# Characterizing and Optimizing Hybrid DRAM-PM Main Memory System with Application Awareness

Yongfeng Wang, Yinjin Fu*, Yubo Liu, Zhiguang Chen, Nong Xiao*
*School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China*
{firstname.lastname}@nscc-gz.cn, xiaon6@mail.sysu.edu.cn

***Abstract*—Persistent memory (PM) has always been used in combination with DRAM to configure hybrid main memory systems that can obtain both the high performance of DRAM and large capacity of PM. There are critical management challenges in data placement, memory concurrency and workload scheduling for the concurrent execution of multiple application workloads. But the non-negligible performance gap between DRAM and PM makes the existing application-agnostic management strategies inefficient in reaching the full potential of hybrid memory. In this paper, we propose a series of application aware optimization strategies, including application aware data placement, adaptive thread allocation and inter-application interference avoiding, to improve the concurrent performance of different application workloads on hybrid memory. Finally, we provide the performance evaluation for our application aware solutions on real hybrid memory hardware with some comprehensive benchmark suites. Our experimental results show that the duration of multi-application concurrent execution on hybrid memory can be reduced by at most 60.7% for application aware data placement, 37.7% for adaptive thread allocation and 34.8% for workload scheduling with inter-application interference avoiding, respectively. And the additive effects of all these three optimization methods can reach 62.8% performance improvement with negligible overheads.**

## I. Introduction

Memory has long been a computing bottleneck, commonly known as “memory wall”, which has been exacerbated by the advent of multi-core processors [1]. Persistent memory, such as Intel’s Optane Memory [2], is persistent, byte addressable, available in much larger capacity and lower cost per gigabyte than traditional DRAM, but suffers several times higher latency and lower bandwidth. It has always been used in combination with DRAM to configure hybrid main memory systems that can obtain both the high performance of DRAM and large capacity of PM. However, the comprehensive performance characteristics of hybrid memory have not been well studied, which makes it difficult for concurrent applications to be fully utilizing system resources on hybrid memory.

Multiple concurrent applications on a multicore chip contend with each other to access main memory. If the limited memory bandwidth is not managed well, it can result in significant degradation in both system performance and individual application performance. In hybrid memory, multi-application concurrency is significantly different from that on a DRAM-only memory [3] or PM-only memory [4] systems. To achieve high system performance, we must consider three aspects of application management in hybrid memory system: the data placement on DRAM or PM for application workloads, the thread allocation of parallel applications, and the execution order of concurrent applications. Intelligent data placement [5]–[7] can achieve higher performance than default configurations on a hybrid-memory system. And it is also significantly important to optimize thread allocation such that it enables efficiently execute multi-threaded programs on hybrid memory architectures [8]–[10]. Moreover, the interference on the hybrid memory system is very different from that on a DRAM-only memory system [11], [12], and it can be alleviated by scheduling the execution order of multiple concurrent applications. However, all the above schemes are application agnostic, and the memory controller is unaware of the individual demands of concurrent applications.

TABLE I
The multi-application concurrency of hybrid memory

| Management Strategies | Application Agnostic | Application Awareness |
|---|---|---|
| Data placement | [5], [6], [7] | Our work |
| Thread allocation | [8], [9], [10] | |
| Interference avoiding | [11], [12] | |

According to our preliminary observations in Section III-V, the non-negligible performance gap between DRAM and PM makes the existing application-agnostic management strategies inefficient in reaching the full potential of hybrid memory. In GPU memory systems, application-aware memory scheduling schemes [13], [14] can improve the fairness and overall system performance by reducing the interference between address translation and data requests. In DRAM main memory systems, application aware memory channel partitioning algorithm [15] can assign preferred memory channels to different applications to reduce inter-application memory interference. Inspired by these works, we propose a series of application aware optimization strategies, including application aware data placement, adaptive thread allocation and inter-application interference avoiding, to improve the concurrent performance of multiple applications on hybrid memory.

This paper makes the following contributions:

- To the best of our knowledge, this is the first work that provides application aware optimization for multi-application concurrency in hybrid DRAM-PM memory.
- We propose an application aware data placement scheme to enhance the average run time of multiple applications by assigning the application workloads which have higher performance improvement to DRAM.
- We can independently adjust the number of threads for each application running on different memory medium to improve the overall system performance.

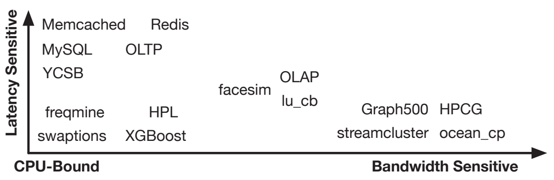

Fig. 1. Application classification

- We propose an application-aware scheduling to optimize the concurrent execution of multiple applications by avoiding inter-application interference.
- We evaluate and analyze the comprehensive performance characteristics of multiple applications that are concurrently executed on a real server platform having a hybrid main memory system.

## II. Application Aware Hybrid Memory Management for Concurrent Applications

According to the differences in system resource demands, we can divide the application workloads of computing servers into CPU-bound and memory-bound. To focus our work on memory resource management, the memory-bound applications are further classified into bandwidth-sensitive applications and latency-sensitive applications. Our classification enables us to clearly distinguish between the workload classes, as each workload class forms its own distinct cluster.

Fig. 1 shows a graphical view of the latency sensitivity and the memory bandwidth demand for all of the workloads. When an application has low values on both x-axis and y-axis, it is a CPU-bound application. The database workloads with high performance requirements have the most sensitivity to latency, and low sensitivity to bandwidth. While the workloads for big data analysis, such as OLAP, lu_cb and facesim, are intermediate in sensitivity to both bandwidth and latency. The HPCG, Graph500 and some data-intensive workloads have the most sensitivity to bandwidth and the least sensitivity to latency. The CPU-bound workloads, such as HPL, swaptions and freqmine, will not show much sensitivity to memory latency or bandwidth.

In hybrid main memory, the effective use of the persistent memory devices helps reducing data movement and is of paramount importance to achieve high performance on modern and future computing systems. However, we observe that the traditional memory system is application agnostic, and the widely used memory scheduler [13] is unaware of the individual demands of concurrent applications. It mainly focuses on improving DRAM page hit rates, and might hurt the overall system performance and fairness. This problem becomes more noticeable when the memory demands of concurrent applications have wide variation, implying that an application with high memory demand attempts to monopolize the memory resource usage over an application with low demand.

Our goal is to provide a better understanding of the interactions of concurrently executing applications in the hybrid DRAM-PM memory subsystem by leveraging application awareness. There are three critical technique challenges for concurrent applications running on hybrid memory:

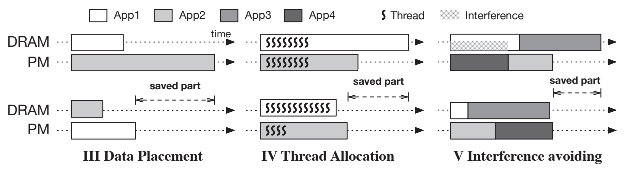

Fig. 2. Application-aware optimization

1) *How to dispense the application workloads to the appropriate memory devices to enhance overall system performance of concurrent applications?*
2) *How to guarantee the fairness of concurrent execution of different workloads by allocating the optimal number of threads for each application?*
3) *What kind of application workload scheduling scheme can avoid or mitigate inter-application interference among two or more concurrently running applications to achieve higher performance?*

As shown in Fig. 2, to address these research issues, we try to perform application aware optimization on data placement, memory concurrency and interference avoiding for multiple concurrent application workloads on the hybrid main memory. The performance boost achieved by moving workload from PM to DRAM is quite different for various applications. In Section III, we attempt to enhance the average run time of multiple applications by assigning the application workloads to DRAM when they have higher performance improvement. Different from the conventional fixed thread allocation strategies, we can adaptively choose the optimal number of threads for each application running on different memory medium to further improve the overall system performance in Section IV, since there is a great difference on the concurrency for applications running on two kinds of memory medium. Moreover, we make an attempt to optimize the concurrent execution of multiple applications by avoiding inter-application interference with an application-aware scheduling in Section V. These methods use offline analysis to get prior knowledge about applications, which is sufficient for most cases.

## III. Data placement of Heterogeneous Memory

We present an application-aware data placement strategy to improve the default memory management policy in Linux.

### A. Hybrid Memory Performance Test

In hybrid memory, the application performance on two kinds of memory medium is quite different, and the magnitude of this impact is highly dependent on the application characteristics. To evaluate this, we run several application workloads chosen from PARSEC [16], splash2x [17] and NAS Parallel benchmarks(NPB) [18] on our testbed. It has an Intel Xeon Gold 6230N processor with 192GB DRAM and 256GB PM. All subsequent experiments are run on the same machine. Fig. 3 shows the performance degradation with the ratio of complete time of application workloads running on PM versus DRAM with 8 threads. Each application will run 10 times on PM and DRAM respectively and we record their average complete time. We observe that the running duration on PM for ocean_cp is 17.5× longer than that on DRAM. In comparison, it is not

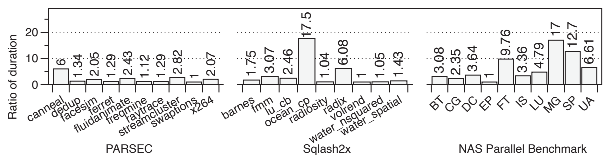

Fig. 3. The ratio of running duration on PM normalized to that on DRAM

**Algorithm 1** Application-aware Data Placement

```
$S$: the size of memory this $App$ need
procedure AllocateSpace($App$, $S$)
    if $S \geq$ Size of free space in DRAM then
        if $App$ is Bandwidth-Sensitive then
            if $S \leq$ DRAM Size of CPU-Bound app then
                Migrate DRAM pages of these app to PM
                return space in DRAM
            else
                return space in PM
        else
            return space in PM
    return space in DRAM
```

obvious for the performance degradation of some applications, like freqmine and swaptions.

The persistent memory has a lower bandwidth than DRAM, which makes the applications running longer on PM. Taking ocean_cp as an example, it streams through its partition of many different grids in different phases of the computation, and can incur substantial capacity and conflict misses as problem sizes increase [17]. When the working set exceeds the cache size, memory bandwidth becomes the bottleneck of this application. In this experiment, ocean_cp consumes 40.9GB/s and 15.3GB/s bandwidth on DRAM for read and write, respectively, while only 2.45GB/s and 0.8GB/s on PM due to its lower bandwidth. This 17× gap of bandwidth can explain the slowdown of ocean_cp on PM. The limited bandwidth of PM is also the main reason for the slowdown of bandwidth-sensitive applications, such as canneal, radix, NPB/FT, NPB/MG and NPB/SP.

Additionally, the run duration of other application workloads on PM is less than 1.5× of that on DRAM, such as freqmine and swaption. They are CPU-bound applications, and can make full use of memory locality to minimize cache misses. As a result, memory is not the bottleneck for this type of application. Hence, the application run on different memory medium has unnoticeable impact on the execution time of these workloads.

**Observation:** *The run duration of bandwidth-sensitive applications on PM is 10s times of that on DRAM, but not for CPU-bound application workloads.*

**Implication:** *The data placement of various application workloads in hybrid memory will significantly impact the system performance. CPU-Bound workloads can run on DRAM or PM freely, while the bandwidth-sensitive application workloads should be stored on DRAM to avoid the significant performance degradation.*

## B. Optimization Solution

The fixed strategy to allocate free space on Linux is inefficient to hybrid memory. Linux's default memory management policy is performed in an application oblivious way. In a server having hybrid DRAM and PM memory, Linux always attempts to maximize the utilization of DRAM and only try to allocate the free space on PM when there is no available space in DRAM. As a result, if a bandwidth-sensitive application is launched on hybrid memory when DRAM does not have enough free space, Linux OS will allocate memory page on PM, which will results in significant performance degradation for the bandwidth-sensitive applications.

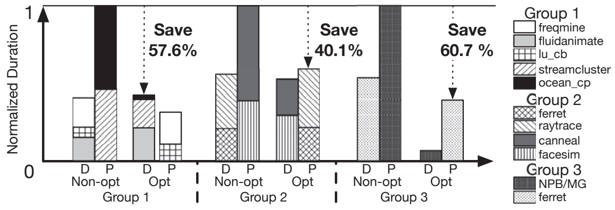

Fig. 4. Optimization Results. D and P refer to DRAM and PM.

In our work, we present an application-aware data placement strategy as shown in Algorithm 1 to improve the Linux default strategy. When an application workload is launched, it firstly tries to assign it on DRAM if there is enough free space. Otherwise, we must judge whether the application is bandwidth-sensitive. For bandwidth-sensitive applications, it can migrate some occupied DRAM pages of CPU-Bound applications to PM, and then allocate these DRAM pages for the new application. If all the application workloads in DRAM are bandwidth-sensitive, we only have to assign the new application to PM. Moreover, we also assign the bandwidth-insensitive applications to PM with low performance degradation. To make our application aware strategy practical, the application type can be defined by user or classified automatically from the information of hardware performance counter.

## C. Evaluation Results

We compare our proposed application aware strategy with the traditional application-agnostic method in three application groups. In traditional scheme, each workload in the application group is executed in parallel on the same memory with 8 threads. To make full use of hybrid memory, it can concurrently executed on different memory channels/devices. In our experiment, we try to reduce the total duration of each application group by exploiting application awareness.

As shown in Fig. 4, we test the run duration of three application groups, and the application workloads are concurrently executed in each group. Here, we only compare our work with the Linux default strategy since the difference in memory medium [5], [6] and hardware support [7] in the related works. In the default strategy of Linux OS data placement, freqmine, fluidanimate and lu_cb applications are assigned to DRAM while ocean_cp and streamcluster is placed on PM for Group 1. According to our application classification, ocean_cp and streamcluster are bandwidth-sensitive applications and their performance bottleneck is the PM bandwidth, while freqmine and lu_cb are bandwidth-insensitive applications and the performance impacts of PM on these workloads are negligible. In our application-aware strategy, it can adjust the data placement by placing bandwidth-sensitive applications on DRAM and moving other workloads freqmine, lu_cb to PM. And it can

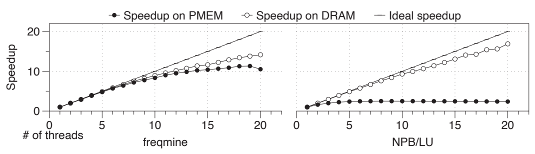

Fig. 5. Scalability of different applications on DRAM and PM

reduce the total run time of Group 1 by 57.6%. Similarly, our application aware strategy can decrease the run duration of Group 2 and Group 3 by 40.1% and 60.7%, respectively. Moreover, we also test the additional time overhead for page migration in hybrid memory, and it does not exceed 0.2% of the run duration in all application groups.

## IV. Memory Concurrency

We describe an adaptive thread allocation scheme to support application concurrency on hybrid main memory.

### A. Scalability Test

The concurrent capacity of application workloads is distinct for different medium in hybrid memory. To test the application concurrency, we run workloads with thread number varies from 1 to 20. For instance, as shown in Fig. 5, freqmine has similar scalability no matter on DRAM or PM. However,unlike running on DRAM, it is hardly to improve the performance of NPB/LU on PM by increasing the number of threads. The parallel efficiency of some applications is hard to be improved maybe due to the limited bandwidth in hybrid memory.

Different applications have various bandwidth requirements, resulting in a significant divergence in their scalability on hybrid memory. In the above experiments, when an application is executed in a single thread, NPB/LU needs 2.3GB/s and 1GB/s of bandwidth for read and write, respectively. As the number of threads increases, its bandwidth requirements increase linearly and will immediately exceed the upper limit of PM, making the memory bandwidth a bottleneck for this application. In our experiment, the bandwidth-sensitive applications have similar results in their scalability. In contrast, freqmine needs only 182MB/s and 66MB/s of bandwidth for read and write per thread. This application has low sensitivity to memory bandwidth, which allows it to scale well until its bandwidth demand larger than the memory bandwidth of hybrid memory.

**Observation:** *The application concurrency varies with the memory medium. The performance of bandwidth-sensitive applications run on PM cannot be synchronously enhanced with the thread number increasing.*

**Implication:** *To achieve the highest parallel performance, the optimal thread number for different application workloads vary with the memory medium type. The bandwidth-sensitive applications should limit the thread number on PM to improve the overall system efficiency.*

### B. Optimization Solution

The determination of the thread number is very complicated for the parallel computing in a hybrid memory system. It is a widely held belief that more threads are preferred to speed up the applications. Though the traditional thread allocation policy

**Algorithm 2** Application-aware Thread Allocation

```
1: $D_t$ : the duration of $App$ using $t$ threads
2: $T$ : the optimal number of threads
3: $S$ : the speedup of $App$
4: procedure OptimalThreadNumber($App$) $\rightarrow T$
5:     $T = 0$, $S = 1$, $S' = 1$, $R = 1$
6:     Get $D_1$ by running $App$ with 1 thread
7:     do
8:         $T = T + 1$, $S' = S$
9:         Get $D_{T+1}$ by running $App$ with $T + 1$ threads
10:        $S = D_1/D_{T+1}$, $R = (S - S')/S'$
11:                            $\triangleright$ R : the ratio of the speedup increasement
12:    while $R \geq 0.1$
13:    return $T$
```

is effective on DRAM, it is not a good choice for the application on hybrid memory any more since the memory bandwidth of PM is much smaller than that of DRAM. The inappropriate thread number can not speed up the application on hybrid memory, but it may even harm the overall performance. Therefore, it is necessary to find an appropriate parallel parameter for the applications running on different memory mediums.

We propose a thread allocation approach for applications on hybrid memory by exploiting application awareness to avoid inefficient thread allocation. We first find the optimal thread number for the selected application by Algorithm 2. It will perform the scalability test and return the optimal number of threads if an addition thread can not measurably increase the speedup. Then, the remain thread resources can be allocated to other applications whose number of threads less than their optimal values. Hence, the overall efficiency can be improved by this per-application threads adjustment. We perform extra tests in advance to get the optimal thread number for each application workload on different memory mediums. With these prior knowledge, it only brings negligible lookup overhead in the run time to implement our adaptive thread allocation.

### C. Evaluation Results

Different from the hybrid DRAM-SRAM memory optimizations in [8], [9], we provide a practical solution for the multi-application concurrency of hybrid DRAM-PM memory systems rather than the only concurrency analysis of OLAP workloads in [10]. As shown in Fig. 6, the overall performance of applications parallel running on hybrid memory can be improved by per-application threads adjustment. This optimization need an extra directory look up operation to get the optimal number of threads and there is almost no visible overhead. In Group 1, the left results show the duration of traditional evenly distributed eight threads for each applications. We find the optimal number of threads NPB/FT on PM is four with the prior knowledge. In addition, freqmine on DRAM can still maintain good scalability with more than eight threads. In our proposed approach, we assign four threads for NPB/FT while twelve threads for frqmine, which can reduce the overall execution time of these applications by 36.5%. We can also improve the performance of Group 2 by 37.7% in a similar way. In Group 3, we consider the optimal number of threads for lu_cb is four. When only reducing the number of threads of lu_cb from 8 to 4, the overall execution time of two 8-thread running applications can also be decreased by 23.9% due to interference mitigation. It will be studied in depth in the next section.

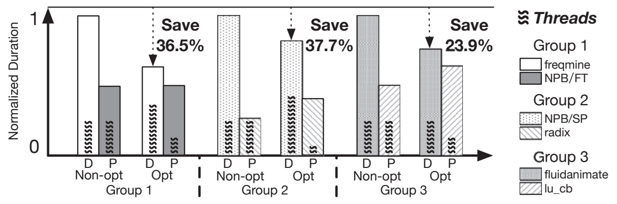

Fig. 6. Optimization Result

## V. Inter-application Interference Avoiding

We propose an application-aware scheduling strategy to alleviate or avoid the interference of different applications for higher overall concurrent performance.

### A. Interference Test

There is bandwidth contention when two or more applications simultaneously access the hybrid memory, which will cause serious interference and degradation on concurrent performance. And this interference in hybrid memory is more complicated than that in the pure DRAM memory system. To demonstrate this, we first try to traversed all memory access modes. Our test include 4 types of memory access patterns (1)SR: Sequential Read, (2)RR:Random Read, (3)SW: Sequential Write, (4)RW: Random Write with 1, 2, 4, 8 and 10 threads to run on DRAM and PM, which will form 40 types of memory access modes. Then, we run the bandwidth benchmark and get the bandwidth of all memory access modes without any interference. At last, 40 types of interference are constructed by 40 types of memory access mode with separate running, we rerun the benchmark test for each memory access modes and get the reduction ratio of the bandwidth under each type of interference. The results are shown in the Fig. 7 and summarized as the following two aspects.

At first, the interference between DRAM and PM is asymmetrical. Accessing DRAM has little effect on the bandwidth of PM, whereas PM reads or writes can significantly reduce the bandwidth of DRAM. The bandwidth of sequential read with 10 threads on PM is 13GB/s and it can be reduced by 15% (11 GB/s) at most with the interference caused by DRAM reads or writes with 10 threads. In comparison, the bandwidth of sequential read on DRAM can be reduced by 80%(94GB/s to 18GB/s) with 10 interfered threads to sequential read on PM. Moreover, for the bandwidth of DRAM, the interference caused by PM reads and writes is more significant than that on DRAM. Therefore, in the following analysis, we will ignore the interfered PM bandwidth caused by accessing DRAM and mainly focus on the interference caused by accessing PM .

Secondly, the number of threads and memory access pattern has a great impact on the interference of concurrent applications. The interference will become more serious with the number of threads increasing. Random write on PM with 1, 2, 4, 8 or 10 threads will reduce the bandwidth of sequential write on DRAM with 10 threads by 33.8%, 68.2%, 89.9%, 96.7%, 97.6%(68GB/s to 1.7GB/s). For the memory access pattern, random write will produce the most severe interference on write performance of DRAM. Considering the intensive write on PM will reduce the bandwidth more than 95%, the bandwidth-

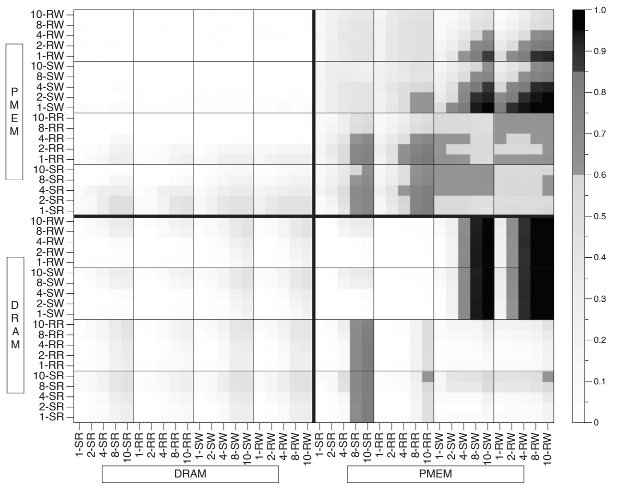

Fig. 7. Bandwidth contention on hybrid memory. X axis presents the interference type and Y axis is benchmark type. The darker the color means the more the bandwidth is reduced compared to when there is no interference.

sensitive application running on DRAM will still be severely affected. Sequential read also have a considerable effect on the sequential read bandwidth of DRAM, since sequential read on PM with 10 threads reduce the bandwidth of sequential read on DRAM with 10 threads by 79% (89.3GB/s to 18.3GB/s).

**Observation:** *Intensive access on PM will significantly reduce the bandwidth of DRAM. Random writes and sequential reads to PM have a more serious interference on write and read bandwidth of DRAM, respectively.*

**Implication:** *We can schedule the concurrent applications on hybrid memory to avoid or reduce the interference by avoiding bandwidth-sensitive applications run on DRAM when write-intensive applications is running on PM.*

### B. Optimization Solution

To minimize the interference among applications on hybrid memory, we need to dynamically adjust the execution order of these application workloads. Our optimization scheme include two steps: 1) detect the potential interference and 2) adjust execution order with application-aware strategy. A potential interference is detected if a write-intensive application is running on PM and a bandwidth-sensitive application is concurrently running on DRAM. It need to check the application type when a new application is launching. Then, our proposed strategy will dispatch the concurrent applications to avoid the severe interference with application-aware selection using two methods. ① When a write-intensive application has already run on PM, it can launch another application that has less susceptible to interference, such as a CPU-bound application, to run on DRAM. This strategy can schedule the application to run ahead of the original bandwidth-sensitive application to avoid inter-application interference. ② When a bandwidth-sensitive application is already running on DRAM, the strategy can schedule another application that cause less interference on PM early rather than a write-intensive application. Though this adjustment may not considerably lower the run duration of

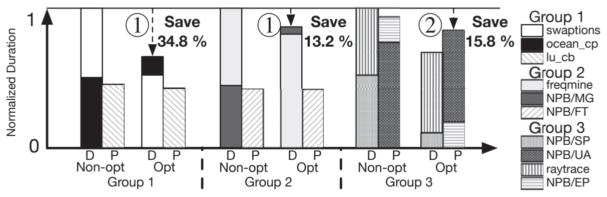

Fig. 8. Optimization Result

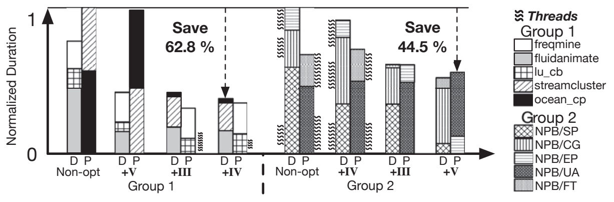

Fig. 9. Overall Optimization Result

applications on the PM, it can improve the overall performance by enhancing the performance of the applications on DRAM.

## C. Evaluation Results

We compare our work with the traditional application schedule strategy. Fig. 8 shows that our proposed optimization strategy can improve the overall performance by minimize inter-application interference. For “Non-opt” setting, these applications will be executed without interference awareness and may lead to inefficiency for concurrent execution on hybrid memory. The scheduler looks up the type of application from prior knowledge and makes a decision with no visible overhead. For Group 1, lu_cb on PM have a severe interference with ocean_cp application on DRAM. Based on our prior knowledge, lu_cb is a write-intensive application, while ocean_cp is a bandwidth-intensive application. Applying the method ①,it can reduce the run duration by 34.8%. In Group 2, though the two applications: freqmine and NPB/MG are both interfered by NPB/FT, the NPB/MG suffers more inter-application interference. The rescheduling can also bring 13.2% performance improvement for the concurrent execution of these three applications. In Group 3, using methods ② brings a 15.8% improvement.

## VI. Overall Optimization

We study the additive effects of all three optimization techniques on hybrid memory and the results are shown in Fig. 9. The default number of threads in Group 1 is 8 for all five application workloads. Since the ocean_cp have a strong interference effect on the fluidanimate, we apply approach in Section V to avoid this inter-application interference, and run the streamcluster first. Then, we exchange the placement of ocean_cp and streamcluster with lu_cb and freqmine on different memory mediums, because the first two workloads are bandwidth-sensitive and they should be executed on DRAM, as noted in Section III.And the last two workloads can be run on PM without significant performance degradation. At last, the overall performance can be further improved by reducing the number of threads from 8 to 4 on workload lu_cb. The additive effect of all three optimization techniques on Group 1 is saving 62.8% duration time. Similarly, the overall execution time of Group 2 can be saved by 44.5% when we perform these optimization schemes in order of thread allocation in Section IV, workload placement in Section III, and application scheduling in Section V. With three optimization enabled, the overhead is not more than 0.2% and it mainly come from page migration in workload placement.

## VII. Conclusions

The concurrent execution of multiple applications are application agnostic in traditional designs and they cannot fully utilize the potential of hybrid DRAM-PM main memory. In this paper, we first evaluate and analyze the comprehensive performance characteristics of multiple concurrent applications on a real server platform having a hybrid main memory system. Then, we propose three kinds of application aware optimization strategies, including application aware data placement, adaptive thread allocation and inter-application interference avoiding, to improve the multi-application concurrency on hybrid memory. Comparing with the existing default designs, the experimental results show that our optimization methods can significantly save the overall execution duration of multiple applications with negligible overheads.

**Acknowledgments:** The work was partially supported by the Natural Science Foundation of China under Grants:61832020, 61872392, Key-Area Research and Development Program of Guang Dong Province (2019B010107001), the Natural Science Foundation of Jiangsu under Grant: BK20191327, Zhejiang Lab (NO. 2021KC0AB04), the Guangdong Major Project of Basic and Applied Basic Research under Grant: 2019B030302002, and the Natural Science Foundation of Guangdong under Grant: 2018B030312002. Yinjin Fu and Nong Xiao are the corresponding authors.

## References

[1] E. Burgener, “Big memory computing emerges to better enable data-intensive it.” IDC Technology Spotlight, Tech. Rep., 2020.
[2] I. Corp., “The challenge of keeping up with data,” Intel® Optane™ DC Persistent Memory Product Brief., Tech. Rep., 2019.
[3] E. Ebrahimi, R. Miftakhutdinov *et al.*, “Parallel application memory scheduling,” in *MICRO*. ACM, 2011, pp. 362–373.
[4] J. Yang, J. Kim *et al.*, “An Empirical Guide to the Behavior and Use of Scalable Persistent Memory,” in *FAST*, 2020, pp. 169–182.
[5] H. Servat, A. J. Peña *et al.*, “Automating the Application Data Placement in Hybrid Memory Systems,” in *CLUSTER*, 2017, pp. 126–136.
[6] I. B. Peng, R. Gioiosa *et al.*, “RTHMS: A tool for data placement on hybrid memory system,” in *ISMM*. ACM, 2017, pp. 82–91.
[7] L. E. Ramos, E. Gorbatov, and R. Bianchini, “Page placement in hybrid memory systems,” in *ICS*. ACM, 2011, pp. 85–95.
[8] T. Rawat and A. Shrivastava, “Enabling multi-threaded applications on hybrid shared memory manycore architectures,” in *DATE*. EDA Consortium, 2015, pp. 742–747.
[9] W.-k. S. Yu, R. Huang *et al.*, “SRAM-DRAM hybrid memory with applications to efficient register files in fine-grained multi-threading,” in *ISCA*. ACM, 2011, pp. 247–258.
[10] B. Daase, L. J. Bollmeier *et al.*, “Maximizing Persistent Memory Bandwidth Utilization for OLAP Workloads,” in *SIGMOD*, 2021, pp. 339–351.
[11] S. Imamura and E. Yoshida, “The Analysis of Inter-Process Interference on a Hybrid Memory System,” in *HPCAsia*. ACM, 2020, pp. 1–4.
[12] A. van Renen, L. Vogel *et al.*, “Building blocks for persistent memory,” *The VLDB Journal*, vol. 29, no. 6, pp. 1223–1241, 2020.
[13] A. Jog, E. Bolotin *et al.*, “Application-aware Memory System for Fair and Efficient Execution of Concurrent GPGPU Applications,” in *GPGPU*, 2014, pp. 1–8.
[14] R. Ausavarungnirun, V. Miller *et al.*, “Mask: Redesigning the gpu memory hierarchy to support multi-application concurrency,” in *ASPLOS*. ACM, 2018, p. 503–518.
[15] S. P. Muralidhara, L. Subramanian *et al.*, “Reducing memory interference in multicore systems via application-aware memory channel partitioning,” in *MICRO*, 2011, pp. 374–385.
[16] Parsec benchmark suite, https://parsec.cs.princeton.edu/download.htm.
[17] The splash-2 programs. https://github.com/staceyson/splash2.
[18] The nas parallel benchmarks. https://www.nas.nasa.gov/software/npb.html.