# FINAL PROJECT
# MQSIM CXL DESIGN TOOL

FROM OVERCOMING THE MEMORY WALL WITH CXL-ENABLED SSDS

Group 2
113522024 游振彤 113522011 胡嘉祐

# AGENDA OVERVIEW

01

PAPER REVIEW

02

MQSIM CXL INTRODUCTION

03

TRACE GENERATOR

04

TRACE TRANSLATOR

05

MQSIM CXL

06

PAPER REPRODUCTION

07

OUR EXPERIMENT

08

REVIEW

# 01 PAPER REVIEW

This paper investigates the feasibility of using inexpensive flash memory on new interconnect technologies such as CXL (Compute Express Link) to overcome the memory wall.

# CURRENT MEMORY ISSUES

### MEMORY WALL

處理器性能增長速度遠快於記憶體性能提升速度，且大型AI模型的出現對於記憶體容量有著更大的需求，導致處理器在等待數據傳輸時閒置限制系統性能

### CXL-ENABLED SSD (CXL MEMORY)

Compute Express Link (CXL) 技術將 SSD (Flash Memory) 整合進主記憶體層次的一部分，由於SSD的大容量，CXL Memory可提供大規模擴展記憶體（數 TB）的能力，很適合處理大數據工作負載。

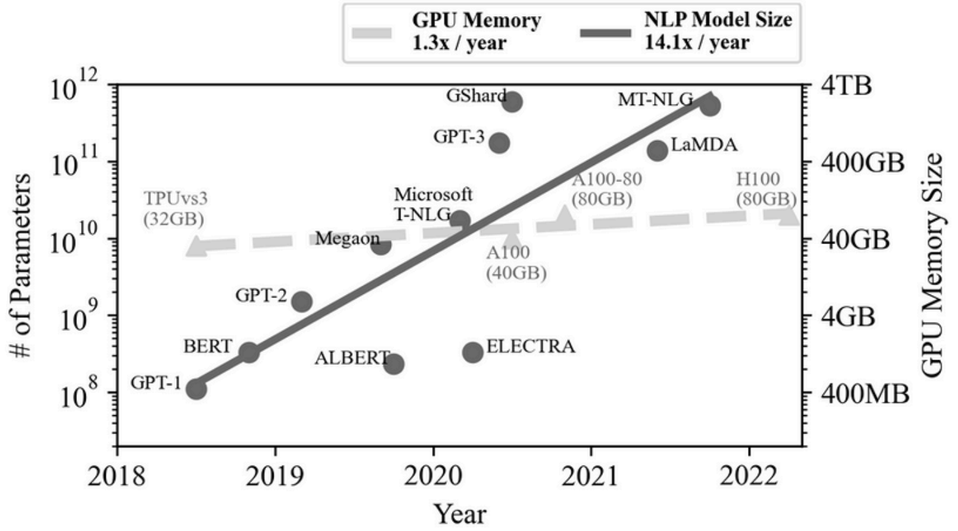



Figure 1: The trend in memory requirements for NLP applications [11, 30, 34, 43]. The number of parameters increases by a factor of 14.1× per year, while the memory capacity in GPUs only grows by a factor of 1.3× every year.

# CXL (FLASH) MEMORY ISSUES

### FLASH'S HISH LATENCY

Flash 的讀寫延遲高於 DRAM 可達到1000倍

### GRANULARITY MISMATCH

Flash 無法做到隨機讀取且是以 page (KB) 為單位進行讀寫，再加上需要經過 erase 操作才能重新寫入。

### LIMITED ENDURANCE

Flash 記憶體在寫入和擦除過程中需要高電壓，這會逐漸損耗記憶體單元，使其最終無法使用，而當 Flash 作為 Memory 使用，此問題會更加嚴重。

Table 1: Overview of memory technology characteristics.

| Technology | Read latency | Program latency | Erase latency | Endurance limit |
|---|---|---|---|---|
| DRAM [50] | 46ns | 46ns | N/A | N/A |
| ULL [46, 76] | 3$\mu$s | 100$\mu$s | 1000$\mu$s | 100K |
| SLC [24] | 25$\mu$s | 200$\mu$s | 1500$\mu$s | 100K |
| MLC [24] | 50$\mu$s | 600$\mu$s | 3000$\mu$s | 10K |
| TLC [24] | 75$\mu$s | 900$\mu$s | 4500$\mu$s | 3K |

# 02
# MQSIM CXL INTRODUCT.

CXL-flash Design Tools contain two primary tools, a memory tracing tool and a trace-driven simulator (this repository), to assist designing a CXL-enabled flash memory device for main memory expansion.

# MQSIM CXL DESIN TOOL

為了模擬 CPU 與 CXL Memory 之間的物理記憶體訪問行為，作者設計一個基於 MQSim 的模擬器。

### TRACE GENERATOR

透過修改 Linux 核心與使用 Valgrind 工具，捕捉應用程序的虛擬記憶體訪問模式，並記錄頁表更新以獲取虛擬到物理地址的映射。

### TRACE TRANSLATOR

將捕捉到的 *.vout 或 *.pout 檔案轉成 *.trace 檔

### MQSIM CXL

基於 MQSim 與其擴展版 MQSim-E，模擬 DRAM Cache、Flash Translation Layer (FTL)、多通道 Flash 管理等。

### MQSIM

MQSim: NAND Flash 開源模擬器
MQSIM-E: 支援更多功能，原本是用來研究 Enterprise SSD

# 03 TRACE GENERATOR

透過修改 Linux 核心與使用 Valgrind 工具，捕捉應用程序的虛擬記憶體訪問模式，並記錄頁表更新以獲取虛擬到物理地址的映射，並在後續的 Simulator 中使用此 trace 來進行模擬

# TRACE GENERATOR

### VALGRIND

Valgrind 是一個開源的程序分析工具套件，主要用於**動態分析**程式執行時的行為。它支持多種分析工具，幫助開發者檢測程序中的潛在問題，特別是在記憶體使用和性能方面。

### LINUX KERNEL MODIFICATION

作者在 Linux Kernel 中紀錄當 Page Fault 發生時的 VPN-PFN Mapping，因此可以重建（翻譯）出執行程式時實際 physical address 的使用狀況

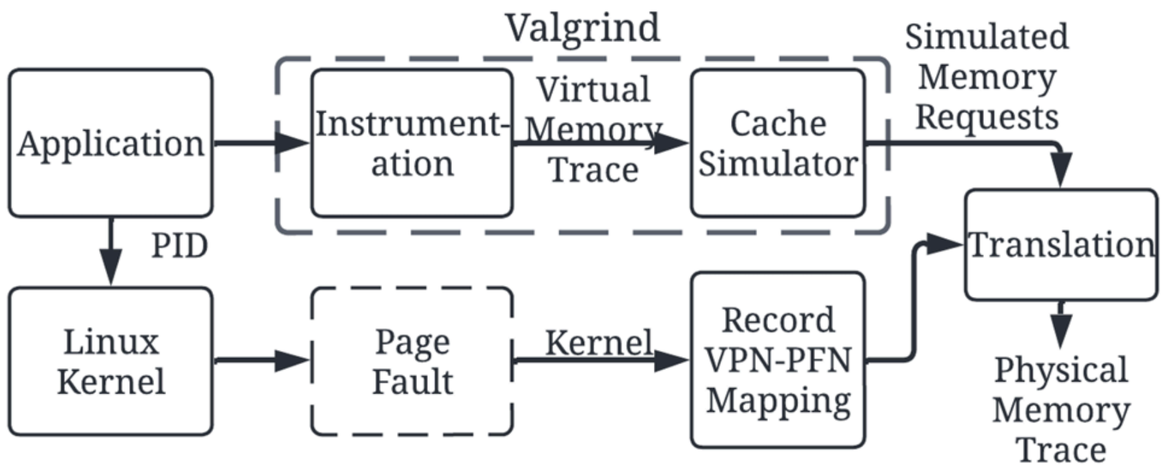



Figure 2: Workflow for collecting physical memory traces. We collect the virtual memory trace using Valgrind and simulate its behavior in the cache. Simultaneously, we capture page fault events to trace the updates to the page table, and use this to generate physical memory traces.

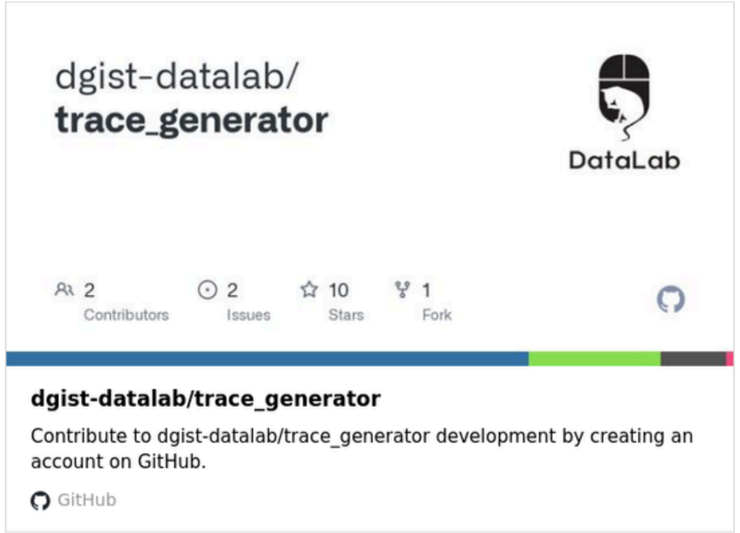
# INSTALLATION

## VALGRIND

```
$ git clone https://github.com/dgist-datalab/valgrind_cachetrace.git
$ cd valgrind_cachetrace
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

## LINUX KERNEL

```
$ git clone https://github.com/dgist-datalab/cxl-kernel.git
$ cd cxl-kernel
$ cp arch/x86/configs/ubuntu_defconfig .config
$ make bindeb-pkg -j64
$ cd ../
$ dpkg -i linux-headers-5.17.4-... linux-image-5.17.4-...
$ reboot
...
```



# TRACE GENERATION

這邊以作者提供的test .sh 檔案作範例，另外若要使用plot功能的
話需要自己額外呼叫 python 檔案

```
jacky@Memory:~/trace_generator/test$ sudo ./test_synthetic.sh
[sudo] password for jacky:
Please select a synthetic workload to test:
1) indirect_delta  3) heap            5) hashmap
2) true_random     4) strided_latjob
Input workload number: 1
You chose indirect_delta
Get only virtual address trace or both virtual/physical address trace? [v/p] v
selected trace type: virutal
$/home/jacky/trace_generator/run_script/run_script.sh --type virutal --pref --outname indirect_delta --
input /home/jacky/trace_generator/test/Synthetic_Workload/indirect_delta
Note: '--input [args]' must be placed at the end
===parsed command line option===
 - input: /home/jacky/trace_generator/test/Synthetic_Workload/indirect_delta
 - type: virutal
 - pref: True
 - nolog:
Valgrind CPU Prefetch: ON
Run /home/jacky/trace_generator/test/Synthetic_Workload/indirect_delta with valgrind..
process pid: 607092 (name: /bin/bash /home/jacky/trace_generator/run_script/run_script.sh --type viruta
l --pref --outname indirect_delta --input /home/jacky/trace_generator/test/Synthetic_Workload/indirect_
delta )
runtime: 106 (sec)
Process end
```

# 04

# TRACE TRANSLATION

作者提供簡單檔案即可將 out 檔轉成給模擬器執行的 trace 檔

# TRACE TRANSLATOR

## USAGE

The *.vout or *.pout trace files need to be translated into a simulator-compatible format. Please utilize example trace translation code to generate *.trace files from the *.vout or *.pout files.

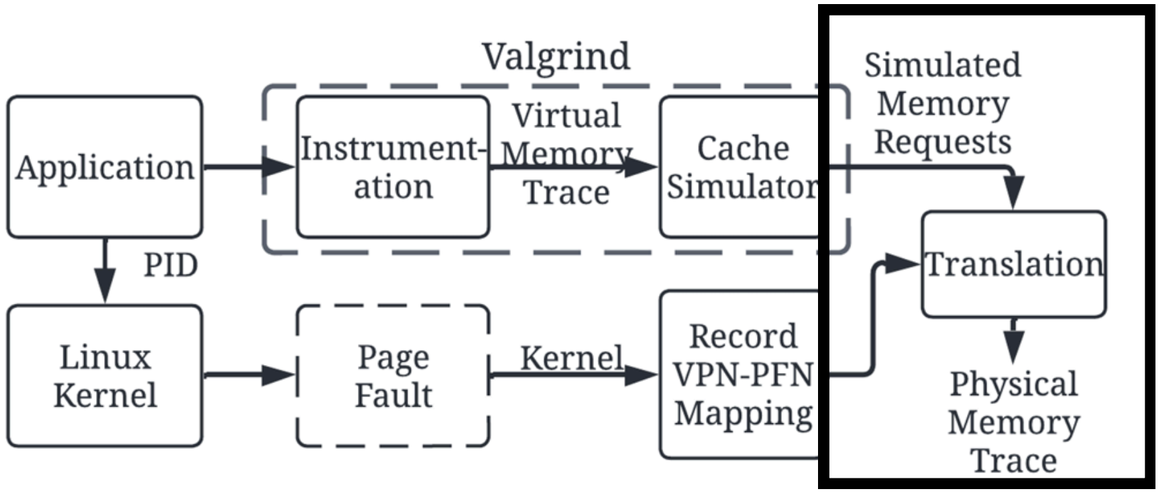



Figure 2: Workflow for collecting physical memory traces. We collect the virtual memory trace using Valgrind and simulate its behavior in the cache. Simultaneously, we capture page fault events to trace the updates to the page table, and use this to generate physical memory traces.

# 05
# MQSIM CXL

MQSim CXL 是基於 MQSim 擴展的模擬工具，專門設計用來模擬 CXL (Compute Express Link) 記憶體系統的性能和行為，此章節將會展示如何使用 MQSIM CXL 進行 CXL Memory 的模擬，並針對三個參數檔案：config.txt、ssdconfig.xml 和 workload.xml 進行參數設置的講解

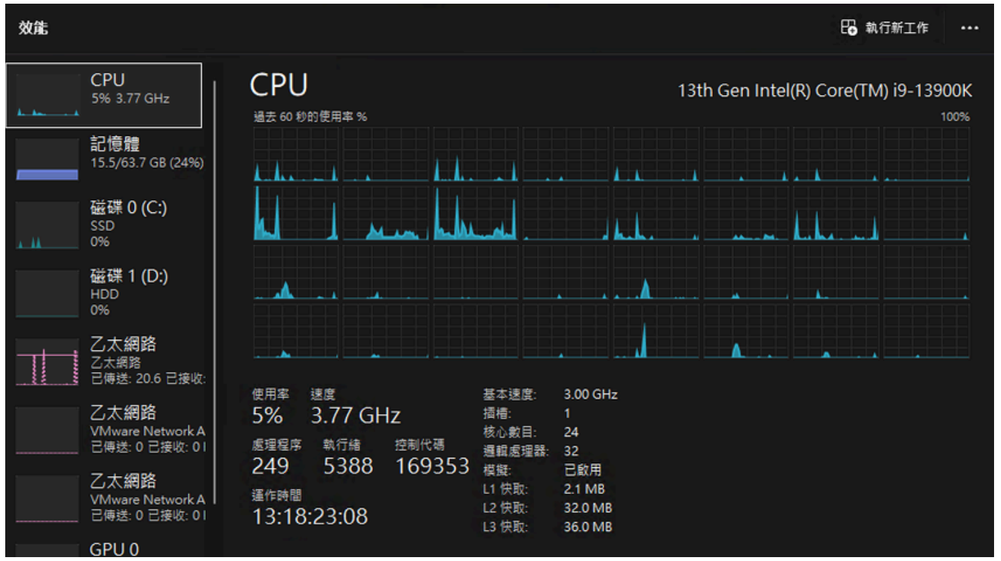
# ENVIRONMENT

## 硬體環境

作業系統：Windows 11

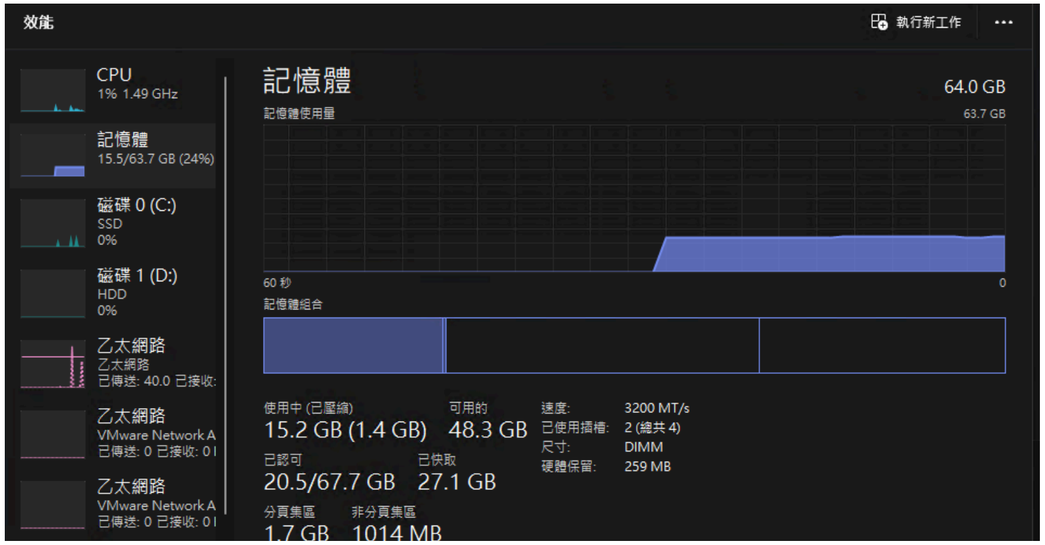
CPU：intel i9-13900K

SSD：64(32*2)GB DDR4 3200MT




## 編譯（開發）軟體

Visual Studio 2022 IDE



# WORKLOAD (TRACE FILES)

## THE SYNTHETIC WORKLOAD

Hash map
Matrix multiply (indirect delta)
Min heap
Random
Stride

## THE REAL-WORLD WORKLOAD

BERT[1]
Page rank (GAPBS[2])
Radiosity (Splash-3[3])
XZ (Spec CPU [4], [5])
YCSB F[6]

zenodo Search records... Communities My dashboard

**Info:** Zenodo's user support line is staffed on regular business days between Dec 23 and Jan 5. Response times may be slightly longer tha

Published May 9, 2023 | Version v1 Dataset Open

### Memory Traces for the CXL-flash Simulator

Shao-Peng Yang[1] ; Minjae Kim[2] ; Sanghyun Nam[3] ; Juhyung Park[2] Show affiliations

This dataset contains memory trace files (*.trace) for various synthetic workloads and real-world workloads.

We utilize Trace Generator to collect traces for these workloads and translate them into *.trace files

We collected these traces on a machine with an Intel i7-9700 CPU and 32GB DDR4 DRAM Memory. The operating system utilized is Ubuntu 20.04.5 LTS with Linux Kernel 5.17.4.

The real-world workloads include:

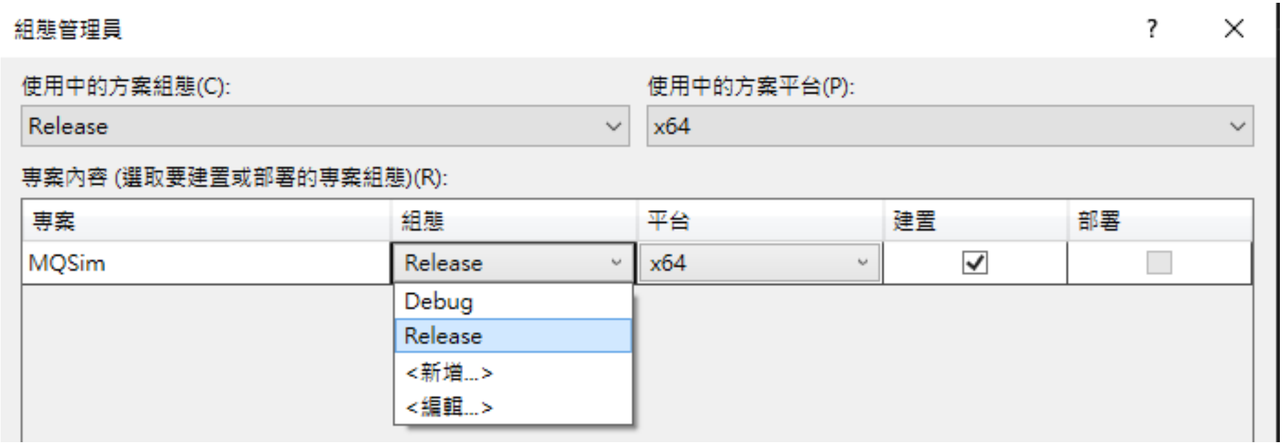
- BERT
- Page rank (GAPBS)
- Radiosity (Splash-3)
- XZ (Spec CPU)
- YCSB F

Users can find the source codes of the real-world workloads from their original code bases.

# SIMULATION WORKFLOW

### 1. COMPILE 模擬器

在Visual Studio中將版本轉換為release後進行建置，若後續有調整到模擬器檔案須重新執行此步




### 2. 設定參數

在 config.txt, ssdconfig.xml 以及 workload.xml 中調整對應參數

### 3. 執行模擬

於repository中執行指令：

```
.\MQSim.exe -i ssdconfig.xml -w workload.xml
```

# CONFIG.TXT

- DRAM_mode：設為1表示僅用DRAM運行，設為0表示使用CXL-flash運行。
- Has_cache：設為1表示包含DRAM快取，設為0表示僅使用Flash後端。
- DRAM_size：設定DRAM快取大小，單位為位元組。
- Mix_mode：設為1混合處理需求未命中和預取數據，設為0分開緩衝區，目前僅支持1。
- Cache_portion_percentage：每個快取部分大小百分比基本為100。
- Has_mshr：設為1表示包含MSHR，設為0則不使用MSHR。
- Cache_placement：設置快取的組關聯性，範圍為1到**DRAM大小除以4096（快取行大小）**。
- Cache_policy：指定快取策略，可選隨機、FIFO、LRU、CFLRU。
- Prefetcher：指定預取策略，可選不啟用、Next-N-line、最佳偏移、跳躍式、直接反饋。
- Total_number_of_requests：指定追蹤文件中的請求總數。

config.txt

```
DRAM_mode 0
Has_cache 1
DRAM_size 67108864
Mix_mode 1
Cache_portion_percentage 100
Has_mshr 1
Cache_placement 16
Cache_policy CFLRU
Prefetcher Best-offset
Total_number_of_requests 20000000
```

realworld workload configurations

# WORKLOAD.XML

workload.xml

```xml
<?xml version="1.0" encoding="us-ascii"?>
<MQSim_IO_Scenarios>
    <IO_Scenario>
        <IO_Flow_Parameter_Set_Trace_Based>
                <Priority_Class>HIGH</Priority_Class>
                <Device_Level_Data_Caching_Mode>TURNED_OFF</Device_Level_Data_Caching_Mode>
                <Channel_IDs>0,1,2,3,4,5,6,7</Channel_IDs>
                <Chip_IDs>0,1,2,3,4,5,6,7</Chip_IDs>
                <Die_IDs>0</Die_IDs>
                <Plane_IDs>0,1,2,3</Plane_IDs>
                <Initial_Occupancy_Percentage>30</Initial_Occupancy_Percentage>
                <File_Path>traces/ycsb_f_v.trace</File_Path>
                <Percentage_To_Be_Executed>100</Percentage_To_Be_Executed>
                <Relay_Count>1</Relay_Count>
                <Time_Unit>NANOSECOND</Time_Unit>
        </IO_Flow_Parameter_Set_Trace_Based>
    </IO_Scenario>
</MQSim_IO_Scenarios>
```

此檔案主要會修改 File_path 讀取不同的workload (trace 檔)，以及
調整 parallelism 時會改變Channel與Chip IDs

# SSDCONFIG.XML

ssdconfig.xml

```xml
<Execution_Parameter_Set>
    <Device_Parameter_Set>
        <Flash_Parameter_Set>
            <Flash_Technology>SLC</Flash_Technology>
            <CMD_Suspension_Support>ERASE</CMD_Suspension_Support>
            <Page_Read_Latency_LSB>3000</Page_Read_Latency_LSB>
            <Page_Read_Latency_CSB>3000</Page_Read_Latency_CSB>
            <Page_Read_Latency_MSB>3000</Page_Read_Latency_MSB>
            <Page_Program_Latency_LSB>100000</Page_Program_Latency_LSB>
            <Page_Program_Latency_CSB>100000</Page_Program_Latency_CSB>
            <Page_Program_Latency_MSB>100000</Page_Program_Latency_MSB>
            <Block_Erase_Latency>1000000</Block_Erase_Latency>
            <Block_PE_Cycles_Limit>10000</Block_PE_Cycles_Limit>
            <Suspend_Erase_Time>100000</Suspend_Erase_Time>
            <Suspend_Program_Time>20000</Suspend_Program_Time>
            <Die_No_Per_Chip>1</Die_No_Per_Chip>
            <Plane_No_Per_Die>4</Plane_No_Per_Die>
            <Block_No_Per_Plane>512</Block_No_Per_Plane>
            <Page_No_Per_Block>512</Page_No_Per_Block>
            <Page_Capacity>16384</Page_Capacity>
            <Page_Metadat_Capacity>448</Page_Metadat_Capacity>
        </Flash_Parameter_Set>
    </Device_Parameter_Set>
</Execution_Parameter_Set>
```

在ssdconfig.xml中會調整模擬的 SSD 之參數，可以細節的去根據所使用的 SSD 技術調整如SLC/ MLC/ TLC甚至是所有 SSD 內部操作的 Latency 時長，預設為了方便在LSB, CSB 和 MSB的速度都為一致，但自行實驗時可以更細節的去調整

# 06
# PAPER REPRODUCE

此節我們將會嘗試復現論文中出現的幾個實驗，並在將數據整理後討論可能與論文中結果不同的原因

# 1 TRACE GENERATION

安裝完成VALGRIND後
執行底下的程式即可進行 TRACE GENERATION 並執行PYTHON檔案作圖

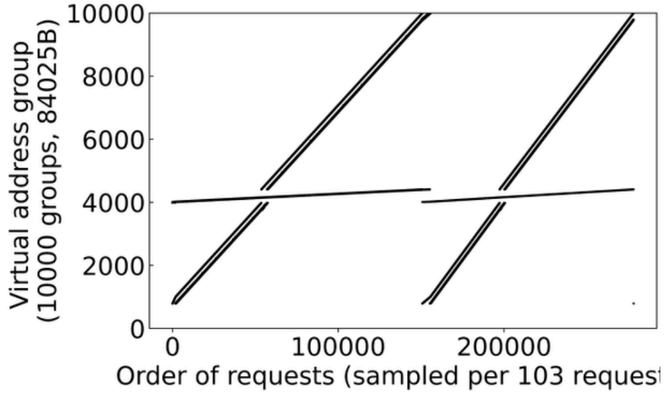
```
$ cd test/
$ sudo ./test_synthetic.sh
$ python3 after_run/graph/cg_histogram.py --input [.vout] --scatter 103
$ python3 after_run/graph/cg_pa_histogram.py --input [.pout] --scatter 103
```

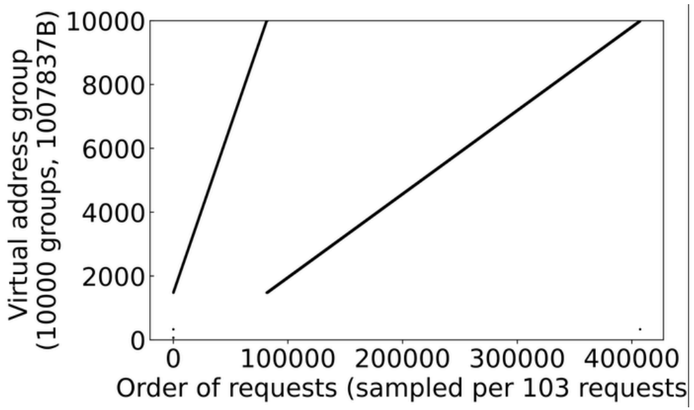
```
jacky@Memory:~/trace_generator/test$ python3 ../after_run/graph/cg_histogram.py --input indirect_delta.
vout --scatter 103
plot scattered order-address graph
linenum of indirect_delta.vout: 28641258
Find min/max of address..
 100% [28641000/28641258]min: 0x109100 max: 0x3225e5c0
group_size: 84025 group_num: 10000
Make scatter list..
 100% [28641000/28641258]len(scatter): 28641077
len(samp): 278069
jacky@Memory:~/trace_generator/test$
```

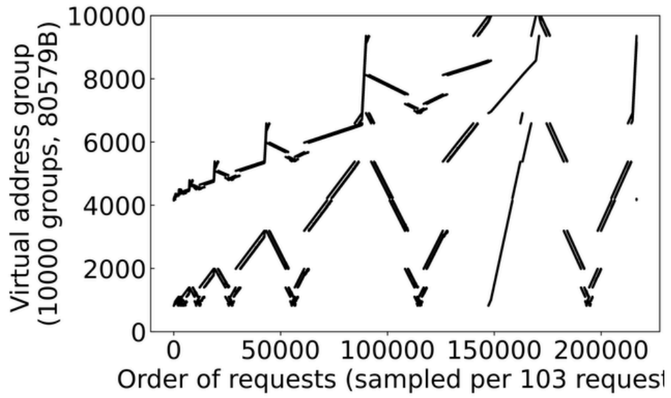
# 1 TRACE GENERATION

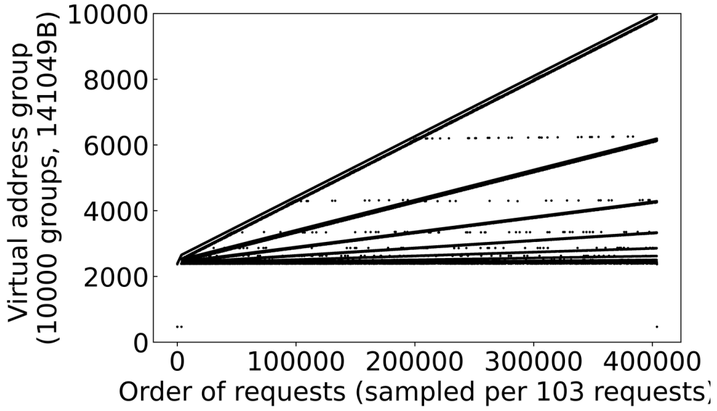
安裝完成VALGRIND後
執行右方的程式即可進行 TRACE GENERATION

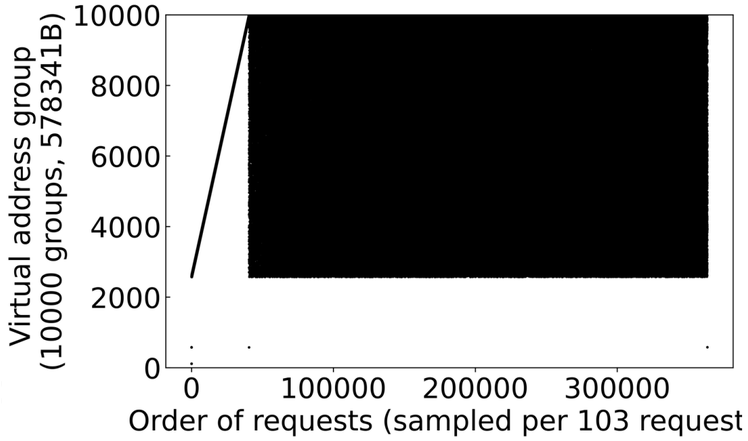
```
$ cd test/
$ sudo ./test_synthetic.sh
$ python3 after_run/graph/cg_histogram.py --input [.vout] --scatter 103
$ python3 after_run/graph/cg_pa_histogram.py --input [.pout] --scatter 103
```



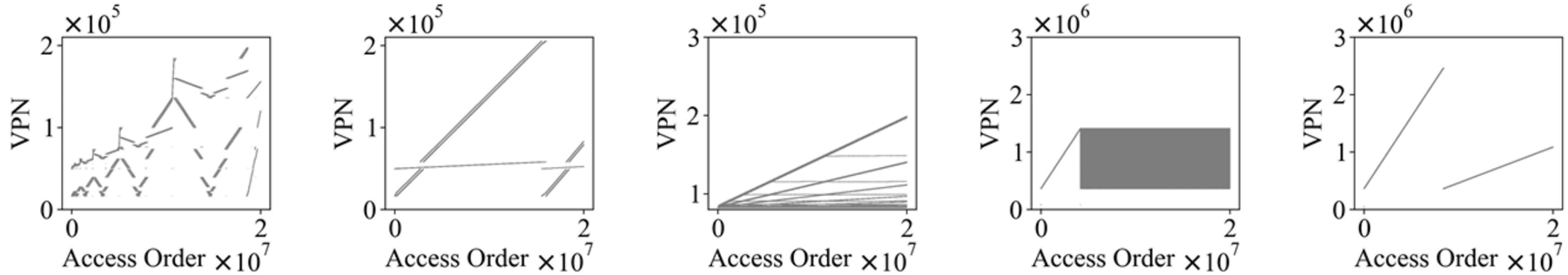



# 1 TRACE GENERATION

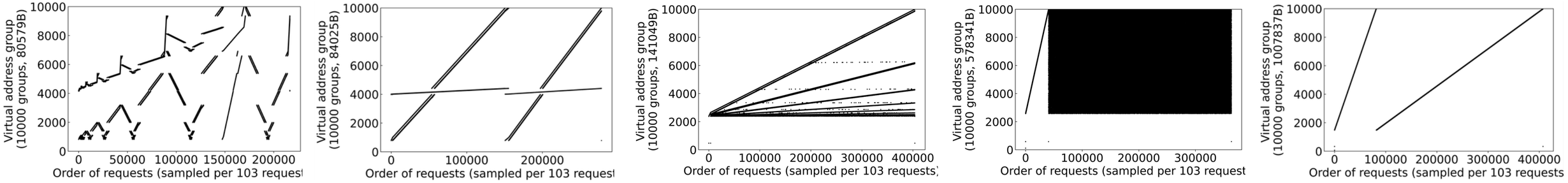
下方是論文內表格與我們生成的結果比較
兩者在PATTERN上基本一致

論文




(a) Hash map (V) (b) Matrix mult. (V) (c) Min heap (V) (d) Random (V) (e) Stride (V)

復現結果



# 2 CACHE PERFORMANCE

## CONFIGURATION 設定

使用github中給予的example synthetic workload，實驗只更改 Has_cache, Has_mshr 以及 DRAM_size 的設置

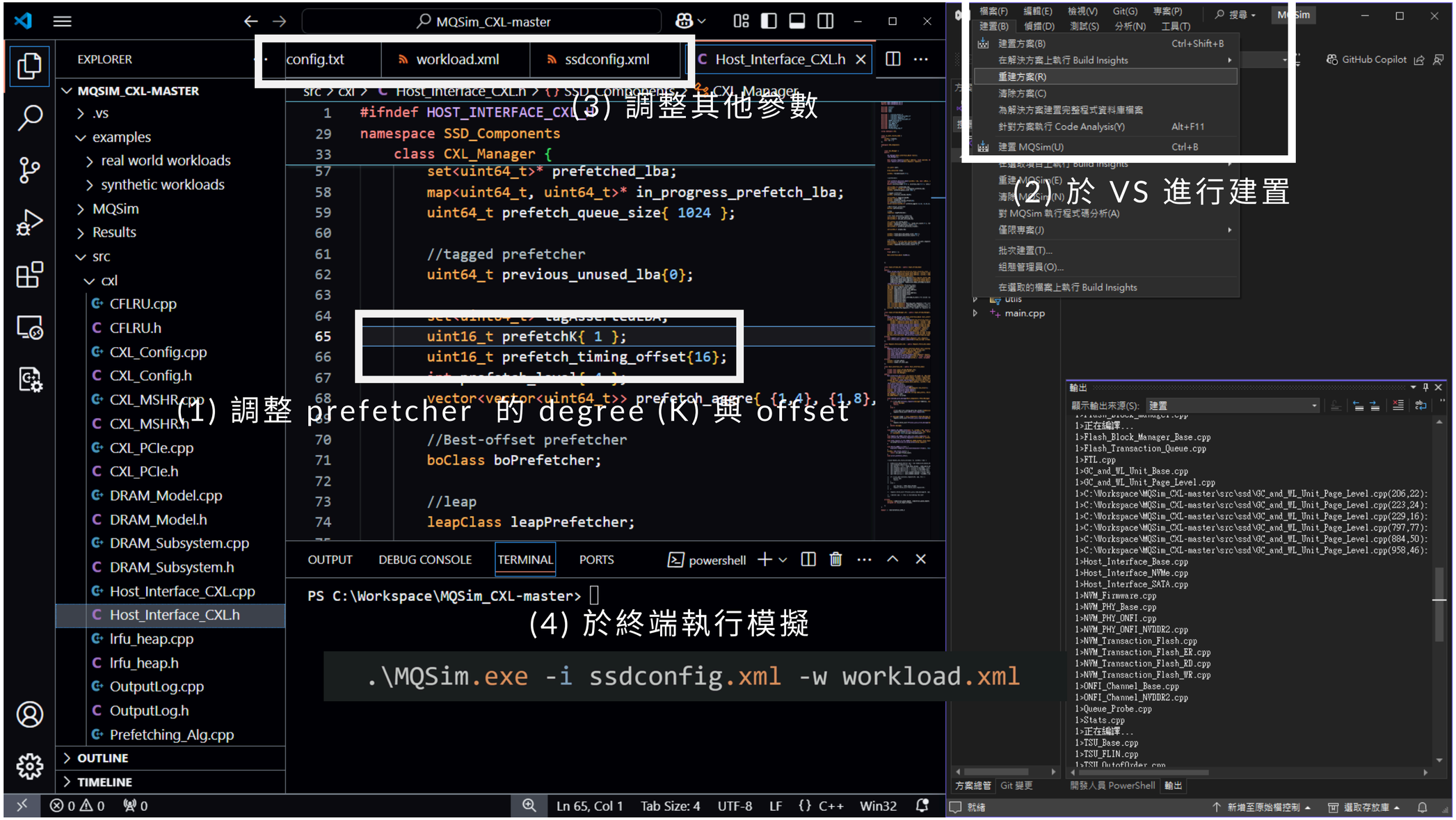
- no cache時將 Has_cahce 與 Has_mshr 設置成 0
- 有使用 cache 時則兩者設為 1 ，並且調整 DRAM_szie，為 (GiB 2^30) 乘上 GB 大小

## WORKLOAD 使用

使用在前述的介紹中有提到，作者提供的 trace file 進行實驗以達到最好的復現結果，包含以下的 synthetic workload (all physical)：

Hash map
Matrix multiply (indirect delta)
Min heap
Random
Stride

# WORKFLOW




倘若沒有進行(1)，則不需要重新建置(2)，只需要執行(3)和(4)即可

# 2 CACHE PERFORMANCE

## AVERAGE ACCESS LATENCY 計算

根據有沒有使用 cache ，再從不同的 latency 檔案中讀取所有 20000000 筆 requests 得出平均值作為 average access latency

## PREFETCHER 未使用

這邊要注意的是此實驗並沒有使用prefetcher進行預取

# 2 CACHE PERFORMANCE

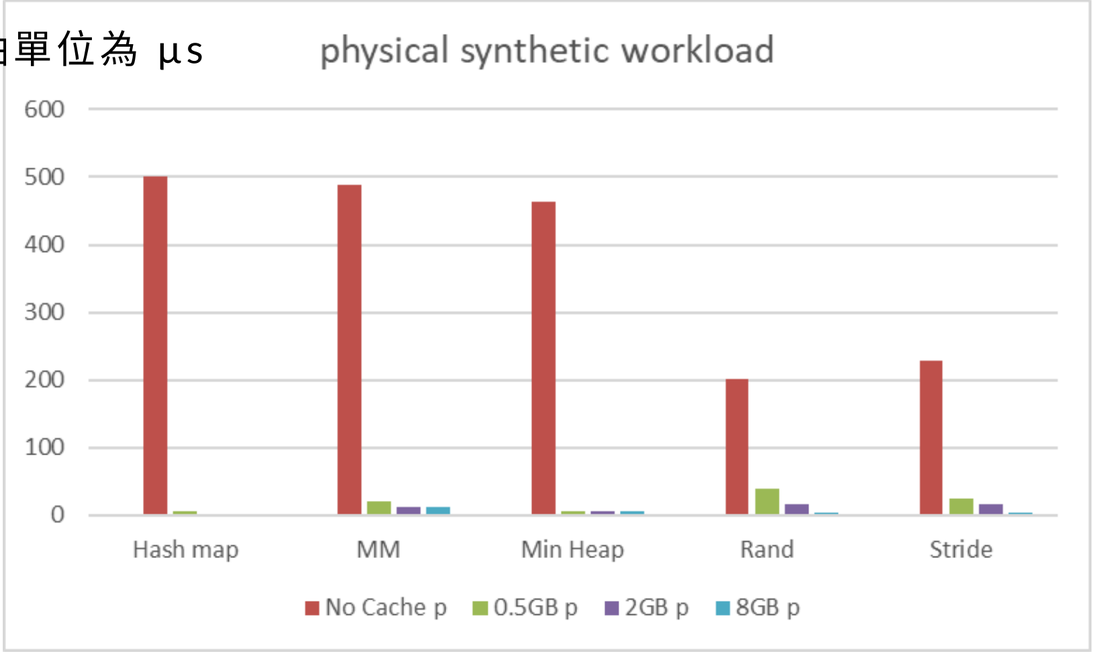
## SYNTHETIC PHYSICAL TRACES 結果

average access latency

| Workload_p | No Cache p | 0.5GB p | 2GB p | 8GB p |
|---|---|---|---|---|
| Hash map | 500.7 | 4.92 | 2.1 | 2.1 |
| MM | 488.657 | 21.461 | 13.046 | 13.046 |
| Min Heap | 463.771 | 5.919 | 5.919 | 5.919 |
| Rand | 200.408 | 38.965 | 16.492 | 4.576 |
| Stride | 229.07 | 24.581 | 16.281 | 4.809 |

單位為 μs

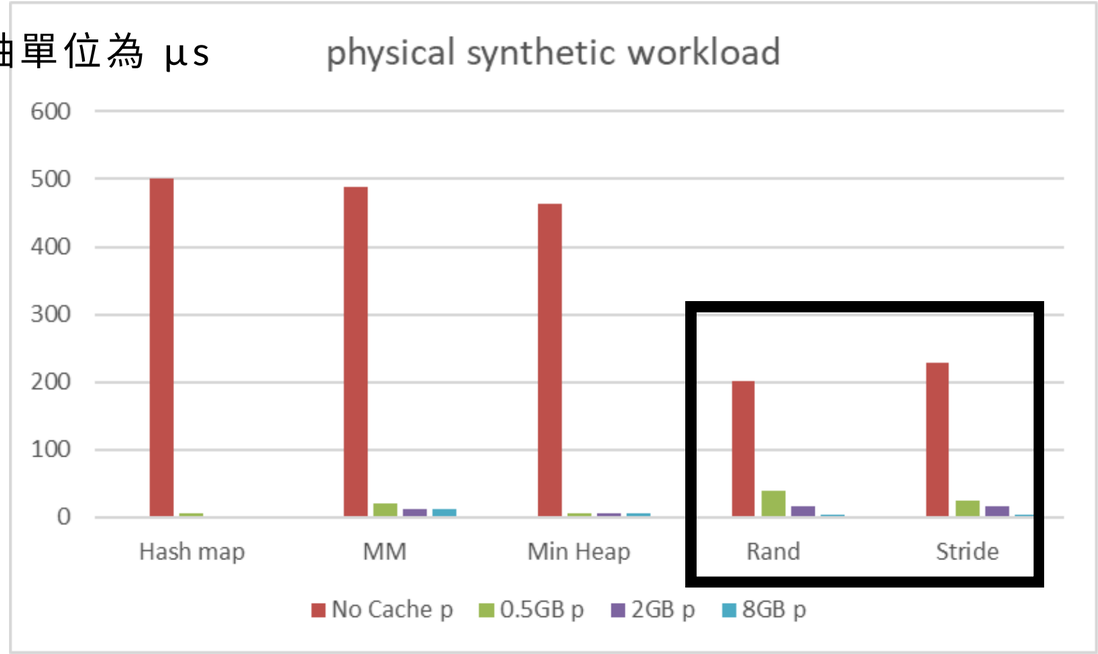
縱軸單位為 μs



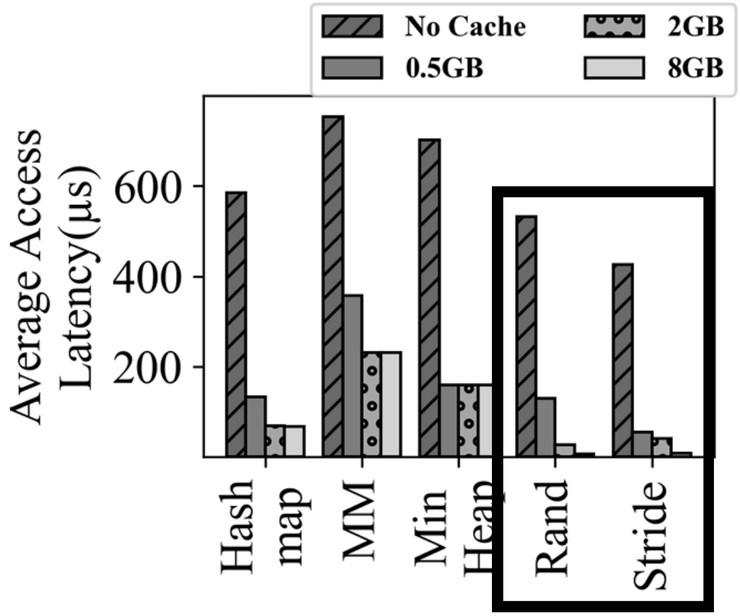
# 2 CACHE PERFORMANCE

## SYNTHETIC PHYSICAL TRACES 結果

縱軸單位為 μs




## 論文表格



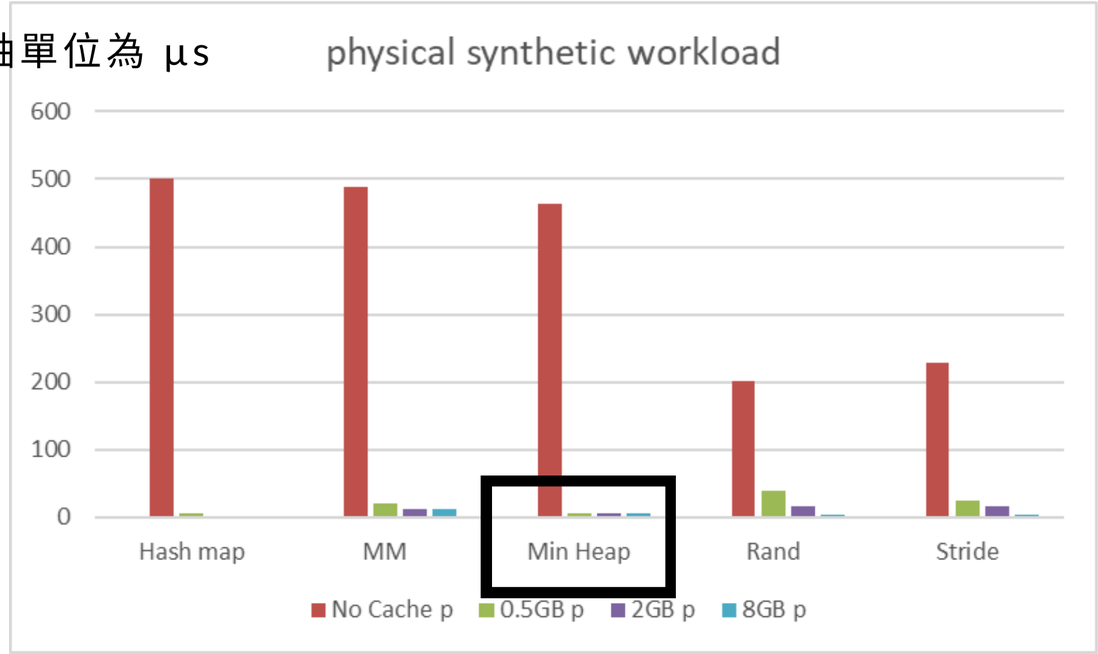

(a) Average access latency

可能是因為在 synthetic workload 中 parallelism degree 達到32（realworld workload只有8），chip id 有32個，因此 random 隨機分布反而能在 no cache 時透過平行執行提高效率

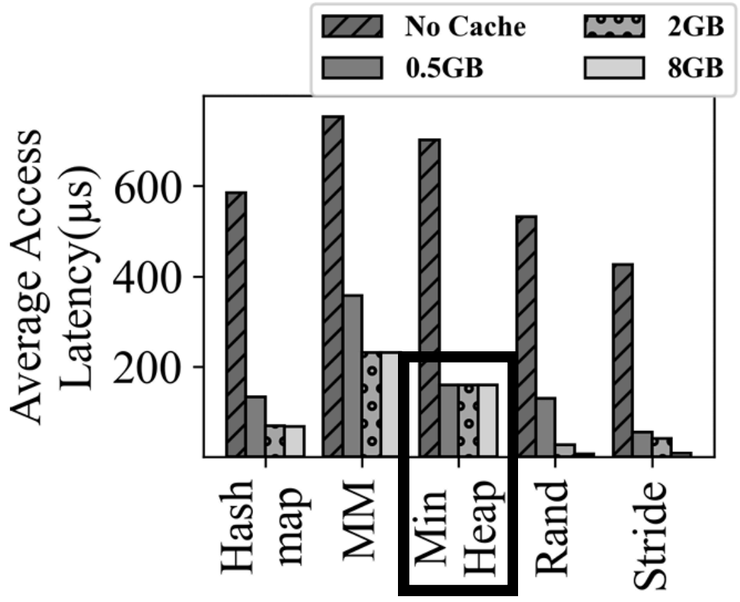
# 2 CACHE PERFORMANCE

## SYNTHETIC PHYSICAL TRACES 結果

縱軸單位為 μs




## 論文表格




(a) Average access latency

除此之外整體數值的趨勢與論文內的**數值趨勢**基本一致，並且可以看到 Min Heap 由於其高 Space Locality，因此當其使用記憶體時（足夠放下整個heap），延遲就不會變動

# SUB-MS RATIO VS AVERAGE LATENCY

## SUB-MS REQUEST RATIO

CXL-flash 結合了 DRAM Cache 和 Flash Memory，設計的目標是提供大容量同時保持低延遲。

Sub-μs Request 比例是驗證設計**是否成功隱藏 Flash 高延遲**的一個具體量化指標。

更能直接反映用戶在延遲敏感應用中的體感。

## AVERAGE ACCESS LATENCY

用於全面評估系統性能，包括冷數據或高延遲場景

對高延遲請求敏感，**可能被極端情況影響。**

與整體性能相關，但對用戶感知的極低延遲場景無法直接表現。

# 3 PREFETCHER

## CONFIGURATION 設定

- Has_cahce 與 Has_mshr 設置成1
- DRAM_szie 為 GiB 2^30*8
- FIFO cache policy
- Flash Parallelism 32x32
- Next-N-line prefetcher

- 調整 Prefetcher 的 (degree, offset) 為：
  - (1,16), (4, 16) (16,16), (4, 4), (4, 64)

- 此外由於調整 Prefetcher 的參數需要更改 /src/cxl/Host_Interface_CXL.h中的 prefetchK 與 prefetch_timing_offset，因此每次皆需要重新建置(compile)

## WORKLOAD 使用

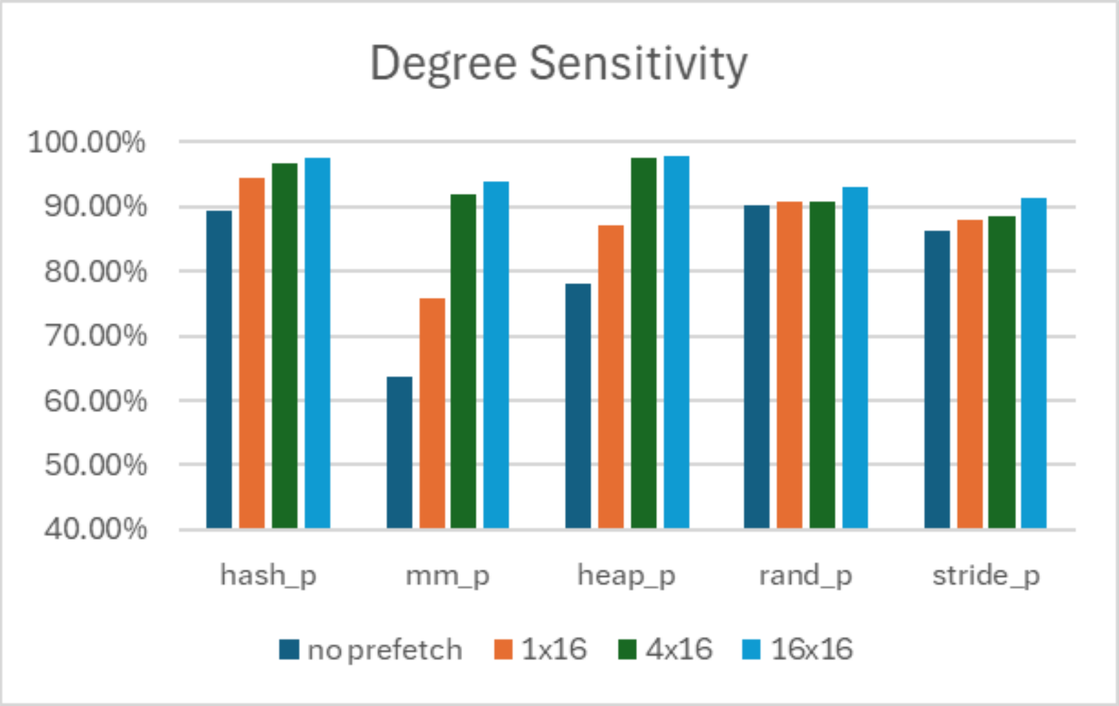
與 cache performance 的實驗一樣，包含以下的 synthetic workload (all physical)：

Hash map
Matrix multiply (indirect delta)
Min heap
Random
Stride

# 3 PREFETCHER

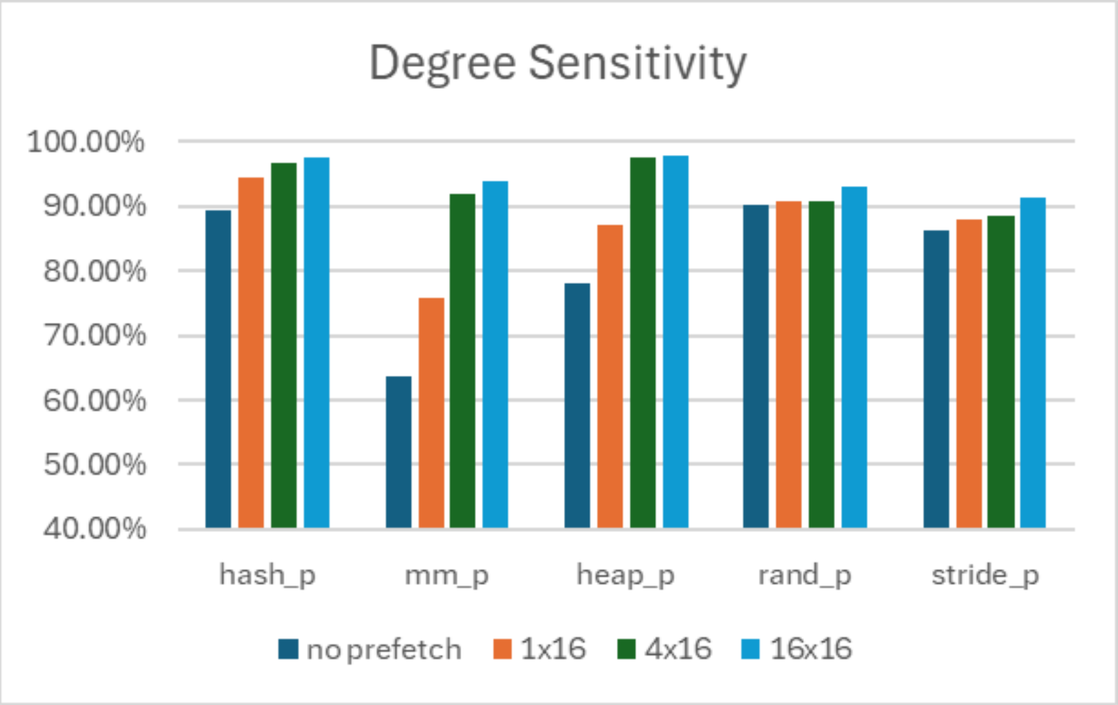
## DEGREE SENSITIVITY (X, 16) X=1,4,16

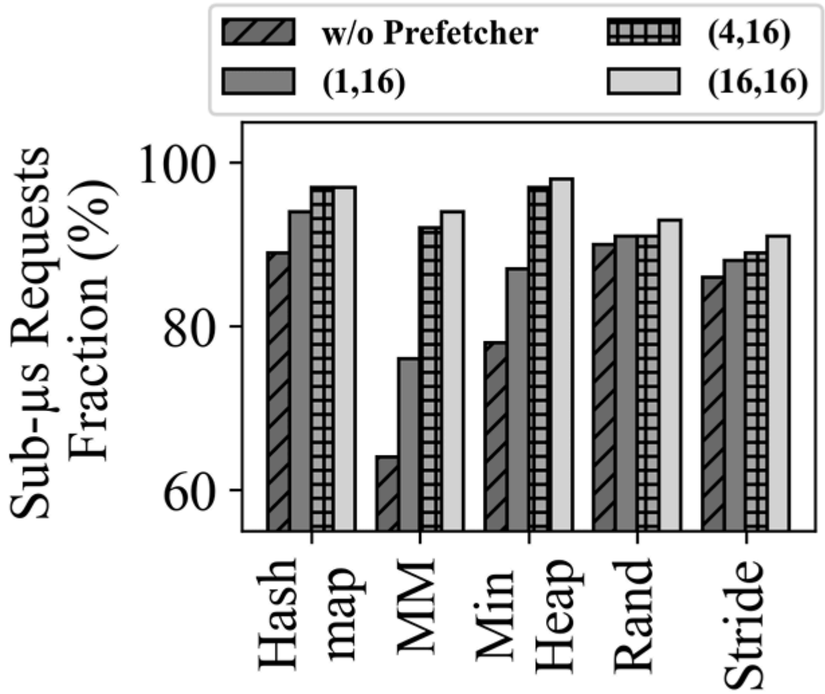
| | hash_p | mm_p | heap_p | rand_p | stride_p |
|---|---|---|---|---|---|
| no prefetch | 89.28% | 63.70% | 78.04% | 90.11% | 86.30% |
| 1x16 | 94.35% | 75.69% | 86.98% | 90.61% | 88.06% |
| 4x16 | 96.56% | 91.92% | 97.42% | 90.69% | 88.56% |
| 16x16 | 97.44% | 93.73% | 97.78% | 93.05% | 91.39% |



# 3 PREFETCHER

## DEGREE SENSITIVITY (X, 16) X=1,4,16



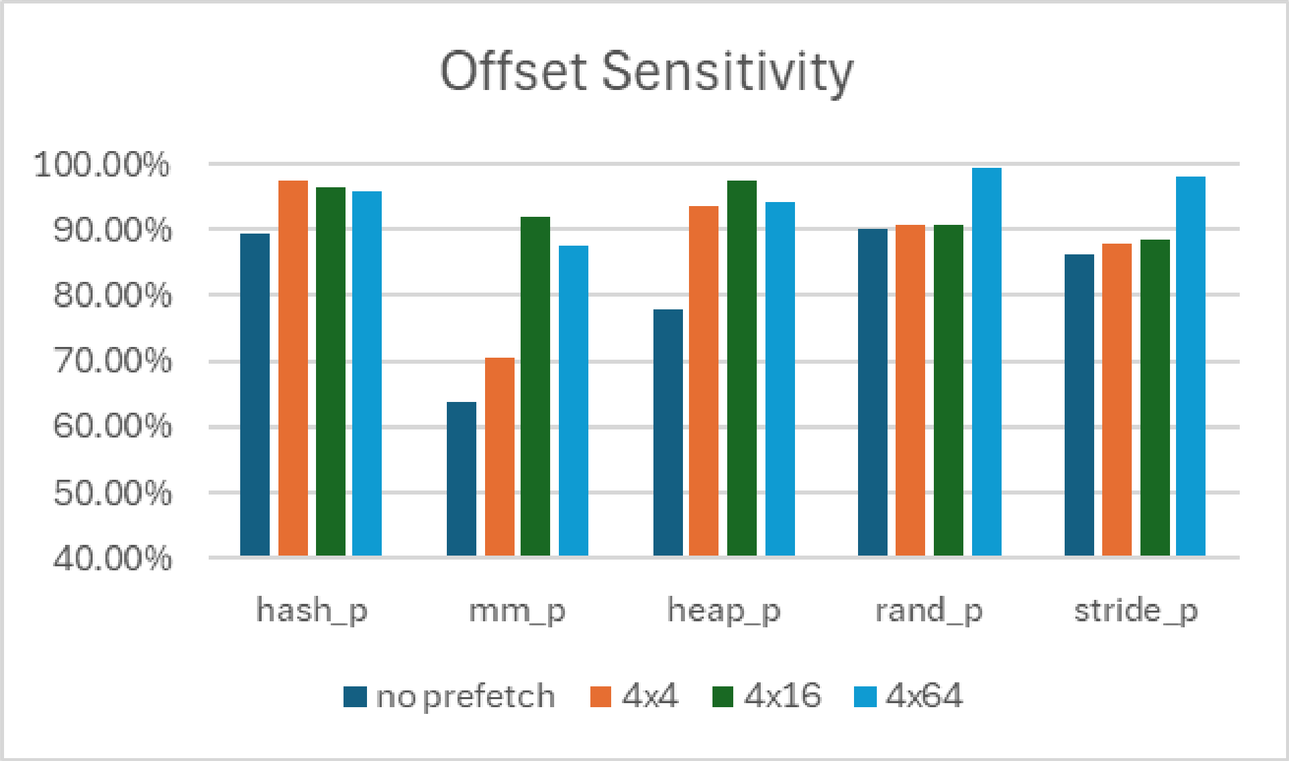




(a) Sensitivity to degree

# 3 PREFETCHER

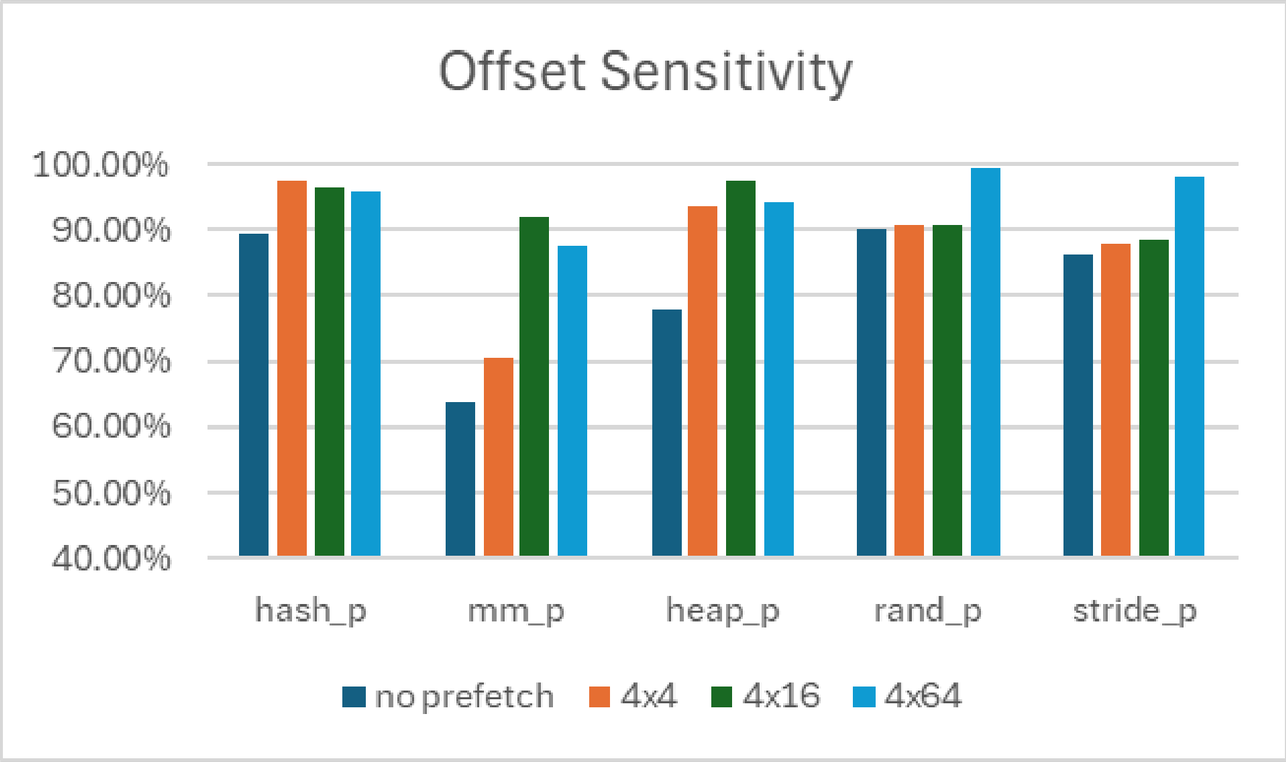
## OFFSET SENSITIVITY (4, Y) Y=4, 16, 64

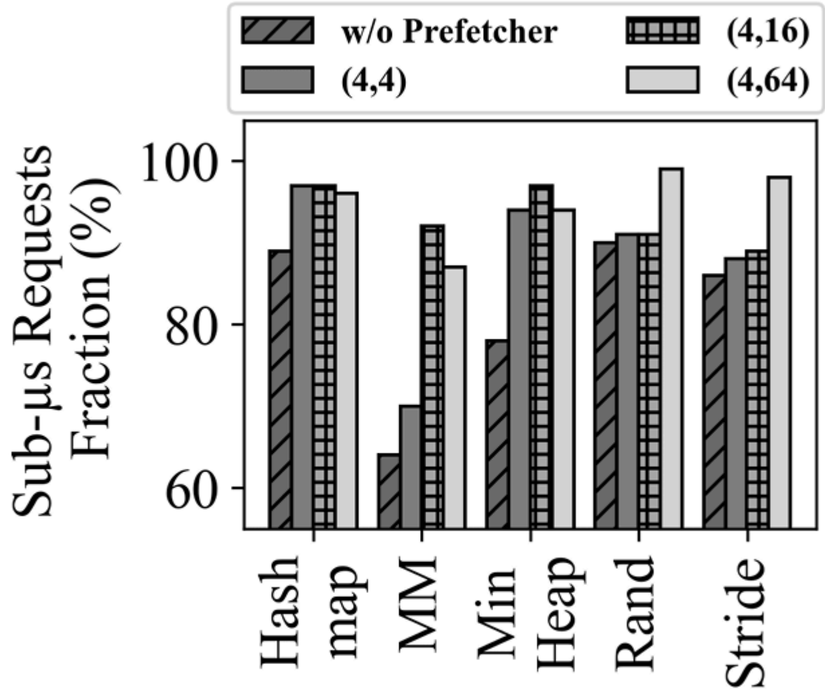
| | hash_p | mm_p | heap_p | rand_p | stride_p |
|---|---|---|---|---|---|
| no prefetch | 89.28% | 63.70% | 78.04% | 90.11% | 86.30% |
| 4x4 | 97.40% | 70.38% | 93.73% | 90.63% | 87.94% |
| 4x16 | 96.56% | 91.92% | 97.42% | 90.69% | 88.56% |
| 4x64 | 95.93% | 87.40% | 94.39% | 99.48% | 97.99% |



# 3 PREFETCHER

## OFFSET SENSITIVITY (4, Y) Y=4, 16, 64



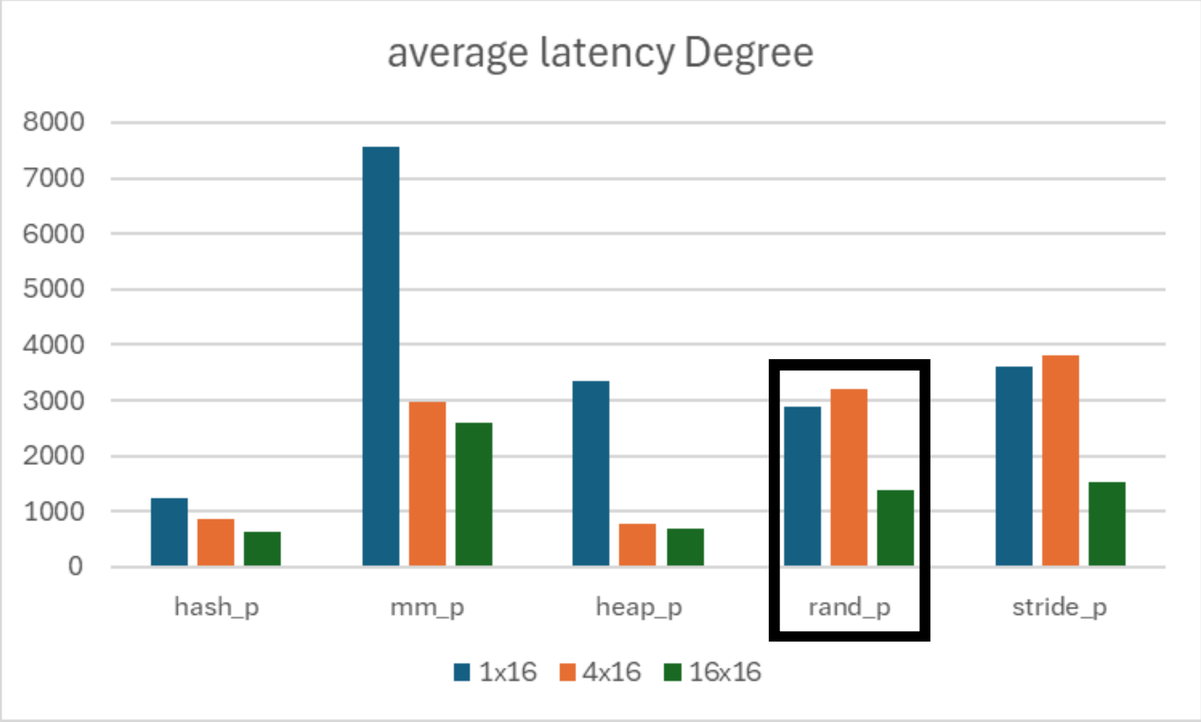



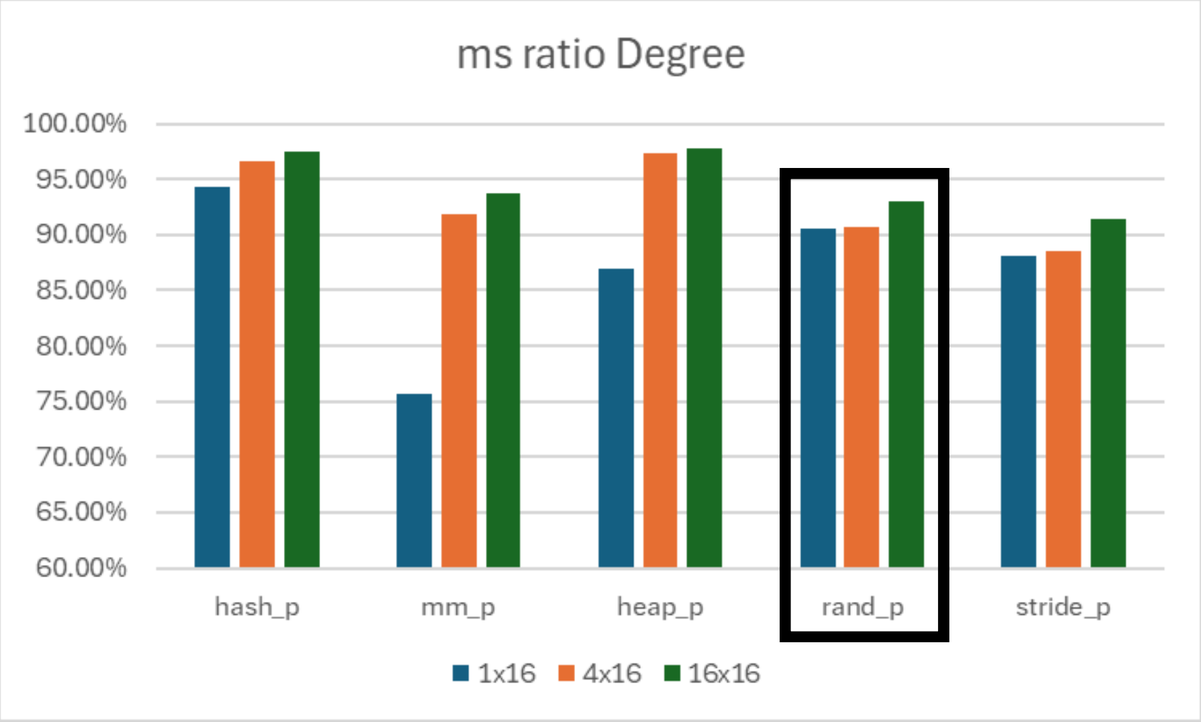

(b) Sensitivity to offset

兩圖皆與論文中基本一致，為成功的復現

# 3 PREFETCHER

## DIVE IN: AVERAGE LATENCY VS MS RATIO




兩者的表現雖大部分為正相關，但反應表現差異很大，主要就是因為 average latency 會受到極值影響

# 07
# OUR EXPERIEMNT

此節我們執行了幾個自行設想且沒有在論文中出現的實驗，並討論其結果是否符合預期

# 1 SYNTHETIC P VS V

## CONFIGURATION 設定

使用github中給予的example synthetic workload，實驗只更改 Has_cache, Has_mshr 以及 DRAM_size 的設置

- no cache時將 Has_cahce 與 Has_mshr 設置成 0
- 有使用 cache 時則兩者設為 1 ，並且調整 DRAM_szie，為 (GiB 2^30) 乘上 GB 大小

## WORKLOAD 使用

和復現中的第二個實驗一樣，使用作者提供的 trace file 進行實驗，包含以下的 synthetic workload：

Hash map
Matrix multiply (indirect delta)
Min heap
Random
Stride

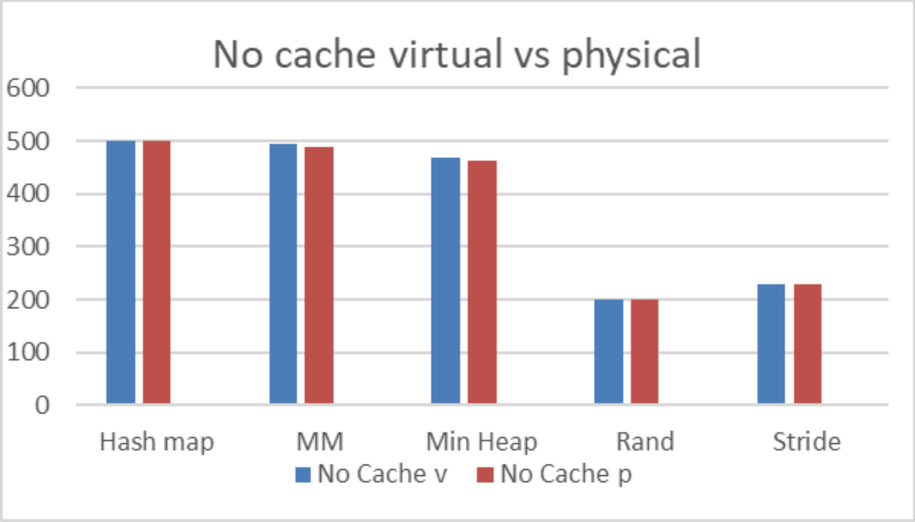
不過我們額外實驗了 virtual traces 進行比較

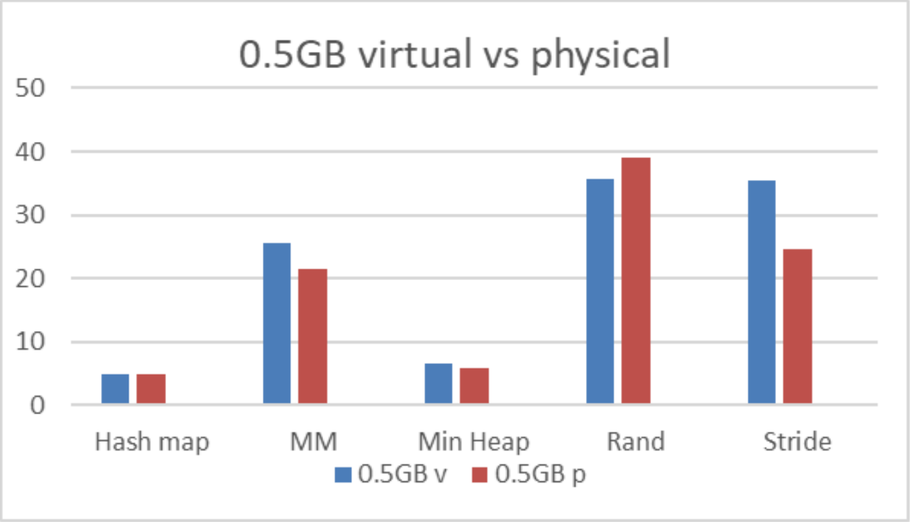
# 1 SYNTHETIC P VS V

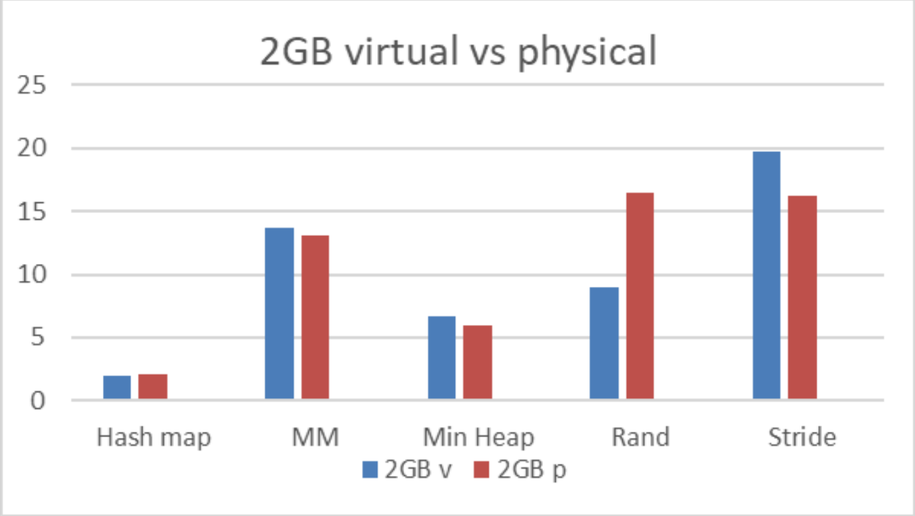
**AVERAGE ACCESS LATENCY**

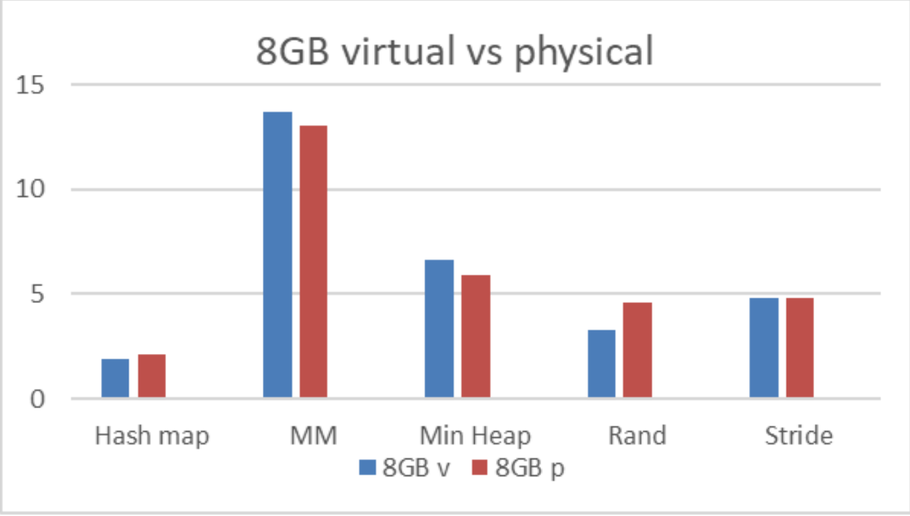
縱軸單位為 μs

較無法觀察出論文中說的「virtual過於樂觀」的情況（差距較小），反而在 Rand workload 出現 physical 好於 virtual 的情形。
推測可能是由於此 workload 為 32x32 平行度，Rand 可能發揮此特點使整體延遲下降



# 1 SYNTHETIC P VS V

**SUB-MS REQUEST RATIO**

所有數值單位為 %

我們觀察 synthetic workload 的 sub-ms request ratio，可發現

1. 較 Random 的程式 virtual 越有樂觀傾向
2. 即使是 real world (bert modle) 的對比，virtual 的樂觀程度也不會太高 (5%)

delta 是 Matrix Multiplication

| | delta_p | delta_v | Diff |
|---|---|---|---|
| 0.5GB | 47.06 | 46.96 | -0.1 |
| 2GB | 63.7 | 63.7 | 0 |
| 8GB | 63.7 | 63.7 | 0 |

| | hash_p | hash_v | Diff |
|---|---|---|---|
| 0.5GB | 79.31 | 80.16 | 0.85 |
| 2GB | 89.28 | 90.66 | 1.38 |
| 8GB | 89.28 | 90.66 | 1.38 |

| | heap_p | heap_v | Diff |
|---|---|---|---|
| 0.5GB | 78.04 | 77.82 | -0.22 |
| 2GB | 78.04 | 77.82 | -0.22 |
| 8GB | 78.04 | 77.82 | -0.22 |

| | rand_p | rand_v | Diff |
|---|---|---|---|
| 0.5GB | 23.5 | 28.45 | 4.95 |
| 2GB | 67.58 | 82.27 | 14.69 |
| 8GB | 90.11 | 93.19 | 3.08 |

| | stride_p | stride_v | Diff |
|---|---|---|---|
| 0.5GB | 51.84 | 49.98 | -1.86 |
| 2GB | 63.57 | 62.81 | -0.76 |
| 8GB | 86.3 | 89.52 | 3.22 |

| | bert_p | bert_v | |
|---|---|---|---|
| realworld | 86.39% | 91.76% | 5% |
| 0.1GB | | | |

bert 為 realworld workload 在此做為參考

=> 論文可能過於強調 physical address 對實驗的重要性（論文中提到 virtual address 有多達20%的樂觀傾向）

# 2 REALWORLD CACHE POLICY

## WHAT AND WHY?

由於論文中使用的 realworld workload 之 DRAM size 以及平行度都較低，我們好奇為何會與測試 synthetic workload 之間有此差異，因此決定以 synthetic workload 之 configuration 來模擬 realworld model 的執行。

# 2 REALWORLD CACHE POLICY

## CONFIGURATION 設定

使用github中給予的example synthetic workload，實驗只更改 cache policy 之種類

- 此外不使用 prefetcher 不使用 prefetcher ，並使用 8GB DRAM

config.txt

```
DRAM_mode 0
Has_cache 1
DRAM_size 8589934592
Mix_mode 1
Cache_portion_percentage 100
Has_mshr 1
Cache_placement 2097152
Cache_policy FIFO
Prefetcher No
Total_number_of_requests 20000000
```

## WORKLOAD 使用

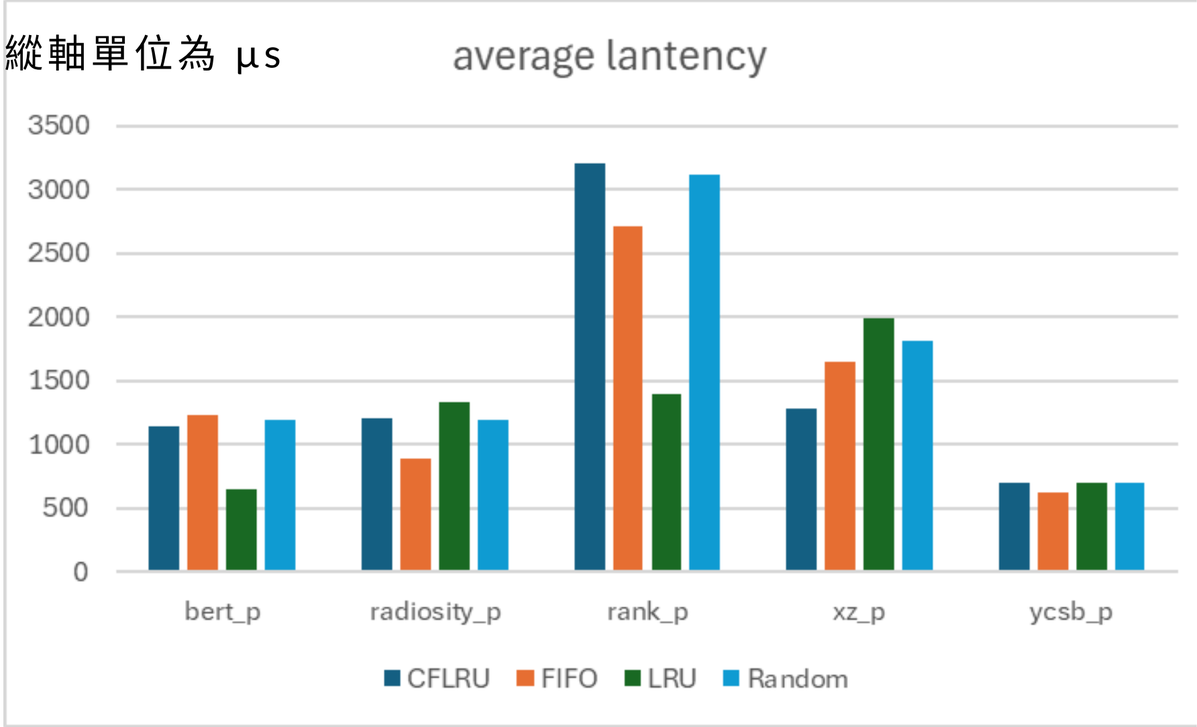
使用作者提供的 trace file 進行實驗，包含以下的 realworld workload：

bert_p
radiosity_p
rank_p
xz_p
ycsb_p

# 2 REALWORLD CACHE POLICY

## AVERAGE ACCESS LATENCY

| Benchmark | CFLRU | FIFO | LRU | Random |
|---|---|---|---|---|
| bert_p | 1138.83 | 1232.49 | 656.08 | 1190.90 |
| radiosity_p | 1202.68 | 887.42 | 1333.15 | 1200.35 |
| rank_p | 3202.00 | 2707.44 | 1400.23 | 3110.46 |
| xz_p | 1284.33 | 1644.33 | 1986.49 | 1812.05 |
| ycsb_p | 696.39 | 625.10 | 698.76 | 701.18 |

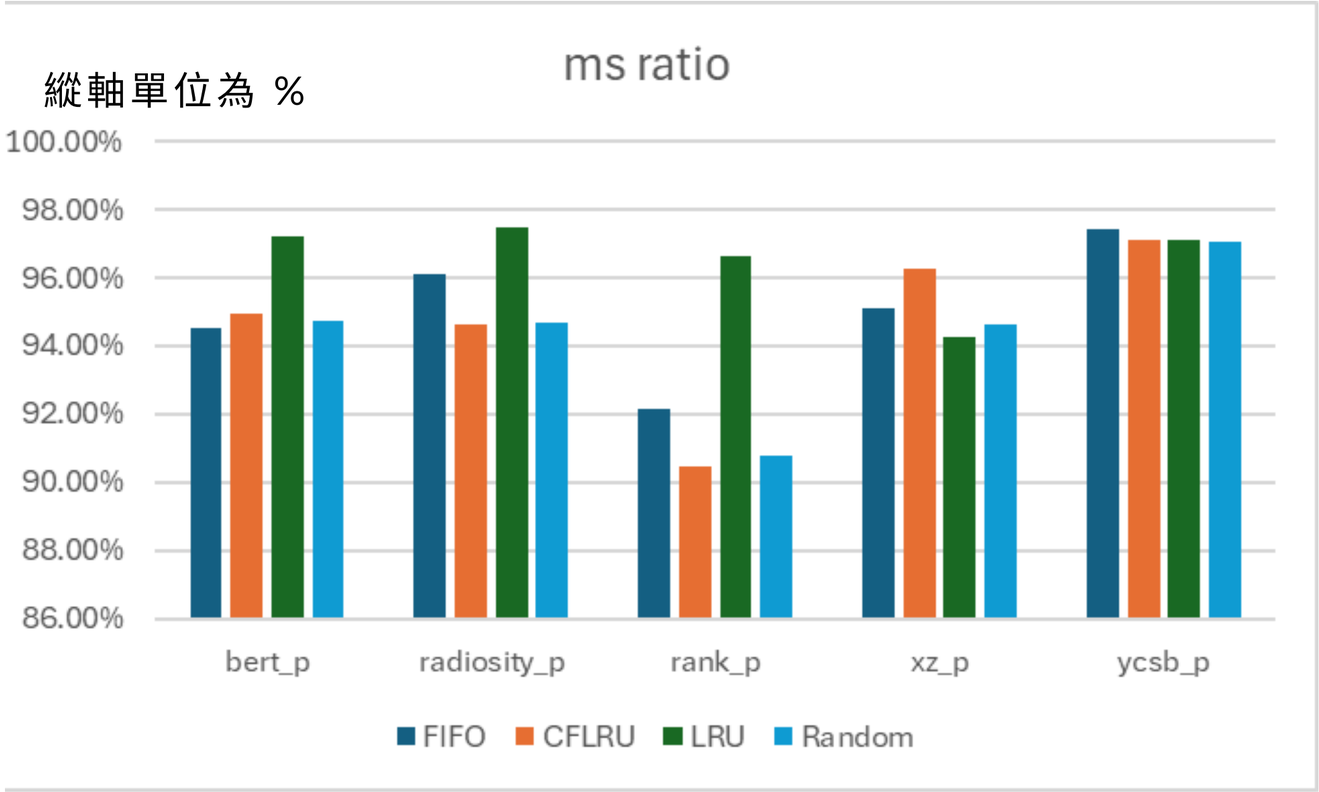



bert 與 rank 性質應該相近（矩陣運算），為何兩者延遲差異大?
推測可能是 bert 主要在訓練參數，因此 locality 較高，使其整題延遲低

# 2 REALWORLD CACHE POLICY

## MS RATIO

| | bert_p | radiosity_p | rank_p | xz_p | ycsb_p |
|---|---|---|---|---|---|
| Random | 94.71% | 94.66% | 90.77% | 94.63% | 97.08% |
| FIFO | 94.51% | 96.11% | 92.15% | 95.11% | 97.42% |
| LRU | 97.19% | 97.50% | 96.62% | 94.24% | 97.09% |
| CFLRU | 94.95% | 94.65% | 90.46% | 96.24% | 97.10% |




可以見到不管是 average latency 或 ms ratio ，圖片中的趨勢都很不明顯，且差異很小，這可能就是為何論文中採用較低的 DRAM Size 與 較低的平行度，才能更明顯的看出 policy 對性能的影響

# 08
# REVIEW

在最後的小節我們將簡單的回顧此實驗的內容與流程

# REVIEW

### 成功執行論文提供之CXL DESIGN TOOL

我們成功的執行作者所開發的 MQSim CXL，
並研究其參數設定進行實驗

### 論文實驗部分復現成功

我們嘗試了一些論文中的實驗，雖其結果與預期
（論文中）有不同之處，但整體來說還算合理，
還需要更進一步的調整參數進行更多實驗測試

### 自行規劃並進行實驗

最後我們結合課堂中所學，嘗試了一些簡單的實驗，
在模擬結果中實際比較與課程中的所學是否一致
比較可惜的是

# THANK YOU

Group 2
113522024 游振彤 113522011 胡嘉佑