Design-space exploration of flash augmented architectures

Thanumalayan S[1], Vijay Chidambaram V[1], Ranjani Parthasarathi[2]
College of Engineering, Guindy, Anna University

Abstract

Flash technologies are rapidly improving in terms of bandwidth and latency. In this paper, we consider incorporating flash technologies at various levels in the existing architecture. For each level, we estimate the access time considering different levels of concurrency. We speculate on how highly concurrent programs might benefit from the incorporation of flash into the architecture. We aim to reduce operating system complexity by integrating some memory management functions with the proposed architecture. Finally, we also predict architectural changes that would become feasible if the trend in reducing latencies and improving bandwidth continues.

1 Introduction

The development of many-core architectures has shifted the focus in design of memory modules to high bandwidth and power efficient solutions [13] rather than low latency ones. In this context, Flash assumes a significant role, as it offers the needed performance at low cost. Researchers are examining closer integration of flash into existing architectures. This has been tried in the software level in the Microsoft Windows ReadyBoost technology [2] and in the usage of flash as a cache for the hard drive. [2,3]

In this paper, we analyse the effects of both replacing secondary storage with flash storage and placing the resulting flash storage at various levels in the architecture. Introduction of flash into the architecture can also lead to reduction in complexity of the operating system, and we speculate on these effects in this paper.

While examining the effect of introducing flash into the architecture, we take a long term view and do not restrict ourselves to the latencies and bandwidth available as of today. Rather, we note the trend taken by flash and speculate on what might be possible if flash evolves to have certain latencies and bandwidth.

This paper examines the effects of integrating flash at various levels in the architecture. We begin with an analysis of previous work (Section 2). We explain the choices available in integrating flash at various levels in existing architectures (Section 3). We estimate the access times in each architecture that would result from the choices (Section 4). We propose methods for further evaluation of some of the architectures (Section 5).

2 Related Work

The IRAM project [6] at Berkeley examined the effects of bringing DRAM onto the chip with the processor. Their architecture involved combining memory and processor into a single unit and using many of these on a single board. This approach need not be the most efficient one when dealing with flash. We analyse architectures other than the IRAM to find the optimal architecture.

The many advantages of having on chip flash have led to the question of whether main memory can be replaced by flash. This question has been studied in the Envy project [7]. Since then, flash technologies have improved in terms of speed and bandwidth. The maturing of NVRAM technology also offers a viable alternative to SRAM/DRAM based main memory.

The unique properties of Flash have been examined and it has been suggested [1,2,3] that flash could be used to act as a cache for the secondary memory. However, we seek to replace secondary memory itself with on-chip flash storage.

[1]Student, Department of Computer Science, CEG
[2]Professor, Department of Computer Science, CEG

Operating system modules have been developed to utilize flash memory to extend the address space of DRAM, as in the Microsoft Windows Vista ReadyBoost feature [2]. Hard drives have been used as main memory [8] to drive memory intensive algorithms. This approach uses many hard drives in parallel to achieve a memory bandwidth comparable to DRAM bandwidth, while allowing a large amount of memory to be used.

Appropriateness of flash in file systems, databases and flash specific algorithms have been studied in great detail [4,9,10,11]. These studies concentrate on methodologies to use flash memory in existing architectures rather than new architectures that could exploit the full potential of flash.

3 Design Choices

In our design, we seek to replace secondary storage with flash technologies. Different types of flash technologies (NAND, NOR, embedded NOR) are available, each having different parameters. We primarily consider NAND flash in our design as it provides high densities. Other flash technologies have been considered at places where speed is very important. These flash technologies can be integrated at a number of places in existing architectures. We examine these choices in Section 3.1. As a result of the integration, we can combine some architectural units into one, which we call the *Memory Management and Translation Unit* (MMT). We explain the MMT in Section 3.2. In section 3.3, we examine whether main memory can be replaced or augmented by the on-chip flash storage.

3.1 Location of flash

Flash can be located at three places in the architecture – on-core, on-chip and off-chip. Each of these options has its own pros and cons. We derive expression E1 for comparing these three choices. Consider a pipelined architecture where B is the transmission time for a byte of data, H is time required to translate the logical address to physical address and F is the latency of flash to read a block of data. The total time required to retrieve *i* instructions (TT_i) will consist of three parts: time to send the C bytes of commands necessary to retrieve the data, time taken to translate the logical address to physical address, time taken to retrieve the data in the flash storage and the time taken to transmit the D bytes of data. Neglecting effects of main memory and cache and adding all the above components together, we get E1.

$$TT_i = i*C*B + H + F + B*D \qquad \text{(E1)}$$

The three options can now be evaluated by supplying different values for B,H and D. Keeping the flash on core will decrease B. However, placing the flash storage on-core limits the sharing of data and instructions among the cores. Programs which require no sharing of data among cores will benefit from on-core flash architectures. Similar arguments can be made for keeping the flash on chip or off chip. Further evaluation is required to find whether the advantage of lesser transmission latency is outweighed by the disadvantage of reduced sharing.

3.2 Memory Management and Translation Unit

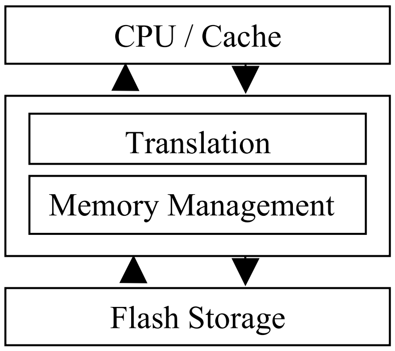

Figure 1 – Position of MMT in the architecture

One of the drawbacks of Flash storage is that a block must be erased before being written. Since the erase latency of flash is about 2 ms [14], efficient flash algorithms have been designed which allow blocks to be pre-erased and later to be used in place of the blocks to be rewritten. These algorithms necessitate translation between the block addresses supplied by the user and the physical block where the data is stored, and these translations are typically stored hierarchically in the flash drive. However, even with efficient algorithms, on average, for a 1 GB flash storage, four levels of translations are needed. These translations will incur writes and reads as well.

The *Memory Management and Translation Unit* (MMT) combines the conversion of logical addresses into physical address with memory management by using hashing internally for the conversion. The MMT can be implemented using SRAM/DRAM. The position of MMT is shown in Figure 1. The MMT can provide a very large logical address space, allowing us to provide segmentation. Upon request for lookup for a logical address, if the logical address has not yet been used, the MMT simply allocates a new block for that address, saves the translation and returns the physical block address. Thus memory management becomes much easier, and is handled almost fully in the hardware.

Consider a program with P instruction executions, with R percentage of instructions being reads and W percentage of instructions being writes. Let l be the no of hierarchical levels needed for translation. Let r be the read latency of flash. Let c be the lookups needed in the hash table. If no instructions are executed in parallel, the time required for reads is the total of time required for translation and time for actual reading of data.

Read time = $P*R*((l+1)r)$ [No MMT] (E2)
Read time = $P*R*(cH+r)$ [with MMT] (E3)

Let μ be the time required to writing a block of data – this consists of the time required to obtain address of a free block, write it and update the free list. The time required for writes would be the sum of times required for translating the address, time required for writing the data, and time required for updating the hierarchy.

Write time = $P*W*(lr+(l+1)*\mu)$[No MMT] (E4)
Write time = $P*W*(2*cH+w)$ [with MMT](E5)

If the cost of hashing becomes too high, the translation can be done directly, with a one-to-one mapping of each logical address to physical address. This reduces c to one, at the cost of losing the large logical address space of the MMT.

3.3 Main Memory

Main memory has been conventionally implemented using SRAM/DRAM. The read times of these technologies is very less compared to NAND flash [14]. Moreover, the read and write times of flash are not symmetrical – flash writes are about ten times slower than reads. Judging by these speeds, one would conclude that flash is not fast enough to replace main memory.

However, there are some advantages to having flash as both main and secondary memory. Typical program execution involves fetching the instruction into memory, and then executing it.

Total time = Time to shift instructions to main memory + Time to read instructions from main memory (E6)

However, a large part of the execution time of instructions is spent on a small subset of the instructions – the rest of the instructions are executed very few times. The first component of E6 will reduce if both main memory and secondary memory were combined with flash, since there would be no unnecessary shifting of instructions between main memory and hard drives. The second component of E6 is where flash slows down in comparison to SRAM/DRAM. However, with emerging trend of parallel execution, the second part of the equation becomes less perceivable to the user. Hence flash may be a viable alternative to main memory.

4 Estimation of access times

We have obtained rough estimates of the access times for the architectures explained in the previous section. The access latencies on flash vary from 80 ns [7] to 25 microseconds [14] for read, and upto 200 microseconds for write, depending on the type of flash used. Apart from the access latencies (during which period the flash module remains busy but other modules can function), the total delay experienced is constituted by:

- the transmission time required for moving control words and data
- the time required to latch the control words and

- the time required for block remapping algorithms.

In our estimation, we ignore the time required for block remapping techniques, as most algorithms such as periodic garbage collection, can be run in the background during periods of inactivity. We also ignore the time required to latch the control words.

We use the expressions mentioned in the previous sections to plot the variation of total access times with flash latency. Figure 2 is obtained when we consider programs that are not executed concurrently. We obtain Figure 3 by considering programs with concurrent execution.

In Figure 2, each line denotes the access times obtained for different ratios of reads and writes. We assume that there are 10000 instruction executions (P), the latency for translation (H) is 10 ns and 2 lookups are needed in the MMT for the translation. We assume that the write latency for flash is 10 times the read latency. For a program of about 1000 instructions, with 10000 instruction executions, read ratio 5% and write ratio 0.2%, the latency with a hard drive storage and main memory is comparable to our flash architecture with access times of about a microsecond for read. However, for large programs, it is comparable only if the flash latency is about 100 ns. This is the worst case scenario, and for programs which are not parallel or incur a lot of pipeline stalls, the development of faster flash technologies will play a major role.

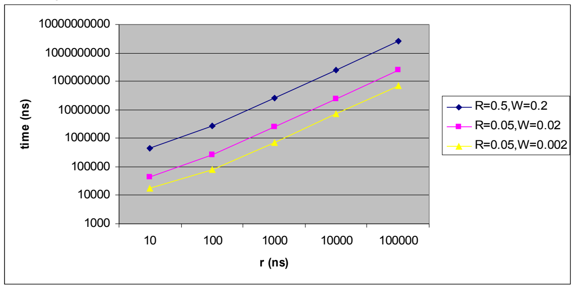

Figure 2 – Total Access times vs. Latency

However, for programs that are highly concurrent or can be executed without pipeline stalls, figure 3 shows that the flash latency does not play a major role in the delay experienced by the user, and for large programs, the delay is comparable to ones in the current architecture. Here, we assume that 5 bytes of command (C) and 4 bytes of data (D) are transmitted and translation latency (H) is 10 ns.

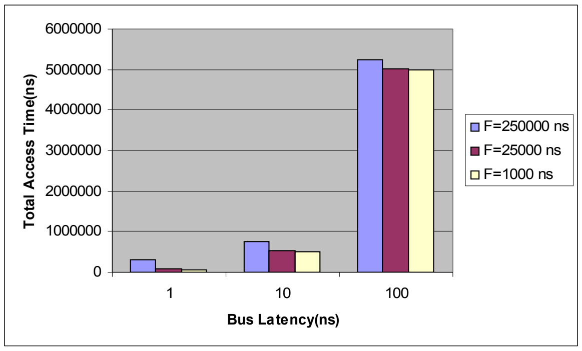

Figure 3 – Total Access times vs. Bus latency

The different bars denote the access time obtained when the flash latency is varied, keeping the bus latency constant. The transmission latencies incurred play a large role in the delay experienced by the user.

Programs having a large amount of concurrency, such as database programs and server programs [5,11] would benefit from having flash on-chip as a replacement for main memory.

5 Further Evaluation

The architectures proposed in this paper need to be further evaluated. This can be carried out using simulation tools such as OpenSparc. For multi-core architectures, the MMT unit can be added as an extra core, as shown in Figure 4. The caches will interface to main memory, and to the MMT. The MMT will also have DMA capabilities, and will be responsible for transfer of data between flash storage and main memory.

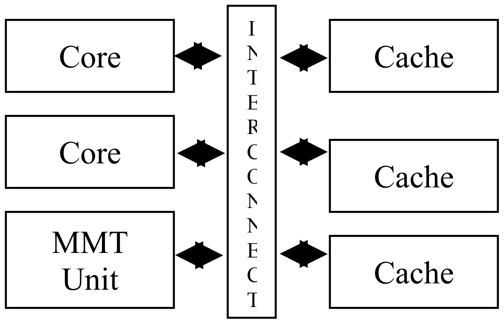

Figure 4 – Introduction of MMT in Multi-core architecture

The user can specify the instructions that are executed only once or twice. We propose that the instructions that are executed at most once or twice be read directly from flash. This will result in saving of time required to shift the instructions to main memory and to execute them. The rest of the program can be transferred to main memory and executed. With this proposed architecture, the size of the main memory can be made smaller. The flash storage will also be significantly more power efficient than hard drive secondary storage.

We intend to modify OpenSparc architecture to test our proposed architecture through simulation. We propose to modify the operating system to:

- Integrate all memory management modules into the MMT unit.
- Eliminate caching of secondary memory data on main memory and associated modules.
- Change the file system to take advantage of translations in the memory management unit and the inherent properties of flash, similar to [4]

We would then be able to quantify the total gains in performance from our architecture.

6 Conclusion and Future Work

In this paper, we have analyzed and estimated the various ways in which flash technology can be leveraged to obtain performance gains over existing architectures. We have also identified changes to operating systems that can be brought about due to the integration of flash. However, further work needs to be done to quantitatively evaluate these changes, and to evaluate how power consumption is affected. It can be concluded that, with the introduction of multi core architectures and applications that have a large amount of parallelism, the integration of flash into the architecture will lead to efficient architectures.

7 References

[1] Leventhal, A. 2008. Flash storage memory. Commun. ACM 51, 7 (Jul. 2008), 47-51.

[2] Matthews, J., Trika, S., Hensgen, D., Coulson, R., and Grimsrud, K. 2008. Intel® Turbo Memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems. Trans. Storage 4, 2 (May. 2008), 1-24.

[3] Min, S. L. and Nam, E. H. 2006. Current trends in flash memory technology: invited paper. In Proceedings of the 2006 Conference on Asia South Pacific Design Automation.

[4] Kyu-Ho Park, Seung-Ho Lim. 2006. An Efficient NAND Flash File System for Flash Memory Storage. IEEE Trans. Comput. 55, 7 (Jul. 2006), 906-912.

[5] Kgil, T. and Mudge, T. 2006. FlashCache: a NAND flash memory file cache for low power web servers. In Proceedings of the 2006 international Conference on Compilers, Architecture and Synthesis For Embedded Systems .

[6] Patterson, D., Anderson, T., Cardwell, N., Fromm, R., Keeton, K., Kozyrakis, C., Thomas, R., and Yelick, K. 1997. A Case for Intelligent RAM. IEEE Micro 17, 2 (Mar. 1997), 34-44.

[7] Wu, M. and Zwaenepoel, W. 1994. eNVy: a non-volatile, main memory storage system, ACM SIGPLAN Notices Volume 29 Issue 11, pg 86-97, November 1994

[8] Kunkle,D. and Cooperman,G. 2008. Solving Rubik's Cube:Disk Is the New RAM. Commun. ACM 51, 4 (April 2008).

[9] Gal, E. and Toledo, S. 2005. Algorithms and data structures for flash memories. ACM Comput. Surv. 37, 2 (Jun. 2005), 138-163.

[10] Benini, L., Macii, A., and Poncino, M. 2003. Energy-aware design of embedded memories: A survey of technologies, architectures, and optimization techniques. Trans. on Embedded Computing Sys. 2, 1 (Feb. 2003), 5-32.

[11] Lee, S., Moon, B., Park, C., Kim, J., and Kim, S. 2008. A case for flash memory ssd in enterprise database applications. In Proceedings of the 2008 ACM SIGMOD international Conference on Management of Data.

[12] Doh, I. H., Choi, J., Lee, D., and Noh, S. H. 2007. Exploiting non-volatile RAM to enhance flash file system performance. In Proceedings of the 7th ACM & IEEE international Conference on Embedded Software.

[13] Oskin, M. 2008. The revolution inside the box. Commun. ACM 51, 7 (Jul. 2008), 70-78.

[14] Inoue A., Wong D. 2003. Nand Flash Applications Design Guide, Revision 1.0, April 2003, Toshiba America Electronic Components, Inc.