CASE SCHOOL
OF ENGINEERING
CASE WESTERN RESERVE
UNIVERSITY

CSDS 451: Designing High Performant Systems for AI

Lecture 6

9/11/2025

Sanmukh Kuppannagari

sanmukh.kuppannagari@case.edu

https://sanmukh.research.st/

Case Western Reserve University

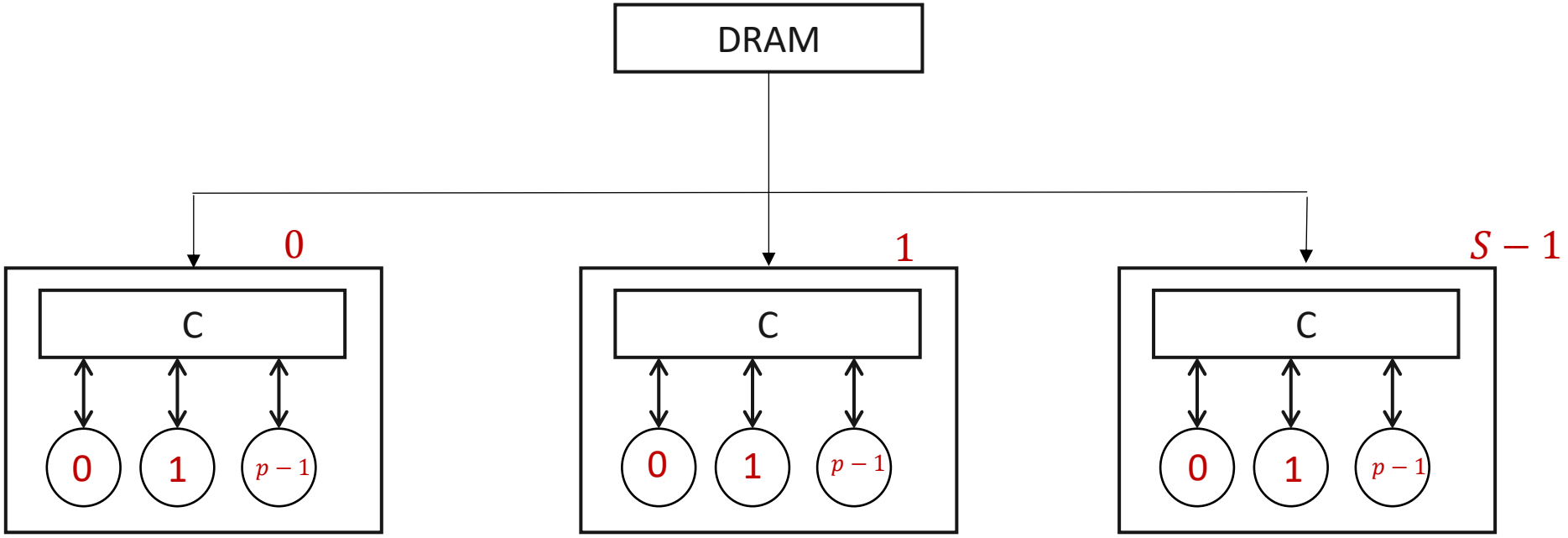
Outline

- Data Parallel Programming
  - Data Parallel Programming Approaches
- Parallel Program Analysis

Outline

- Data Parallel Programming
  - Data Parallel Programming Approaches
- Parallel Program Analysis

Modeling GPU Architectures



- Number of Blocks can be (in fact should be) greater than the number of SMPs
- Number of threads per block need not be equal to the number of compute cores per SMP
  - It is good to have it as a multiple though
- We will discuss these considerations when we discuss GPU programming later

- For now, assume the simple model above.
  - S Blocks
  - p Threads per block

GPU Model

- Think in terms of what each block computes, and what each thread within a block computes
- Each thread on the GPU executes the *same instructions* on *different data*
- We need to rethink our algorithms in terms of the above
- Think Data Parallel

Data Parallelism

- Key Idea:
  - Partition data into chunks –> Each chunk should require similar amount of work
  - Assign tasks to each partition so that they can proceed concurrently
- Note: data parallelism doesn't mean each task "need" to perform exact same computations
  - Usually it is true
- SPMD (Single Program Multiple Data) is a special case of data parallelism where same "program" is run on different data

Data Parallelism - Approaches

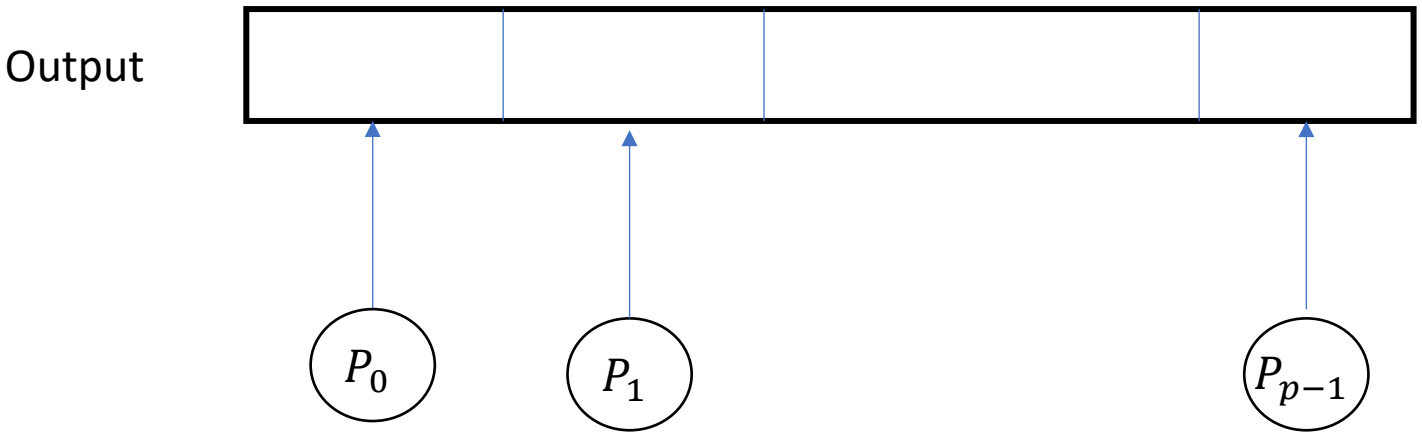
- #1 Partition Output data
  - Each task is responsible for computing an output partition

Data Parallelism - Approaches

- #1 Partition Output data
  - Each task is responsible for computing an output partition

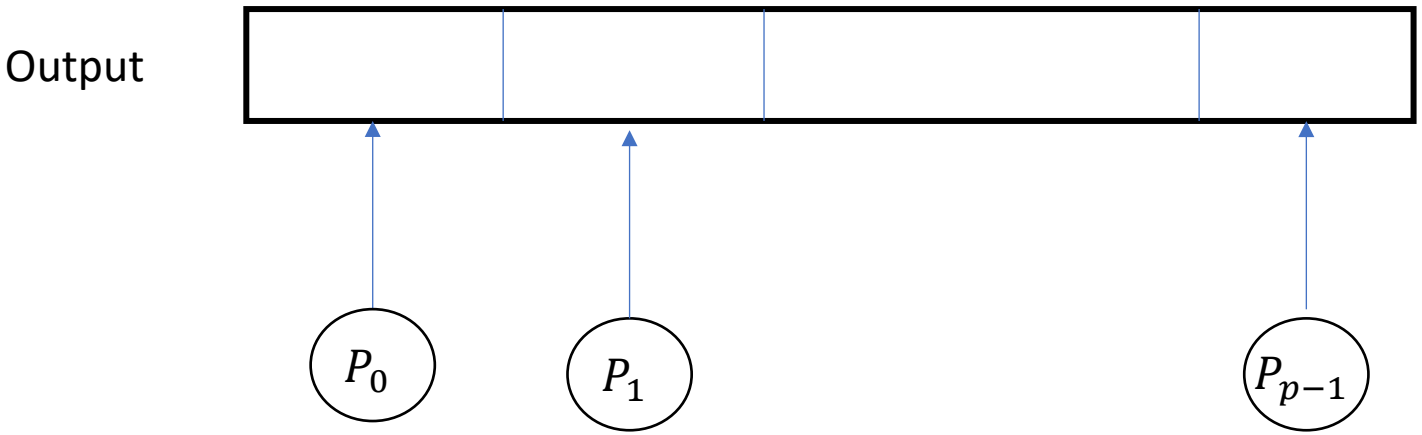
Data Parallelism - Approaches

- #1 Partition Output data
  - Each task is responsible for computing an output partition



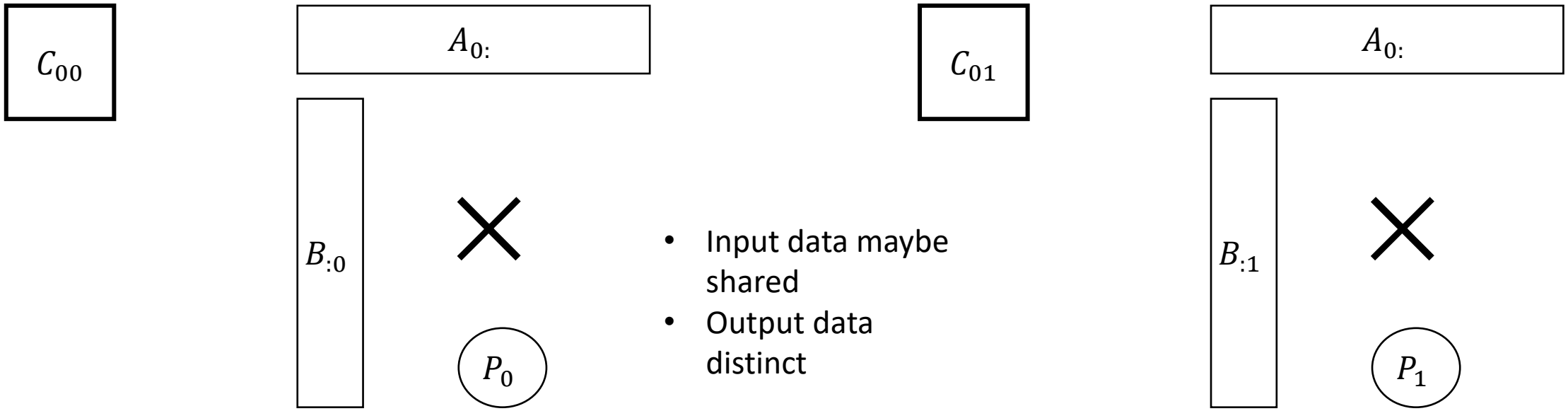
Data Parallelism - Approaches

- #1 Partition Output data
  - Useful when partitions of output can be computed independently of each other
  - Example: Blocked Matrix Multiplication



Data Parallelism – Approaches

- #1 Partition Output data
  - Matrix Multiplication

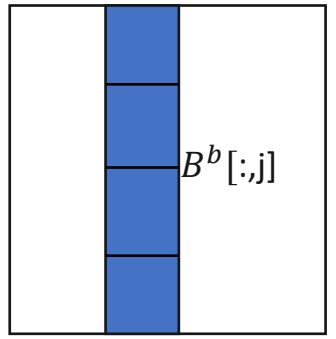


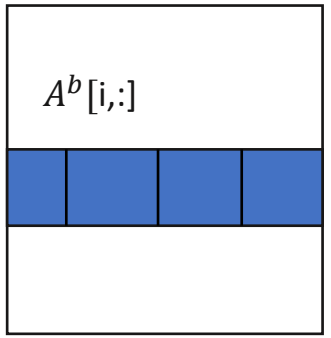
- Input data maybe shared
- Output data distinct

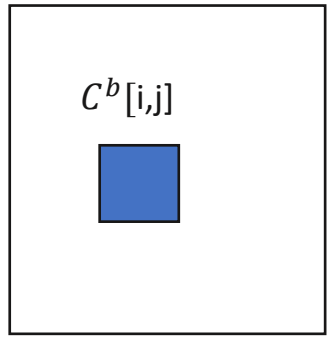
Block Matrix Multiplication

- Key Idea: Partition matrices A/B/C into $b \times b$ size blocks
- Blocks denoted as $A^b[1:\frac{N}{b}][1:\frac{N}{b}]$/ $B^b[1:\frac{N}{b}][1:\frac{N}{b}]$/ $C^b[1:\frac{N}{b}][1:\frac{N}{b}]$

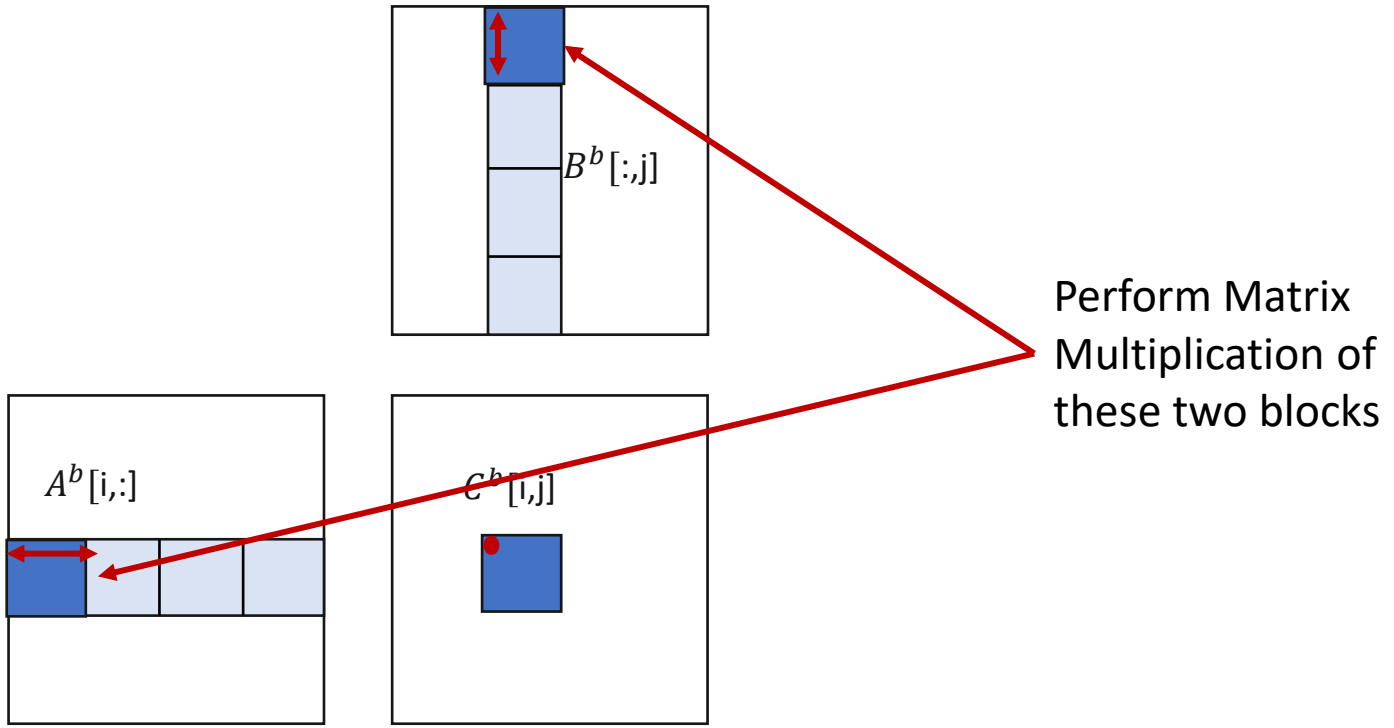
- C^b[i][j] = BlockedMM(A^b[i][:], B^b[:][j])
- For $k = 0$ to N/b
  - Fetch A^b[i][k], B^b[k][j]
  - Matrix Multiply A^b[i][k], B^b[k][j] and Accumulate C^b[i][j]



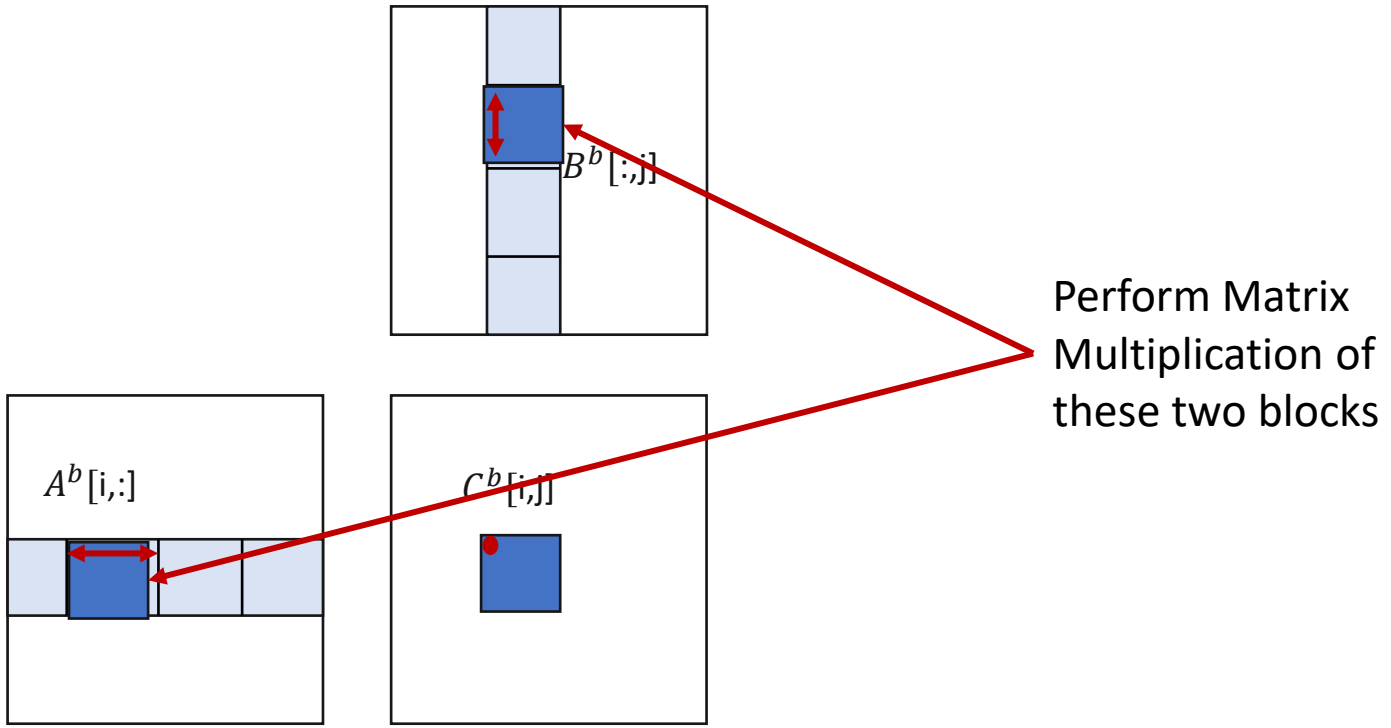




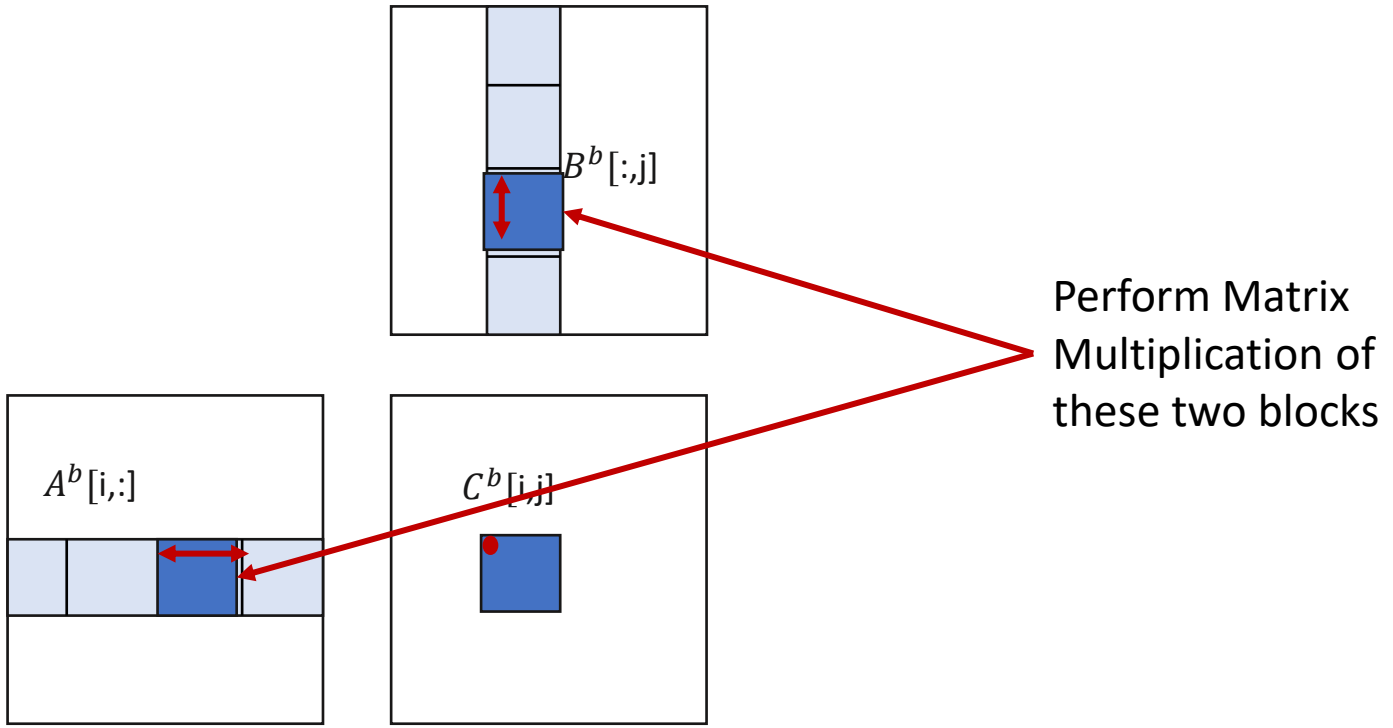
Block Matrix Multiplication



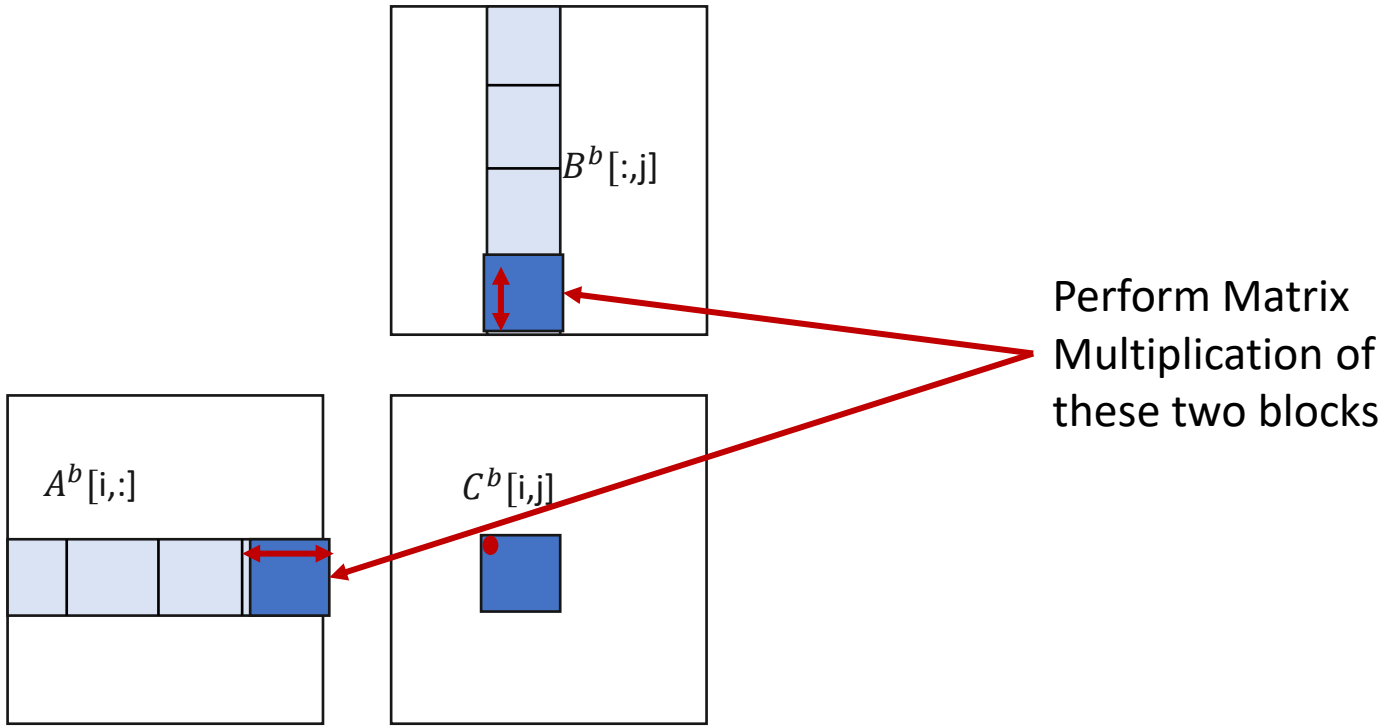
Block Matrix Multiplication



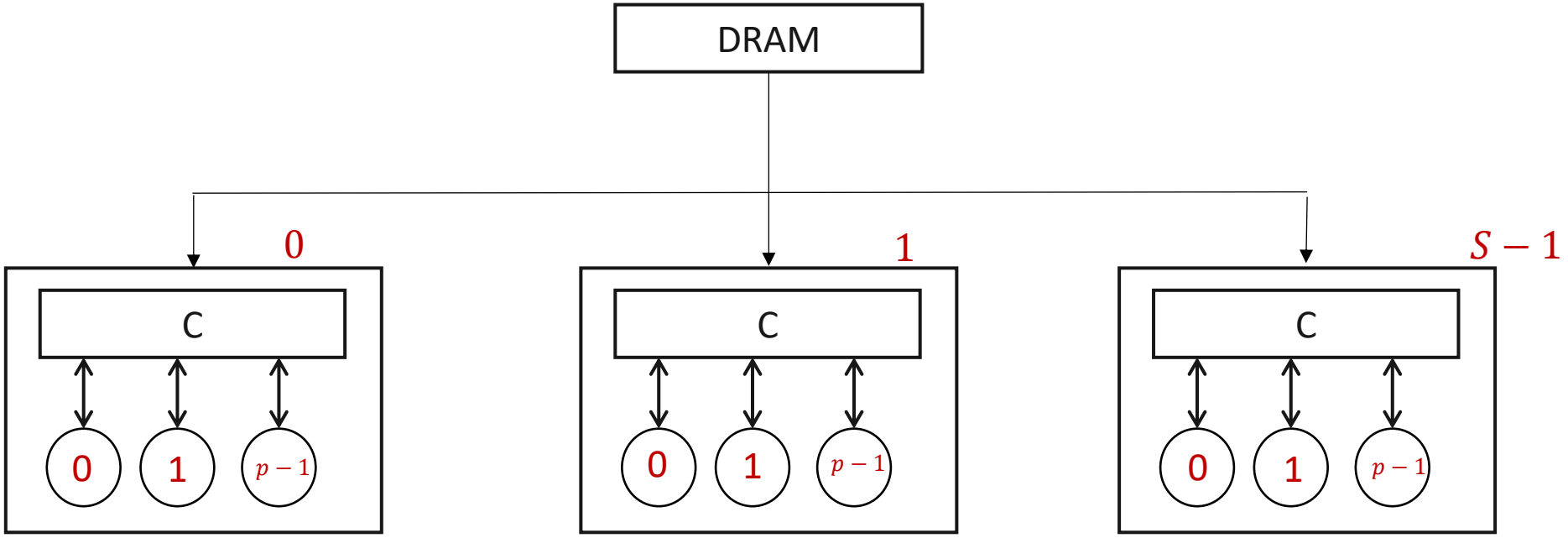
Block Matrix Multiplication



Block Matrix Multiplication



Modeling GPU Architectures



- S Blocks (I will call them SMPs in the next few slides to avoid confusion with Blocks of matrices)
- p Threads per block

Block Matrix Multiplication

- What is a natural way to split the computations of Matrix Multiplication to the SMPs of GPUs?

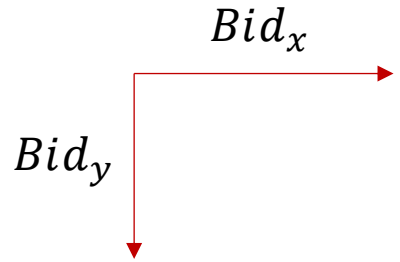
Block Matrix Multiplication

- What is a natural way to split the computations of Matrix Multiplication to the SMPs of GPUs?

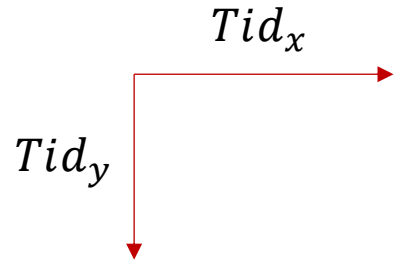
- Assign each output block $C^b[i][j]$ to an SMP

- Use the p threads to compute N/b block matrix multiplication

Block Matrix Multiplication



- Assume $S = \frac{N}{b} \times \frac{N}{b}$ SMPs
  - With block IDs: Bid_y, Bid_x



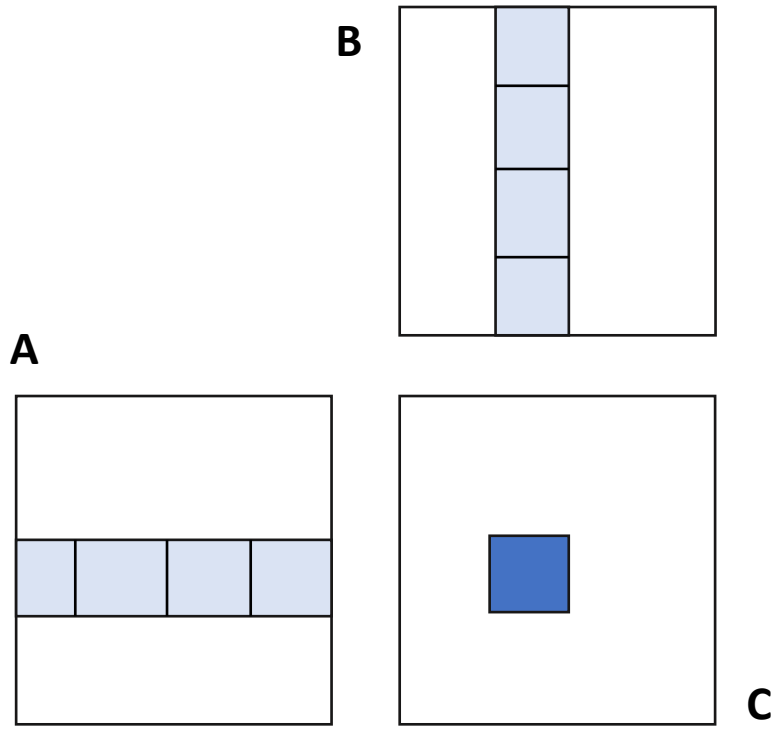
- Assume $p = b \times b$ threads
  - With thread IDs: Tid_y, Tid_x

- Need to convert the following into GPU model:

- For $k = 0$ to N/b
  - Fetch $A^b[i][k]$, $B^b[k][j]$
  - Matrix Multiply $A^b[i][k]$, $B^b[k][j]$ and Accumulate $C^b[i][j]$

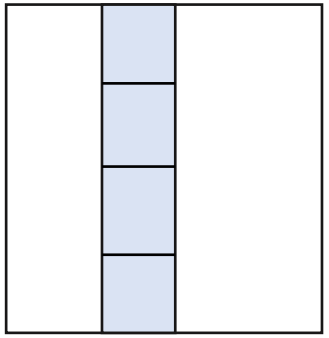
Block Matrix Multiplication

- In $Bid_y, Bid_x, Tid_y, Tid_x$ {
- For $k = 0$ to $\frac{N}{b} - 1$
  - Read A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$Tid_x$]
  - Read B[$k * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$]
  - Syncthreads()
  - For l = 0 to $b - 1$:
    - C[$Bid_y * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$] +=
    - A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$l$]* B[$k * \frac{N}{b} + l$][$Bid_x * \frac{N}{b} + Tid_x$]
  - Syncthreads()
- Store C[$Bid_y * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$] }



SMP Bid_y, Bid_x produces the output block Bid_y, Bid_x

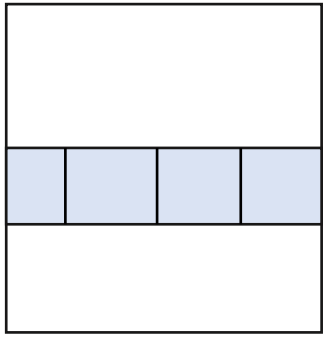
Block Matrix Multiplication

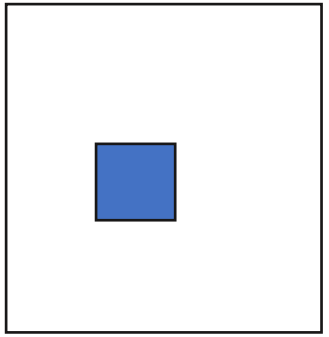


B

What are the elements of C that will be output by SMP Bid_y, Bid_x?

A

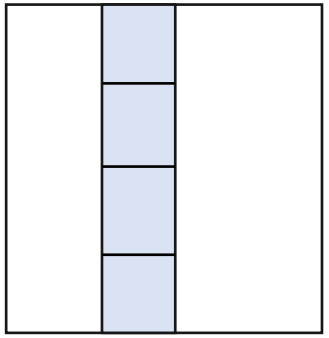




C

SMP Bid_y, Bid_x produces the output block Bid_y, Bid_x

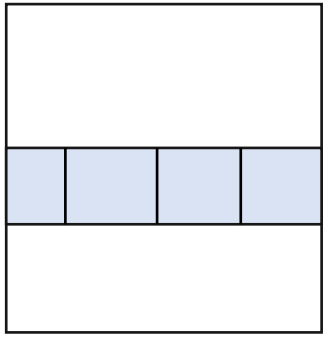
Block Matrix Multiplication

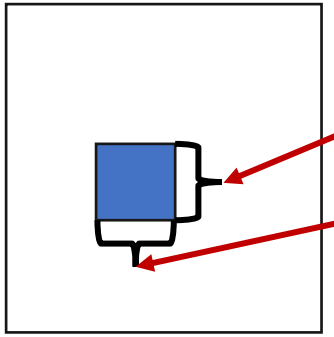


B

What are the elements of C that will be output by SMP Bid_y, Bid_x?

A





C

C[$Bid_y * \frac{N}{b}$: $Bid_y * \frac{N}{b}$ +b-1][$Bid_x * \frac{N}{b}$: $Bid_x * \frac{N}{b}$ +b-1]

SMP Bid_y, Bid_x produces the output block Bid_y, Bid_x

Block Matrix Multiplication

- In $Bid_y, Bid_x, Tid_y, Tid_x$ {
- For $k = 0$ to $\frac{N}{b} - 1$
  - Read A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$Tid_x$]
  - Read B[$k * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$]
  - Syncthreads()
  - For l = 0 to $b - 1$:
    - C[$Bid_y * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$] +=
    - A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$l$]* B[$k * \frac{N}{b} + l$][$Bid_x * \frac{N}{b} + Tid_x$]
  - Syncthreads()
- Store C[$Bid_y * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$] }

Looking at the store instruction of this code, can you explain how this is happening?

Block Matrix Multiplication

- In $Bid_y, Bid_x, Tid_y, Tid_x$ {
- For $k = 0$ to $\frac{N}{b} - 1$
  - Read A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$Tid_x$]
  - Read B[$k * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$]
  - Syncthreads()
  - For l = 0 to $b - 1$:
    - C[$Bid_y * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$] +=
    - A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$l$]* B[$k * \frac{N}{b} + l$][$Bid_x * \frac{N}{b} + Tid_x$]
  - Syncthreads()
- Store C[$Bid_y * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$] }

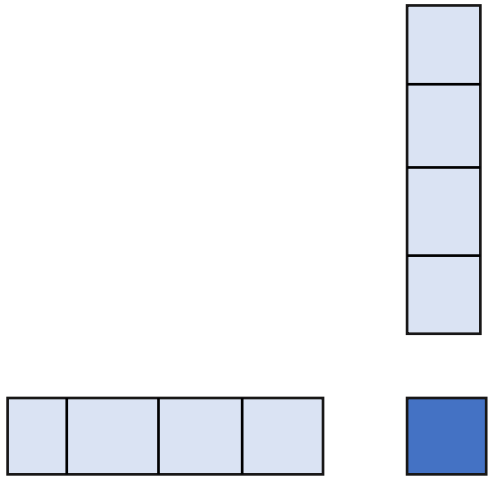
Looking at the store instruction of this code, can you explain how this is happening?

Tid_y, Tid_x vary from 0 to $b - 1$

Block Matrix Multiplication

- In $Bid_y, Bid_x, Tid_y, Tid_x$ {
- For $k = 0$ to $\frac{N}{b} - 1$
  - Read A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$Tid_x$]
  - Read B[$k * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$]
  - Syncthreads()
  - For l = 0 to $b - 1$:
    - C[$Bid_y * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$] +=
    - A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$l$]* B[$k * \frac{N}{b} + l$][$Bid_x * \frac{N}{b} + Tid_x$]
  - Syncthreads()
- Store C[$Bid_y * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$] }

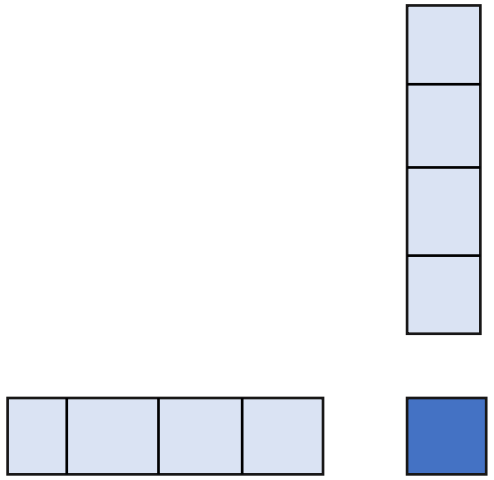
We go block by block as before.



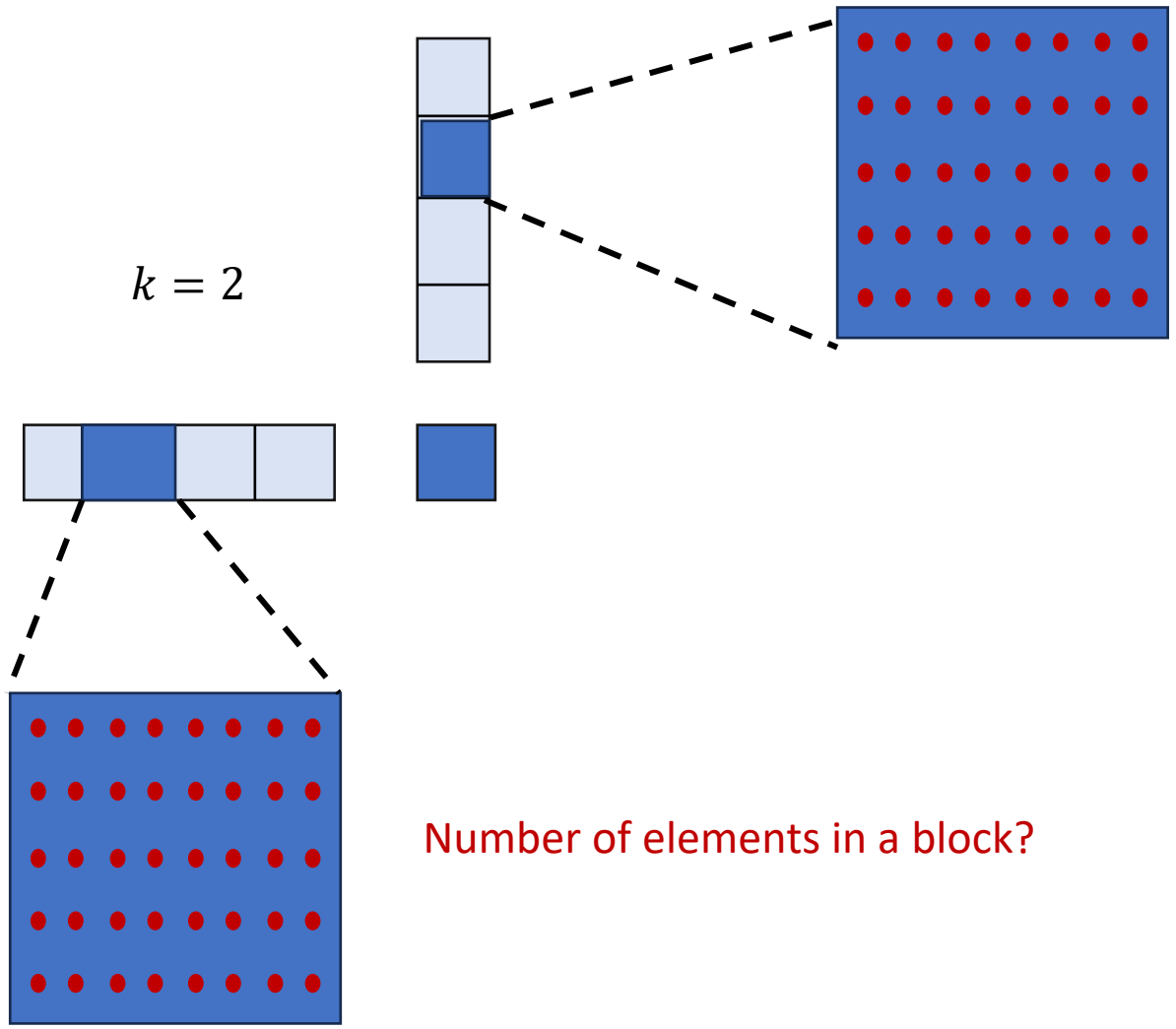
Block Matrix Multiplication

- In $Bid_y, Bid_x, Tid_y, Tid_x$ {
- For $k = 0$ to $\frac{N}{b} - 1$
  - Read A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$Tid_x$]
  - Read B[$k * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$]
  - Syncthreads()
  - For l = 0 to $b - 1$:
    - C[$Bid_y * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$] +=
    - A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$l$]* B[$k * \frac{N}{b} + l$][$Bid_x * \frac{N}{b} + Tid_x$]
  - Syncthreads()
- Store C[$Bid_y * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$] }

For each k, we fetch the blocks of A and B first before performing computations



Block Matrix Multiplication



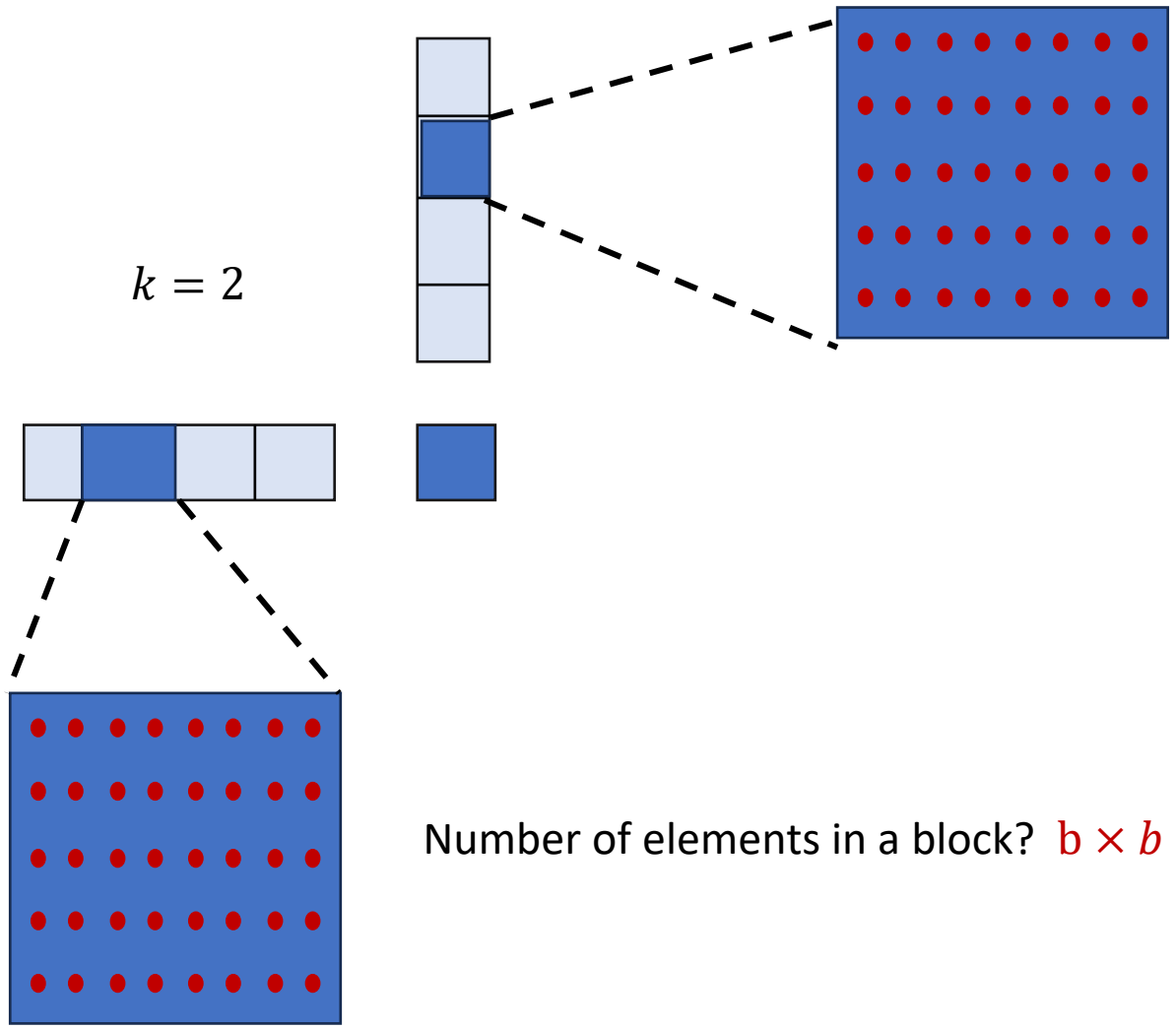
For a given K, each thread fetches exactly 1 element from A and B blocks

Read A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$Tid_x$]

Read B[$k * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$]

Number of elements in a block?

Block Matrix Multiplication



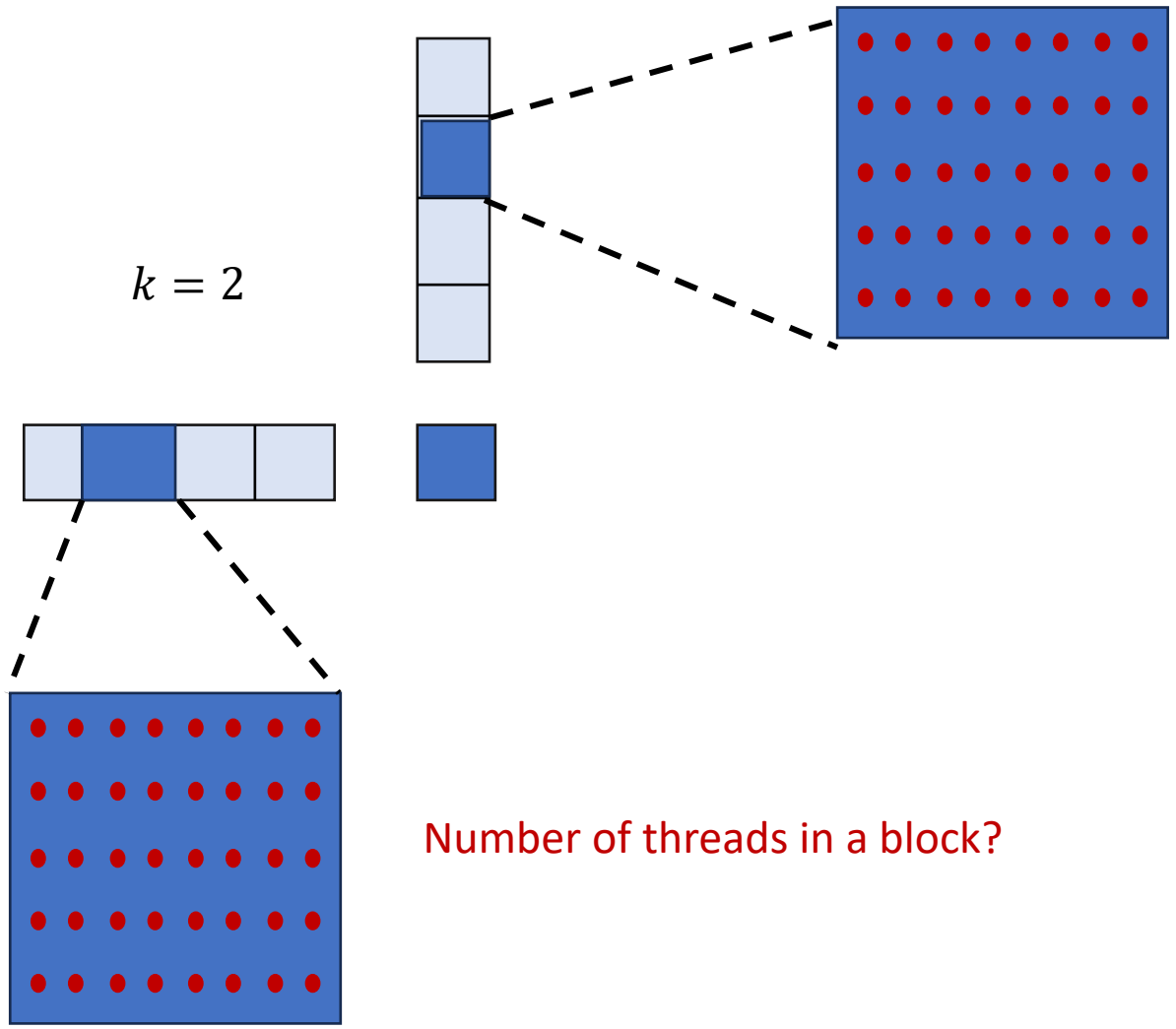
For a given K, each thread fetches exactly 1 element from A and B blocks

Read A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$Tid_x$]

Read B[$k * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$]

Number of elements in a block? $b \times b$

Block Matrix Multiplication



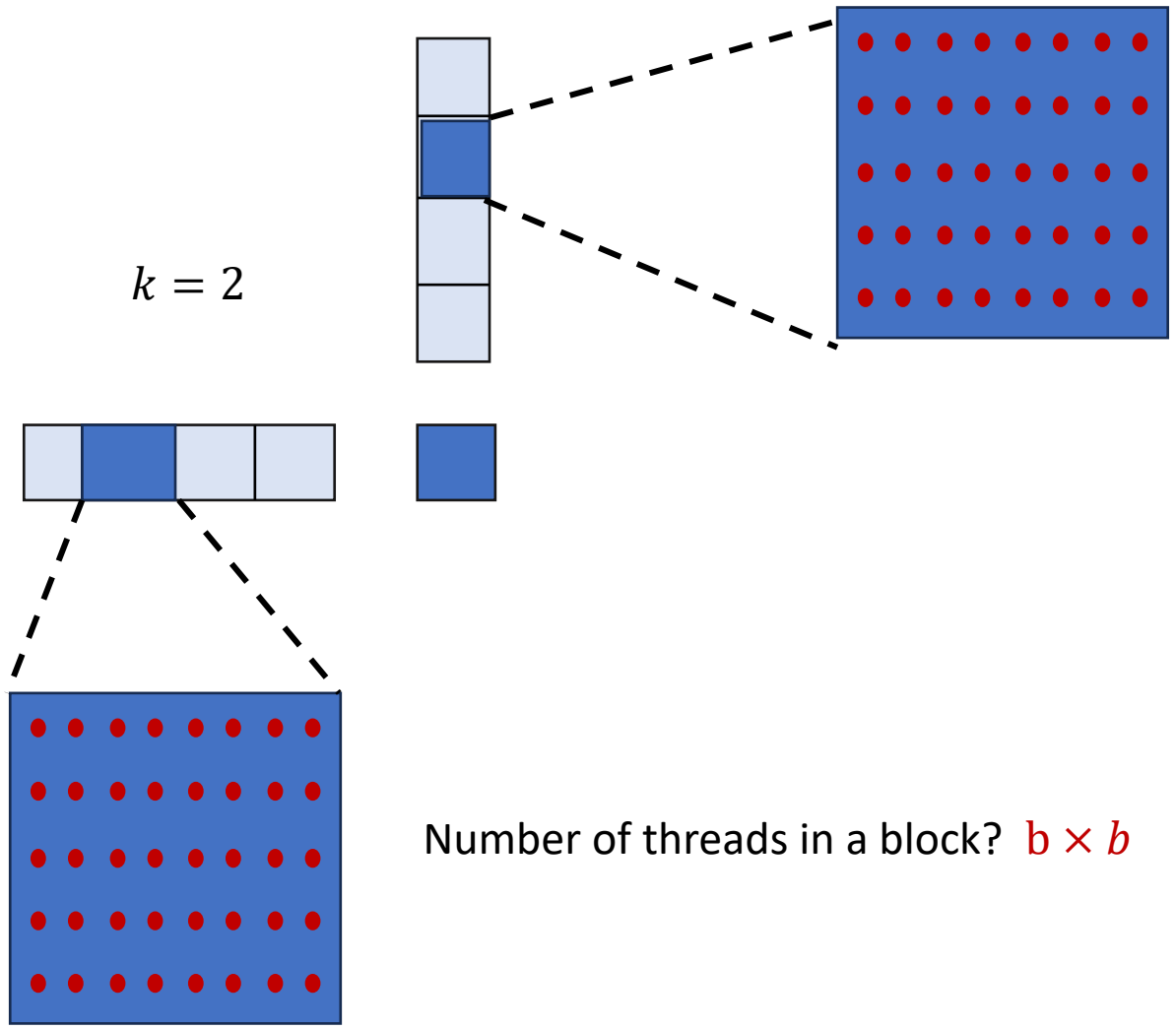
For a given K, each thread fetches exactly 1 element from A and B blocks

Read A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$Tid_x$]

Read B[$k * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$]

Number of threads in a block?

Block Matrix Multiplication



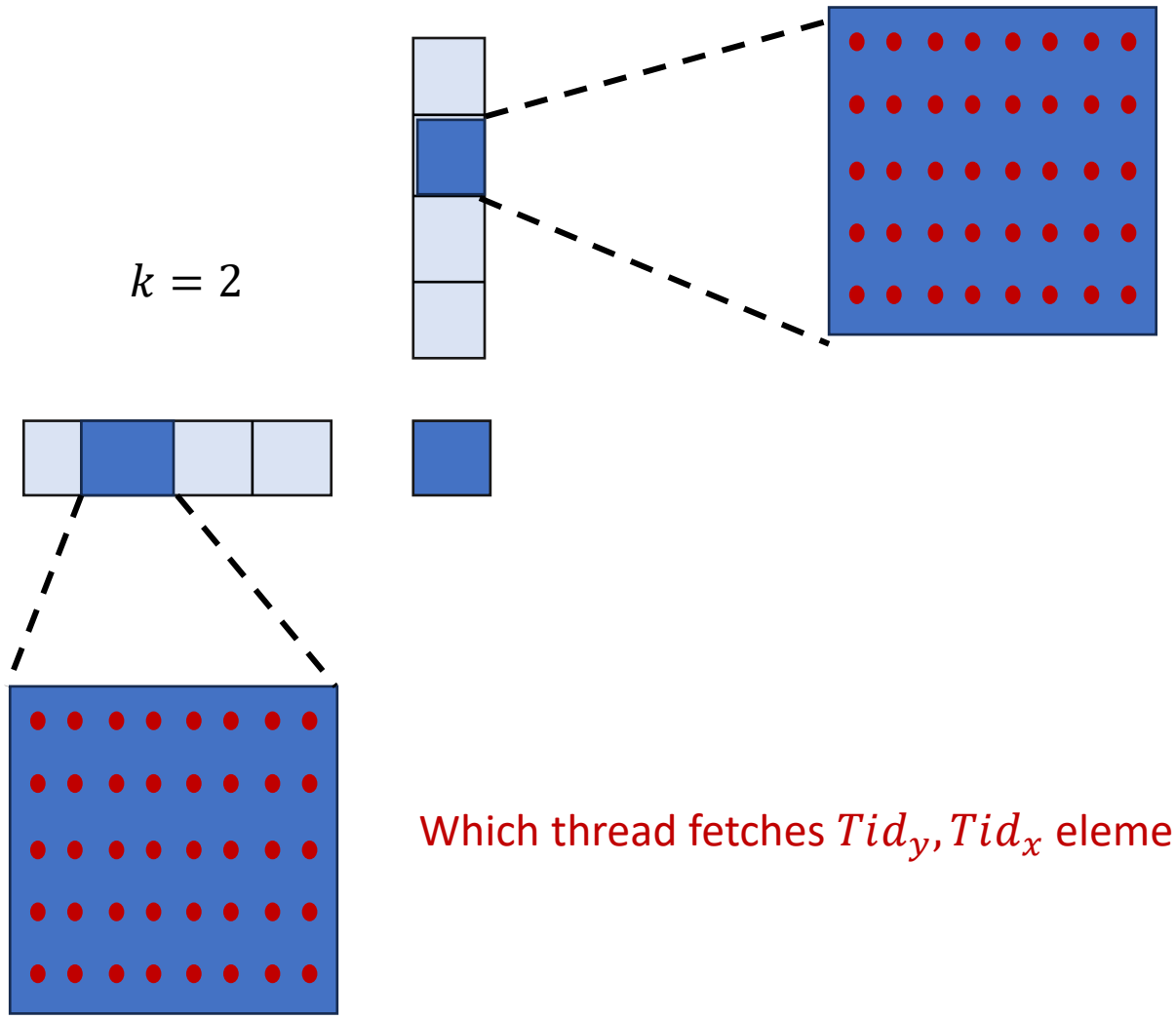
For a given K, each thread fetches exactly 1 element from A and B blocks

Read A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$Tid_x$]

Read B[$k * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$]

Number of threads in a block? $b \times b$

Block Matrix Multiplication



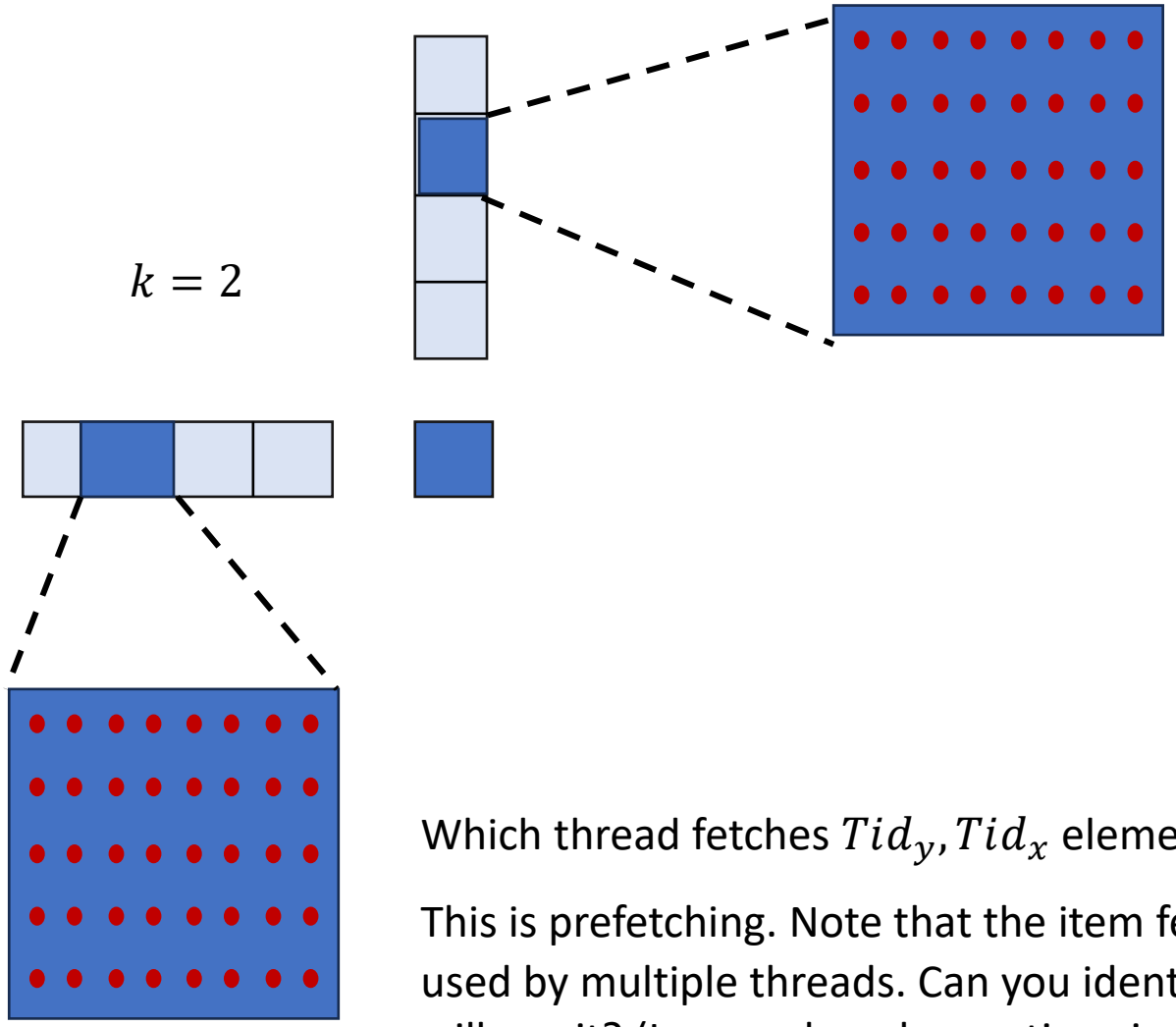
For a given K, each thread fetches exactly 1 element from A and B blocks

Read A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$Tid_x$]

Read B[$k * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$]

Which thread fetches Tid_y, Tid_x element of A (or B)?

Block Matrix Multiplication



For a given K, each thread fetches exactly 1 element from A and B blocks

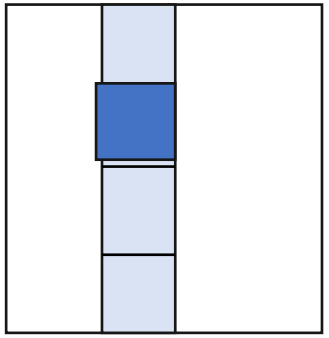
Read A[$Bid_y * \frac{N}{b} + Tid_y$][k*N/b+$Tid_x$]

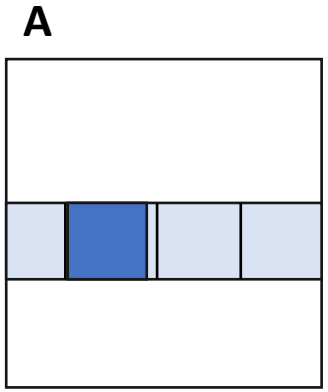
Read B[$k * \frac{N}{b} + Tid_y$][$Bid_x * \frac{N}{b} + Tid_x$]

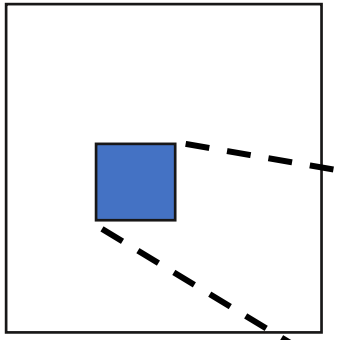
Which thread fetches Tid_y, Tid_x element of A (or B)? Tid_y, Tid_x

This is prefetching. Note that the item fetched by Tid_y, Tid_x will be used by multiple threads. Can you identify the range of threads that will use it? (I may ask such questions in the exam)

Block Matrix Multiplication







For a given K, each thread computes exactly 1 element of C block

$$C[Bid_y * \frac{N}{b} + Tid_y][Bid_x * \frac{N}{b} + Tid_x] \mathrel{+}=$$
$$A[Bid_y * \frac{N}{b} + Tid_y][k*N/b+l]*$$
$$B[k * \frac{N}{b} + l][Bid_x * \frac{N}{b} + Tid_x]$$

HW: check the equation above to see if it correct or not.

SMP Bid_y, Bid_x produces the output block Bid_y, Bid_x

Data Parallelism – Approaches

- #1 Partition Output data
- Notice how we thought about how to divide the output that is being produced among the parallel blocks and threads?
- Notice how there are no write conflicts and each block can execute independently?
  - Threads can too for the most part, but need synchronization steps in between.
- This approach, if applicable, leads to the best parallelism on GPUs

Data Parallelism - Approaches

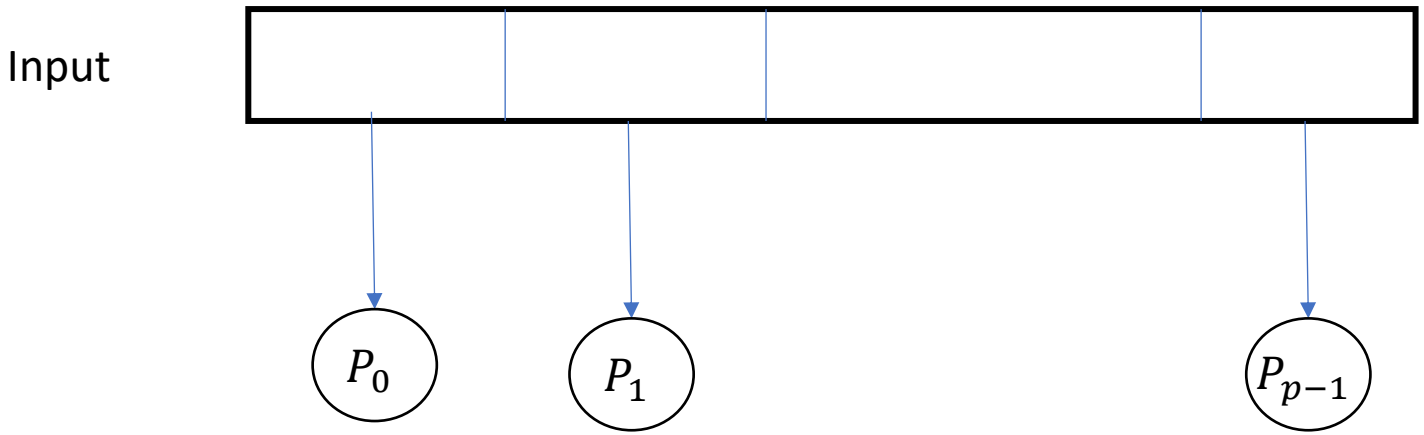
- #2 Partition Input data
  - Each task is responsible for performing “all” operations on the partition

Data Parallelism - Approaches

- #2 Partition Input data
  - Each task is responsible for performing “all” operations on the partition

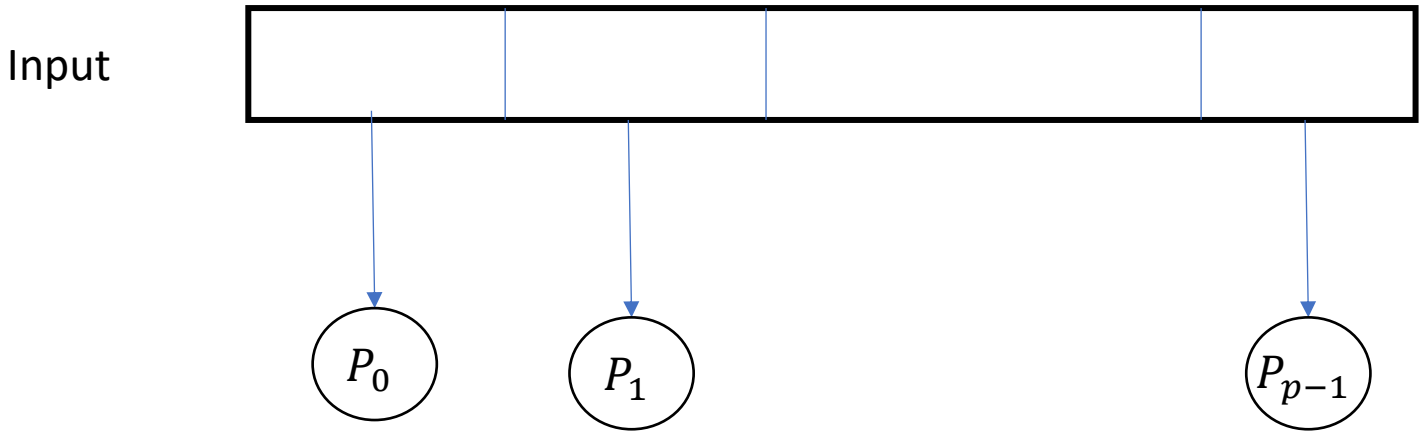
Data Parallelism - Approaches

- #2 Partition Input data
  - Each task is responsible for performing “all” operations on the partition



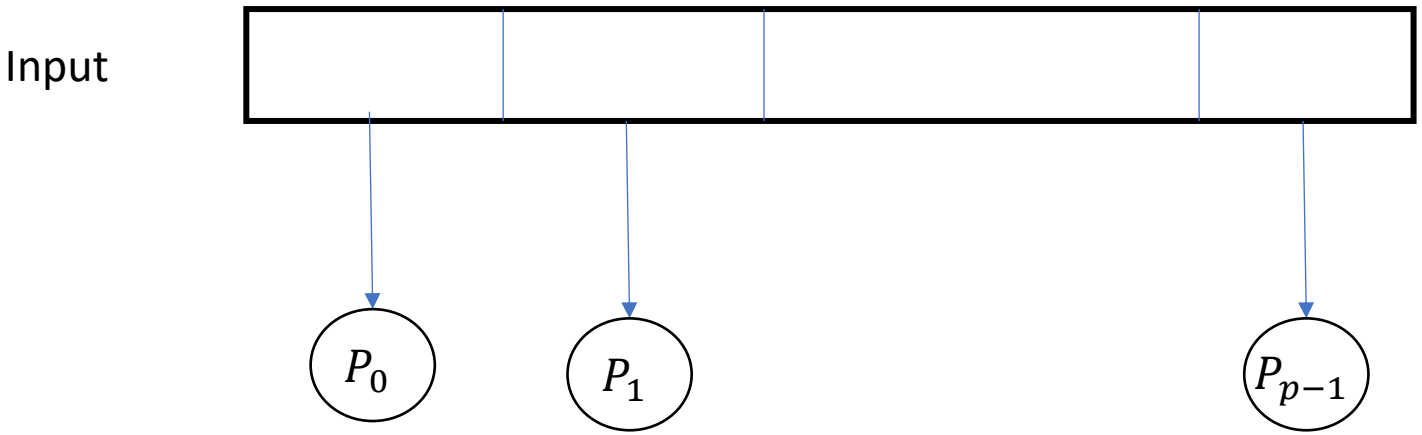
Data Parallelism - Approaches

- #2 Partition Input data
  - Tasks can either directly produce the final output
  - Or produce intermediate output which require another set of tasks to produce the final output



Data Parallelism - Approaches

- #2 Partition Input data
  - Suitable when outputs cannot be independently computed
    - Example: Aggregation functions, Hashing, …



Sum of Elements in an Array

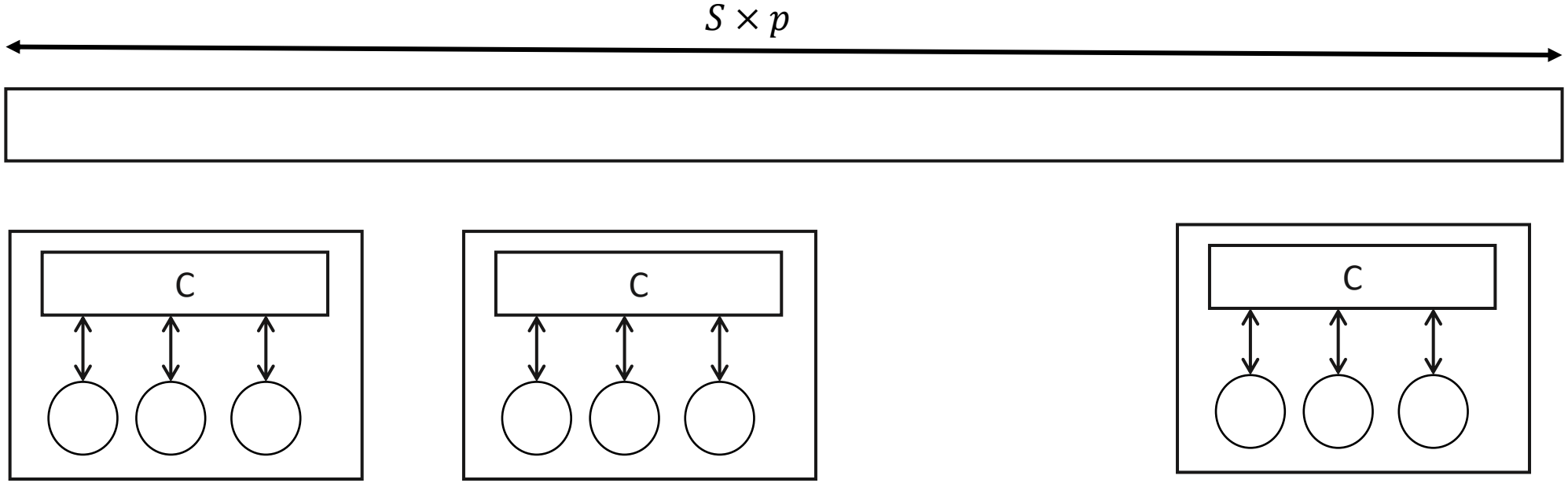
- Input: An array A of N elements
- Output: An array with element $N-1$ storing $\sum_i A[i]$
- Assume $N = S \times p$
- Resources:
  - https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-39-parallel-prefix-sum-scan-cuda
  - https://courses.cs.washington.edu/courses/cse332/21sp/lectures/19-ParallelPrefix.pdf
  - They discuss the more general (and complicated) version called prefix sum
  - But consider only single SMP
  - Ungraded HW assignment: Review the algorithm presented on this link, may ask in exam

Sum of Array

- Input: [1, 3, 5, 7, 9]
- Output: [x, x, x, x, 25]

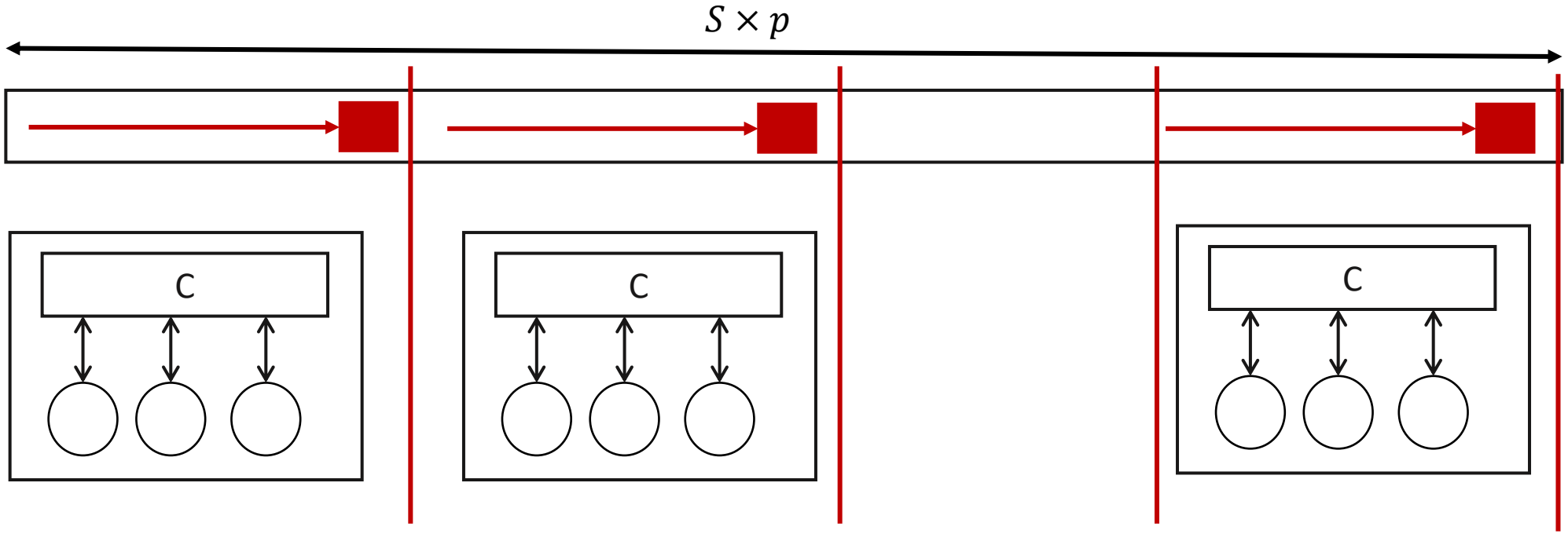
Sum of Array

- Key Idea:
  - Tree like computation to achieve independent computations
  - Two phase algorithm, 1 within a SMP and 1 across SMPs



Sum of Array

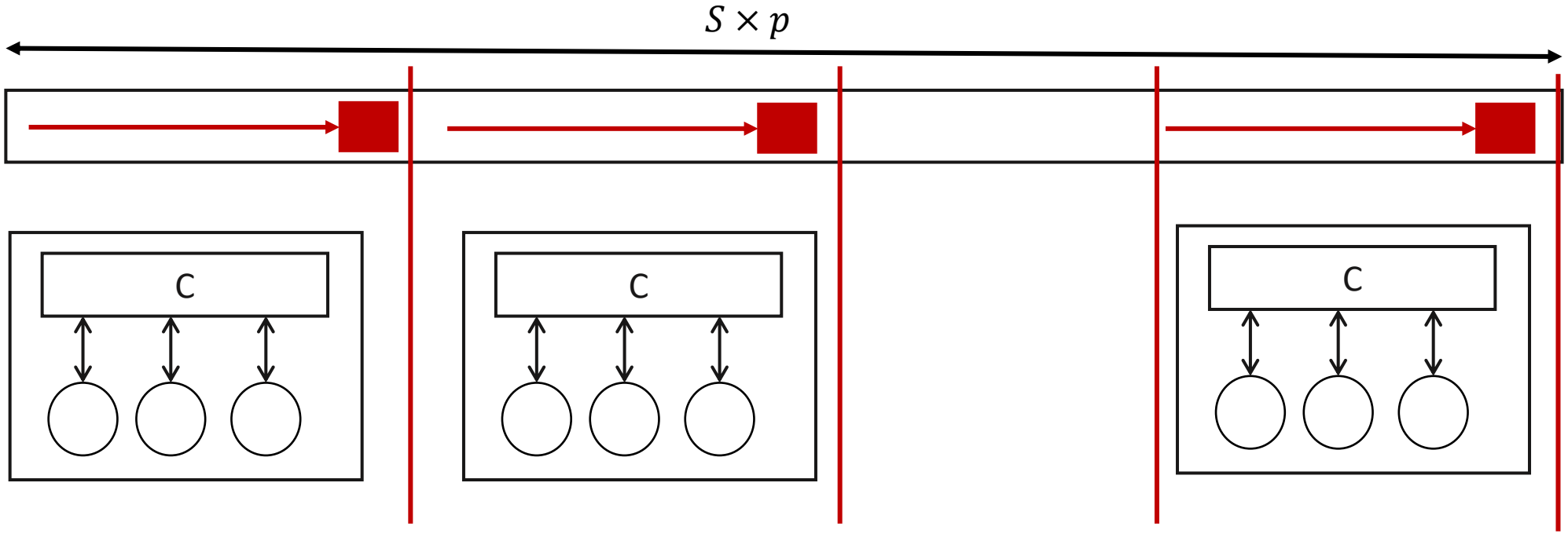
- Key Idea:
  - Tree like computation to achieve independent computations
  - Two phase algorithm, 1 within a SMP and 1 across SMPs



Phase I – Compute the sum of p elements per Block

Sum of Array

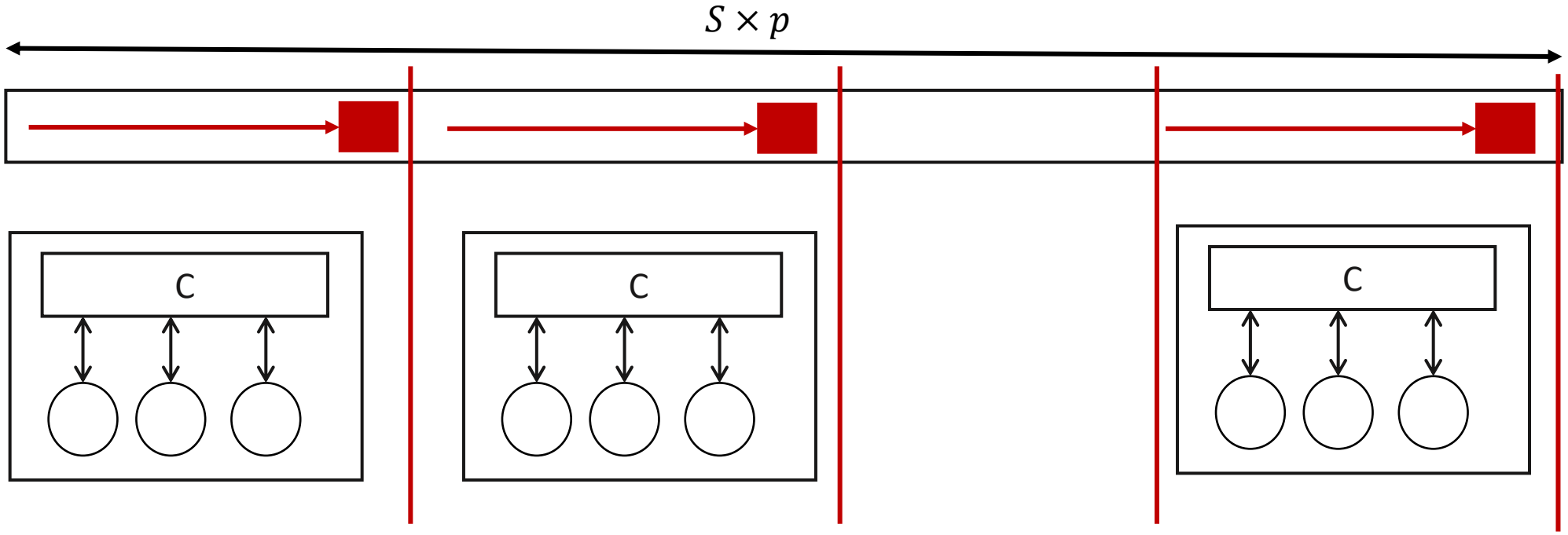
- Key Idea:
  - Tree like computation to achieve independent computations
  - Two phase algorithm, 1 within a SMP and 1 across SMPs



How many sums will remain after Phase I?

Sum of Array

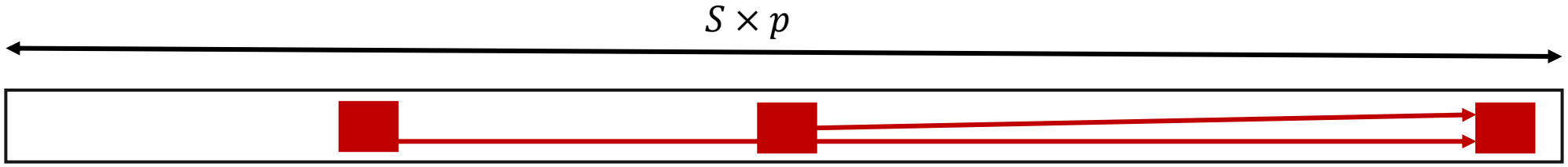
- Key Idea:
  - Tree like computation to achieve independent computations
  - Two phase algorithm, 1 within a SMP and 1 across SMPs

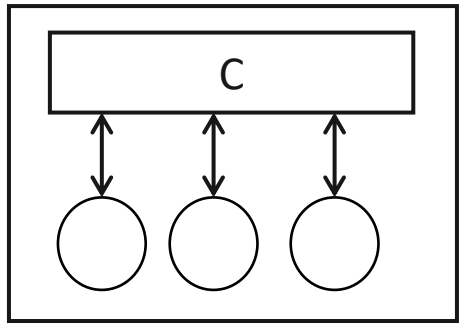


How many sums will remain after Phase I? S

Sum of Array

- Key Idea:
  - Tree like computation to achieve independent computations
  - Two phase algorithm, 1 within a SMP and 1 across SMPs





Phase II – Using one SMP compute the sum of the S partial sums
(Assume, $S < p$)

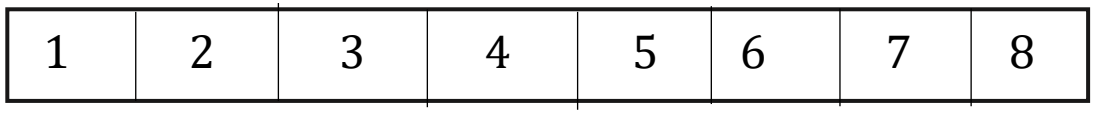
Sum of Array

- Phase I: Calculate the sum of p elements and store into the last location of the array
  - Use p processors

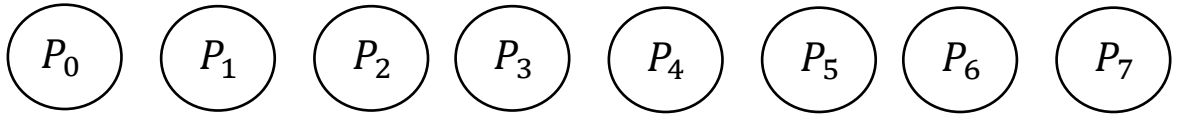
- Recursive Doubling Algorithm

Sum of Array

Data Array



Processors



Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

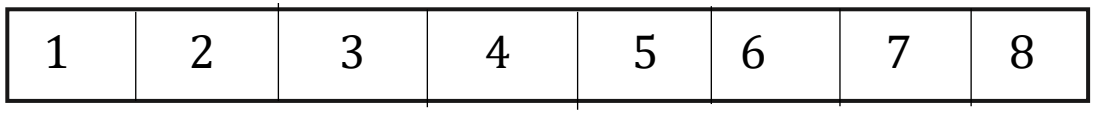
- $k = 1\ to \log_2 p$

In step k
We say, Processor i is active if $i = m2^k - 1$, where m is an integer

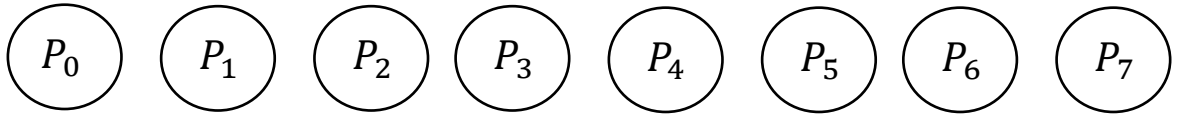
Processor i
$A[i] \leftarrow A[i] + A[i\ - 2^{k-1}]$

Sum of Array

Data Array



Processors



Step 1: $k = 1$ Active Processors??

Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

- $k = 1 \; to \log_2 p$

In step k

We say, Processor i is active if $i = m2^k - 1$, where m is an integer

Processor i

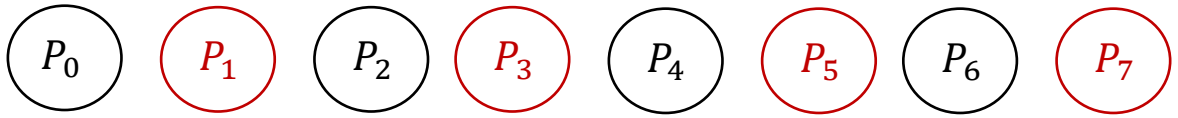
$A[i] \leftarrow A[i] + A[i - 2^{k-1}]$

Sum of Array

Data Array

1	2	3	4	5	6	7	8

Processors



Step 1: $k = 1$ Active Processors?? 1, 3, 5, 7

Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

- $k = 1 \; to \log_2 p$

In step k

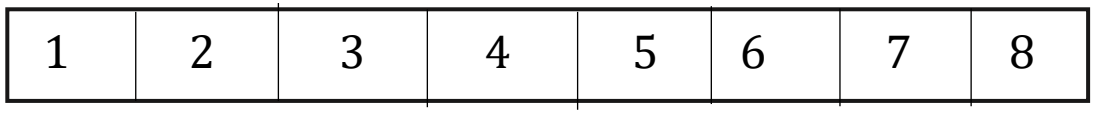
We say, Processor i is active if $i = m2^k - 1$, where m is an integer

Processor i

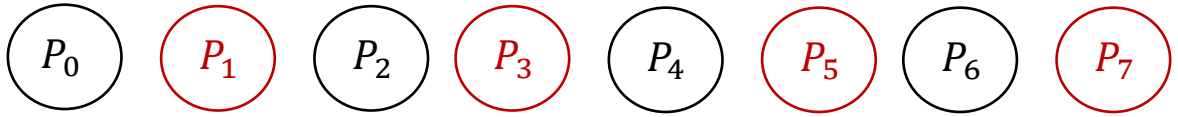
$A[i] \leftarrow A[i] + A[i - 2^{k-1}]$

Sum of Array

Data Array



Processors



Step 1: $k = 1$ Result of Computation?

Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

- $k = 1 \; to \log_2 p$

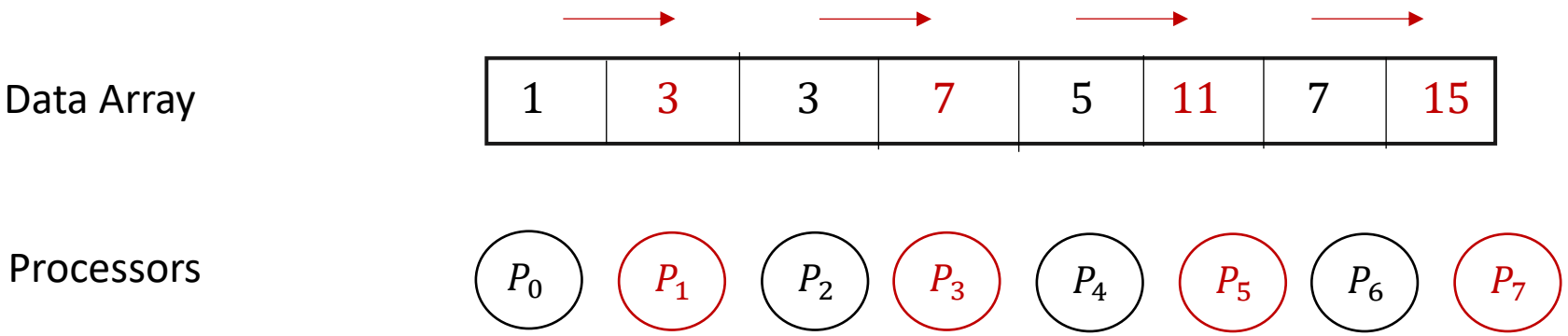
In step k

We say, Processor i is active if $i = m2^k - 1$, where m is an integer

Processor i

$A[i] \leftarrow A[i] + A[i - 2^{k-1}]$

Sum of Array



Step 1: $k = 1$ Result of Computation?

Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

- $k = 1 \; to \log_2 p$

In step k

We say, Processor i is active if $i = m2^k - 1$, where m is an integer

Processor i

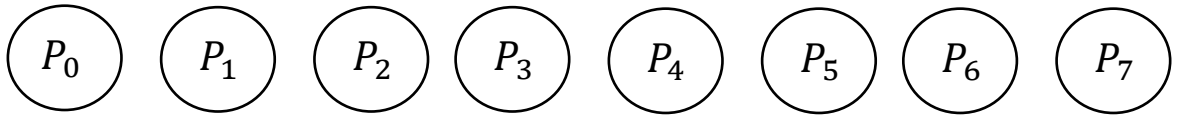
$A[i] \leftarrow A[i] + A[i - 2^{k-1}]$

Sum of Array

Data Array

1	3	3	7	5	11	7	15

Processors



Step 1: $k = 2$ Active Processors??

Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

- $k = 1\ to \log_2 p$

In step k

We say, Processor i is active if $i = m2^k - 1$, where m is an integer

Processor i

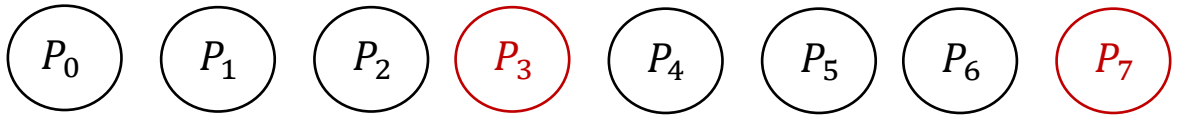
$A[i] \leftarrow A[i] + A[i - 2^{k-1}]$

Sum of Array

Data Array

1	3	3	7	5	11	7	15

Processors



Step 1: $k = 2$ Active Processors?? 3,7

Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

- $k = 1\ to \log_2 p$

In step k

We say, Processor i is active if $i = m2^k - 1$, where m is an integer

Processor i

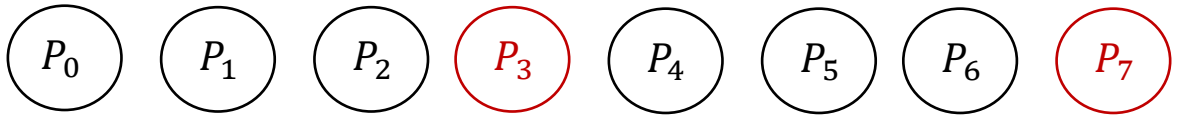
$A[i] \leftarrow A[i] + A[i - 2^{k-1}]$

Sum of Array

Data Array

1	3	3	7	5	11	7	15

Processors



Step 1: $k = 2$ Computations?

Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

- $k = 1\ to \log_2 p$

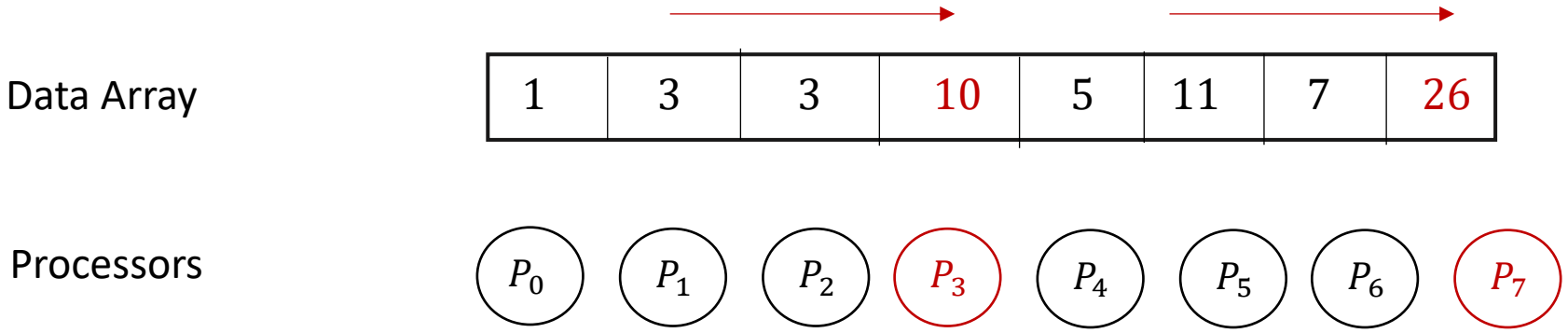
In step k

We say, Processor i is active if $i = m2^k - 1$, where m is an integer

Processor i

$A[i] \leftarrow A[i] + A[i - 2^{k-1}]$

Sum of Array



Step 1: $k = 2$ Computations?

Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

- $k = 1\ to \log_2 p$

In step k

We say, Processor i is active if $i = m2^k - 1$, where m is an integer

Processor i

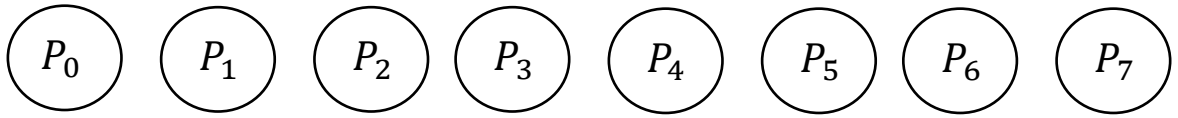
$A[i] \leftarrow A[i] + A[i - 2^{k-1}]$

Sum of Array

Data Array

1	3	3	10	5	11	7	26

Processors



Step 1: $k = 3$ Active Processors?

Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

- $k = 1\ to \log_2 p$

In step k

We say, Processor i is active if $i = m2^k - 1$, where m is an integer

Processor i

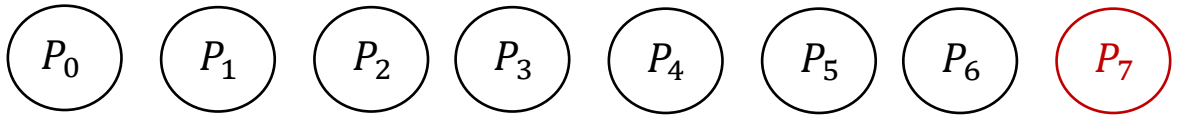
$A[i] \leftarrow A[i] + A[i - 2^{k-1}]$

Sum of Array

Data Array

1	3	3	10	5	11	7	26

Processors



Step 1: $k = 3$ Active Processors? 7

Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

- $k = 1\ to \log_2 p$

In step k

We say, Processor i is active if $i = m2^k - 1$, where m is an integer

Processor i

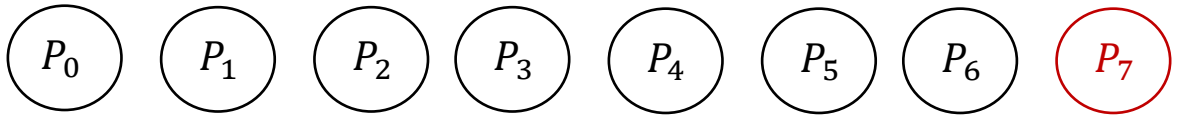
$A[i] \leftarrow A[i] + A[i - 2^{k-1}]$

Sum of Array

Data Array

1	3	3	10	5	11	7	26

Processors



Step 1: $k = 3$ Computations?

Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

- $k = 1\ to \log_2 p$

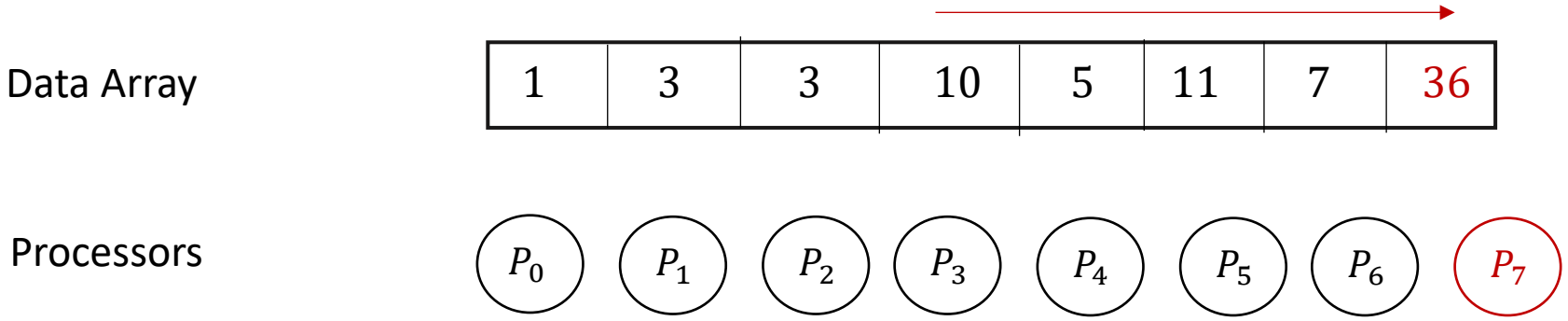
In step k

We say, Processor i is active if $i = m2^k - 1$, where m is an integer

Processor i

$A[i] \leftarrow A[i] + A[i\ - 2^{k-1}]$

Sum of Array



Step 1: $k = 3$ Computations?

Recursive Doubling

Algorithm runs in $\log_2 p$ steps,

- $k = 1 \; to \log_2 p$

In step k

We say, Processor i is active if $i = m2^k - 1$, where m is an integer

Processor i

$A[i] \leftarrow A[i] + A[i - 2^{k-1}]$

Sum of Array – Phase II

- Use S processors to calculate the sum of S elements produced in Phase I

- Ungraded HW Assignment: Write a GPU code for this algorithm using BlockID and ThreadID notation that we have been using

- Ungraded HW Assignment: Can you now improve Matrix Vector Multiplication using this approach?

Data Parallelism – Hybrid Approaches

- #3 Partition both Input and Output data
- For each output partition, further partition the inputs
- Blocked MM
  - Input blocks can be further assigned to different SMPs, achieving input parallelism in addition to output parallelism

Data Parallelism

- We will discuss a couple more algorithms in the next class

Outline

- Data Parallel Programming
  - Data Parallel Programming Approaches
- Parallel Program Analysis

Techniques to Improve Application Performance

- Compute Parallelization
  - Decompose tasks to enable concurrent processing

- Memory Optimizations
  - Optimizations addressing latency
  - Optimizations targeted toward maximizing bandwidth utilization
  - Optimizations that reduce the amount of data transfer

But…

- Before we dig into parallelizing an application, how do we know if it is worth parallelizing?

- Performance Metrics discussed till now can reason about an algorithm as a whole
  - Cannot reason about what kinds of benefits we may get from parallelization
  - Cannot reason about the maximum benefit that can be obtained and if it is worth spending the effort on parallelization or not

Speedup

- Improvement in time due to parallelization

$$\text{Speedup} = \frac{\text{Serial time (on a uniprocessor system)}}{\text{Time after parallelization}}$$

- Serial Time = 1
- Parallel Time = 0.8

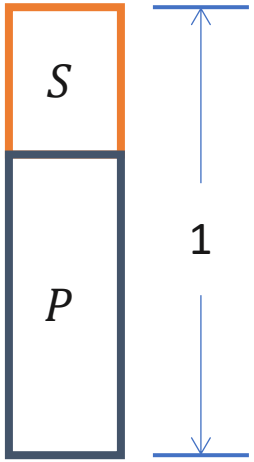
- Speedup = $\frac{1}{0.8} = 1.25$

Amdahl's Law

- Amdahl's Law — Limit on speedup achievable when a program is run on a parallel machine
- Given an input program
  - Total execution time: $S + P$

Serial Portion

Parallelizable Portion



  - Time on a uni-processor machine: $1 = S + P$
  - Time on a parallel machine: $S + P/f$

    f = speedup factor

Amdahl's Law

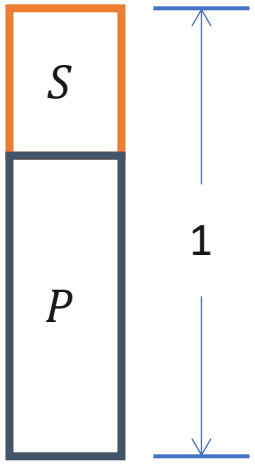
$$= \frac{\text{Serial Time}}{\text{Parallel Time}}$$

$$= \frac{S + P}{S + P/f}$$

$$= \frac{1}{S + P/f}$$

$$\leq \frac{1}{S}$$

f = speedup factor

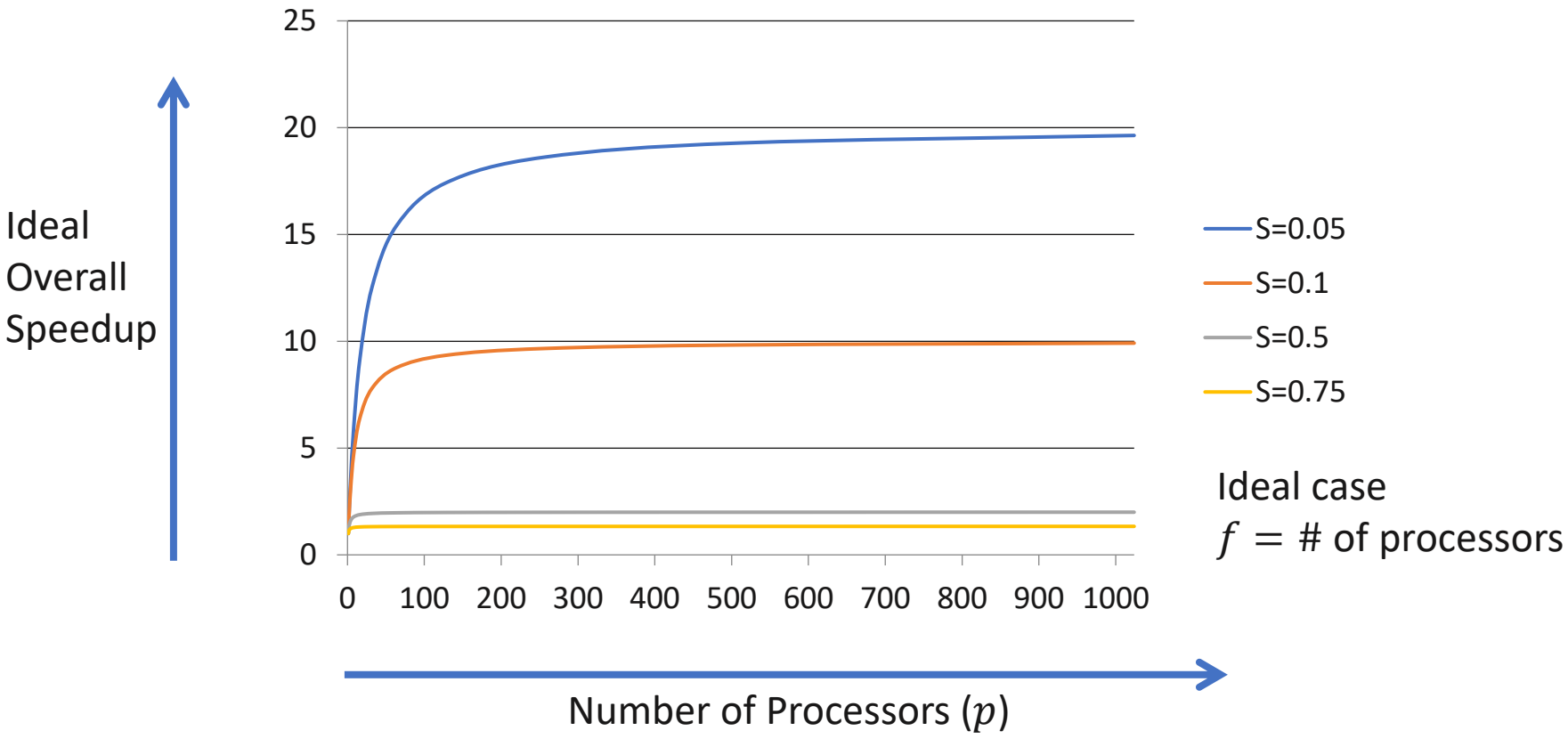


If serial portion is 50 %, then the **overall speedup ≤ 2**

Note: Speedup factor (f) ≤ # of processors

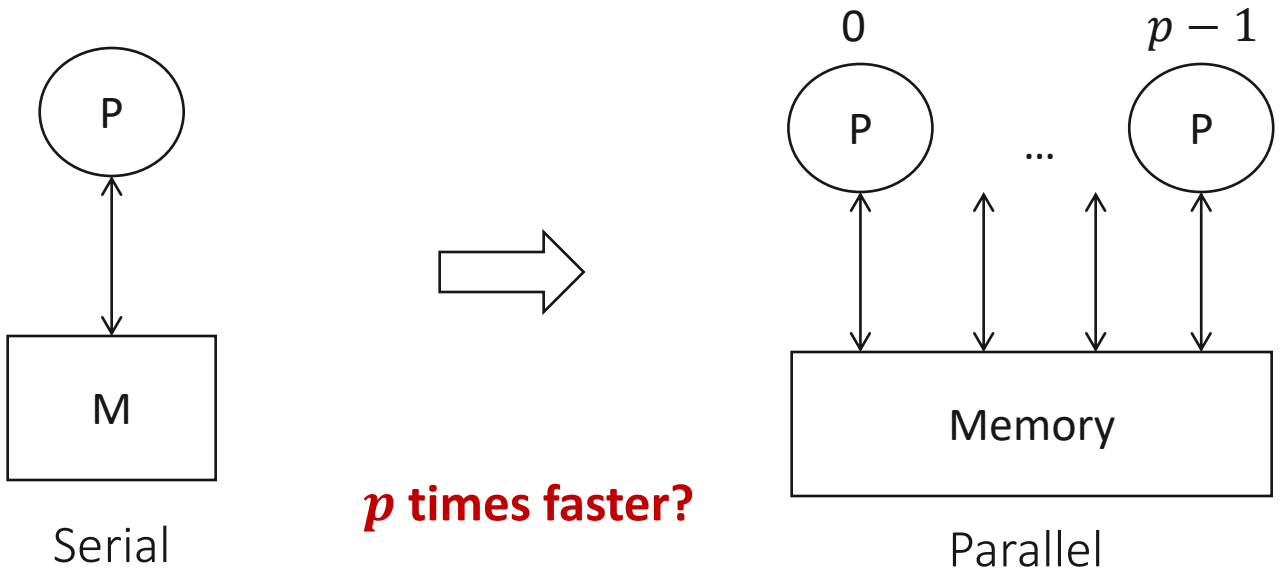
Amdahl's Law

$$\text{Overall Speedup} = \frac{1}{S + P/f} = \frac{1}{S + (1 - S)/f} \rightarrow \frac{1}{S}$$



Scalability (1)

Does performance (execution time) improve as we use more resources (processors)



Scalability (2)

$$\text{Speedup} = \frac{\text{Serial time (on a uniprocessor system)}}{\text{Parallel time using } p \text{ processors}}$$

If speedup = $O(p)$, then it is a **scalable** solution

Overheads in Parallel Computation

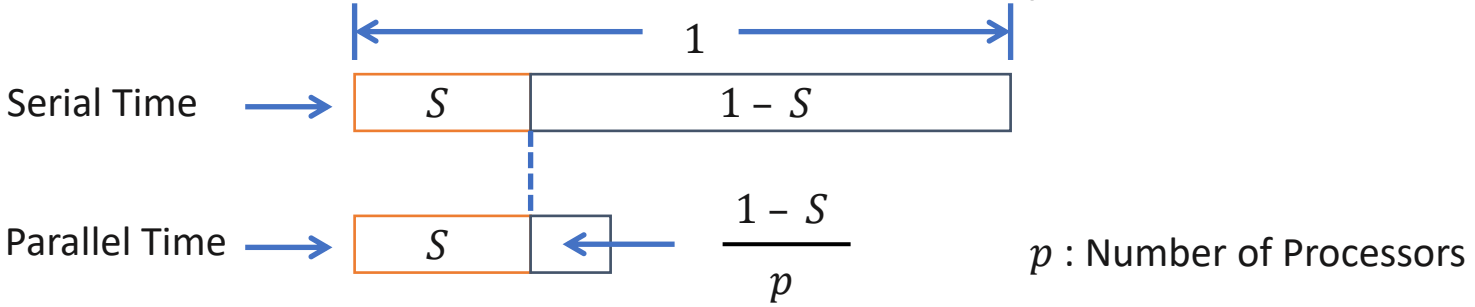
- Communication
- Synchronization
- Load balance (processors may idle)
- Resource contention

Scaled Speedup (Gustafson's Law) (1)

- Amdahl's Law — Serial portion of code limits performance
  (As we use more processors)
- As we use more processors
  → we use more data
  → e.g., more fine grained model
- E.g. Processing $N \times N$ image
  Using p processors
  As we increase p we usually increase image size

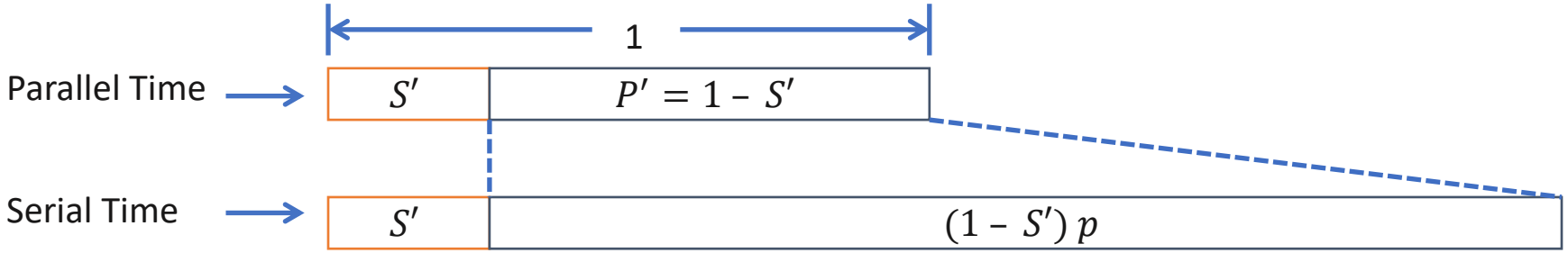
Scaled Speedup (Gustafson's Law) (2)

Amdahl's Law: Fixed amount of computations



p : Number of Processors

Gustafson's Law: Increase p and amount of computations



If parallelism scales linearly with p, number of processors

$$\text{Scaled Speedup} = \frac{\text{Serial time}}{\text{Parallel time}} = \frac{S' + (1 - S')p}{S' + P'} = \frac{S' + (1 - S')p}{1}$$

Scaled Speedup (Gustafson's Law) (3)

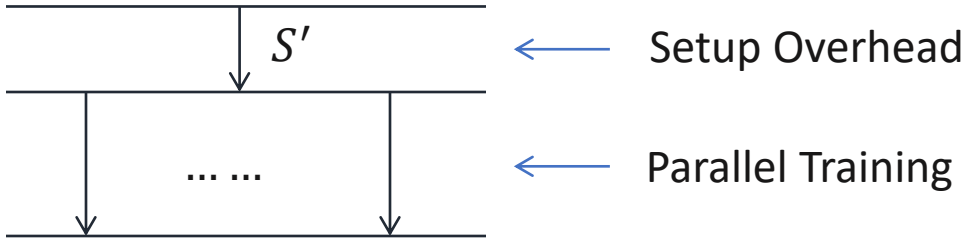
Gustafson's Law

$$\text{Scaled Speedup} = S' + (1 - S')p$$
$$\approx (1 - S')p$$

$S' = 0 \quad \rightarrow \quad$ Scaled Speedup = Ideal speedup (p)

$S' = 0.5 \quad \rightarrow \quad$ Scaled Speedup = $0.5\,p$, 50% of Ideal speedup

Example: ML Training



Scaled Speedup $\propto (1 - S') \times$ # of threads

Scaled Speedup (Gustafson's Law) (4)

Gustafson's Law

- If we increase
  - the number of processors (p)
  - the amount of parallelizable portion of computations

- Scaled speedup is limited by the fraction of program that can be parallelized (higher the fraction, higher the speedup).

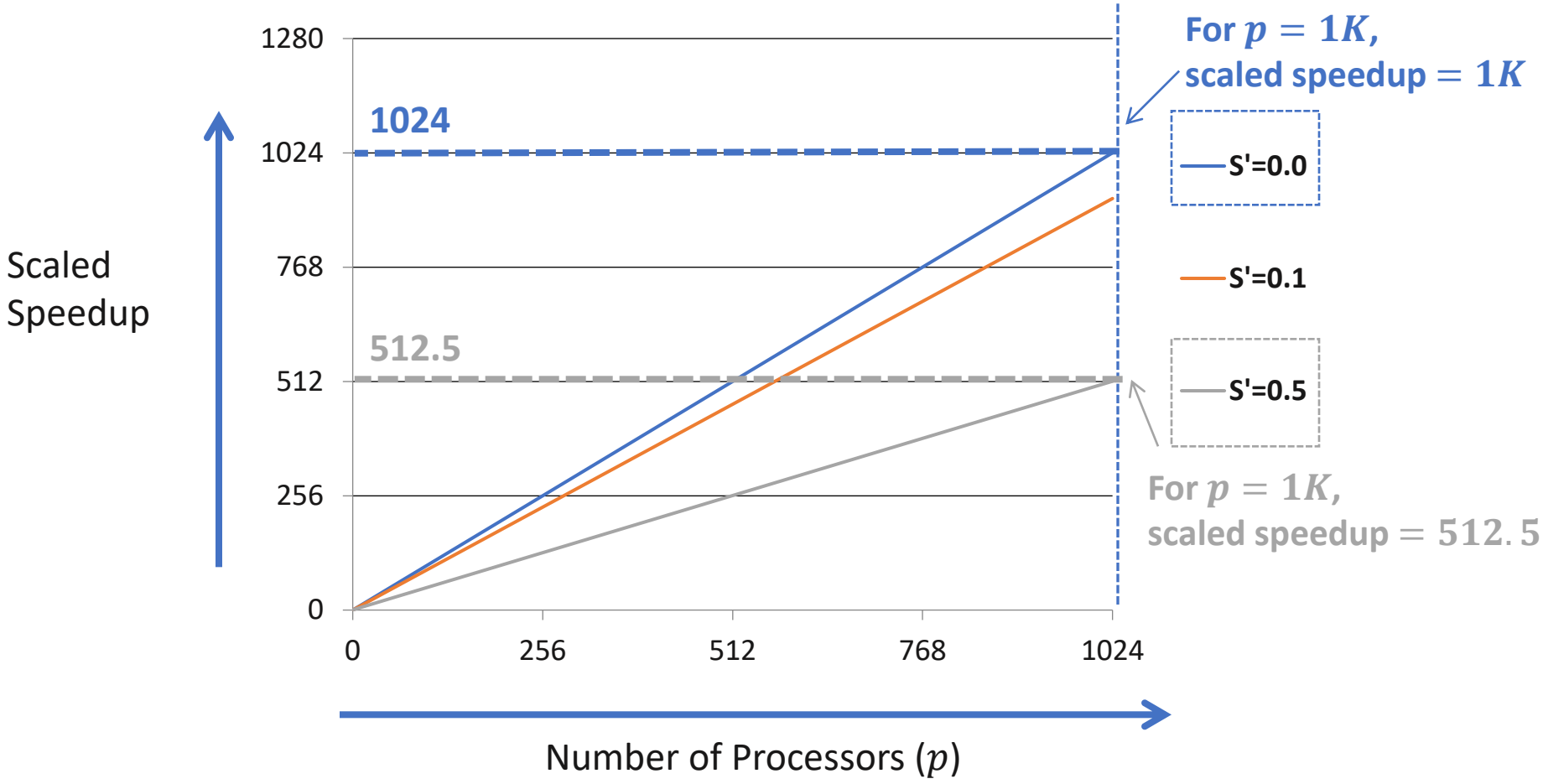
Scaled Speedup (Gustafson's Law) (5)

- $S = 0.5$
- $p = 32$

- Amdahl's Law – Cannot achieve more than 2x speedup

- Gustafson's Law – Can achieve 16x speedup

Scaled Speedup (Gustafson's Law) (6)



Strong or Weak Scaling

- Strong Scaling
  - Governed by Amdahl's Law
  - The number of processors is **increased** while the problem size **remains constant**
  - Results in a **reduced** workload per processor

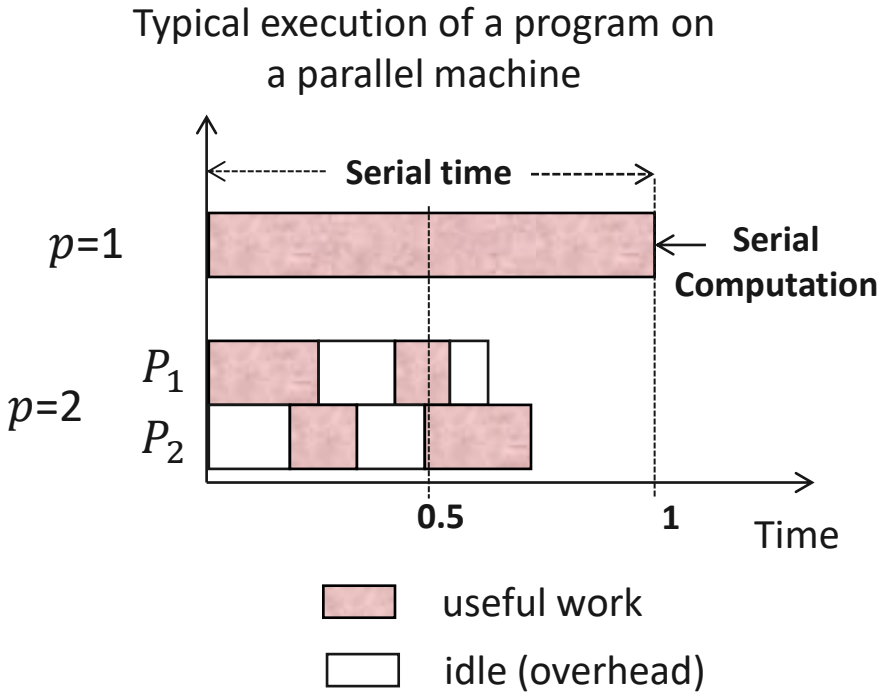
- Weak Scaling
  - Governed by Gustafson's Law
  - **Both** the number of processors and the problem size are **increased**
  - Results in a **constant** workload per processor

Performance (1)

Efficiency

- Question: If we use p processors, is speedup = p ?

- Efficiency $\triangleq$ Fraction of time a processor is usefully employed during the computation

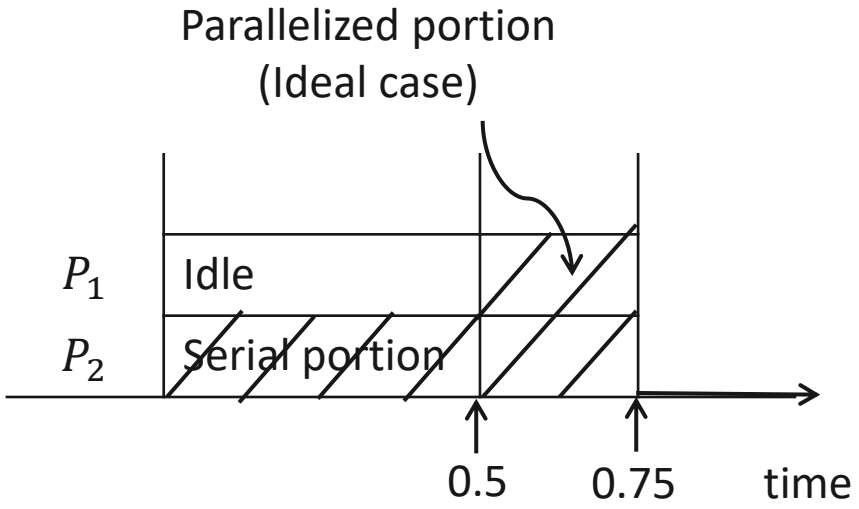


- E = Speedup / # of processors used
  - E is the average efficiency over all the processors
  - Efficiency of each processor can be different from the average value

Performance (2)

Ex. $S = 0.5$
$P = 0.5$

2 processor system



Speedup $= \frac{1}{0.75} = {}^{4}/_{3}$

(Average) Efficiency $= \frac{{}^{4}/_{3}}{2} = {}^{2}/_{3}$

Efficiency of $P_1 = \frac{0.25}{0.75} = {}^{1}/_{3}$

Efficiency of $P_2 = \frac{0.75}{0.75} = 1$

(Average) Efficiency $= \frac{{}^{1}/_{3}+1}{2} = 2/3$

Performance (3)

- Cost = Total amount of work done by a parallel system
  = Parallel Execution Time x Number of Processors
  = $T_p \times p$
- Cost is also called Processor Time Product
- COST OPTIMAL (or WORK OPTIMAL) Parallel Algorithm
  - Total work done = Serial Complexity of the problem

Blocked Matrix Multiplication

- Serial Algorithm - $O(n^3)$

- Parallel Algorithm:
- Number of steps n/b
- Work done by each processor in each step: $O(b)$

Total Number of Processors

- Cost of the algorithm: $O\left(n^2 \times \left[\frac{n}{b} \times b\right]\right) = O(n^2 \times n) = O(n^3)$

Work done in each processor

- Cost of the algorithm ~ serial complexity. So work optimal.

Sum of Elements in an Array

- Serial Algorithm: $O(p)$

- Parallel Algorithm:
- Number of steps - $O(\log_2 p)$
- Number of processors in each step - $O(p)$
  - Note: We can also consider only active processors to come up with a different analysis.

- Cost of the Algorithm - $O(p \log_2 p)$

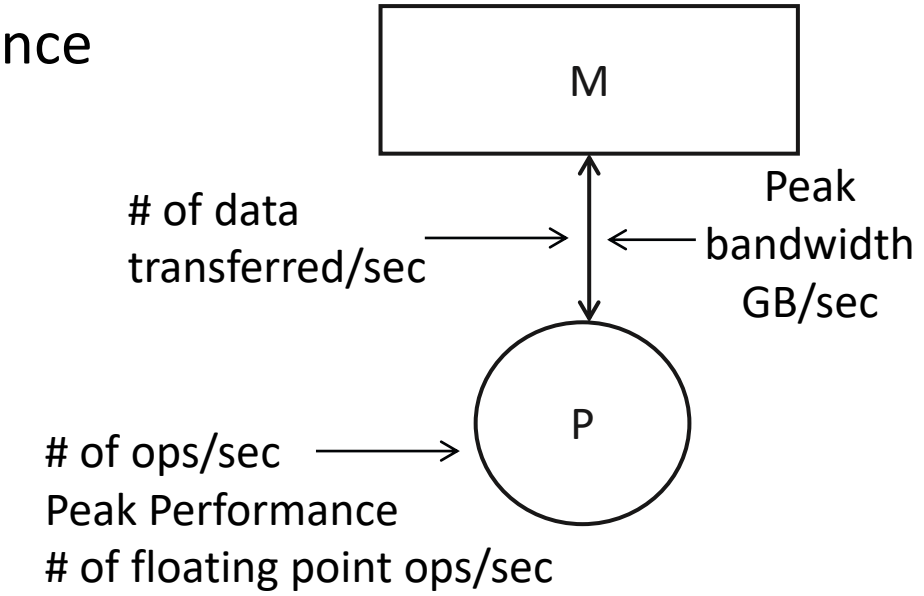
- Cost of the algorithm !~ serial complexity. So not work optimal (assuming GPU model). It is work optimal if we only consider active processor in the analysis.

Roofline Model

- **In Practice:** Processor Performance ≫ Memory Performance

- Processor may idle waiting for data

$$\text{Machine Balance} = \frac{Peak\ Performance\ (Flops/sec)}{Peak\ Bandwidth\ (GB/sec)}$$



The Roofline Model
Samuel Williams, Lawrence Berkeley National Lab
https://escholarship.org/content/qt0qs3s709/qt0qs3s709.pdf

Roofline Model

- Arithmetic Intensity (AI) = For every fetched data (from memory) how many ops make use of them

$$= \frac{\#\ of\ ops\ performed}{Amount\ of\ data\ moved\ from\ external\ memory}$$

- Algorithm Dependent

- Example

  AI of vector inner product $= \frac{2n}{2n} = O(1)$

  AI of Dense matrix multiplication $= \frac{2n^3}{2n^2} = O(n)$

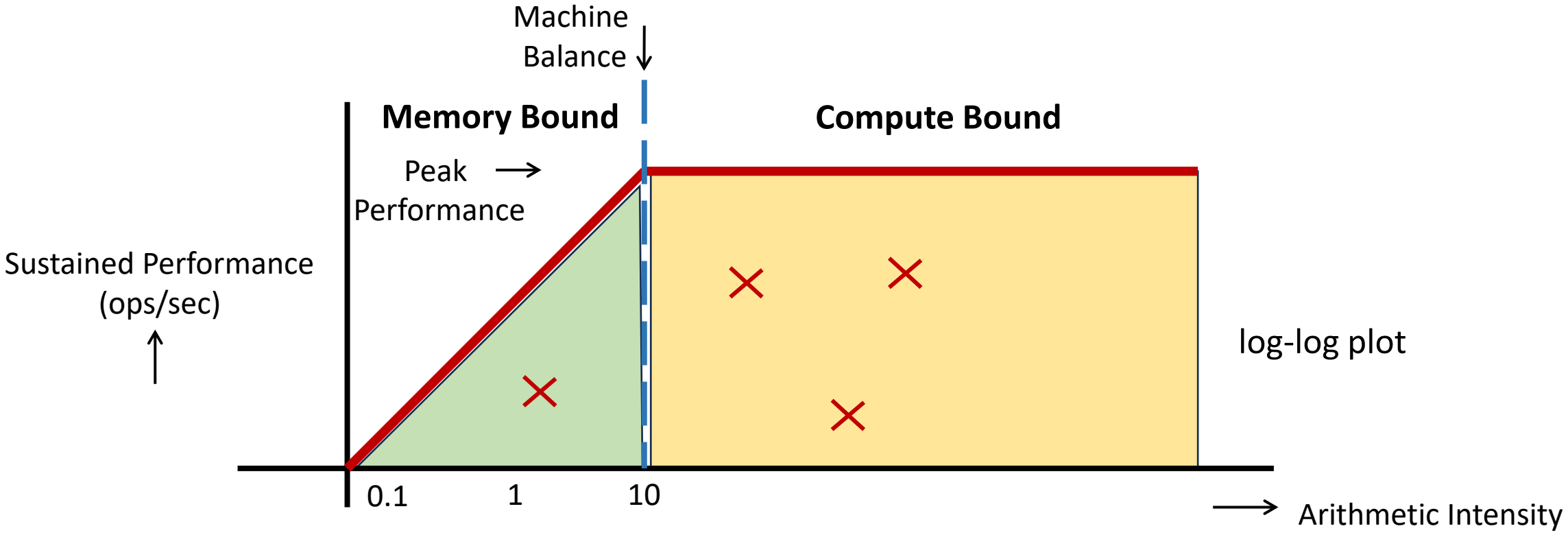
Roofline Model

- Understand system bottlenecks – Computation vs Memory
- High level model using few parameters (ignores ISA etc.)
- Can be used to understand and evaluate an application performance on a target
- Can be used to make architectural decisions for custom accelerators

The Roofline Model
Samuel Williams, Lawrence Berkeley National Lab
https://escholarship.org/content/qt0qs3s709/qt0qs3s709.pdf

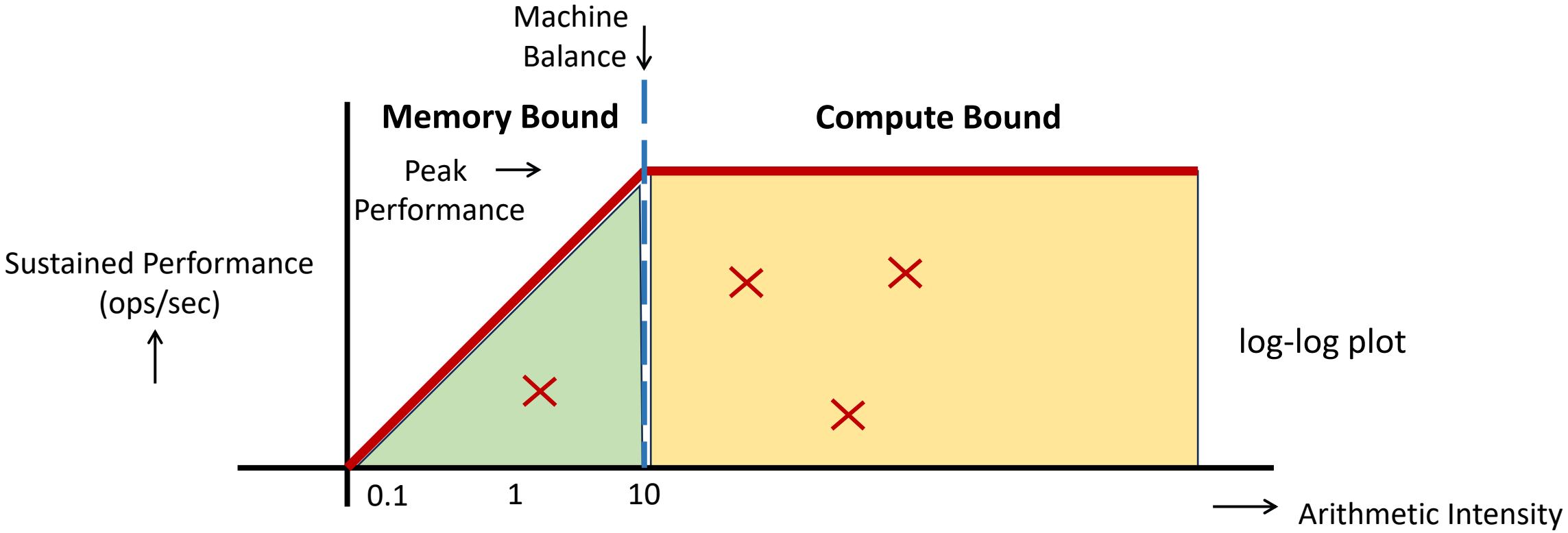
Compute Bound vs Memory Bound

Roofline Model (for a given platform)



Compute Bound vs Memory Bound

Roofline Model (for a given platform)



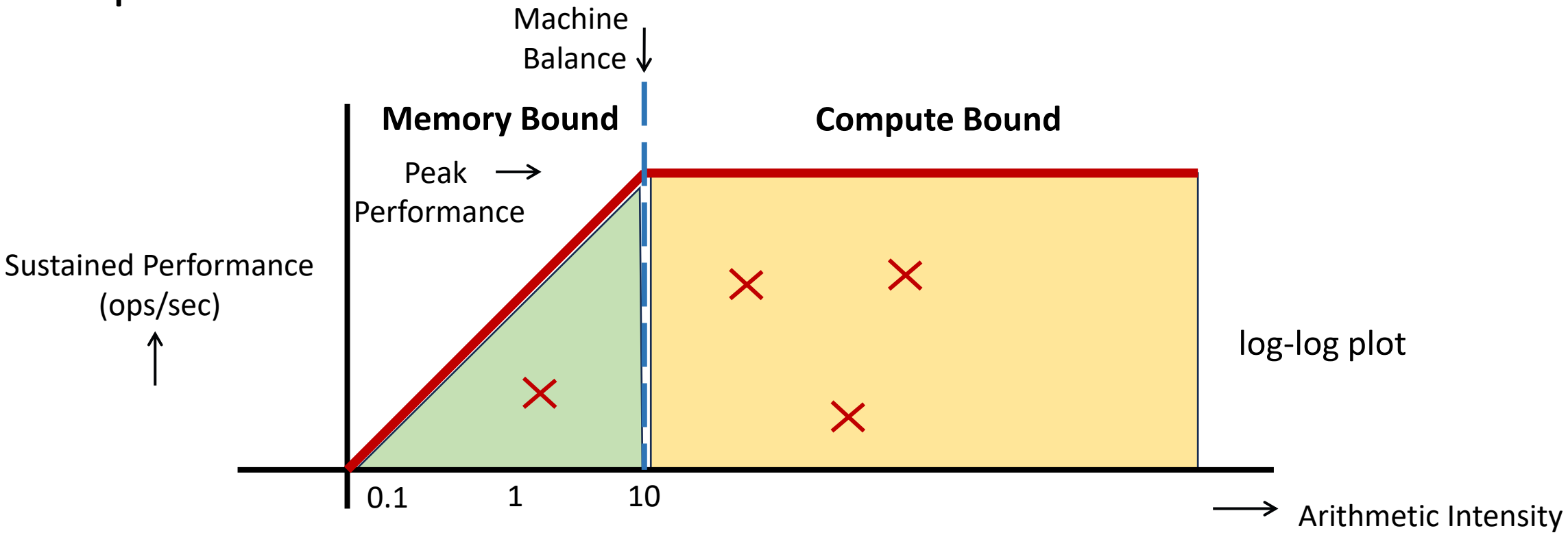
- Bold Red line denotes the maximum achievable performance for a given arithmetic intensity
- Memory bound region: maximum achievable performance bounded by memory bandwidth
- Compute bound region: maximum achievable performance bounded by compute capability

Compute Bound vs Memory Bound Roofline Model (for a given platform)

- Compute Bound: Maximum Performance is limited by available compute resources
  - if we add more compute resources, the performance may improve (for a given bandwidth)
  - Useful in designing accelerators. Not a focus of this class

- Memory Bound: Maximum Performance is limited by memory bandwidth
  - Reorder computations to more efficiently utilize bandwidth
  - In other words, improve arithmetic intensity to move right on x axis, which also improves y axis

Roofline Model – What about actual performance?



- For a fixed AI: improving performance (going up) requires better parallelism – Data and Task Parallel Techniques
- Once the Red bold line is reached
  - Green Region: Write better algorithm to improve arithmetic intensity
  - Yellow Region: No algorithmic solution, need to add more computation power

WA 1

- You should be able to answer all questions except the sorting one. We will discuss parallel sort in the next class

Next Class

- 9/16 Lecture 7 – Parallel Sorting; Task Parallelism

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Thank You

- Questions?

- Email: sanmukh.kuppannagari@case.edu