Evaluation of the Performance of the Memory Tagging Extensions

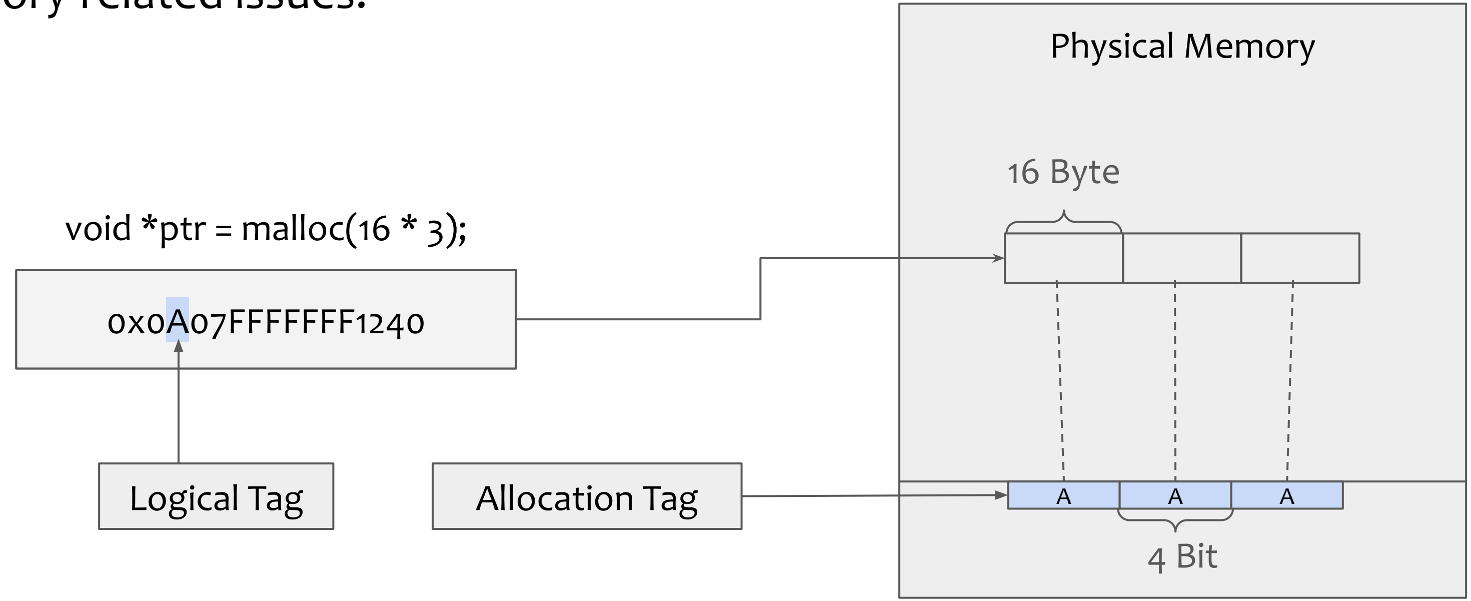
Raphael Dichler
Supervisor: Martin Fink and Ilya Meignan--Masson

2025-04-30

Memory Tagging Extension

In a nutshell

Is part of the ARMv8.5 specification, providing an approach to detect memory-related issues.



Outline

- ~~Memory Tagging Extension~~
- Motivation
- Experiments

Motivation

Strong Security Guarantees

- MTE is a hardware feature designed to detect memory safety violations.

Performance Remains an Open Question

- Unknown impact on memory-bound and multi-threaded workloads.
- Integration in high-performance systems like databases is not well understood.

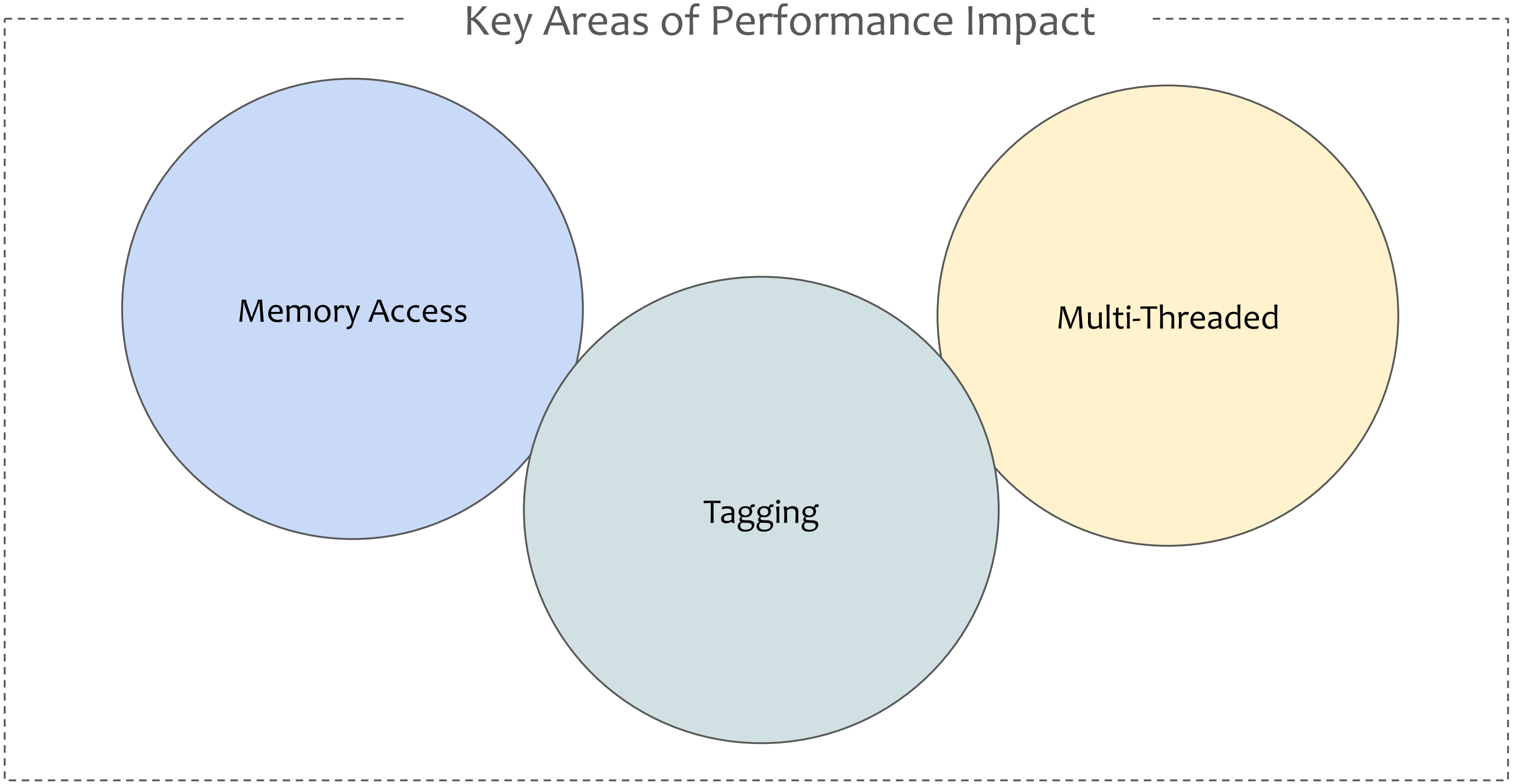
Key Challenges

- Does tag fetching and checking introduce significant overhead?
- Extra steps in allocation and deallocation.
- Effects on caching, concurrency, and thread synchronization are unclear.

Outline

- ~~Memory Tagging Extension~~
- ~~Motivation~~
- Experiments

Experiment



Key Areas of Performance Impact

Experiment: Non-Contiguous Memory Access

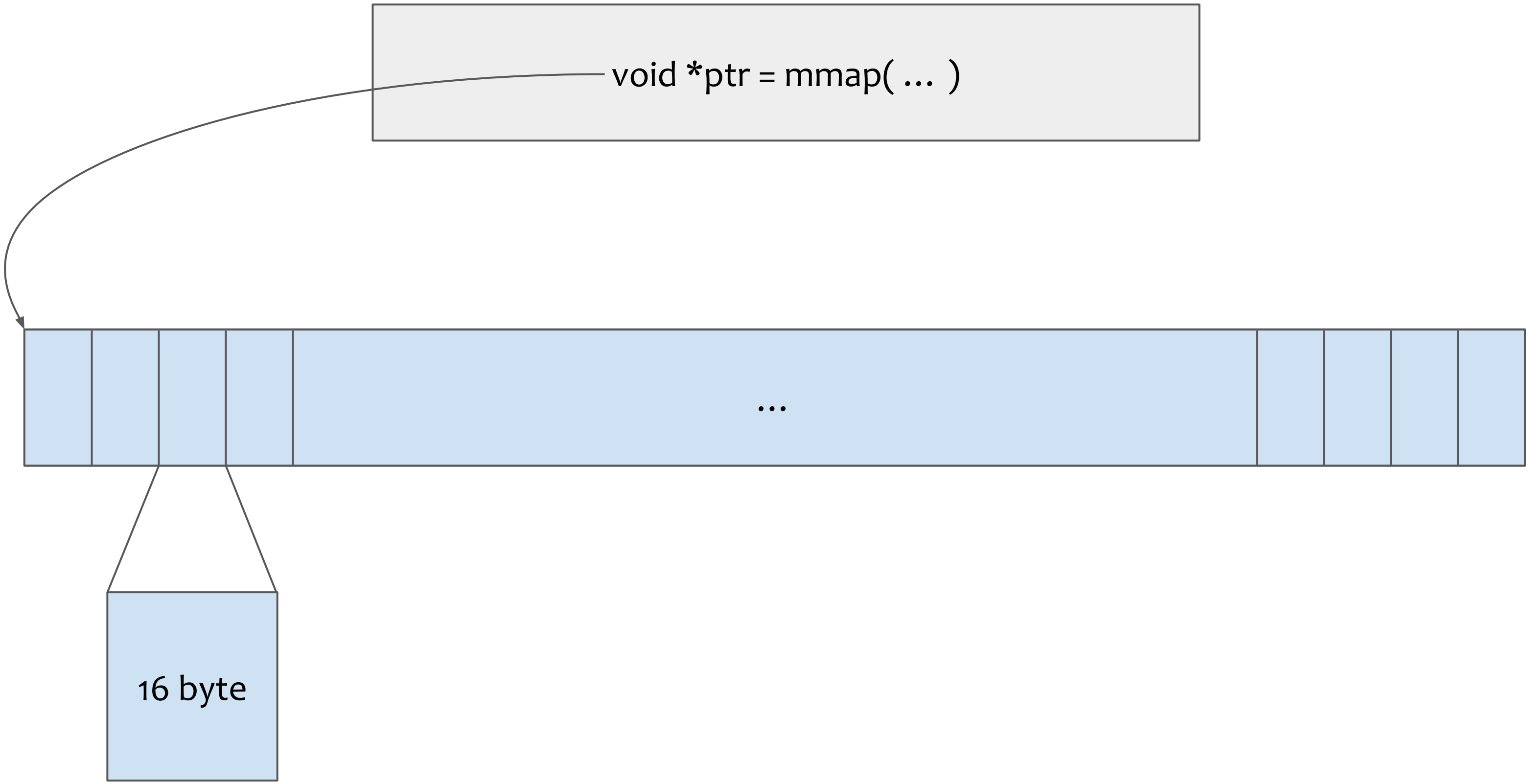
Idea

Analyze the overhead of non-contiguous memory access in the presence of MTE across different caches and main memory levels.

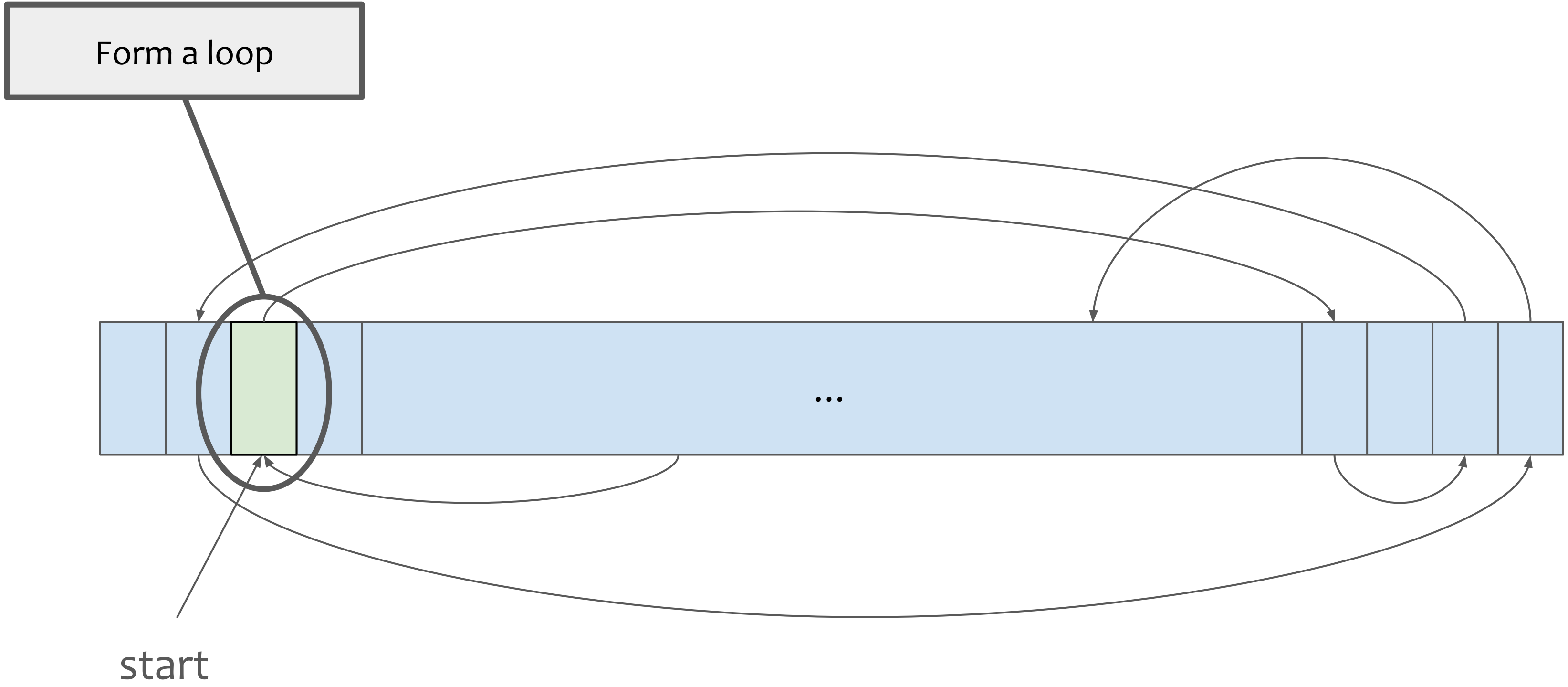
Experiment in a nutshell

1) Construct a linked list, which forms a loop
2) Measure the time to traverse over 100.000.000 nodes
3) Compare MTE-enabled with MTE-disabled

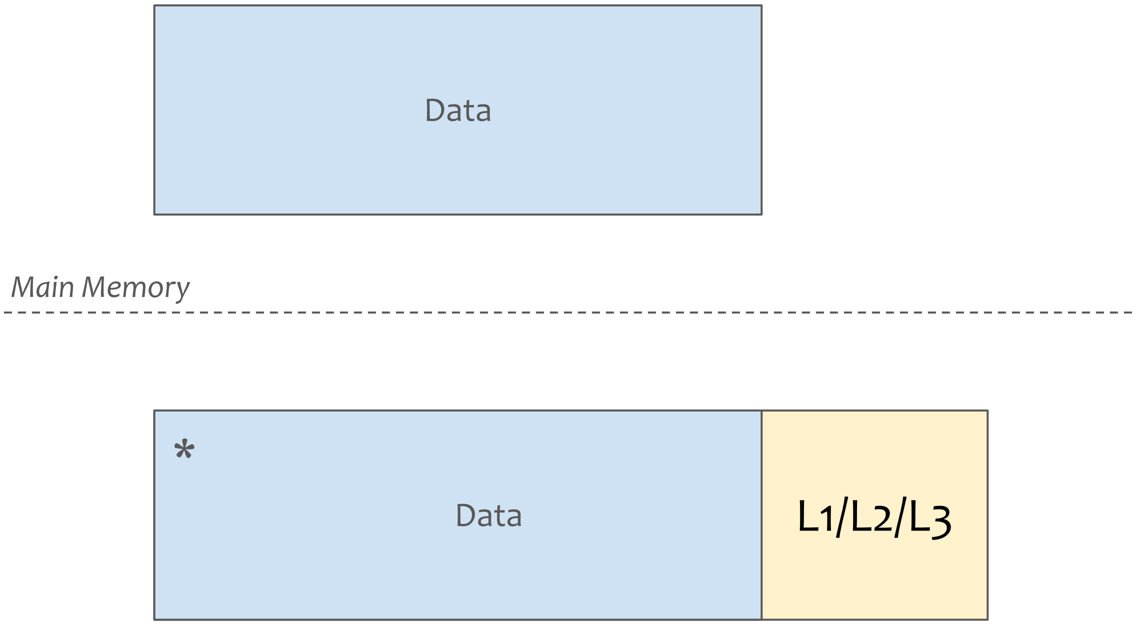
Experiment: Non-Contiguous Memory Access


void *ptr = mmap(…)

Experiment: Non-Contiguous Memory Access



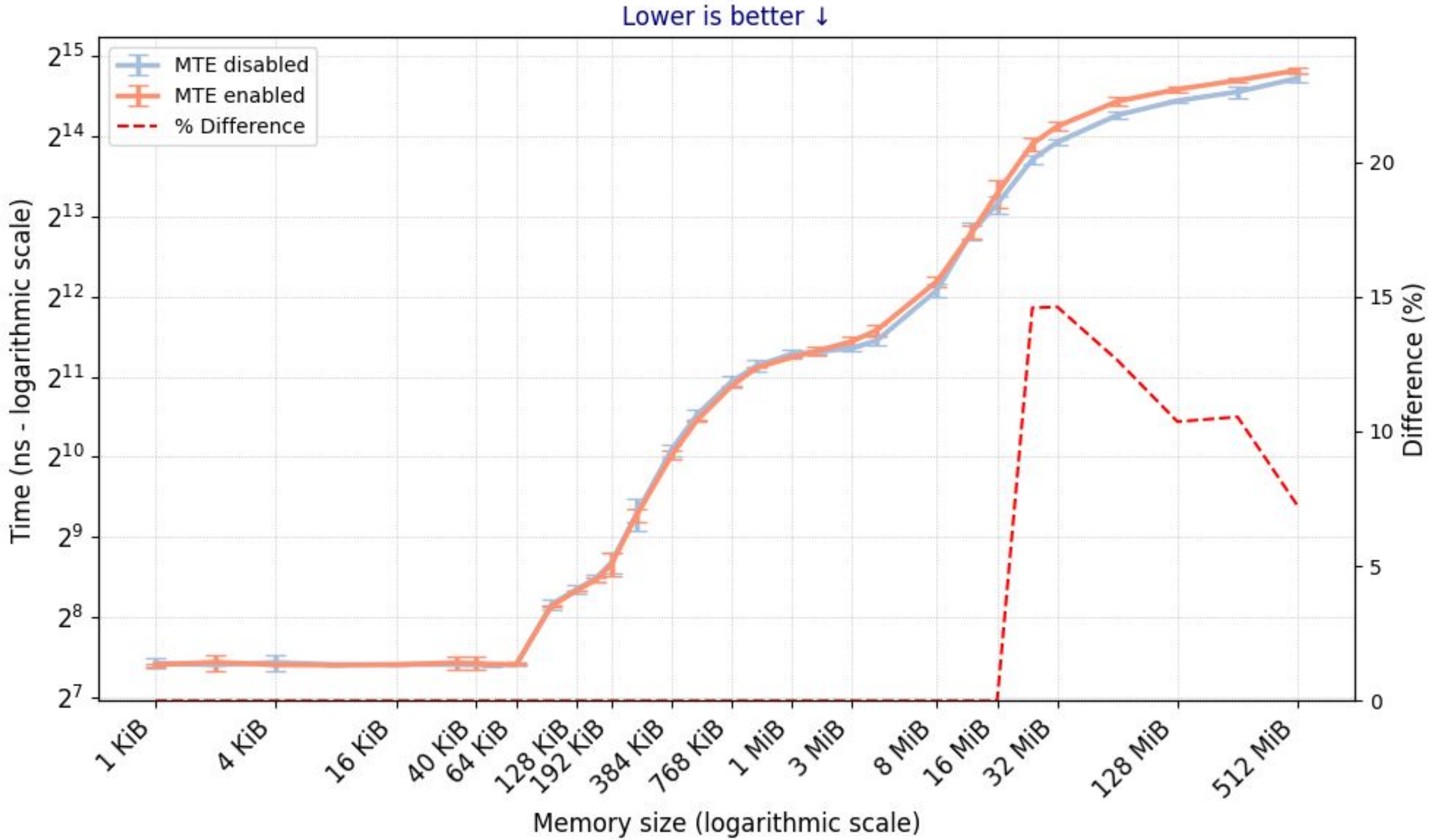
Rationale: Non-Contiguous Memory Access



- Data is small; all fits into L1 Cache
- As long the complete data fits in L1, we see no change in performance
- Holds for all Cache-Levels

* Due to specific cache replacement policies or access patterns, data may be evicted from L1/L2/L3 caches even if there appears to be sufficient space available.

Result: Non-Contiguous Memory Access



Experiment: Contiguous Memory Access

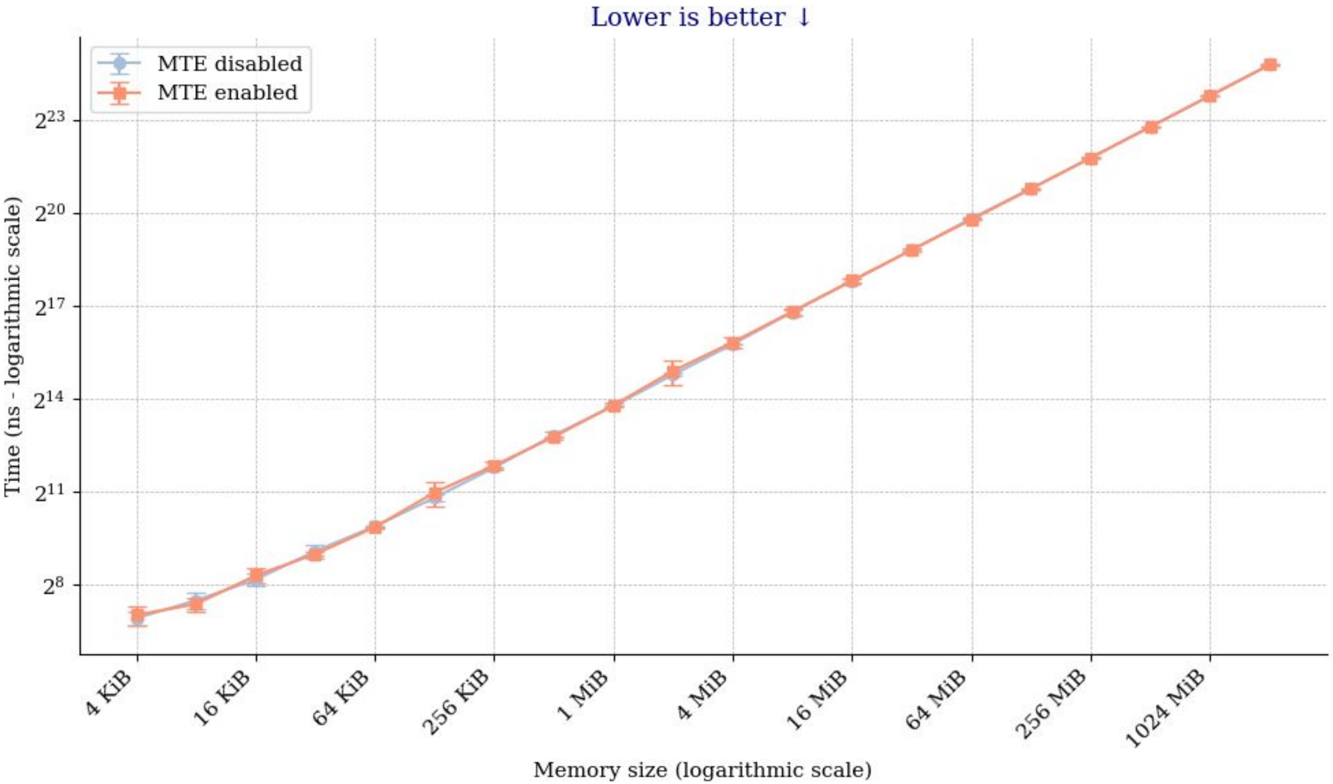
Idea

Analyze the overhead of memory access when accessing memory contiguously in the presence of MTE.

Experiment in a nutshell

1) Allocate a block of contiguous memory
2) Measure the time to iterate once
3) Compare MTE-enabled with MTE-disabled

Result: Contiguous Memory Access



Experiment: Tagging

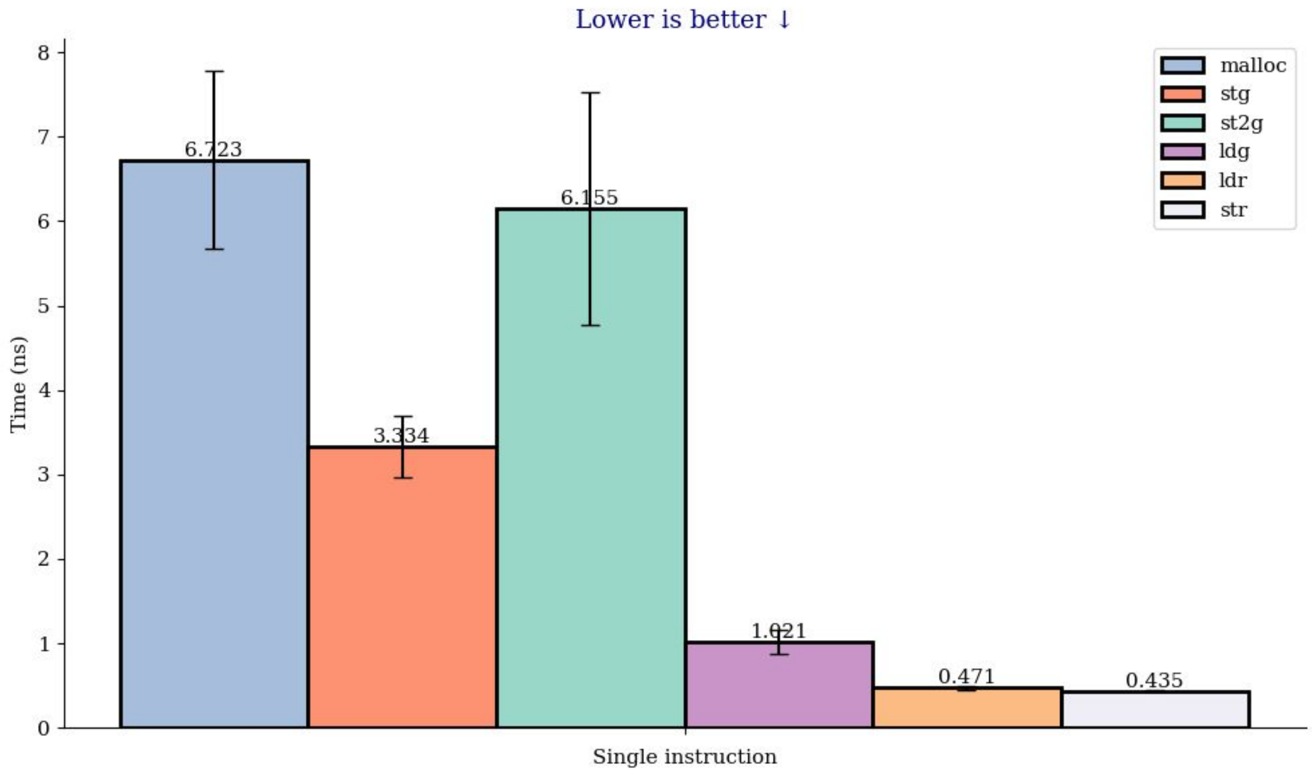
Idea

Compare the execution time to tag memory with load and store.

Experiment in a nutshell

1) Allocate a block of contiguous memory
2) Measure the time of tagging (stg, st2g), loading, or storing data

Result: Tagging



Experiment: Concurrent Read and Write

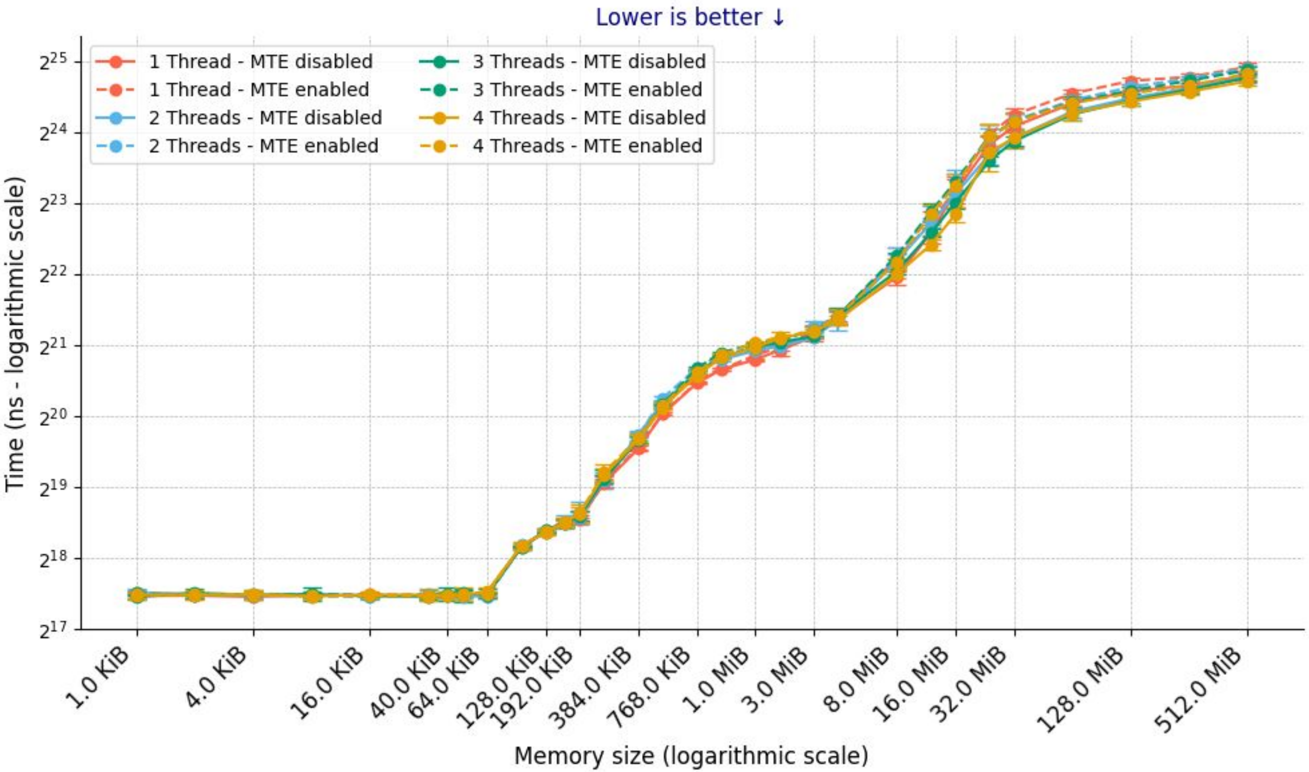
Idea

Analyze the overhead of concurrent read and write when accessing memory in the presence of MTE.

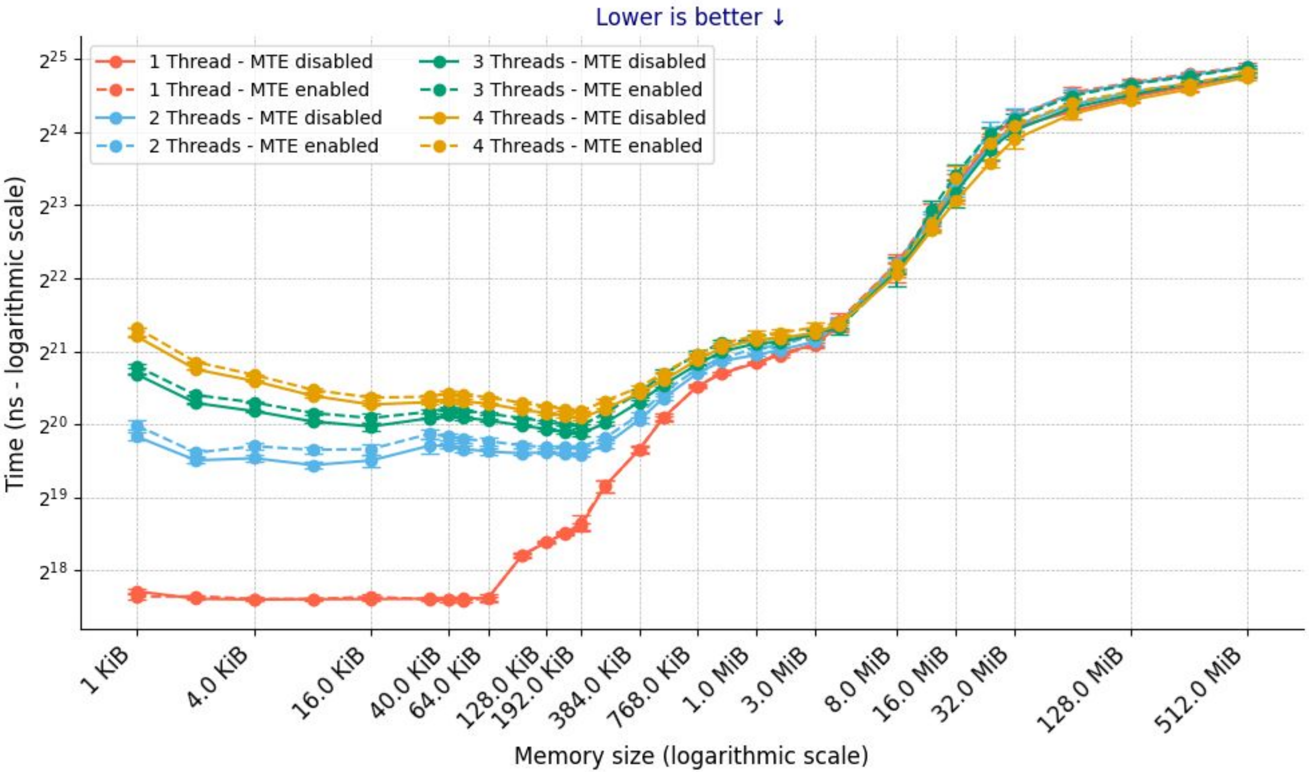
Experiment in a nutshell

1) Construct a linked list like in the non-contiguous memory access experiment.
2) Create multiple threads and pass the start of the linked list to them.
3) Measure the time to traverse over 100.000.000 nodes by each thread

Result: Concurrent Read



Result: Concurrent Write



Outline

- ~~Memory Tagging Extension~~
- ~~Motivation~~
- ~~Experiments~~

Conclusion

Key Areas of Performance Impact

As long data resides inside the **cache or is prefetchable** MTE introduces **no overhead**

If data is **not prefetchable nor located inside** the cache MTE introduces **5-15% overhead**

Tagging takes about **x7 times longer** than load/store operations

Concurrent reads don't introduce any overhead when MTE is enabled

Concurrent writes introduce an overhead of 5-15% when MTE is enabled

Backup

Experiment: Synchronization Operation (CAS)

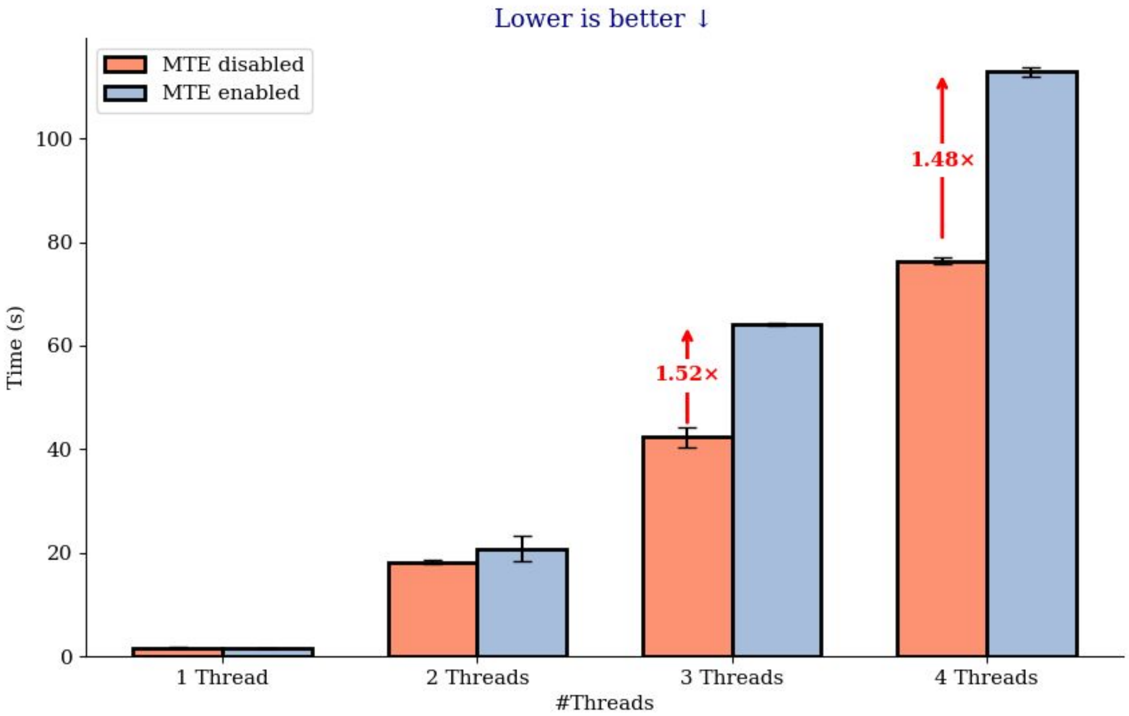
Idea

Analyze the overhead of Compare-And-Swap (CAS) when accessing memory in the presence of MTE.

Experiment in a Nutshell

1) Allocate a block of shared memory.
2) Create multiple threads and pass them the pointer to the memory.
3) Each thread increments the shared value by one 1.000.000 times.
4) Compare MTE-enabled with MTE-disabled

Result: Tagging


Lower is better ↓
MTE disabled
MTE enabled
Time (s)
1.52×
1.48×
1 Thread
2 Threads
3 Threads
4 Threads
#Threads