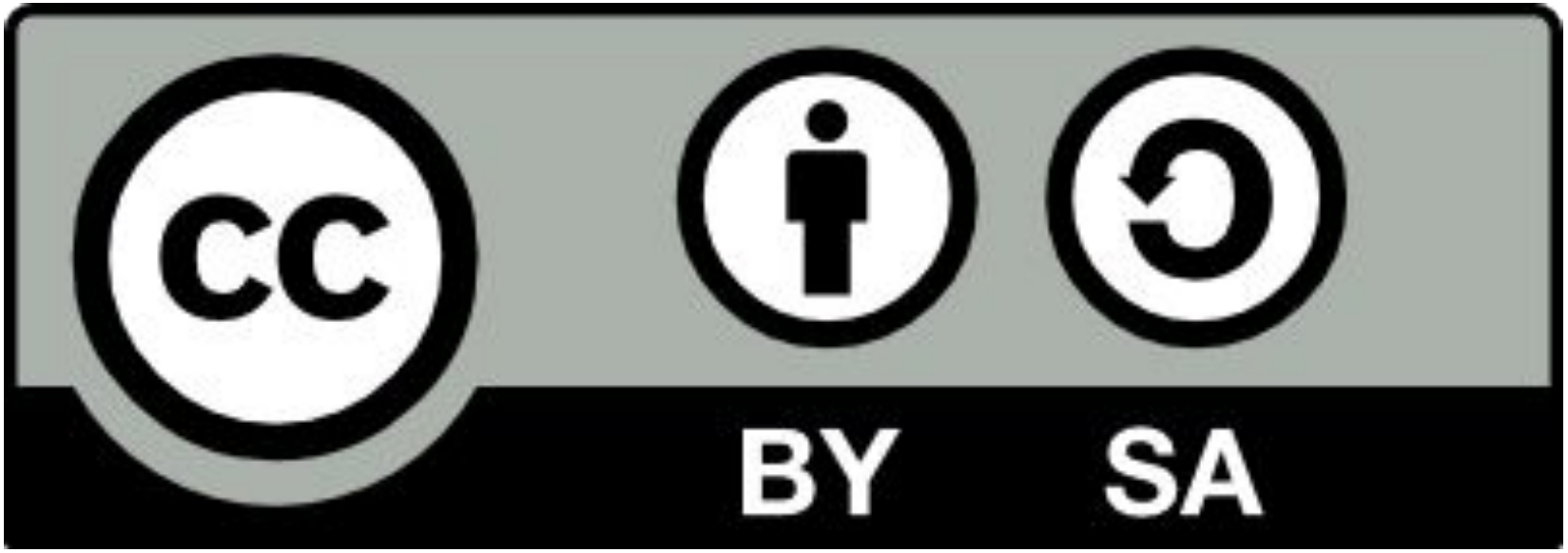
Tracing Data Access Pattern with Bounded Overhead and Best-effort Accuracy

SeongJae Park <sj38.park@gmail.com>

These slides were presented during
The Kernel Summit 2019
(https://linuxplumbersconf.org/event/4/contributions/548/)



I, SeongJae Park <sj38.park@gmail.com>

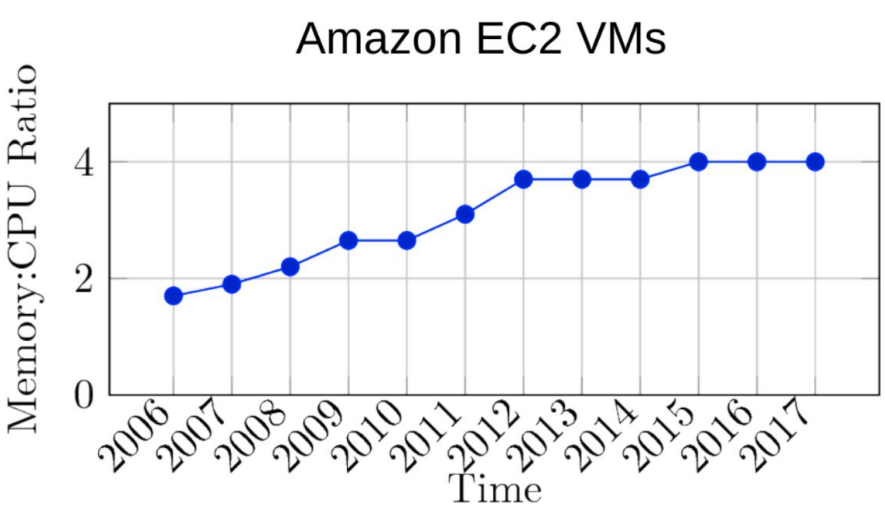
- Interested in memory management and parallel programming for OS kernels
- Recently received a PhD in Seoul National University
- Will start work in Amazon from next week

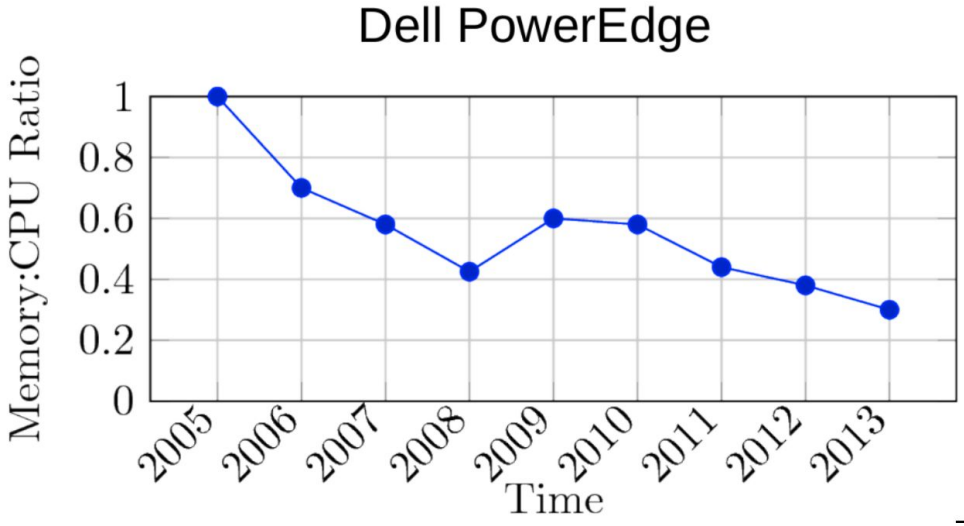
Overview

- Needs for Data Accesses Monitoring
- Existing Schemes and Their Limitations
- DAMON: Data Access MONitor
- Evaluations of DAMON
- Future Plans
- Conclusion

Increasing Working Sets and Decreasing DRAM

- Modern workloads (e.g., cloud, big data, ML) have common characteristics
  - Huge working sets and low locality (different from traditional workloads)
- DRAM capacity is decreasing in relative to the WSS increase
- A survey of Memory:CPU ratio tendency clearly shows this change
  - Increase in virtual machines shows increase of working sets size
  - Decrease in physical machines shows relative decrease of DRAM size
- Therefore, DRAM would not be able to fully accommodates working sets

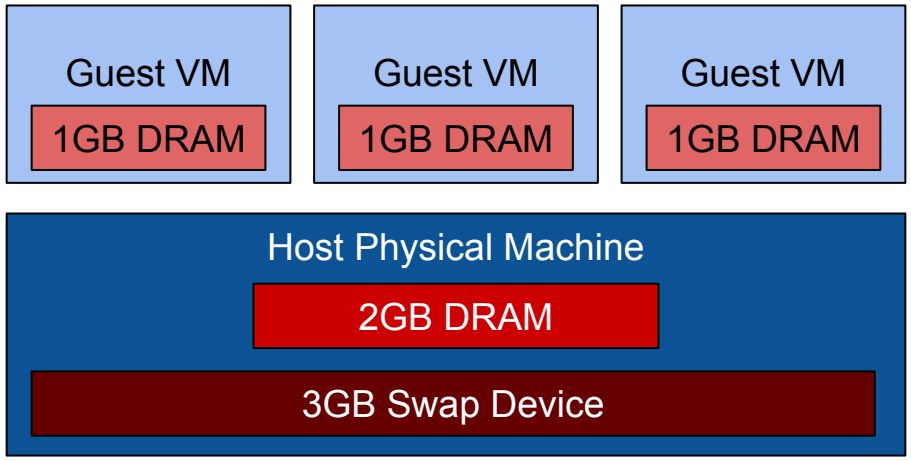




Nitu, Vlad, et al. "Welcome to zombieland: Practical and energy-efficient memory disaggregation in a datacenter." *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018.

Multi-tier Memory Systems Arrives at Last

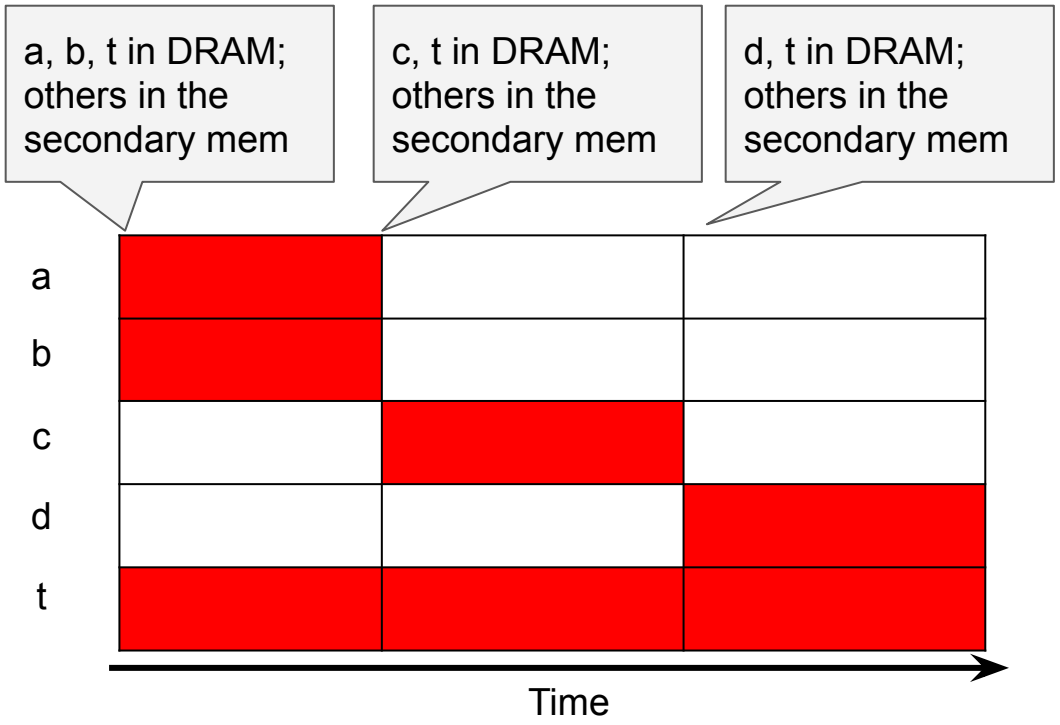
- Storage devices (using FLASH and/or PCM) have continuously evolved
  - Much faster than HDD so that could be used as memory
  - Much larger than DRAM so that could accommodate the huge modern working sets
  - That said, those are obviously slower than DRAM and smaller than HDD
- Consequently, multi-tier memory systems will be prevalent
  - There are similar concepts such as Heterogeneous or hierarchical memory
  - Configure DRAM as main memory and modern fast storage devices as secondary memory
  - SWAP, which has considered harmful in past, would be a straight-forward configuration



How Multi-tier Memory Should be Managed?

- Applications have multiple data objects and multiple execution phase
- Each execution phase has different data access pattern (DAP)
  - The term, DAP in this talk means: Access frequencies of each data object for a given time
- For each phase, objects that will be frequently accessed should be in the main memory while others be in the secondary memory

```
def foo(a,b,c,d):
    t = sum(a, b)
    t = mult(t, c)
    d = t
```

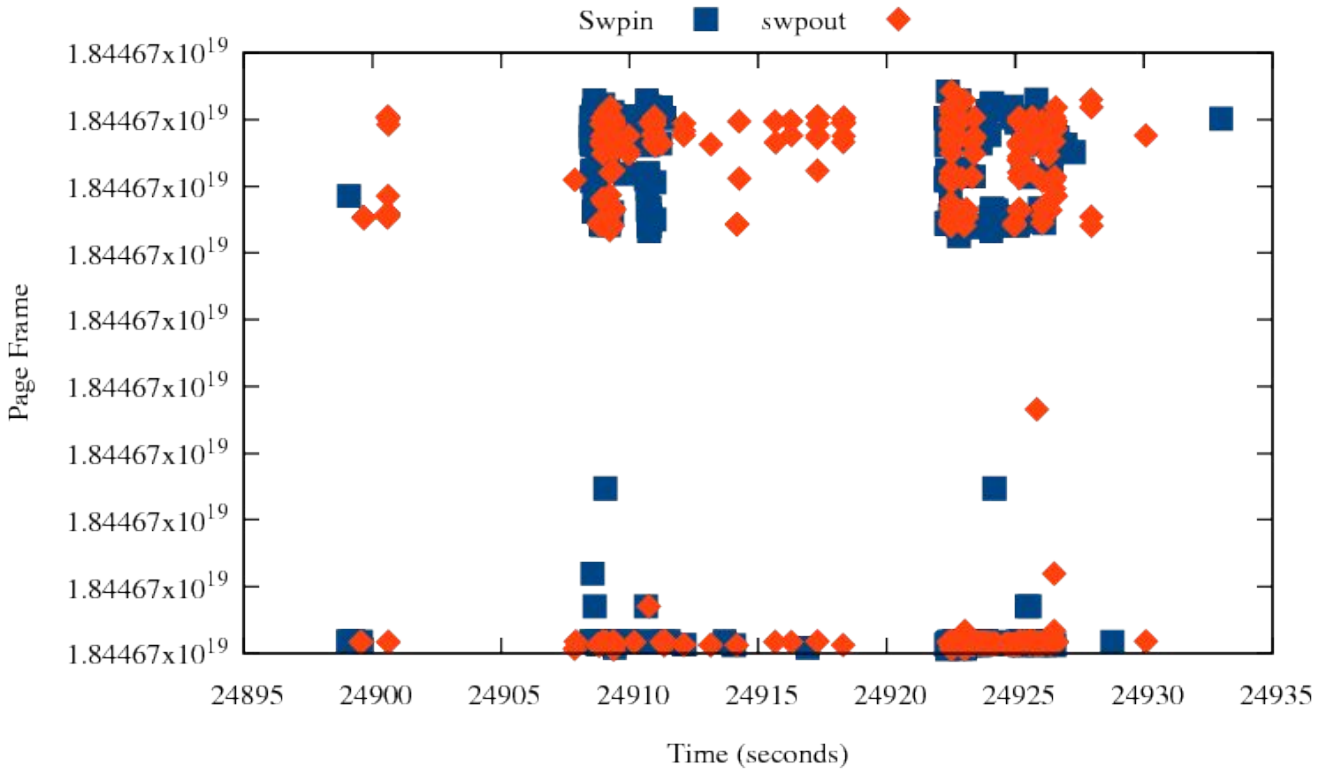


Outdated Memory Management Mechanisms

- Designed upon the old assumptions
  - Assumes small working sets, high locality, and slow secondary memory
- Relies on DAP estimation algorithms based on LRU or simple heuristics
- Needs to be optimized for the modern workloads and memory
- Several problems have reported with solutions
  - Let’s explore such optimization cases
    (2 cases related with kernel space and 1 case related with user space)

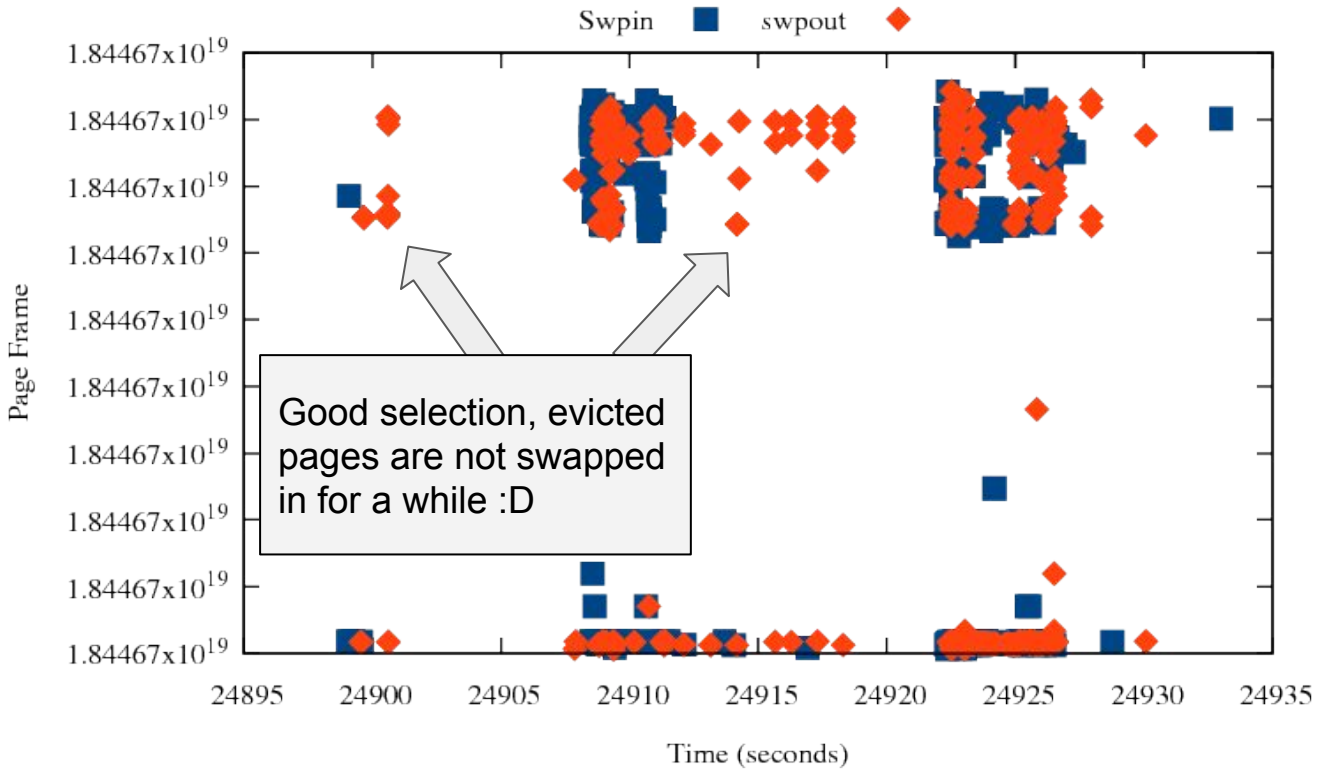
Case 1: LRU-based Page Reclamation

- Linux selects pages to be reclaimed based on a pseudo-LRU algorithm
- Problems
  - Not useful if workloads has no strong locality
  - Modern workloads tend to have complex and dynamic access patterns
  - Swapping out pages to be used in near future degrades performance



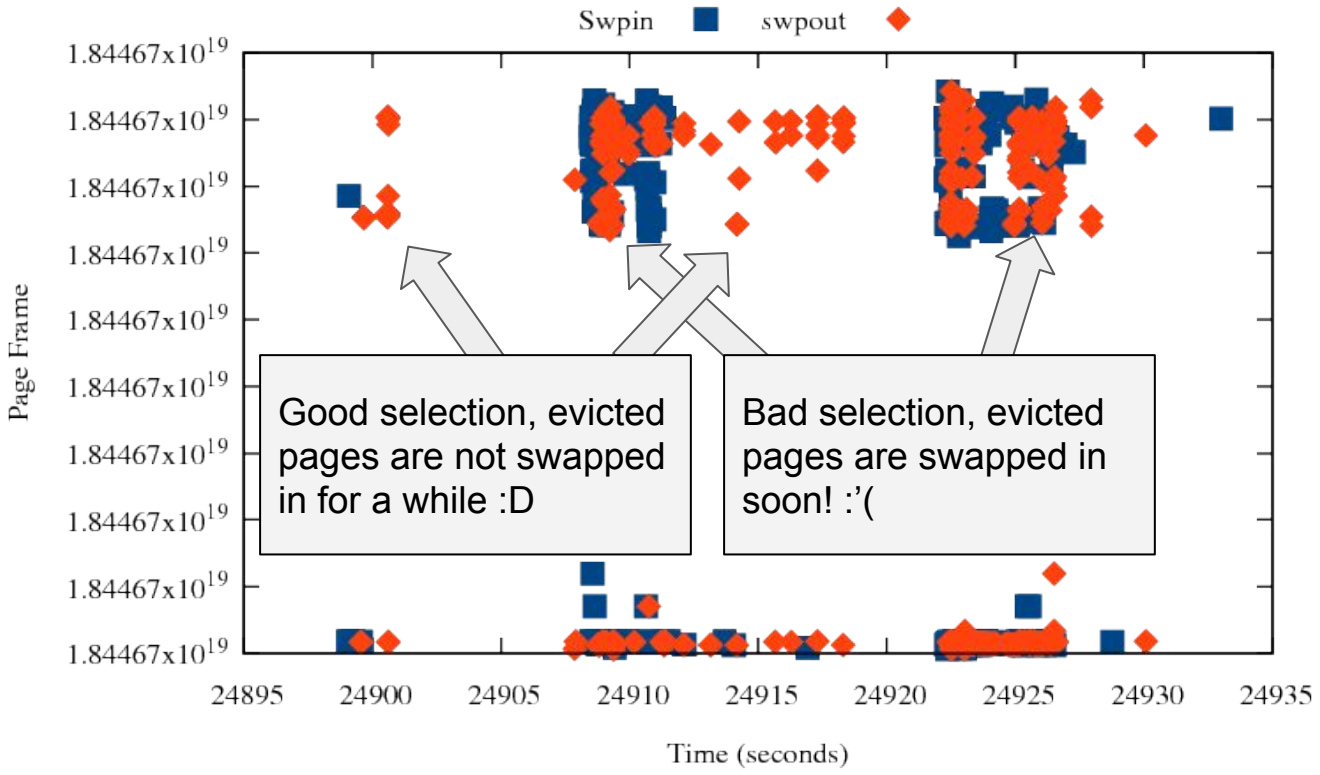
Case 1: LRU-based Page Reclamation

- Linux selects pages to be reclaimed based on a pseudo-LRU algorithm
- Problems
  - Not useful if workloads has no strong locality
  - Modern workloads tend to have complex and dynamic access patterns
  - Swapping out pages to be used in near future degrades performance



Case 1: LRU-based Page Reclamation

- Linux selects pages to be reclaimed based on a pseudo-LRU algorithm
- Problems
  - Not useful if workloads has no strong locality
  - Modern workloads tend to have complex and dynamic access patterns
  - Swapping out pages to be used in near future degrades performance



Case 1: LRU-based Page Reclamation

- Linux selects pages to be reclaimed based on a pseudo-LRU algorithm
- Problems
  - Not useful if workloads has no strong locality
  - Modern workloads tend to have complex and dynamic access patterns
  - Swapping out pages to be used in near future degrades performance
- A proposed DAP-based optimization: proactive reclamation[1,2]
  - Identify cold pages (idle for >2 minutes) and reclaim those to secondary memory (ZRAM)
  - Use page table access bits for the identification of the cold pages
  - Successfully applied to Google internal clusters
  - Nevertheless, the idle page identification overhead is not trivial; one dedicated CPU required

[1] Lagar-Cavilla, Andres, et al. "Software-Defined Far Memory in Warehouse-Scale Computers." *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019.
[2] Jonathan Corbet, “Proactively reclaiming idle memory.” https://lwn.net/Articles/787611/

Case 2: Aggressive THP Promotions

- THP does promotions and demotions between regular pages and huge pages
- Problems
  - The selection of candidate pages to be promoted are very simple; It aggressively promotes any page as long as 2 MiB contiguous memory region can be allocated
  - If promoted huge pages are not fully accessed, memory is wasted
- A proposed DAP-based optimization, Ingens[1]
  - Start from a regular page and promotes to a huge page if adjacent 2MiB are really used
  - Demotes a huge page into 512 regular pages if those are not fully accessed
  - Track accesses using page table accessed bits, meaning arbitrarily increasing overhead

[1] Kwon, Youngjin, et al. "Coordinated and efficient huge page management with ingens." 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 2016.

Case 3: `madvise()` Improvements

- User programs are allowed to voluntarily report their data access pattern hints
  - Programmers would know the data access pattern of their program, right?
    - Sadly, not always, especially in case of huge and complex program
  - `mlock(), madvise(), ...`
  - Proper use of the system calls significantly improves the performance under memory pressure
- Problems
  - Hints are not rich enough to be useful for the modern systems
  - Allows only voluntary reports
- A proposed optimization: memory hinting API for external process[1]
  - Implements more hints: MADV_COLD and MADV_PAGEOUT
  - Allows external processes to be able to report other's notification
  - Successfully adopted for Android phones
  - Just leaves the data access pattern monitoring task to user space

[1] Minchan Kim, "introduce memory hinting API for external process."
https://lore.kernel.org/lkml/20190531064313.193437-1-minchan@kernel.org/

The Unsolved Problem: Data Access Monitoring

- All the three optimizations we explored have no solution for Scalable, efficient, and effective data accesses monitoring
- Proactive reclamation and Ingens have unbounded monitoring overhead
  - In other words, those are not scalable
  - It would be hard to be merged into upstream
  - LSFMM19 discussions about proactive reclamation also pointed the scalability problem
- madvise() improvements simply leaves the task to users

Overview

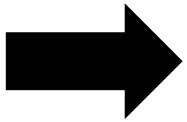
- Needs for Data Accesses Monitoring
- Existing Schemes and Their Limitations
- DAMON: Data Access MONitor
- Evaluations of DAMON
- Future Plans
- Conclusion

Workload Instrumentation

- Install a hook function, which will be invoked for every memory access, into target workloads in compile time or using binary reversing techniques
- The hook function then records every related information
  - Including Time, Type of the access, the access target address, the value written to or read by
- Pros
  - Can monitor all informations of every memory access
  - Can be used for various purposes including safety checks
- Cons
  - Incurs high overhead
    - A workload having 10 minutes of runtime and 600 MiB of working sets (433.milc) consumed more than 24 hours of CPU time and 500 GiB storage spaces when an instrumentation based memory monitoring scheme (PIN) is applied

```
r1 = X; /* LOAD r1, X */

Y = r2; /* STORE r2, Y */
```



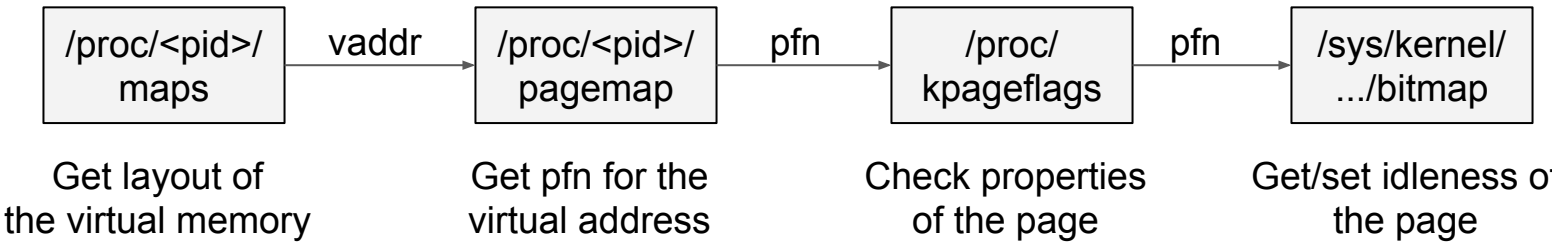
```
record_load(X, r1);
r1 = X; /* LOAD X, r1 */
record_store(Y, r2);
Y = r2; /* STORE Y, r2 */
```

Page Table Accessed-bit Tracking

- When a page is accessed, the ‘`accessed`’ bit of the PTE for the page is set
- By periodically clearing and monitoring the bits for the target workloads, we can monitor memory accesses
- Many approaches including the proactive reclamation and Ingens use this
- Pros
  - Incurs lower overhead compared to the workload instrumentation
  - Simple to be applied
- Cons
  - Cannot monitor every information of every accesses
    - Know only existence of accesses
    - That said, not a big problem for multi-tier memory focused performance optimization
  - Overhead arbitrarily increases as the size of the target working sets grows

Idle Page Tracking

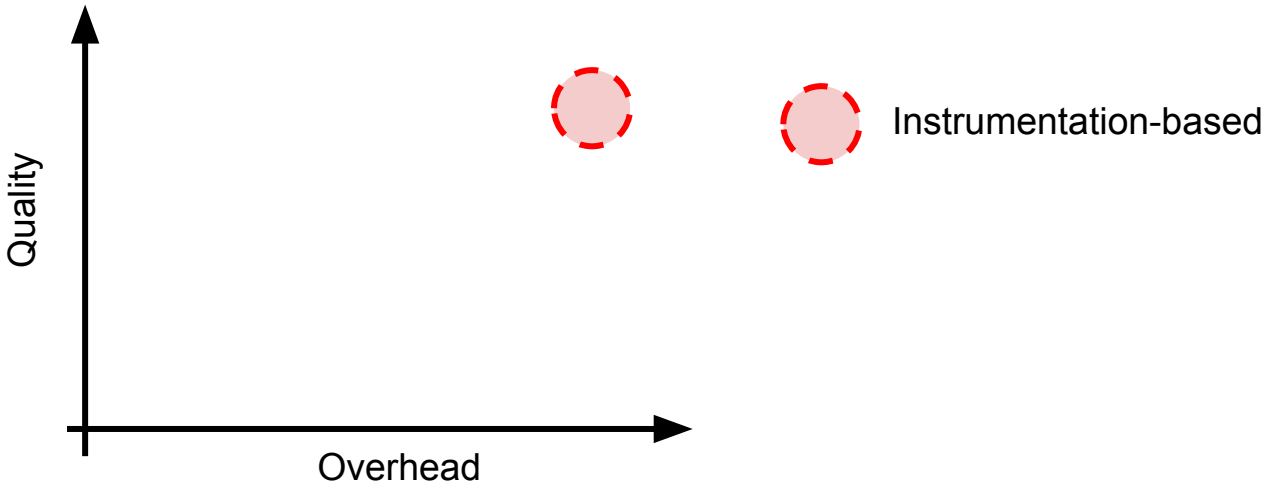
- Page table accessed bit tracking based Linux kernel feature
- Allow users to clear and monitor the access bits using a sysfs interface
  - Points a page using pfn
- Complicated to be used from user space
  - Need to read and write up to 4 files (maps, pagemap, kpageflags, and idle_page_tracking/bitmap)
    - Finally read/write idle_page_tracking/bitmap
    - I tried this to measure WSS[1] and found that I am using wrong tool for me
    - A few of improvements for better interface have proposed[2]
- Of course, the overhead arbitrarily increases because this feature is using the page table access bit tracking



[1] SeongJae Park, “Idle Page Tracking Tools.” https://sjp38.github.io/post/idle_page_tracking/
[1] Joel Fernandes, “[PATCH v1 1/2] mm/page_idle: Add support for per-pid page_idle using virtual indexing.” https://lore.kernel.org/lkml/20190722213205.140845-1-joel@joelfernandes.org/

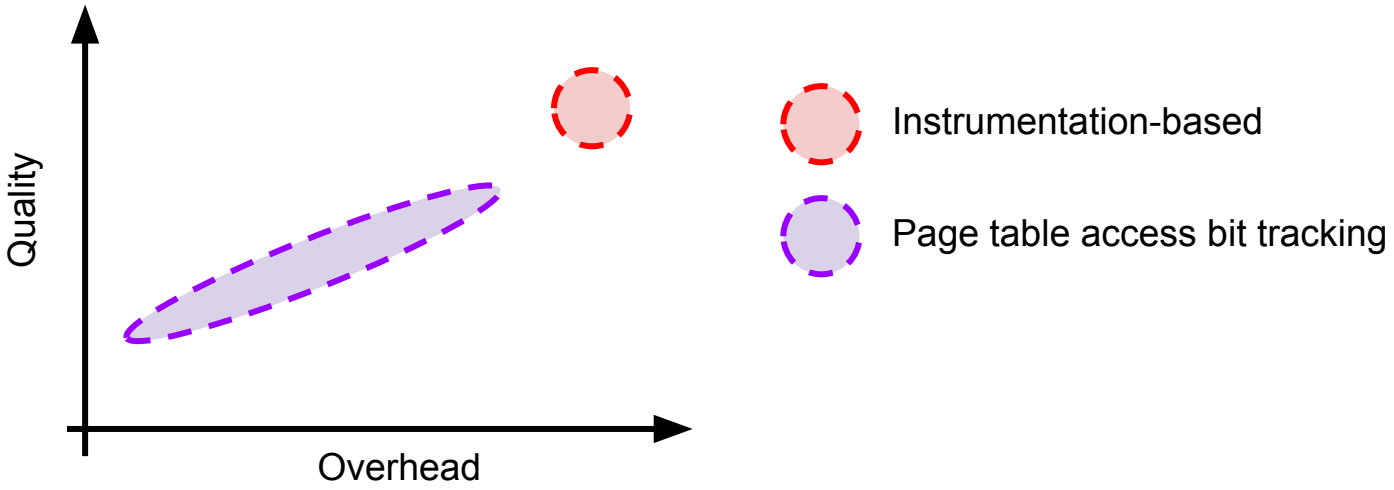
Existing Memory Access Monitoring Schemes

- Instrumentation-based monitoring
  - Incurs unacceptably high overhead for unnecessarily detailed information



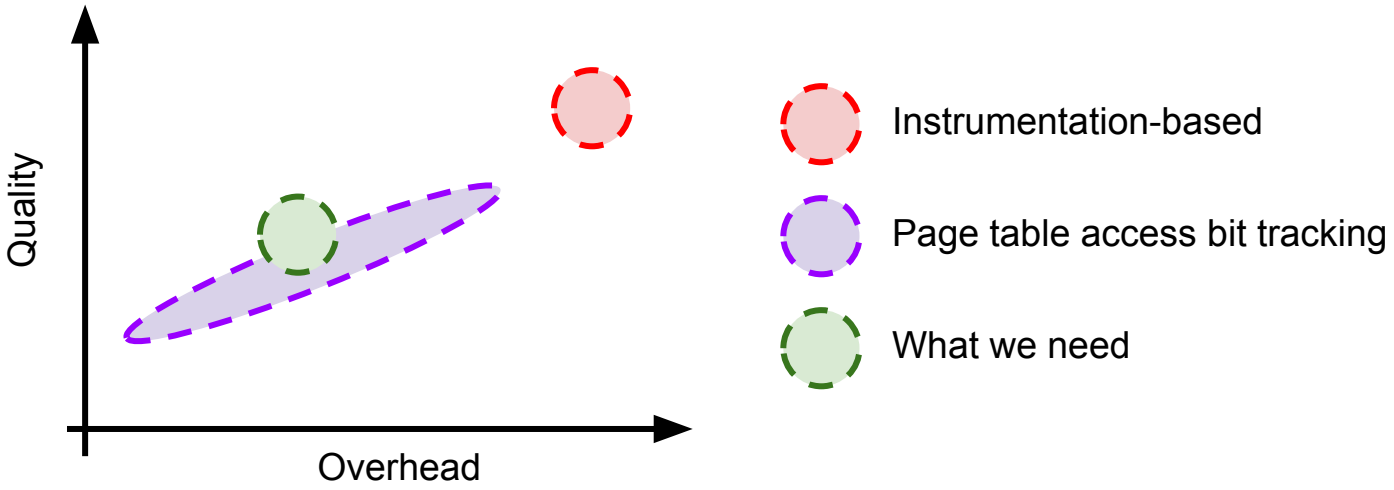
Existing Memory Access Monitoring Schemes

- Instrumentation-based monitoring
  - Incurs unacceptably high overhead for unnecessarily detailed information
- Page table access bit tracking
  - The overhead arbitrarily grows as size of target workload increases (Unscalable for huge working sets)



Existing Memory Access Monitoring Schemes

- Instrumentation-based monitoring
  - Incurs unacceptably high overhead for unnecessarily detailed information
- Page table access bit tracking
  - The overhead arbitrarily grows as size of target workload increases
    (Unscalable for huge working sets)
- For performance optimization of modern memory workloads, we need…
  - Monitoring overhead that is bounded regardless of the size of the target workloads
  - Quality that able to be used for the performance optimization



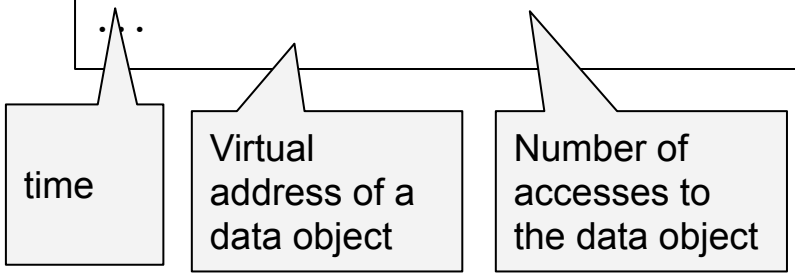
Overview

- Needs for Data Accesses Monitoring
- Existing Schemes and Their Limitations
- DAMON: Data Access MONitor
- Evaluations of DAMON
- Future Plans
- Conclusion

DAMON: Data Accesses MONitor

- A kernel subsystem for data accesses monitoring that I am developing
- Allow users to set upper-bound monitoring overhead while preserving the quality of the outputs
- Aims to be used from both user space and kernel space
- Receives a pid of a target process and returns data access patterns of the processes in real-time
  - Again, the definition of DAP in this talk: Access frequencies of each data object

```
0-1s <0x0001-0x0010, 10>, <0x0010-0x100, 5>,   <0x100-0x200, 1000>
1-2s <0x0001-0x0010, 0>,  <0x0010-0x100, 200>, <0x100-0x200, 200>
2-3s <0x0001-0x0010, 10>, <0x0010-0x100, 200>, <0x100-0x200, 200>
3-4s <0x0001-0x0010, 0>,  <0x0010-0x100, 0>,   <0x100-0x200, 200>
...
```

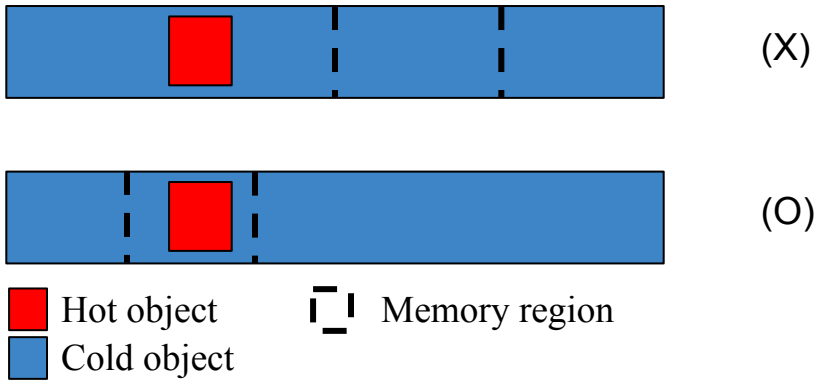


Region-based Sampling

- Damon defines data objects in access pattern oriented way
  - A data object is a memory region that all page frames in the region have a similar hotness
  - By the definition, if a page in a data object is accessed, every other pages in the region for the object has probably accessed and vice versa
- Owing to the definition, DAMON samples only one page per each region instead of every page while preserving the quality of monitoring
  - The monitoring overhead of DAMON becomes in proportion to the number of regions
  - DAMON thus let users to set the maximum number of regions for the upper-bound overhead

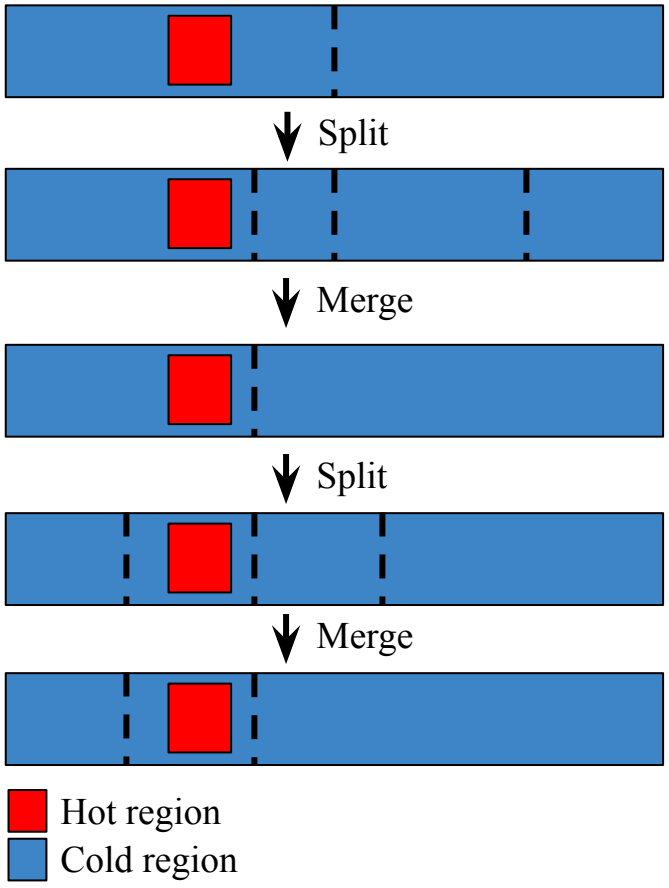
Region-based Sampling

- Damon defines data objects in access pattern oriented way
  - A data object is a memory region that all page frames in the region have a similar hotness
  - By the definition, if a page in a data object is accessed, every other pages in the region for the object has probably accessed and vice versa
- Owing to the definition, DAMON samples only one page per each region instead of every page while preserving the quality of monitoring
  - The monitoring overhead of DAMON becomes in proportion to the number of regions
  - DAMON thus let users to set the maximum number of regions for the upper-bound overhead
- Nonetheless, the quality cannot be preserved if the regions are not well constructed as defined above



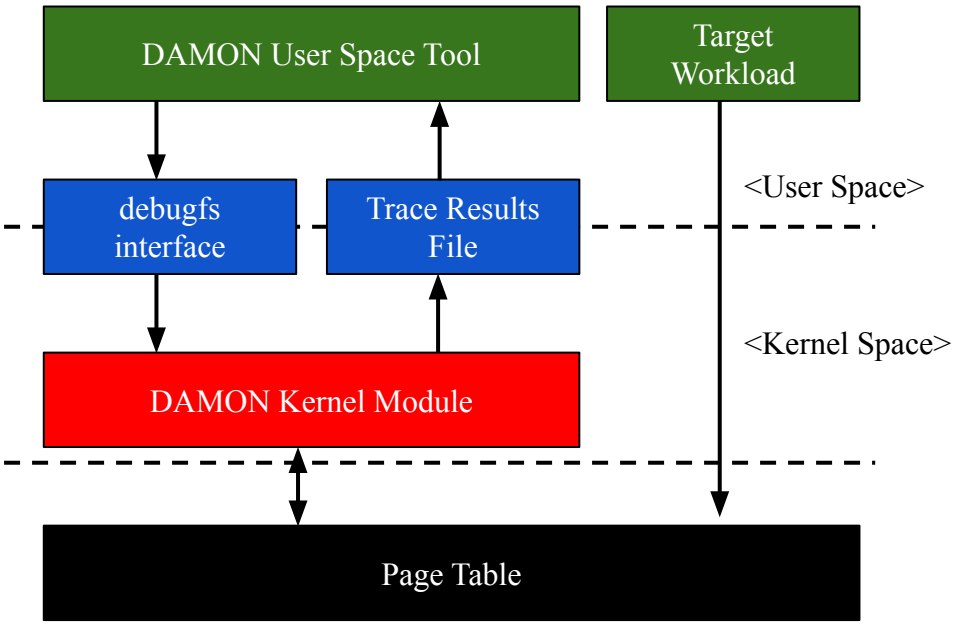
Adaptive Regions Adjustment

- Starts with regions covering entire virtual memory areas of the target workload; Periodically check vma updates and apply the change
- Periodically, merges adjacent regions having similar access frequencies to one region
- After the merge process, splits each region into two regions in random size ratio if the number of regions is smaller than half of the maximum number of regions
- If a split was meaningless, next merge process will merge them
- If a split was meaningful, next merge process will not merge that
- This procedure will find the answer at last



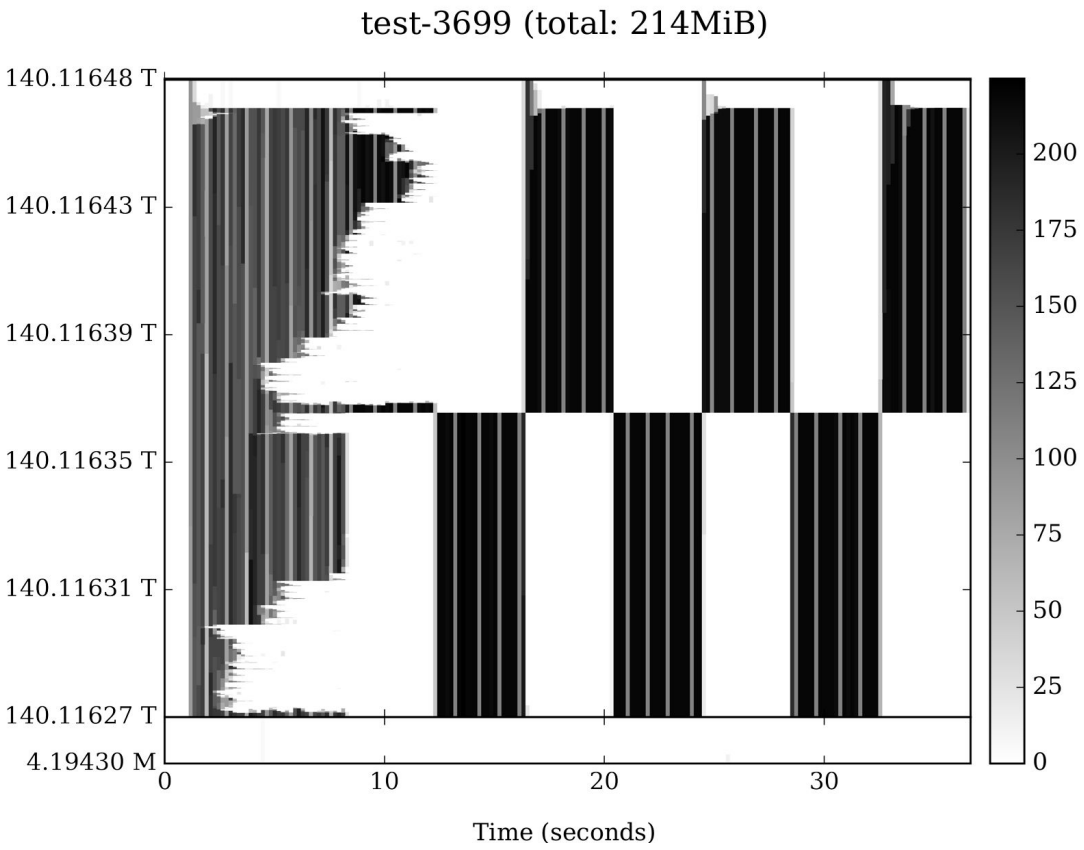
DAMON Prototype and a Toolset Implementation

- DAMON prototype is implemented as a kernel module, of course
  - 1200 lines of code and 529 lines of kunit based test code
  - Provides an interface for user space using debugfs and regular fs
  - Interface for kernel space is not implemented yet
- For convenient use and evaluation of DAMON, my colleague, Yunjae, implemented several user space tools
  - Provides human-friendly interfaces, access pattern visualization, and output analysis



Demo Time!

- Video Link: https://asciinema.org/a/268004
- Result of the last command is as below:



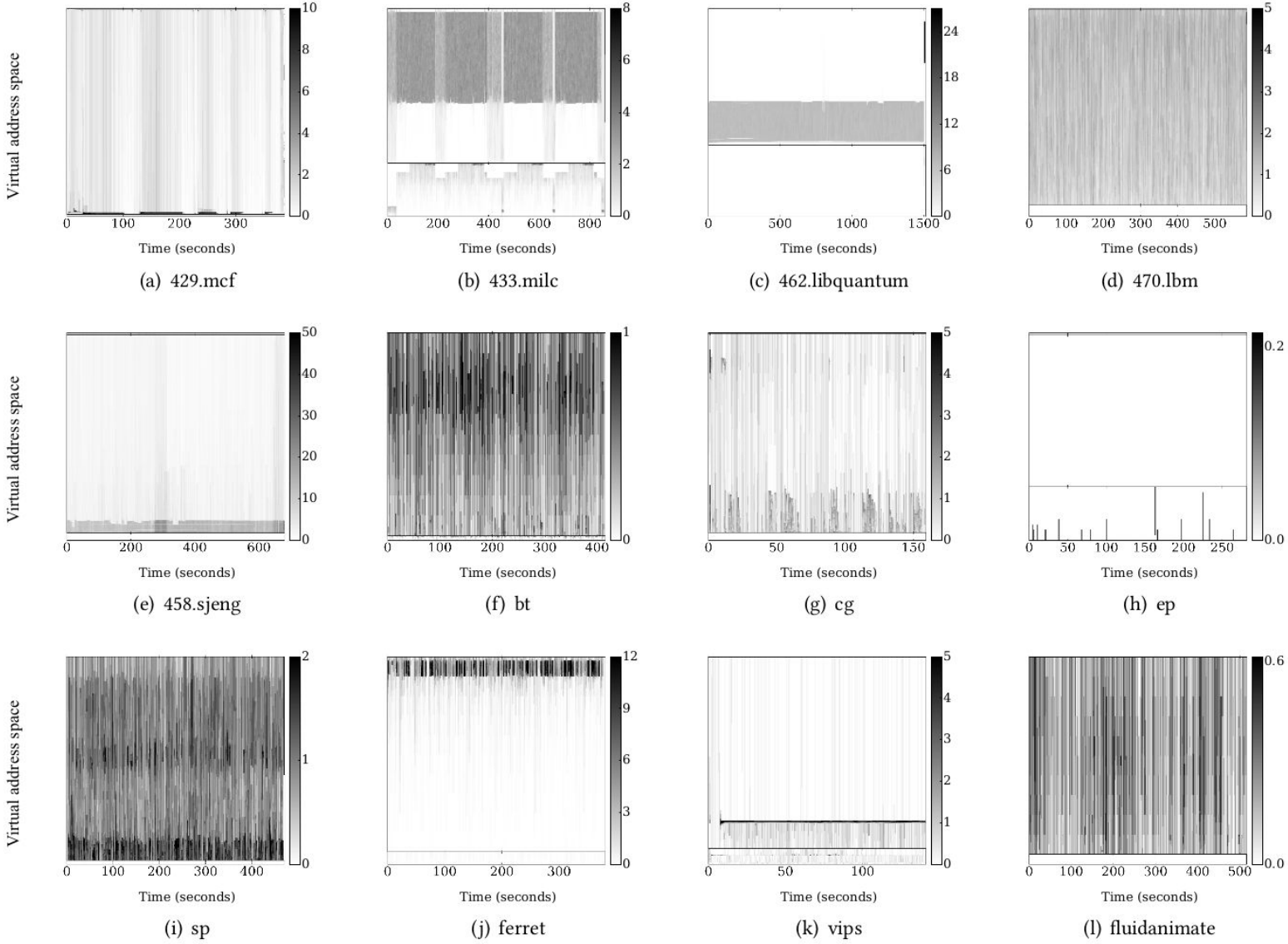
Overview

- Needs for Data Accesses Monitoring
- Existing Schemes and Their Limitations
- DAMON: Data Access MONitor
- Evaluations of DAMON
- Future Plans
- Conclusion

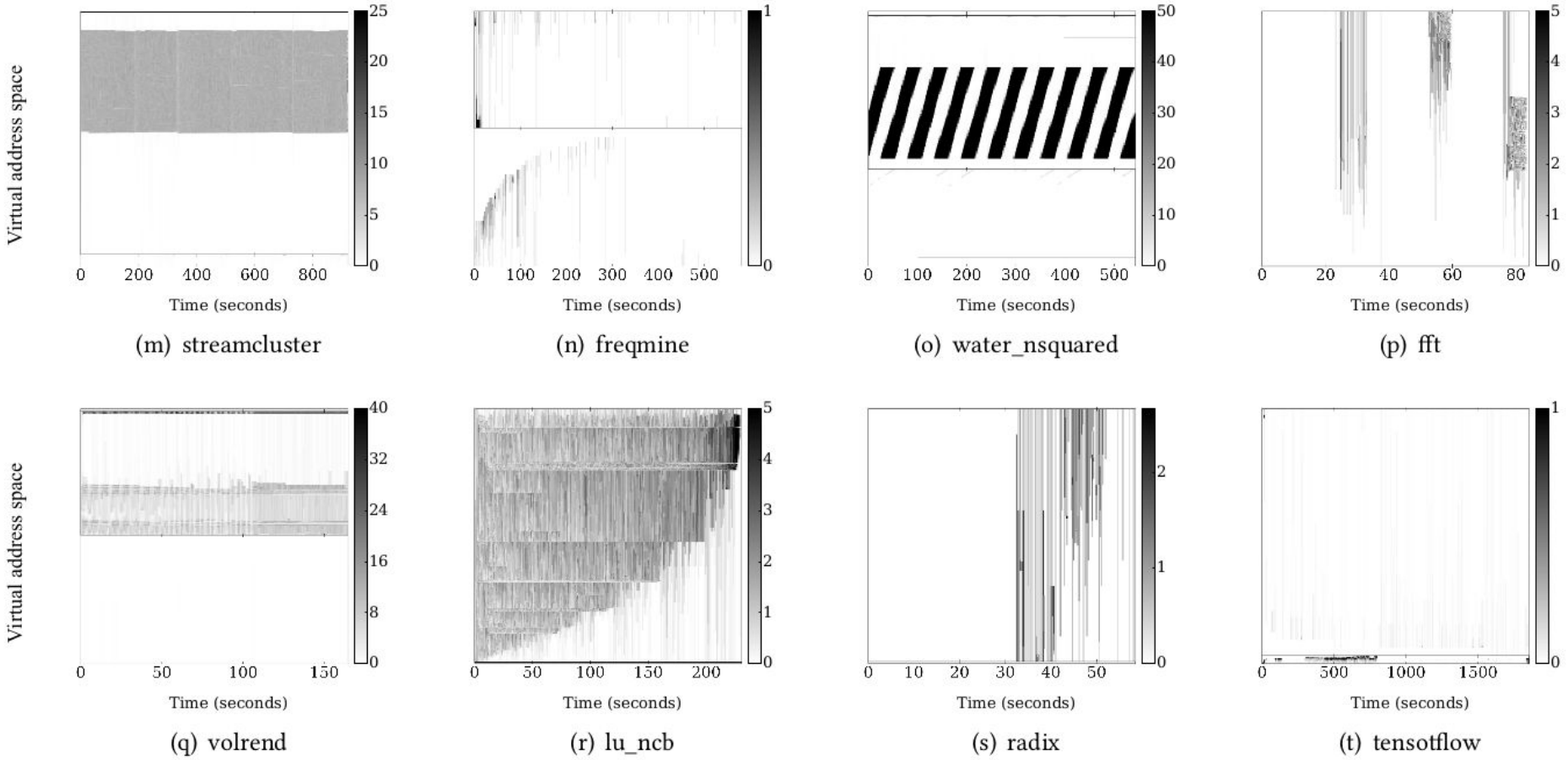
Evaluation Setup

- Server machine
  - Intel Xeon E7-8837
  - 128 GB DRAM
  - Ubuntu 18.04 LTS Server
  - Linux kernel v4.20
- Workloads
  - SPEC CPU 2006, NAS, Tensorflow Benchmark, SPLASH-2X, and PARSEC 3
- DAMON configuration
  - Maximum number of regions: 1000
  - Sampling interval: 1 millisecond
  - Aggregation interval: 100 millisecond

Visualized Data Access Patterns (1/2)

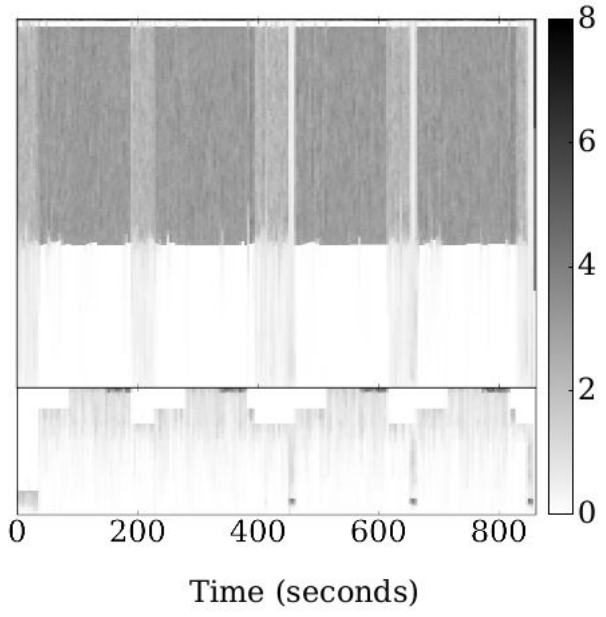


Visualized Data Access Patterns (2/2)

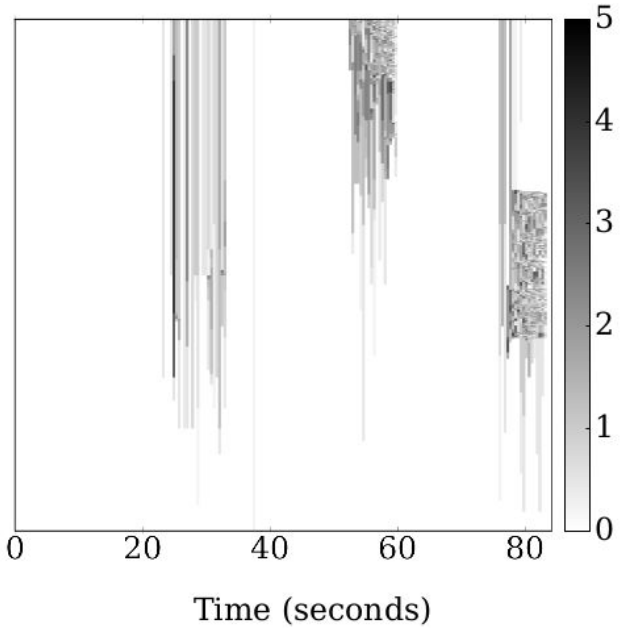


Clear Access Patterns for each Phase

- 433.milc and fft shows clearly different access patterns for each phases



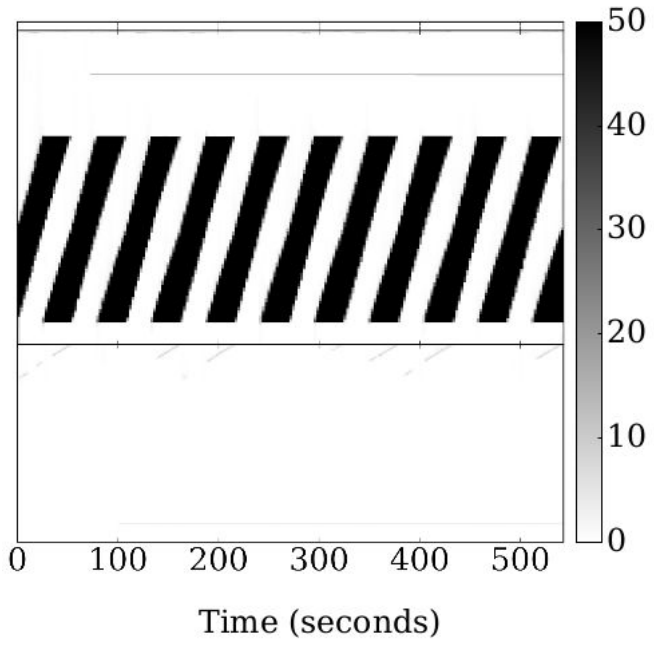
433.milc



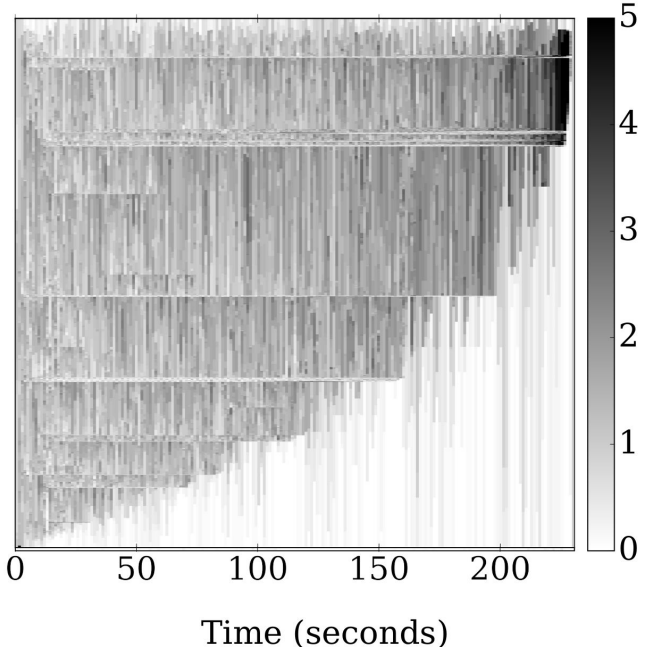
fft

Sequential Accesses

- Splash-2x and lu_ncb shows sequential accesses to specific objects



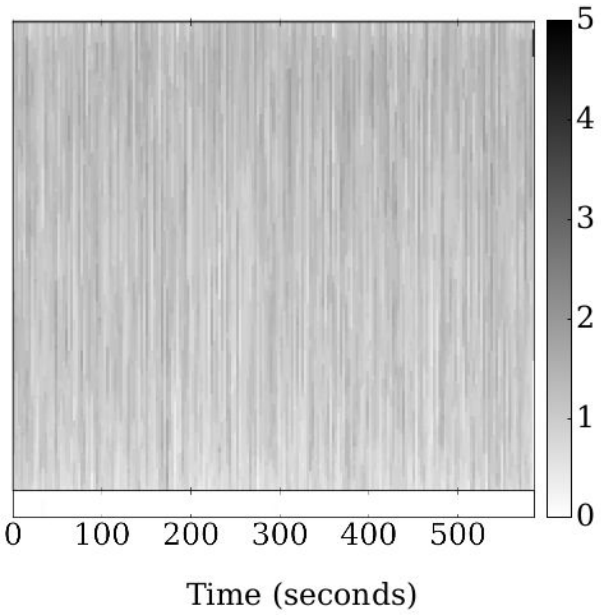
splash-2x



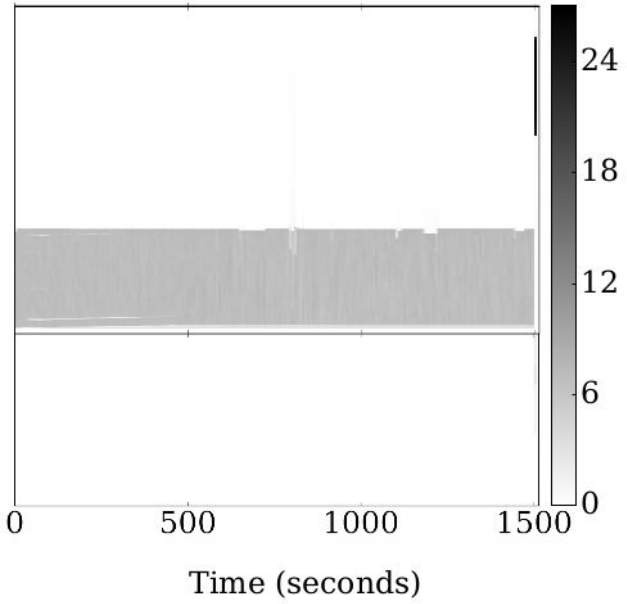
lu_ncb

Uniform Data Access Pattern

- 470.lbm and 462.libquantum shows uniform access pattern to specific objects



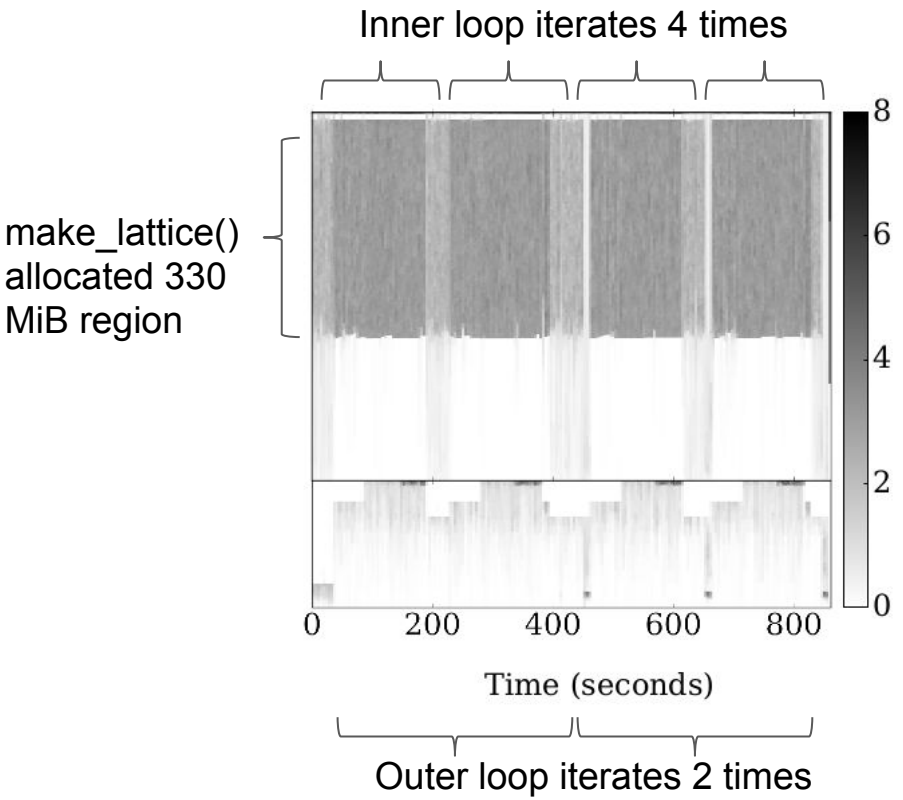
470.lbm



462.libquantum

Detailed Analysis of 433.milc

- Main function's big loops shown in the visualized access pattern
- Upperside hot region of 330 MiB was allocated by make_lattice() function

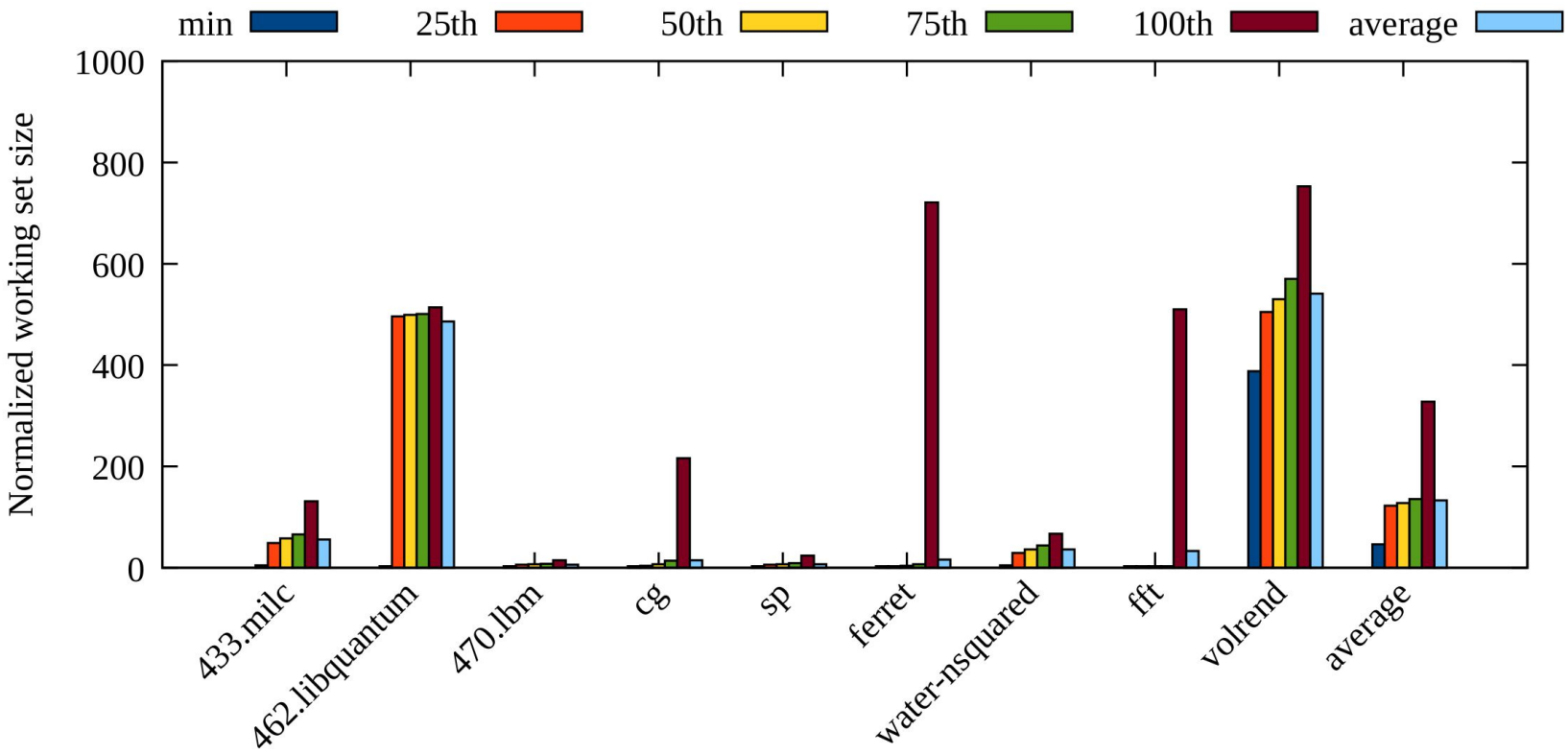



```
20 int main( int argc, char **argv ){
...
31      /* loop over input sets */
32      while( readin(prompt) == 0){
...
44          for( traj_done=0; traj_done < trajecs;
                        traj_done++ ){
...
93          }    /* end loop over trajectories */
…
118      }
119      return 0;
120 }
```

<contol.c>

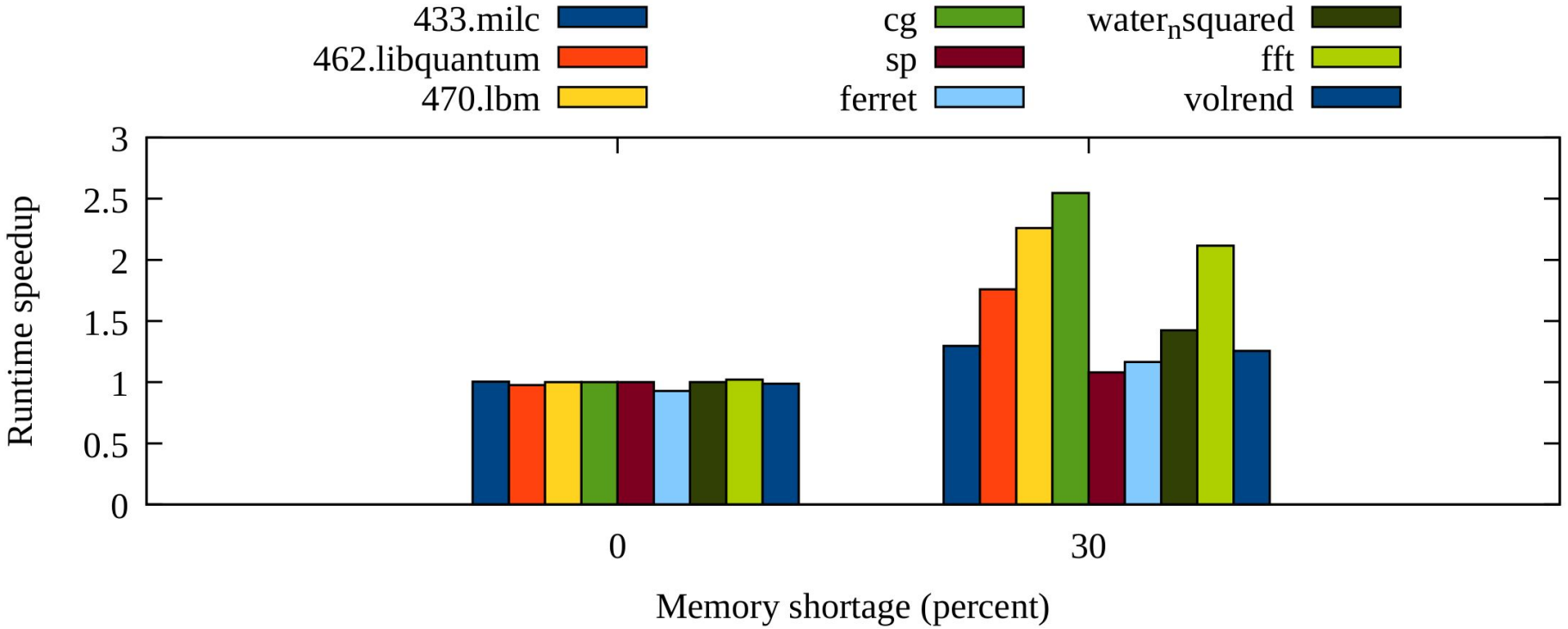
Bounded Monitoring Overhead

- Measured proportion of number of regions that made while monitoring
- Only 132.88 regions in total average
- DAMON not only bounds the overhead, but also minimizes it



Manual Optimizations for Memory Pressures

- Based on damon monitored pattern, classified hot objects and inserted mlock() calls in the source code so that those hot objects to be in DRAM
  - The optimizations are very simple because this is for evaluation of DAMON, not itself
- Measured runtime speedups for varying levels of memory pressure
- Consistently improved performance, without considerable overhead



Future Plans

- Goal is to be merged in upstream, so that hopefully used for
  - User space tools for data access monitoring and
  - Kernel space subsystems requiring data access pattern based optimizations
- Design and implement interface for Kernel internal use
- Adapt to other kernel memory management schemes
- Code cleanup and patchset submission

Summary

- Use of multi-tier memory systems will be common
  - Size of modern workloads are growing so that DRAM cannot fully accommodate
  - Fast storage devices have evolved and available now
- Seems no proper memory monitoring tool for modern environment exists
  - Incurs too high overhead for unnecessarily detailed monitoring
  - Overhead arbitrarily increases as the size of the workload grows
- DAMON monitor data accesses with bounded monitoring overhead regardless of the size and complexity of the target workloads
- Our evaluation composed with 20 realistic workloads show its efficiency and effectiveness

Thanks