D A M O N: Data Access MONitoring Framework for Fun and Memory Management Optimizations

SeongJae Park <sjpark@amazon.de>
Disclaimer

- The views expressed herein are those of the speaker; they do not reflect the views of his employers.

- My cat might come up on the screen. The cat has no ‘- - silent’ option. Sorry, please don’t surprise; keep calm and blame COVID19 :P
I, SeongJae Park <sjpark@amazon.de>

- Kernel / Hypervisor Engineer at Amazon Web Services
- Interested in the memory management and the parallel programming
  - Before joining Amazon, developed Guaranteed Contiguous Memory Allocator and HTM-based update-side synchronization for RCU on NUMA systems
TL; DR

- **When**: Now
- **Who**: Sysadmins and kernel subsystems
- **Where**: On `CONFIG_DAMON=y` kernel
- **What**: Can monitor the data accesses
- **Why**: For fun and better MM

**How (for sysadmins):**

```bash
# damo record $(pidof my_workload)
# damo report heats -heatmap a.png;
# eog a.png
```

- **Review the patches**, please!
Overview

- Motivation
- DAMON
- Live Demo
- Evaluation
- Plans
- Conclusion
Overview

- Motivation
- DAMON
- Live Demo
- Evaluation
- Plans
- Conclusion
Data Access Patterns and The Memory Management

- For a good memory managements, access patterns are needed (e.g., to keep warm data close and hot data closer to CPU)

- Linux kernel memory management (MM) gets the information with
  - PTE ‘Accessed’ bits manipulation, fundamentally

- Cons: Coarse
  - Only accessed or not, between some events such as memory pressures
  - 2 LRU lists and heuristics help this a lot (The heuristics makes MM more complicated, though)

- Pros: Low overhead
  - Extracting finer information will result in higher overhead

- So, it’s a trade-off that carefully taken. But, wait... is it still the best?
Is MM Happy In This Modern World

• Recent trends might changed some things
  – Working sets are continuously growing
  – DRAM is relatively reducing due to the price and energy consumption
  – New memory devices that slower but cheaper and larger than DRAM evolving

• Based on the trend, a number of optimizations made
  – The works optimize MM in creative ways using finer data access patterns
  – Most of the works show impressive improvements;
    Even my simple approach also showed up to 2.55x performance

• But, most of the works had only small interest in the pattern extraction
  – Only naive approaches incurring high/unscalable overhead are used
  – This made most of the works not acceptable in the mainline
  – So, MM needs a subsystem for the fine information, first
Overview

- Motivation
- DAMON
- Live Demo
- Evaluation
- Plans
- Conclusion
Overview

- Motivation

- DAMON
  - What it is and what you can get from it
  - How to use it
  - How it works
  - Misc (how it can be used from user space, how it is tested)

- Live Demo

- Evaluation

- Plans

- Conclusion
Overview

- Motivation
- DAMON
  - What it is and what you can get from it
  - How to use it
  - How it works
  - Misc (how it can be used from user space, how it is tested)
- Live Demo
- Evaluation
- Plans
- Conclusion
DAMON: Data Access MONitor

- Data access monitoring framework for the Linux kernel
  - Provides access frequency of each address range

![Diagram showing DAMON-based Operation Schemes for data access monitoring]
**DAMON: Data Access MONitor**

- Data access monitoring framework for the Linux kernel
  - Provides access frequency of each address range
  - Both kernel space and user space can use it for analysis, memory management optimizations, and fun ;)
Main Design Requirements

- DAMON is designed to mitigate the overhead-accuracy problem.
- For the goal, it fulfills below 4 requirements:
  - Accuracy: The monitoring result should be useful for DRAM level MM.
  - Overhead: Should light-weight enough for online monitoring.
  - Scalability: The upper-bound overhead should be controllable regardless of the size of the monitoring target systems and workloads.
  - Generality: The mechanism should be applicable for wide use-cases (e.g., virtual address, physical address, cache-line granularity, ...).
Overview

• Motivation
• DAMON
  – What it is and what you can get from it
  – How to use it
  – How it works
  – Misc (how it can be used from user space, how it is tested)
• Live Demo
• Evaluation
• Plans
• Conclusion
The Usage: Programming Interface

- **Step 1:** Set the requests in `struct damon_ctx` instances
  - How, what memory regions should be monitored
    - The upper-bound monitoring overhead is included here (how)
  - What function should be periodically called back with the results
    - Users can read the monitoring results inside the function

- **Step 2:** Pass the requests to DAMON via `damon_start()`
  - Then, a kernel thread for the monitoring is created for each request
  - The thread does monitoring and calls the notification callback

- Unless `damon_start()` is called, the system gets no change at all
  - No overhead incurred by just installing DAMON=y kernel
  - IOW, No harm in DAMON=y ;)

void on_notification(struct damon_ctx *c);

static int __init demo(void)
{
    struct damon_ctx ctx = {
        .aggr_interval = 100,
        .aggregate_cb = on_notification,
        ...
    };
    damon_start(&ctx, 1);
    msleep(1000*60);
    damon_stop(&ctx, 1);
    return 0;
}

void on_notification(struct damon_ctx *c);
{
    struct damon_region *r;
    damon_for_each_region(r, c)
        pr_info("%lu-%lu accessed %u times during last 100ms\n", 
            r->start, r->end, r->nr_accesses);
}
Overview

- DAMON
  - What it is and what you can get from it
  - How to use it
  - How it works
    - control of the overhead and accuracy
    - Support of Various Address Spaces
    - DAMON-based MM optimizations
  - Misc (how it can be used from user space, how it is tested)

- [...]
Overview

• [...]  

• DAMON
  - What it is and what you can get from it
  - How to use it
  - How it works
    • control of the overhead and accuracy
    • Support of Various Address Spaces
    • DAMON-based MM optimizations
  - Misc (how it can be used from user space, how it is tested)

• [...]
Control of Overheads and Monitoring Accuracy

- DAMON becomes DAMON in below steps
  - Straightforward Access Monitoring (Fixed granularity)
  - Region-based Sampling (Elastic granularity)
  - Adaptive Regions Adjustment (Best-effort accuracy)
Straightforward Access Monitoring (Fixed Granularity)

- Periodically check if each monitoring target page is accessed
  - Let’s call the period as ‘sampling interval’

- Aggregate the observations into access frequencies
  - Count number of observed accesses and periodically reset the counter
  - Let’s call the period as ‘aggregation interval’

- By notifying the users just before the reset of the counter, we can provide the access frequency of the pages to the users

- Pros: Fine-grained (page size) monitoring
  - Might not strictly required in some performance-centric optimizations

- Cons: High and unscalable monitoring overhead
  - The overhead arbitrarily increases as the target size grows
Region-based Sampling (Elastic Granularity)

• Let's define data objects in access pattern oriented way
  − “A data object is a memory region that all page frames in the region have similar access frequencies”
  − By the definition, if a page in a region is accessed, other pages of the region has probably accessed, and vice versa
  − Thus, checks for the other pages can be skipped

• By limiting the number of regions, we can control the monitoring overhead regardless of the target size

• However, the accuracy will degrade if the regions are not properly set

![Diagram showing Hot region, Cold region, and Target region](Will result in poor accuracy)

![Diagram showing Hot region, Cold region, and Target region](Will result in reasonable accuracy)
Adaptive Regions Adjustment (Best-effort Accuracy)

- Starts with minimal number of regions covering entire target memory areas
- For each aggregation interval,
  - merge adjacent regions having similar access frequencies to one region
  - Split each region into two (or three, depend on state) randomly sized regions
  - Avoid merge/split if the number of regions might be out of the user-defined range
- If a split was meaningless, next merge process will revert it (vice versa)
- In this way, we can let users limit the upper bound overhead while preserving best-effort accuracy
Overview

- [...] 

- DAMON
  - What it is and what you can get from it
  - How to use it
  - How it works
    - control of the overhead and accuracy
    - Support of Various Address Spaces
    - DAMON-based MM optimizations
  - Misc (how it can be used from user space, how it is tested)

- [...]
Primitives for Fundamental Access Monitoring

- The previous descriptions (intentionally) didn’t explain
  - How the monitoring target regions identified, and
  - How each page is checked whether it has accessed or not

- These two unexplained tasks are
  - Dependent on the detailed use-cases
    - Virtual address space VS physical address space
    - Super high accuracy VS only reasonable accuracy
      (suppose some arch provides dedicated super light access check primitives)
  - Independent with the overhead-accuracy handling core logic
Many Use Cases Are Imaginable

- There are many realistic use cases including
  - Full or parts (e.g., stack) of virtual address spaces of specific processes
  - Full or parts (e.g., NUMA) of physical address spaces of the machine
  - Memory regions backed by specific file or device
  - PTE Accessed bit or LRU position as the access check primitive
  - Dedicated h/w feature of special arch as the access check primitive

- Implementing those in DAMON makes it only complex and inflexible
The Core Logics Are Highly Simple and Flexible

- The minimal access check granularity could be anything
  - The description mentioned page size, but it doesn’t have to be
  - Page size, cache line size, 42 bytes, or even one byte is OK if the regions are addressable and access check is possible

- ‘Adaptive regions adjustment' can be turned off
  - Setting the min # of regions same to the max # of regions will turn off the mechanism, as any split and merge will violate the condition
  - Setting the two values same to `\texttt{target\_size / PAGE\_SIZE}` will result in the straightforward page granularity monitoring

- Even multiple CPUs can be used if necessary
  - If you could afford multiple CPUs for super high accuracy, you could
    1) partition the target region into multiple requests and
    2) send those to `\texttt{damon\_start()}` at once
Separation of The Primitives and The Core Logic

- The core logic uses the primitives via only cleanly defined interfaces
- The interface is 4 function pointers in the ‘struct damon_ctx’
  - init_regions(): Initialize the monitoring target regions
  - update_regions(): Update the monitoring target regions if there were some changes (e.g., mmap() or hot-plug)
  - prepare_access_check(): Set next sampling target address and mark it not accessed
  - access_check(): Check if the sampling target address is accessed
- Any primitives following the interface can be configured to be used
DAMON is Extensible

- DAMON users can extend DAMON for their specific usages
  - Implement own primitives and configure damon_ctx to use it
- By default, reference implementations of the primitives for the virtual address spaces and the physical address space are provided
Overview

- [...] 

- DAMON
  - What it is and what you can get from it
  - How to use it
  - How it works
    - control of the overhead and accuracy
    - Support of Various Address Spaces
    - DAMON-based MM optimizations
  - Misc (how it can be used from user space, how it is tested) 

- [...]
The DAMON-based Optimizations

• Now DAMON-based optimizations are available
  – Both the kernel space and the user space can make the optimization by:
    • Step 1: Run DAMON
    • Step 2: Analyze the monitoring results offline or online
    • Step 3: Make some changes based on the analysis
  – The works unaccepted due to the problem could revisited

• I will optimize the kernel in this way, once DAMON is merged in
  – So the upstream kernels will just work (nearly) optimal someday
  – However, downstream kernel / user space optimizations will still required
    • Because special cases always exist
Risks of Non-upstream DAMON-based Optimizations

- It could be difficult, dangerous, dirty, or restrictive
- Optimizations in the kernel space could be difficult and dangerous
  - Not every sysadmin is experienced kernel programmer
  - Kernel bugs are dangerous
- Optimizations in the user space could be dirty and restrictive
  - Receiving and analyzing the results require some lines of code
    - No library for now; To be honest, I want to do everything on the shell
  - Actions to monitored memory regions are restricted
    (madvise_process() might solve most of this problem)
DAMOS: DAMON-based Operation Schemes

- Yet another kernel feature for easy MM optimizations built on DAMON
- Receives ‘schemes’, each constructed with
  - 3 conditions: Size, access frequency, and age of memory regions
  - 1 memory management action
    - Currently supported actions include: `MADV_(WILLNEED|COLD|PAGEOUT|HUGE|NOHUGE)`
- DAMON automatically finds the memory region of the condition and applies the action to the region
- Now users can make DAMON-based optimizations without code

```
# format is:
# <min/max size> <min/max frequency (0-100)> <min/max age> <action>
#
# if a region of size >=4KB didn’t accessed for >=2mins, page out
4K max 0 0 2m max pageout
```
**DAMOS Stats**

- Special DAMOS action, ‘stat’ does nothing but count
  - Total number of regions matched in the condition
  - Total size of regions matched in the condition
- Users can directly get only meaningful numbers such as
  - Size and number of regions of varying access frequencies and ages
  - No need to get the full results and manually analyze it

```
# format is:  
# <min/max size> <min/max frequency (0-100)> <min/max age> <action>
#  
min max    0  49       2mins max stat
min max    50 100     2mins max stat
min max    0  49       4mins max stat
min max    50 100     4mins max stat
```
Overview

- Motivation

- DAMON
  - What it is and what you can get from it
  - How to use it
  - How it works
  - Misc (how it can be used from user space, how it is tested)

- Live Demo

- Evaluation

- Plans

- Conclusion
Interfaces for The User Space

- **Tracepoint**
  - Provide the monitoring results
  - The monitoring should be manually turned on/off

- **DAMON debugfs interface**
  - Receive monitoring requests and provide monitoring records
  - Support both the virtual address and physical address
  - The ABI for development of other user space tools

- **User space tool: DAMON Operator (DAMO)**
  - A reference implementation of user space tool built on the debugfs
  - Provide human friendly interfaces and monitoring results visualization
Tests

- DAMON has several automated tests for it
- Inside the patchset (should be merged together)
  - User space tests: Test debugfs interface based on kselftest
  - Unit tests: Test internal code based on kunit
- Outside the patchset (would not be merged in the mainline)
  - Correctness tests: Time consuming tests including build/accuracy tests
  - Performance tests: Constructed with 25 realistic workloads
- Outside-the-patchset tests might be open-sourced eventually
  - Hope to be also used as a getting started guide
Live Demo using DAMO

$ git clone https://github.com/sjp38/masim
$ cd masim; make; ./masim ./configs/zigzag.cfg &
$ sudo damo record $(pidof masim)

$ damo report raw
$ damo report heats --heatmap access_pattern_heatmap.png
$ damo report wss --range 0 101 1 --plot wss_dist.png
$ damo report wss --range 0 101 1 --sortby time --plot wss_time.png

A demo video is also available: https://youtu.be/l63eqbVBZRY
Overview

- Motivation
- DAMON
- Live Demo
- Evaluation
- Plans
- Conclusion
Evaluation Questions

- How lightweight DAMON is?
- How accurate DAMON is?
- How useful DAMON-based optimizations are?
Evaluation Environment

- Test machine
  - QEMU/KVM virtual machine on AWS EC2 i3.metal instance
  - 36 vCPUs, 128 GB memory, 4 GB zram swap device
  - Ubuntu 18.04, THP enabled policy madvise
  - Linux v5.8 + DAMON patchsets (The source tree is available)

- Workloads: 25 realistic benchmark workloads
  - 13 workloads from PARSEC3
  - 12 workloads from SPLASH-2X

- DAMON monitoring attributes: The default values
  - 5ms sampling, 100ms aggregation, and 1s regions update intervals
  - Number of regions: [10, 1000]
Evaluation Targets

• Variants
  - orig: DAMON turned off (same to vanilla v5.8)
  - rec: DAMON for virtual address of the workload turned on
  - prec: DAMON for entire physical address of the system turned on
  - thp: THP enabled policy set always (to be compared with ethp)
  - ethp: ethp DAMON-based operation scheme is applied

$ cat ethp.damos
# for regions having 5/100 access frequency, apply MADV_HUGEPAGE
min max 5 max min max hugepage
# for regions >=2MB and not accessed for >=7 seconds, apply MADV_NOHUGEPAGE
2M max min min 7s max nohugepage

- prcl: prcl DAMON-based operation scheme is applied

$ cat prcl.damos
# for regions >=4KB and not accessed for >=10 seconds, apply MADV_PAGEOUT
4K max 0 0 10s max pageout
Evaluation Methodology

• Measurement
  - Runtime of the workload
  - System memory usage (\text{MemTotal} - \text{MemFree})
  - Residential Set Size (RSS) of the workload
  - For each of the workload x variant combinations (25 x 6 = 150)
  - Memory usages are periodically measured and averaged
• Every data is average of 5 different runs (150 x 5 = 750)
  - Run each evaluation on 5 different QEMU VMs on different i3.metal instance
  - Effects from weird outliers might be minimized (still fluctuates, though)
• The test automation code might be open-sourced at last
• Detailed results are available online
### Runtime

<table>
<thead>
<tr>
<th>runtime (overhead)</th>
<th>orig</th>
<th>rec</th>
<th>(overhead) prec</th>
<th>(overhead) thp</th>
<th>(overhead) ethp</th>
<th>(overhead) prcl</th>
</tr>
</thead>
<tbody>
<tr>
<td>parsec3/blksohles</td>
<td>137.688</td>
<td>139.910</td>
<td>(1.61)</td>
<td>138.226</td>
<td>(0.39)</td>
<td>150.562 (9.35)</td>
</tr>
<tr>
<td>parsec3/bodytrack</td>
<td>124.496</td>
<td>123.294</td>
<td>(-0.97)</td>
<td>124.874</td>
<td>(0.01)</td>
<td>126.380 (1.51)</td>
</tr>
<tr>
<td>parsec3/canneal</td>
<td>196.513</td>
<td>209.465</td>
<td>(6.59)</td>
<td>189.302</td>
<td>(-3.67)</td>
<td>199.453 (1.50)</td>
</tr>
<tr>
<td>parsec3/dedup</td>
<td>18.060</td>
<td>18.128</td>
<td>(0.38)</td>
<td>18.378</td>
<td>(1.76)</td>
<td>18.397 (1.87)</td>
</tr>
<tr>
<td>parsec3/facesim</td>
<td>343.697</td>
<td>344.917</td>
<td>(0.36)</td>
<td>337.696</td>
<td>(-1.75)</td>
<td>344.805 (0.32)</td>
</tr>
<tr>
<td>parsec3/ferret</td>
<td>288.886</td>
<td>286.110</td>
<td>(-0.95)</td>
<td>292.308</td>
<td>(1.19)</td>
<td>287.814 (0.36)</td>
</tr>
<tr>
<td>parsec3/fluidanimate</td>
<td>342.267</td>
<td>337.743</td>
<td>(-1.32)</td>
<td>330.680</td>
<td>(-3.39)</td>
<td>340.664 (-0.49)</td>
</tr>
<tr>
<td>parsec3/freqmine</td>
<td>437.385</td>
<td>436.854</td>
<td>(-0.12)</td>
<td>437.641</td>
<td>(0.06)</td>
<td>436.998 (0.32)</td>
</tr>
<tr>
<td>parsec3/raytrace</td>
<td>183.036</td>
<td>182.039</td>
<td>(-0.54)</td>
<td>184.859</td>
<td>(1.00)</td>
<td>187.330 (2.35)</td>
</tr>
<tr>
<td>parsec3/streamcluster</td>
<td>611.075</td>
<td>675.108</td>
<td>(10.48)</td>
<td>656.373</td>
<td>(7.41)</td>
<td>541.711 (-11.35)</td>
</tr>
<tr>
<td>parsec3/swaptions</td>
<td>220.338</td>
<td>220.948</td>
<td>(0.28)</td>
<td>220.891</td>
<td>(0.07)</td>
<td>219.986 (-0.16)</td>
</tr>
<tr>
<td>parsec3/vips</td>
<td>87.710</td>
<td>88.581</td>
<td>(0.99)</td>
<td>88.423</td>
<td>(0.81)</td>
<td>88.471 (0.87)</td>
</tr>
<tr>
<td>parsec3/x264</td>
<td>114.927</td>
<td>117.774</td>
<td>(2.48)</td>
<td>116.630</td>
<td>(1.48)</td>
<td>112.237 (-2.34)</td>
</tr>
<tr>
<td>splash2x/barnes</td>
<td>131.034</td>
<td>130.895</td>
<td>(-0.11)</td>
<td>129.088</td>
<td>(-1.48)</td>
<td>118.213 (-9.78)</td>
</tr>
<tr>
<td>splash2x/fft</td>
<td>59.805</td>
<td>60.237</td>
<td>(0.72)</td>
<td>59.895</td>
<td>(0.15)</td>
<td>47.008 (-21.40)</td>
</tr>
<tr>
<td>splash2x/lu_cb</td>
<td>132.353</td>
<td>132.157</td>
<td>(-0.15)</td>
<td>132.473</td>
<td>(0.09)</td>
<td>131.561 (-0.60)</td>
</tr>
<tr>
<td>splash2x/lu_ncb</td>
<td>149.050</td>
<td>150.496</td>
<td>(0.97)</td>
<td>151.912</td>
<td>(1.92)</td>
<td>150.974 (1.29)</td>
</tr>
<tr>
<td>splash2x/ocean_cp</td>
<td>82.189</td>
<td>77.735</td>
<td>(-5.42)</td>
<td>84.466</td>
<td>(2.77)</td>
<td>77.498 (-5.71)</td>
</tr>
<tr>
<td>splash2x/ocean_ncp</td>
<td>154.934</td>
<td>154.656</td>
<td>(-0.18)</td>
<td>164.294</td>
<td>(5.98)</td>
<td>101.861 (-34.26)</td>
</tr>
<tr>
<td>splash2x/radix</td>
<td>142.710</td>
<td>141.643</td>
<td>(-0.75)</td>
<td>143.940</td>
<td>(0.86)</td>
<td>141.982 (-0.51)</td>
</tr>
<tr>
<td>splash2x/radix</td>
<td>50.357</td>
<td>50.331</td>
<td>(-0.05)</td>
<td>50.717</td>
<td>(0.72)</td>
<td>45.664 (-9.32)</td>
</tr>
<tr>
<td>splash2x/raytrace</td>
<td>134.039</td>
<td>132.650</td>
<td>(-1.04)</td>
<td>134.583</td>
<td>(0.41)</td>
<td>131.570 (-1.84)</td>
</tr>
<tr>
<td>splash2x/volrend</td>
<td>120.769</td>
<td>120.220</td>
<td>(-0.45)</td>
<td>119.895</td>
<td>(-0.72)</td>
<td>120.159 (-0.56)</td>
</tr>
<tr>
<td>splash2x/water_nsquared</td>
<td>376.599</td>
<td>373.411</td>
<td>(-0.85)</td>
<td>382.601</td>
<td>(1.59)</td>
<td>348.701 (-7.41)</td>
</tr>
<tr>
<td>splash2x/water_spatial</td>
<td>132.619</td>
<td>133.432</td>
<td>(0.61)</td>
<td>135.505</td>
<td>(2.18)</td>
<td>134.865 (1.69)</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td>4772.510</td>
<td>4838.740</td>
<td>(1.39)</td>
<td>4862.740</td>
<td>(1.89)</td>
<td>4568.970 (-4.26)</td>
</tr>
</tbody>
</table>

This is also available online. Summarized analysis is in following slide, so you don’t need to read this now.
## System Memory Usage (MemTotal - MemFree)

<table>
<thead>
<tr>
<th>Memused_avg</th>
<th>orig</th>
<th>rec</th>
<th>(overhead)</th>
<th>prec</th>
<th>(overhead)</th>
<th>thp</th>
<th>(overhead)</th>
<th>ethp</th>
<th>(overhead)</th>
<th>prcl</th>
<th>(overhead)</th>
</tr>
</thead>
<tbody>
<tr>
<td>parsec3/blackscholes</td>
<td>1825022.800</td>
<td>1863815.200</td>
<td>(2.13)</td>
<td>1830082.000</td>
<td>(0.28)</td>
<td>1800999.800</td>
<td>(1.32)</td>
<td>187743.800</td>
<td>(0.95)</td>
<td>1580027.800</td>
<td>(-13.42)</td>
</tr>
<tr>
<td>parsec3/bodytrack</td>
<td>1425506.800</td>
<td>1438323.400</td>
<td>(0.90)</td>
<td>1439260.600</td>
<td>(0.96)</td>
<td>1400505.600</td>
<td>(1.75)</td>
<td>141229.200</td>
<td>(0.93)</td>
<td>141275.600</td>
<td>(-0.89)</td>
</tr>
<tr>
<td>parsec3/canneal</td>
<td>545844.600</td>
<td>550680.000</td>
<td>(0.89)</td>
<td>543658.600</td>
<td>(-0.40)</td>
<td>523220.200</td>
<td>(-2.48)</td>
<td>539429.600</td>
<td>(-1.19)</td>
<td>478836.800</td>
<td>(-13.74)</td>
</tr>
<tr>
<td>parsec3/dedup</td>
<td>351288.600</td>
<td>352672.000</td>
<td>(-7.20)</td>
<td>326265.600</td>
<td>(-8.37)</td>
<td>304054.800</td>
<td>(-13.65)</td>
<td>317529.000</td>
<td>(-9.90)</td>
<td>315352.400</td>
<td>(-10.96)</td>
</tr>
<tr>
<td>parsec3/facesim</td>
<td>651597.600</td>
<td>580045.200</td>
<td>(-10.98)</td>
<td>578297.400</td>
<td>(-11.25)</td>
<td>569431.600</td>
<td>(-12.61)</td>
<td>577322.800</td>
<td>(-11.40)</td>
<td>482061.600</td>
<td>(-26.02)</td>
</tr>
<tr>
<td>parsec3/ferret</td>
<td>352118.600</td>
<td>326782.600</td>
<td>(-7.20)</td>
<td>99148.600</td>
<td>(-8.37)</td>
<td>322645.600</td>
<td>(-13.65)</td>
<td>317529.000</td>
<td>(-9.90)</td>
<td>313532.400</td>
<td>(-10.96)</td>
</tr>
<tr>
<td>parsec3/fluidanimate</td>
<td>4150.600</td>
<td>25679.200</td>
<td>(518.69)</td>
<td>19914.800</td>
<td>(379.80)</td>
<td>8577.000</td>
<td>(106.64)</td>
<td>17348.200</td>
<td>(317.97)</td>
<td>-100.000</td>
<td>(0.00)</td>
</tr>
<tr>
<td>parsec3/freqmine</td>
<td>123425.400</td>
<td>151548.200</td>
<td>(22.79)</td>
<td>1194143.600</td>
<td>(-1.15)</td>
<td>1280450.200</td>
<td>(0.04)</td>
<td>121267.000</td>
<td>(0.38)</td>
<td>878554.667</td>
<td>(-27.27)</td>
</tr>
<tr>
<td>parsec3/raytrace</td>
<td>2989801.200</td>
<td>3003285.400</td>
<td>(0.45)</td>
<td>3012055.400</td>
<td>(0.74)</td>
<td>2958369.000</td>
<td>(-1.05)</td>
<td>2970897.800</td>
<td>(-0.63)</td>
<td>2962063.000</td>
<td>(-0.93)</td>
</tr>
<tr>
<td>parsec3/swaptions</td>
<td>978625.000</td>
<td>970556.300</td>
<td>(-0.82)</td>
<td>939106.600</td>
<td>(-4.84)</td>
<td>1208450.200</td>
<td>(0.04)</td>
<td>121267.000</td>
<td>(0.38)</td>
<td>878554.667</td>
<td>(-27.27)</td>
</tr>
<tr>
<td>splash2x/barnes</td>
<td>511356.200</td>
<td>526566.400</td>
<td>(3.01)</td>
<td>513236.400</td>
<td>(0.41)</td>
<td>508923.800</td>
<td>(-0.26)</td>
<td>516382.000</td>
<td>(1.01)</td>
<td>418078.333</td>
<td>(-18.21)</td>
</tr>
<tr>
<td>splash2x/fft</td>
<td>1462030.400</td>
<td>1468050.000</td>
<td>(0.41)</td>
<td>1454997.600</td>
<td>(-0.48)</td>
<td>146085.400</td>
<td>(0.34)</td>
<td>146177.400</td>
<td>(-0.02)</td>
<td>44132.000</td>
<td>(-69.81)</td>
</tr>
<tr>
<td>splash2x/radiosity</td>
<td>1462030.400</td>
<td>1468050.000</td>
<td>(0.41)</td>
<td>2251124.600</td>
<td>(-4.90)</td>
<td>2251124.600</td>
<td>(0.41)</td>
<td>3379639.800</td>
<td>(0.79)</td>
<td>3326416.000</td>
<td>(-0.80)</td>
</tr>
<tr>
<td>splash2x/radix</td>
<td>398553.800</td>
<td>391492.900</td>
<td>(0.24)</td>
<td>3877493.200</td>
<td>(-0.72)</td>
<td>3877493.200</td>
<td>(0.81)</td>
<td>7853394.000</td>
<td>(80.61)</td>
<td>463335.000</td>
<td>(18.63)</td>
</tr>
<tr>
<td>splash2x/raytrace</td>
<td>1462030.400</td>
<td>1468050.000</td>
<td>(0.41)</td>
<td>1446997.600</td>
<td>(-0.48)</td>
<td>146695.400</td>
<td>(0.34)</td>
<td>146177.400</td>
<td>(-0.02)</td>
<td>44132.000</td>
<td>(-69.81)</td>
</tr>
<tr>
<td>splash2x/volrend</td>
<td>398553.800</td>
<td>391492.900</td>
<td>(0.24)</td>
<td>3877493.200</td>
<td>(-0.72)</td>
<td>3877493.200</td>
<td>(0.81)</td>
<td>7853394.000</td>
<td>(80.61)</td>
<td>463335.000</td>
<td>(18.63)</td>
</tr>
<tr>
<td>splash2x/water_nsquared</td>
<td>398553.800</td>
<td>391492.900</td>
<td>(0.24)</td>
<td>3877493.200</td>
<td>(-0.72)</td>
<td>3877493.200</td>
<td>(0.81)</td>
<td>7853394.000</td>
<td>(80.61)</td>
<td>463335.000</td>
<td>(18.63)</td>
</tr>
<tr>
<td>splash2x/water_spatial</td>
<td>398553.800</td>
<td>391492.900</td>
<td>(0.24)</td>
<td>3877493.200</td>
<td>(-0.72)</td>
<td>3877493.200</td>
<td>(0.81)</td>
<td>7853394.000</td>
<td>(80.61)</td>
<td>463335.000</td>
<td>(18.63)</td>
</tr>
<tr>
<td>total</td>
<td>41472600.000</td>
<td>41526700.000</td>
<td>(0.12)</td>
<td>4079690.000</td>
<td>(-1.63)</td>
<td>44592600.000</td>
<td>(7.52)</td>
<td>41840100.000</td>
<td>(0.89)</td>
<td>38456146.000</td>
<td>(-7.27)</td>
</tr>
</tbody>
</table>

This is also available online. Summarized analysis is in following slide, so you don’t need to read this now.
## Residential Set Size (RSS)

<table>
<thead>
<tr>
<th>rss.avg</th>
<th>orig</th>
<th>rec</th>
<th>(overhead)</th>
<th>prec</th>
<th>(overhead)</th>
<th>thp</th>
<th>(overhead)</th>
<th>ethp</th>
<th>(overhead)</th>
<th>prcl</th>
<th>(overhead)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>parsec3/blackscholes</em></td>
<td>587078.800</td>
<td>586930.400</td>
<td>(-0.03)</td>
<td>586355.200</td>
<td>(-0.12)</td>
<td>586147.400</td>
<td>(-0.16)</td>
<td>585203.400</td>
<td>(-0.32)</td>
<td>243110.800</td>
<td>(-58.59)</td>
</tr>
<tr>
<td><em>parsec3/bodytrack</em></td>
<td>32479.800</td>
<td>32488.400</td>
<td>(0.05)</td>
<td>32351.000</td>
<td>(-0.37)</td>
<td>32433.400</td>
<td>(-0.12)</td>
<td>32429.000</td>
<td>(-0.13)</td>
<td>18084.000</td>
<td>(-42.09)</td>
</tr>
<tr>
<td><em>parsec3/canneal</em></td>
<td>84218.600</td>
<td>84242.800</td>
<td>(0.00)</td>
<td>84396.000</td>
<td>(0.23)</td>
<td>84576.400</td>
<td>(-0.20)</td>
<td>84212.900</td>
<td>(-0.14)</td>
<td>182529.200</td>
<td>(-2.63)</td>
</tr>
<tr>
<td><em>parsec3/facesim</em></td>
<td>1180700.000</td>
<td>1179309.200</td>
<td>(-0.07)</td>
<td>116047.800</td>
<td>(-1.66)</td>
<td>119879.200</td>
<td>(1.58)</td>
<td>1171802.600</td>
<td>(-0.79)</td>
<td>595531.600</td>
<td>(-49.54)</td>
</tr>
<tr>
<td><em>parsec3/freer</em></td>
<td>99792.200</td>
<td>99641.800</td>
<td>(-0.15)</td>
<td>99044.800</td>
<td>(-0.75)</td>
<td>102041.800</td>
<td>(2.25)</td>
<td>100854.000</td>
<td>(1.96)</td>
<td>83628.200</td>
<td>(-16.20)</td>
</tr>
<tr>
<td><em>parsec3/fluidanimate</em></td>
<td>530735.400</td>
<td>530759.000</td>
<td>(0.00)</td>
<td>530865.200</td>
<td>(0.02)</td>
<td>532440.800</td>
<td>(0.32)</td>
<td>522778.600</td>
<td>(-1.50)</td>
<td>433547.400</td>
<td>(-18.31)</td>
</tr>
<tr>
<td><em>parsec3/freqmine</em></td>
<td>552951.000</td>
<td>552788.000</td>
<td>(-0.03)</td>
<td>552761.800</td>
<td>(-0.03)</td>
<td>556004.400</td>
<td>(0.55)</td>
<td>554001.200</td>
<td>(0.19)</td>
<td>47881.200</td>
<td>(-91.34)</td>
</tr>
<tr>
<td><em>parsec3/raytrace</em></td>
<td>883966.600</td>
<td>880061.400</td>
<td>(-0.44)</td>
<td>883144.800</td>
<td>(-0.01)</td>
<td>871786.400</td>
<td>(-1.38)</td>
<td>881000.200</td>
<td>(-0.34)</td>
<td>267218.000</td>
<td>(-69.77)</td>
</tr>
<tr>
<td><em>parsec3/streamcluster</em></td>
<td>110901.600</td>
<td>110863.400</td>
<td>(-0.03)</td>
<td>110893.600</td>
<td>(-0.03)</td>
<td>115612.600</td>
<td>(4.25)</td>
<td>114976.800</td>
<td>(3.67)</td>
<td>109728.600</td>
<td>(-1.86)</td>
</tr>
<tr>
<td><em>parsec3/swaptions</em></td>
<td>5708.800</td>
<td>5712.400</td>
<td>(0.06)</td>
<td>5681.400</td>
<td>(-0.48)</td>
<td>5720.400</td>
<td>(-0.00)</td>
<td>5726.000</td>
<td>(-0.36)</td>
<td>180.000</td>
<td>(-16.20)</td>
</tr>
<tr>
<td><em>parsec3/vips</em></td>
<td>32272.200</td>
<td>32427.400</td>
<td>(0.48)</td>
<td>31959.800</td>
<td>(-0.97)</td>
<td>34177.800</td>
<td>(5.99)</td>
<td>33306.400</td>
<td>(3.20)</td>
<td>28869.000</td>
<td>(-10.55)</td>
</tr>
<tr>
<td><em>parsec3/x264</em></td>
<td>81878.000</td>
<td>81914.200</td>
<td>(0.04)</td>
<td>8123.600</td>
<td>(-0.07)</td>
<td>85749.000</td>
<td>(2.08)</td>
<td>83236.800</td>
<td>(1.86)</td>
<td>81220.667</td>
<td>(-0.80)</td>
</tr>
<tr>
<td><em>splash2x/barnes</em></td>
<td>1211917.400</td>
<td>1211328.200</td>
<td>(-0.05)</td>
<td>1212450.400</td>
<td>(0.04)</td>
<td>1221591.000</td>
<td>(0.83)</td>
<td>1219924.600</td>
<td>(0.58)</td>
<td>489430.333</td>
<td>(-59.62)</td>
</tr>
<tr>
<td><em>splash2x/fft</em></td>
<td>987439.600</td>
<td>943941.200</td>
<td>(-0.01)</td>
<td>983789.600</td>
<td>(-0.31)</td>
<td>1020484.600</td>
<td>(3.34)</td>
<td>998640.400</td>
<td>(1.88)</td>
<td>790388.100</td>
<td>(-29.07)</td>
</tr>
<tr>
<td><em>splash2x/lu_cb</em></td>
<td>509086.200</td>
<td>50922.600</td>
<td>(-0.03)</td>
<td>509659.600</td>
<td>(-0.08)</td>
<td>505954.600</td>
<td>(0.10)</td>
<td>509479.000</td>
<td>(0.08)</td>
<td>316568.667</td>
<td>(-36.02)</td>
</tr>
<tr>
<td><em>splash2x/lu_ncb</em></td>
<td>509192.200</td>
<td>508437.400</td>
<td>(-0.15)</td>
<td>509331.000</td>
<td>(0.03)</td>
<td>509666.000</td>
<td>(0.08)</td>
<td>509578.200</td>
<td>(0.08)</td>
<td>412985.667</td>
<td>(-19.87)</td>
</tr>
<tr>
<td><em>splash2x/ocean_cp</em></td>
<td>338028.300</td>
<td>3380301.000</td>
<td>(0.00)</td>
<td>337767.200</td>
<td>(-0.08)</td>
<td>341653.200</td>
<td>(1.97)</td>
<td>338945.200</td>
<td>(0.28)</td>
<td>339804.000</td>
<td>(-29.06)</td>
</tr>
<tr>
<td><em>splash2x/ocean_ncp</em></td>
<td>3917913.600</td>
<td>392452.900</td>
<td>(0.17)</td>
<td>393491.800</td>
<td>(-0.43)</td>
<td>7123907.400</td>
<td>(81.83)</td>
<td>4763263.000</td>
<td>(20.05)</td>
<td>2428288.000</td>
<td>(-38.02)</td>
</tr>
<tr>
<td><em>splash2x/radiosky</em></td>
<td>1467978.600</td>
<td>1468655.400</td>
<td>(0.05)</td>
<td>146534.000</td>
<td>(-0.03)</td>
<td>1477722.000</td>
<td>(0.66)</td>
<td>1471036.000</td>
<td>(0.21)</td>
<td>148573.333</td>
<td>(-89.88)</td>
</tr>
<tr>
<td><em>splash2x/radix</em></td>
<td>2413933.400</td>
<td>2408367.600</td>
<td>(-0.23)</td>
<td>238112.200</td>
<td>(-1.36)</td>
<td>2480169.400</td>
<td>(2.74)</td>
<td>2367118.800</td>
<td>(-1.94)</td>
<td>1848857.000</td>
<td>(-23.41)</td>
</tr>
<tr>
<td><em>splash2x/raytrace</em></td>
<td>658664.400</td>
<td>660276.800</td>
<td>(0.25)</td>
<td>660334.000</td>
<td>(-0.26)</td>
<td>660491.000</td>
<td>(0.29)</td>
<td>660634.000</td>
<td>(0.31)</td>
<td>394246.667</td>
<td>(-53.80)</td>
</tr>
<tr>
<td>total</td>
<td>29292400.000</td>
<td>29350400.000</td>
<td>(0.20)</td>
<td>29224634.000</td>
<td>(-0.23)</td>
<td>32985491.000</td>
<td>(12.61)</td>
<td>30157300.000</td>
<td>(2.95)</td>
<td>18340700.000</td>
<td>(-37.39)</td>
</tr>
</tbody>
</table>

This is also available online. Summarized analysis is in following slide, so you don’t need to read this now.
Overhead is Modest

- Normally only 0.3 %CPU is used by the monitoring thread.
- In total, virtual/physical address monitoring respectively incur:
  - 1.39% / 1.89% target workload slowdown
  - 0.12% / -1.63% memory usage overhead
  - 0.20% / -0.23% RSS overhead
- Note the small diff between two, despite of the big target size difference:
  - 128GB for physical, 6MB-9GB for virtual address space
  - The target size doesn’t affect the overhead, as promised.
Monitoring Results Seems Reasonably Accurate

- Various visualizations of the monitoring results including
  - Heatmap
  - Working set size distribution based on size and time
  - Number of (adaptively constructed) monitoring target regions
- All cleanly show reasonable access patterns and distributions

This is also available online.
DAMOS: Access-aware THP Hints (ethp)

- ‘ethp’ preserves the speedup while reducing the memory bloat

- Interesting case: splash2x/ocean_ncp
  - thp achieves 34.26% speedup but 80.61% system memory waste (best speedup and worst memory waste among the 25 workloads)
  - ethp achieves 7.96% speedup but only 18.63% system memory waste
  - Hence, ethp removes 76.90% of memory waster while preserving 23.23% of speedup for the workload

- Removes 88.16% of thp’s system memory waste in total

- Preserves 88.73% of thp’s speedup in total

- NOTE: ethp is only for proof-of-concept and thus not optimized
  - The ethp speedup could be higher if khugepaged promote pages marked with MADV_HUGEPAGE more aggressively
DAMOS: Proactive Reclamation (prcl)

- ‘prcl’ reduces working sets while making only modest slowdown
- Incurs 8.74% speed down in total
- Reduces 37.39% RSS in total
- Best case: parsec3/freqmine
  - Recuces 91.34% of RSS while incurring only 1.58% speed down
- NOTE: prcl is only for proof of concept and thus not optimized
  - Paging out 10sec inactive pages might be too aggressive (Google’s proactive reclamation waits 2 minutes)
  - With faster swap devices, the speed down could further reduced
Evaluation Wrap-up: DAMON Is...

- Lightweight
- Accurate
- Useful for MM optimizations
Overview

- Motivation
- DAMON
- Live Demo
- Evaluation
- Plans
- Conclusion
History of DAMON Project

- 2019.03: A prototype research project, **DAPTRACE** kicked-off
- 2019.09: Renamed into DAMON, Presented at **Kernel Summit’2019**
- 2019.12: The development resumed in Amazon
- 2020.01: Posted the RFC v1 patchset
- The work was also introduced in several conferences and medias
  - **Kernel Summit’19**, **MIDDLEWARE Industry’19**, **Phoronix**, **LWN**, **Google Linux Kernel Exchange’20**, and **Kernel Summit’20**
Now: DAMON patchsets series

- **V20 DAMON patchset**
  - The core of DAMON (framework part)
  - Provide a DAMON primitives for the virtual address spaces

- **Two RFC patchsets for future changes are in below order**
  - Only to show how DAMON could be evolved
  - Might not ready-to-be-merged level quality
  - **RFC v14** of DAMON-based Operation Schemes (DAMOS)
    - Support only virtual addresses
  - **RFC v7** of Physical Memory Address Space Support
    - Implement another DAMON primitives for the physical address space
    - Support only mapped LRU pages
Before DAMON

User space

Kernel space

perf

vma

PTE

rmap
DAMON Patchset 1-4/15

User space

perf

Kernel space

DAMON

vma  PTE  rmap

Framework
DAMON Patchset 5-6/15

User space

 perf

Kernel space

 DAMON

 For vaddr

 vma  PTE  rmap

 Framework

 Low level primitives
DAMON Patchset 7/15

User space

Kernel space

Data Accesses Recording

Kernel space applications

DAMON

Framework

Low level primitives

For vaddr

vma  PTE  rmap
DAMON Patchset 8-10/15

- User space interfaces
  - perf
  - tracepoints
  - debugfs

- Kernel space applications

- Framework

- Low level primitives

- User space interfaces

- Kernel space applications

- Framework

- Low level primitives

- DAMON

- For vaddr

- vma

- PTE

- rmap
DAMON Patchset 11/15

- 12-15 are for documentations and tests
DAMOS Patchset

User space tools
User space interfaces
Kernel space applications
Framework
Low level primitives

User space

Kernel space

Data Accesses Recording
DAMON-based Operation Schemes
DAMON
For vaddr
vma
PTE
rmap

perf
DAMO
tracepoints
debugfs
Physical Address Support Patchset

User space
- perf
- DAMO
- tracepoints
- debugfs

Kernel space
- Data Accesses Recording
- DAMON-based Operation Schemes
- DAMON
- For vaddr
- For phys addr
- vma
- PTE
- rmap

User space tools
User space interfaces
Kernel space applications
Framework
Low level primitives
TODOs

• Make current DAMON patchsets series merged in the mainline

• Support more address spaces
  – Cgroup, cached pages, specific file-backed pages, swap slots, …
  – Physical address support from DAMOS

• Improve the user space interface
  – Multiple contexts, CPU usage charge, …

• Optimize for special use-cases
  – Page granularity monitoring, accessed-or-not monitoring, …

• DAMON-based MM Optimizations
  – Page reclaim, THP, compaction, NUMA balancing, …
Need Your Opinions

• Are there tasks you want to…
  – Put in the *current* DAMON patchsets before those be merged in,
  – Put in the *future* DAMON patchsets, or
  – Assign higher/lower prioritize?
VOTE: What Task Should Have Highest Priority?

A. Make current DAMON patchsets series merged in the mainline

B. Support more address spaces
   - Cgroup, cached pages, specific file-backed pages, swap slots, ...
   - Physical address support from DAMOS

C. Improve the user space interface
   - Multiple contexts, CPU usage charge, ...

D. Optimize for special use-cases
   - Page granularity monitoring, accessed-or-not monitoring, ...

E. DAMON-based MM Optimizations
   - Page reclaim, THP, compaction, NUMA balancing, ...

F. Something other? Reply to the patchset, please
Summary

- DAMON is a kernel subsystem providing data access monitoring with
  - Reasonable best-effort accuracy
  - Lightweight overhead and scalable control of it
- Both kernel space and user space could use DAMON for
  - Analysis, MM optimizations, and fun
- Evaluations with 25 realistic workloads say the benefit could be big
- The patchset and two RFC patchsets for future features are available
  - V20 DAMON, RFC v14 DAMOS, RFC v7 Physical address support
  - Review, please!
More Resources: https://damonitor.github.io

- Source Tree: https://github.com/sjp38/linux/tree/damon/next
- Patchsets:
  V20 DAMON, RFC v14 DAMOS, RFC v7 Physical address support
- Visualized monitoring results
  - https://damonitor.github.io/test/result/visual/next/rec.heatmap.1.png.html
  - https://damonitor.github.io/test/result/visual/next/rec.wss_sz.png.html
  - https://damonitor.github.io/test/result/visual/next/rec.wss_time.png.html
- Hidden index page: https://damonitor.github.io/_index
Questions?