



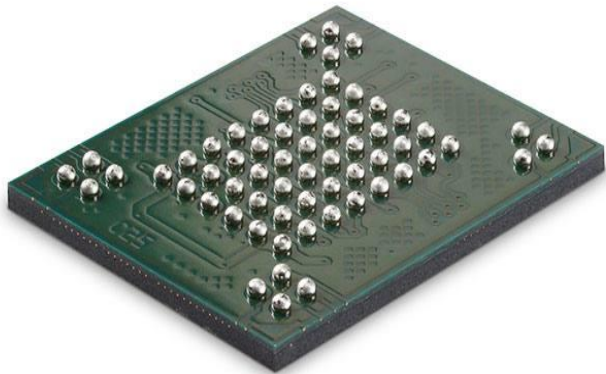
Maximizing Bandwidth Management FTL Based on Read and Write Asymmetry of Flash Memory

Bihan Li¹, Wei Tong, Jingning Liu¹, Dan Feng¹, Yazhi Feng¹, Junqing Qin¹, Peihao Li¹ and Bo Liu²

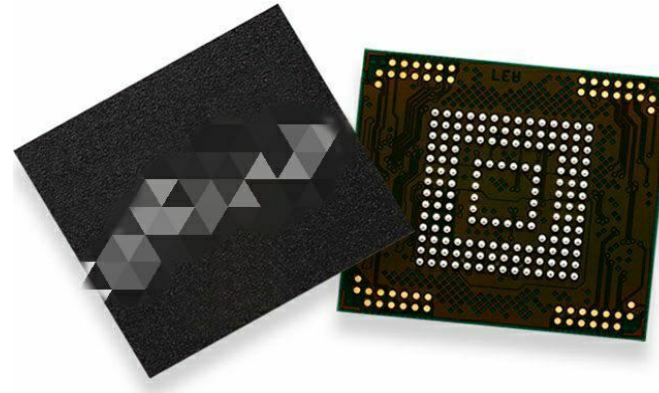
1. Huazhong University of Science and Technology

2. Hikstor Technology Co., LTD

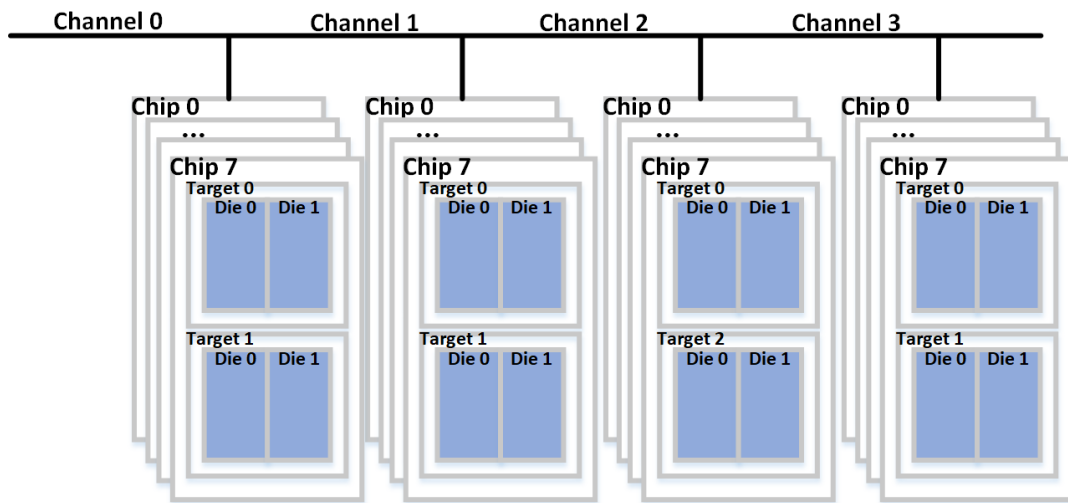
Chip Trends



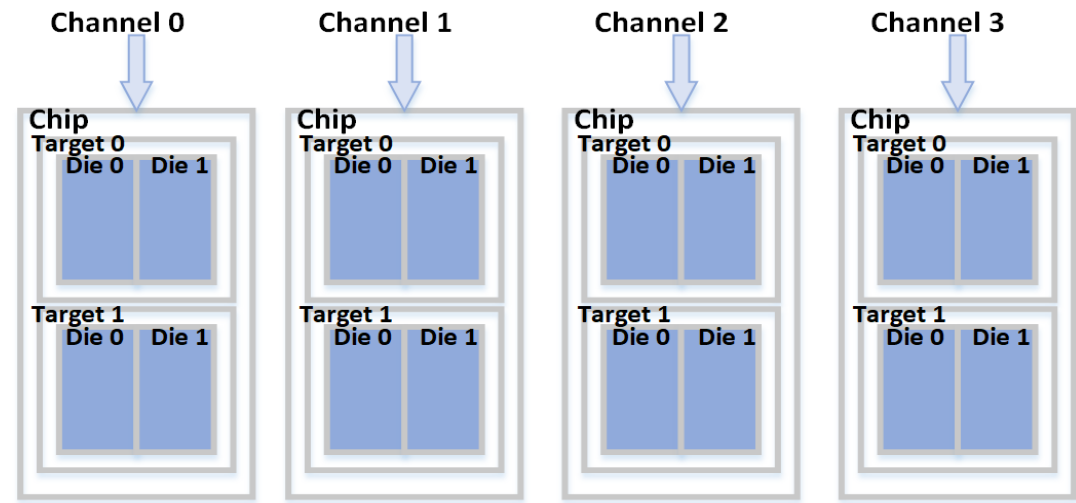
a 2D flash chip containing four 64 Gb dies



a 3D flash chip containing four 512Gb dies

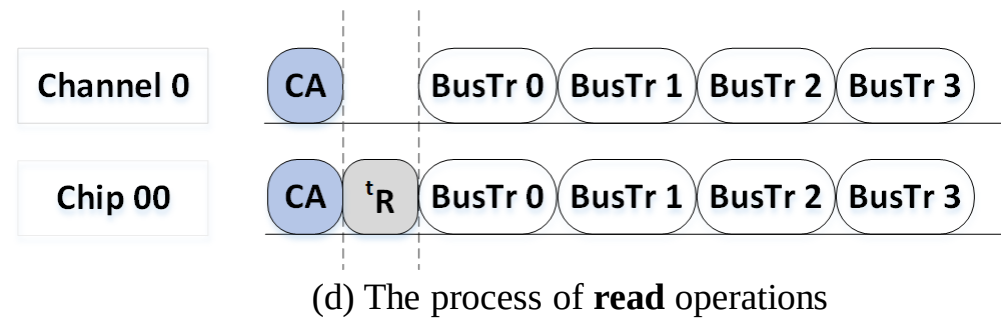
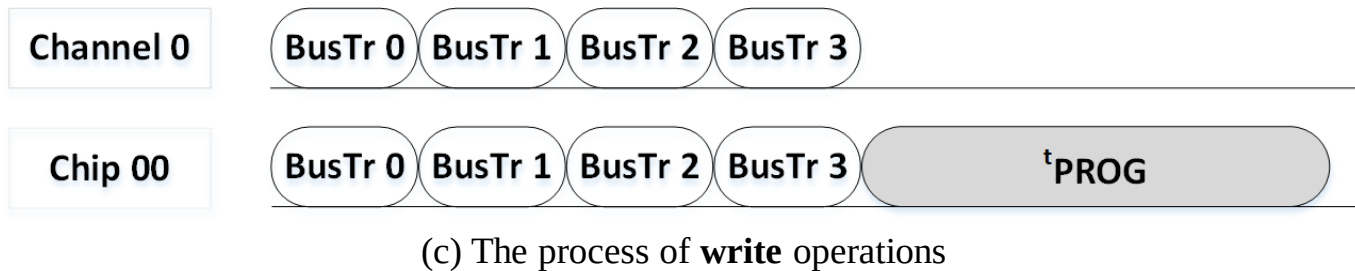
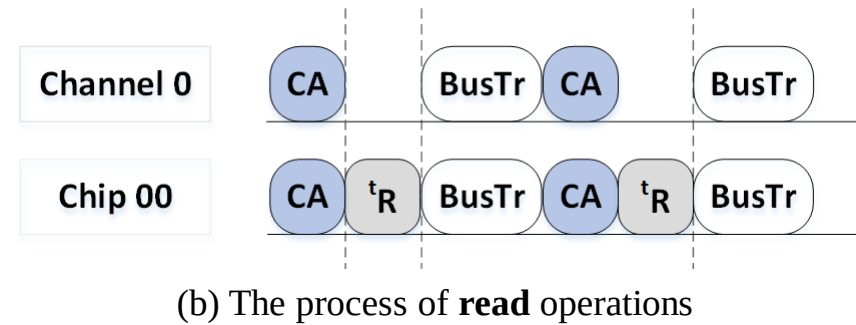
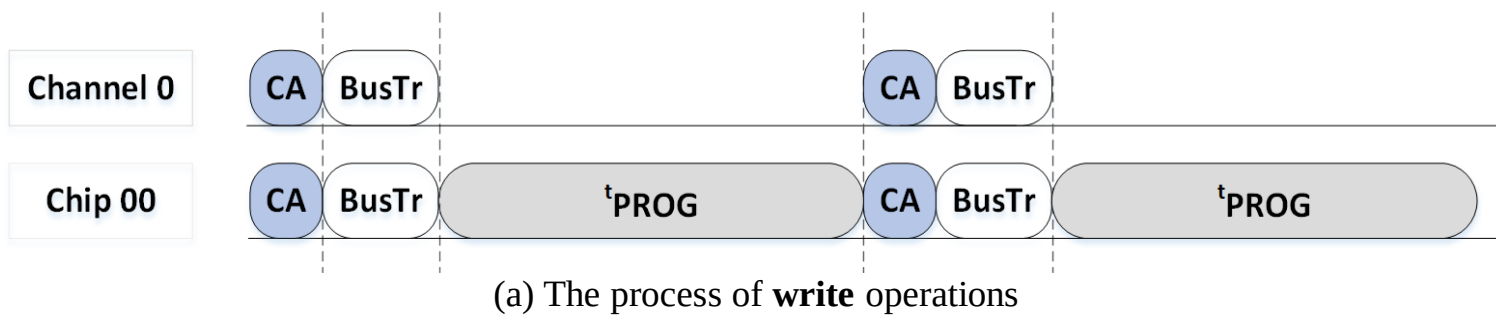


(a) a 1TB flash device using 2D flash chips require 32 chips



(b) a 1TB flash device using 3D flash chips require 4 chips

FTL Processing Requests

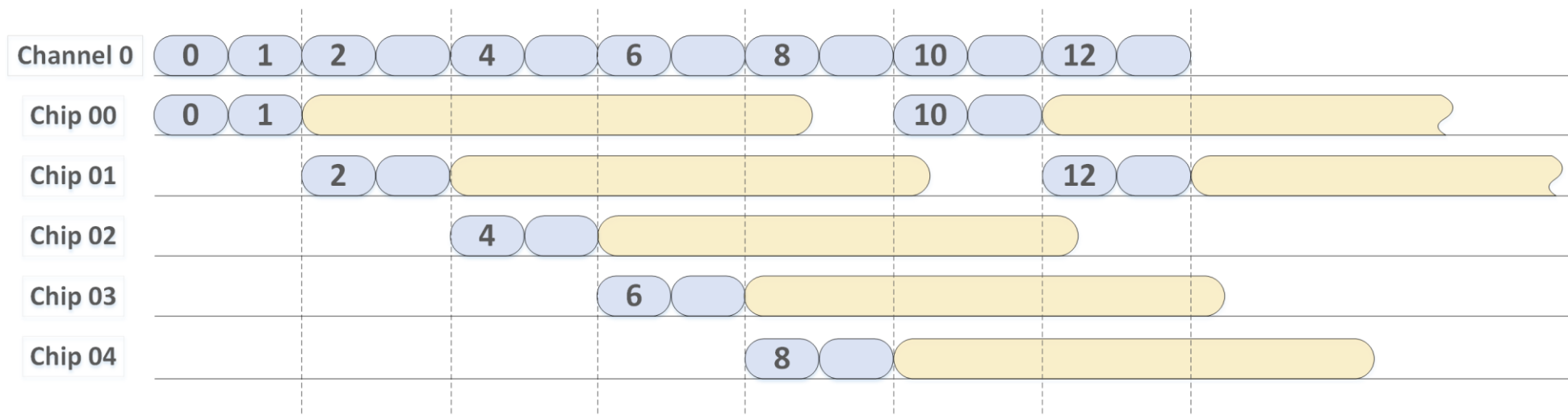


Three Different Cases for Write Operation

Case 1:

- ① each channel contains **five** flash chips
- ② available parallelism degree is **two** now

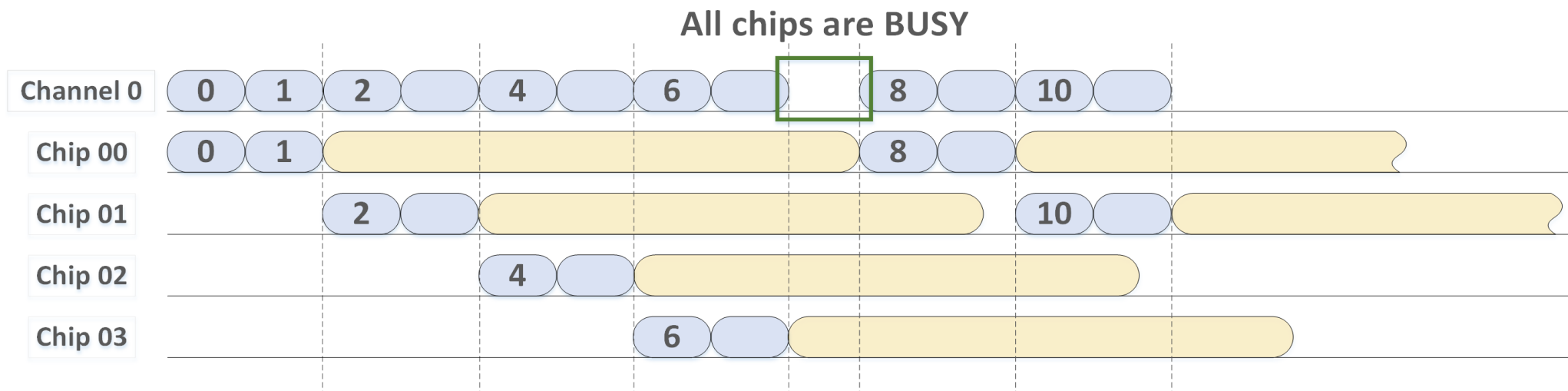
0 : Data transfer time of subrequest 0 : Flash array program time



Three Different Cases for Write Operation

Case 2:

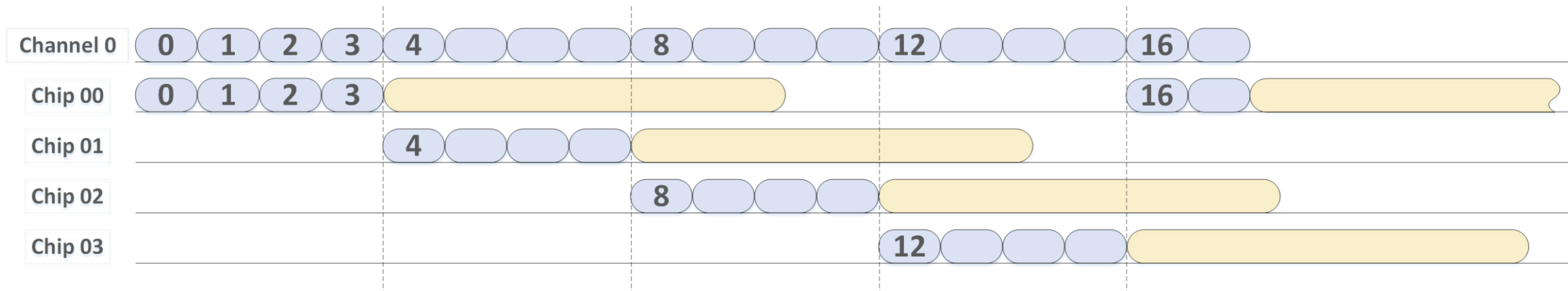
- ① each channel contains **four** flash chips
- ② available parallelism degree is **two** now



Three Different Cases for Write Operation

Case 3:

- ① each channel contains **four** flash chips
- ② available parallelism degree is **four** now

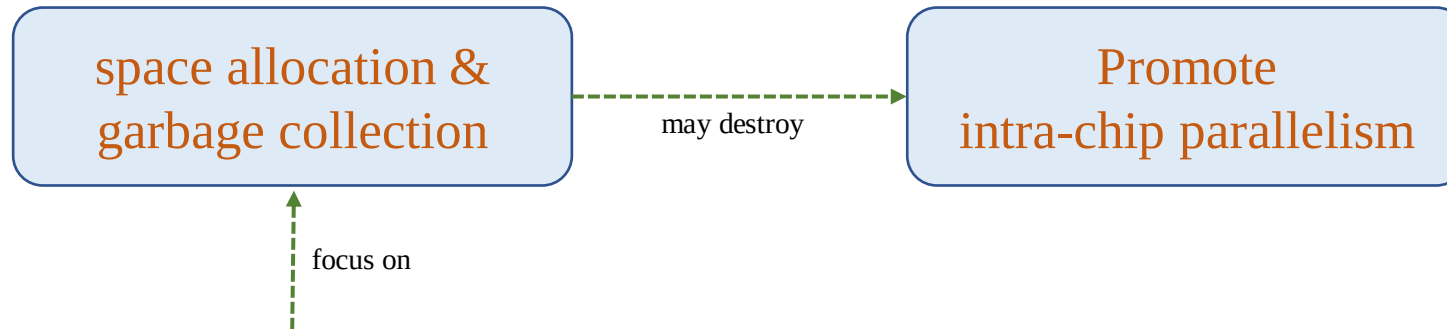


Three Different Cases for Write Operation



Three specific examples indicate:

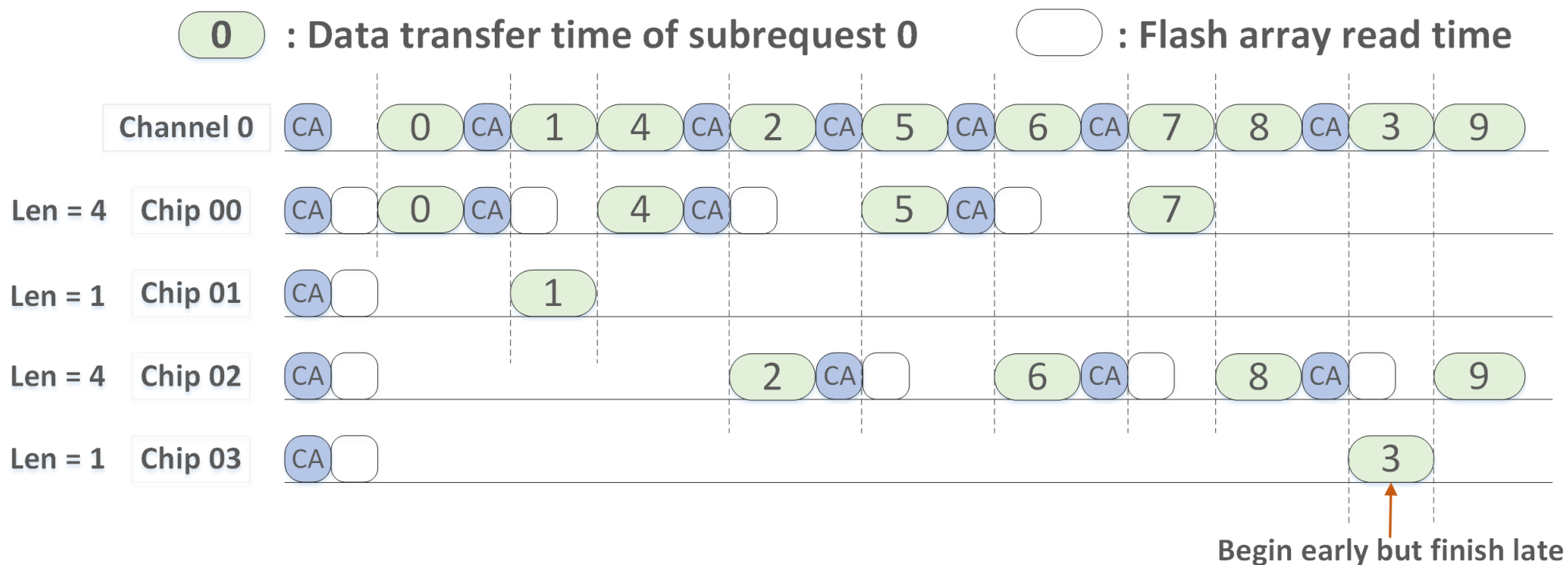
- ① to provide better performance when executing write operations, **promoting utilization of intra-chip parallelism** is very important. It can compensate for the negative effect of the chip number reduction on channel bandwidth.
- ② However, **space allocation and garbage collection** may destroy plane-level parallelism



Two Different Cases for Read Operation

Case 4:

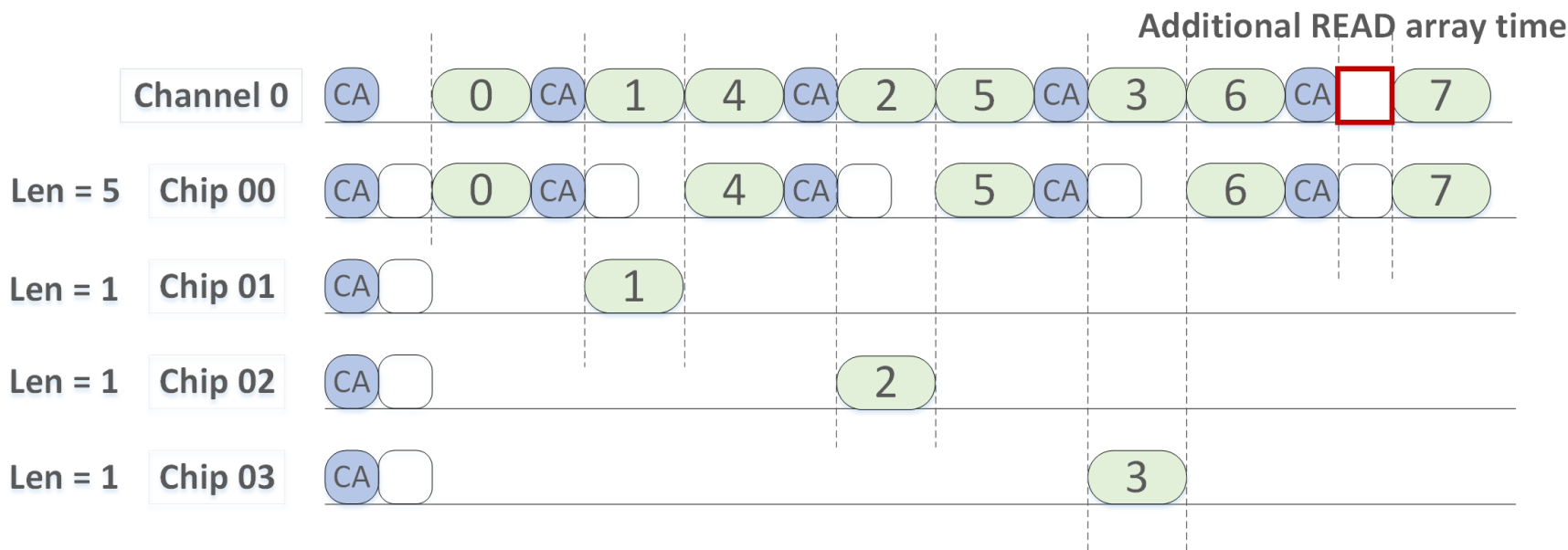
- ① one channel employs **four** chips
- ② read subrequests **do not satisfy parallel read constraints**
- ③ **four, one, four, and one** subrequest locating in four chips respectively



Two Different Cases for Read Operation

Case 5:

① subrequests distributed in one chip are greater than the total subrequests distributed in other chips



For read operations, **providing response time guarantee** is more important and Moreover, a **uniform distribution of data** is crucial. It helps eliminate the worst response time.

Existing approaches?



① Sprinkler

☑ schedules I/O requests based on internal resource layout rather than the order imposed by the device level queue

☒ neglects the strict restriction of **plane-level parallelism**, request execution is not efficient

② TBM

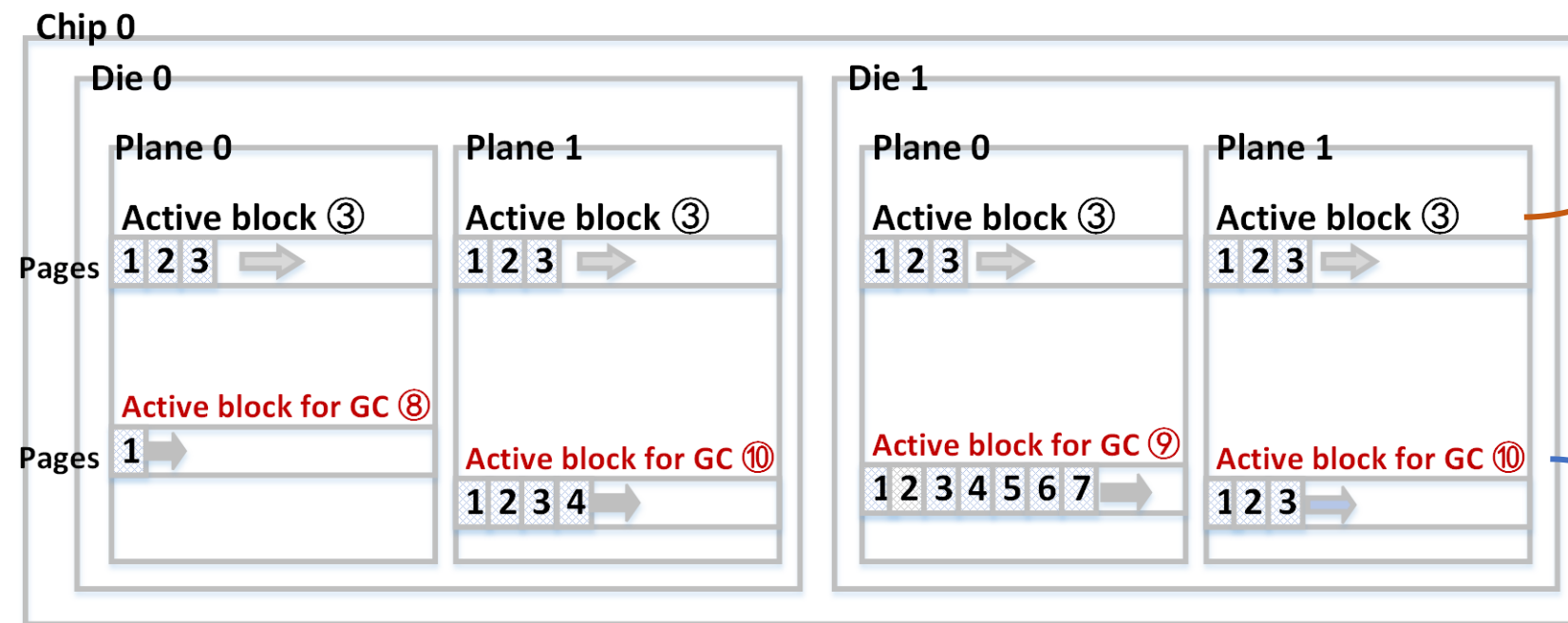
☑ symmetrically conducts usage and recycling of the flash block addresses on the planes of a die

☒ executing requests **in parallel unconditionally** sacrifices response time especially for read

MBM's Goals:

Maximize Bandwidth!
Don't sacrifice response time!

WS: extends an active block for GC



last written page in the active block can **keep the same** by using a **round robin assignment strategy** for page allocation

When executing GC to reclaim space, **valid data** is migrated to a **dedicated active block instead of the block being written**

WS: an algorithm for Channel_Max_Subs()

Algorithm 1 Channel_Max_Subs().

Input:

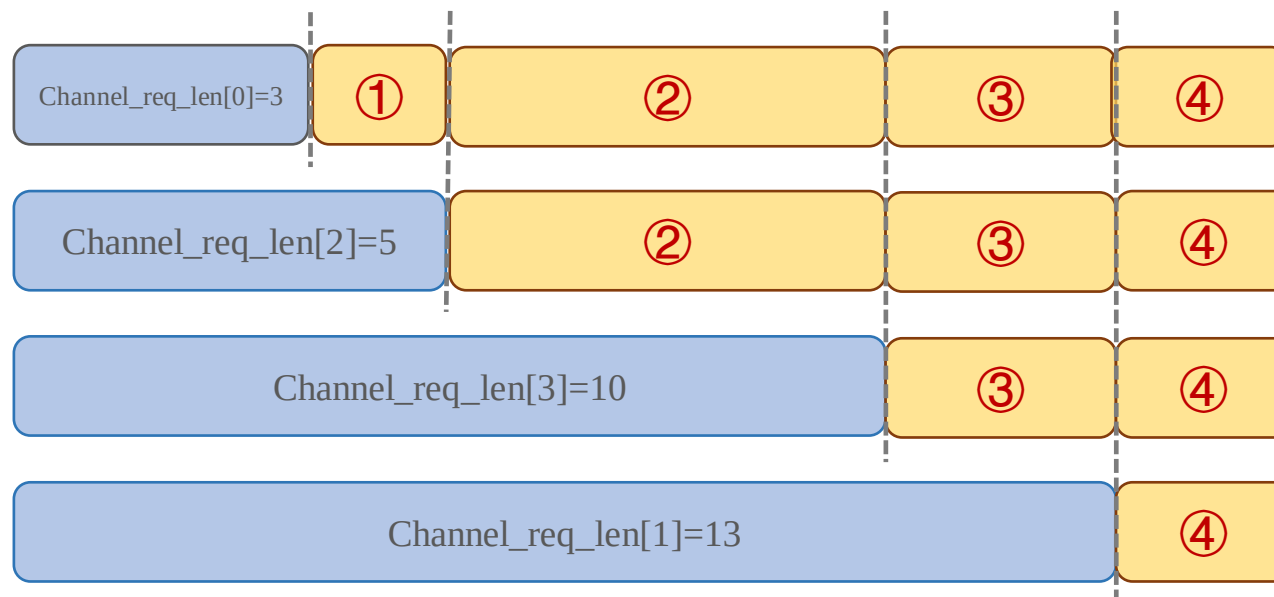
Read requests queue length of the channel, *channel_num[].read_len*;
 Number of write subrequests that the channel has executed, *channel_num[].execute_w_subs*;
 Number of channel, *channel_number*;
 Requests number of SSD write requests queue, *w_req*;

Output:

Subrequests number that channel can execute, *channel_max_subs*;

```

1: for  $i = 0$  to  $channel\_number$  do
2:   Count the total number of subrequests per channel,
      $channel\_req\_len[i] = channel\_num[i].read\_len + channel\_num[i].execute\_w\_subs$ ;
3: end for
4: Sort  $channel\_req\_len[channel\_number]$  from small to large;
5: for  $i = 0$  to  $channel\_number$  do
6:   if SSD number of write subrequests < subrequests number of Channel[i]- subrequests number of Channel[i]
     then
7:      $channel\_max\_subs = (w\_req \% i) ? (w\_req / i + 1 + channel\_req\_len[i-1]) : (w\_req / i + channel\_req\_len[i-1])$ ;
8:     set the number of write requests to 0,  $w\_req = 0$ ;
9:     break;
10:  else
11:    update the number of write subrequests,  $w\_req = (channel\_req\_len[i] - channel\_req\_len[i-1]) * i$ ;
12:  end if
13: end for
14: if subrequests number of write requests queue  $\neq 0$  then
15:   average the number of remaining write subrequests to each channel and update the  $channel\_max\_subs$ ;
16: end if
    
```



① count the total number of subrequests in each channel (Line 1-3) and sort channels from smallest to the largest total number (Line 4)

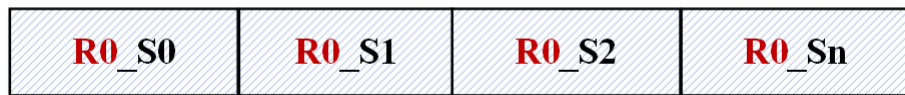
① Gradually allocate subrequests to the channel according to the order
 If subrequests allocation complete, set the Channel_Max_Subs for every channel (Line 5-9).

② If not, modify the number of remaining subrequests and continue the above steps (Line 10-12)

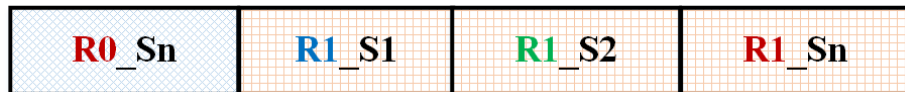
③ When all Channel_Max_Subs reach the same, but there are still some subrequests in the queue, we distribute them into all channels on average and update the Channel_Max_Subs (Line 14-16).

RS: executing read in parallel conditionally

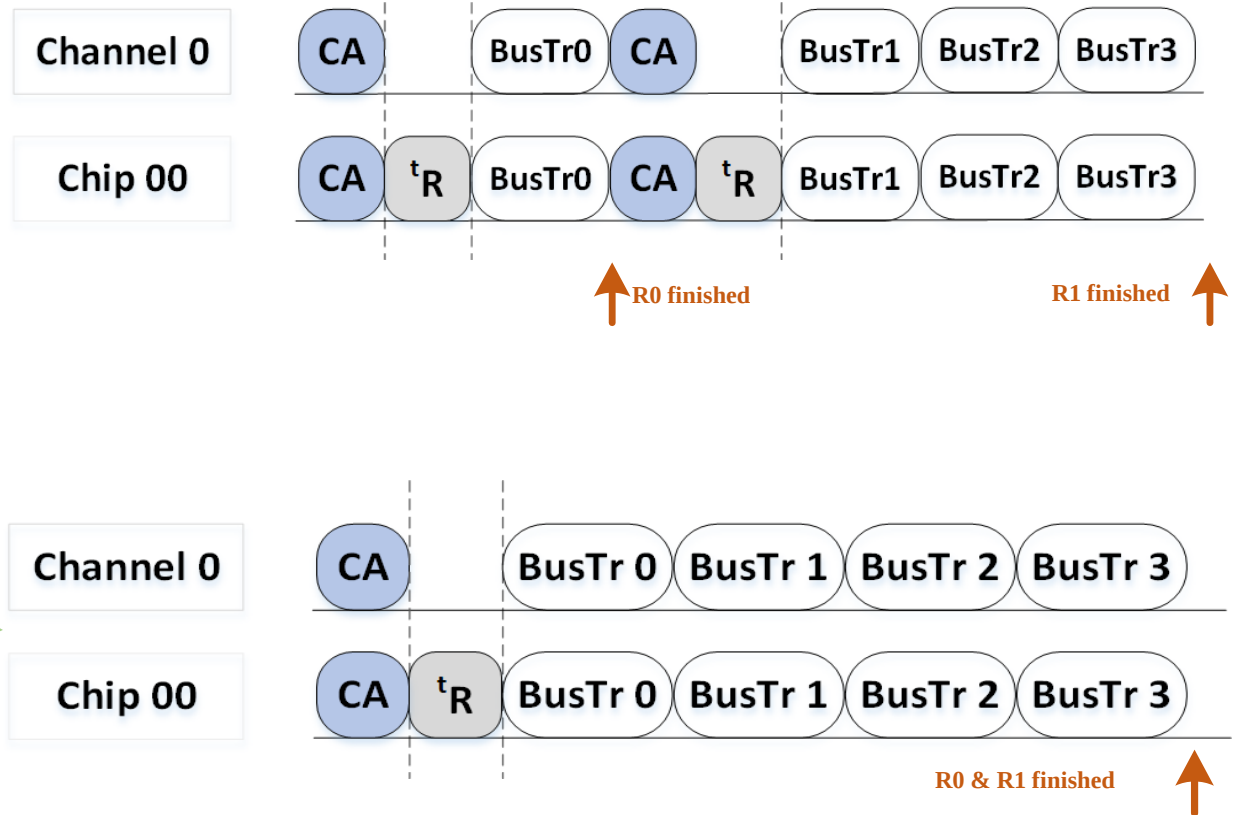
Rn: request_n Sn: subrequest_n



(a) An example of parallel read.



(b) An example of responding to first coming subrequest.



The remaining Problem:

Problem: subrequest distribution is extremely unbalanced.

We propose a more strict data layout for the read-intensive area.

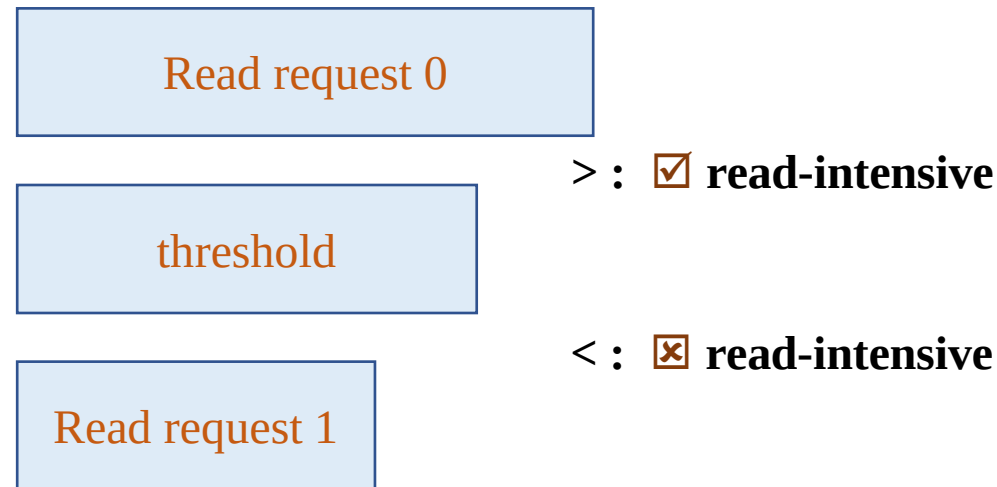
A Read intensive area is judged by space locality.

Specifically, we set a threshold of a read request length.

When a read request **length > threshold: read-intensive;**

a read request length < threshold: not read-intensive

If write requests are allocated to a read-intensive area, we will distribute them into all corresponding dies of the area averagely.

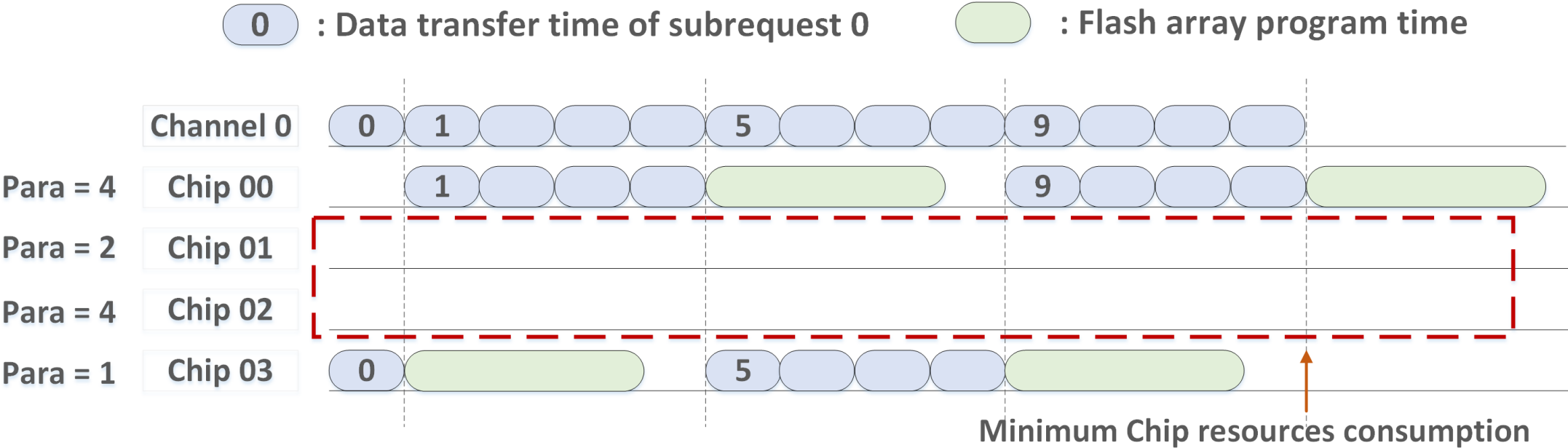


MCS of MBM

When allocating pages for write requests, MCS

- ① **first selects an idle chip with the least parallelism degree.** When the chip completes the current task, it will restore the maximum intra-chip parallelism and execute more subrequests next time.
- ② **Then MCS selects a free chip with the maximal parallelism degree** so that more data transfer time can cover the busy time of the working chip.

Overall, the SSD can have more free chips for subsequent requests.



Evaluation

- **Setup:**

Implementing MBM on SSDsim

four levels of parallelism are supported

TBM [15] and Sprinkler [14] as a comparison

The detailed parameters in Table I

TABLE I
FLASH PARAMETERS FOR EXPERIMENT

Parameter	Value
Page Size	8KB
Page Number/Block	1024
Channel Number	4
Chip Number/Channel	2
Die Number/Chip	2
Plane Number/Die	2
Data Transfer Rate	10ns/8bit
Erase Time	15ms
Flash Array Program Time	1,300,000ns
Flash Array Read Time	78,000ns
Command & Address Transfer Time	7*10ns
Data Transfer Mode	Synchronous mode

- **Trace:**

6 representative real workloads presented SNIA and MSR Cambridge [20] characteristics of traces are given in Table II

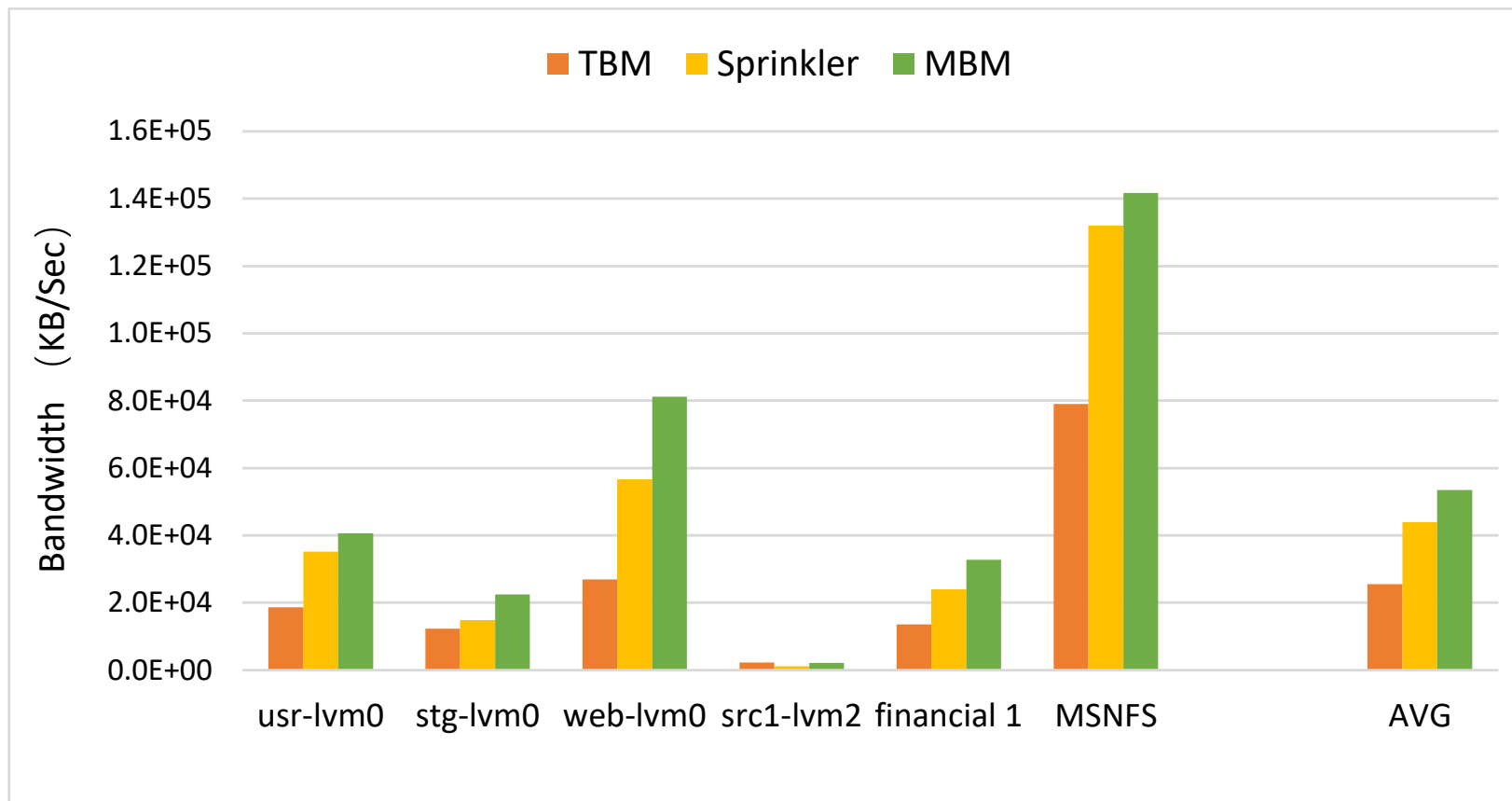
TABLE II
WORKLOADS ANALYSIS

Workloads	Avg.Size (Read/Write) (KB)	Read/Write Ratio(%) (KB)	Annotation
usr-lvm0	40.91/10.28	40.42/59.58	User home directories
stg-lvm0	24.92/9.19	15.19/84.81	Webstaging
web-lvm0	29.99/8.59	29.88/70.12	Web/SQLserver
src1-lvm2	19.11/32.50	25.37/74.63	Source contrl
Financial	2.38/3.85	21.96/78.04	Financial institution
MSNFS	14.88/10.60	79.57/20.43	MSN File System

MBM increases the overall bandwidth

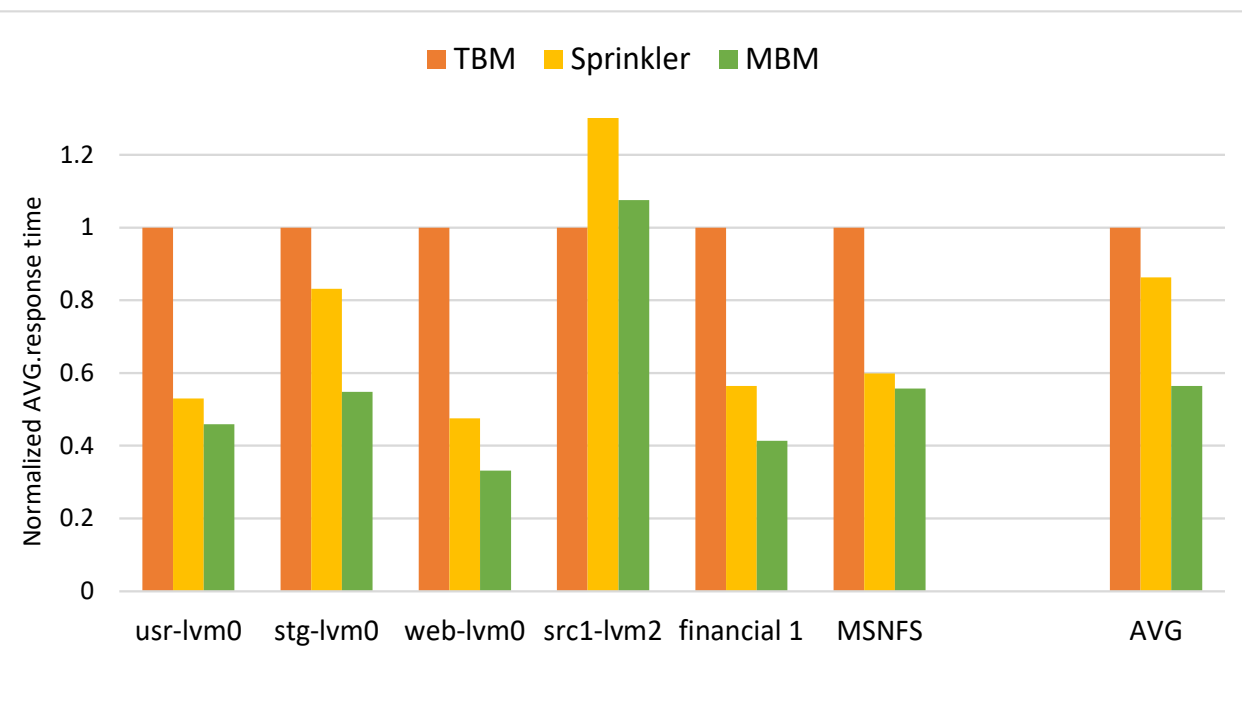
by up to 3x

by 2.1x averagely

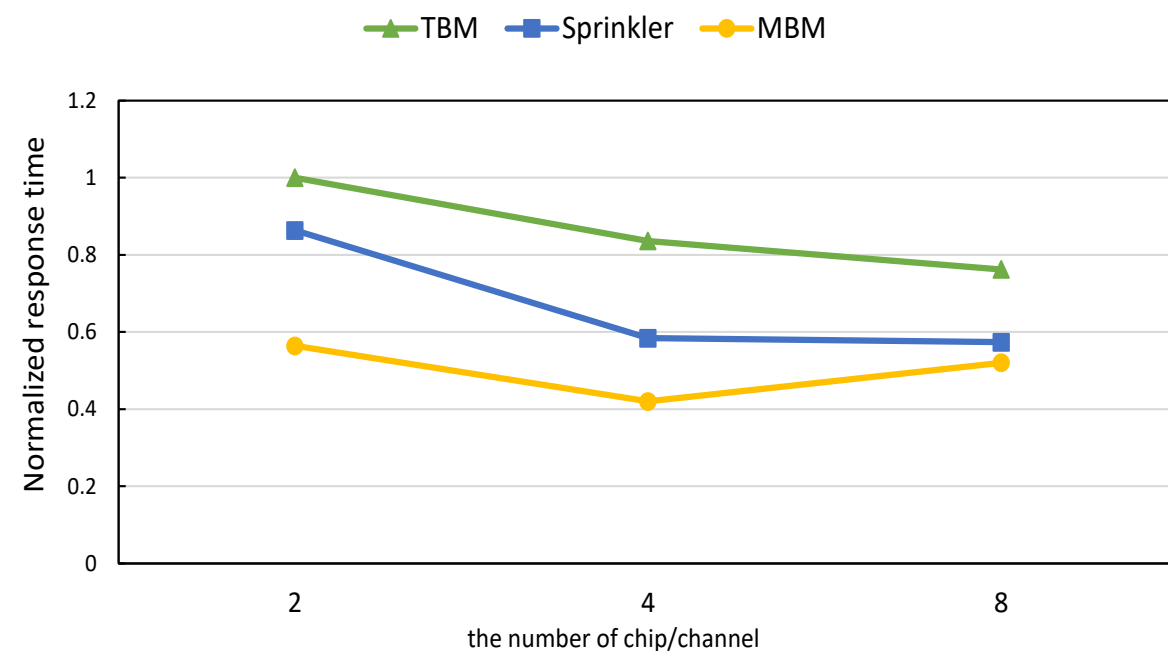


Response Time

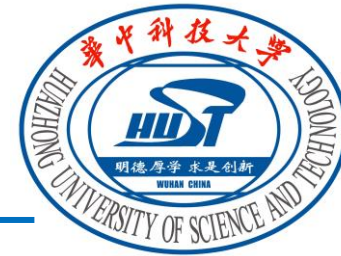
MBM reduces the average response time of the baseline by up to 66.8%
an average of 43.6%.



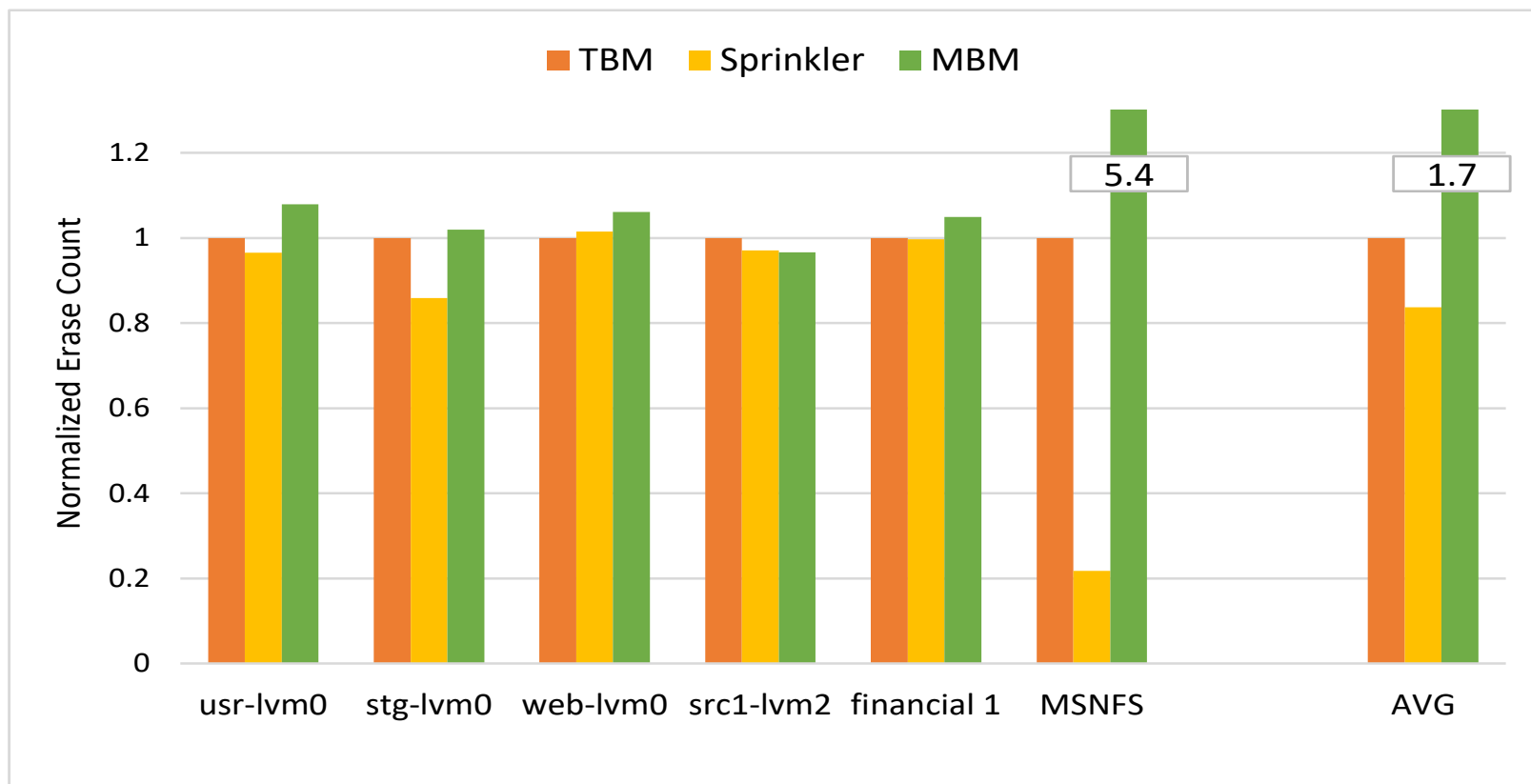
- MBM reduces the response time of the TBM by 43.6% and 31.6% when two and eight chips in a channel
- with chip number decreasing, MBM achieves almost the same performance when the chip number reduces



Erase Count



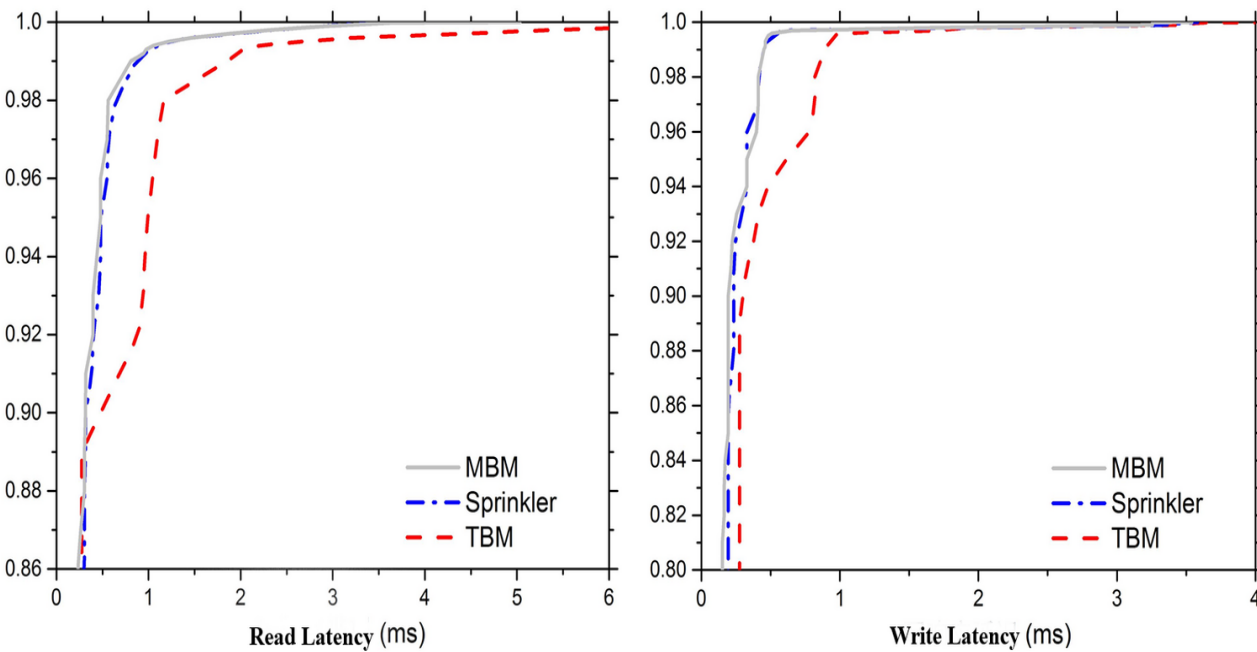
- ❑ the **erase count** of MBM **slightly increases**
- ❑ Under MSNFS, the normalized result shows a large difference
- ❑ As data volume increases such as the other five workloads, GC increment of MBM **averages 3.5% and the maximum is 7.9%.**



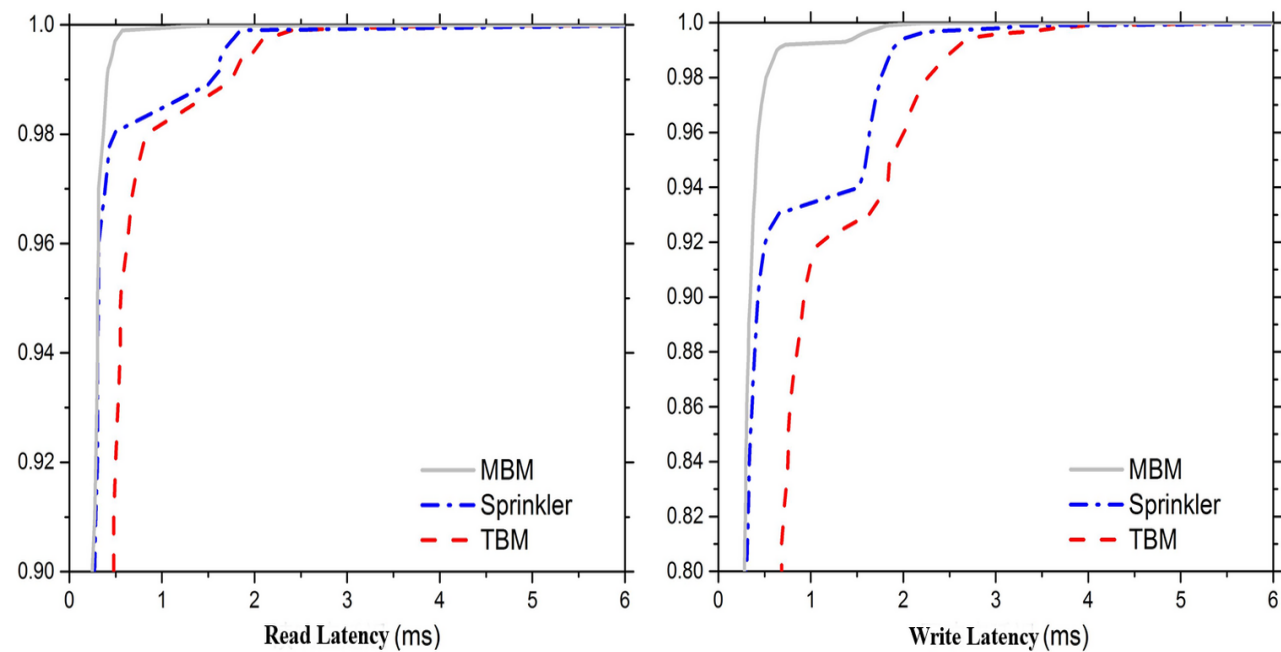
Tail Latency



at high percentiles
compared to Sprinkler and TBM
MBM consistently and significantly show better performance



(a) MSNFS read and write tail latency



(b) Financial read and write tail latency

Conclusion And Future Work



Conclusion:

for improving channel bandwidth and system performance, write and read operations need to focus on different problems

We propose a parallelism-enhanced Write Strategy, a parallelism-relaxed Read Strategy and a Minimizing Chip consumption Strategy

on average, the overall performance of MBM outperforms two state-of the-art optimization strategies.

Future Work:

The RS of MBM causes a little more erase counts, which are carefully controlled and evaluated.

Optimizing the read-intensive area determination method in RS can further improve parallelism and reduce average response time.

we will modify MBM to achieve better performance in the future work.



Thank you and questions?