Flash Memory Based Storage System
- SmartSaver: Turning Flash Drive into a Disk Energy Saver for Mobile Computers, ISLPED’06
- Energy-Aware Flash Memory Management in Virtual Memory System, islped’06
Flash memory has become popular storage in mobile devices

- low power
- light weight
- shock resistant
NOR allows XIP (Execution-in-Place)
NAND flash is commonly used for data storage due to its lower cost and higher density than NOR

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Read latency</td>
<td>110ns (6ns)</td>
<td>25μs (50ns)</td>
</tr>
<tr>
<td>(sequential latency)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write latency</td>
<td>110us</td>
<td>200us</td>
</tr>
<tr>
<td>Read/Write Unit</td>
<td>Word(2B)</td>
<td>2KB</td>
</tr>
<tr>
<td>Erase time</td>
<td>0.6s</td>
<td>2ms</td>
</tr>
<tr>
<td>Block size</td>
<td>16K/64K</td>
<td>128K</td>
</tr>
</tbody>
</table>

1. Samsung K8C1215EBM
2. Samsung K9F1G08R0A
Organization of a Typical NAND Flash Memory

- Read/Write one page
- 1 Page
- Block 0
- Block 1
- Block 2
- Block 3
- Erase one block

- Samsung K9F1208R0B
  - 1 Block = 32 pages
  - 1 Page = 512B
- Write once
  - Written page can not be overwritten
Flash Memory Characteristics

- Write once
  - Written page can not be overwritten
- Write once
  - Written page can not be overwritten
- Out-place update
Flash Memory Characteristics

- When # of free pages $\leq GCT$ (*Garbage Collection Threshold*)
  - Trigger *Garbage collection* to reclaim dead pages
    - Via erase operations
    - Basic unit of erase operations is a block
Flash Memory
Characteristics (cont’d)

- Garbage collection to reclaim dead pages
  - Live data copying
  - Block erase
- Garbage collection overheads
  - Live data copying
  - Block erase
Flash Life-Time

- Flash memory only guarantee a limited number of erases
- Wear leveling
  - Allows data writes to be evenly distributed over flash blocks
SSD uses **solid-state memory** to store persistent **data**. An SSD **emulates** a conventional **hard disk** drive, thus easily replacing it in any application.
Flash-based Solid State Drive Architecture

Logic Block Address

address translation table

<table>
<thead>
<tr>
<th>LBA</th>
<th>Physical address (bank, block, page)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0, 0, 3)</td>
</tr>
<tr>
<td>1</td>
<td>(0, 1, 2)</td>
</tr>
<tr>
<td>2</td>
<td>(1, 2, 1)</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Garbage Collection

Command translation

Flash Memory

FTL layer

MTD layer

Physical device
Hybrid Disk Drive
Writes are Problematic

- Writes consume more energy than reads
- Frequent writes result in dead pages on flash memory
  - Trigger frequent garbage collections

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read (page)</td>
<td>47.2 ns</td>
<td>679 nJ</td>
</tr>
<tr>
<td>Write (page)</td>
<td>533 us</td>
<td>7.66 mJ</td>
</tr>
<tr>
<td>Erase (block)</td>
<td>3 ms</td>
<td>43.21 mJ</td>
</tr>
</tbody>
</table>
Key Design Principles for Energy-Efficient Flash Memory

- Reduce writes to flash memory
- Efficient garbage collection

Diagram:

- Block X
- Block Y
- Recycle block X: 2 writes, gain 14 free pages
- Recycle block Y: 11 writes, gain 5 free pages
- An invalided page
- A live page
- Energy-Aware Flash Memory Management in Virtual Memory System, islped’06
Operating System for Supporting Multi-tasking & Virtual Memory System

Main Memory

Page Table

Disk Buffer

Hard Disk

Task

Operating System

Main Memory
Traditional virtual memory system assumes disk as the secondary storage.
Flash Memory has very different characteristics from disk:
- Write once
- Out-place update
- Garbage collection

1. The need to revisit virtual memory system design
2. Energy-efficiency is the main design concern
- A memory page contains \( n \) flash pages
  - At a page fault, \( n \) flash pages of the victim virtual page are written back to back to flash memory
- Two important observations
  - Unnecessary writes from replacing a virtual page
  - Intra-page locality
In conventional virtual memory system, a full victim page is written to the secondary storage.
In conventional virtual memory system, a full victim page is written to the secondary storage.
Dirty Ratio

- A victim page often contains a significant amount of unmodified data.

<table>
<thead>
<tr>
<th>Application</th>
<th>kword</th>
<th>mozilla</th>
<th>kspread</th>
<th>openoffice</th>
<th>gqview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dirty Ratio</td>
<td>89.73%</td>
<td>48.49%</td>
<td>88.61%</td>
<td>66.59%</td>
<td>98.86%</td>
</tr>
<tr>
<td>Application</td>
<td>kword +juk</td>
<td>mozilla +juk</td>
<td>kspread +juk</td>
<td>openoffice +juk</td>
<td>gqview +juk</td>
</tr>
<tr>
<td>Dirty Ratio</td>
<td>69.41%</td>
<td>40.90%</td>
<td>72.40%</td>
<td>59.31%</td>
<td>97.62%</td>
</tr>
</tbody>
</table>

Dirty ratio = \[
\frac{\text{the number of dirty 512B block in a dirty memory page}}{\text{the number of 512B blocks in a main memory page}}\]
What is Intra-page Locality?

- Flash pages in one main memory page are written to flash memory back to back.
Why is Preserving Intra-Page Locality Important?

- It affects the efficiency of garbage collection
Garbage Collection Threshold vs. Intra-Page Locality

\( GCt \): garbage collection threshold

\( m \): \# of flash pages in one memory page

\( n \): \# of flash pages in one flash block

- \( GCt \mod m = 0 \)
- \( GCt \mod n \geq n - m \)
- \( GCt \mod m \neq 0 \) and \( GCt \mod n < n - m \)

![Graph showing the relationship between Garbage Collection Threshold and Normalized Energy Consumption.](image)
Proposed Energy Efficient VM Design

- Reduce # of writes to flash memory
  - Subpaging
  - HotCache
- Efficient garbage collection
  - Duplication-aware garbage collection
Subpaging

- Divide a virtual memory page into a set of subpages in the granularity of flash page size
- Each subpage is associated with a dirty bit.

Memory Page

Flash Page Size

Dirty Bit

One Write to Flash

Flash Memory
- Caching writes only
- Preserving intra-page locality
- Capturing hot data
How to Capture Hot Data?

- Three management policies
  - Two-level LRU (2L)
  - Time frequency (TF)
    - Replace the HotCache block with smallest \( \text{timestamp} \times \text{write_counts} \)
  - Time frequency locality (TFL)
    - TF policy with intra-page locality preserved
- Exploit data redundancy between the main memory and flash memory to eliminate unnecessary live page copying during garbage collection
SWAP system

Read(LBA) Write(LBA, PID, VPN) Swap_clean(LBA) Swap_free(PID, VPN)

FTL

Garbage Collection

Address Translation Table (ATT)

<table>
<thead>
<tr>
<th>LBA</th>
<th>Physical address (bank, block, page)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>1</td>
<td>(0, 1, 2)</td>
</tr>
<tr>
<td>2</td>
<td>(1, 2, 5)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Block Allocation Map (BAM)

<table>
<thead>
<tr>
<th>Physical address (bank, block, page)</th>
<th>LBA</th>
<th>State</th>
<th>PID</th>
<th>VPN</th>
<th>In_memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0, 1)</td>
<td>0</td>
<td>Free</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(0, 0, 2)</td>
<td>9</td>
<td>Invalid</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>(0, 0, 3)</td>
<td>8</td>
<td>Valid</td>
<td>1</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Experimental Setup

- Trace-driven simulation
  - Valgrind: captures the memory access trace while application executing in real-time.

- Applications
  - kword, kspread, mozilla, openoffice, gqview
  - Multi-programming workloads: kword+juk, kspread+juk, mozilla+juk, openoffice+juk, gqview+juk

- Configuration
  - Main memory
    - 16MB, 4K page size.
  - Flash memory
    - 16K block, 512 B page
    - 128K block, 2KB pages.

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kword</td>
<td>word processor</td>
</tr>
<tr>
<td>kspread</td>
<td>spreadsheet application</td>
</tr>
<tr>
<td>mozilla</td>
<td>web browser</td>
</tr>
<tr>
<td>openoffice</td>
<td>popular office suite similar to Microsoft office</td>
</tr>
<tr>
<td>gqview</td>
<td>image viewer</td>
</tr>
<tr>
<td>juk</td>
<td>MP3 jukebox program</td>
</tr>
</tbody>
</table>
Smaller flash page size leads to more energy reduction
- 512B page size: 20% energy reduction on average
- 2KB page size: 8% energy reduction on average

Save more energy for multiprogramming workload
- Single-program workload: openoffice (14% energy reduction)
- Multi-program workload: openoffice+juk (31% energy reduction)
<table>
<thead>
<tr>
<th>Cache size</th>
<th>512KB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FIFO</td>
</tr>
<tr>
<td>Replacement Policy</td>
<td></td>
</tr>
<tr>
<td>Average hit rate</td>
<td>0.41%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache size</th>
<th>1MB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FIFO</td>
</tr>
<tr>
<td>Replacement Policy</td>
<td></td>
</tr>
<tr>
<td>Average hit rate</td>
<td>3.34%</td>
</tr>
</tbody>
</table>
TF causes higher overhead per GC due to breaking intra-page locality
- Up to 50% of energy reduction
- Average energy reduction rate is 24%
Energy reduction of HotCache + Subpaging + DA-GC
- Ranging from 9.3% to 75%

1MB HotCache & 512KB flash pages
Conclusions

- We revisit virtual memory system design with flash memory as the secondary storage

- Three energy-efficient VM design
  - Subpaging
  - HotCache management
  - Duplication-aware garbage collection

- Joint use of Subpaging & TFL policy & DA-GC
  - Reduce up to 75% of flash memory energy

- “An Energy-Efficient Virtual Memory System with Flash Memory as the Secondary Storage”, Tseng et. Al. islped’06
- “Energy-Aware Flash Memory Management in Virtual Memory System”, Le et. al, in revision for TLVSI.
- SmartSaver: Turning Flash Drive into a Disk Energy Saver for Mobile Computers, ISLPED’06
Motivation

- Disk is a big energy consumer in mobile systems
  - Can we reduce disk power?
    - If the disk is idle, we can change it to power saving mode
      - But the disk idle time may be less than the break even time
    - Could we extend the disk idle time?
      - Caching could reduce disk access
      - Prefetching can move disk access time forward
      - Write buffering can move disk access time backward

| $P_{active}$  | Active Power   | 2.00 W |
| $P_{idle}$    | Idle Power    | 1.60 W |
| $P_{standby}$ | Standby Power | 0.15 W |
| $E_{spinup}$  | Spin up Energy| 5.00 J |
| $E_{spindown}$| Spin down Energy| 2.94 J |
| $T_{spinup}$  | Spin up Time  | 1.60 sec |
| $T_{spindown}$| Spin down Time| 2.30 sec |

The energy consumption parameters for the Hitachi-DK23DA hard disk.
Use flash memory as a disk buffer
**Challenge**

- Flash drive does not fit well in the storage hierarchy
  - Flash may be bottleneck because of its write bandwidth is smaller than disk
- One-time access data should not be cached at all
  - We should not cache all accessed data
- What to cache is critical
  - Very frequently reused data
  - Data that move slowly from/to the disk
- What to replace is also critical
- Caching
- Prefetching
- Write buffer
- Balance the three areas
Disc DPM Policy

- Idle -> entering the Idle mode immediately
- Time-out policy for standby mode

<table>
<thead>
<tr>
<th>( P_{\text{active}} )</th>
<th>Active Power</th>
<th>2.00 W</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_{\text{idle}} )</td>
<td>Idle Power</td>
<td>1.60 W</td>
</tr>
<tr>
<td>( P_{\text{standby}} )</td>
<td>Standby Power</td>
<td>0.15 W</td>
</tr>
<tr>
<td>( E_{\text{spinup}} )</td>
<td>Spin up Energy</td>
<td>5.00 J</td>
</tr>
<tr>
<td>( E_{\text{spindown}})</td>
<td>Spin down Energy</td>
<td>2.94 J</td>
</tr>
<tr>
<td>( T_{\text{spinup}} )</td>
<td>Spin up Time</td>
<td>1.60 sec</td>
</tr>
<tr>
<td>( T_{\text{spindown}})</td>
<td>Spin down Time</td>
<td>2.30 sec</td>
</tr>
</tbody>
</table>
The energy that could be saved by avoiding a busy period may vary greatly
- gzip can compress 20MB data in few seconds
- glimpse may need a few minutes to process the same file

How to define the profit of caching data?
- Energy Saving Rate (ESR)
  - Energy saving per MB cached data

\[
\text{disk active energy} + \text{disk idle energy} + \text{the energy of spinning up/down disk} - \text{disk standby energy}
\]

amount of data accessed during the busy period
- **Envelope**
  - All accessed data during a busy period
  - Envelope is characterized with
    - H: $H(e) = ESR(e) + L$
    - L: H value of the victim envelope & is initialized to 0
- **Envelope stack**
  - Sorted in the increasing order of H from bottom to top
- **Replacement policy**
  - Envelop at the stack bottom is selected as the victim
Prefetching sequential file access
Rely on OS’s hints on sequential file accesses
- Buffer all data written to disk when disk is in standby mode
- Write back the data when
  - 90% of the writeback area is used
  - Disk is in active state
Balancing Areas

- How to divide flash to these three areas?
  - Least productive area give some space to most productive area
  - Productivity: energy saved/total flash blocks
Experimental Results

- Original access pattern
Experimental Results (cont.)

Baseline: caching everything

### Experimental Results (cont’d)

**Energy consumption and Read/Write volume in the flash drive**

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Mem</th>
<th>Flash</th>
<th>Energy (J)</th>
<th>Flash R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>64</td>
<td>N/A</td>
<td>7325.0</td>
<td>N/A</td>
</tr>
<tr>
<td>Linux</td>
<td>128</td>
<td>N/A</td>
<td>6317.3</td>
<td>N/A</td>
</tr>
<tr>
<td>Linux</td>
<td>256</td>
<td>N/A</td>
<td>5231.6</td>
<td>N/A</td>
</tr>
<tr>
<td>Baseline</td>
<td>64</td>
<td>128</td>
<td>6365.7</td>
<td>64/493</td>
</tr>
<tr>
<td>Baseline</td>
<td>64</td>
<td>256</td>
<td>5312.9</td>
<td>104/442</td>
</tr>
<tr>
<td>SmartSaver</td>
<td>64</td>
<td>128</td>
<td>4318.0</td>
<td>96/263</td>
</tr>
<tr>
<td>SmartSaver</td>
<td>64</td>
<td>256</td>
<td>4046.2</td>
<td>110/316</td>
</tr>
</tbody>
</table>