CAUSE: Critical Application Usage-Aware Memory System using Non-volatile Memory for Mobile Devices

Yeseong Kim, Mohsen Imani, Shruti Patil and Tajana S. Rosing

Computer Science and Engineering
University of California, San Diego
Email: {yek048, moimani, patil, tajana}@ucsd.edu

***Abstract*—Mobile devices are severely limited in memory, which affects critical user-experience metrics such as application service time. Emerging non-volatile memory (NVM) technologies such as STT-RAM and PCM are ideal candidates to provide higher memory capacity with negligible energy overhead. However, existing memory management systems overlook mobile users application usage which provides crucial cues for improving user experience. In this paper, we propose CAUSE, a novel memory system based on DRAM-NVM hybrid memory architecture. CAUSE takes explicit account of the application usage patterns to distinguish data criticality and identify suitable swap candidates. We also devise NVM hardware design optimized for the access characteristics of the swapped pages. We evaluate CAUSE on a real Android smartphone and NVSim simulator using user application usage logs. Our experimental results show that the proposed technique achieves 32% faster launch time for mobile applications while reducing energy cost by 90% and 44% on average over non-optimized STT-RAM and PCM, respectively.**

I. Introduction

Modern mobile devices are expected to provide fast processing and near-instantaneous responses, however they operate under severe resource constraints due to low area and power budgets. In particular, DRAM-based main memory capacities are limited in state-of-the-art mobile phones (e.g., 1GB for iPhone 6). This limitation affects process service times, ultimately leading to user-experience degradation. As an example, in Android systems, mobile applications must be terminated when available memory is insufficient. Prior work [1] has observed that more than 15% of applications must be relaunched due to limited memory, resulting in slow application launching time. Solutions such as swap memory have been used in traditional OSes, but have not yet fully exploited in mobile systems due to slow speeds and low endurance of eMMC flash memory [3]. As memory demands of mobile applications continue to grow, support for larger memories with a low energy overhead is an important design requirement that affects the quality of user experience.

Non-Volatile Memory (NVM) technologies, such as Phase Change Memory (PCM) and Spin-Torque Transfer RAM (STT-RAM) are ideal candidates for dense, low-power mobile systems, since they can provide large memory space at a small energy cost [13], [14]. NVMs consume near-zero leakage power due to their non-volatility. While STT-RAMs offer comparable density and read performance, PCMs are 2-4X denser than DRAM [13]. However, both NVMs show extremely low write performance efficiency [8], making drop-in replacement of NVMs into the memory hierarchy challenging. On the other hand, NVMs have been shown to be promising candidates as swap memory blocks [3]. The swap functionality provides extended memory space as a backing storage by migrating (i.e. swapping out) recently-unused memory data to the swap memory area. While DRAM memory services most memory requests from the processor, NVMs provide augmented memory area with comparable read latencies for swapped-out data.

However, current swap management techniques do not consider application usage of real users. For example, the present swap policy cannot distinguish mobile applications launched specifically by the user among all running processes. Agnostic of applications of interest to the user, the swap policy treats pages from both foreground and background applications uniformly. This leads to frequent swapping out of critical pages from foreground interactive applications, whose processing time is paramount for user experience. Intuitively, in order to provide more intelligent memory management, mobile systems must consider the real users' application usage pattern. In this paper, we propose Critical Application Usage-aware Swap Enhanced memory system or *CAUSE*, that efficiently exploits application usage patterns in conjunction with NVM-based swap memory to achieve higher user experience with low energy overheads. Our approach utilizes the NVM as an extension of DRAM, and intelligently decides when and which pages should be migrated into the NVM by tapping into usage characteristics. At the high-level, CAUSE identifies critical memory pages based on application usage history. Our swap policy manages these pages so as to reduce their probability of being swapped-out. This allows pages from applications deemed as non-critical for user experience to become more suitable swap candidates, thereby reducing frequent app termination for applications of interest. On the hardware side, we propose a dual-partitioned NVM swap memory. We optimize the partitions for the access patterns expected for the swapped out pages. We evaluate NVM-based swap memory with both PCM and STT-RAM technologies to support the application-aware data migration with marginal power cost. Our experiments are conducted on the Qualcomm MDP8660

978-1-4673-8388-2/15/$31.00 ©2015 IEEE

smartphone [22] using actual application usage traces. The results show that the proposed memory management technique can save 90% and 44% of memory energy consumption over non-optimized STT-RAM and PCM, respectively, while providing better application launch time by 33%.

II. Related Work

Some prior efforts have leveraged non-volatile memories in mobile devices to address energy challenges. The memory design in [3], [4] shows that mobile device energy can be reduced by using NVM as the swap area. The authors analyze the increase in page accesses when the swap functionality is enabled, and compute the energy benefit when the swap area is running on the NVM. However, issues due to page migration are not addressed. Similar investigations which exploit NVM techology to build a hybrid memory system were proposed in [5], [6], however they didn't consider how such configurations are related to user experience in smart devices. In contrast to those work, our scheme facilitates user-aware page migration to provide better user experience. Chen et al. [9] propose a unified NVM and NAND flash memory architecture for mobile devices. Based on the observation that mobile application usage exhibits unique file access pattern in the history, files expected to be used in the near future are migrated from NAND Flash to emerging NVM such as STT-RAM and PCM. However, emerging NVMs provide orders of magnitude higher performance compared to NAND Flash [17]. Thus, NVM-based swap area can react to memory accesses at speeds comparable to DRAM. Therefore, CAUSE uses NVM as a backing storage for main memory instead of the NAND Flash memory.

Prior studies have also shown that user experience on mobile devices suffers due to the limited memory size. Wook et al. [1] observe that, in current Android platforms, many applications must be terminated and relaunched, resulting in poor user experience. Their technique uses a prediction model that determines which applications will be used in the near future based on the application usage pattern, and proposes an application eviction management policy that can reduce application restarting ratios. Yan et al. [2] also proposes a predictive application pre-launching technique that leverages user's context behavior for application usage patterns to provide better application launching time. Our proposed design is different from these efforts in that our approach exploits the fast access times provided by NVMs to address these issues. Kim et al. [7] propose a technique that reduces application launching time by storing frequently accessed applications and libraries to NVMs for startup. However, the proposed design is fundemantally different in that the CAUSE technique enables the swap management and avoids unexpected application process terminations, resulting in preserving application state, such as loaded webpages and played game levels, as well as reducing launching time. Furthermore, CAUSE also identifies criticality of different applications by taking advantages of users' application usage pattern.

III. Critical application usage-aware swap memory architecture

CAUSE proposes a novel swap policy in the swap management system, and NVM design for its hardware support.

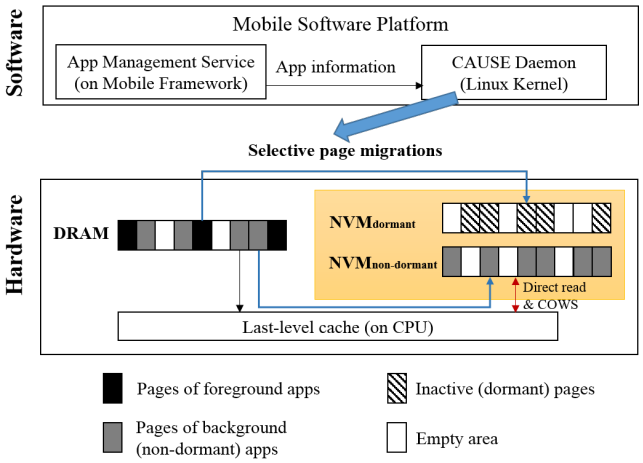

Fig. 1. An architectural overview of CAUSE

In this section, we first explain how the existing swap policy migrates the memory data to the dedicated swap area, and then, describe the CAUSE design in two parts from the software and hardware perspective.

A. Existing swap memory management

The default swap memory management utilizes DRAM-based main memory and a dedicated swap memory area, typically located in the disk. The memory space consists of a number of fixed-size memory units, or *pages*, whose size is typically 4KB in modern Linux systems. Pages are migrated between DRAM and swap memory if the available main memory is insufficient. The swap memory management utilizes a swap policy to determine which pages should be migrated into the swap area.

We briefly describe the default swap policy here. In Linux systems, memory management framework exploits a pseudo LRU-based mechanism using two page lists, called the active list and the inactive list. The pages of the inactive list are considered as the candidates to be swapped out. If a page in the inactive list is accessed, the page is moved to the active list by assuming that the page is likely to be used in the near future. On the other hand, when the inactive list becomes too small compared to the active list, the pages that have been least recently added to the active list are moved to the inactive list, balancing the two lists. When a new application is launched, if the current available memory is insufficient, the swap policy selects the pages of the inactive list to be evicted from the main memory in order of least-recently-added page.

A major limitation of this policy is that it does not consider different user experience-related requirements of pages. For example, in the Android platform, a large number of background service and widget applications run periodically. The pages accessed by these applications are moved to the active list since they appear as recently accessed pages. This interferes with the recently used pages from foreground applications actively being used by the user, which hold higher impact for user experience compared to background applications. As a result, pages from foreground applications become the victim pages to free memory space for new incoming pages. If

the swap functionality is not activated in Android platform, some applications are terminated even though they have been recently used in foreground [3], known as the low memory killer mechanism. On the other hand, even if swap policy is enabled, the systems are likely to swap out recent pages belonging to foreground applications.

B. Overview of CAUSE

The design goal of the proposed CAUSE is to efficiently exploit the swap functionality in the mobile device by considering the user's actual application usage pattern. An important design issue is that information about application usage is typically available at the application layer, i.e. in the high-level system software, while the swap management and hardware support is performed at the low-level, i.e. kernel or hardware level. To address this, we take a cross-layer approach in the design of CAUSE.

Figure 1 shows an architectural overview of CAUSE. The application (or 'app') management service distinguishes the applications that have been recently launched by users from the other running applications. In the Android framework, the higher layer system service such as the Activity Manager can recognize applications launched by users. CAUSE utilizes this information to track the applications recently launched in foreground. Then, the app management service sends the application information, including process IDs and types of the running applications to the CAUSE daemon in the Linux kernel, to inform the daemon of the applications that are likely to be used by the user in the near future. On the other hand, memory pages related to not-recently-launched applications are deemed to be non-critical to the user experience, i.e. their application launch time and process service time are not required to be optimized. Therefore, the CAUSE daemon migrates these pages into the inactive page list, marking them as candidates to be swapped out to NVM. This procedure is elaborated in Section 3.3.

Our swap policy migrates pages to the NVM when main memory must be freed for other incoming pages. We consider two possible types for each page that is swapped to the NVM: *dormant* and *non-dormant*. *Dormant* pages are those that are swapped out by the default swap policy because they are indeed not recently used. In contrast, the *non-dormant* pages belong to background and widget applications, and are likely to be accessed soon (or periodically) in the future. Thus, dormant and non-dormant pages show significantly different characteristics in terms of the page access pattern. To cater to these two types of access patterns, we partition the swap memory in two parts, *NVMdormant* and *NVMnon-dormant*, each designed with a different optimization goal. We describe this hardware design in Section 3.4. When a read request for a swapped page is initiated by the CPU, this is serviced directly from the swap area without copying memory to the DRAM. This is also called direct read and copy-on-write swap-out (COWS) mechanism, described in [4]. Since the read latency of optimized NVM is close to that of DRAM, this provides fast access to the swapped out pages, alleviating the performance loss due to main memory capacity limitation.

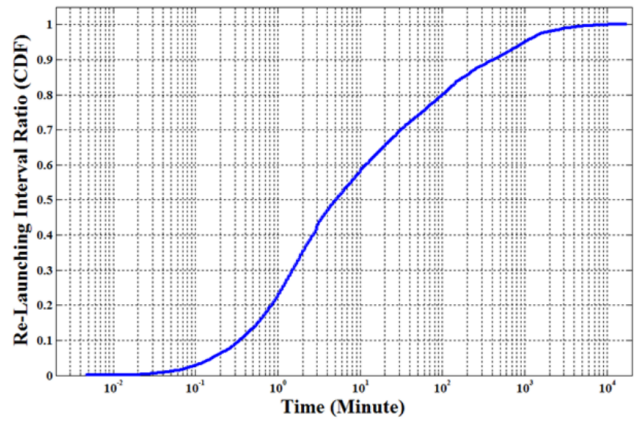

Fig. 2. Application relaunching pattern for 10 users

C. Software design of CAUSE

The software component of CAUSE handles swap page migration between DRAM to dual-partitioned NVM, based on application types. CAUSE seeks to retain pages from applications launched by users into DRAM for as long as possible since they are highly likely to be relaunched. Whenever an application is started or stopped, the app management service informs the Linux kernel of the list of process IDs of running applications and their types. Applications are of the type *critical* or *non-critical*. The *critical* type means the apps which are likely to be used by users in the near future. The non-critical apps are assumed not to be used by user, but their pages might be accessed by the background and widget applications. The proposed CAUSE identifies the two application types based on the user's application launching pattern. In order to understand application launching patterns, we analyzed the application usage logs which were collected from a user study [1] that monitored when mobile applications were launched by the user. Based on the collected logs for two weeks from 10 users, we investigated when each application was relaunched as foreground applications by the user. We define a metric for application relaunches, called re-launching interval which refers the time interval between two application launches for a certain application. For example, if a browser application was executed at T1 and the application was re-launched at T2, the time interval is computed by (T2-T1). Figure 2 summarizes the analysis result of re-launching intervals in a CDF graph. The results show that, once an application is launched, the application is highly likely to be used again in the near future. For example, 80% of applications were reused by the user within 100 minutes. Based on this result, we assume that the applications which are not used within 100 minutes will be not used again in the near future. We classify such applications as non-critical applications. Based on this information, the CAUSE daemon decides whether a page should be a swap candidate.

The CAUSE daemon is responsible for two functions: inactive list management and freeing memory pages. Algorithm 1 and 2 describes how these two functions enable swap page migration in the kernel layer. Algorithm 1 is triggered when new application information is sent by the app management service. For each new application, it first checks whether the

Algorithm 1: Inactive list management

```
1  // 𝔸_list is the list of applications
2  // 𝒜 is an application of 𝔸_list
3  for each 𝒜 in 𝔸_list do
4      if isNonCritical(𝒜) then
5          // 𝒫 is a page of 𝒜
6          for each 𝒫 of 𝒜 do
7              if 𝒫 in active_list then
8                  deactive_page(𝒫)
9              end
10         end
11     end
12 end
```

application is classified as non-critical application (Line 3 to 4). Since these applications do not significantly affect to the user experience, their pages are moved to the inactive list to provide more memory space to critical applications (Line 6 to 8).

Algorithm 2: Freeing memory pages

```
1  // 𝒫 is a page of inactive list
2  for each 𝒫 in inactive_list do
3      if isPageOfNonCritical(𝒫) then
4          swap_out_page(𝒫, NVM_nondormant)
5      else
6          swap_out_page(𝒫, NVM_dormant)
7      end
8      if isMemorySufficient() or isSwapFull() then
9          Break
10     end
11 end
12 if ¬ isMemorySufficient() then
13     // Run the low memory killer
14     do_shinker()
15 end
```

When the system needs to secure more free memory, Algorithm 2 is activated on top of the existing swap policy. Since the CAUSE architecture provides two types of NVM swap regions, this algorithm also decides where the page should reside in the swap memory, based on the application type that contains the swapped-out page (Line 3 to 7). This procedure executes until enough memory has been freed, or until the swap area is full (Line 8 to 10). If the system still needs more free memory, it executes the low memory killer implemented as part of the original system.

D. Hardware design of CAUSE

CAUSE is supported by NVM swap memory composed of emerging technologies such as STT-RAM or PCM. These are byte-addressable, low power, and fast access memories compared to the non-volatile NAND-Flash technology. However, the two technologies differ in their endurance[1]. STT-RAM can endure $10^{10} - 10^{15}$ write requests, which is orders of magnitude higher than PCM whose endurance is around $10^5 - 10^9$ [13], [14]. In this paper, we evaluate the performance and power benefits of both candidate technologies as NVM swap memories.

[1]The number of times a memory cell can be rewritten

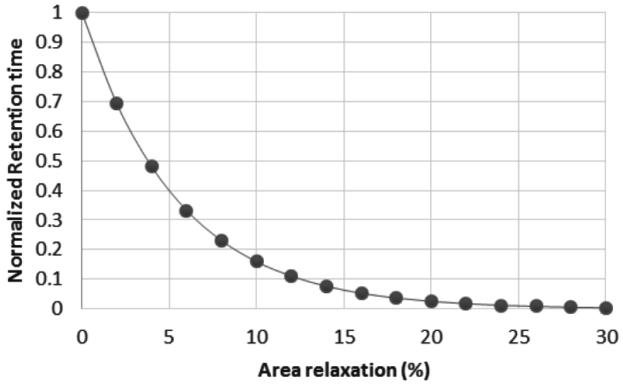

Fig. 3. MTJ retention time with reduced cell area

Several power and performance optimizations are possible for these memories when architecturally acting as NVM swap area alongside the main memory. In particular, we perform row buffer optimization with both PCM and STT-RAM to improve the access latency of the swap region. In addition, STT-RAM are amenable to other optimization opportunities such as retention time relaxation as described below.

Retention time optimization Data retention time is a well-known metric of how long a non-volatile memory cell can maintain the written bit without unintended bit-flips. The retention time of STT-RAM cell, more commonly known as Magnetic Tunnel Junction (MTJ) is proportional to its thermal stability factor. The more thermally stable an MTJ is, the longer it retains written data. The thermal stability factor is proportional to the volume of a MTJ cell [**?**], [15]. With reduced planar-area of free layer or the thickness of the MTJ, the retention time decreases dramatically. Trading off retention time by reducing area (also called as 'relaxed STT-RAM' [15]) is advantageous for power and performance. In particular, reduced retention time lowers both, the write current required in MTJ and the write latency.

Relaxed STT-RAM can be nearly as fast as DRAM in write performance while saving a significant amount of write energy [15]. Figure 3 shows data retention time for varying degrees of area relaxation, compared to an MTJ designed for retention time of ten years [19]. This relaxation can either decrease the MTJ write current by 65% while maintaining the same write latency, or improve the write latency by 83% for the same write current. In this paper, we decrease the free layer area of MTJ by 20%, reducing the data retention time of MTJ from ten years to one month. If higher retention time is required, a refresh scheme similar to DRAM can be used.

Row buffer optimization In NVM structures, row buffers have been proposed as temporary storage to speed up the write/read operations in NVMs [16]. The row buffers are composed of DRAM cells and can be optimized for area or latency [18], trading-off leakage and latency characteristics of the swap memory. In Table 1 and 2, we illustrate these improvements with respect to both types of optimizations for

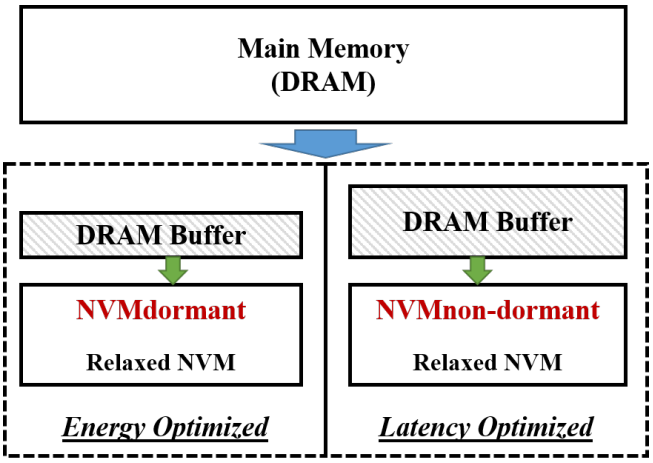

Fig. 4. Hardware architecture design of CAUSE

TABLE I. STT-RAM CHARACTERISTICS WITH AREA-OPTIMIZED BUFFER DESIGN

	STT-RAM			
	32MB	*64MB*	*128MB*	*256MB*
Read Latency	28.702ns	46.241ns	67.597ns	118.151ns
write latency	26.45ns	67.630ns	73.04ns	184.81ns
Read Energy	50.742pJ	84.435pJ	126.774pJ	273.936pJ
Write Energy	83.327pJ	134.052pJ	229.492pJ	493.931pJ
Leakage Power	29.175uW	43.15uW	53.65uW	84.162uW

TABLE II. STT-RAM CHARACTERISTICS WITH LATENCY-OPTIMIZED BUFFER DESIGN

	STT-RAM			
	32MB	*64MB*	*128MB*	*256MB*
Read Latency	9.045ns	22.423ns	24.775ns	79.058ns
write latency	18.932 ns	57.056ns	61.590ns	226.668ns
Read Energy	53.183pJ	86.024pJ	129.338pJ	278.472pJ
Write Energy	84.462pJ	136.637pJ	234.968pJ	500.058pJ
Leakage Power	147.592uW	295.561uW	310.482uW	593.722uW

STT-RAM based swap memory of varying sizes. Results are obtained using the NVSIM tool [18]. The latency optimization significantly decreases the write and read latencies, but has area and leakage power overhead. On the other hand, area optimization reduces the area and leakage power of buffer with huge overhead on read and write latency. Thus, performance-optimized buffers are suitable for frequently accessed memory blocks where latencies are important, while area-optimized buffers are more suitable for memory blocks with low access activity.

As explained in Section 3.2, swapped pages are of two types, dormant or non-dormant depending on the cause of swap. The two types are characterized by diverse access patterns, either frequent read/writes for non-dormant pages, or few to none read/write requests for dormant pages. Thus, the two types of pages benefit from memory optimized for their specific characteristics. In particular, we optimize the non-dormant region (NVMnon-dormant) for performance, and the dormant region (NVMdormant) for power and area. In CAUSE, we achieve this optimization by tuning the row buffer size and the type of optimization applied to the buffers. For example, in 32MB STT-RAM memory, latency optimized row buffer uses 1KB buffer to achieve low latencies, while in dormant region, the area optimization decreases the size of row buffer to 64B which results in 1.7X energy improvement. This improvement is up to 3X for PCM. Similar results were obtained in [16]. Figure 4 shows the high-level architecture of the NVM swap memory. For non-dormant type pages, we expect frequent, periodic access requests, therefore, non-dormant is designed with high speed, low access energy and large row buffer size to achieve high performance. In contrast, NVMdormant is expected to contain inactive pages. Thus, we focus on optimizing the memory area based on power consumption. In our experiments, the ratio of non-dormant pages over the total migrated pages was found to be 40% or less for all user logs in our experiment. To ensure sufficient memory for non-dormant pages, we size the two swap regions equally. If any region becomes full, the incoming page is directed to the second region. If both regions are full, then applications are terminated as in the original scheme.

IV. EXPERIMENT RESULT

A. Experimental setup

We implement the CAUSE system on a Qualcomm MSM8660 smartphone running Android 4.1 with Linux kernel 3.0.6. Since NVM memory chips are not readily available in the market, we partitioned a main memory area to be used as the swap area using *ram disk* [20]. The size of the main memory on the device is 1GB. We partition this memory so that we have 768MB of effective main memory, while the remaining is used for NVM simulation. The NVM is mounted as a block device and simulated with varying memory sizes (up to 256MB), which include the two regions. We collected detailed memory access traces of the swap area using *blktrace* [21]. We then evaluate latency and power consumption of the collected memory trace using NVSim. In the experiment, we executed a one-day application launch log, out of two weeks, from each user by employing a custom application usage replayer environment used in [1]. The launch logs include 20 Android applications, which cover the number and types of applications of the user logs, such as Facebook, Browser, News, Messenger and Camera, representing realistic application workloads. To account for the effect of background services and widget activities, we also activated custom background service applications. The custom application replayer launched the applications and background services according to the actual launching sequences of the collected 10 user logs. However, an exact execution of the one-day logs require long simulation times. To reduce experiment times, we scaled down the running period of each log so that all of them can be executed within approximately 20 minutes. Since application launching is the dominant factor that affects the swap policy or the application termination mechanism, we could faithfully evaluate the logs even with scaled down execution times.

B. Energy saving

The main goal of the proposed CAUSE is to provide higher memory capacity without significant energy cost. In order to understand how much energy is required to employ the CAUSE design, we evaluated the energy consumption of different memory technologies while reproducing the logs. Figure 4

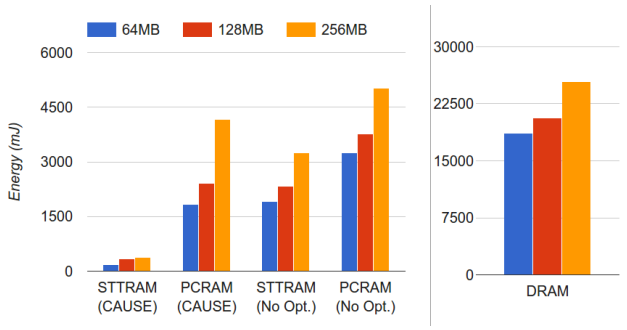

Fig. 5. Energy consumption of CAUSE

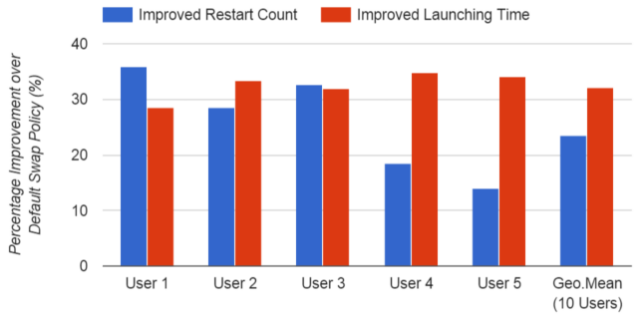

Fig. 6. Application launching experience improvement

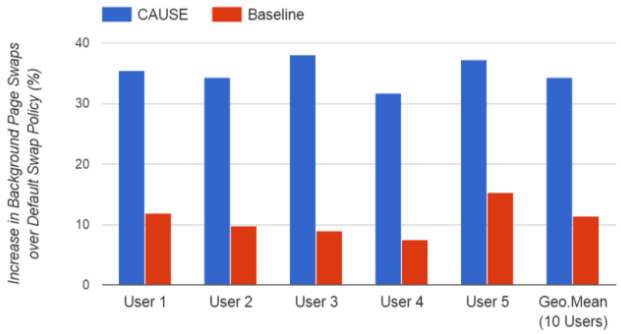

Fig. 7. Background page balancing

presents the average energy consumption results for all 10 users. The energy consumption of the DRAM is separately plotted on a different scale. The result shows that the energy cost to employ the CAUSE systems is very marginal. For example, we can observe that with 64MB PCM, CAUSE uses 10.2X lower energy compared to the DRAM-based design. This power saving primially comes from the lower static power offered by NVM technologies. In addition, the result shows the benefit of the hardware optimization of the CAUSE systems. For example, for the 64MB case, the CAUSE shows energy saving by about 90% and 44%, for the STT-RAM and PCM case respectively. This result shows that our hardware design and optimization strategy is effective in achieving a high energy efficiency. By considering access characteristics for different applications, the CAUSE design could further improve the energy efficiency compared to the exsiting swap apporaches based on non-optimized NVMs, and also allow larger memory space for the user-related applications.

C. Launch experience improvement

Another goal of CAUSE is to provide better user experience by allowing more foreground applications of interest to reside in main memory. This reduces the number of applications that are killed (and subsequently relaunched) due to the limited memory size. This, in turn, reduces the launch time of applications when they are recalled as active foreground applications, improving the user experience. In order to evaluate the improvements in launch time as well as the reduced restart count, we esimated the application launch time based on the DRAM memory available in the Android platform. Figure 6 shows the percentage improvement enabled by CAUSE in the application restart count and the launch time, using the default kernel, which does not use the swap policy, as the baseline. The result shows that, on average, 23.4% of application relaunches can be reduced by allowing larger memory space for the foreground applications. The improvement results in shorter application launch delays. As shown in the result, CAUSE can improve the launching time by 32% on average. As the NVM technology matures with respect to access time, we believe that the proposed CAUSE system can provide higher user experience.

D. Background page balancing

Lastly, we evaluate the effectiveness of our software migration policy. The goal of the foreground application-aware migration policy is to bias migration towards non-critical applications. Therefore, we analyze the number of background pages that are migrated by CAUSE compared to the default swap policy which does not consider criticality of applications. As shown in Figure 7, CAUSE migrated 23% more pages of background applications over the default policy. The results show that, since more number of the pages of background applications are moved to the swap area, CAUSE can guarantee more space to the foreground applications, resulting in better process service time for the users.

V. Conclusion

In this paper, we proposed a novel memory systems for smart devices, called CAUSE, which leverages NVMs for the swap functionality. By considering application usage pattern of users, CAUSE intelligently manages page migration between main memory and swap memory, in order to provide better user experience with respect to application launch time. In addition, the CAUSE system carefully optimizes NVM hardware to incorporate different requirements of migrated pages. We evaluated the CAUSE system over different memory technologies using real user traces on a real Android smartphone device. The CAUSE approach can be extended and improved in various ways. From a software perspective, we believe that accurate application launching prediction techniques based on the user context can help to make better decisions about which pages are migrated. Our future work also includes exploiting and optimizing NVMs for memory management in other units in the mobile SoC, such as the display.

ACKNOWLEDGMENT

This work has been supported by National Science Foundation(NSF) SHF grant 1218666.

REFERENCES

[1] Song, W., Kim, Y., Kim, H., Lim, J., and Kim, J., "Personalized optimization for android smartphones," ACM Transactions on Embedded Computing Systems (TECS), 2014, 13(2s), 60.

[2] Yan, T., Chu, D., Ganesan, D., Kansal, A., and Liu, J., "Fast app launching for mobile devices using predictive user context," in Proceedings of the 10th international conference on Mobile systems, applications, and services (pp. 113-126), 2012, ACM.

[3] Zhong, K., Zhu, X., Wang, T., Zhang, D., Luo, X., Liu, D., and Sha, E. H. M., "DR. Swap: energy-efficient paging for smartphones," in Proceedings of the 2014 international symposium on Low power electronics and design (pp. 81-86), 2014, ACM.

[4] Zhong, K., Wang, T., Zhu, X., Long, L., Liu, D., Liu, W., and Sha, E. H., "Building high-performance smartphones via non-volatile memory: The swap approach," In Embedded Software (EMSOFT), International Conference on), 2014, pp. 1-10, IEEE.

[5] Duan, R., Bi, M., and Gniady, C. "Exploring memory energy optimizations in smartphones," In Green Computing Conference and Workshops (IGCC), 2011, pp. 1-8, IEEE.

[6] Abe, K., Noguchi, H., Kitagawa, E., Shimomura, N., Ito, J., and Fujita, S., "Novel hybrid DRAM/MRAM design for reducing power of high performance mobile CPU," In Electron Devices Meeting (IEDM), 2012, pp. 10-5, IEEE.

[7] Kim, H., Lim, H., Manatunga, D., Kim, H., and Park, G. H., "Accelerating Application Start-up with Nonvolatile Memory in Android Systems," Micro, IEEE, 2015, 35(1), 15-25.

[8] Lee, B. C., Ipek, E., Mutlu, O., and Burger, D., "Architecting phase change memory as a scalable dram alternative," ACM SIGARCH Computer Architecture News, 2009, 37(3), 2-13.

[9] Chen, R., Wang, Y., Hu, J., Liu, D., Shao, Z., Guan, Y., Unified Non-Volatile Memory and NAND Flash Memory Architecture in Smartphones. in ASP-DAC, 2015, ACM.

[10] Z. Diao, Z. Li, S. Wang, Y. Ding, A. Panchula, E. Chen, et al., "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory," Journal of Physics: Condensed Matter, vol. 19, p. 165209, 2007.

[11] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, et al., "Cache revive: architecting volatile STT-RAM caches for enhanced performance in CMPs," in Proceedings of the 49th Annual Design Automation Conference, 2012, pp. 243-252.

[12] S. Ahn, Y. Song, C. Jeong, J. Shin, Y. Fai, Y. Hwang, et al., "Highly manufacturable high density phase change memory of 64Mb and beyond," in Electron Devices Meeting, 2004. IEDM Technical Digest. IEEE International, 2004, pp. 907-910.

[13] Bheda, Rishiraj A., et al., "Energy efficient phase change memory based main memory for future high performance systems," in Proceedings of Green Computing Conference and Workshops (IGCC), 2011 International, IEEE, 2011.

[14] Kultursay, Emre, et al., "Evaluating STT-RAM as an energy-efficient main memory alternative," in Proceedings of Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on., IEEE, 2013.

[15] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in Proceedings of High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on, pp. 50-61, 2011.

[16] Meza, Justin, Jing Li, and Onur Mutlu., "A case for small row buffers in non-volatile main memories," in Proceedings of Computer Design (ICCD), 2012 IEEE 30th International Conference on., IEEE, 2012.

[17] Jeffrey C. Mogul, Eduardo Argollo, Mehul Shah, and Paolo Faraboschi., "Operating system support for NVM+DRAM hybrid main memoryr," in Proceedings of the 12th conference on Hot topics in operating systems (HotOS'09), 2009, USENIX Association, Berkeley, CA, USA, 14-14.

[18] Dong, Xiangyu, et al., "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2012, 994-1007.

[19] Xuanyao Fong, Sri Harsha Choday, Panagopoulos Georgios, Charles Augustine, Kaushik Roy, "SPICE Models for Magnetic Tunnel Junctions Based on Monodomain Approximation," NanoHub, 2013

[20] Linux ram disk overview, http://www.ibm.com/developerworks/library/l-initrd/

[21] blktrace User Guide, http://www.cse.unsw.edu.au/∼aaronc/iosched/doc/blktrace.html

[22] Qualcomm MSM8660, https://developer.qualcomm.com/mobile-development/development-devices-boards/mobile-development-devices/snapdragon-mdp-legacy-devices