CASE SCHOOL
OF ENGINEERING
CASE WESTERN RESERVE
UNIVERSITY

CSDS 451: Designing High Performant Systems for AI

Lecture 2

8/28/2025

Sanmukh Kuppannagari

sanmukh.kuppannagari@case.edu

https://sanmukh.research.st/

Case Western Reserve University

CASE SCHOOL
OF ENGINEERING
CASE WESTERN RESERVE
UNIVERSITY

Outline

- Why Performance Modeling?
- Processor Memory Architectures
  - Basic Execution Model
  - Memory System Modeling

Outline

- Why Performance Modeling?
- Processor Memory Architectures
  - Basic Execution Model
  - Memory System Modeling

Performance Modeling

- Identifying important parameters that determine the performance of an **algorithm** mapped to the **platform**
  - Algorithm – Parallel algorithm that we have designed
  - Platform – The target hardware on which we wish to run the parallel algorithm

- Identifying equations that govern the performance

Why Performance Modeling?

- Serial Code
- For $i = 1$ to N
  - $j = 2^i$
  - If $j \geq M$ break
  - Else: Do work()
- What are the important parameters?
- What are the equations governing the performance of the algorithm?

Why Performance Modeling?

- Serial Code
- For $i = 1$ to N
  - $j = 2^i$
  - If $j \geq M$ break
  - Else: Do work()

- What are the important parameters?

M,N

- What are the equations governing the performance of the algorithm?

Min{log M,N}

Why Performance Modeling?

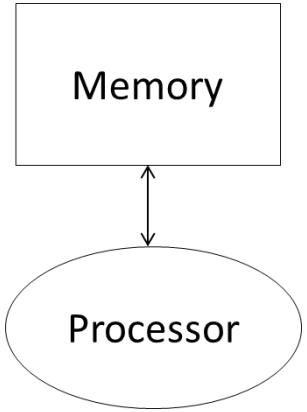
- Complexity Analysis of Serial Algorithms
  - Bubble sort (n^3) vs Merge sort (n log n)
  - Parameter that determine performance: input size n
  - Equations governing performance – complexity of the algorithm
- Provide understanding of bottlenecks and what to optimize
- Do not provide actual implementation estimates
  - Quick sort vs Merge sort ???

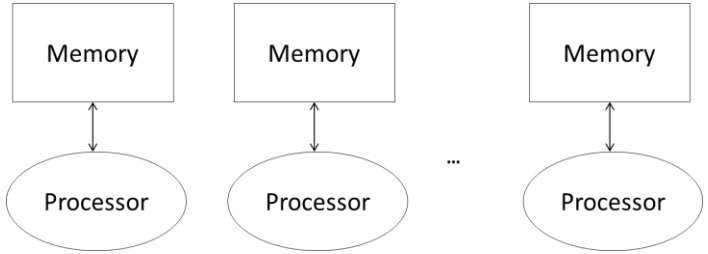
Parallel Algorithm Performance Modeling

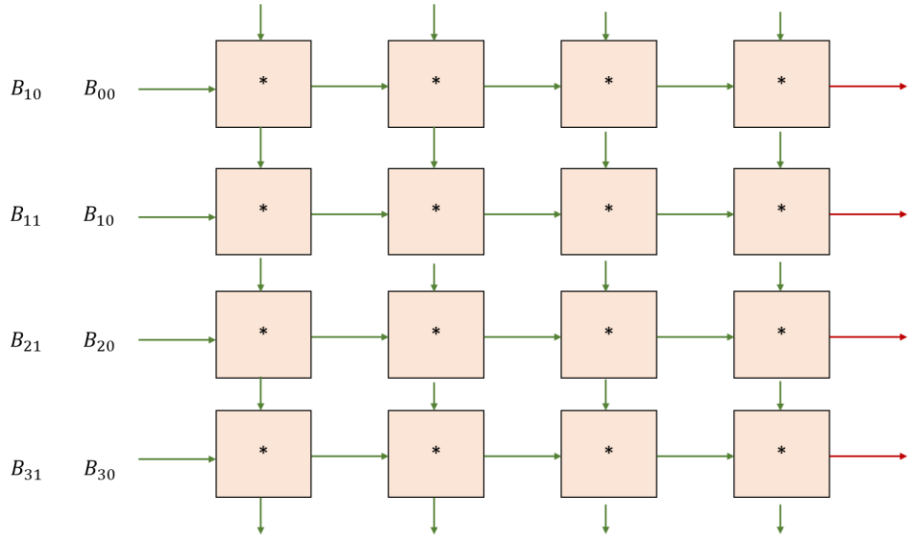
- Performance is platform dependent
- Different platforms -> different parameters
- Different platforms -> different equations governing performance

Targeted Platforms

- Processor-Memory Architecture
  - CPUs, GPUs
- Systolic Array Architecture
  - TPUs, Tensor Cores
- Clusters
  - HPC







Why Performance Modeling?

- Most importantly, to get a mental model of how the algorithm you are designing runs on the hardware, without going into specifics of the hardware.
- Parallel programs can be notoriously hard to reason about
- Simple performance models, even if not accurate, can help us get clarity on execution and aid in debugging

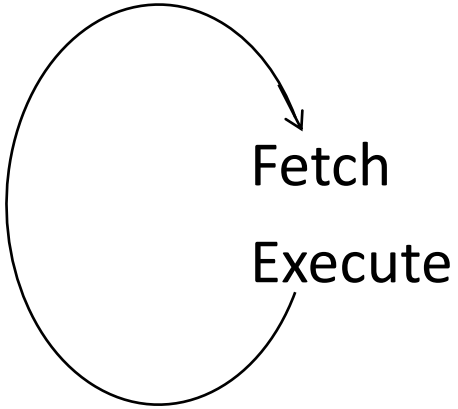
Outline

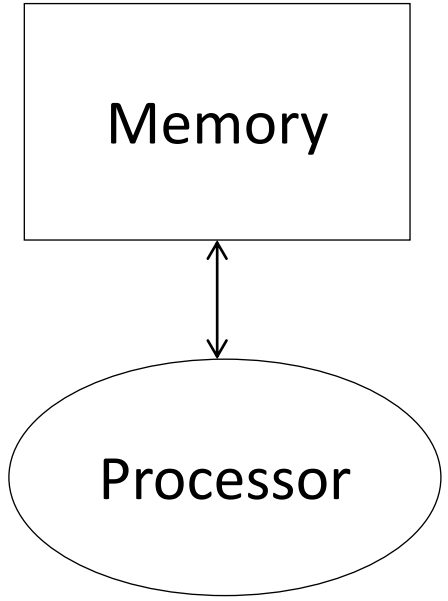
- Why Performance Modeling?
- Processor Memory Architectures
  - Basic Execution Model
  - Memory System Modeling

Basic Execution Model

Fetch-Execute

A simple view





Parameters:

Memory access = 1 cycle for any location
Execution = 1 cycle

Basic Execution Model
Calculating Algorithm Performance

- Consider the following computation
- C[i] = $\sum_k A[i][k] * B[k]$

- i $-$ 0 to M
- k $-$ 0 to N

- What is the above operation?

Basic Execution Model
Calculating Algorithm Performance

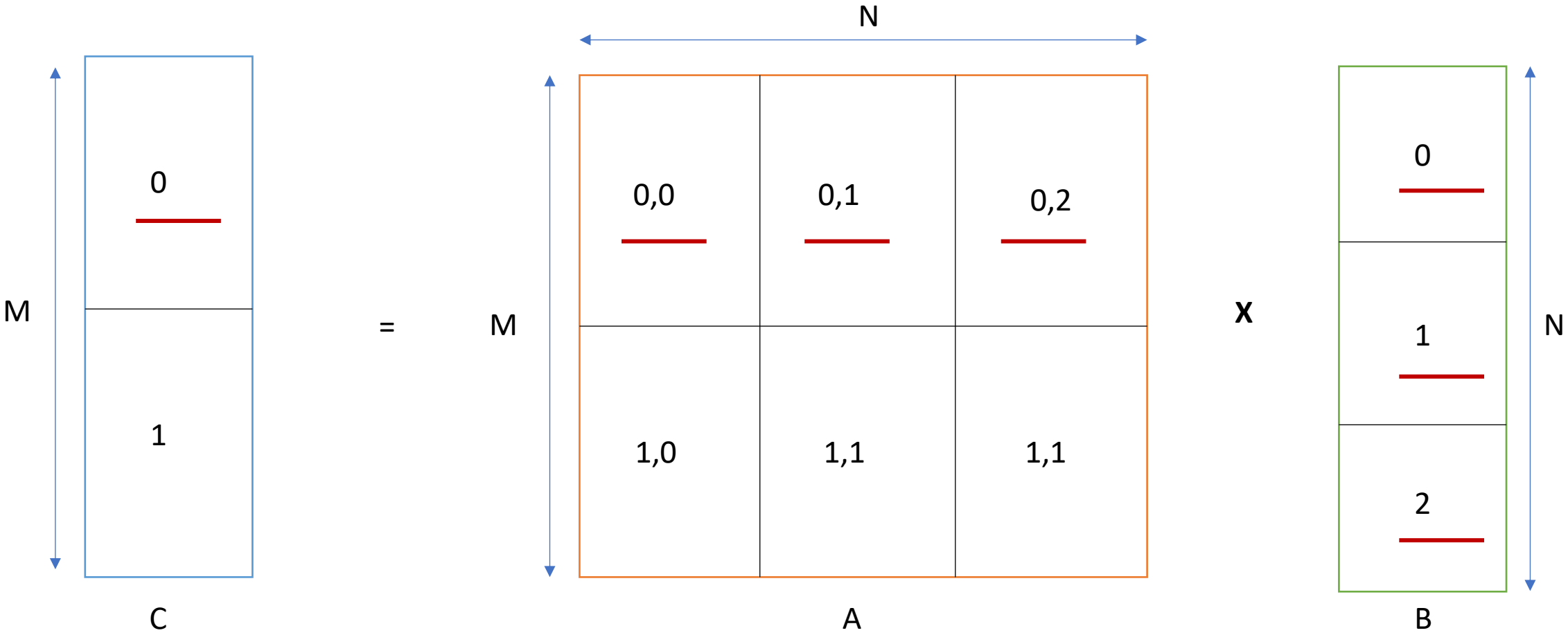
- Consider the following computation
- C[i] = $\sum_k A[i][k] * B[k]$

- i $-$ 0 to M
- k $-$ 0 to N

- What is the above operation? Matrix-Vector Multiplication

Matrix Vector Multiplication

$$C[i] = \sum_k A[i][k] * B[k]$$



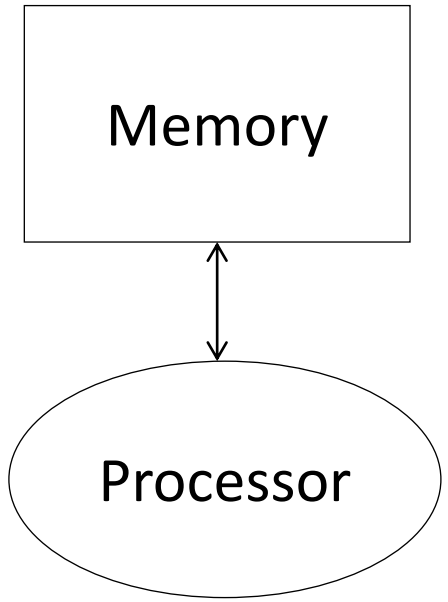
Basic Execution Model
Calculating Algorithm Performance

- C[i] = $\sum_k A[i][k] * B[k]$

Decompose the equation into two types of instructions

Fetch: Read from memory or Write to memory

Execute: Arithmetic/Logical Operation



Basic Execution Model
Calculating Algorithm Performance

- C[i] = $\sum_k A[i][k] * B[k]$

RA: Read A[i][k]

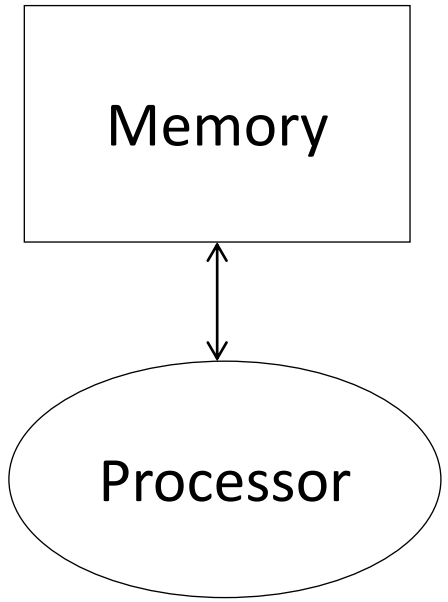
RB: Read B[k]

M: Res <- Mult A[i][k]*B[k]

RC: Read C[i]

A: Add C[i] + Res

S: Store C[i]



Basic Execution Model
Calculating Algorithm Performance

- C[i] = $\sum_k A[i][k] * B[k]$

RA: Read A[i][k]

RB: Read B[k]

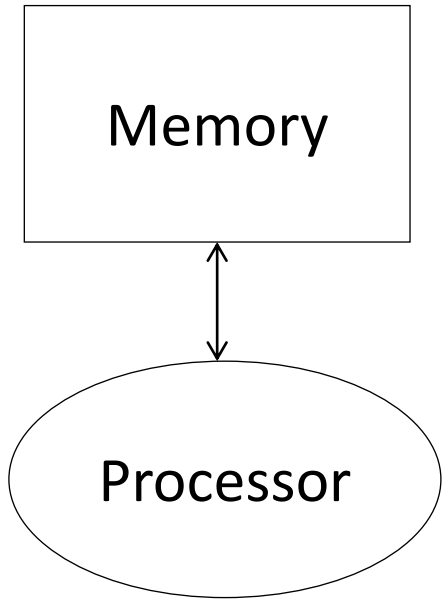
M: Res <- Mult A[i][k]*B[k]

RC: Read C[i]

A: Add C[i] + Res

S: Store C[i]

Assume we are storing these in a local memory (such as registers) in the processor



Basic Execution Model

- Assumptions
- No limit on the local memory
  - Putting a limit may require us to use a different set of instructions
- Addition and Multiplications are each single cycle operations
  - Not necessarily true in modern architectures

Basic Execution Model
Calculating Algorithm Performance

- C[i] = $\sum_k A[i][k] * B[k]$

RA: Read A[i][k]

RB: Read B[k]

M: Res <- Mult A[i][k]*B[k]

RC: Read C[i]

A: Add C[i] + Res

S: Store C[i]

6 cycles per entry of A

Total Time for the entire matrix = ??

Basic Execution Model
Calculating Algorithm Performance

- C[i] = $\sum_k A[i][k] * B[k]$

RA: Read A[i][k]

RB: Read B[k]

M: Res <- Mult A[i][k]*B[k]

RC: Read C[i]

A: Add C[i] + Res

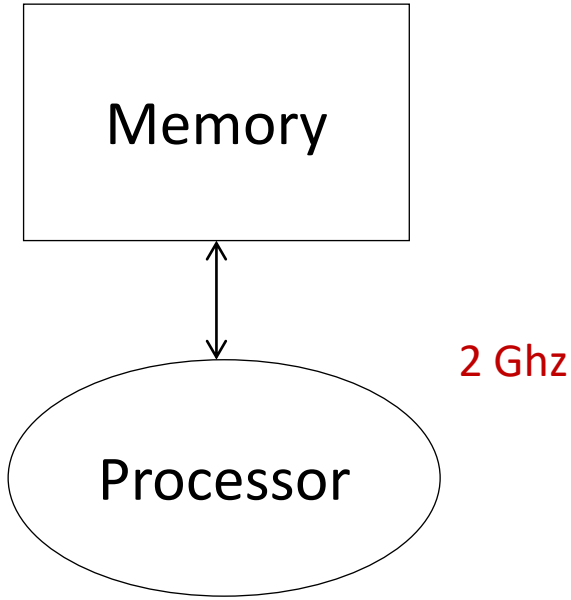
S: Store C[i]

6 cycles per entry of A

Total Time for the entire matrix = 6*M*N cycles

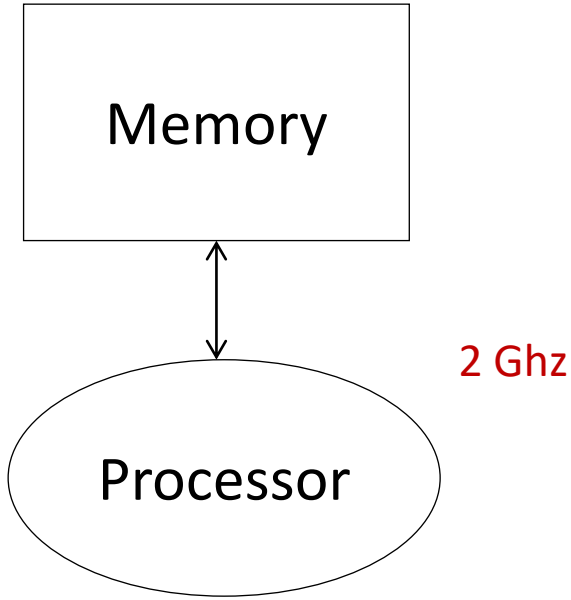
Basic Execution Model

- Assume Processor is running at 2 GHz
- Total time (in seconds) to execute Matrix-Vector Multiplication as per Basic Execution Model
- ???



Basic Execution Model

- Assume Processor is running at 2 GHz
- Total time to execute Matrix-Vector Multiplication as per Basic Execution Model
- $3MN$ seconds



Basic Execution Model: Limitations

- Does not account for memory transfer latencies
- Does not account for differences in the number of cycles in various arithmetic operations
  - Multiplication usually requires more cycles than addition

Basic Execution Model: Uses

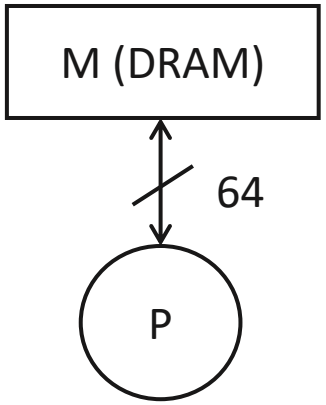
- It will be a part of more complicated models that we will study next
- With modern pipelined architectures, 1 cycle per execute operation is not a bad assumption

Outline

- Why Performance Modeling?
- Processor Memory Architectures
  - Basic Execution Model
  - Memory System Modeling

Memory System

- System performance (execution time) depends on the “rate” at which data can be accessed by the program



Updated Execution Model

Latency: Time to receive the first word since the fetch instruction

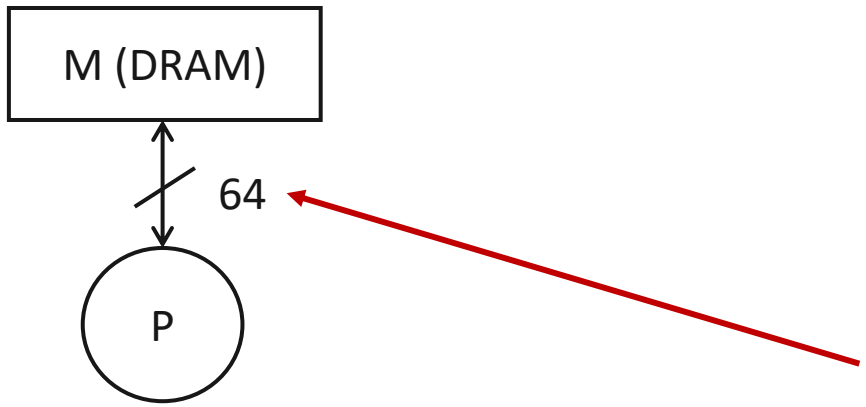
Bandwidth: Rate of data transfer = bitwidth of the interconnect x frequency

Execute – 1 cycle

Fetch – Latency + Data size/bandwidth cycles

Memory System

- System performance (execution time) depends on the “rate” at which data can be accessed by the program



Updated Execution Model

Latency: Time to receive the first word since the fetch instruction

Bandwidth: Rate of data transfer = bitwidth of the interconnect x frequency

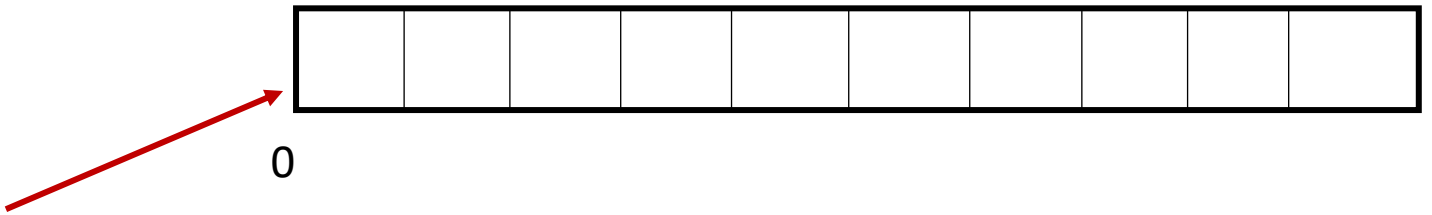
Execute – 1 cycle

Fetch – Latency + Data size/bandwidth cycles

Memory System

- **Latency:** Every fetch request requires some setup operation in memory before it can actually start transferring the data

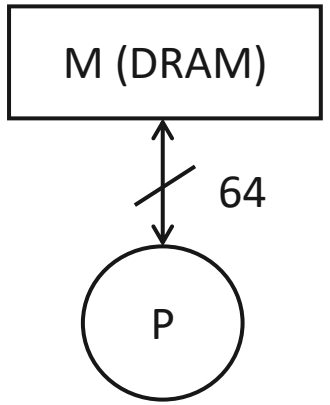
- **Bandwidth:** Once the data starts transferring, it can continuously transfer without any interruptions depending upon the speed of the interconnect, as long as it is contiguous



Fetch request: Given me 10 data elements starting at location 0

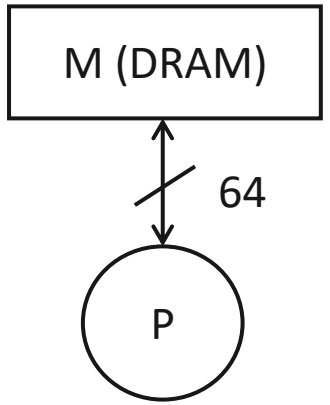
Memory System

- Read A[0-31]
- Latency: 10 cycles
- Bandwidth: 8 data elements per cycle (assuming 8 bits/data element)
- Number of cycles to finish the read operation?



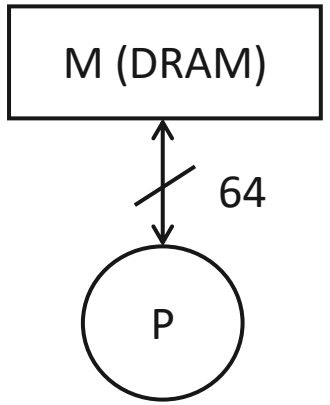
Memory System

- Read A[0-31]
- Latency: 10 cycles
- Bandwidth: 8 data elements per cycle (assuming 8 bits/data element)
- Number of cycles to finish the read operation? 10 + 4 = 14 cycles



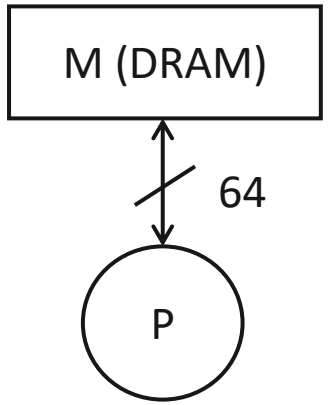
Memory System

- Read A[0-31], A[32-63], A[64-95]
- Latency: 10 cycles
- Bandwidth: 8 data elements per cycle (assuming 8 bits/data element)
- Number of cycles to finish the read operation?



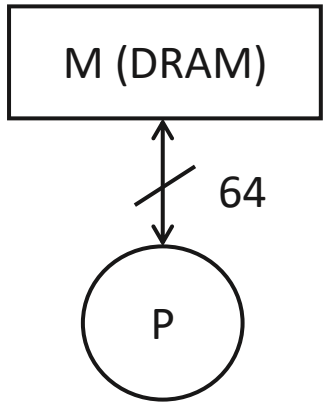
Memory System

- Read A[0-31], A[32-63], A[64-95]
- Latency: 10 cycles
- Bandwidth: 8 data elements per cycle (assuming 8 bits/data element)
- Number of cycles to finish the read operation? 10 + 4 + 10 + 4 + 10 + 4 = 42 cycles



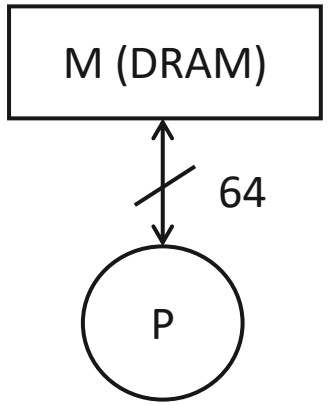
Memory System

- Read A[0-95]
- Latency: 10 cycles
- Bandwidth: 8 data elements per cycle (assuming 8 bits/data element)
- Number of cycles to finish the read operation?



Memory System

- Read A[0-95]
- Latency: 10 cycles
- Bandwidth: 8 data elements per cycle (assuming 8 bits/data element)
- Number of cycles to finish the read operation? 10 + 12 = 22 cycles

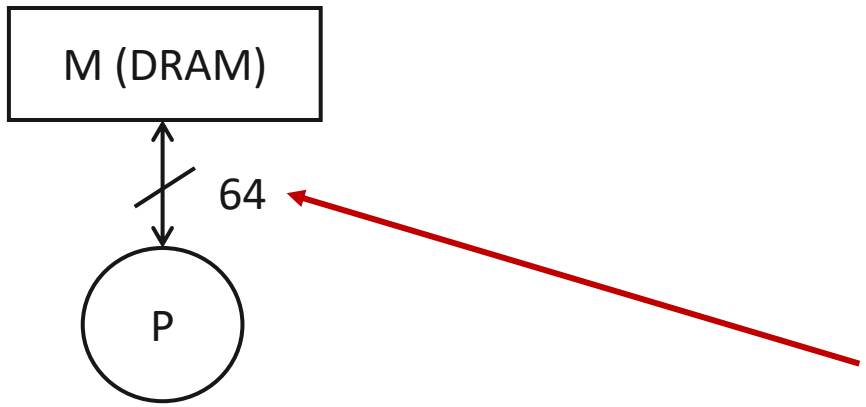


Memory System

- By fetching the entire 96 words in a single go, we saved 20 cycles
- Prefetching:
  - If you know what data your algorithm needs to access beforehand (not as easy as it sounds)
  - Prefetch it to save on latency
- (Note: above 20 cycles is based on a simple model, in reality, it is difficult to exactly calculate the savings, but we do see substantial savings)

Memory System

- System performance (execution time) depends on the “rate” at which data can be accessed by the program



Updated Execution Model

Latency: Time to receive the first word since the fetch instruction

Bandwidth: Rate of data transfer = bitwidth of the interconnect x frequency

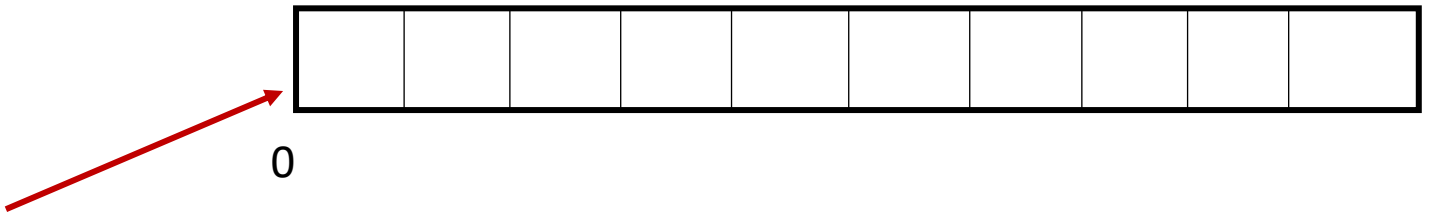
Execute – 1 cycle

Fetch – Latency + Data size/bandwidth cycles

Memory System

- **Latency:** Every fetch request requires some setup operation in memory before it can actually start transferring the data

- **Bandwidth:** Once the data starts transferring, it can continuously transfer without any interruptions depending upon the speed of the interconnect, as long as it is contiguous



Fetch request: Given me 10 data elements starting at location 0

Memory System: Modeling

- Performance will be measured in terms of “System Performance Metric”
- Number of “useful computations” performed per second
- Useful computation – arithmetic computations such as Add, Mult, etc.
  - Does not include memory fetching time in calculations

System Performance Metric

- FLOPs – Floating Point Operations Per Second
- Number of useful “floating point” operations in a second
- GFLOPs – Giga (1024) Floating Point Operations Per Second
- Note: Unless explicitly specified, we will use 64 bit floating point as data type for performance modeling in this class.
- We will call each data element as a **word**

System Performance Metric

- 1024 read operations
- For every two read operations, we perform one add operation.
- All the operations mentioned above take a total of 10 seconds.
- What is the system performance?

System Performance Metric

- 1024 read operations
- For every two read operations, we perform one add operation.
- All the operations mentioned above take a total of 10 seconds.
- What is the system performance? 512 Floating point operations in 10 seconds

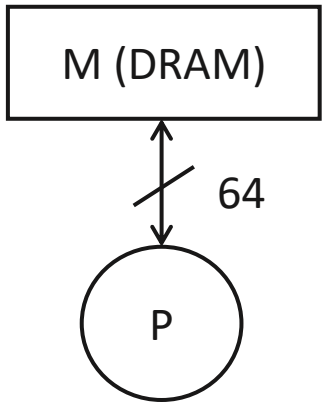
System Performance Metric

- 1024 read operations
- For every two read operations, we perform one add operation.
- All the operations mentioned above take a total of 10 seconds.
- What is the system performance? 51.2 FLOPs

System Performance Metric

- 1024 read operations
- For every two read operations, we perform one add operation.
- All the operations mentioned above take a total of 10 seconds.
- What is the system performance? 0.05 GFLOPs

Memory System Modeling



Processor

2 Ghz (0.5 ns per cycle)

Memory

Memory Latency = 10 ns (20 cycles)
Bandwidth = 64 bits at 1 GHz

(64 Gbits/sec or 8 GB/sec) or

(64 bits in every 2 processor cycles)

Memory System Modeling

- We will consider three metrics:
  - Peak Performance
  - Sustained Performance (best case)
  - Sustained Performance (worst case)

- Actual obtained performance on the hardware should be somewhere in between Sustained Performance (worst case) and Peak Performance
  - If not, there was some error in modeling or the implementation of the algorithm
  - It can be higher or lower than the Sustained Performance (best case), this information governs what optimizations to use

Memory System Modeling: Peak Performance

- Captures the Raw Computer power of the processor (memory not considered)

- Peak Performance = Clock Rate X Num ops per cycle

= 2 (Ghz) x 1 (ops/cycle)

= 2 Giga Ops per Second

- If Floating point ops = 2 GFLOPs (Giga Floating Point Operations per second)

Memory System Modeling: Peak Performance

- Captures the Raw Computer power of the processor (memory not considered)

- Peak Performance = Clock Rate X Num ops per cycle

For multiple processors/GPUs, this number will change

= 2 (Ghz) x 1 (ops/cycle)

= 2 Giga Ops per Second

- If Floating point ops = 2 GFLOPs (Giga Floating Point Operations per second)

Memory System Modeling: Sustained Performance (Worst Case)

- **Each fetch takes: Latency + (data size = 1)/bandwidth ≈ latency number of cycles**

- C[i] = $\sum_k A[i][k] * B[k]$

RA: Read A[i][k] ← Latency

RB: Read B[k] ← Latency

M: Res <- Mult A[i][k]*B[k] ← 1

RC: Read C[i] ← Latency

A: Add C[i] + Res ← 1

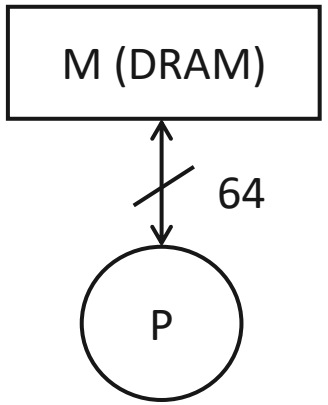
S: Store C[i] ← Latency

Memory System Modeling: Sustained Performance (Worst Case)

- **Each fetch takes: Latency + (data size = 1)/bandwidth ≈ latency number of cycles**

- Example: When Prefetching is not used
  - In reality, the data size will be cache line length, but we are simplifying the model here
  - Also, the sophisticated caching mechanisms in modern architectures ensure the performance is not this bad

Memory System Modeling: Sustained Performance (Worst Case)



Processor

2 Ghz (0.5 ns per cycle)

Memory

Memory Latency = 10 ns (20 cycles)
Bandwidth = 64 bits at 1 GHz

(64 Gbits/sec or 8 GB/sec) or

(64 bits in every 2 processor cycles)

Can you calculate the Sustained Performance (worst case) in GFLOPs for the entire matrix?

Memory System Modeling: Sustained Performance (Worst Case)

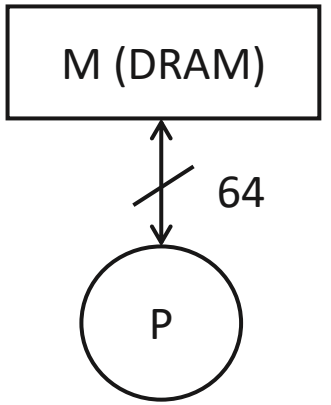
- Fetch = ??
- Execute = ??
- Total time (per each loop over i) = ??
- Number of Operations (per each loop over i) = ??
- Performance: ?? GFLOPs

Memory System Modeling: Sustained Performance (Worst Case)

- Fetch = 10 ns
- Execute = 0.5 ns
- Total time (per each loop over i) = $4 \times 10 + 2 \times 0.5 = 41$ ns
- Number of Operations (per each loop over i) = 2
- Performance: 2/41 ns = 0.048 GFLOPs

Memory System Modeling: Sustained Performance (Best Case)

- In Sustained Performance (worst case), we were fetching one word at a time.
  - Separate fetch instructions
  - Each instruction incurring latency



- Sustained Performance (best case): Assume data is contiguously stored
  - Single fetch can be performed
  - Assume latency is 0

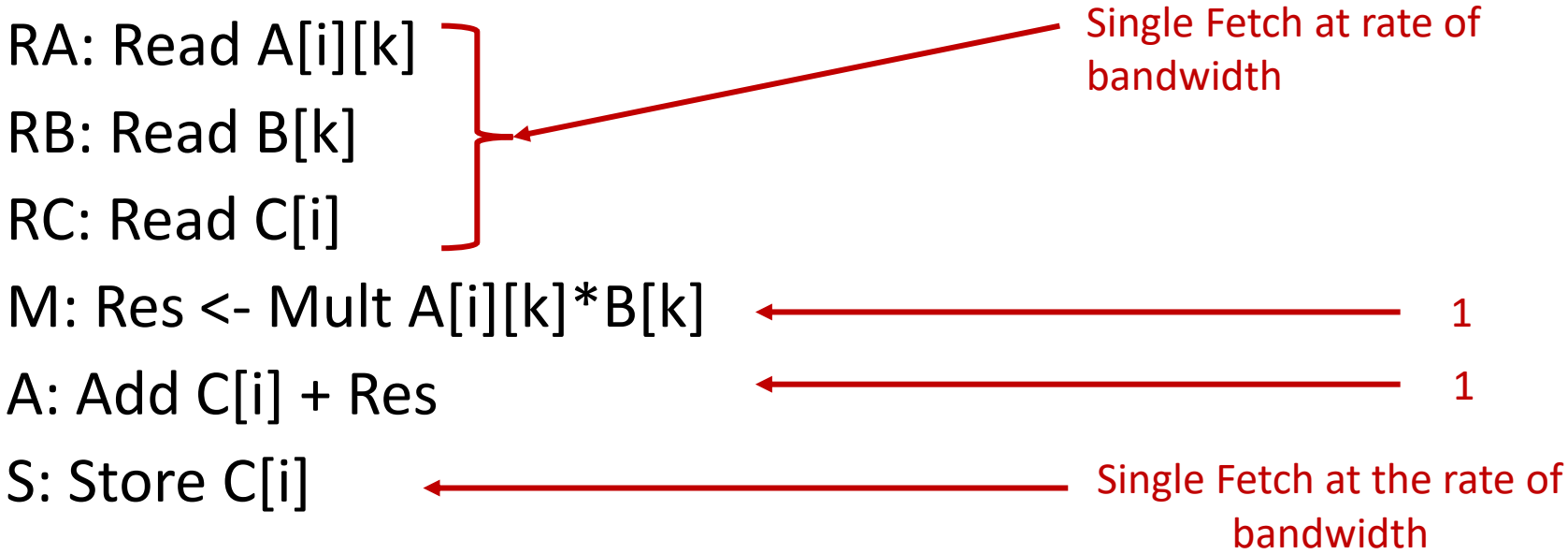
Memory System Modeling: Sustained Performance (Best Case)

- How can the latency be 0?
- Data prefetching: Processor initiates data transfer request way before the data is actually needed
- Predicting data prefetching parameters using ML is a hot research topic: https://scholar.google.com/scholar?as_ylo=2018&q=data+prefetching+machine+learning&hl=en&as_sdt=0,36
- We will not cover in this class

Memory System Modeling: Sustained Performance (Best Case)

- **Each fetch takes: Latency (=0) + data size/bandwidth ≈ data size/bandwidth**

- C[i] = $\sum_k A[i][k] * B[k]$



Memory System Modeling: Sustained Performance (Best Case)

- **Each fetch takes: Latency (=0) + data size/bandwidth ≈ data size/bandwidth**

- C[i] = $\sum_k A[i][k] * B[k]$

RA: Read A[i][k] ← Time in ns??

RB: Read B[k]

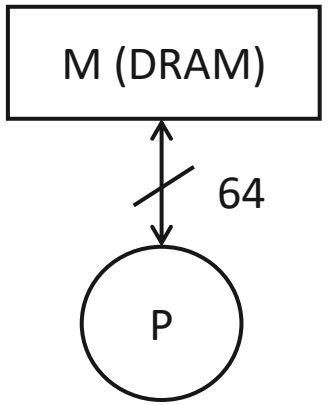
RC: Read C[i]

M: Res <- Mult A[i][k]*B[k] ← Time in ns??

A: Add C[i] + Res ← Time in ns??

S: Store C[i] ← Time in ns??

Memory System Modeling: Sustained Performance (Best Case)



Processor

2 Ghz (0.5 ns per cycle)

Memory

Memory Latency = 10 ns (20 cycles)
Bandwidth = 64 bits at 1 GHz

(64 Gbits/sec or 8 GB/sec) or

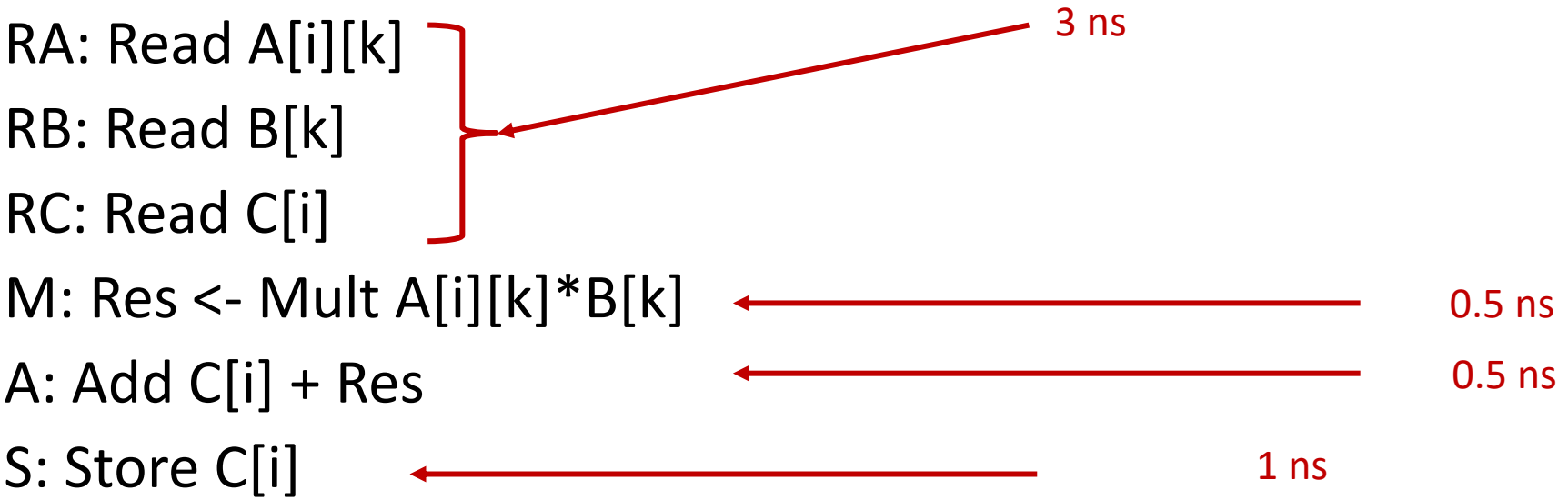
(64 bits in every 2 processor cycles)

Can you calculate the Sustained Performance (best case) in GFLOPs for the entire matrix?

Memory System Modeling: Sustained Performance (Worst Case)

- **Each fetch takes: Latency (=0) + data size/bandwidth ≈ data size/bandwidth**

- C[i] = $\sum_k A[i][k] * B[k]$



Memory System Modeling: Sustained Performance (Best Case)

- Total time (per loop over i) = 5 ns
- Number of Operations (per loop over i) = 2
- Performance: 2/5 ns = 0.4 GFLOPs
- Notes
  - Again we assumed no limit on local memory of processor
  - Also, note how we reordered the read instructions to maximize bandwidth use

System Performance Metrics

Actual System Performance

Latency Hiding Mechanisms

Efficient Bandwidth Utilization

0.048 Gflops/s

0.4 Gflops/s

2 Gflops/s

Sustained Worst Case
- Latency dominant

Sustained Best Case
- Latency Hidden
- Memory bandwidth dominant

Peak Performance
- Compute Only

Memory System: Conclusion

- A simple model to perform “pen and paper” calculations to analyze an algorithm
- Compare various algorithms in a similar spirit as complexity analysis
- Get a mental model of the algorithm execution to potentially find better optimizations

Memory System: Conclusion

- Ungraded HW assignment 1: Can you create a simple code to perform these calculations? (May help you in your assignments later)

Next Class

- 9/2 Lecture 3 – Processor-Memory Architecture II
  - Memory Optimizations,
  - Processor Memory Architectures with Cache
  - Blocked Matrix Multiplication

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Thank You

- Questions?

- Email: sanmukh.kuppannagari@case.edu