Power Modeling of Solid State Disk for Dynamic Power Management Policy Design in Embedded Systems

Jinha Park, Sungjoo Yoo, Sunggu Lee, Chanik Park†

Department of Electronic and Electrical Engineering,
POSTECH (Pohang university of Science and Technology)
790-784 Hyoja-dong, Nam-gu, Pohang, Korea
{litrine, sungjoo.yoo, slee}@postech.ac.kr

†Flash Software Team, Memory Division, DS Solution
Samsung Electronics
Hwasung-gun, Gyeonggi-do, Korea
ci.park@samsung.com

Abstract. Power consumption now becomes the most critical performance limiting factor to solid state disk (SSD) in embedded systems. It is imperative to devise design methods and architectures for power efficient SSD designs. In our work, we present the first step towards low power SSD design, i.e., power estimation of SSD. We present a practical approach of SSD power estimation which tries to keep the advantage of real measurement, i.e., accuracy, while overcoming its limitations, i.e., long execution time and lack of repeatability (and high cost) by a trace-based simulation. Since it is based on real measurements, it takes into account the power consumption of SSD controller as well as that of Flash memories. We show the effectiveness of the presented method in designing a dynamic power management policy for SSD.

Keywords: Solid state disk, power consumption, measurement, trace-based simulation, dynamic power management, low power states

1 Introduction

Flash-based storage is becoming more and more popular in embedded systems such as smart card, smart phone, net book, labtop, etc. as well as in desktop PC and server. Compared with conventional non-volatile memory, namely hard disk drive (HDD), Flash-based storage can give several advantages, e.g., high performance, low power consumption, reliability, etc. Especially, solid state disk (SSD) is becoming a major non-volatile memory while replacing HDD in smart phones (e.g., iPhone 3GS with 32GB Flash memory) and net books as well as in notebook PCs and servers [1].

1.1 Low Power SSD Design Problem

SSD is inherently more power efficient than HDD since HDD requires large driving power for running mechanical parts, i.e., motors for spindle and head as well as performing I/O operations (between disk, DRAM buffer and host). However, SSD does not require the mechanical parts, but consumes power only for the electrical ones.

SSD can achieve higher performance than HDD mostly by parallel accesses to relatively low speed Flash devices (e.g., achieving a throughput higher than 240MB/s by accessing 8 Flash devices with 33Mbytes/sec each). High performance SSD inherits the same level of power consumption constraints that traditional HDD has in embedded systems. For instance, SSD has a peak power budget of about 1A and an average power consumption budget of about 1.2W in typical notebook PCs [2] and is expected to have much lower power budget in smart phones.

Further performance improvement in SSD will require more power consumption especially due to more aggressive parallel accesses to Flash devices. Such an aggressive parallel scheme is not easily applicable due to the given power consumption constraints of embedded systems. However, there is no more large room in peak and average power budget to be used, by aggressively parallel accesses, for further performance improvement of SSD in embedded systems. Power consumption now becomes the most critical performance limiting factor in SSD. It is imperative to devise design methods and architectures for power efficient SSD designs.

There has been little work on low power SSD designs. In our work, we present the first step towards low power SSD design, i.e., power estimation of SSD. We also report our application of the power estimation method to a low power SSD design where the parameter of dynamic power management is explored in a fast and cost effective way with the help of the presented power estimation method.

1.2 Power Estimation of SSD

The design space for low power SSD design will be huge due to various possible design choices, e.g., parameter sets in dynamic power management (e.g., time-out parameters, DPM policies, etc.), Flash Translation Layer (FTL) algorithms and SSD controller architectural parameters (e.g., I/O frequency in Flash devices, DRAM buffer size and caching/prefetch methods in the controller, etc.).[1] When exploring the design space, there can be two typical methods of evaluating the power consumption of design candidates: real measurement and full system simulation with a power model. Real measurement, which gives accurate power information, is in use in SSD product designs. There are two critical problems with real measurements: long design cycle (e.g., hours of SSD execution is required for the evaluation of one design candidate) and changing battery characteristics over repeated runs [3].[2] The two

[1] In this paper, we consider only software-based solutions. There can be hardware design candidates such as # of channels, # ways/channel, DRAM capacity/speed, # CPUs, etc.

[2] Battery lifetime measurements require a procedure of fully charging and then completely discharging the battery while running the system. The battery characteristics degrade significantly after several times (~10 times) of such procedures. Thus, a new battery needs to be used for subsequent battery lifetime measurements.

problems prevent designers from performing extensive design space exploration which may require evaluating numerous design candidates. Thus, it will be impractical to evaluate all the choices with real SSD executions due to the long execution time and high cost of battery.[3]

The second candidate, cycle-level full system simulation with a power model, is prohibitive due to tool long a simulation runtime. Assuming a 200MHz SSD controller and ~100 Kcycles/sec of simulation speed, it may take 125 days for the simulation of less than 1.5 hours of real SSD execution. Thus, SSD power estimation based on a detailed simulation model may not be practical in real SSD designs.

1.3 Our Contribution

In this paper, we present a practical approach of SSD power estimation which tries to keep the advantage of real measurement, i.e., accuracy, while overcoming its limitations, i.e., long execution time and lack of repeatability (and high cost). The power estimation method takes as input real power measurements and SSD access trace. Then, it performs a trace-based simulation of SSD operation gathering the information of power consumption. Finally, it gives as output power profile (power consumption over time). Since it is based on real measurements, it takes into account the power consumption of SSD controller as well as that of Flash memories. The presented method of SSD power estimation gives fast estimation, via trace-based simulation, and accuracy, based on real power measurements. We also present an application of our method to designing a DPM policy for SSD.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces Flash memory and SSD operations. Section 4 explains the power estimation method. Section 5 reports experimental results including the application to DPM policy design. Section 6 concludes the paper.

2 Related Work

There have been several works for power characterization and optimization of HDD [4][5][6][7][8][9][10]. Hylick, *et al.* explain that the mechanical parts incur large overheads of power consumption especially when HDD starts to run the spindle and head [4]. Zedlewski, *et al.* present a performance and power model of HDD based on a Disk simulation model, DiskSim [5]. The HDD power model considers the entire HDD as a single entity and has four active-mode power states (seeking, rotation, reading and writing) and two idle-mode ones. IBM reports that fine-grained power states enable further energy reduction than conventional coarse-grained ones by enabling to exploit short idle periods to enter low power states more frequently and stay there for a longer period of time [6]. Lu, *et al.* compare several DPM policies for HDD [7]. Douglis, *et al.* offer a DPM policy where the time-out (if there is no new

[3] Statistical approximation and optimization methods, e.g., response surface model can also be applied to reduce the number of real executions.

access during the time-out, a low power state is entered) is determined adaptively based on the accuracy of previous time-out prediction [8]. Helmhold, *et al.* present a machine learning-based disk spin-down method [9]. Bisson, *et al.* propose an adaptive algorithm to adaptively calculate the time-out for disk spin down utilizing multiple time-out parameters and considering spin-up latency cost [10].

Regarding SSD, a performance model is presented only recently in [11]. However, there is little work on power characterization and modeling for SSD. In terms of power optimization in Flash-based storage, there are two categories of approaches depending on the optimization target between active and idle-mode power consumption. Joo, *et al.* present a low power coding scheme for MLC (multi-level cell) Flash memory which has value-dependent power characteristics (e.g., in case of 2 bit MLC, the power consumptions of coding two two-bit data, 00 and 01 are different) [12]. Recently, a low power solution is presented for 3D die stacking of Flash memory, where multiple Flash dies share a common charge pump circuit [13]. Regarding the idle-mode power reduction, in commercial SSD products [14], a fixed time-out and DIPM (device initiated power management) are utilized when the SSD detects an idle period and asks the host of a grant for transition to the low power state.

3 Preliminary: Flash Memory Operation and SSD Architecture

Typically, a Flash memory device contains, in a single package, multiple silicon Flash memory dies in a 3D die stacking [13]. The Flash memory dies share the I/O signals of device in a time-multiplexed way. We call the I/O signals of package *channel* and each memory die *way*.[4] A single Flash memory device can support up to the data throughput of data width*I/O frequency, e.g., 33MBps at 33MHz. In order to support higher bandwidth, we need to access Flash memory dies (ways) and devices (channels) in parallel. Fig. 1 shows an example of SSD architecture consisting of two channels (i.e., two Flash devices) and four ways (i.e., four Flash memory dies on a single device) per channel. The controller takes commands from the host (e.g., smartphone or notebook CPU) and performs its Flash Translation Layer (FTL) algorithm to find the physical page address(es) to access. Then, it accesses the corresponding channels and ways, if needed, in parallel. In terms of available parallelism, each way can run in parallel performing internal operations, e.g., internal read and program to transfer data between the internal memory cell array and its I/O buffer. However, the controller can access, at a time, only one way on each channel, i.e., the I/O signals of the package, in order to transfer data between the I/O buffer of the Flash memory die and the controller. Thus, the peak throughput is determined by the number of channels * I/O frequency.

One of salient characteristics in Flash memory is that no update is allowed on already written data. When a memory cell needs an update, it first needs to be erased before a new data is written to it. We call such a constraint “erase before write”. In order to overcome the low performance due to “erase before write”, log buffers (often called update blocks) are utilized and the new write data is written to the log buffers.

[4] We use two terms (channel and the I/O signals of device, and way and die) interchangeably throughout this paper.

The Flash translation layer (FTL) on the SSD controller maintains the address mapping between the logical data and the physical data [15][16][17]. In reality, the controller, to be specific, FTL, determines the performance and power consumption, especially, of random reads/writes. Thus, the controller overhead needs to be taken into account in the power estimation of SSD.

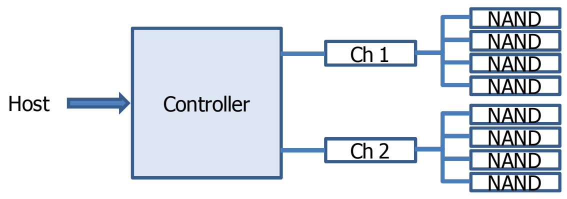


Fig. 1 Multi-channel/multi-way SSD controller architecture

4 SSD Power Estimation

The power estimation requires as input

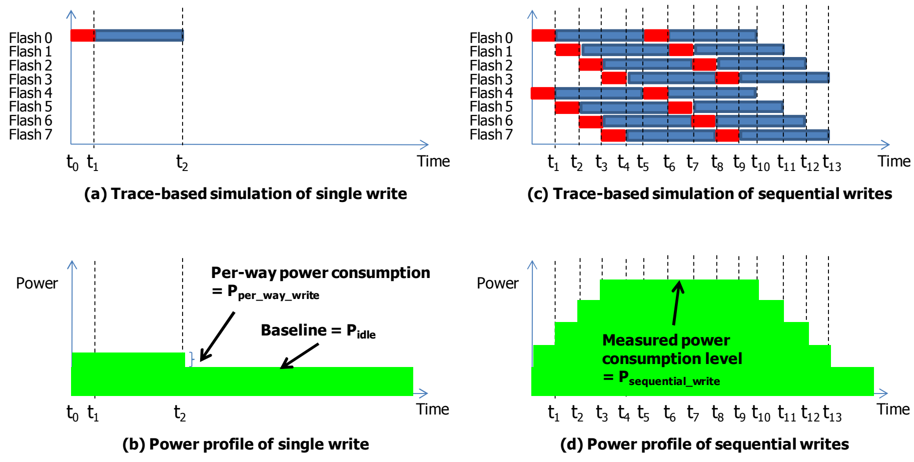
- Performance and power measurement data (read/write latency for each of read/write data sizes of 1/2/4/8/16/32/64/128/256 sectors, power consumption of sequential reads/writes, power consumption per power state (*idle*, *partial*, and *slumber*), and power state transition delay values),
- Information of SSD architecture (# channels, ways/channel, tR/tRC/tPROG/tERASE), and
- SSD access trace obtained from a real execution on the host system.

We perform a trace-based simulation of the entire SSD (controller as well as Flash memories) considering critical architectural resource conflicts at a channel level. Then, we obtain power profile over time as well as output execution trace (e.g., the status of each channel/way and latency of each SSD access). The trace-based simulation also allows for the simulation of DPM policy where power state transitions are performed based on the given DPM policy.

4.1 Performance and Power Modeling

Resource conflict modeling is critical in performance modeling. Given the performance data per SSD access size and the information of SSD architecture, we model the critical resource conflict at a channel level. To do that, we decompose the measured latency into two parts in terms of resource parallelism: the latency of Flash I/O and that of controller and Flash internal operation. We model the channel-level resource conflict with the Flash I/O latency since only one Flash I/O operation can be active at a time at each channel. However, the controller and Flash internal operation (read from cell array to I/O buffer, program, and erase) can be performed in parallel.

Fig. 2 (a) illustrates the decomposition of latency for a single one-sector SSD write operation (i.e., a SATA write command of one sector size) in the case of SSD architecture in Fig. 1. At time t_0, the controller transfers, via the corresponding channel, one sector data to the I/O buffer of target Flash die. After the I/O operation is finished at time t_1, we model that the controller and Flash program operations take the other portion of latency from time t_1 to t_2.



Flash 0
Flash 1
Flash 2
Flash 3
Flash 4
Flash 5
Flash 6
Flash 7
t0 t1 t2 Time
(a) Trace-based simulation of single write
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 Time
(c) Trace-based simulation of sequential writes
Power
Per-way power consumption = P_per_way_write
Baseline = P_idle
(b) Power profile of single write
Measured power consumption level = P_sequential_write
(d) Power profile of sequential writes

Fig. 2 Performance and power modeling

For the power modeling of active-mode operation, we decompose the power consumption into two parts: baseline and access-dependent parts. The baseline part corresponds to the power consumption of idle state when there is no SSD access in progress while the SSD controller is turned on. We measure the power consumption of idle state and use it as the baseline part. The access-dependent part is obtained by subtracting the baseline from the measured power consumption of sequential reads/writes. The access-dependent part is further decomposed into the power consumption of per-way operation.

Fig. 2 (b) illustrates the decomposition. Fig. 2 (b) shows the power profile for the case of Fig. 2 (a). The baseline, i.e. the power consumption of idle state is always consumed over the entire period in the figure (to be specific, until a transition to a low power state is made). The access-dependent part for a single write operation (its derivation will be explained later in this section) is added to the total power between time t_0 and t_2 during which the write operation, including Flash I/O, program, and controller execution, is executed. Due to the limitation in measuring power consumption, we do not make a further decomposition to separately handle each of Flash I/O, read, program, erase and controller execution. We expect more detailed power measurement will enable such a fine-grained decomposition to give more accurate power estimation, which will be left for our future work.

Figs. 3 (c) and (d) illustrate how we derive the power consumption of per-way operation from the access-dependent part of sequential reads/writes. Fig. 2 (c) shows the result of trace-based simulation for the case of 8 page sequential writes to the SSD of Fig. 1. At time t_0, t_1, t_2, and t_3, the controller starts to access one Flash die on each channel to transfer a page data from the controller to the I/O buffer of the Flash die.

Then, it initiates a program operation in the Flash die. After the latency of program and controller, the next Flash I/O operations start at time t_5, t_6, t_7, and t_8, respectively.

Fig. 2 (d) shows the corresponding power profile. First, the baseline portion covers the entire period. Then, we add up the contribution of each Flash die to the total power as the figure shows. Thus, we see the peak plateau from t_3 to t_{10} when all the eight Flash dies and controller are active. The measured power consumption of sequential writes corresponds to the power consumption at the plateau. Thus, we obtain the per-way power consumption of write operation ($P_{per\_way\_write}$) as follows.

$P_{per\_way\_write} = (P_{sequential\_write} - P_{idle})$/# active Flash dies at the plateau

The per-way power consumption of read operation is calculated in the same way.

4.2 Trace-based Simulation of Performance, Power and DPM Policy

The trace-based simulation covers DPM policy as well as performance and power consumption. Fig. 3 (a) illustrates how a given DPM policy is simulated in the trace-based simulation. In the figure, we assume that a time-out (TO)-based DPM policy is simulated. Thus, the SSD needs to enter a low power state after an idle period of TO since the completion of previous access. In the figure, at time t_{13}, an idle period starts and a TO timer starts to count down. At the same time, the power consumption drops to the level of idle state power consumption. At t_{14}, the TO timer expires and a low power state is entered. The figure shows that the total power consumption drops again down to the level of the entered low power state. At t_{15}, a read command for 8 pages arrives. However, since the SSD is in a low power state, it takes a wakeup time, T_{wakeup} to make a state transition to the active state. Thus, the read operations start at time t_{16} after the wakeup delay. Fig. 3 (b) shows the current profile obtained in the trace-based simulation.

Fig. 4 shows the pseudo code of trace-based simulation. The simulation is event-driven and handles three types of event: host command arrival (e.g., SATA read/write command), start/end of Flash operations (I/O operation, and read/program/erase operation), and power state transition (e.g., when a time-out counter expires). The simulation advances the simulation time, T_{now} to the time point when the next event occurs (line 2 in the figure). At the time point when there is any event (line 3), we select the event in the order of end → start → state transition, where 'end' and 'start' represent events for the end and start of Flash operation, respectively (line 4). If the selected event is a host command, then we run the FTL algorithm to find the corresponding physical page addresses (PPAs) (line 6). Note that, if there is no available log buffer space, then the FTL can call a garbage collection which schedules data move and erase operations on the corresponding channels and ways. The garbage collection method is specific to the FTL algorithm. Note also that the power consumption and runtime of FTL algorithm is included in the access-dependent part of per-way power consumption and the controller latency.

If the current state is a low power state (line 8), we need a state transition to the active state. Thus, during the wakeup period, we mark the power state as an intermediate state of state transition (PState = Transition2Active), set the total power to the idle state power consumption ($P_{total} = P_{idle}$), and remove, if any, a future event for transition to a low power state (lines 8—12).

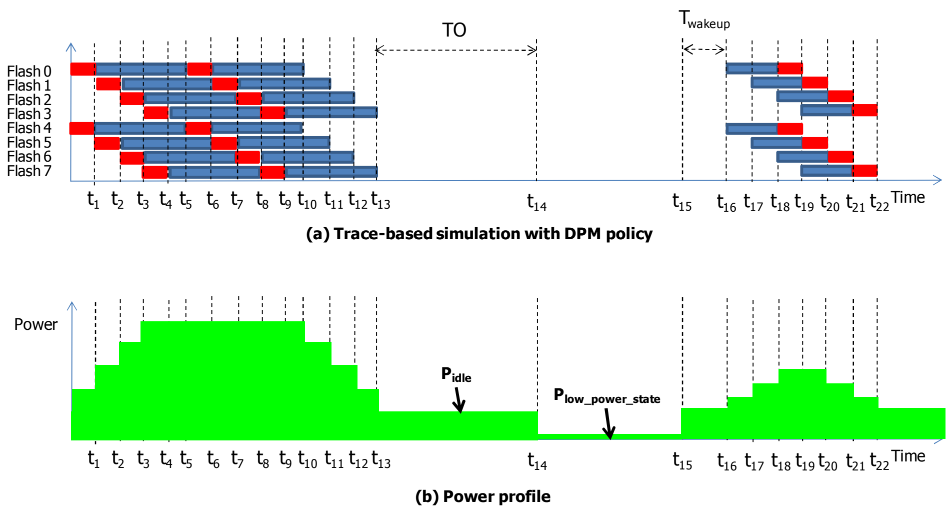


Fig. 3 Trace-based simulation of DPM policy

A host command can create 2 to 128 event pairs for the start/end of Flash operations. If it is a read or write command for a data size less than and equal to the page size, it creates two event pairs: one pair, <start, end> for controller and Flash internal read or write operation and the other pair, <start, end> for Flash I/O operation. The 128 event pairs are created by a host command for 64 page (256 sector) read or write. In the trace-based simulation, the created event pairs are scheduled by a function, Schedule_events_for_Flash_operations(PPAs, T_{init}), where PPAs is a list of physical page addresses obtained from the FTL algorithm run (on line 6). The function performs an ASAP (as soon as possible) scheduling algorithm to schedule the event pairs. Thus, it schedules each event pair at the earliest time point when the corresponding Flash channel (for Flash I/O operation) or way (for Flash internal operations and controller operation) becomes available. If the current state is a low power state, the scheduled time of the first event pair is adjusted to account for the wakeup delay (lines 11 and 13).

If the new event (selected on line 4) is a start event and the current power state is a low power state, then the power state is set to the active state (line 17). Then, the power consumption of newly started operation is added to the total power consumption (line 18). If the new event is an end event (line 19), then that of the just finished operation is subtracted from the total power consumption (line 20). If there is no more future event for Flash operation, then it means that there is no active Flash channel and way and an idle period starts. Thus, we can insert into this point the function of DPM policy under development. Since we assume a simple TO-based DPM policy in Fig. 4, we schedule a TO event at T_{now} + TO (line 23).

If the new event (selected in line 4) is a power state transition event (TO event in the DPM policy example), then the power state is set to the low power state (line 26).

The total power consumption is set to that of low power state (line 27). If there is any lower power state, i.e., in the case of TO-based DPM policy with more than one low power states, then we can schedule another TO event (at line 28). The entire trace-based simulation continues until all the input SSD accesses are simulated.

```
1 while($T_{now}$ < end of simulation) {
2   Advance time $T_{now}$ to the next event
3   while (any event at time $T_{now}$) {
4     new_event = pop(event_list($T_{now}$)) // pop the events of end of Flash operation first
5     If (new_event == host command)
6       Run FTL to find the corresponding PPAs
7       $T_{init} = T_{now}$
8       If (current status == low power state)
9         PState = Transition2Active
10        $P_{total} = P_{idle}$
11        $T_{init} = T_{now} + T_{wakeup}$
12        Clear future events for transition to low power state
13      Schedule_events_for_Flash_operations(PPAs, $T_{init}$)
14    Else if (new_event==start or end of Flash operation)
15      If (new event == start of Flash operation), then
16        If (PState==Transition2Active), then
17          PState=Active
18        Add the power consumption of the newly started operation to $P_{total}$
19      Else, // new_event = end of Flash operation
20        Subtract the power consumption of the just finished operation from $P_{total}$
21        If there is no more future event for Flash operation, then
22          // Insert DPM policy here. The following is a TO-based DPM policy example
23          Schedule a TO event at $T_{now}$+TO
24    Else // power state transition event
25      // TO event for a power state transition in the DPM policy example
26      PState = LowPowerState
27      $P_{total} = P_{low\_power\_state}$
28      // If there is any lower power state, then schedule a TO event here
29  } // end of "any event at time $T_{now}$"
30 }
```

Fig. 4 Pseudo code of trace-based simulation algorithm

6 Experiments

We designed the power estimator in Matlab. For the experiments, we used a Samsung SSD (2.5", 128GB, and SATA2)[5] [18]. As the input data of performance and power consumption, we used the measurement data obtained from the real usage of SSD on a notebook PC running Windows VISTA. For performance, we used the measured latency of read/write commands for the data sizes of 1, 2, 4, 8, 16, 32, 64, 128, and 256 sectors, respectively. We also used the measured power consumption of sequential reads/writes and that of low power states (power consumption was measured for the idle state and two low power states called *partial* and *slumber*). We collected the input traces of SSD accesses from the notebook PC by running three

[5] We assumed MLC, 4KB/page, 33MHz I/O frequency, 8 channels and 8 ways/channel based on the peak performance and capacity.

scenarios of MobileMark 2007: Reader, Productivity, and DVD [19]. We also used PCMark05 to collect a SSD access trace as heavy SSD usage case [20].

The accuracy of performance/power estimation was evaluated by comparing the estimation results and corresponding measurement data. The comparison showed that the trace-based simulation gives the same estimation results as measurement data in both power consumption (of sequential reads/writes) and latency (of all the read/write data sizes).

We applied the trace-based simulation to a time out (TO)-based DPM policy design for SSD with two low power states (*partial* and *slumber*). TO-based DPM is used in HDD [6] and SSD [14] products. DPM in SSD is different from that in HDD since DPM in SSD can exploit short idle periods (much shorter than a second) which could not be utilized in HDD due to the high wakeup delay of mechanical parts (in seconds). Thus, DPM in SSD can give more reduction in energy consumption by entering low power states more frequently.

TO-based DPM policy requires selecting a suitable TO which gives the minimum energy consumption over various scenarios. Fig. 5 shows performance estimation results obtained by sweeping the TO value for each of three MobileMark scenarios. In the TO sweep, for simplicity, we set two TO parameters (one for the transition from the active state to the first low power state *partial*, and the other for the transition from *partial* to the second power state *slumber*) to the same value. A single run of trace-based simulation takes 1~20 minutes depending on the number of accesses and TO values, which is 6~ 100+ times faster than real SSD runs.[6]

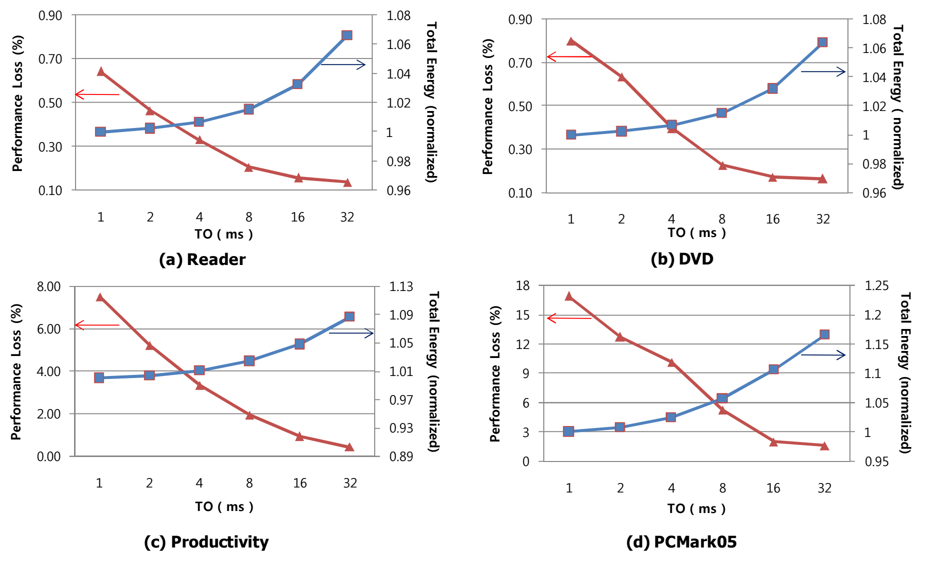


Fig. 5 MobileMark 2007 (a)~(c) and PCMark05 results (d)

[6] We expect faster trace-based simulation can be achieved when implementing the algorithm in C/C++ rather than in Matlab.

Fig. 5 shows that there can be a trade-off between energy reduction and performance drop. The general trend is that as TO increases, energy consumption increases since less idle periods are utilized for energy reduction while performance penalty (due to accumulated wakeup delay) decreases since there are less wakeups. As shown in the figure, in the case of MobileMark scenarios, the sensitivity of performance drop to TO is the most significant in the Productivity scenario. It is because Productivity scenario has 5.34 and 6.26 times the of SSD accesses of DVD and Reader scenarios, respectively. However, the absolute level of performance drop is significant in the case of Reader and DVD scenarios and moderate in the case of Productivity scenario. It is because MobileMark runtime is dominated by idle periods which occupy about 95% of total runtime. Thus, the performance impact of DPM policy is not easily visible with the MobileMark traces. However, users may experience performance loss due to aggressive DPM policies (e.g., short TO such as 1ms) when the SSD is heavily accessed. PCMark05 represents such a scenario where the host runs continuously, thus, accessing the SSD more frequently than MobileMark. Fig. 5 (d) shows the result of PCMark05 which incurs up to 16.9% performance drop in the case of aggressive DPM policy (TO=1ms). It is mainly because PCMark05 has 9.7 times more SSD accesses (per minute) than Productivity scenario.

Considering the results in Fig. 5, there can be a trade-off between energy reduction and performance drop which designers need to investigate in low power SSD design. As a final remark, Fig. 5 shows that there is still a large gap between the result of Oracle DPM (where we obtain maximum energy reduction without performance drop) and that of optimal single TO case. Our power estimation method will contribute to the design of sophisticated DPM policies to exploit the trade-off while closing the gap.

7 Conclusion

In this paper, we presented a power estimation method for SSD in embedded systems. It takes as input a SSD access trace and the measurement data of performance and power consumption of SSD accesses and gives as output the power profile considering the DPM policy under development. The presented method gives accuracy in power consumption based on real measurement data and fast simulation by applying a trace-based approach. We also presented a case study of applying the method to designing a DPM policy for SSD. As our future work, we will perform an extensive analysis on the accuracy of power estimation with the comparisons against real measurements on various scenarios.

Acknowledgement

This work was supported in part by Samsung Electronics.

References

1. Kim, B.: Design Space Surrounding Flash Memory. In: International Workshop on Software Support for Portable Storage, IWSSPS (2008)
2. Creasey, J.: Hybrid Hard Drives with Non-Volatile Flash and Longhorn. In: Windows Hardware Engineering Conference (WinHEC), MicroSoft (2005)
3. Communications with Samsung engineers.
4. Hylick, A., Rice, A., Jones, B., Sohan, R.: Hard Drive Power Consumption Uncovered. In: ACM SIGMETRICS Performance Evaluation Review, vol. 35, issue 3, pp. 54—55, ACM (2007)
5. Zedlewski, J., Sobti, S., Garg, N., Zheng, F., Krishnamurthy, A., Wang, R.: Modeling Hard-Disk Power Consumption. In: the USENIX Conference on File and Storage Technologies (FAST), pp. 217—230, USENIX Association (2003)
6. IBM.: Adaptive Power Management for Mobile Hard Drives. In: http://www.almaden.ibm.com/almaden/mobile_hard_drives.html, IBM (1999)
7. Lu, Y., De Micheli, G.: Comparing System-Level Power Management Policies. In: IEEE Design & Test of Computers, vol. 18, no. 2, pp. 10-19, IEEE (2001)
8. Douglis, F., Krishnam, P., Bershad, B.: Adaptive Disk Spin-down Policies for Mobile Computers. In: 2nd Symposium on Mobile and Location-Independent Computing, pp. 121-137. USENIX Association (1995)
9. Helmbold, D., Long, D., Sconyers, T., Sherrod, B.: Adaptive Disk Spin-Down for Mobile Computers. In: Mobile Networks and Applications, vol. 5, issue 4, pp. 285—297, Kluwer Academic Publishers (2000)
10. Bisson, T., Brandt, S.: Adaptive Disk Spin-Down Algorithms in Practice. In: the USENIX Conference on File and Storage Technologies (FAST), USENIX Association (2004)
11. Dirik, C., Jacob, B.: The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization. In: International Symposium on Computer Architecture, pp. 279—289, ACM (2009)
12. Joo, Y., Cho, Y., Shin, D., Chang, N.: Energy-Aware Data Compression for Multi-Level Cell (MLC) Flash Memory. In: Design Automation Conference, pp. 716—719, ACM (2007)
13. Ishida, K., Yasufuku, T., Miyamoto, S., Nakai, H., Takamiya, M., Sakurai, T., Takeuchi, K.: A 1.8V 30nJ adaptive program-voltage (20V) generator for 3D-integrated NAND flash SSD. In: International Solid-State Circuits Conference, pp. 238—239, IEEE (2009)
14. Intel: X25-M and X18-M Mainstream SATA Solid-State Drives. In: http://www.intel.com/design/flash/nand/mainstream/index.htm, Intel (2009)
15. Lee, S., Park, D., Jung, T., Lee, D., Park, S., Song, H.: A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation. In: ACM Transactions on Embedded Computing Systems (TECS), vol. 6, issue 3, ACM (2007)
16. Kang, J., Cho, H., Kim, J., Lee, J.: A Superblock-based Flash Translation Layer for NAND Flash Memory. In: the 6th ACM & IEEE International conference on Embedded software (EMSOFT), pp.161—170, ACM (2006)
17. Lee, S., Shin, D., Kim, Y., Kim, J.: LAST: locality-aware sector translation for NAND flash memory-based storage systems. In: ACM SIGOPS Operating Systems Review, vol. 42, issue 6, ACM (2008)
18. Samsung SSD. In: available at http://www.samsung.com/global/business/semiconductor/products/flash/ssd/2008/product/pc.html, Samsung (2009)
19. Business Applications Performance Corporation: MobileMark 2007. In: http://www.bapco.com/products/mobilemark2007/, BAPCo (2007)
20. Futuremark, co.: PCMark05. In: available at http://www.futuremark.com/products/pcmark05/, Futuremark (2009)