Computer Architecture
Final Project: Matrix Chain Multiplication

Speaker：Billy

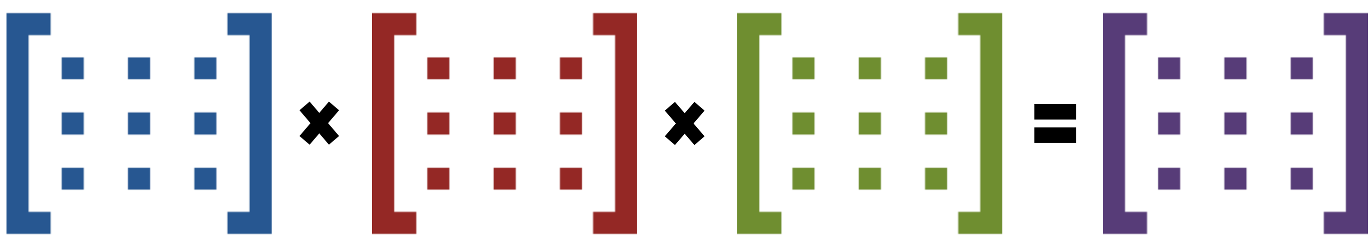
Advisor：Prof. An-Yeu Wu

Date：2025/4/28

Overview

- Given a sequence of matrices, implement **assembly code** to compute the matrix chain multiplication.

- Your implementation will be scored based on its performance, which is determined by **the number of cycles and cache size**.

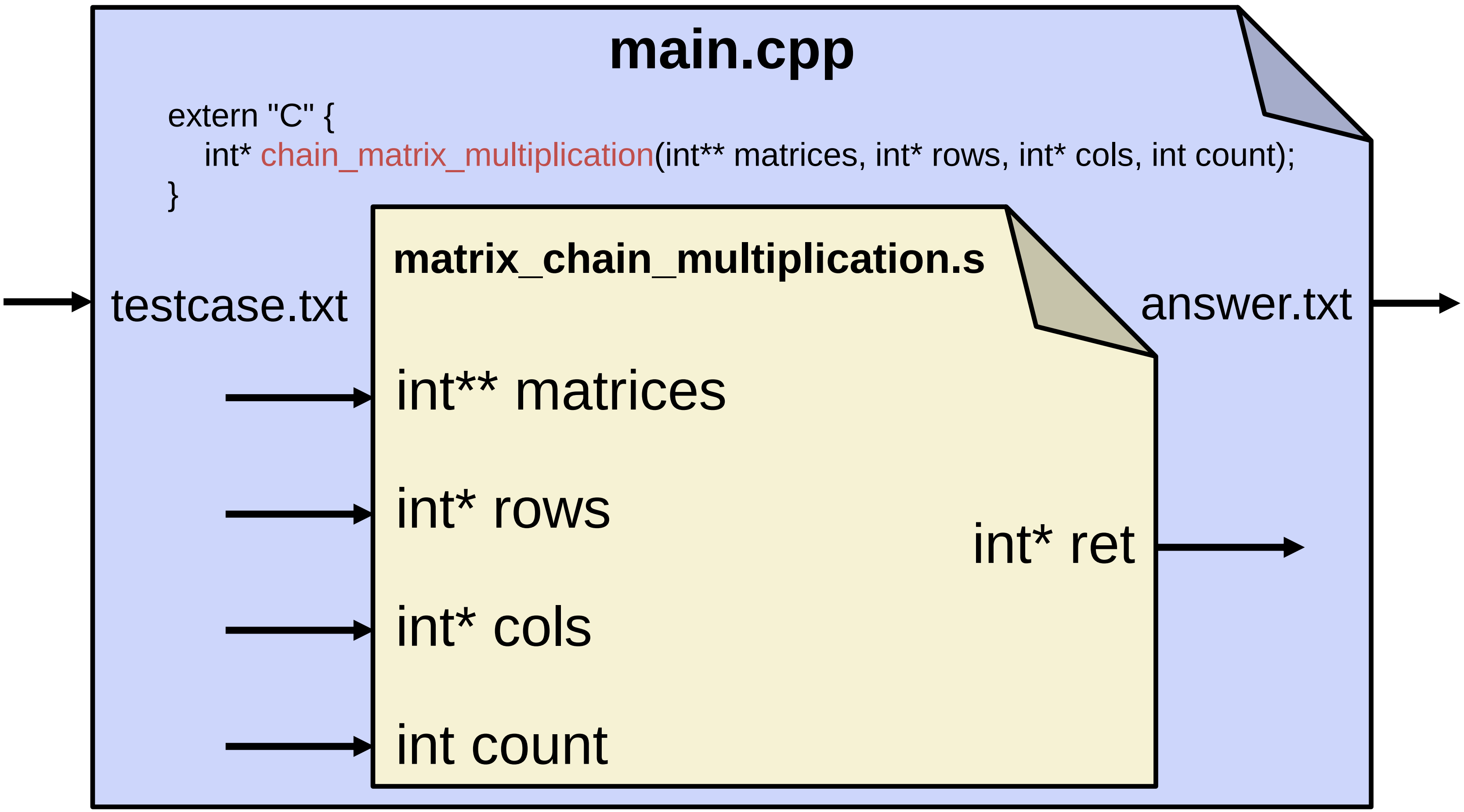


Project Goal

- Accelerate the function to achieve better performance
  - Increase hit rate of cache
  - Use better algorithm (e.g. Dynamic Programming)

- Performance Formula
  - $Time \times \left(\log_2 Size_{L1\_ICache} + \log_2 Size_{L1\_DCache} + \frac{1}{2}\log_2 Size_{L2\_Cache}\right)$
  - Goal: minimize the performance formula

Block Diagram



Input / Output

Type	Port	Reg	I/O	Description
Int **	matrices	`x10 (a0)`	I	An array storing the addresses of all matrices*.
Int *	rows	`x11 (a1)`	I	An array storing the row size of each matrix.
Int *	cols	`x12 (a2)`	I	An array storing the column size of each matrix.
Int	count	`x13 (a3)`	I	An integer representing the number of matrices.
Int *	ret	`x10 (a0)`	O	An address storing the result matrix* after computation.

*All matrices are stored in row-major order.

- In the RISC-V calling convention
  - Function arguments are passed starting from the `a0` register
  - Return value is stored in the `a0` register too

Input

- Example:

$$\begin{bmatrix} 23 & 7 \\ 16 & 11 \end{bmatrix} \times \begin{bmatrix} 2 & 13 & 5 \\ 8 & 26 & 18 \end{bmatrix} \times \begin{bmatrix} 9 & 15 & 17 \\ 31 & 28 & 4 \\ 14 & 27 & 21 \end{bmatrix}$$

2×2 matrix   2×3 matrix   3×3 matrix

Addr	Value
matrices	0x1000

Addr	Value
0x1000	0x2000
0x1004	0x2100
0x1008	0x2200

Addr	Value
0x2000	23
0x2004	7
0x2008	16
0x200c	11

Input

- Example:

$$\begin{bmatrix} 23 & 7 \\ 16 & 11 \end{bmatrix} \times \begin{bmatrix} 2 & 13 & 5 \\ 8 & 26 & 18 \end{bmatrix} \times \begin{bmatrix} 9 & 15 & 17 \\ 31 & 28 & 4 \\ 14 & 27 & 21 \end{bmatrix}$$

2×2 matrix 2×3 matrix 3×3 matrix

Addr	Value
rows	0x3000

Addr	Value
0x3000	2
0x3004	2
0x3008	3

Addr	Value
cols	0x4000

Addr	Value
0x4000	2
0x4004	3
0x4008	3

Addr	Value
count	3

Output

- Example:

$$\begin{bmatrix} 23 & 7 \\ 16 & 11 \end{bmatrix} \times \begin{bmatrix} 2 & 13 & 5 \\ 8 & 26 & 18 \end{bmatrix} \times \begin{bmatrix} 9 & 15 & 17 \\ 31 & 28 & 4 \\ 14 & 27 & 21 \end{bmatrix} = \begin{bmatrix} 19203 & 21505 & 8719 \\ 20286 & 23138 & 9854 \end{bmatrix}$$

result

Addr	Value
ret	0x9000

You need to store this value in `a0` before return

Addr	Value
0x9000	19203
0x9004	21505
0x9008	8719
0x900c	20286
0x9010	23138
0x9014	9854

RISC-V Calling Convention

- Input / output start from `a0`
- Return address is store in `ra`
- `s0` needs to be saved and restored across function call

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5–7	t0–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller

Memory Allocation

- Allocation of memory for storing matrices is needed in this project
  - At least for the result matrix of matrix chain multiplication

- You can allocate memory by `call malloc` as example below:
  - Allocate 10 bytes memory space in this example

```
li a0, 10         # Store required memory size in a0 (function argument)
call malloc       # Function call to allocate memory

sw t0, 0(a0)      # Allocated memory address will be stored in a0
                  # you can than store data in the allocated memory
```

  - Value in register `a0`, `a1`, `a2`, `a3` will be overwritten after this function call

- Release memory space by `call free`

Matrix Multiplication Assembly Example

- Basic idea of matrix multiplication: Three loops

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        for (int k = 0; k < n; k++) {
            C[i][j] += A[i][k] * B[k][j]
        }
    }
}
```

Matrix Multiplication Assembly Example

```
    .data
A:  .word 1, 2, 3, 4                # Matrix A
B:  .word 5, 6, 7, 8                # Matrix B
C:  .space 16                       # Space for Matrix C (2 x 2 = 4 integers = 16 bytes)

    .text
    .globl main
main:
    # load base addresses
    la s0, A                        # s0 = base address of A
    la s1, B                        # s1 = base address of B
    la s2, C                        # s2 = base address of C

    # i initialization
    li t0, 0                        # i = 0 (row index)

outer_loop_i:
    # j initialization

outer_loop_j:
    # k initialization
```

Data definition

Address of matrix data

Loop initializatoin

Matrix Multiplication Assembly Example

```
inner_loop_k:
    li t3, 2                        # matrix size = 2

    # Compute address of A[i][k]
    mul t4, t0, t3                  # t4 = i * matrix size
    add t4, t4, t2                  # t4 = i * matrix size + k
    slli t4, t4, 2                  # offset = (i * matrix size + k) * 4
    add t5, s0, t4                  # (base address of A) + offset
    lw t6, 0(t5)                    # load A[i][k] from memory

    # Compute address of B[k][j]
    # Similar to load A

    # multiplication and accumulation
    mul s7, t6, s6                  # A[i][k] * B[k][j]
    add s3, s3, s7                  # sum += A[i][k] * B[k][j]

    # Update k = k + 1
    addi t2, t2, 1                  # k++
    li s8, 2                        # matrix size = 2
    blt t2, s8, inner_loop_k        # if k < 2, continue inner loop

    # Compute address of C[i][j]
    # Similar to load A and B

    # Update j = j + 1

    # Update i = i + 1

    # End of program
    jr ra
```

Get $A[i][k]$ address and load

Get $B[k][j]$ address and load

Compute $C[i][j]$, iterate k

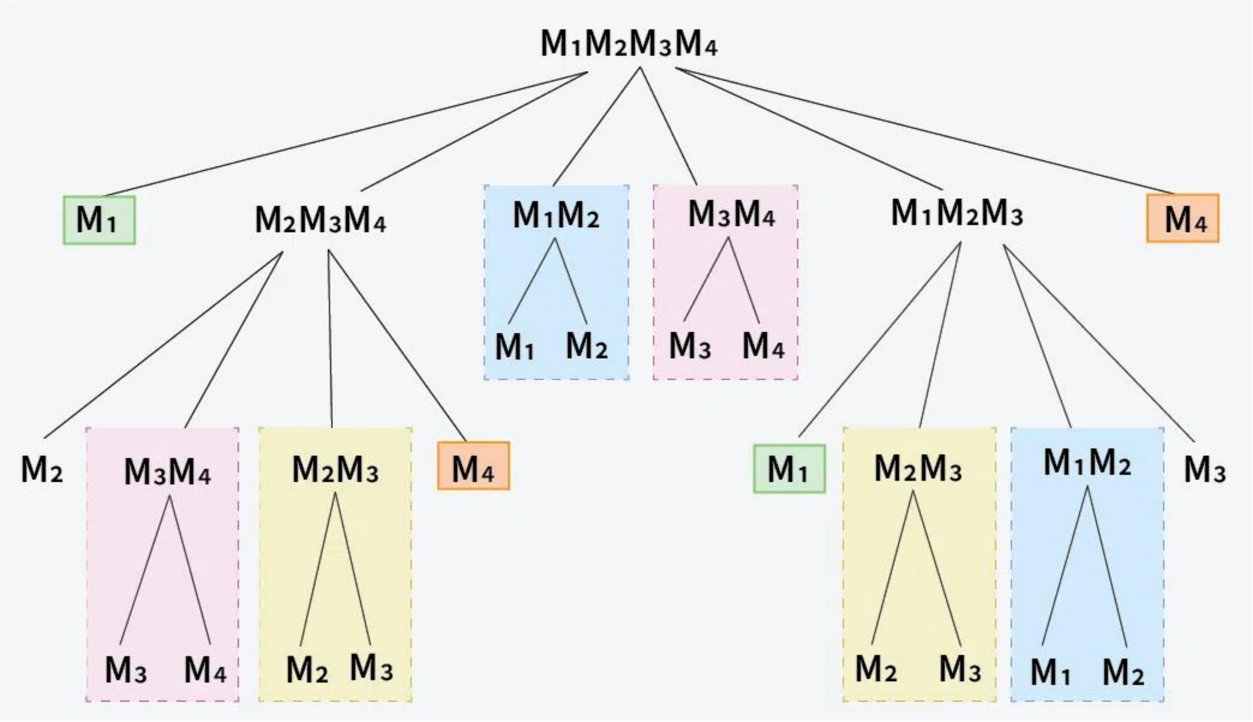
Get $C[i][j]$ address and store

Iterate i, j

End of program

Dynamic Programming

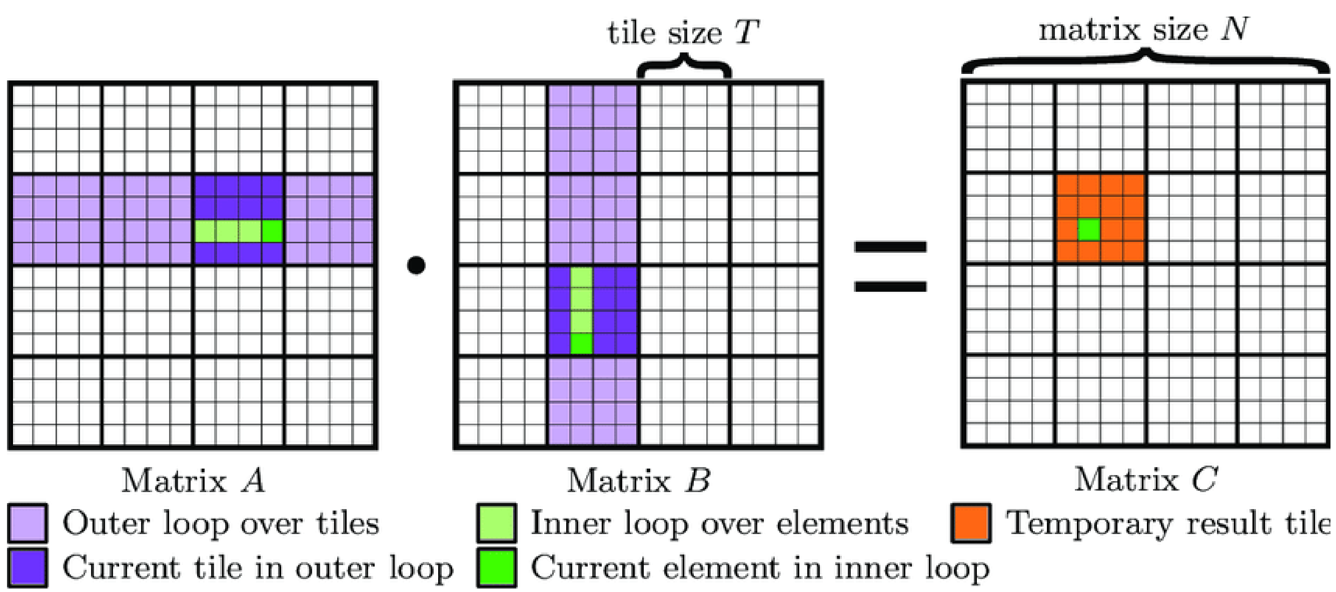
- When multiplying a $k \times m$ matrix with an $m \times n$ matrix
  - $k \times m \times n$ multiply-add operations are required
  - Dynamic programming to find the optimal multiplication order



source: www.geeksforgeeks.org

Tiled Matrix Multiplication

- Tiling matrix into small block that can fit in DCache
  - Outer loop: iterate through each block of matrices
  - Inner loop: iterate through elements inside the block
- Co-optimization of algorithm and cache size



Source: https://www.researchgate.net/figure/Performance-critical-A-B-part-of-the-GEMM-using-a-tiling-strategy-A-thread-iterates_fig1_320499173

Provided Files

Files	Description
Files you should not modify	
main.cpp	Main program
final_config.py	Gem5 configuration file
testbench.py	Python file for checking the correctness
score.py	Python file for calculate performance score
testcase/public/testcase_xx.txt	Public testing data
golden/public/golden_xx.txt	Public golden data
Makefile	Including make commands used in final
Files you need to modify	
matrix_chain_multiplication.s	Main design file
gem5_args.conf	Gem5 argument file, including cache setting

About final_config.py

- Currently doesn’t including config for L2 cache
  - Implementation of L2 cache is part of HW3 (upcoming)

- After HW3 due, final_config.py with L2 cache config will be released
  - Please be aware of the upcoming updates to the final project file
  - Your final result should be simulated by latest version of final_config.py

Design & Simulation Flow (1)

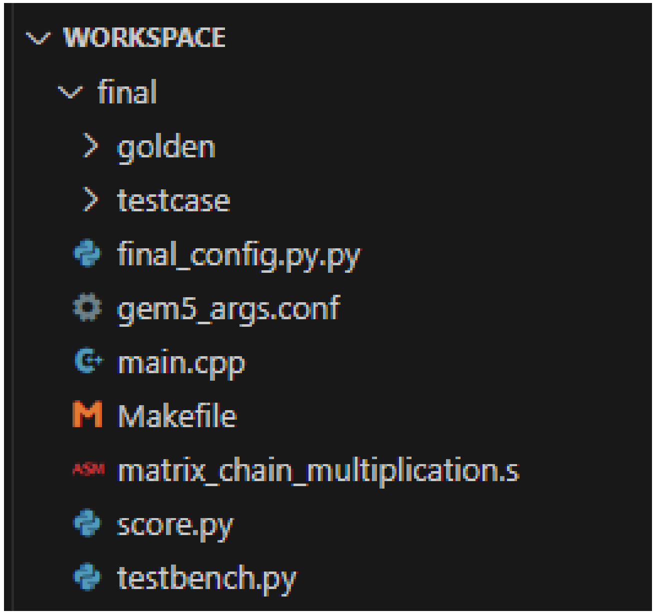
- Start docker

```
docker start -i <docker_name>
```

- Create folder for final

```
mkdir workspace/final
```

```
cd workspace/final
```



- Put all provided file in workspace/final

- Complete the design in matrix_chain_multiplication.s

Design & Simulation Flow (2)

- Modify the cache setting in gem5_args.conf
  - Settings you can change (mark in red):
  - GEM5_ARGS = --l1i_size 4kB --l1i_assoc 2 --l1d_size 4kB --l1d_assoc 2 --l2_size 16kB --l2_assoc 4

```
# Modify the GEM5 arguments for the simulation
GEM5_ARGS = --l1i_size 4kB --l1i_assoc 2 --l1d_size 4kB --l1d_assoc 2 --l2_size 16kB --l2_assoc 4
```

- Compile main.cpp with matrix_chain_multiplication.s

```
make g++_final
```

Design & Simulation Flow (3)

- Run simulation with all testcases or single testcase
  - This step might take a few minutes

```
make gem5_public_all
```

```
make gem5_public ARGS=P0/P1/P2/P3/P4/P5
```

- Checking if generated answer.txt match golden.txt

```
make testbench_public
```

- Compute performance score based on performance formula
  - Read content from m5out/config.json and m5out/out_exec.txt

```
make score_public
```

Test Cases Information

- Constraints:
  - Maximum number of matrices (N): 16
  - Maximum dimension of matrices (D): 64
  - Maximum value of matrices (V): 128

- Public Test Cases:
  - 00: N = 3, D = 3, V = 10
  - 01: N = 3, D = 5, V = 10
  - 02: N = 4, D = 6, V = 16
  - 03: N = 5, D = 10, V = 20
  - 04: N = 6, D = 8, V = 12
  - 05: public performance test

Group Formation

- 2 students per group
- Find your groupmate and submit the form before 5/4 (Sun.) 23:59
  - https://forms.gle/rKvznPAk3oMZWjWdA
- You may use NTU COOL 討論區 to find your groupmate
  - https://cool.ntu.edu.tw/courses/45288/discussion_topics/393734
- For those who haven’t submit the form, the TAs will randomly assign groupmate for you
- Email to andrew@access.ee.ntu.edu.tw for any private questions related to group formation
  - Subject should start with [113-2 CA 分組]

Submission

- Deadline: **2025/6/8 23:59:59 (UTC+8)**

- matrix_chain_multiplication.s
  - Your assembly code that implement matrix chain multiplication
- gem5_args.conf
  - Your simulation arguments (cache size & cache associative)
- report.pdf
  - 1-2 pages description of your design spec, special techniques used, team's division of work, and reflections.

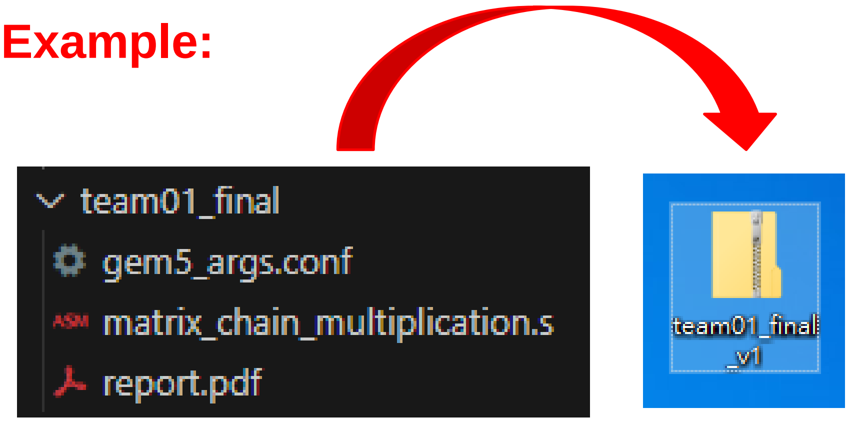
- Submit 組內互評 form
  - Contribution score for you and your teammate

Submission – Group

- Compress teamID_final/ in a zip file
  - named teamID_final_vk.zip (k: version number, e.g., 1,2,...)
- Upload teamID_final_vk.zip to NTUCOOL (ex: team01_final_v1.zip)

- File structure:
- teamID_final_vk.zip
  - teamID_final/
    - matrix_chain_multiplication.s
    - gem5_args.conf
    - report.pdf

Example:



- Incorrect file name or format would get **10% penalty**

Submission – Individual

- Submit 組內互評 form
  - https://forms.gle/x2cKjfPaR9GKfmji8
- Contribution score
  - 0 = no contribution
  - 10 = full contribution
  - Summation of all the group should be 10
- **Everyone** needs to submit this form !

Grading Policy

- Testcase 40% + Baseline 20% + Performance 20% + Report 20%
  - 0 points for late submission or plagiarism
  - -10% for incorrect naming or format

Item	%	Description
Public Testcases	30	Pass all 6 public testcases (5% each)
Private Testcases	10	Pass all 5 private testcases (2% each)
Baseline	20	Pass simple / standard / strong baseline
Public Ranking	15	Assign scores based on the PR value of your performance score of P5 (public).
Private Ranking	5	Assign scores based on the PR value of your performance score of P10 (private).
Report	15	1~2 pages
Contribution	5	Based on 組內互評

Grading Policy

- Baseline (total 20%)
  - Simple baseline (10%)
    - Performance score of testcase P5 $< 1.7 \times 10^8$
  - Standard baseline (5%)
    - Performance score of testcase P5 $< 1.2 \times 10^8$
  - Strong baseline (5%)
    - Performance score of testcase P5 $< 7 \times 10^7$

- Ranking (total 20%)
  - Performance score is acquired only after **passing all testcases**.
  - Public ranking (15%)
    - Based on performance of testcase P5
  - Private ranking (5%)
    - Based on performance of testcase P10

Discussion

- NTU COOL Discussion Forum: Final project 討論區
  - TAs will prioritize answering questions on the NTU COOL discussion forum

- Email: r13943002@ntu.edu.tw
  - Title should start with [113-2 CA Final Project]

Reference

- Matrix-chain multiplication
  - Matrix Chain Multiplication | GeeksforGeeks
- Dynamic Programming
  - Dynamic Programming or DP | GeeksforGeeks
- Tiled matrix multiplication
  - penny-xu.github.io/blog/tiled-matrix-multiplication/