Memory Contention Aware Swap Space Management

Student：Su-Wei Yang

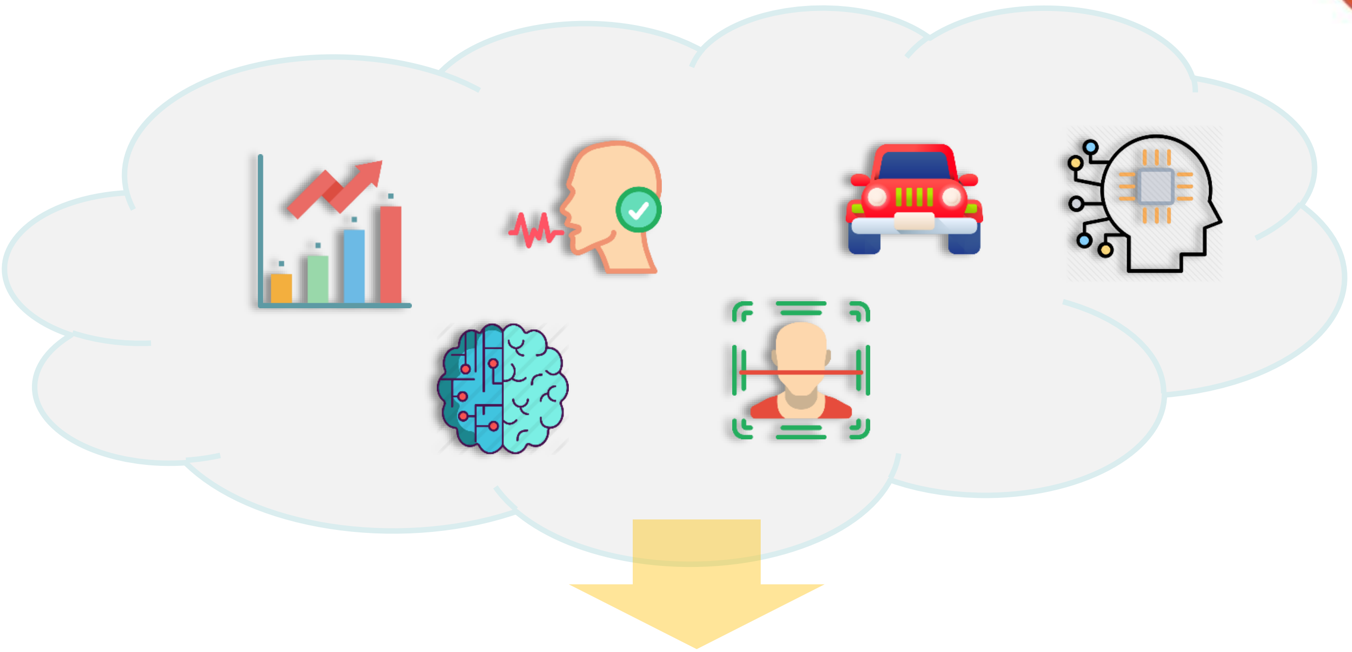
Advisor：Prof. Ya-Shu Chen

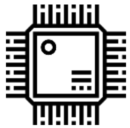
2019.07.29

Outline

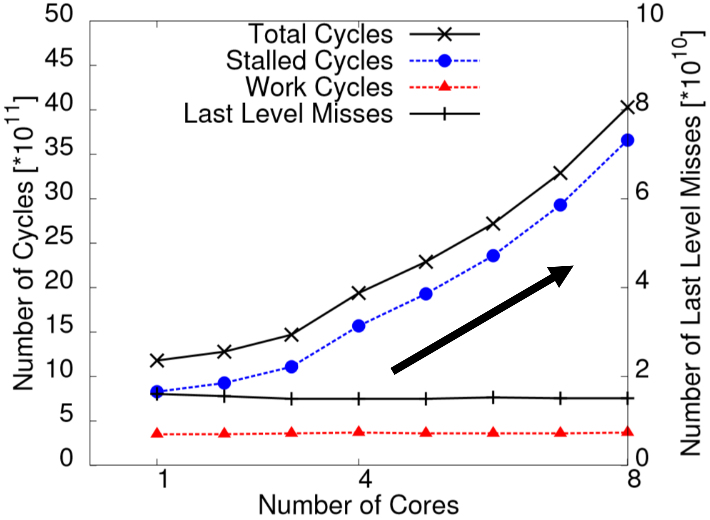
- Introduction
- Problem definition
- Related work
- System model
- Approach
- Experiment
- Conclusion

Introduction





Introduction



(a) Intel UMA: Xeon E5320

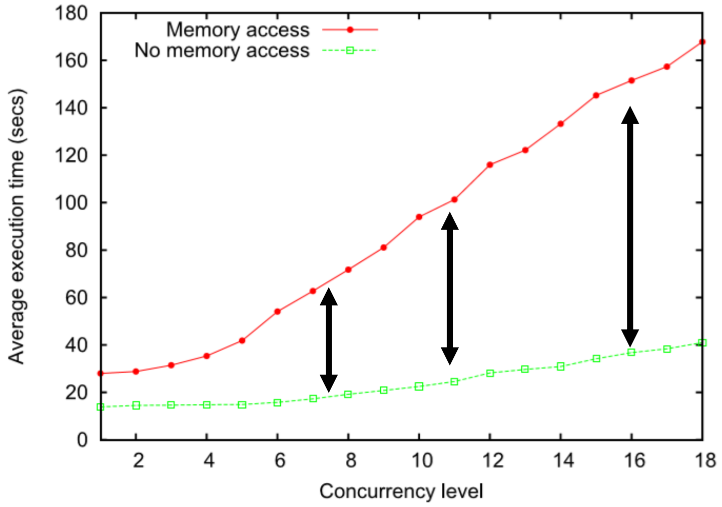
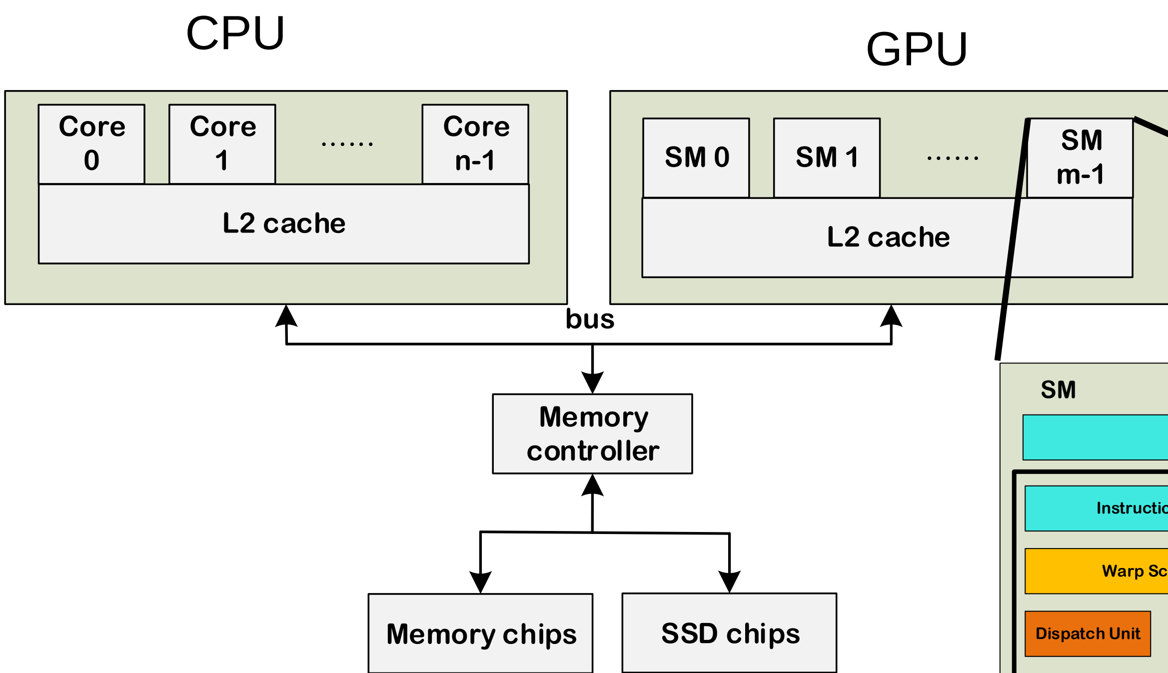


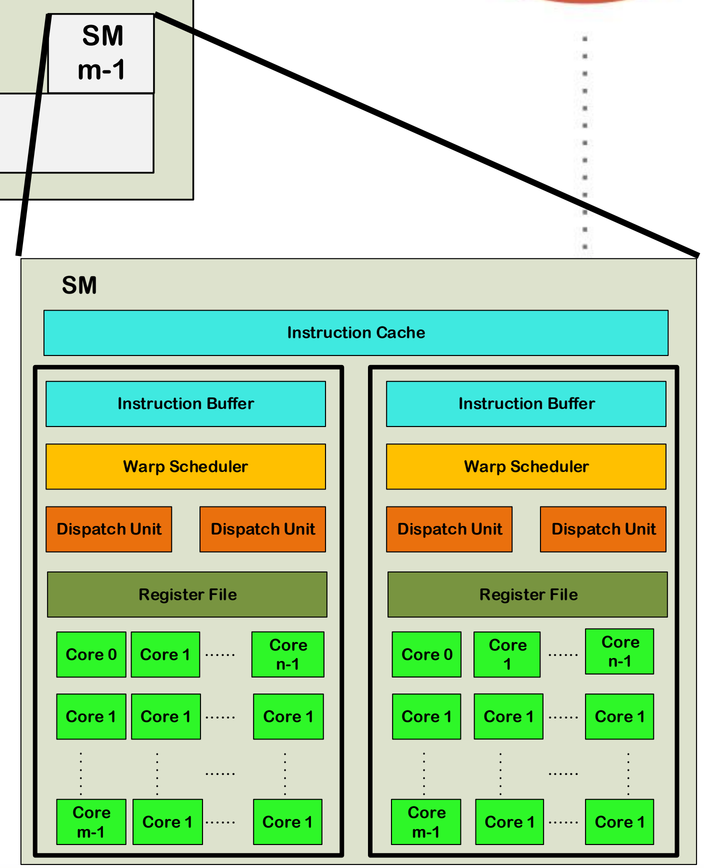
Fig. 1. Effect of memory contention on a 12-core Westmere EP machine.

BARDHAN, Shouvik; MENASCÉ, Daniel A. Predicting the effect of memory contention in multi-core computers using analytic performance models. *IEEE Transactions on Computers*, 2014, 64.8: 2279-2292.

TUDOR, Bogdan Marius; TEO, Yong Meng; SEE, Simon. Understanding off-chip memory contention of parallel programs in multicore systems. In: 2011 International Conference on Parallel Processing. IEEE, 2011. p. 602-611.

Introduction





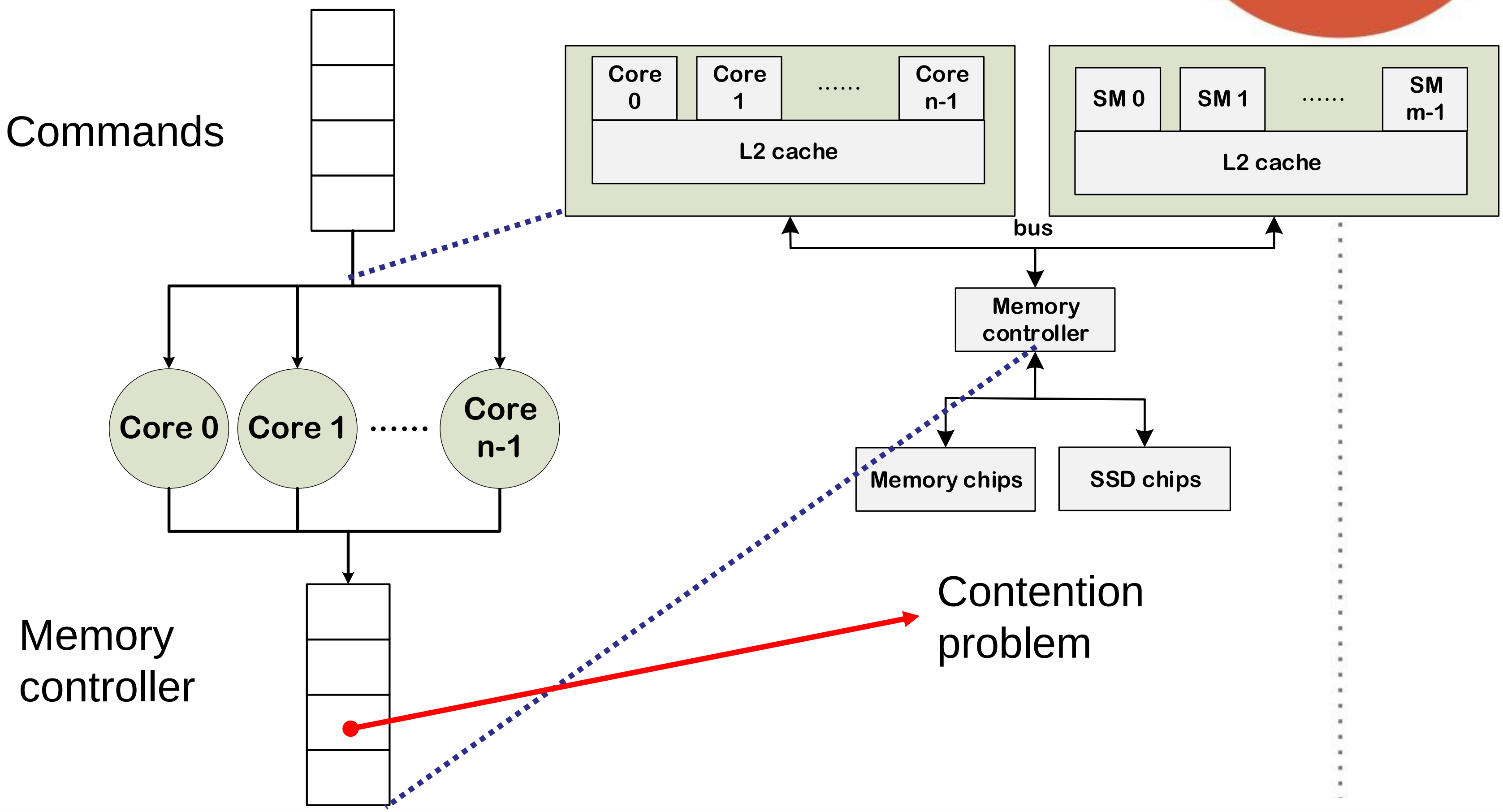
Unified Memory Architecture (UMA)

https://devblogs.nvidia.com/unified-memory-in-cuda-6/

TUDOR, Bogdan Marius; TEO, Yong Meng; SEE, Simon. Understanding off-chip memory contention of parallel programs in multicore systems. In: 2011 International Conference on Parallel Processing. IEEE, 2011. p. 602-611.

Introduction

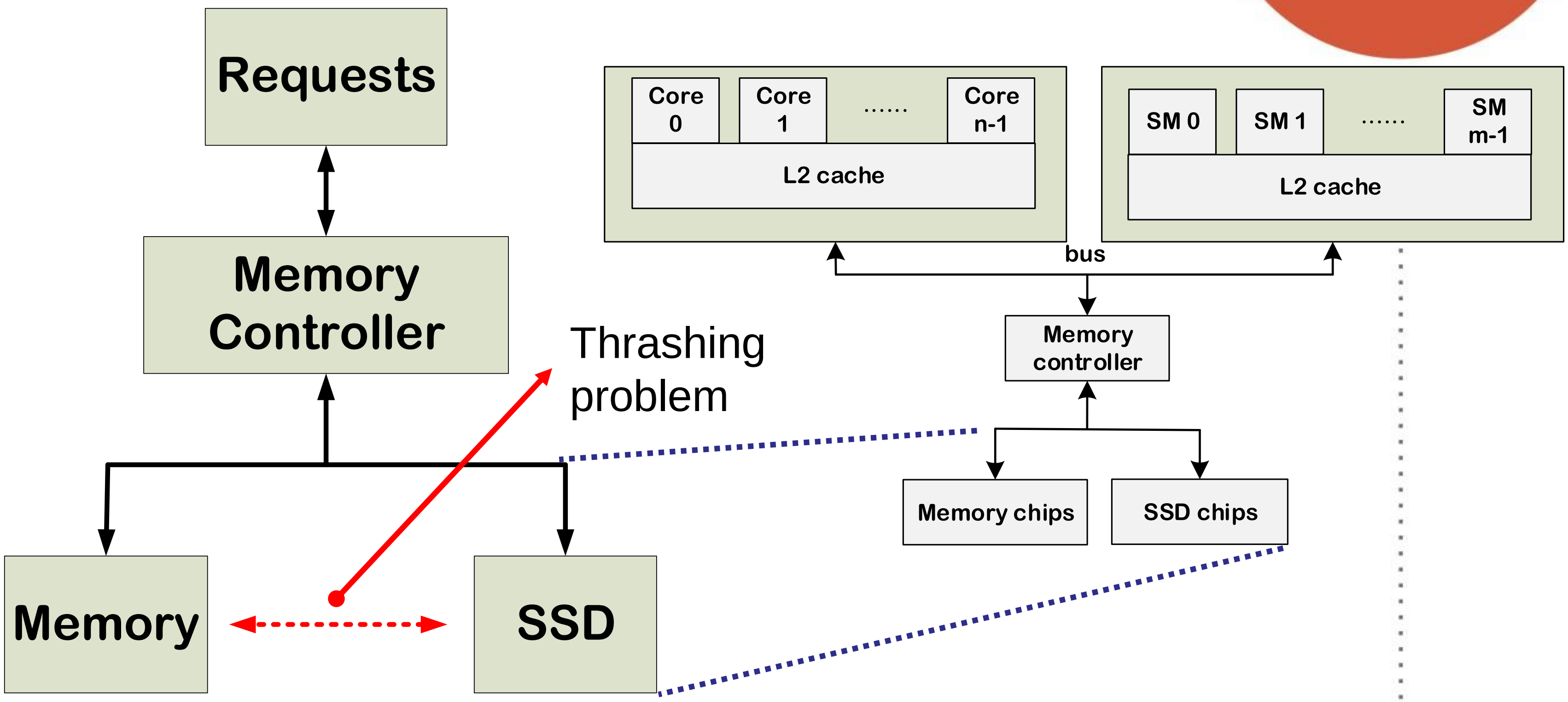
Memory Contention



BARDHAN, Shouvik; MENASCÉ, Daniel A. Predicting the effect of memory contention in multi-core computers using analytic performance models. *IEEE Transactions on Computers*, 2014, 64.8: 2279-2292.

Introduction

Memory Thrashing



BARDHAN, Shouvik; MENASCÉ, Daniel A. Predicting the effect of memory contention in multi-core computers using analytic performance models. IEEE Transactions on Computers, 2014, 64.8: 2279-2292.

Problem definition

- **Given**
  - Applications
- **Goal**
  - Minimize latency by reducing memory contention and thrashing
- **Method**
  - Contention-aware scheduling
  - Swap space assignment
- **Constraint**
  - Memory Size

Related work

- **Memory contention**
  - JOUPPI, Norman P., et al. [1] present the bottleneck from computing component to memory when the chip is powerful
  - TUDOR, Bogdan Marius; TEO, Yong Meng; SEE, Simon. [2] provides the latency measurement to evaluate memory contention.
  - BARDHAN, Shouvik; MENASCÉ, Daniel A. [3] present the memory contention increased with an increased number of applications or cores.

[1] JOUPPI, Norman P., et al. In-datacenter performance analysis of a tensor processing unit. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA).

[2] TUDOR, Bogdan Marius; TEO, Yong Meng; SEE, Simon. Understanding off-chip memory contention of parallel programs in multicore systems. International Conference on Parallel Processing. IEEE, 2011.

[3] BARDHAN, Shouvik; MENASCÉ, Daniel A. Predicting the effect of memory contention in multi-core computers using analytic performance models. IEEE Transactions on Computers, 2014, 64.8: 2279-2292.

Related work

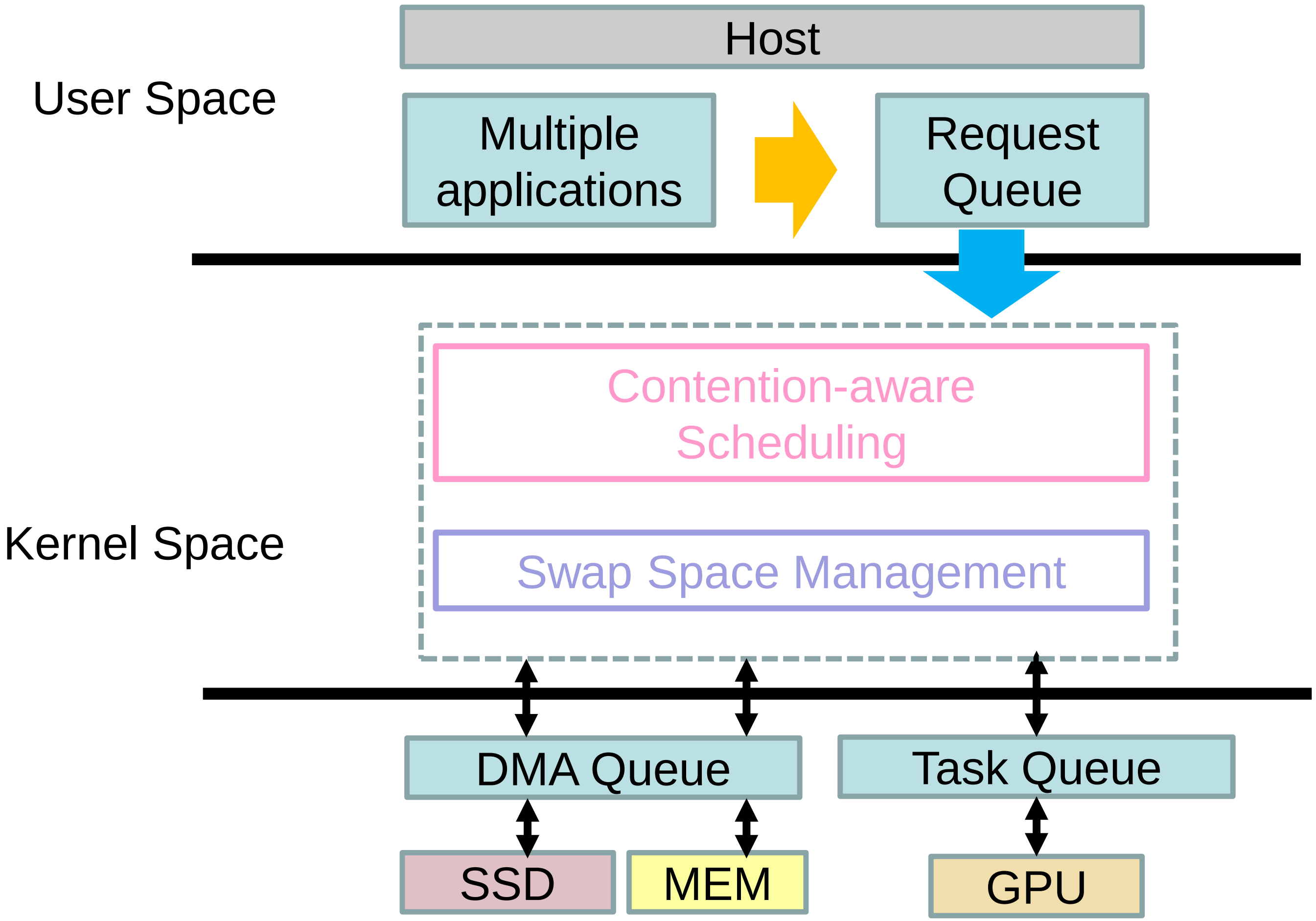
- **Reducing memory thrashing**
  - Chen, Li, et al.[4] present a way to reduce memory thrashing by controlling SM open and close while memory thrashing occurs.
- **Reducing latency by swapping**
  - Zhu, Xiao, et al. [5] present to swap applications memory pages out early to reduce latency
  - ZHUANG, Zhenyun, et al. [6] present the latency increased by inappropriate swapping

[4] LI, Chen, et al. A Framework for Memory Oversubscription Management in Graphics Processing Units. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019. p. 49-63..
[5] Zhu, Xiao, et al. "SmartSwap: High-performance and user experience friendly swapping in mobile systems." Proceedings of the 54th Annual Design Automation Conference 2017. ACM, 2017

[6] ZHUANG, Zhenyun, et al. Taming memory related performance pitfalls in linux cgroups. In: 2017 International Conference on Computing, Networking and Communications (ICNC). IEEE, 2017.

Framework

System Model



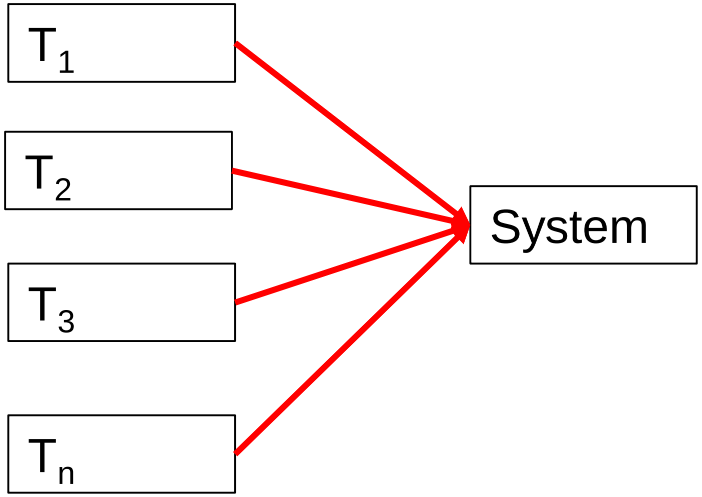
Approach

- Contention-aware Scheduling
  - Memory contention evaluation
  - Contention-aware priority assignment
- Swap space management
  - Direct swap space accessing
  - Contention-aware swap-in strategy
  - Swap space assignment

Approach

Scheduling

- Memory contention degree evaluation
  - $\omega_i(n) = \frac{c_i(n) - ci(1)}{c_i(1)} * \frac{1}{N_i}$



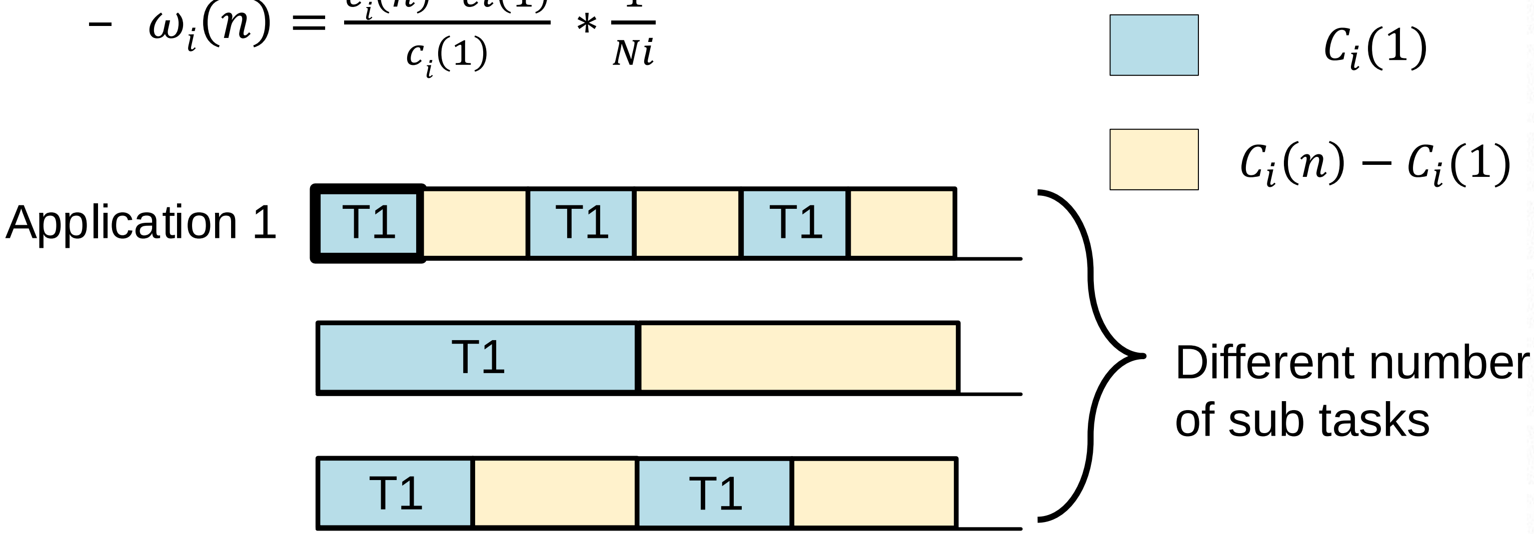
- $c_i(n)$: response time under multiple applications
- $c_i(1)$: response time under single application T_i
- N_i: Number of threads of the corresponding application

TUDOR, Bogdan Marius; TEO, Yong Meng; SEE, Simon. Understanding off-chip memory contention of parallel programs in multicore systems. In: *2011 International Conference on Parallel Processing*. IEEE, 2011.

Scheduling

Approach

- Memory contention degree evaluation
  - $\omega_i(n) = \frac{c_i(n) - ci(1)}{c_i(1)} * \frac{1}{Ni}$



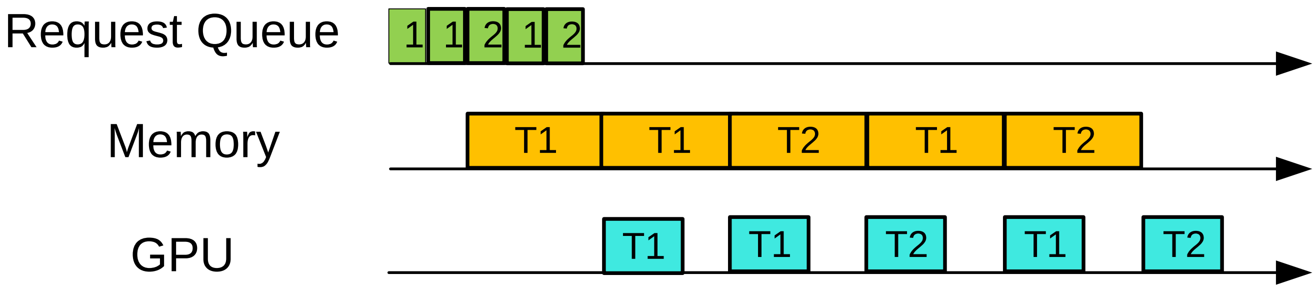
- $c_i(n)$: response time under multiple applications
- $c_i(1)$: response time under single application T_i
- N_i: Number of sub tasks of the corresponding application

TUDOR, Bogdan Marius; TEO, Yong Meng; SEE, Simon. Understanding off-chip memory contention of parallel programs in multicore systems. In: *2011 International Conference on Parallel Processing*. IEEE, 2011.

Scheduling

Contention-aware priority assignment

- Setting the priority by contention degree.
- The higher contention degree the higher priority.

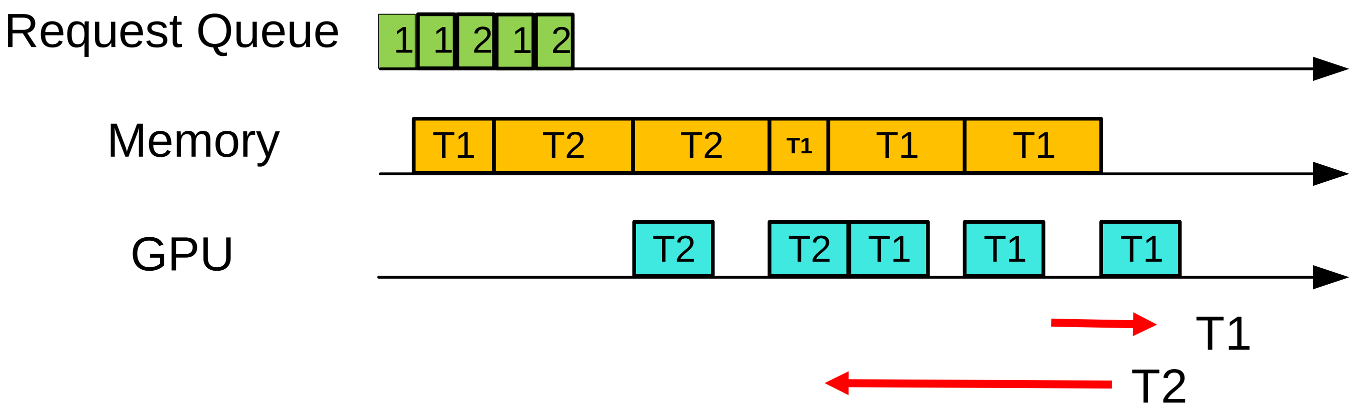


Approach

Scheduling

Contention-aware priority scheduling

- Reduce the response time of the higher priority
- Reduce the latency caused by memory contention



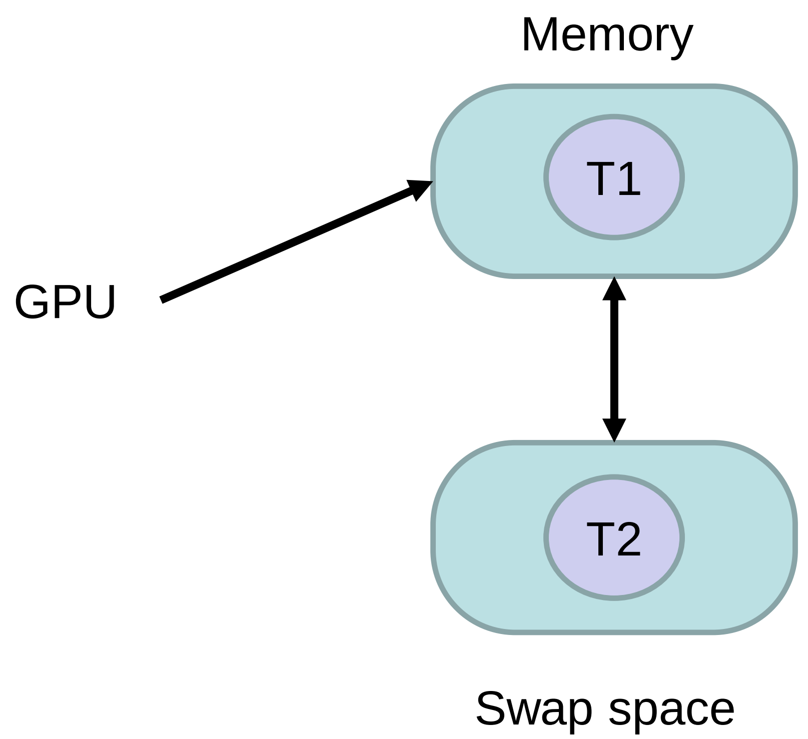
Approach

- Contention-aware Scheduling
  - Memory contention evaluation
  - Contention-aware priority assignment
- Swap space management
  - Direct swap space accessing
  - Contention-aware swap-in strategy
  - Swap space assignment

Thrashing Problem

Reduce thrashing

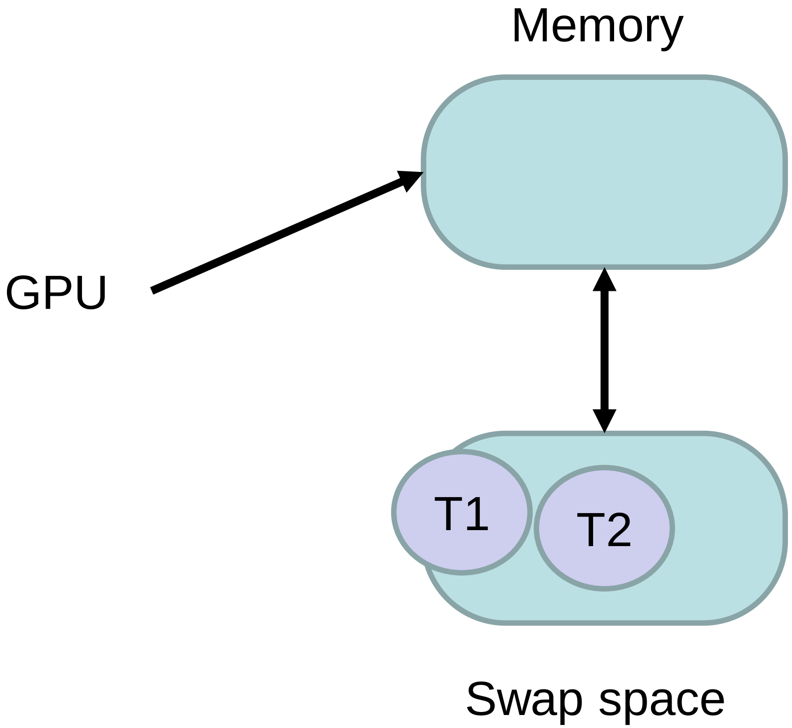
- Memory-oriented execution



Thrashing Problem

Reduce thrashing

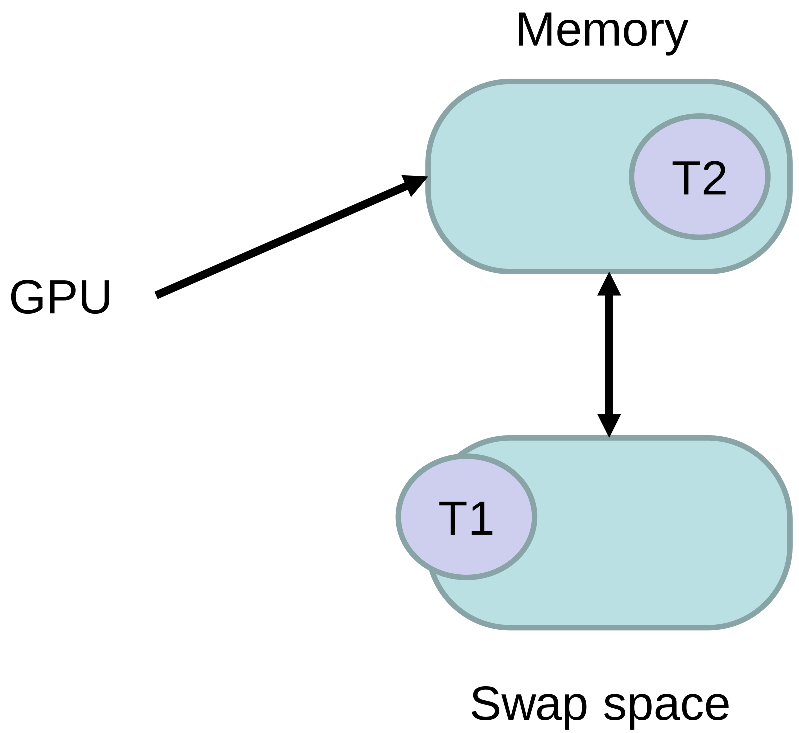
- Memory-oriented execution



Thrashing Problem

Reduce thrashing

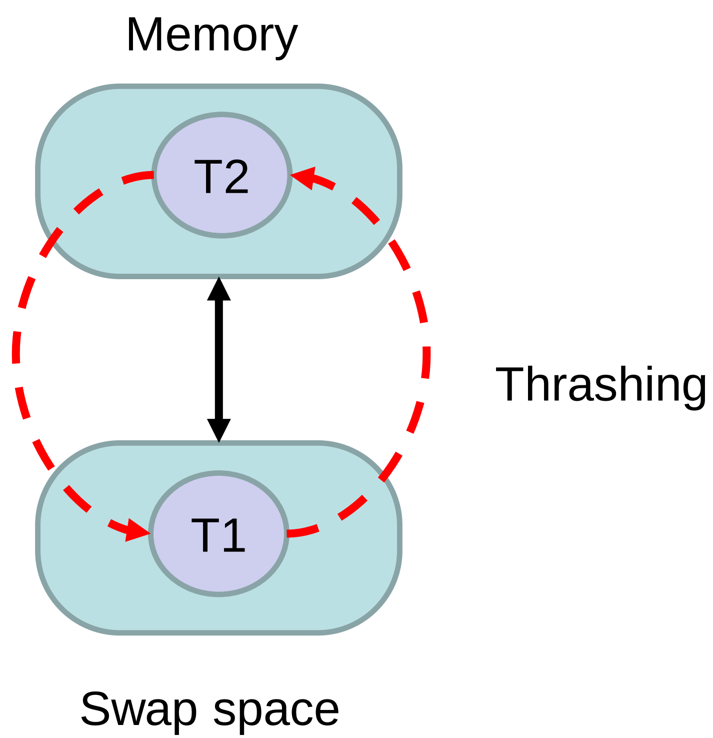
- Memory-oriented execution



Thrashing Problem

Reduce thrashing

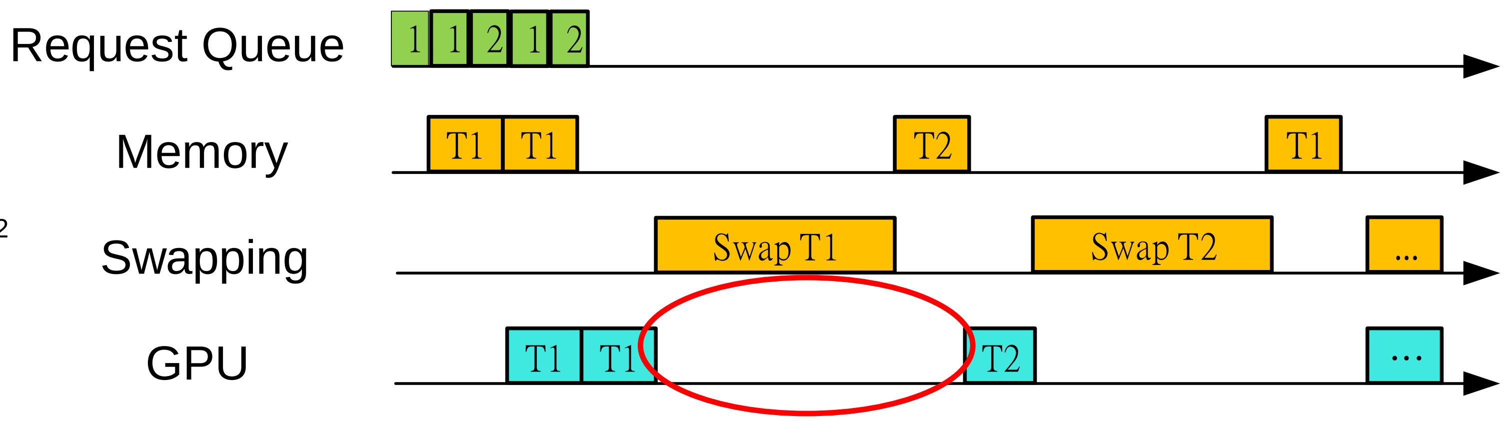
- Memory-oriented execution



Thrashing Problem

Reduce thrashing

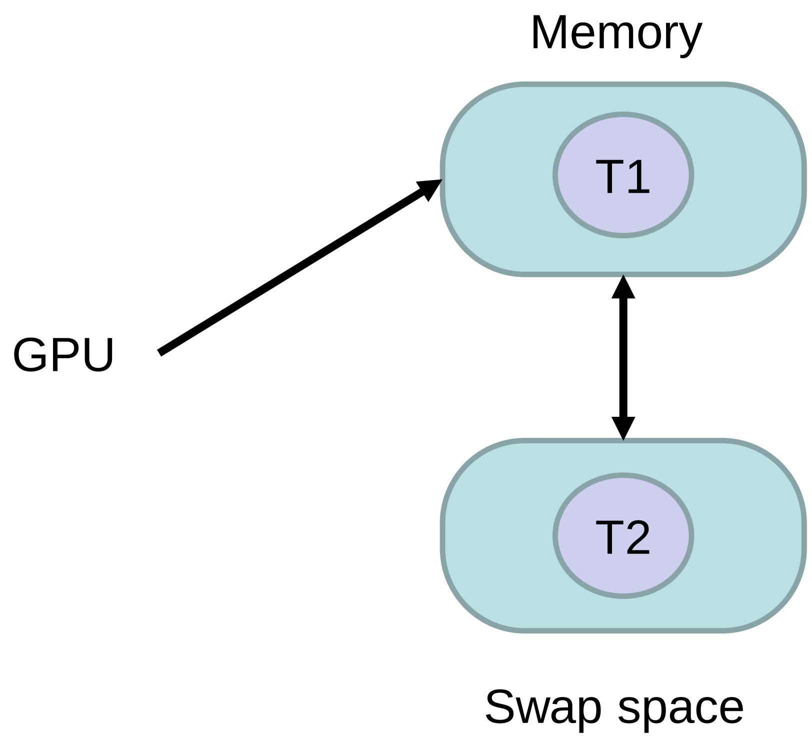
- Latency increased by swapping



Swap Space Management

Approach

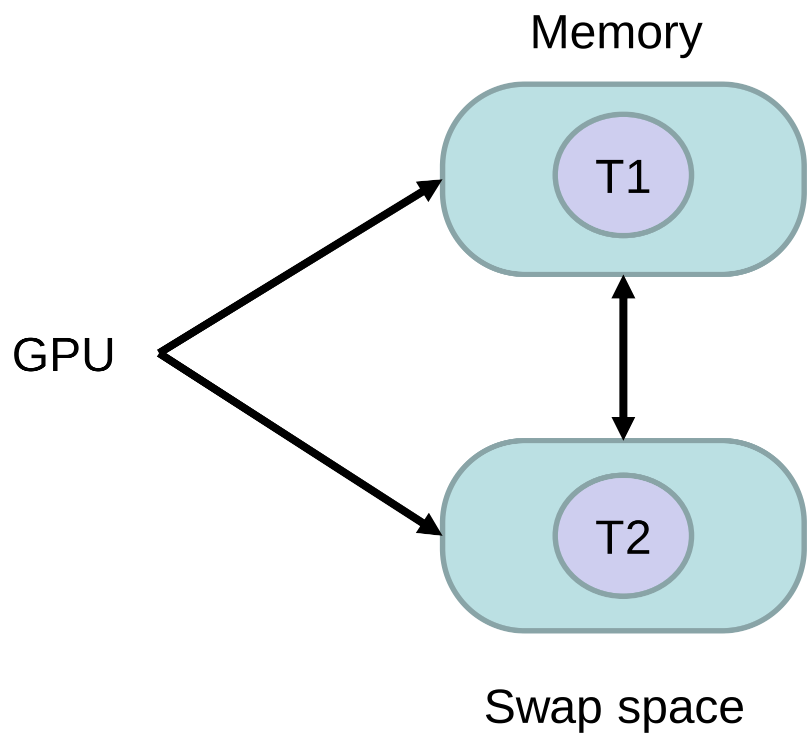
- Direct swap space accessing



Swap Space Management

Approach

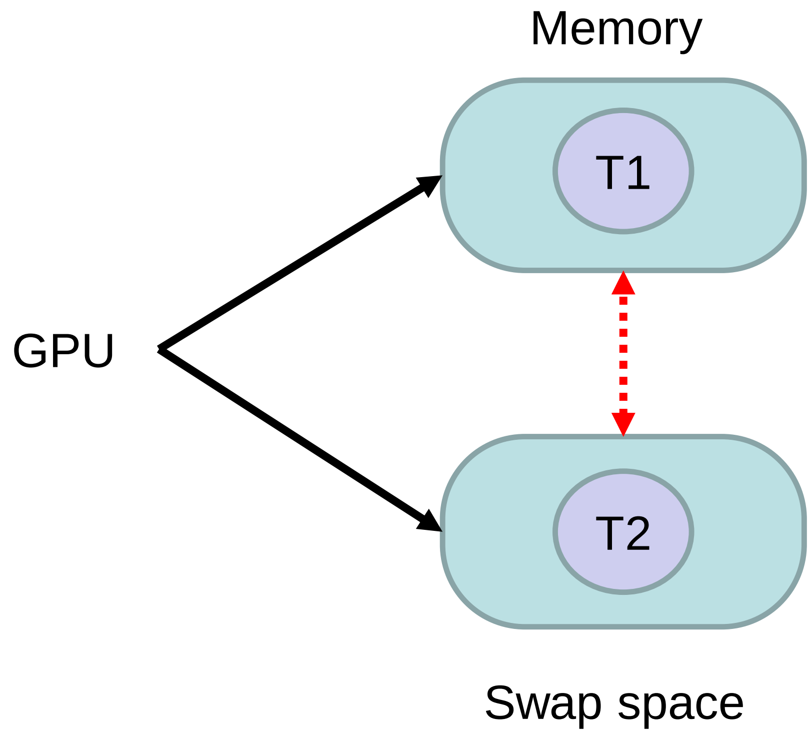
- Direct swap space accessing



Swap Space Management

Approach

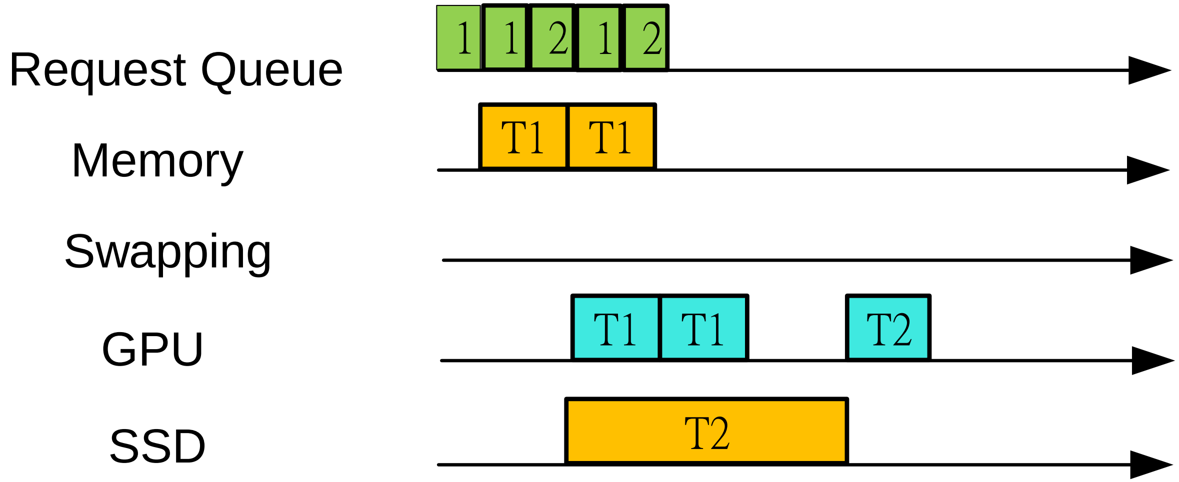
- Direct swap space accessing



Swap Space Management

Approach

- Reduce latency
  - Direct swap space accessing



Swap Space Management

- Contention-aware swap-in strategy
  - Swap in high priority application to reduce latency
  - Execute low priority application by direct swap space accessing to reduce thrashing
- How to assign the swap space size to reduce thrashing?
  - Memory usage evaluation
  - Average distribute direct swap space accessing

Swap Space Assignment

- Reduce thrashing
  - Check the required memory size for all executing applications
    - $SWAP_{req} = \sum MEM(i) - MEM_{\mathrm{sys}}$
- Sort all applications with priority

α_i : swap ratio of $\mathrm{T_i}$
$SWAP_{req}$: required swap space
MEM_{sys} : system memory size
$MEM(i)$: $All\ required\ memort\ size\ of$ Ti

Swap Space Assignment

Approach

- Required swap space assignment from lowest priority application T_i
  - $\alpha_i = \begin{cases} \frac{SWAP_{req}}{MEM(i)}, SWAP_{req} < MEM(i) \\ 1, \quad SWAP_{req} \geq MEM(i) \end{cases}$
  - $SWAP_{req}$ updated as $SWAP_{req} - \alpha_i \; MEM(\mathrm{i})$
  - Repeat assign the swap space for the lower priority application T_i until $SWAP_{req}$ as zero

α_i : swap ratio of $\mathrm{T_i}$
$SWAP_{req}$: required swap space
MEM_{sys} : system memory size
$MEM(i)$: $All\ required\ memory\ size\ of$ Ti

Experiment Setup

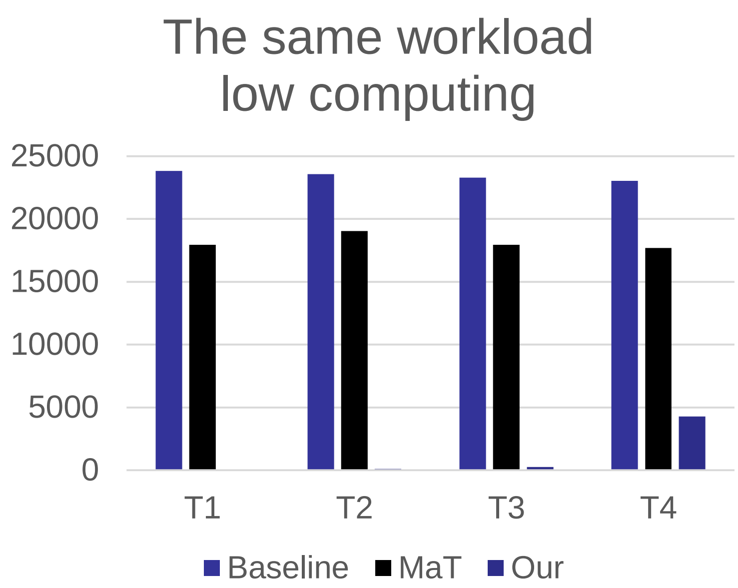
Result

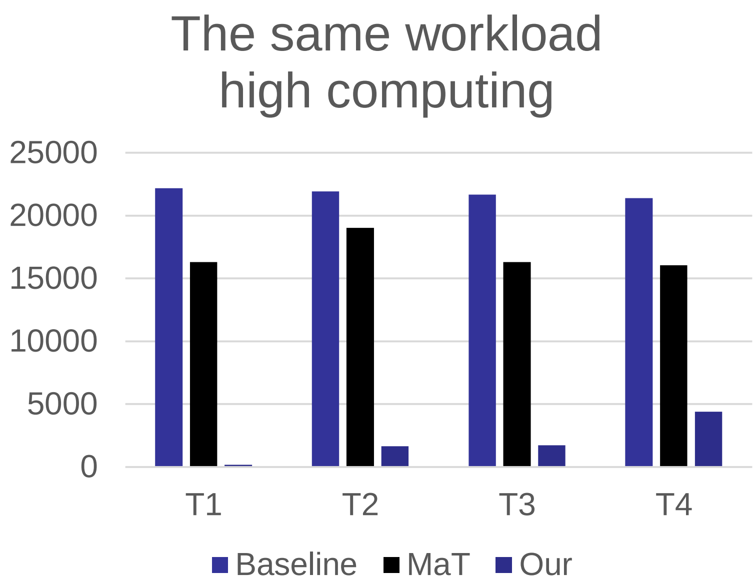
- Setting
  - Applications number : 4
  - Sub tasks for each application : 5,7,9,11,128,256
  - Required memory of all applications : 24GB, 320MB
  - System memory constraint : 18GB,240MB
- Performance Evaluation
  - Lengthened latency
- Comparison
  - Baseline(non-swap management)
  - MaT[4]

[4] LI, Chen, et al. A Framework for Memory Oversubscription Management in Graphics Processing Units. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019. p. 49-63.

Performance

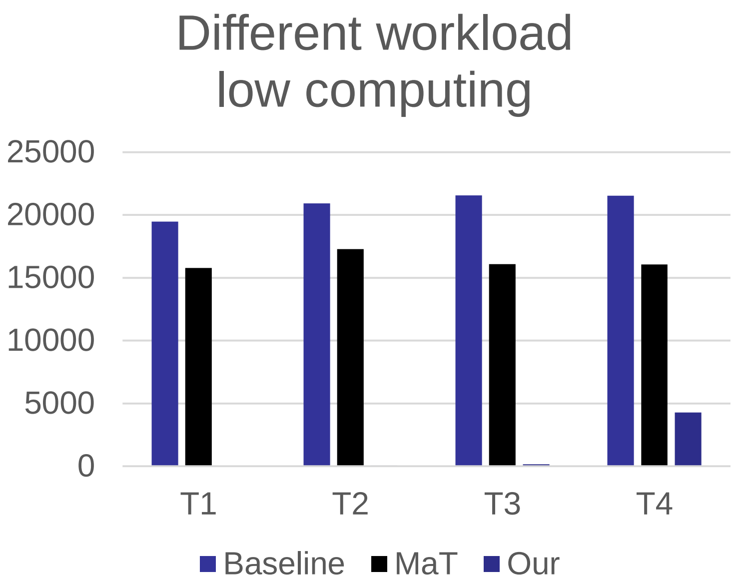
The same

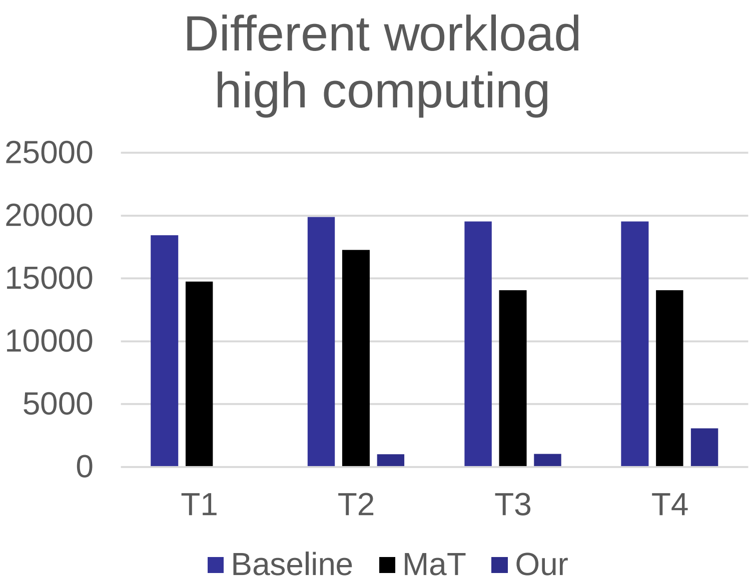




Performance

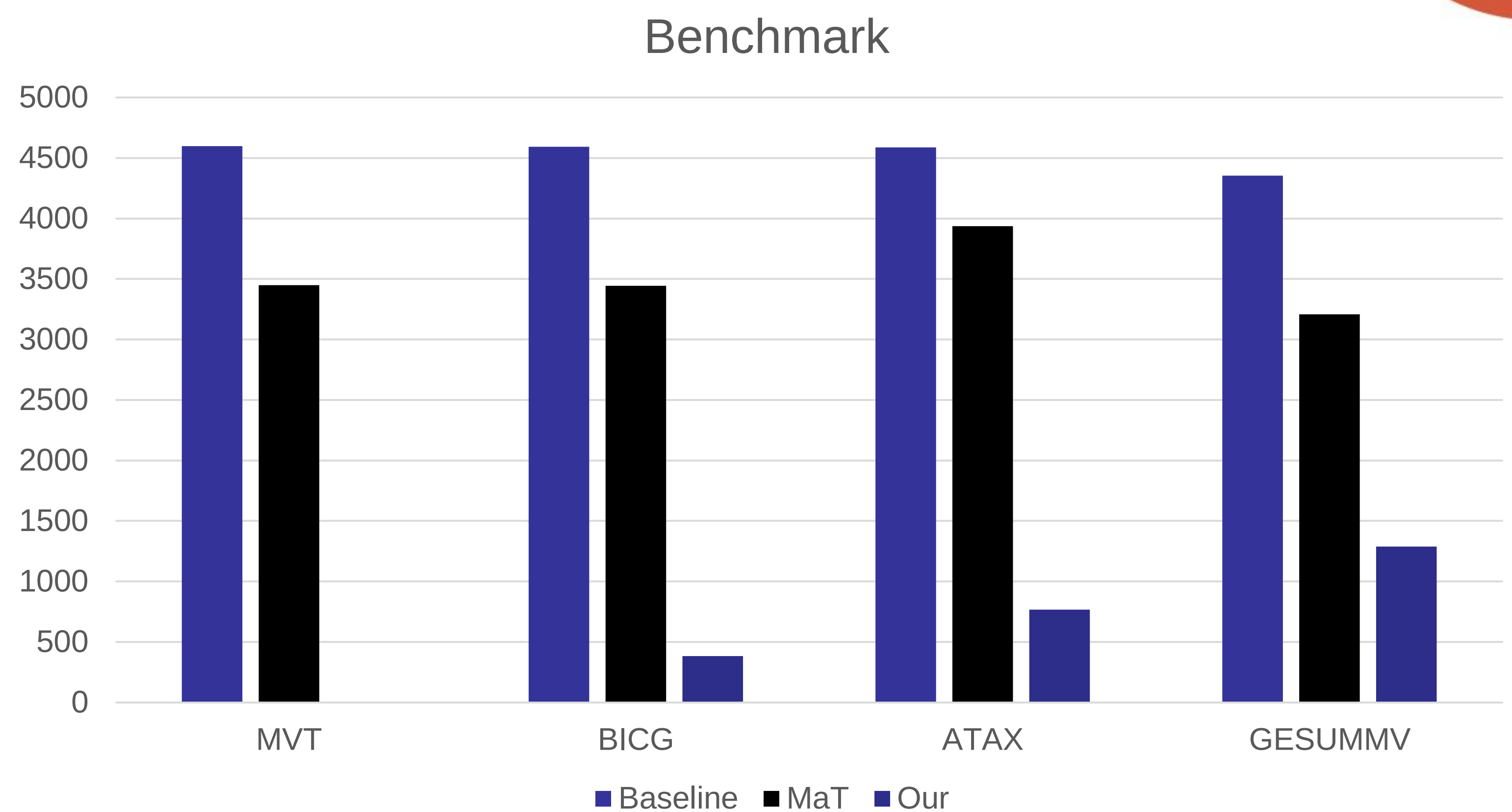
The different





Performance

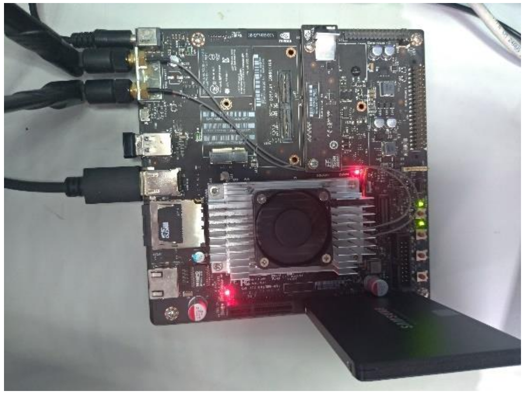
benchmark



Result

Case Study

- **Platform (Nvidia TX2)**
  - CPU ： Denver CPU (2 cores)+ Cortex A57 CPU (4 cores)
  - Memory ： 8GB LPDDR4
  - Total swap space：16GB
  - Storage device ：Samsung 860 EVO SSD (SATA3.0)



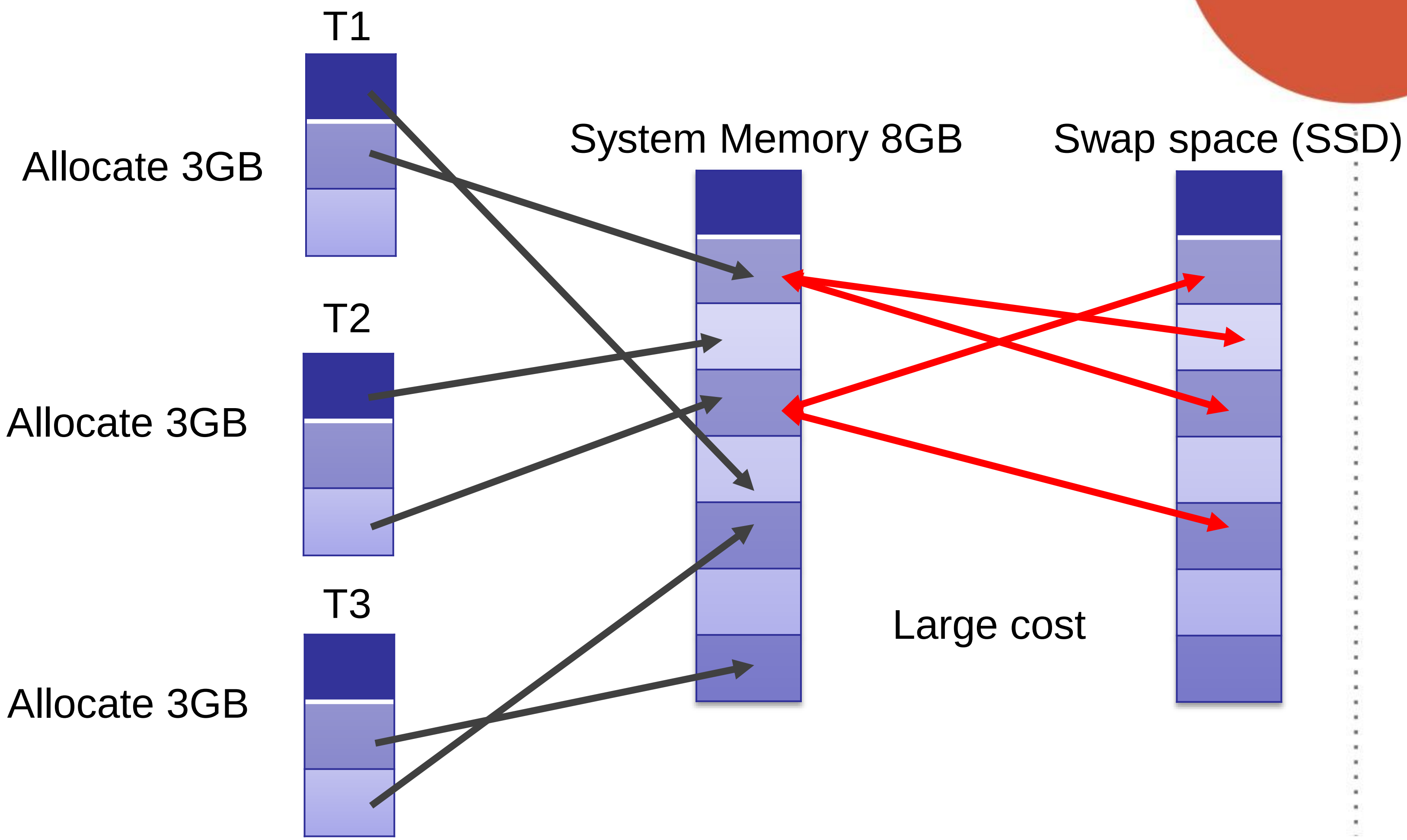
https://www.arrow.com/en/products/945-82771-0000-000/nvidia
https://www.samsung.com/tw/memory-storage/860-evo-sata-3-2-5-ssd/MZ-76E250BW/

Experiment Setup

- **Applications**
  - Application number : 3
  - Required of all applications : 9GB
  - OS:Ubuntu16.0.4
  - Kernel version : Linux 4.4.38
  - Container: Cgroup
- **Performance Evaluation**
  - Response time

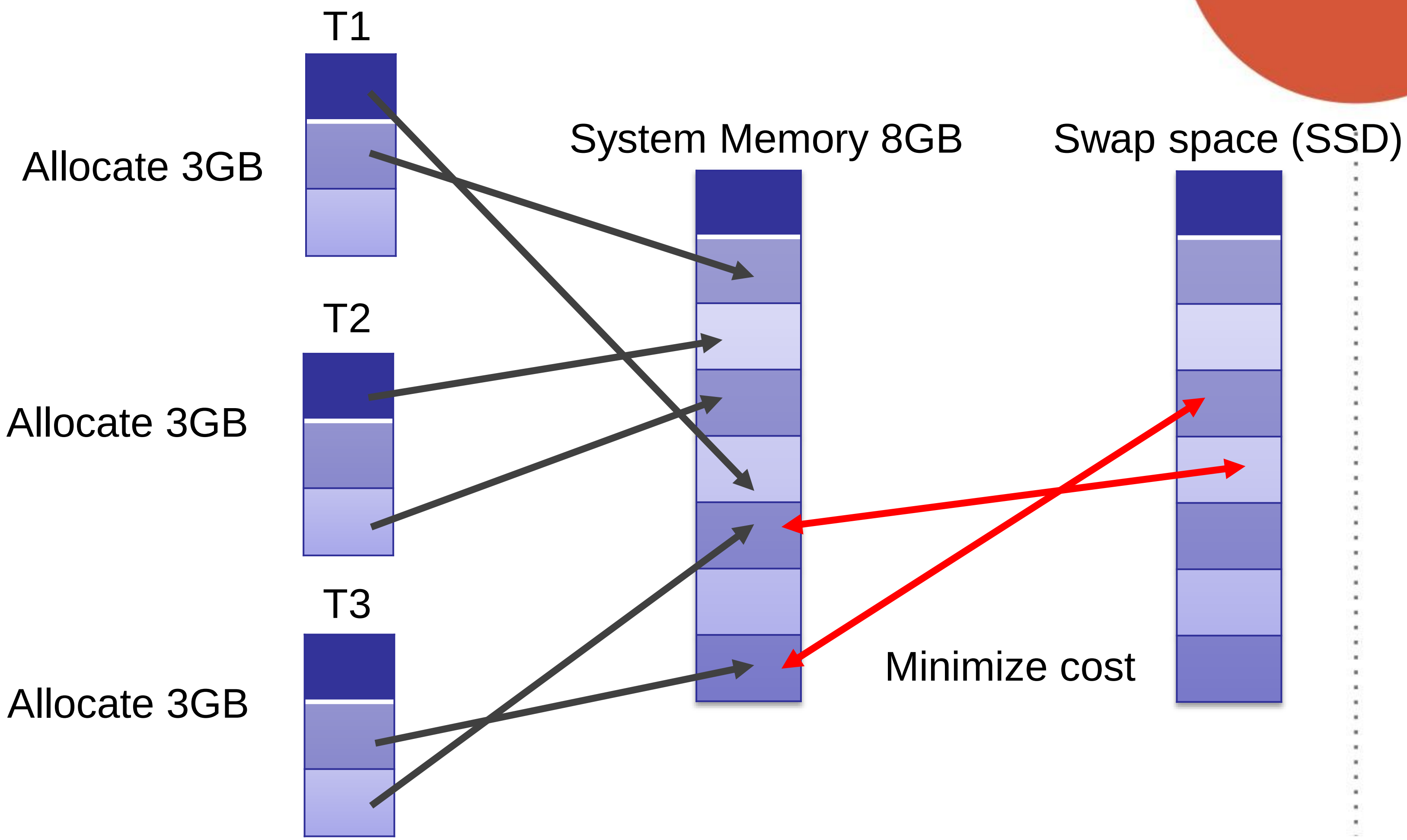
Default Management

Results

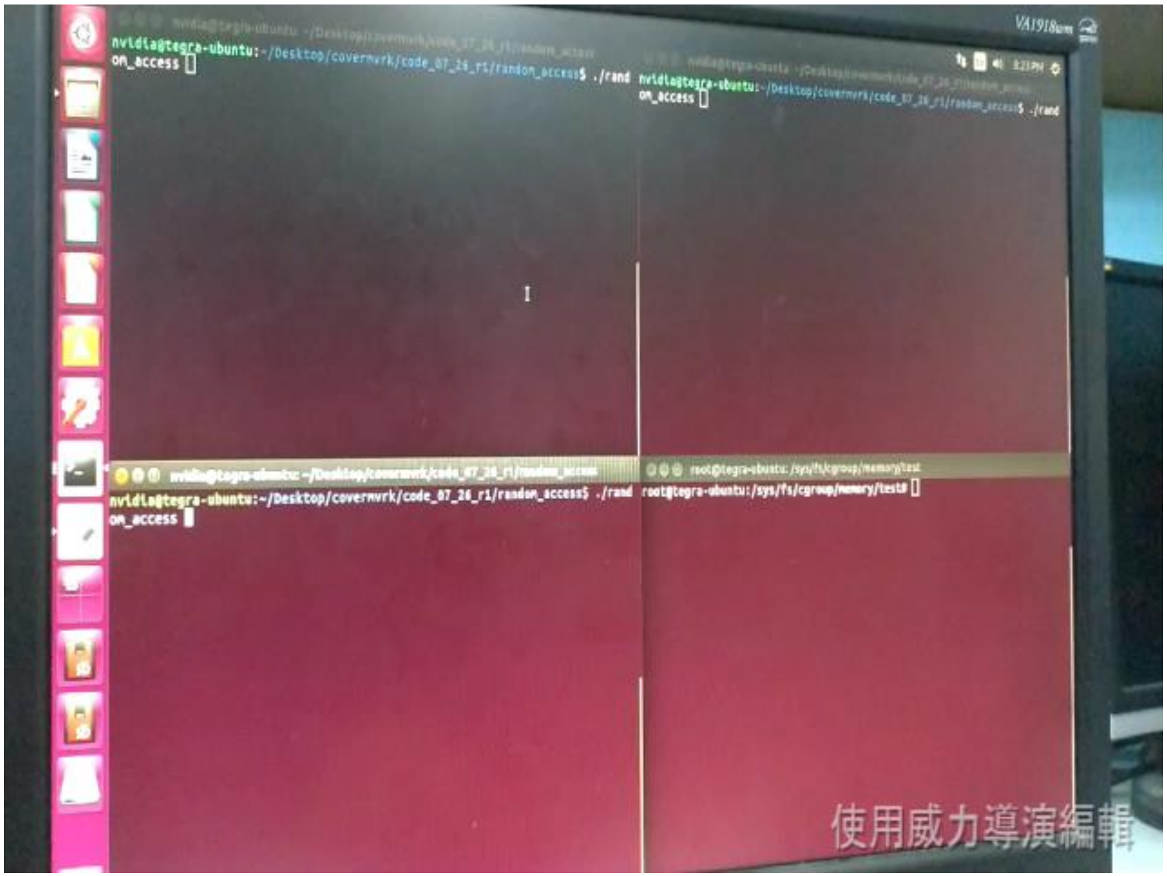


Results

Our Swap Management



Video



Results

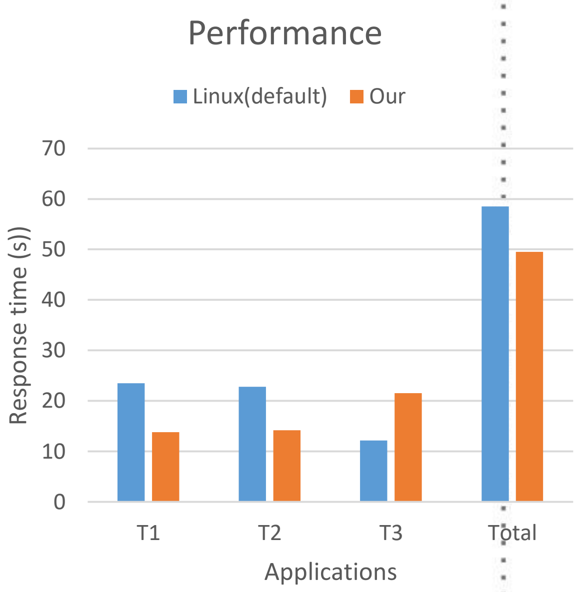
Result

Without swap management

Response time(s)	T1	T2	T3	Total
First	23.39	22.97	12.44	58.80
Second	23.27	22.75	11.65	58.37
Third	23.9	22.68	12.48	59.06

With swap management

Response time(s)	T1	T2	T3	Total
First	13.44	14.31	21.52	49.27
Second	14.48	13.85	21.64	49.97
Third	13.53	14.33	21.37	49.23



Conclusion

- Executing multiple data-oriented applications introduces
  - Memory contention
  - Thrashing
- Proposed memory contention-aware scheduling and thrashing elimination strategy
  - Contention-aware scheduling
  - Swap space management
- Performance evaluation
  - Combing scheduling and direct swap accessing can minimize latency up to 41.3%
  - Eliminate the thrashing can reduce latency up to 16.6% on the real platform

THANKS

Q&A

NTUST National Taiwan University of Science and Technology

ESS LAB Embedded System Software Laboratory

Experiment

On TX2

	Time(s)			Average(s)
Application 1	13.34	10.45	10.90	11.56
Application 2	11.7	11.18	10.91	11.26
Application 3	12.51	11.85	10.81	11.72
				11.52

	Time(s)			Average(s)
Application 1	10.65	9.72	11.44	10.60
Application 2	10.71	10.63	10.91	10.75
Application 3	10.73	10.63	11.73	11.03
				10.79