILIARY MEMORY CELL FOR SIMULATION OF ACCESSES TO CELL OF S_i
- SIMULATION OF ONE ORIGINAL READ OPERATION

EACH P'_i , $i=1, \dots, \text{MAX}(P, M')$ REPEATS FOR $K=1, \dots, M/M'$

1. WRITE THE VALUE OF THE K -TH CELL OF S_i INTO $M'[i]$, $i=1, \dots, M'$,
 2. READ THE VALUE WHICH THE SIMULATED PROCESSOR P_i , $i=1, \dots, P$, WOULD READ IN THIS SIMULATED SUBSTEP, IF IT APPEARED IN THE SHARED MEMORY
- THE LOCAL COMPUTATION SUBSTEP OF P_i , $i=1, \dots, P$ IS SIMULATED IN ONE STEP BY P'_i
 - SIMULATION OF ONE ORIGINAL WRITE OPERATION IS ANALOGOUS TO THAT OF READ

WHY PRAM?

- LARGE BODY OF ALGORITHMS
- EASY TO THINK ABOUT
- SYNC VERSION OF SHARED MEMORY → ELIMINATES SYNC AND COMM ISSUES, ALLOWS FOCUS ON ALGORITHMS
 - BUT ALLOWS ADDING THESE ISSUES
 - ALLOWS CONVERSION TO ASYNC VERSIONS
- EXIST ARCHITECTURES FOR BOTH
 - SYNC (PRAM) MODEL
 - ASYNC (SM) MODEL
- PRAM ALGORITHMS CAN BE MAPPED TO OTHER MODELS

The background of the slide is a light gray gradient. It is decorated with numerous realistic water droplets of various sizes. Some droplets are large and prominent, while others are small and subtle. They are scattered across the slide, with a higher concentration in the top-left and bottom-right corners. The droplets have highlights and shadows, giving them a three-dimensional appearance.

AMDAHL'S LAW VS GUSTAFSON'S LAW

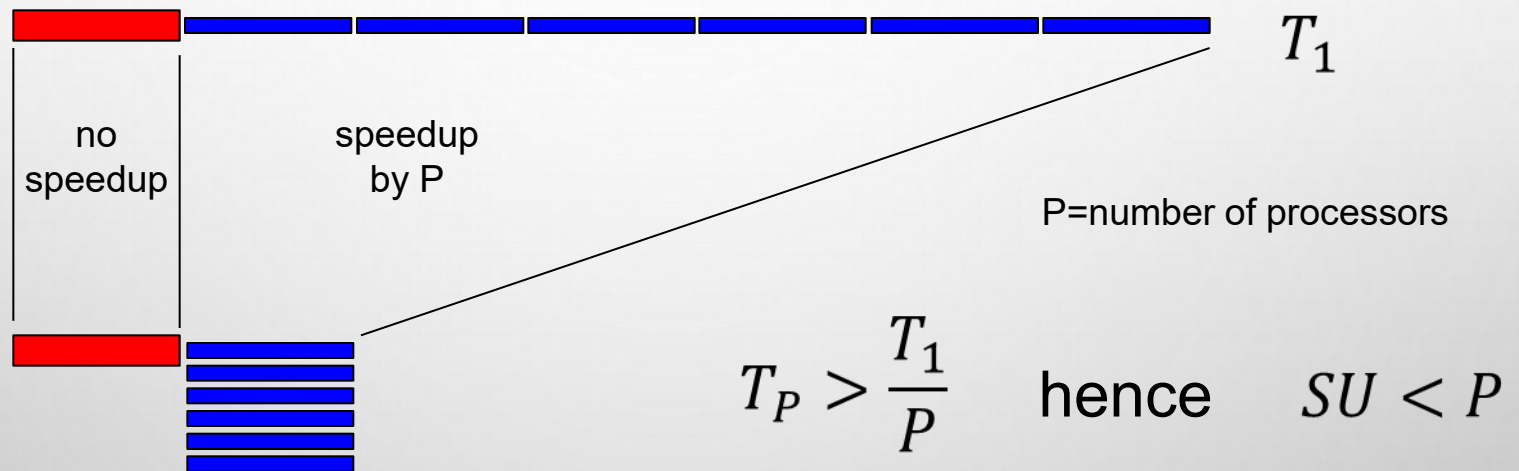
AMDAHL'S LAW

Strong scaling

- GENE AMDHAL WAS ONE OF THE THREE ARCHITECTS OF IBM MAINFRAMES
 - AMDAHL, BLAAUW & BROOKS (1964), ARCHITECTURE OF THE IBM SYSTEM/360, IBM J. R&D, 8(2):87-101.
- HE OBJECTED TO PARALLELISM
 - AMDAHL (1967), VALIDITY OF THE SINGLE PROCESSOR APPROACH TO ACHIEVING LARGE-SCALE COMPUTING CAPABILITIES, AFIPS CONFERENCE PROCEEDINGS (30): 483–485. [HTTP://WWW-INST.EECS.BERKELEY.EDU/~N252/PAPER/AMDAHL.PDF](http://www-inst.eecs.berkeley.edu/~N252/PAPER/AMDAHL.PDF)
- HIS MODEL HAS BEEN ABUSED EVER SINCE...

AMDAHL'S LAW

- MODEL: COMPUTATION CONSISTS OF INTERLEAVED SEGMENTS OF TWO TYPES:
 - SERIAL SEGMENTS – CANNOT BE PARALLELIZED
 - PARALLELIZABLE SEGMENTS

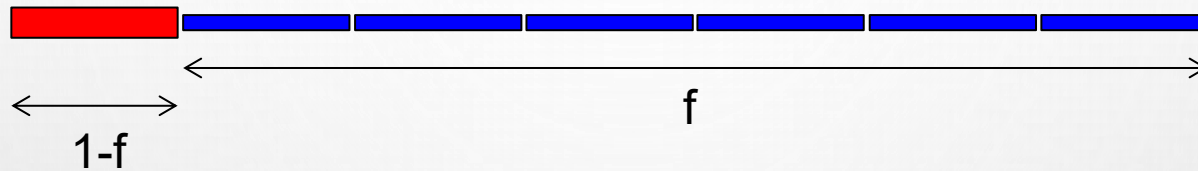


- CONTINUOUS SCENARIO IN THE MODEL:



AMDAHL'S LAW

- MODEL: IN A SERIAL VERSION, THE PARALLELIZABLE PART IS A FIXED FRACTION f

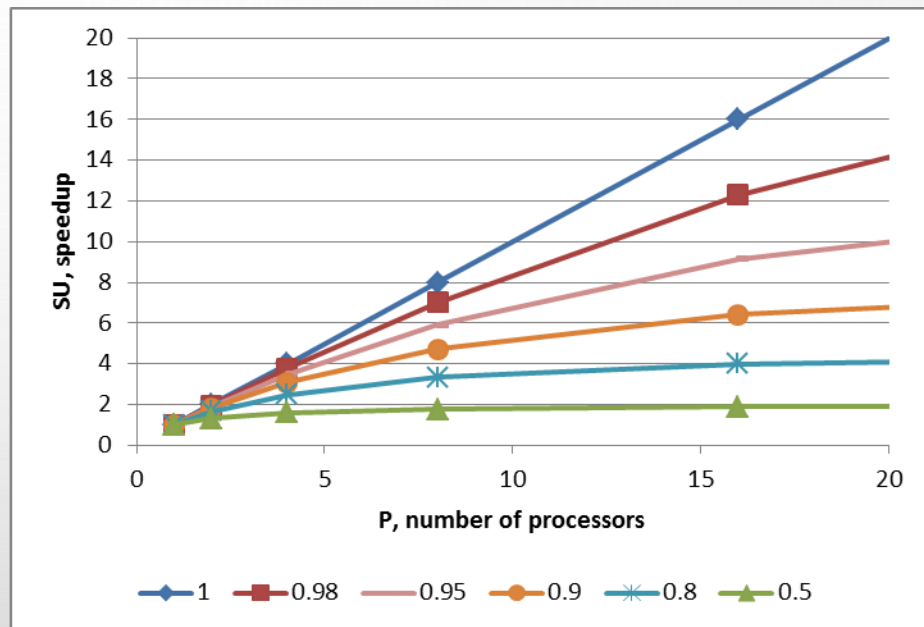
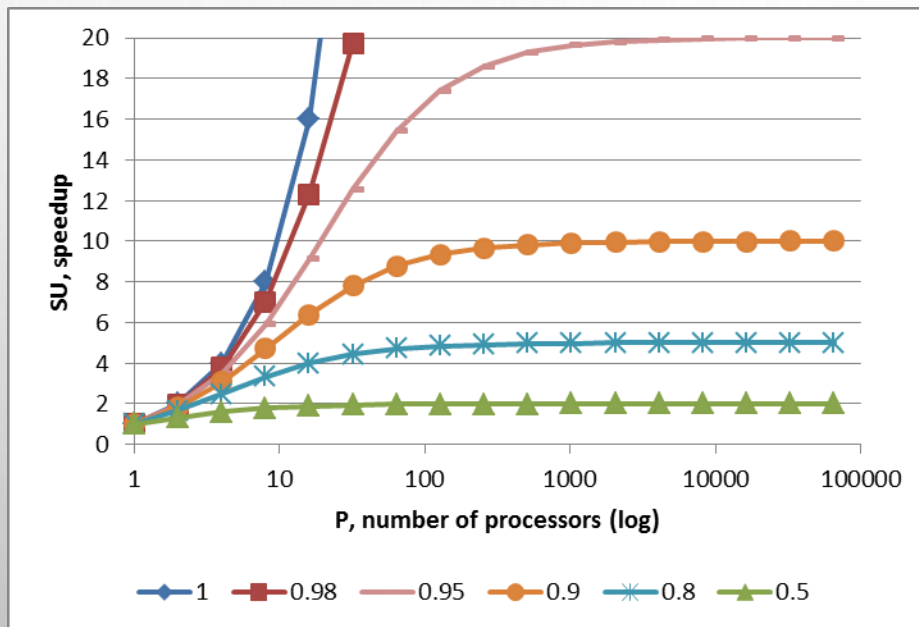


- EXPRESSION:

$$SU(P, f) = \frac{T_1}{T_P} = \frac{T_1}{T_1 \cdot (1 - f) + \frac{T_1 \cdot f}{P}} = \frac{1}{(1 - f) + \frac{f}{P}}$$
$$\lim_{P \rightarrow \infty} SU(P, f) = \frac{1}{1 - f}$$

AMDAHL'S LAW

SU(P), parameter f



Note the pessimism: given a problem with inherent $f=90\%$, there is no points in using more than 10 processors.

GUSTAFSON DIDN'T AGREE

- JOHN L. GUSTAFSON (1988), REEVALUATING AMDAHL'S LAW
[HTTP://HINT.BYU.EDU/DOCUMENTATION/GUS/AMDAHLSLAW/AMDAHLS.HTML](http://hint.byu.edu/documentation/gus/amdahlslaw/amdahls.html) (SANDIA NATIONAL LABS)
- ON 1024 HYPERCUBE MPP

Problem	Total Speedup	Speedup of parallel parts
beam stress analysis using conjugate gradients	1021	1023.9969
wave simulation using explicit finite differences	1020	1023.9965
unstable fluid flow using flux-corrected transport	1016	1023.9965

- SUGGESTED: *“WE FEEL THAT IT IS IMPORTANT FOR THE COMPUTING RESEARCH COMMUNITY TO OVERCOME THE "MENTAL BLOCK" AGAINST MASSIVE PARALLELISM IMPOSED BY A MISUSE OF AMDAHL'S SPEEDUP FORMULA.”*

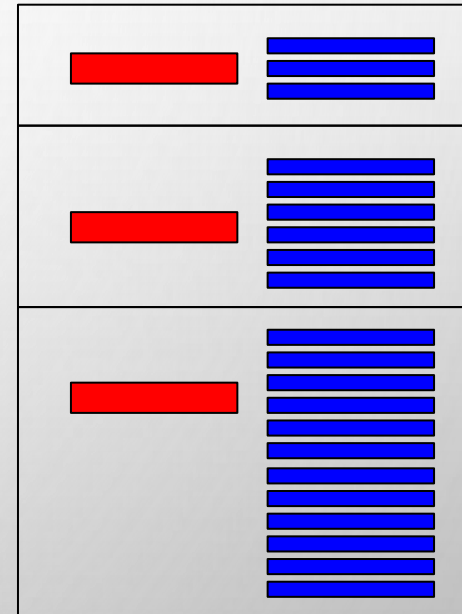
GUSTAFSON'S LAW

Weak scaling

- KEY POINTS

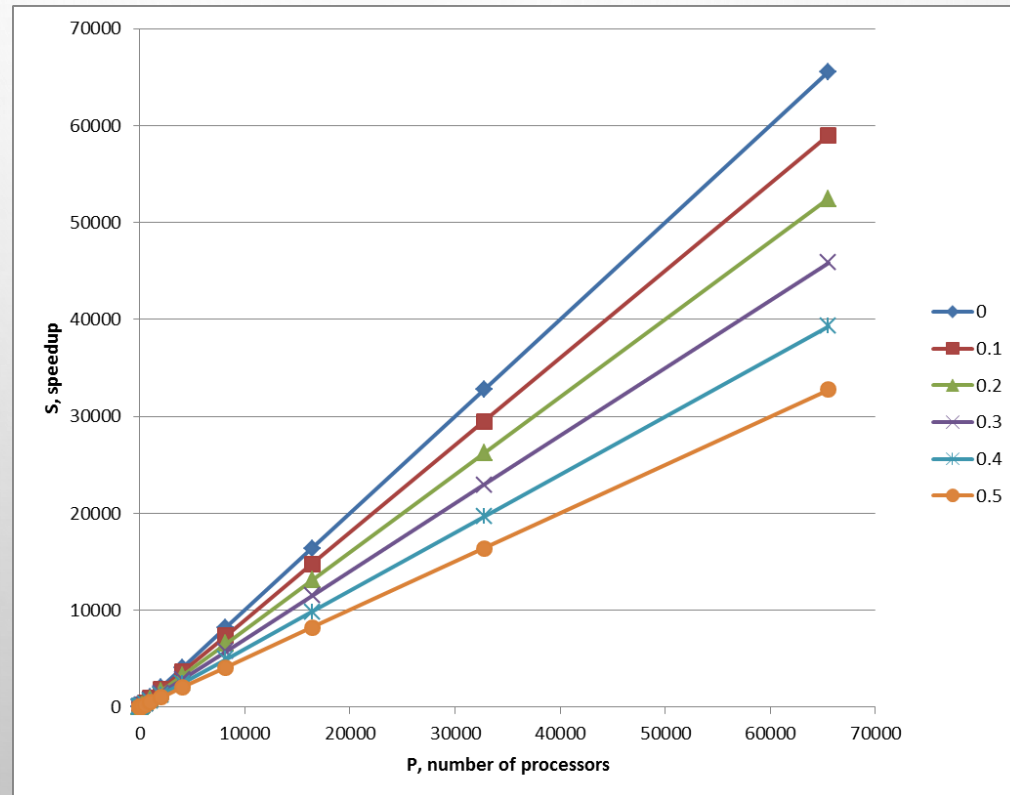
- PORTION F IS NOT FIXED
- ABSOLUTE SERIAL TIME IS FIXED
- PARALLEL PROBLEM SIZE IS INCREASED TO EXPLOIT MORE PROCESSORS
- INVARIANTS:
 - FIXED SERIAL TIME (s OF TOTAL)
 - FIXED PARALLEL TIME ($1-s$ OF TOTAL)
- 'FIXED TIME MODEL'
 - AMDAHL'S IS 'FIXED SIZE MODEL'

$$SU(P) = \frac{T_1}{T_P} = \frac{s + P \cdot (1 - s)}{s + (1 - s)} = s + P \cdot (1 - s)$$



GUSTAFSON'S LAW

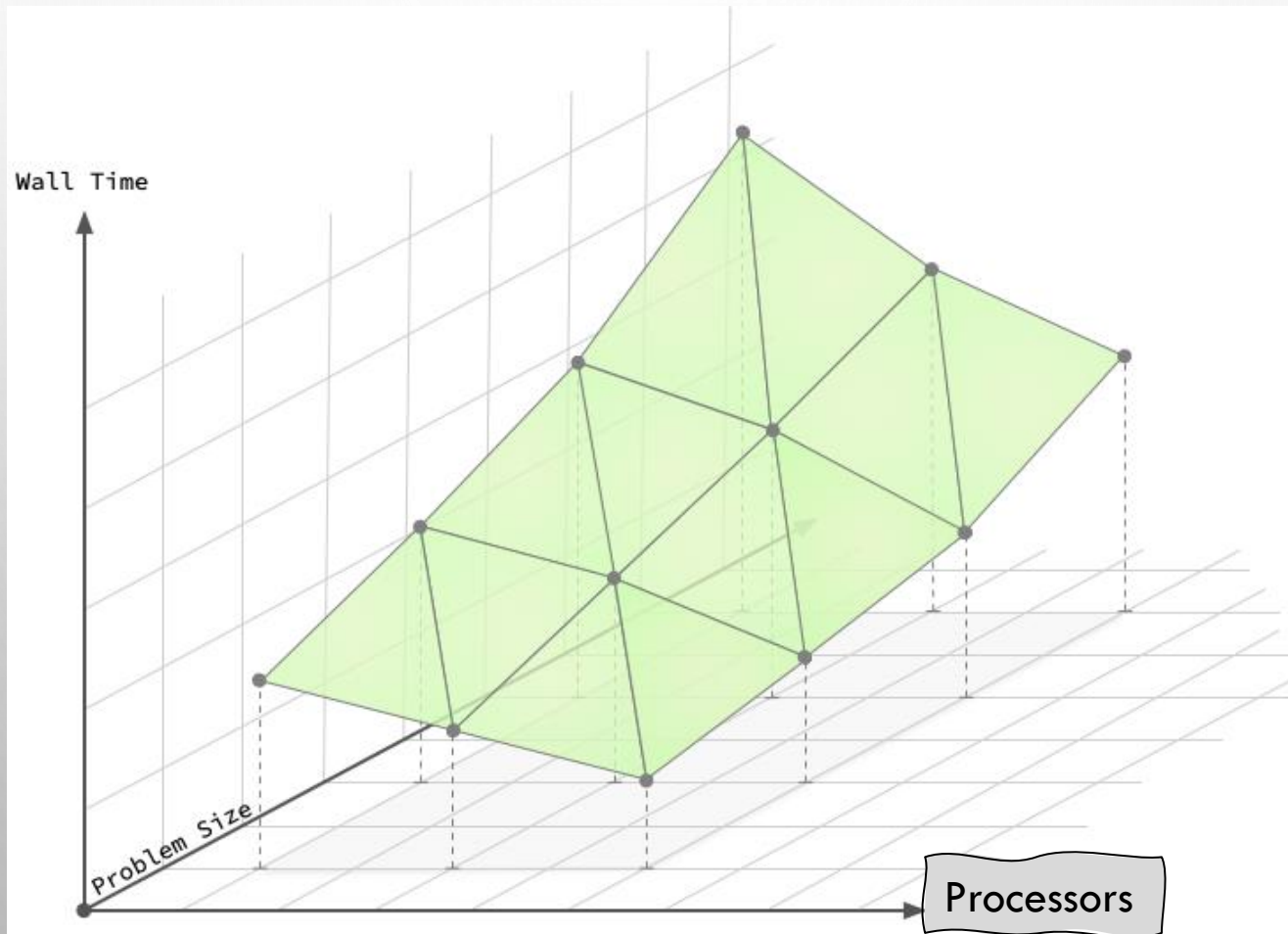
- LINEAR SPEEDUP!
- EMPIRICALLY APPLICABLE TO HIGHLY-PARALLEL ALGORITHMS





SCALING

The relationship between problem size, number of workers, and solution can be visualized as a three-dimensional surface.

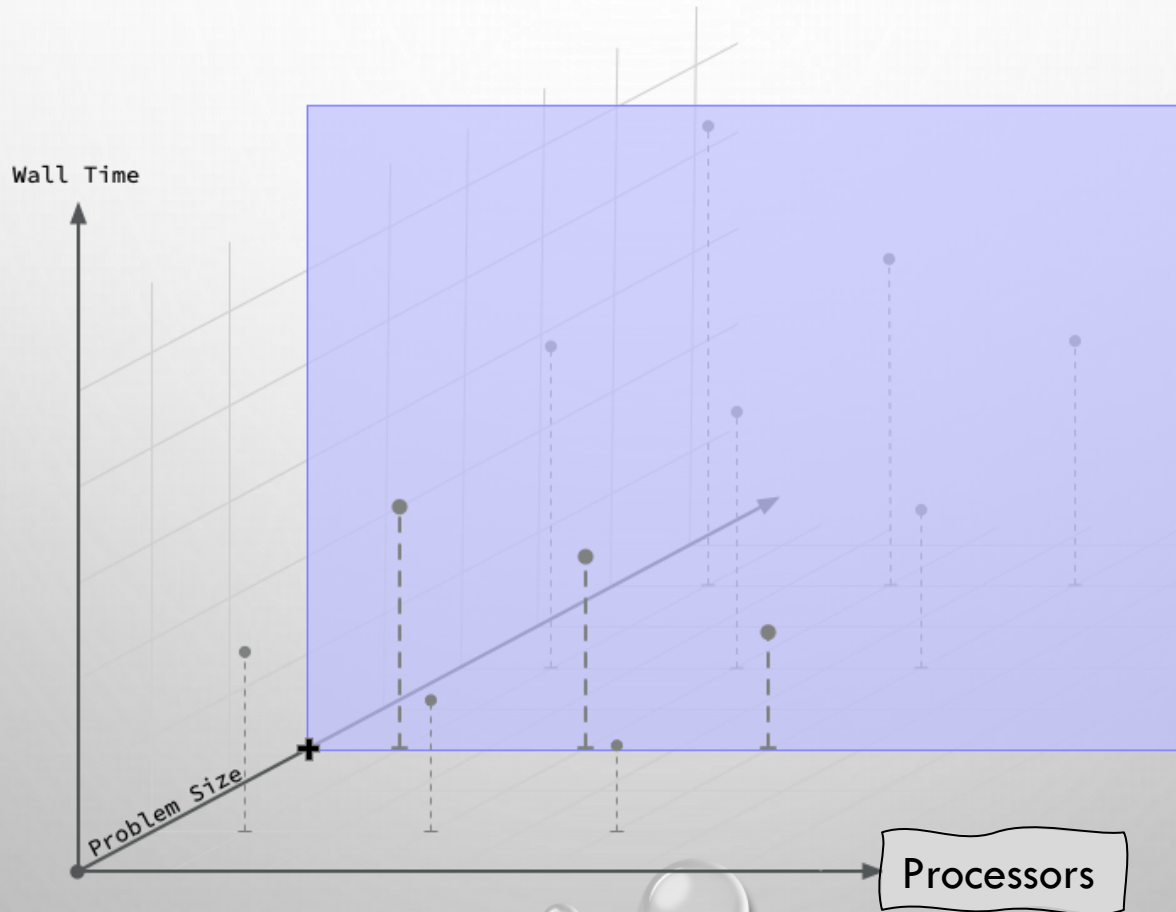


STRONG SCALING

The term "strong scaling" refers to the ability of a system to improve the performance of a parallelized program when the number of processors is increased while keeping the problem to be solved constant. In other words, it measures how quickly a program can be executed with more processors without changing the size of the problem.

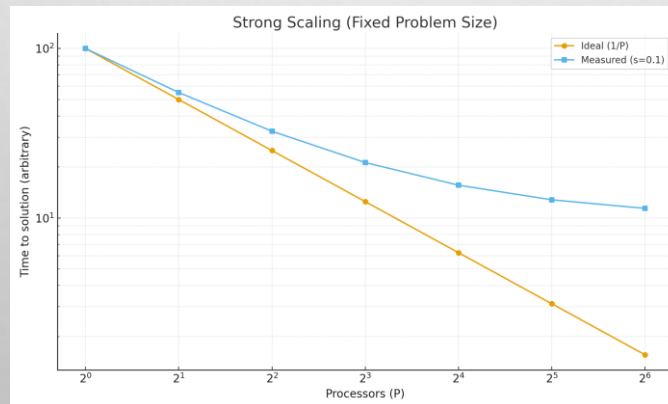
The term "strong" refers to the system's robustness in handling a fixed problem with an increasing number of processors. Amdahl's law is often used to analyze this type of scaling, as it highlights the limits imposed by the serial portion of the program.

STRONG SCALING IS CONCERNED WITH A SLICE OF THE SCALING SURFACE AT A PARTICULAR PROBLEM SIZE.

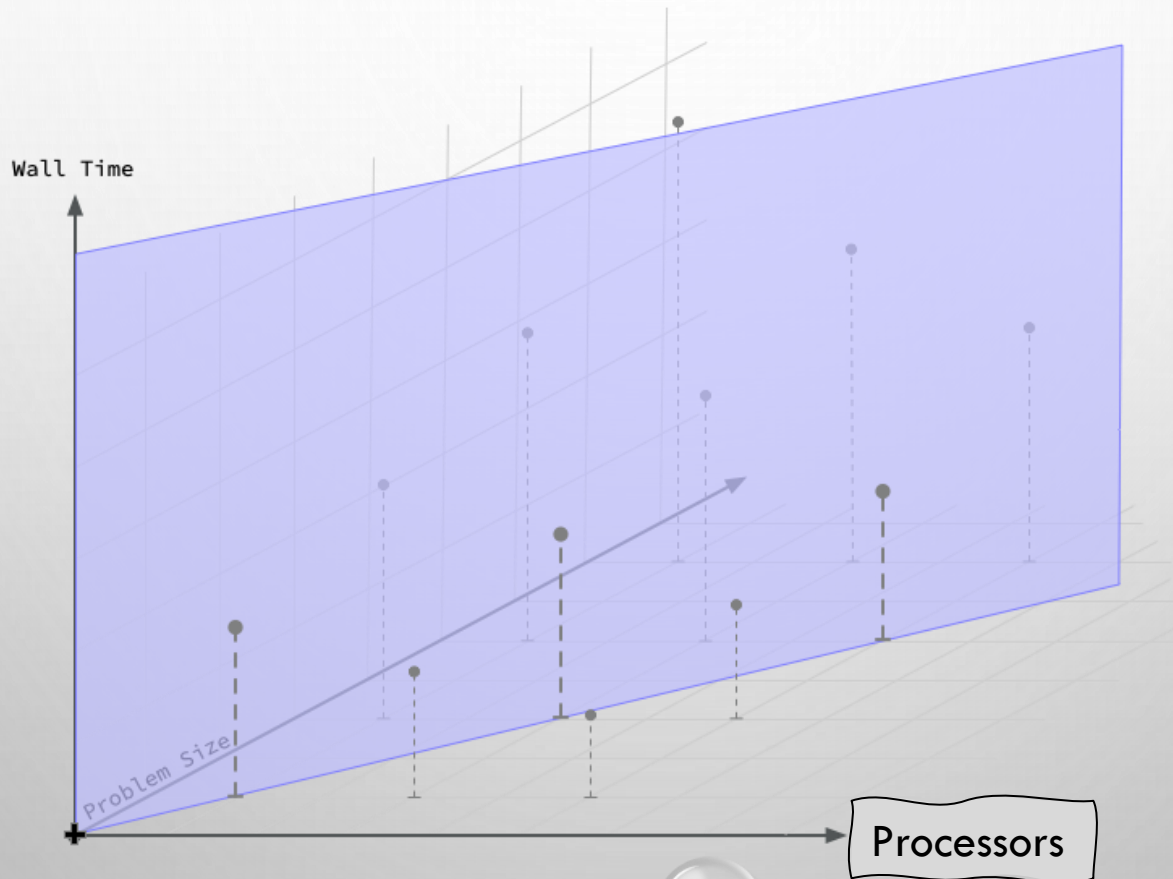


STRONG SCALING (FIXED PROBLEM SIZE)

- GOAL: REDUCE TIME-TO-SOLUTION BY ADDING PROCESSORS WHILE KEEPING N FIXED.
- IDEAL: $T \propto 1/P$ (SLOPE -1 ON A LOG-LOG PLOT).
- BOUNDED BY AMDAHL'S LAW: $SU(P) = 1 / (S + (1-S)/P)$, WHERE S IS THE SERIAL FRACTION.



Weak scaling is concerned with a slice of the scaling surface where the work per task is constant.

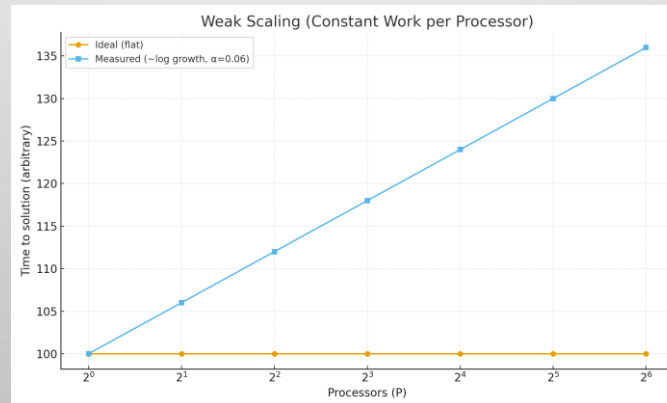


WEAK SCALING

- Weak scaling refers to a system's ability to maintain efficiency when the number of processors is increased while the workload is proportionally increased. In other words, it measures how well a system can handle a larger problem with more processors while maintaining a constant workload per processor.
- Weak scaling means that the system is evaluated based on its ability to scale with an increasing workload rather than with a fixed problem.
- This type of scaling is useful for applications where the problem grows in proportion to the increase in available resources.
- Gustafson's law is often used to analyze this type of scaling.

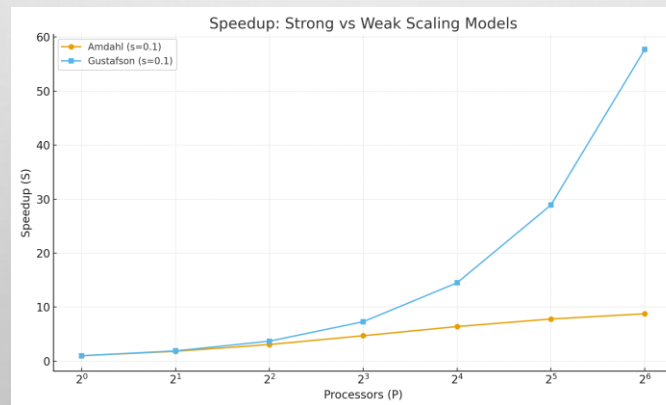
WEAK SCALING (CONSTANT WORK PER PROCESSOR)

- GOAL: SOLVE A PROPORTIONALLY LARGER PROBLEM IN THE SAME TIME AS P INCREASES.
- IDEAL: T IS FLAT VS P; DEVIATIONS REVEAL COMMUNICATION/OVERHEAD GROWTH.
- MODEL OFTEN EXPLAINED BY GUSTAFSON'S LAW: $SU_p = S + (1 - S) \cdot P$.



AMDAHL VS. GUSTAFSON (MODEL COMPARISON)

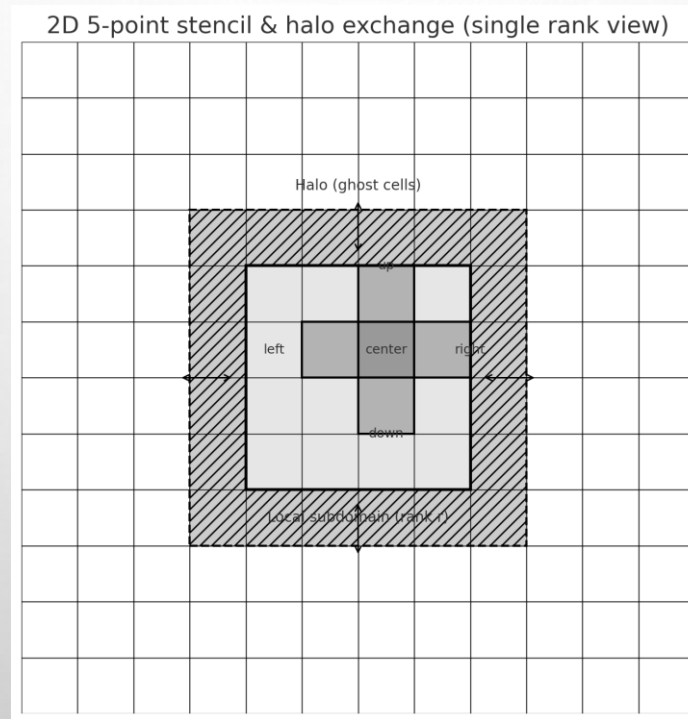
- AMDAHL (FIXED-SIZE/STRONG SCALING) PREDICTS DIMINISHING RETURNS AS P GROWS WHEN $S > 0$.
- GUSTAFSON (FIXED-TIME/WEAK SCALING) SHOWS LINEAR GROWTH WHEN WE SCALE THE PROBLEM WITH P .



HOW TO MEASURE SCALING

- STRONG: KEEP TOTAL INPUT N AND ALGORITHM FIXED; VARY P ; RECORD $T_p(N)$, SU_p , E_p .
- WEAK: KEEP PER-RANK WORKLOAD n FIXED; SET $N=P \cdot n$; VARY P ; RECORD $T_p(n \cdot P)$ AND EW_p .
- USE MULTIPLE RUNS; REPORT MEDIAN; PIN THREADS; DISABLE DVFS/TURBO DRIFT; FIX PROBLEM SEEDS.
- PLOT ON LOG–LOG (STRONG) AND SEMILOG (WEAK) AS APPROPRIATE; ANNOTATE SLOPES/EFFICIENCIES.

2D STENCIL PROBLEM



EXAMPLE (STRONG): 2D STENCIL ON A FIXED GRID

- KERNEL: 5-POINT STENCIL ON A 4096×4096 GRID FOR 100 STEPS; DOMAIN DECOMPOSITION.
- EXPECTATION: COMMUNICATION/HALO EXCHANGE AND MEMORY BANDWIDTH CAP SPEEDUP BEYOND SOME P.
- REPORT: T_p , SU_p , E_p ; CHECK IMBALANCE AND NUMA EFFECTS IF EFFICIENCY DROPS EARLY.

EXAMPLE (WEAK): 2D STENCIL WITH CONSTANT LOCAL GRID

- KERNEL: 5-POINT STENCIL; FIX PER-RANK GRID TO 1024×1024 ; INCREASE P SO TOTAL GRID GROWS.
- EXPECTATION: T_p STAYS \sim CONSTANT UNTIL SURFACE/VOLUME RATIO MAKES COMMUNICATION DOMINATE.
- METRIC: $E_p = T^*(n) / PT_p(n)$; TRACK NETWORK LATENCY/BANDWIDTH EFFECTS.

EXAMPLE

PATHWAYS LANGUAGE MODEL (PALM): SCALING TO 540 BILLION PARAMETERS FOR BREAKTHROUGH PERFORMANCE

- PALM DEMONSTRATES THE FIRST LARGE-SCALE USE OF THE PATHWAYS SYSTEM TO SCALE TRAINING TO 6144 CHIPS, THE LARGEST TPU-BASED SYSTEM CONFIGURATION USED FOR TRAINING TO DATE. THE TRAINING IS SCALED USING DATA PARALLELISM AT THE POD LEVEL ACROSS TWO CLOUD TPU V4 PODS, WHILE USING STANDARD DATA AND MODEL PARALLELISM WITHIN EACH POD.
- PALM ACHIEVES A TRAINING EFFICIENCY OF 57.8% HARDWARE FLOPS UTILIZATION

QUESTION ANSWERING

ARITHMETIC



LANGUAGE UNDERSTANDING

8 billion parameters

- AMDAHL'S LAW PRESUPPOSES THAT THE COMPUTING REQUIREMENTS WILL STAY THE SAME, GIVEN INCREASED PROCESSING POWER. IN OTHER WORDS, AN ANALYSIS OF THE SAME DATA WILL TAKE LESS TIME GIVEN MORE COMPUTING POWER.

CONCLUSIONS

- GUSTAFSON, ON THE OTHER HAND, ARGUES THAT MORE COMPUTING POWER WILL CAUSE THE DATA TO BE MORE CAREFULLY AND FULLY ANALYZED
- WHERE IT WOULD NOT HAVE BEEN POSSIBLE OR PRACTICAL TO SIMULATE THE IMPACT OF NUCLEAR DETONATION ON EVERY BUILDING, CAR, AND THEIR CONTENTS (INCLUDING FURNITURE, STRUCTURE STRENGTH, ETC.) BECAUSE SUCH A CALCULATION WOULD HAVE TAKEN MORE TIME THAN WAS AVAILABLE TO PROVIDE AN ANSWER, THE INCREASE IN COMPUTING POWER WILL PROMPT RESEARCHERS TO ADD MORE DATA TO MORE FULLY SIMULATE MORE VARIABLES, GIVING A MORE ACCURATE RESULT.

CONCLUSIONS