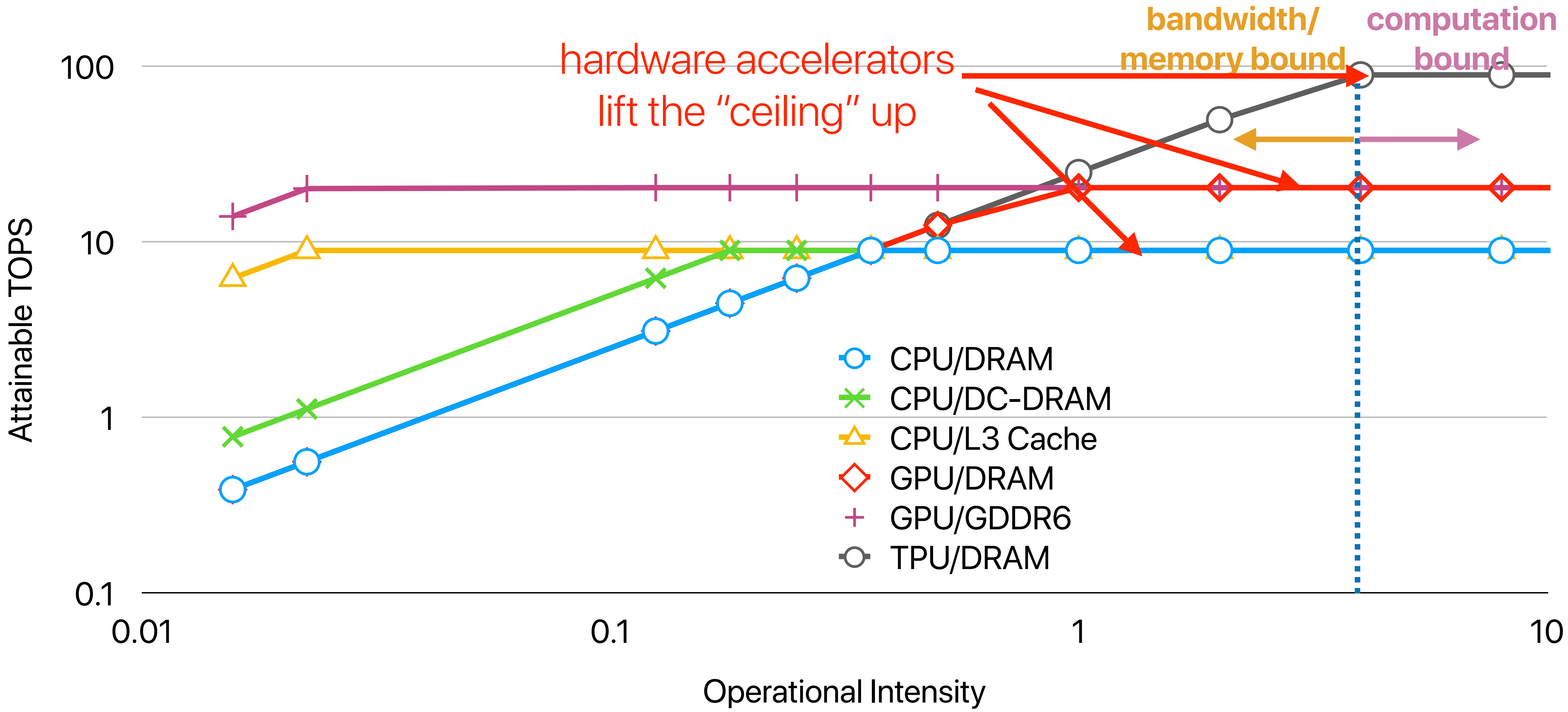
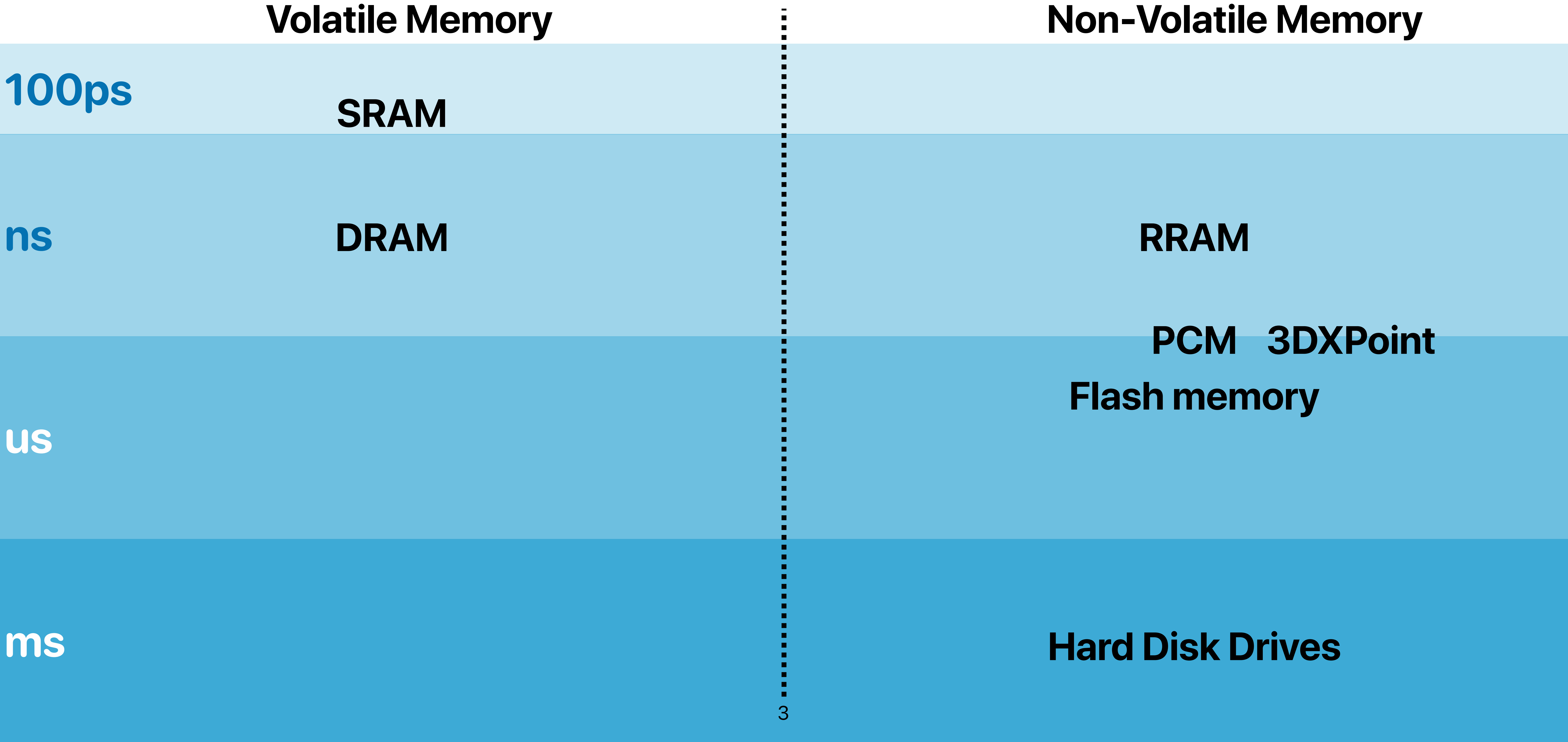
Modern Heterogeneous Computers: (4) Memory Components II

Hung-Wei Tseng

Recap: the roofline after using hardware accelerators



Memory technologies we have today



Volatile Memory

Non-Volatile Memory

100ps

SRAM

ns

DRAM

RRAM

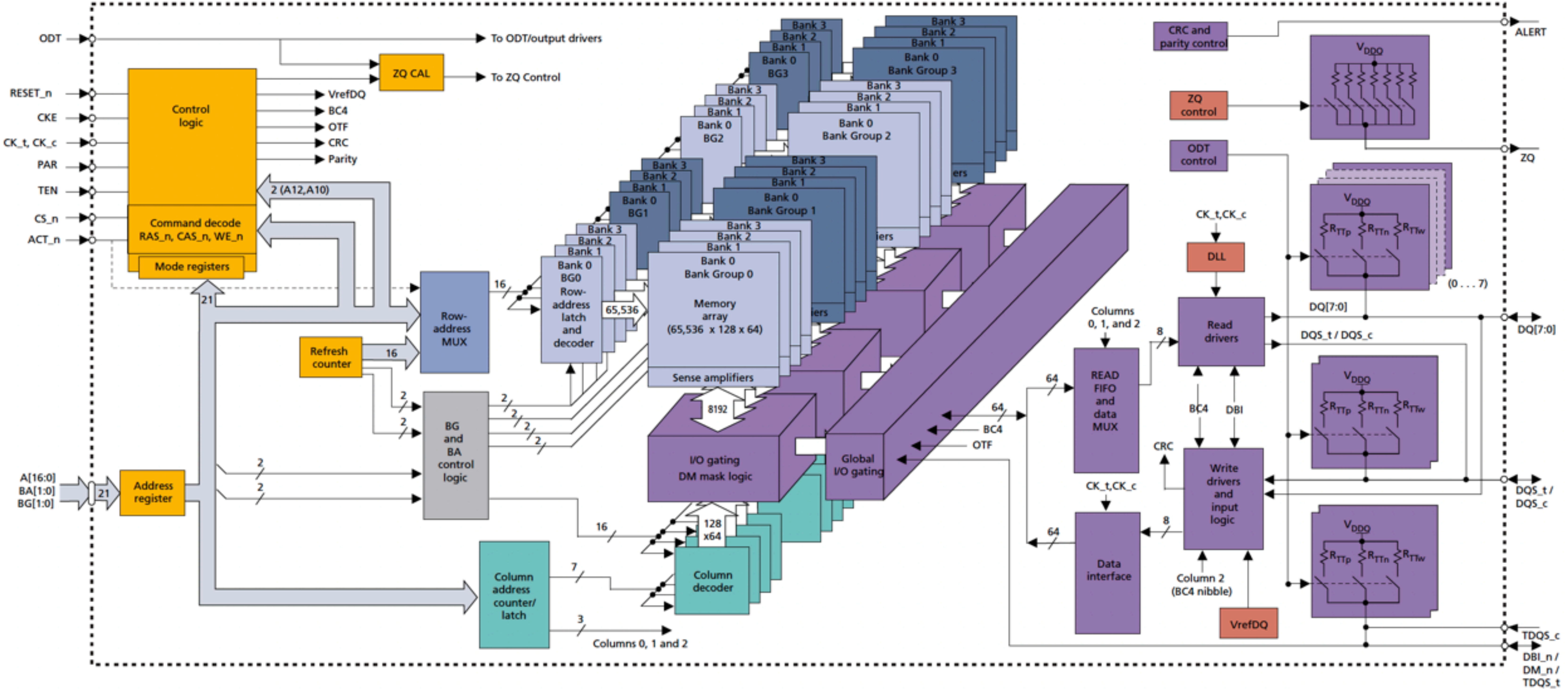
PCM 3DXPoint

Flash memory

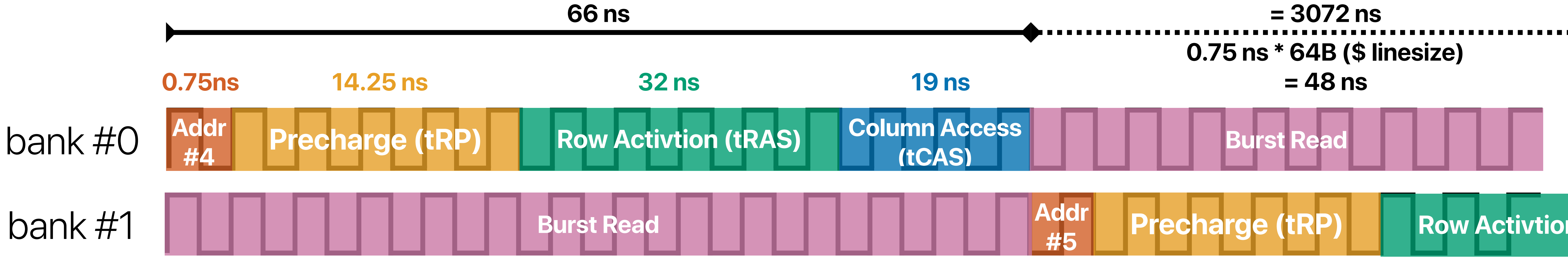
us

ms

Hard Disk Drives


ODT
RESET_n
CKE
CK_t, CK_c
PAR
TEN
CS_n
ACT_n
A[16:0]
BA[1:0]
BG[1:0]
21
To ODT/output drivers
ZQ CAL
To ZQ Control
Control logic
VrefDQ
BC4
OTF
CRC
Parity
2 (A12,A10)
Command decode
RAS_n, CAS_n, WE_n
Mode registers
21
Address register
Refresh counter
16
Row-address MUX
16
2
2
BG and BA control logic
2
2
2
2
2
2
16
Column address counter/ latch
7
3
Columns 0, 1 and 2
Bank 0 BG0 Row-address latch and decoder
Bank 1
Bank 2
Bank 3
65,536
Bank 0 Bank Group 0
Memory array (65,536 x 128 x 64)
Sense amplifiers
Bank 0 BG1
Bank 0 Bank Group 1
Bank 0 BG2
Bank 0 Bank Group 2
Bank 0 BG3
Bank 0 Bank Group 3
8192
I/O gating DM mask logic
128 x64
Column decoder
Global I/O gating
64
BC4
OTF
Columns 0, 1, and 2
64
READ FIFO and data MUX
64
Data interface
CK_t,CK_c
CRC and parity control
ZQ control
ODT control
CK_t,CK_c
DLL
8
Read drivers
BC4
DBI
CRC
Write drivers and input logic
8
Column 2 (BC4 nibble)
VrefDQ
V_{DDQ}
R_{TTp}
R_{TTn}
R_{TTw}
(0 . . . 7)
DQ[7:0]
DQS_t / DQS_c
ALERT
ZQ
DQ[7:0]
DQS_t / DQS_c
TDQS_c
DBI_n / DM_n / TDQS_t

Recap: DRAM Accesses

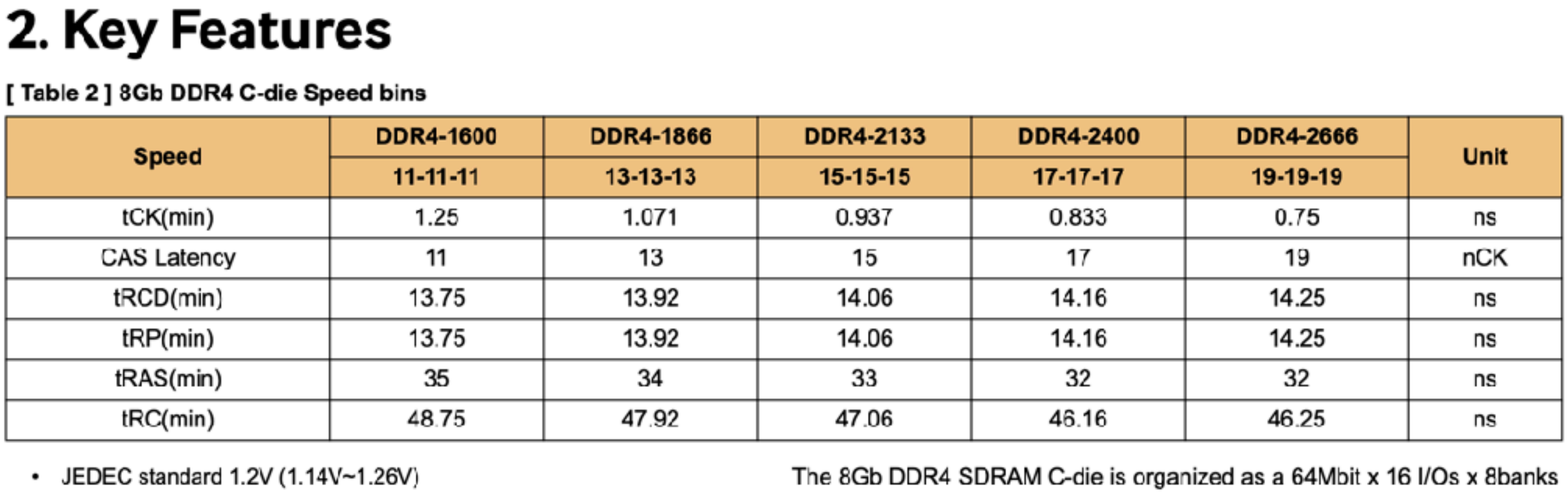


0.75 ns * 4K (if you have 4KB in a row)
= 3072 ns

66 ns

0.75 ns * 64B ($ linesize)
= 48 ns

The latency of pre-charge, row/column accesses is fully covered!



2. Key Features

[Table 2] 8Gb DDR4 C-die Speed bins

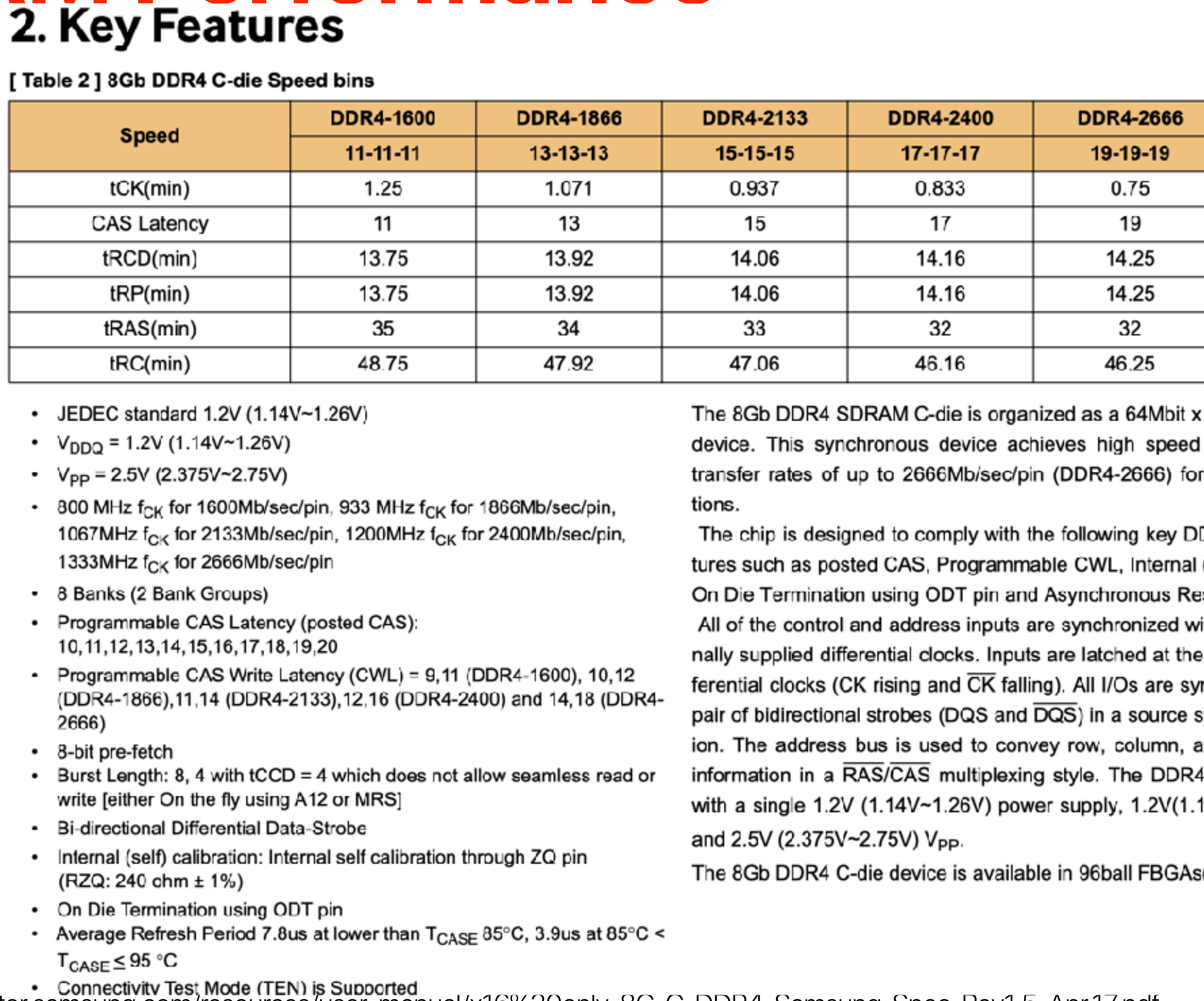
Speed	DDR4-1600	DDR4-1866	DDR4-2133	DDR4-2400	DDR4-2666	Unit
	11-11-11	13-13-13	15-15-15	17-17-17	19-19-19	
tCK(min)	1.25	1.071	0.937	0.833	0.75	ns
CAS Latency	11	13	15	17	19	nCK
tRCD(min)	13.75	13.92	14.06	14.16	14.25	ns
tRP(min)	13.75	13.92	14.06	14.16	14.25	ns
tRAS(min)	35	34	33	32	32	ns
tRC(min)	48.75	47.92	47.06	46.16	46.25	ns

- JEDEC standard 1.2V (1.14V~1.26V)

The 8Gb DDR4 SDRAM C-die is organized as a 64Mbit x 16 I/Os x 8banks

Recap: DRAM Performance

- Latency per "8-bit" — 0.75 ns (if it's row-buffered)
- Bandwidth per die = $$\frac{1}{0.75ns} = 1.33GB/sec$$
- 16 chips = $$16 \times \frac{1}{0.75ns} = 21.33GB/sec$$



2. Key Features

[Table 2] 8Gb DDR4 C-die Speed bins

Speed	DDR4-1600	DDR4-1866	DDR4-2133	DDR4-2400	DDR4-2666
	11-11-11	13-13-13	15-15-15	17-17-17	19-19-19
tCK(min)	1.25	1.071	0.937	0.833	0.75
CAS Latency	11	13	15	17	19
tRCD(min)	13.75	13.92	14.06	14.16	14.25
tRP(min)	13.75	13.92	14.06	14.16	14.25
tRAS(min)	35	34	33	32	32
tRC(min)	48.75	47.92	47.06	46.16	46.25

- JEDEC standard 1.2V (1.14V~1.26V)
- V_{DDQ} = 1.2V (1.14V~1.26V)
- V_{PP} = 2.5V (2.375V~2.75V)
- 800 MHz f_{CK} for 1600Mb/sec/pin, 933 MHz f_{CK} for 1866Mb/sec/pin, 1067MHz f_{CK} for 2133Mb/sec/pin, 1200MHz f_{CK} for 2400Mb/sec/pin, 1333MHz f_{CK} for 2666Mb/sec/pin
- 8 Banks (2 Bank Groups)
- Programmable CAS Latency (posted CAS): 10,11,12,13,14,15,16,17,18,19,20
- Programmable CAS Write Latency (CWL) = 9,11 (DDR4-1600), 10,12 (DDR4-1866),11,14 (DDR4-2133),12,16 (DDR4-2400) and 14,18 (DDR4-2666)
- 8-bit pre-fetch
- Burst Length: 8, 4 with tCCD = 4 which does not allow seamless read or write [either On the fly using A12 or MRS]
- Bi-directional Differential Data-Strobe
- Internal (self) calibration: Internal self calibration through ZQ pin (RZQ: 240 ohm ± 1%)
- On Die Termination using ODT pin
- Average Refresh Period 7.8us at lower than T_{CASE} 85°C, 3.9us at 85°C < $T_{CASE} \leq$ 95 °C
- Connectivity Test Mode (TEN) is Supported

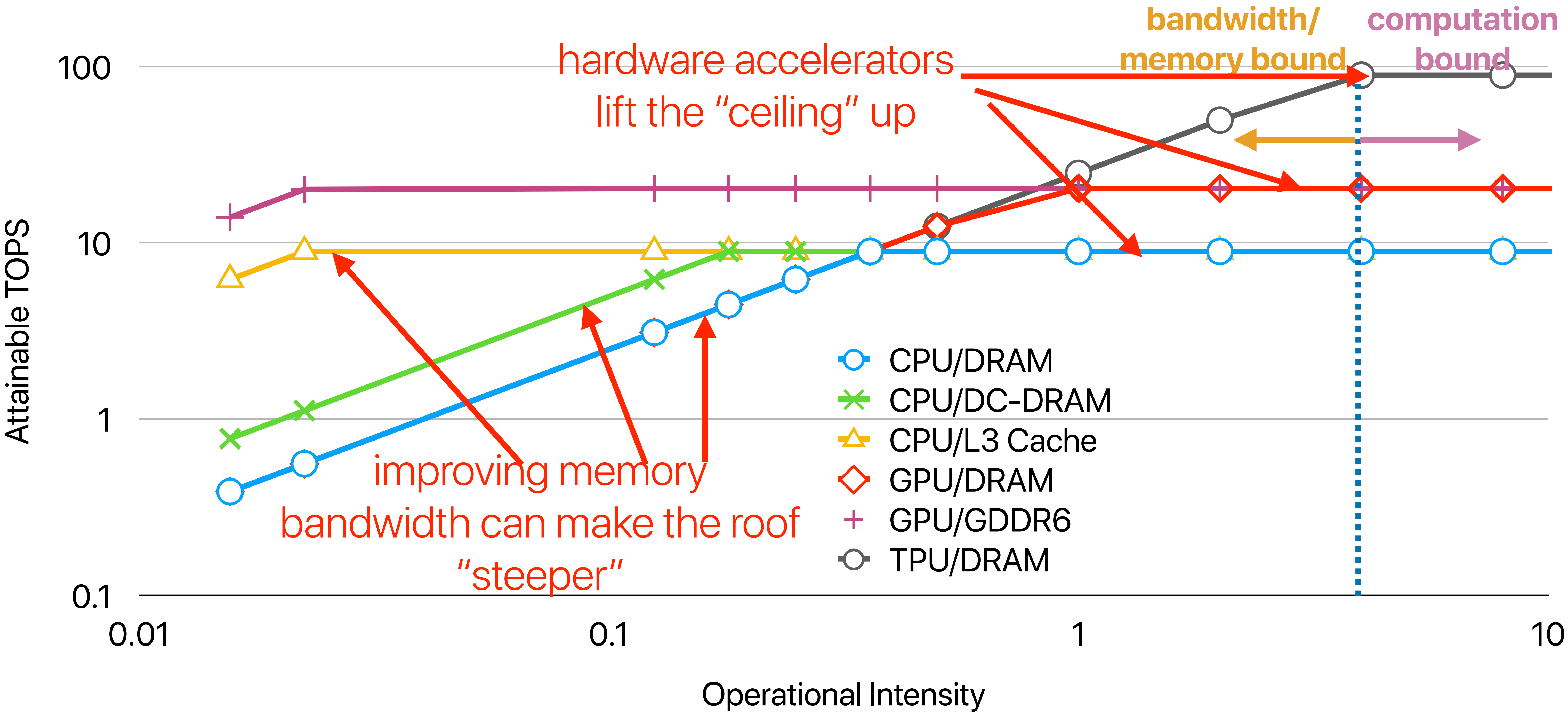
The 8Gb DDR4 SDRAM C-die is organized as a 64Mbit x
device. This synchronous device achieves high speed
transfer rates of up to 2666Mb/sec/pin (DDR4-2666) for
tions.

The chip is designed to comply with the following key DD
tures such as posted CAS, Programmable CWL, Internal
On Die Termination using ODT pin and Asynchronous Re
All of the control and address inputs are synchronized wi
nally supplied differential clocks. Inputs are latched at the
ferential clocks (CK rising and $\overline{CK}$ falling). All I/Os are sy
pair of bidirectional strobes (DQS and $\overline{DQS}$) in a source s
ion. The address bus is used to convey row, column, a
information in a $\overline{RAS}/\overline{CAS}$ multiplexing style. The DDR4
with a single 1.2V (1.14V~1.26V) power supply, 1.2V(1.1
and 2.5V (2.375V~2.75V) V_{PP}.

The 8Gb DDR4 C-die device is available in 96ball FBGAs

https://semiconductor.samsung.com/resources/user-manual/x16%20only_8G_C_DDR4_Samsung_Spec_Rev1.5_Apr.17.pdf

Recap: the roofline after using hardware accelerators



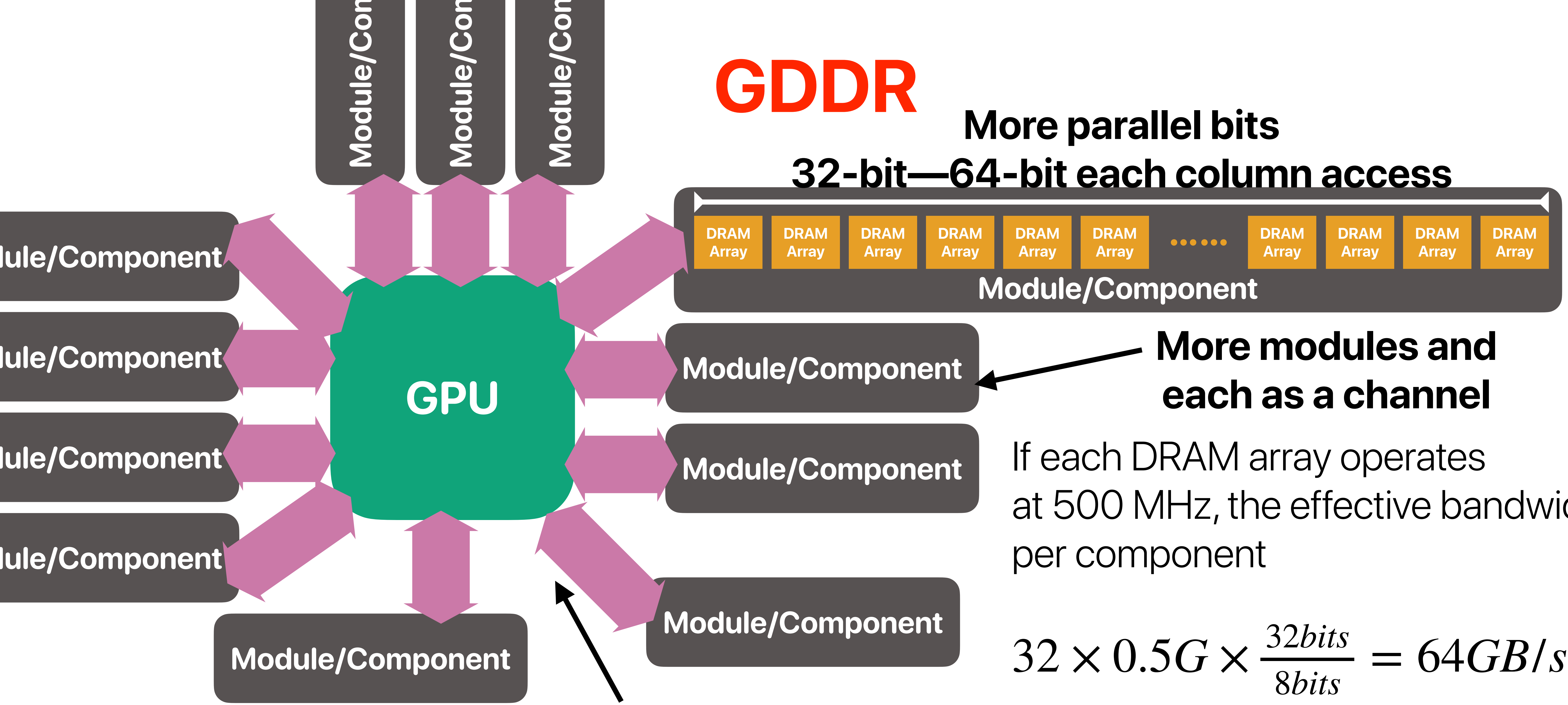
hardware accelerators lift the "ceiling" up

improving memory bandwidth can make the roof "steeper"

Ideas of increasing bandwidth

- More parallel bits
- More banks
- More channels
- Widen the memory-processor bus

GDDR



More parallel bits

32-bit—64-bit each column access

More modules and each as a channel

If each DRAM array operates at 500 MHz, the effective bandwidth per component

$$32 \times 0.5G \times \frac{32bits}{8bits} = 64GB/sec$$

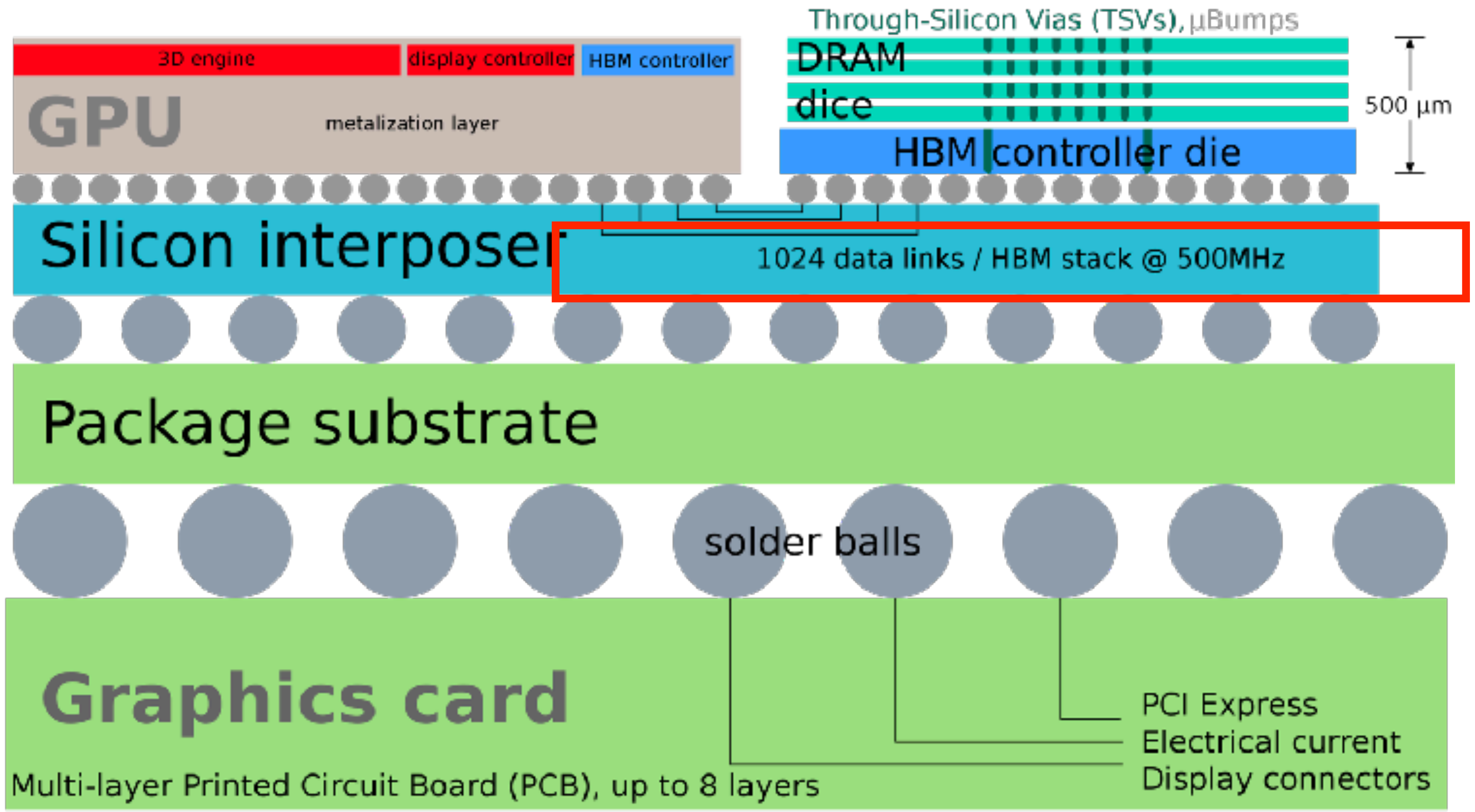
Wider bus: 192-bit — 384-bit bus

If the bus is 256-bit (32B) wide, the memory controller needs to be at

$$\frac{64GB/sec}{32B} = 2GHz$$

If you have 12x modules — 12*64 = 768 GB/sec

HBM (High Bandwidth Memory)

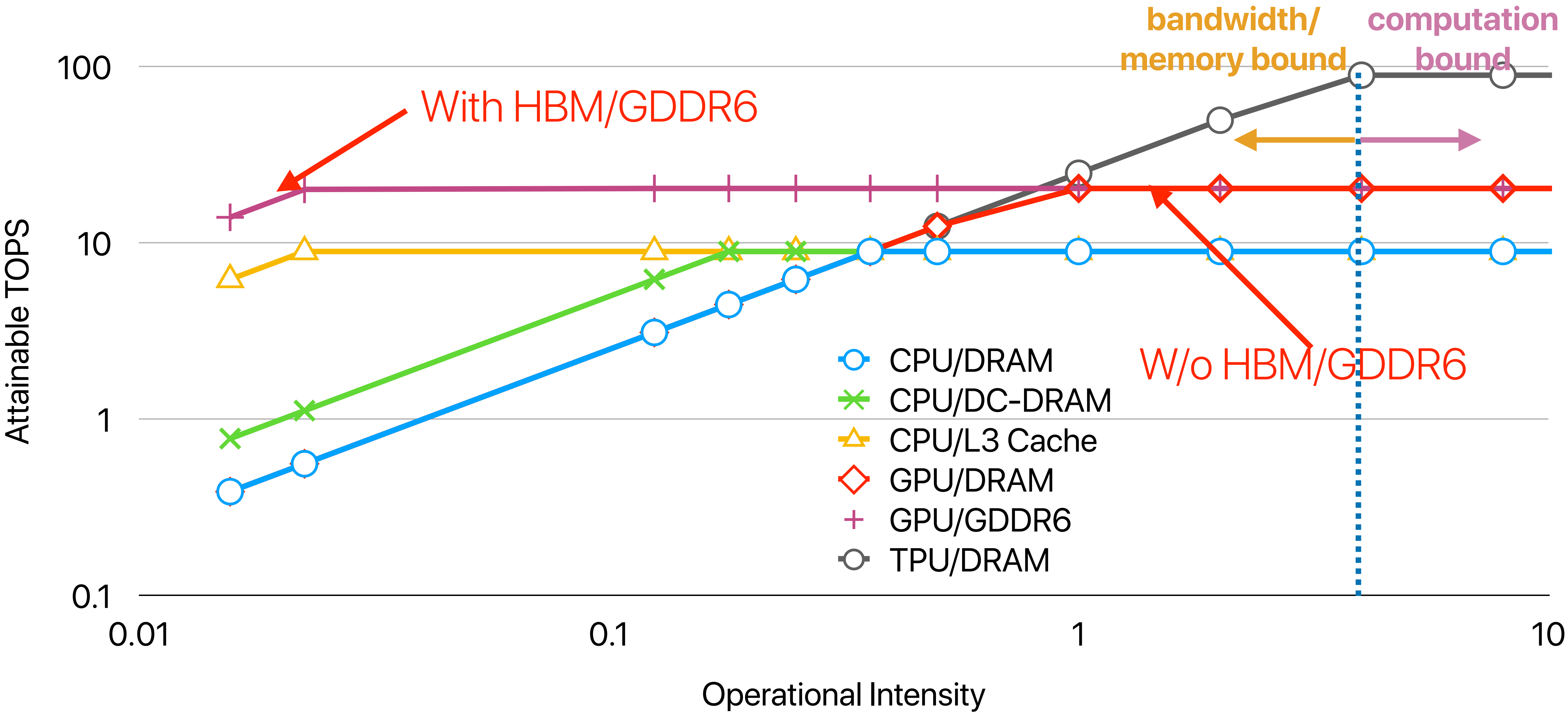


$$0.5G \times \frac{1024bits}{8bits} = 64GB/sec$$

If you have 4x modules — 4*64 = 256 GB/sec

HBM2 increase the clock rate to 2GHz — 1TB/sec

Recap: the roofline after using hardware accelerators



3:00

What’s the pros & cons of GDDR v.s. HBM

GDDR v.s. HBM

GDDR v.s. HBM

- HBM’s bandwidth is wider
- HBM is more expensive — pin counts are more expensive
- HBM is limited by the per-chip heat dissipation — less powerful

3:00

Which one do you prefer?
A. Single chip, heterogeneous processor (e.g., NVIDIA/intel) or
B. standalone accelerators? (e.g., TPU)

Single-chip or standalone ones?

Is system-wide heterogeneous processing a better idea?

- Easier for heat dissipation
  - Each processor can be more powerful
    - gamers/datacenters still prefers discrete GPUs
    - Cloud TPUs are standalone ones
- System size is larger, cost of ownership can be higher
- Data movement going through system interconnects
- Allow the “computation core” of the accelerator to enjoy larger bandwidth, if you do it right.

Case: what's the goal of the following program and what do you expect the performance would change when size varies?

3:00

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h> /* for clock_gettime */
#include <malloc.h>
#include <immintrin.h>


void write_memory_avx(void* array, size_t size) {
  __m256i* varray = (__m256i*) array;

  __m256i vals = _mm256_set1_epi32(1);
  size_t i;
  for (i = 0; i < size / sizeof(__m256i); i++) {
    _mm256_store_si256(&varray[i], vals);  // This will
generate the vmovaps instruction.
  }
}

int main(int argc, char **argv)
{
    size_t *array;
    size_t size;
    double total_time;
    struct timespec start, end;
    int i;
    size = atoi(argv[1])/sizeof(size_t);
```

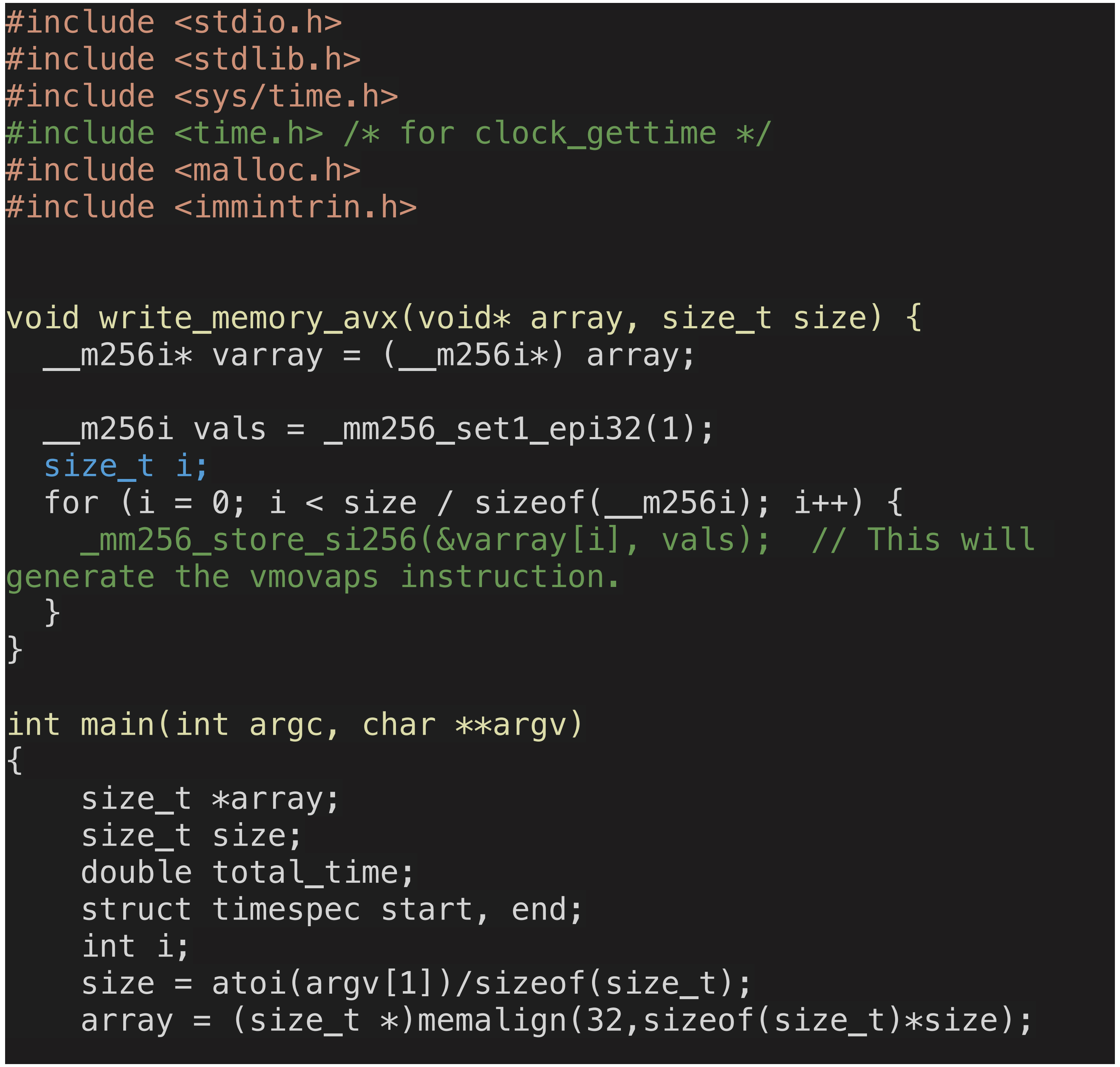
```c
    array = (size_t *)memalign(32,sizeof(size_t)*size);
    clock_gettime(CLOCK_MONOTONIC, &start); /* mark start
time */
    write_memory_avx(array, size);

    clock_gettime(CLOCK_MONOTONIC, &end); /* mark the end
time */
    total_time = ((end.tv_sec * 1000000000.0 + end.tv_nsec)
- (start.tv_sec * 1000000000.0 + start.tv_nsec));
    fprintf(stderr,"Latency: %.0lf ns, GBps:
%lf\n",total_time, (double)((double)size*sizeof(size_t)/
(total_time)));
    clock_gettime(CLOCK_MONOTONIC, &start); /* mark start
time */
    for(i = 0 ; i< 10; i++)
        write_memory_avx(array, size);

    clock_gettime(CLOCK_MONOTONIC, &end); /* mark the end
time */
    total_time = ((end.tv_sec * 1000000000.0 + end.tv_nsec)
- (start.tv_sec * 1000000000.0 + start.tv_nsec));
    fprintf(stderr,"Latency (10x average): %.0lf ns, GBps
(10x average): %lf\n",total_time/10, (double)
((double)size*10*sizeof(size_t)/(total_time)));
    return 0;
}
```

https://github.com/hungweitseng/EE277/tree/main/demo/memory

The program

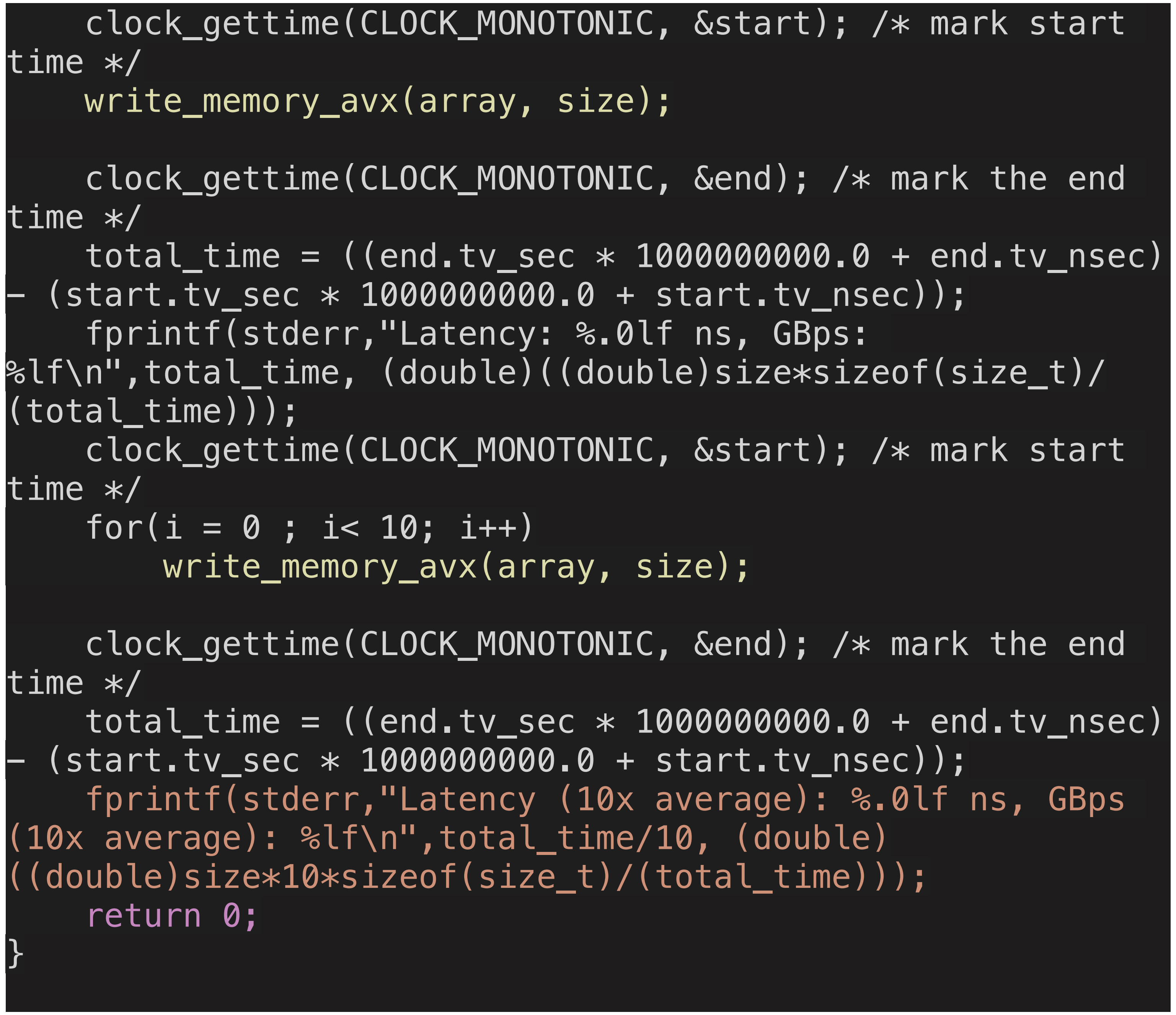


```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h> /* for clock_gettime */
#include <malloc.h>
#include <immintrin.h>


void write_memory_avx(void* array, size_t size) {
  __m256i* varray = (__m256i*) array;

  __m256i vals = _mm256_set1_epi32(1);
  size_t i;
  for (i = 0; i < size / sizeof(__m256i); i++) {
    _mm256_store_si256(&varray[i], vals);  // This will
generate the vmovaps instruction.
  }
}

int main(int argc, char **argv)
{
    size_t *array;
    size_t size;
    double total_time;
    struct timespec start, end;
    int i;
    size = atoi(argv[1])/sizeof(size_t);
    array = (size_t *)memalign(32,sizeof(size_t)*size);
```

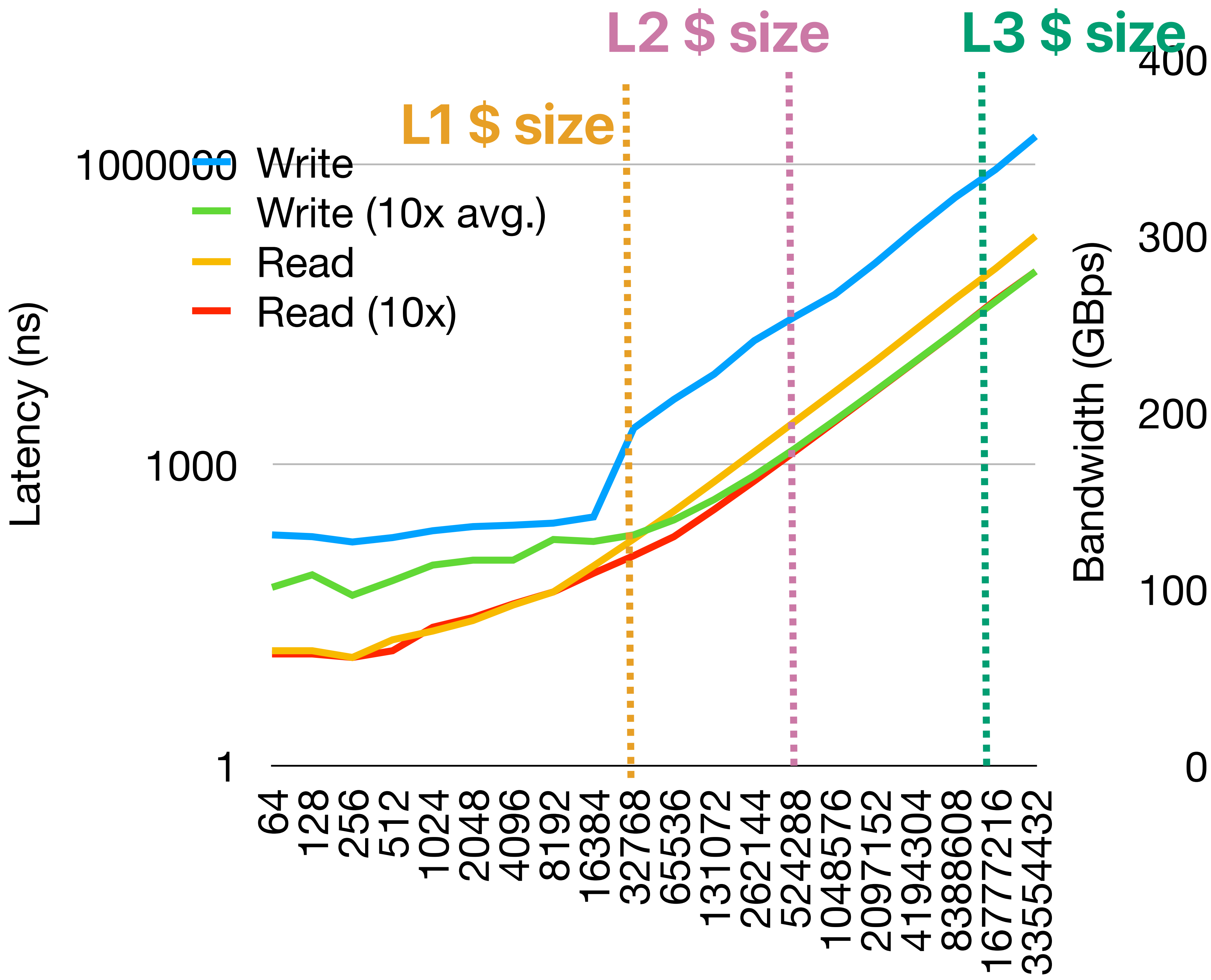


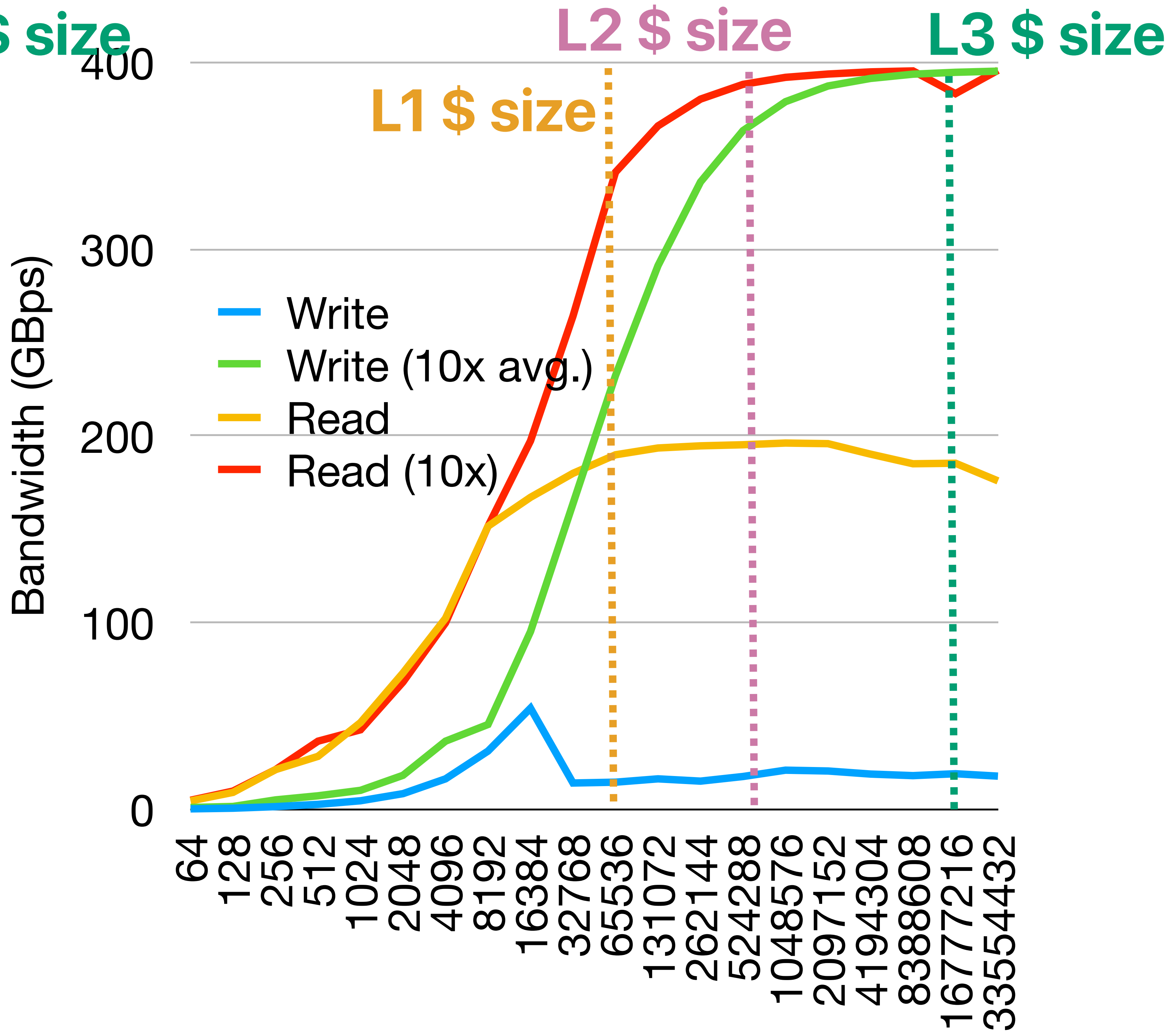
```
    clock_gettime(CLOCK_MONOTONIC, &start); /* mark start
time */
    write_memory_avx(array, size);

    clock_gettime(CLOCK_MONOTONIC, &end); /* mark the end
time */
    total_time = ((end.tv_sec * 1000000000.0 + end.tv_nsec)
- (start.tv_sec * 1000000000.0 + start.tv_nsec));
    fprintf(stderr,"Latency: %.0lf ns, GBps:
%lf\n",total_time, (double)((double)size*sizeof(size_t)/
(total_time)));
    clock_gettime(CLOCK_MONOTONIC, &start); /* mark start
time */
    for(i = 0 ; i< 10; i++)
        write_memory_avx(array, size);

    clock_gettime(CLOCK_MONOTONIC, &end); /* mark the end
time */
    total_time = ((end.tv_sec * 1000000000.0 + end.tv_nsec)
- (start.tv_sec * 1000000000.0 + start.tv_nsec));
    fprintf(stderr,"Latency (10x average): %.0lf ns, GBps
(10x average): %lf\n",total_time/10, (double)
((double)size*10*sizeof(size_t)/(total_time)));
    return 0;
}
```

Performance Chart (on AMD RyZen 5 2600)





3D-die stacking

- Providing huge bandwidth between memory chips and processors (e.g., HBM)
- The renaissance of near-memory processing!
  - Zhu, Qiuling, Berkin Akin, H. Ekin Sumbul, Fazle Sadi, James C. Hoe, Larry Pileggi, and Franz Franchetti. A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing.In 3DIC. 2013.
  - Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L. Greathouse, Lifan Xu, and Michael Ignatowski. TOP-PIM: throughput-oriented programmable processing in memory. HPDC '14. 2014
  - Farmahini-Farahani, Amin, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In HPCA. 2015.
  - Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A scalable processing-in-memory accelerator for parallel graph processing. ISCA '15. 2015.
  - Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning. In MICRO '52. 2019

Electrical
Computer Science
Engineering

277

つづく