國立交通大學

資訊科學與工程研究所

碩 士 論 文

適用於多通道固態硬碟的 prefetch 方法

Adaptive prefetch for multi-channel architecture SSD

研 究 生：黃士庭

指導教授：張立平　教授

中 華 民 國　九 十 八　年 九 月

適用於多通道固態硬碟的 prefetch 方法
Adaptive prefetch for multi-channel architecture SSD

研 究 生：黃士庭　　Student：Shih-Ting Huang

指導教授：張立平　　Advisor：Li-Ping Chang

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

September 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年九月

適用於多通道固態硬碟的 prefetch 方法

學生：黃士庭　　指導教授：張立平

國立交通大學資訊科學與工程研究所碩士班

摘　　要

SSD 是使用 NAND Flash Memory 來當作儲存元件，它具有抗震、讀取速度快等特性。SSD 為了加快它讀寫的速度，都會採用 multi-channel 的架構，而 SSD 的容量逐漸增加，內部的 flash chips 也越來越多，使得一個 request 的 size 沒有辦法讓所有的 flash chips 工作，造成有些 chips 處於 idle 狀態。Device 一次只能接受一個 request 讀取，而且 OS 對於 sequential read access 都會分為好幾個 requests 下給 device，所以我們提出一個 prefetch 的方法，用來預測 sequential read access，希望把這些 sequential read requests 連接起來，這麼一來可以加速 SSD 處理 sequential read access 的速度。然後我們還在不同的硬體架構下，觀察 prefetch 所造成的影響。最後實驗結果顯示，我們的 prefetch 方法在 8-channel 的 SSD 以及每個 channel 裡有 4 個 flash chips 的架構下，可以改善約 22.5% 的效能。

關鍵字: 固態硬碟(Solid-State Disk)，預先快取(prefetch)，多通道架構，

NAND 快閃記憶體(NAND Flash Memory)

Adaptive prefetch for multi-channel architecture SSD

student：Shih-Ting Huang　　Advisors：Dr. Li-Ping Chang

Department（Institute）of Computer Science
National Chiao Tung University

ABSTRACT

The storage component of SSD is NAND flash memory, it has shock resistance and high speed of read operation. SSD uses multi-channel architecture for increasing speed of read/write operation. Therefore the storage size of SSD and flash chips in it is increasing such that a request does not cause all chips to work, and some chips are idle. A device can service up to one request at a time, and OS separates the sequential read access to many requests to command the device. We proposed a prefetch policy for predicting sequential read access, and hoped to connect the sequential read requests. By this way it could speed up the progress of sequential read access on SSD. Then we also observe the effect of prefetch on different hardware architectures. Finally, our prefetch policy used on the SSD architecture with 8-channel and four flash chips in each channel can improve about 22.5%.

Keyword: SSD(Solid-State Disk)，prefetch，multi-channel architecture，

NAND flash memory，

誌謝

誠摯的感謝指導教授張立平老師，老師悉心的教導使我得以了解嵌入式系統的深奧，不時的討論並指點我正確的方向，使我在這些年中獲益匪淺。因為有你的體諒及幫忙，使得本論文能夠更完整而嚴謹。

感謝在這段期間，蘇宥全、郭郡杰、楊明毅、廖秀芬同學的幫忙，使得我能順利走過這兩年。實驗室的黃偉杰、郭晉廷、黃義勛學弟和黃莉君學妹們當然也不能忘記，你/妳們的幫忙及搞笑我銘記在心。

研究口試期間，感謝謝仁偉老師和陳雅淑老師不辭辛勞細心審閱，不僅給予我指導，並且提供寶貴的建議，使我的論文內容可以更臻完善，在此由衷的感謝。

感謝系上諸位老師在各學科領域的熱心指導，讓我增進各項知識範疇， 在此一併致上最高謝意。

最後，謹以此文獻給我摯愛的雙親。

目錄

表目錄

圖目錄

一、 Introduction

NAND Flash Memory 具有耐震、省電以及體積小的硬體優勢，因此被廣泛的使用在各種嵌入式系統中，如：隨身碟、記憶卡、多媒體播放器(PMP)等，而現今技術的進步，NAND flash memory 的容量逐漸變大，出現了一個新的應用固態硬碟 Solid-State Disk(SSD)，用來取代行動電腦中的傳統硬式硬碟 Hard Drive Disk(HDD)。

NAND Flash Memory 可分爲 SLC(Single-Level Cell) NAND Flash[4]跟 MLC(Multi-Level Cell) NAND Flash[5]，MLC 利用將 Memory Cell 電位分爲多階段，目的是讓存儲密度可以達到加倍的效果，跟 SLC 相比，MLC 的優勢在於容量高且成本低。不管是 SLC 還是 MLC，他們的最高讀取速度在 40MB/s 左右，而傳統硬碟平均讀取速度約爲 40MB/s 至 100MB/s，但是 SSD 可以在硬體的架構上做調整，如 multi-channel 跟 interleaving，不像傳統硬碟是機械式的方式運作，這麼一來可以使得 SSD 的讀取速度倍增至 100MB/s ~ 200MB/s 而逐漸取代傳統硬碟。

爲了增加 SSD 的 performance，現今所有 SSD 的硬體架構都是採用 multi-channel 的架構，因爲每個 channel 都有一條個別的 data bus 來傳輸資料，即使 request 所需要的資料只在 channel 1 跟 channel 2，每次還是會讀所有的 channel，因爲它們的 data transfer time 是平行的，所以不會有 overhead，跟 single channel 相比，有多少個 channel，multi-channel 的傳輸速度就增加幾倍。

隨著 SSD 的容量越來越大，flash chips 越來越多，因爲 data bus 的 bandwidth 有限，所以都會採用 shared bus 的方式來組織 chips，每個 channel 中的 flash chips 共用一條 data bus，SSD 除了採用 multi-channel 的架構外，還會使用 interleaving 來減少 response，interleaving 的方式主要是將 flash chip busy 的時間重疊，以及在共用一條 data bus 下，利用其他 flash chip 在 busy 的時候來傳輸資料，達到時間平行的效果，在同一個 channel 中，因爲 data bus 是共用的，所以 chips 必須輪流使用 data bus，沒辦法像 multi-channel 的架構那樣，每次都讀全部的 chip，當讀到不需要的資料時，會有 data transfer 的 overhead。

SSD 中 channel 的數量跟 channel 中 flash memory chip 的數量逐漸增加，而且 device 每次只能接受一個 request，在處理 request 的時候常常沒有辦法使所有的 flash chip 都運作，使得有些 flash chip 處在 idle 狀態，沒有完全利用 SSD 當中的 flash chips，如 figure 1 所示，每次只接受一個 request，而每個 request 都只用到所有 channel 的一個 chip，使得在處理一個 request 的時候，另外三個 chip 處在 idle 狀態，最理想的情況是同時所有的 chips 都在 busy，這樣才能夠加快 request 的處理。

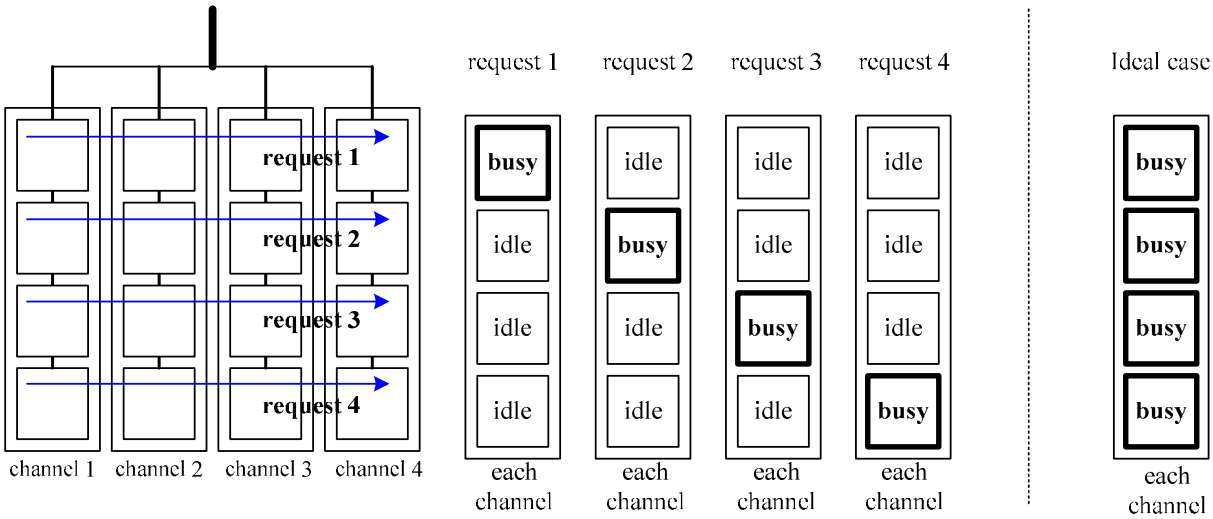


Figure 1. The problem for the current SSD

爲了能夠有效利用 SSD 提供的 interleaving，我們提出一個應用在 SSD 上的 prefetch 機制，主要是能夠預測出 sequential read access 然後執行 prefetch(即 read ahead)，使得 SSD 中的 flash chip 都處在 busy 狀態，如果 prefetch 大量的資料都命中的話，就能夠大幅改善 response，但對於 prefetch 會有額外的 overhead，就是當 prefetch 不需要的資料太多的時候，仍然要浪費時間傳送這些沒用的資料，若因爲擔心 prefetch 造成的 overhead 太大，而每次僅 prefetch 小量資料，這反而限制了 prefetch 的效能，這兩者之間造成了 trade-off，所以我們觀察在不同的 SSD 架構下，不同的 prefetch policy 對各種 SSD 架構的影響。

假設原本 request 只會讀 chip 1 跟 chip 2，經過了 prefetch 之後，我們就從 chip 1 至 chip 4 讀取資料，當 prefetch 預測成功，就能夠賺到 figure 2(a)中那段平行的時間，如果預測失敗，我們就必須承受 figure 2(b)所指的 overhead。我們提的 prefetch policy 會在最小的 overhead 下達到不錯的 performance。

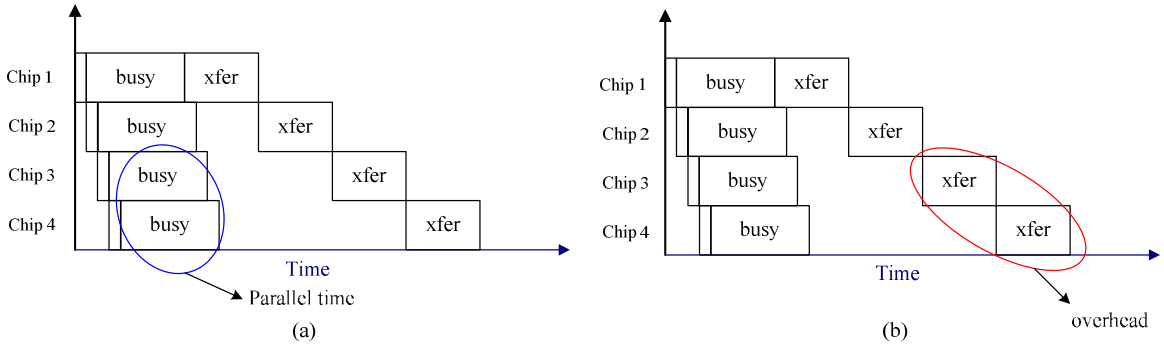


Figure 2. The improvement and overhead of prefetch

本論文的結構如下，第二章爲說明 Flash memory 的基本 operation 以及

multi-channel 跟 interleaving 如何加速 SSD 的 throughput，第三章會說明在此論文中主要的 SSD 硬體架構以及 prefetch 的 implement 方法，第四張將會使用模擬的方式來實驗，實驗部份會在不同的 prefetch 機制以及不同的硬體架構觀察 prefetch 如何影響 SSD 的 performance。

二、 Background

2.1 Flash memory characteristics

快閃記憶體分爲 NAND flash 跟 NOR flash，NOR flash 速度較快，具有 eXeute in place，但是成本較高，而 NAND flash 成本較低，SSD 裡都是以 NAND flash 爲主，所以本篇論文以 NAND flash 爲研究基礎。如 figure 3 所示，NAND flash memory 裡會分爲多個 block，而每個 block 裡又再區分成多個 page。NAND flash 主要有三個動作：read operation、write operation 和 erase operation，其中 read operation 跟 write operation 都是以一個 page 爲最小讀寫單位，而 erase operation 則是以一個 block 爲最小抹除單位，當一個 page 已被寫入資料，必須經過 erase 才能夠再次寫入。

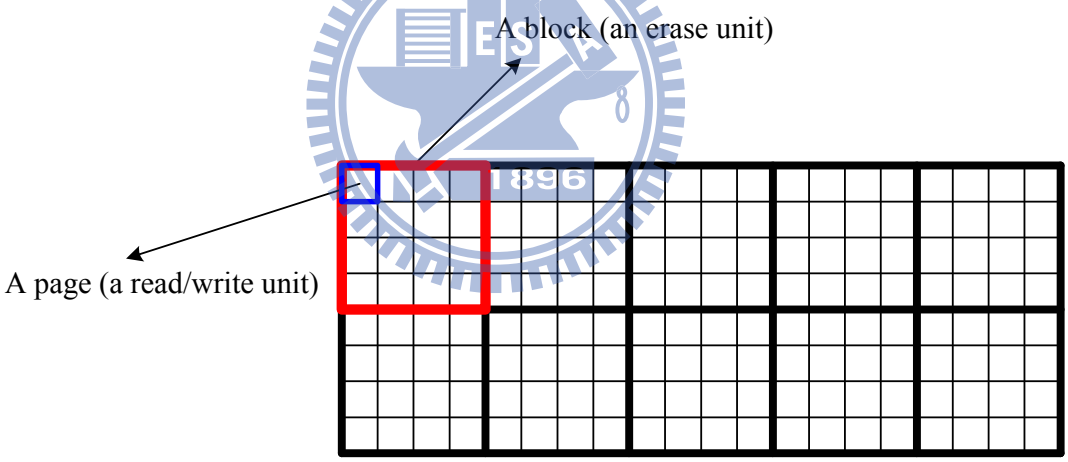


Figure 3. The structure of flash memory

Read operation 可分爲三個階段：setup、busy 跟 data transfer，setup 階段是透過 I/O pin 對 flash memory chip 下 command，給它 requested data 的實體位址，然後 flash chip 就會進入 busy 階段，flash chip 把需要的 data 放到 flash 內部的 register，之後進入 data transfer 階段，DMA 會從 register 把資料讀出來。而 write operation 也是分爲三個階段，跟 read operation 一樣是 setup、busy 和 data transfer，不同的地方是它們執行的順序，write operation 開始時一樣會先對 flash chip 下 command，再來會先進入 data transfer 階段，把要寫入的資料先寫到 flash 內部的 register，之後才是 busy 階段，將 register 的資料真正寫到 flash cell 裡，就完成一次 writer operation。Erase operation 則只分爲兩個階段：setup 和 busy，但是 erase operation 所消耗的時間卻是

遠大於另外兩個 operations。

2.2 Multi-channel and interleaving

現今市面上的 SSD 幾乎都是 multi-channel 的架構，從 2 channels，4 channels，甚至到 10 channels 都有，每個 channel 都有自己的一條 data bus，使用 multi-channel 的好處是可以同時對多個 flash chip 下 command，讓多個 flash chip 一起執行並將資料一起傳到 host 端，這樣跟 single channel 比起來，channel 的數量就相當於能改善多少倍數的 performance，如 figure 4 所示，以 4 channels 爲例，假設我們要執行 read operation，controller 會一起對所有 channel 裡的 flash chip 下 command，flash 經過 busy phase 後，在一起將 data 傳回 host，從 figure 4(b)來看，所有 channel 都是同時執行的，single channel 執行一次 read operation 的時間，在 4 channels 架構下，可以達到 4 倍的 performance。

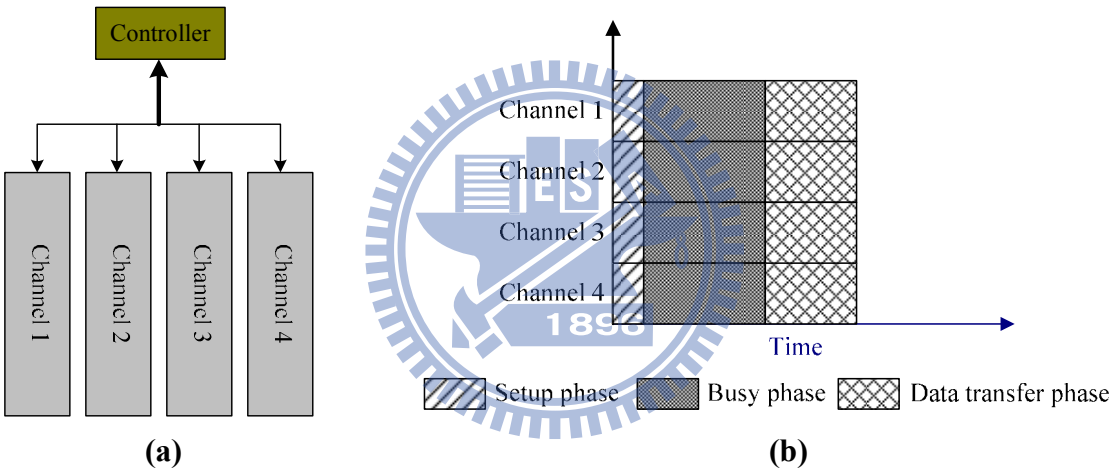


Figure 4. Multi-channel and read operation diagram

當執行 read operation，在 setup 階段 controller 透過 bus 下指令給 flash chip，而在 data transfer 階段 controller 也透過 bus 從 flash chip 接收資料，只有在 busy 階段 bus 才會是空閒的，**interleaving** 主要的想法就是當一個 flash chip 在 busy 時，利用這 bus 空閒的時間再對另一顆 flash chip 下 command 或接收資料。以 Figure 5(a)爲例，有 4 個獨立的 flash chip 共享一條 bus (single channel)，這 4 個 flash chip 分別稱爲 CE1 到 CE4，會有 1 個 controller 負責控制對哪一個 CE 下 command 及接收資料，假設我們要對 CE1 到 CE4 執行 read operation，其 timing model diagram 如 figure 5(b)，controller 對 CE1 下完 command 後，趁 CE1 進入 busy 階段時 controller 再對 CE2 下 command，以此類推。Interleaving 所改善的 performance 在於其 busy phase 重疊的時間，所以當 busy phase 的時間遠大於 setup phase 的時間加 data transfer 的時間的話，interleaving 就能改善較多的 performance。Multi-channel 跟 interleaving 這兩種架構並不衝突，可以個別使用，也可以組合一起使用，而我們的架構會在 section 3 說明。

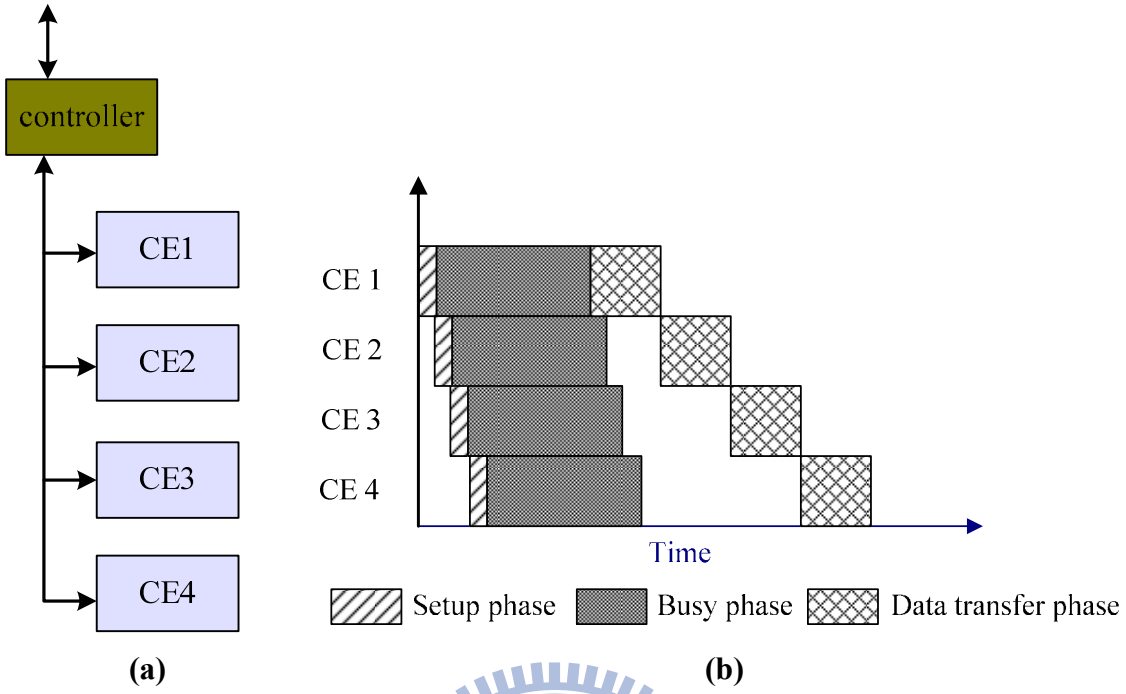


Figure 5. Single channel and read operation diagram

2.3 Motivation

一般 read access 分爲 random read 跟 sequential read，random read 的部份並不是此文所要討論的議題，本篇論文著重在 sequential read，OS 對 disk 下 sequential read access 時，如果所要求的 size 太大，一般會被分成好幾個 request，而且 device 一次只能接受一個讀取，device 在處理一個 request 時，request 通常都不會對齊 channel 的邊界，如 figure 6，而且 request 所要求的資料也不會覆蓋到所有 channel，因爲 multi-channel 的關係，SSD 會讓所有的 channel 一起運作，但這麼一來也是浪費的。

假設 Figure 6(a)(b)(c)是 OS 連續對 device 下的三個 requests，它們是 sequential read requests，圖中線段所指的是這些 requests 在 flash chips 上所讀取資料的分布，這三個 requests 都有相同的地方是都沒有對齊 channel 的邊界以及所讀取的資料都不會覆蓋所有的 channels，造成了 channels 當中許多的 idle slots，而這些 idle slots 所會讀取的資料部分，處理的方式可以分爲保留或者捨棄。如果選擇捨棄的方式，則 figure 6 的這個例子的 timing diagram 就如 figure 7(a)所表示，當 device 在處理 request2 跟 request3 時，會發現跟前一個 request 造成重複讀取的情形。若我們選擇把 idle slots 所讀取的資料存到 buffer 裡來避免重複讀取的話，而 figure 7(b)則是此方法的 timing diagram，跟 figure 7(a)相比，已經不會有重複讀取的 overhead 了，但這樣還是無法改善太多效能，而因爲 device 一次只能處理一個 request，這個例子中每次 device 在處理 request 時只會使得一個 CE 在 busy，這麼一來就沒有辦法利用 interleaving

來減少 response，而我們比較希望的情形是像 figure 7(c)這樣，利用 prefetch 將 sequential read requests 連接起來一次處理，這樣才能夠利用 interleaving 來減少 SSD 的 response。Figure 7(c)這種情形跟 figure 7(a)和 figure 7(b)比起來分別可以改善約 58%跟 52%的 response。

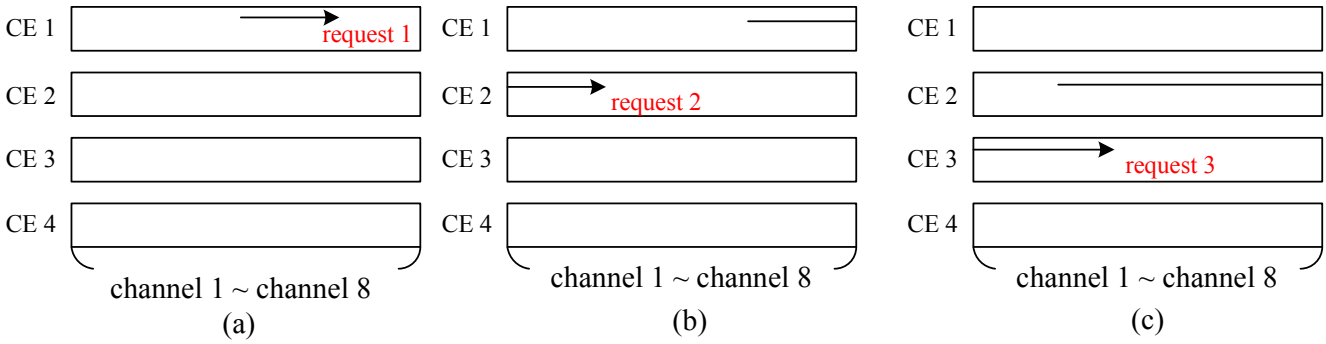


Figure 6. The example for unaligned requests

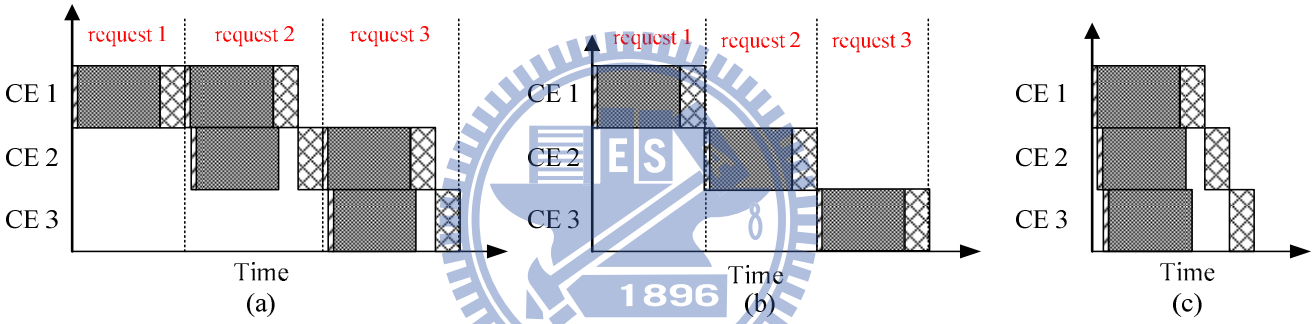


Figure 7. The timing diagrams of three approaches for unaligned requests

而我們的問題在於要如何來執行 prefetch，prefetch 需要注意的因素有二，一是 prefetch 的時機，另外一個就是 prefetch 的 size，當一個 request 進來時，就可以考慮需不需要做 prefetch，可是如果隨便就做 prefetch，把原本不需要的資料也事先讀進來，增加了這次的 response，每次都 prefetch 的話，會嚴重影響到 SSD 的 performance，所以我們期待能夠在第一時間就做 prefetch 而且 prefetch 的 size 也適當，沒有讀取到不需要的資料，而這在現實上是辦不到的，因爲每次只接受一個 request，無法得知它是 random read 還是 sequential read，即使是 sequential read，仍然也不曉得該 prefetch 多少 sectors，而我們提出的 prefetch policy 主要是能提供不錯的 performance 而且將 prefetch 所造成的 overhead 減至最低。

2.4 Related work

由於 flash memory chip 在沒有其他硬體架構的幫助下，它的最高讀取速度也只

有 40 MB/s，目前也有許多論文提出一些加快 flash memory performance 的硬體架構，像是在[1]這篇論文中，它提出了一個名叫 DUMBO 的 multi-channel hardware architecture，並利用此 architecture 提出了三種讓 flash memory chips 平行處理的方法，striping、interleaving 以及 pipelining，前兩種方法都是利用兩個以上可以獨立運作的 channel，當一個 channel 在 busy 的時候，controller 趁機對另一個 channel 下 command，好讓所有 channels 都在 busy，而 pipelining 則是利用 channel 當中的 ping-pong buffer 來達到平行化。

[3]這篇論文裡提到用 gang 的方式可以提高 SSD 的 bandwidth，並舉出兩種 gang 來達到平行化，shared bus gang 跟 shared control gang，shared bus gang 相當於 section 2.2 中所提的 interleaving 的架構，讓所有可獨立運作的 flash chips 共用一條 data bus，但是它的 bandwidth 還是沒變，只有 interleaving 的效果，可以減少 SSD 的 response，而 shared control gang 則相當於 section 2.2 所提的 multi-channel 的架構，由同一條 control pin 控制許多 flash chips，但是這些 chips 都擁有自己的 data bus 傳輸資料及 command，這麼一來可以達到 bandwidth 加倍的效果。

而在[2]這篇論文中，它也是利用多個可獨立運作的 flash chips，在文中稱作 multi-banks，來將各個 flash chip 的 busy time 重疊，不同的是，它還考慮到 bank 的平衡性，特別提出了 dynamic striping scheme，一般對於 multi-banks 的 address translation 都屬於平均分配的，如 RAID-0，但是會導致 bank 之間不平衡的原因在於 write request 的 locality 以及各個 bank 的使用不平均，而 dynamic striping policy 的方法是當要執行 write page 的時候，會先選擇 idle 而且還有空間的 bank 配置空間來寫入資料，利用這種動態配置空間的方式來讓各個 bank 都能平均被利用。

[1,3]都是利用 multi-channel 來增加 bandwidth 以及 interleaving 來減少 response，而[2]除了利用 interleaving 外，還特別對 multi-banks 做 load balance，對於現今的 SSD 爲了加快 throughput，均採用 multi-channel 的架構跟 interleaving，而且 channel 的數量和可獨立運作的 flash chip 數量都逐漸增加，導致 device 在處理 request 時無法使所有的 flash chip 都 busy，浪費了 SSD 原本可提供的平行度。

在[6]這篇論文中，它分別去觀察不同的 interleaving degree 跟不同 channel 數的 multi-channel 架構對 SSD performance 的影響，它還說增加 channel 的數量跟增加 interleaving degree 數相比，能改善較多的 performance，但是增加 channel 數量會增加設計 controller 的複雜度，而增加 interleaving degree 時就沒有這個困擾，這兩者之間存在著 trade-off。另外它也發現到當 interleaving degree 增加到一定數量後，對於 read requests 所能改善的效能就會趨於平緩，他認爲主要的原因在於太多個 banks 共用一條 data bus，即使會增加 bus 的使用率，相對地也會使得 bus 出現塞車的狀況，尤其是對於頻寬較小的 data bus 更是嚴重，而之後我們的實驗也會去觀察 interleaving

degree 對於 prefetch 的影響。

過去的論文並沒有對 flash memory 的 prefetch 做討論，flash memory 是電子式的儲存裝置，不像一般 hard disk 是屬於機械式的，hard disk 在讀資料時是靠磁碟的旋轉跟讀取頭，所以 hard disk 利用 prefetch (read ahead)主要是爲了減少 seek time，在本篇論文中，我們對 SSD 使用 prefetch 主要是爲了增加 flash chip 的平行運算。

三、 Adaptive Prefetch for Multi-channel SSD

3.1 Overall hardware architecture

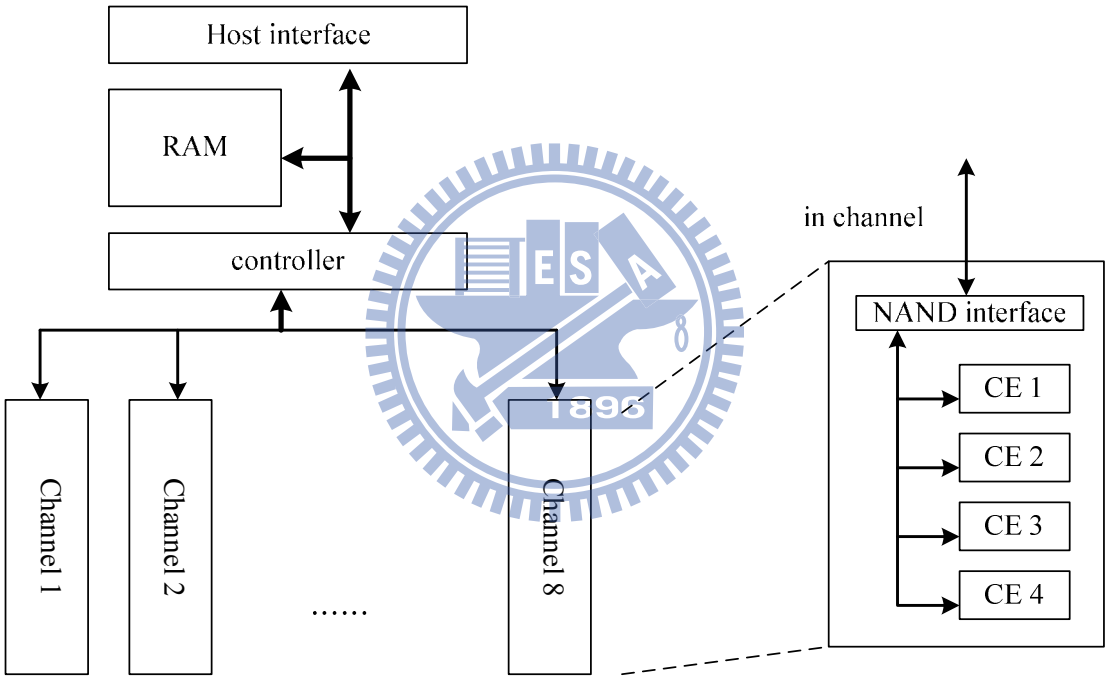


Figure 8. Overall architecture

因爲 SSD 的容量越來越大，flash memory chips 越來越多，礙於 data bus 的寬度有限，channel 的數量不行無限的增加，必須使用 shared bus 的方式來組織 chips，並且利用 interleaving 來改善 SSD 的 response。Figure 8 則是我們所使用的 SSD hardware architecture，host 透過 Serial ATA 跟 SSD 連接，並利用此傳送 block device commands(read/write request)，而 SSD 則透過 host interface 來接收這些 requests。我們實驗的架構使用 8 channel 的 architecture，每個 channel 中有 4 個 flash chips(CE) shared 一條 data bus，即它們的 interleaving degree 爲 4，channel 裡還有一個 NAND interface 負責控制對 flash memory chip 的 read/write operation，而 channel 外有一個 controller 及 RAM。我們在 section 4 部分會對不同的 architecture 做實驗，包含 channel

的數量、channel 中 flash chip 的個數以及 RAM 的 size，觀察在不同的 architecture 如何影響我們的 prefetch policy。

Host interface 接收到 read/write request 後，會透過 controller 中的 address translation 將 logical address 轉換成 physical address，然後到對應的 physical address 做存取，figure 9 則是我們的 mapping scheme，因爲是 multi-channel 的架構，在處理 request 時都會動到所有的 channels，在 8 channels 下最小的 read/write 單位爲 8 pages，在本篇論文裡我們稱作 1 個 **logical unit size**，controller 除了轉換爲 physical address 外還必須對齊 logical unit size，所以我們把不同 channel 中所有 CE1 相同的實體位址當作是連續的 logical address，CE2 至 CE4 亦相同，在 figure 9 中，每個 CE 中的數字代表的是 page number，page number 0 到 7 就是一個 logical unit，並且分別是 channel 1 到 channel 8 的第一個 CE 的第一個 page，而 page number 8 到 page number 15 我們放在第二個 CE 的第一個 page 的地方，以此類推，這樣 striping 的方式主要目的是當如果要讀 page number 0 到 31 時，在最小的 request size 下就能讓所有 CE 忙碌。

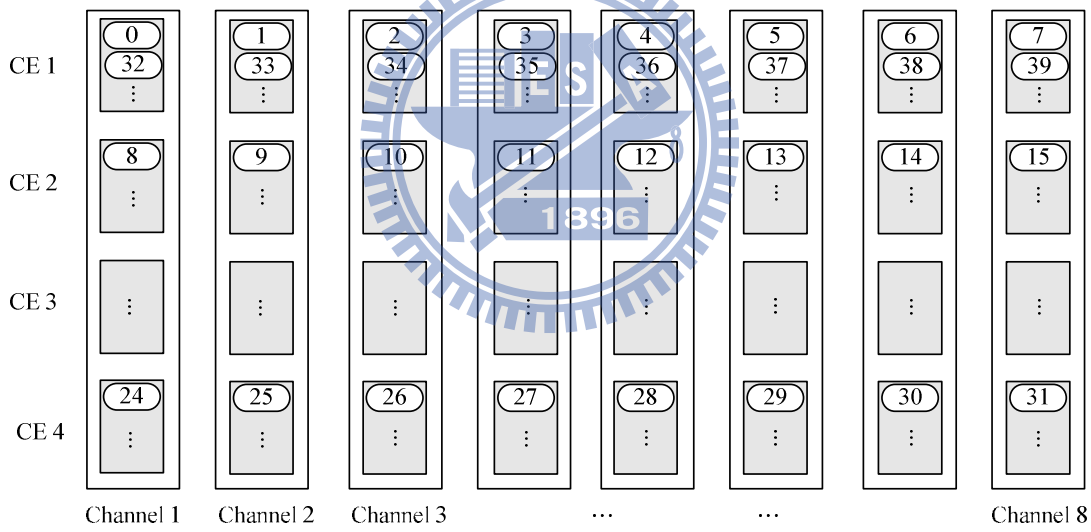


Figure 9. Page mapping scheme

因爲是 multi-channel 的關係，每次最小存取單位爲 8 pages，所以每次 request 都會被對齊至 8 pages 的倍數，而我們的 prefetch policy 決定在 channel 中 chips，從 figure 10(a)來看，假設有四個 requests，它們是 sequential read access，因爲 device 一次只能接受一個讀取，所以每次都只有一個 chip 在 busy，而我們的 prefetch policy 主要是將這種情形的 requests 串聯起來，如 figure 10(b)，device 只接收一個 request 就使得所有的 chips 都 busy，好達到 interleaving 的效果，我們可以從 figure 11 看出 prefetch 的效果，figure 11(a)因爲 device 每次只接受一個 request，所以每次都必須等上一個 request 執行完才換下一個 request，而 figure 11(b)則是把 figure 10(a)中的

四個 request 給串起來，將他們的 busy time 重疊，這麼一來可以大幅減少 response。

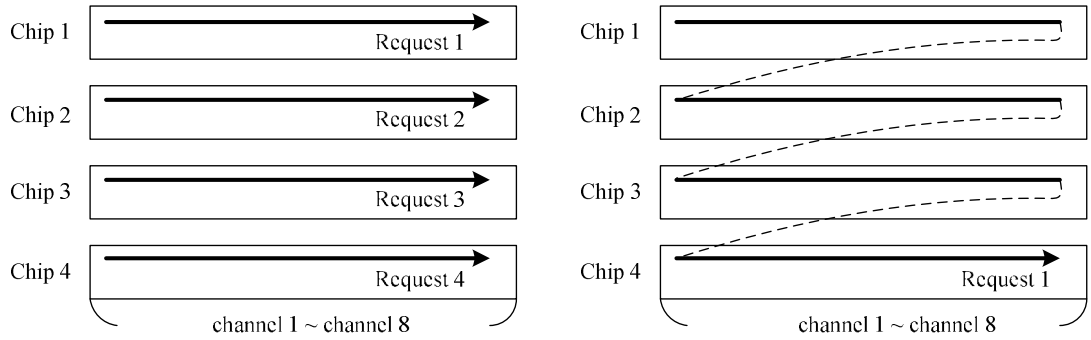


Figure 10. The effect of prefetching

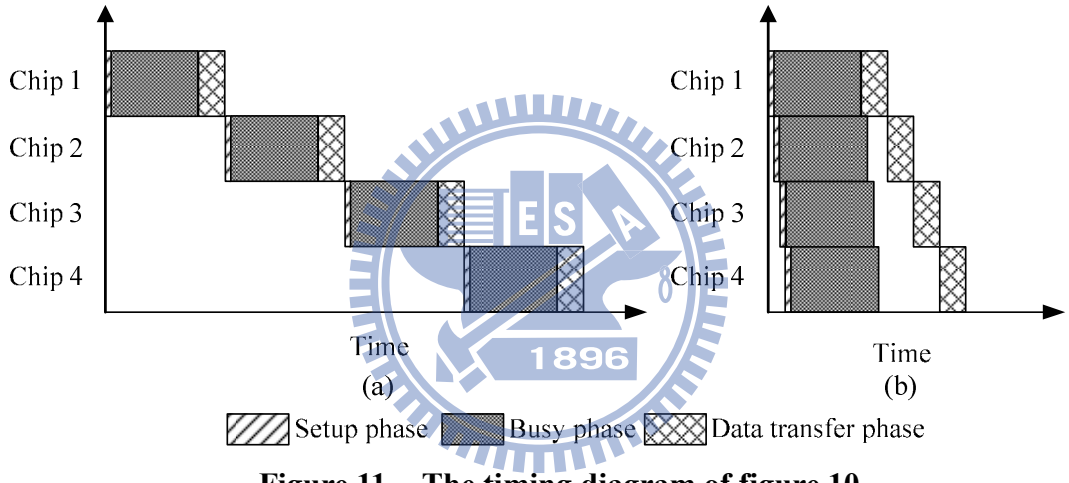


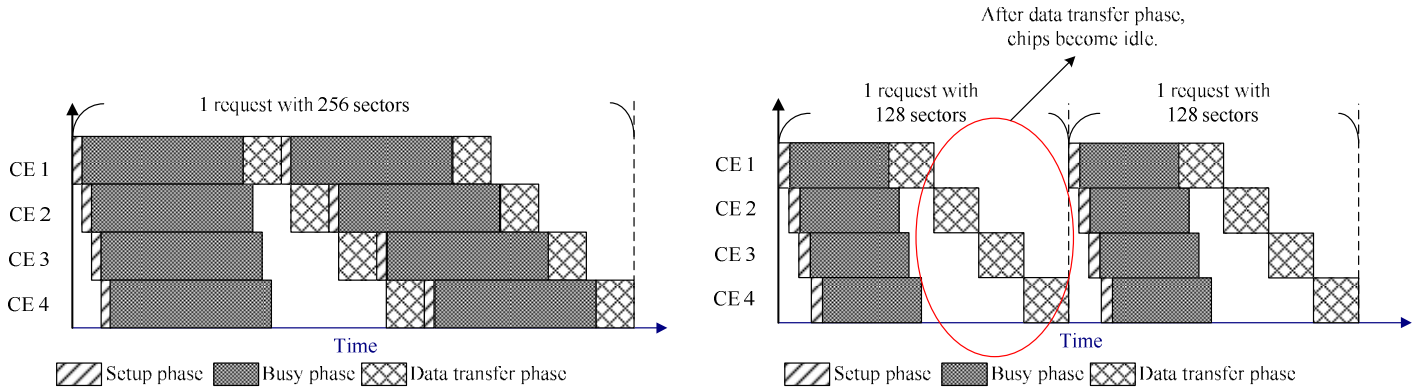
Figure 11. The timing diagram of figure 10

而接下來的章節要介紹 prefetch policy、用來記錄 sequential read access 的 thread table 以及存放 prefetch 資料的 prefetch buffer，當中又包含了 thread table 的管理和 buffer replacement 的機制。

3.2 Prefetch policy

因爲 shared bus 的關係，當我們 prefetch 到不需要的資料時，那些 flash chips 仍然要排隊將資料傳回去，而傳送這些沒用到的資料的時間，就是 prefetch 所必須承受的 overhead，但是如果 prefetch 的資料都有被使用到，就能夠大大的減少 response，而且 prefetch size 越大，所改善的 response 越多，如 figure 12 所示，figure 12(a)prefetch 的 size 是 figure 12(b)的兩倍，在 figure 12(b)中圈起來那段時間，因爲是 shared bus，所以 chips 必須排隊傳資料回去，先傳回去的 chip 就會變成 idle，反觀 figure 12(a)，當第一個 chip 傳完資料後，controller 會再對它下 command，讓它

繼續 busy，填補了這段 idle 的時間，這兩者相較之下，figure 12(a)可多減少約 21% 的 response time，而當一個 sequential read access size 越大，一次讀完會比分段讀取越有效率。所以我們需要一個 smart policy 來判斷是否要做 prefetch，來降低 prefetch 的 overhead 並且能改善不錯的效能。



(a) (b)

Figure 12. The effect of the different prefetch sizes

在做 prefetch 的時候，主要考慮的是 (一) 做 prefetch 的時機，以及 (二) 要 prefetch 多少 sectors 才算適當。而 figure 13 是描述在不同時機及不同 prefetch size 的情形，當太早做 prefetch，如 figure 13(a)，容易誤判爲它是 sequential read requests，而 prefetch 到沒有用的資料，造成額外的 overhead，而這種現象都發生在小而 sequential 的 request，剛開始就只 sequential read 一些 requests，讓我們認爲他是 sequential read access 而執行了 prefetch，而之後此 sequential read access 就剛好只讀開始那一段 address 而已，使得 prefetch 的資料都變成了 overhead。但如果太晚做 prefetch 的話，如 figure 13(b)，在 sequential read 的最後才做 prefetch，這種情形造成的 overhead 比較小，但是判斷 prefetch 太過保守，這樣 prefetch 能得到改善的效能反而會受到限制。當已經決定要做 prefetch 之後，接著要考慮的是要 prefetch 多少 sectors (read ahead size)，如果 prefetch 的 size 太小，如 figure 13(c)，雖然可以減少 prefetch 到錯誤資料的機會，卻也限制了 prefetch 的效能，但是當 prefetch size 太大，如 figure 13(d)，如果 prefetch 的資料能準確命中，是可以大幅改善 response，但是當 prefetch 到沒用的資料，就必須承受額外的 overhead，而最理想的情況是 figure13 (e)，在 sequential read access 開始的時候就做 prefetch，而每次 prefetch 的量都剛好是此 sequentilal read access 結束的位址，但這種情形在現實上是不可能達成的，因爲我們沒辦法知道未來的事情，我們只能傾向將 overhead 降到最低並且還能有不錯的 performance。

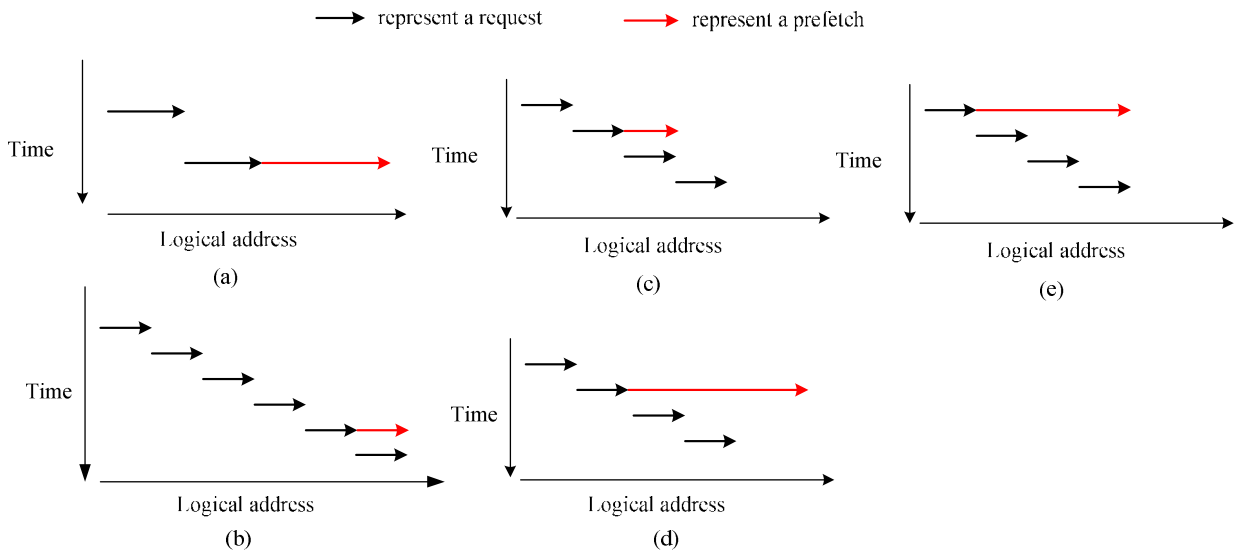


Figure 13. The effect of different prefetch timing and prefetch size

Prefetch 主要是加快 sequential read access，但要如何判斷它是 sequential read 並且開始做 prefetch，figure 14 是 section 4 實驗部分所使用的 trace 所統計出來的 sequential read access，X 座標爲已經讀了多少 sectors，Y 座標則是 sequential read access 的個數，圖中紅色圈起來的地方大部分都是 random read 跟小而連續的 request，而我們發現當 read request 連續讀了一定量的 sectors 後，就會有很大的機會接著讀下面連續的 sectors，即圖中藍色圈起來的地方，所以我們用目前已 sequential read 多少 sectors 來判斷 prefetch 的時機。

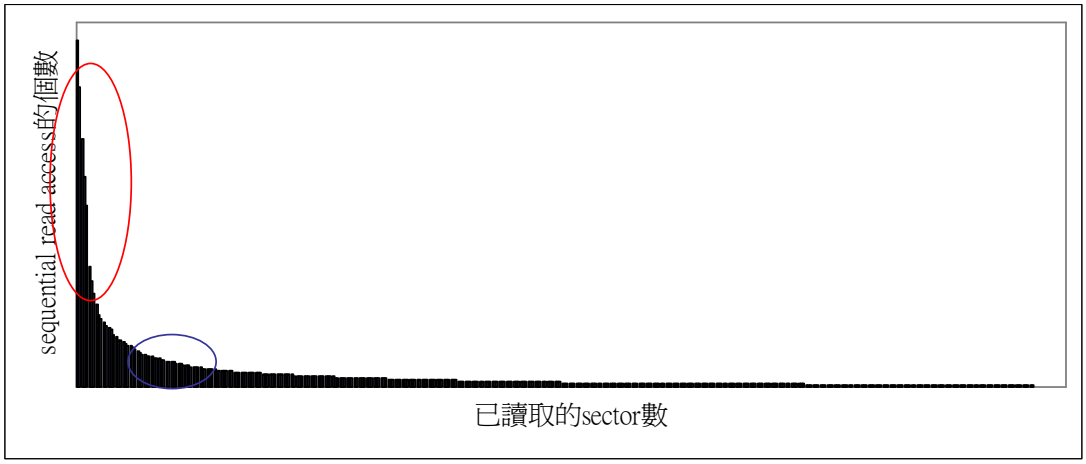


Figure 14. The statistics for the sequential read access

prfetch 的時機跟 prefetch 的 size 是息息相關的，如果要早點就做 prefetch 的話，prefetch size 若設的太大，對於那些小而 sequential 的 request 很多的時候，會造成非

常多的 overhead，但若是 prefetch size 定的太小，overhead 就也相對較小，但遇到大的 sequential read access 時，也就限制了 prefetch 的效能，而 prefetch 能改善的效能主要就來自大的 sequential read access，所以我們決定放棄在小而 sequential read 所能改善的 performance，把 prefetch 的重點放在大的 sequential read access。

而我們 prefetch 的時機跟 prefetch size 將在 section 4.2 做實驗說明，以及在不同的 prefetch 時機跟 prefetch size 對不同的硬體架構有何影響。

3.3 Thread identification

因爲 OS 都有支援 multitasking，同一時間會有多個應用程式在執行，使得一個 sequential read access 的 requests 被其他的 request 所打斷，而沒辦法根據連續的 request 的 logical address 來判斷是否爲 sequential read access 來做 prefetch。所以我們以 thread 爲單位來紀錄一個 sequential read access，一個 thread 就代表一個連續的讀取單位，每個 thread 會紀錄起始的 logical address、結束的 logical address 以及目前已連續讀取的多少 sectors，主要的目的是用來判斷做 prefetch 的時機，以及避免 sequential read 的時候被其他的 request 所打斷而無法做 prefetch。

以 figure 15 爲例，左邊是 OS 依序對 device 下的 requests，很明顯可以看出當中交錯著三個 sequential read access，圖示是爲了好說明 thread 的處理方式，實際上 thread 只會記錄起始位址、結束位址和連續了多少 sectors，當執行 sector 0 這個 request 時，我們就會建立一個 thread 紀錄此 sequential read 情形，執行到 sector 1，發現剛好有接在 thread 1 後面，就更新 thread 1 的資訊，接下來的 sector 63 跟 sector 127 都沒與 thread 相連接，就各自建立一個 thread，之後以同樣的方式繼續執行下去，到最後有多少個 thread，就代表有多少個 sequential read access。

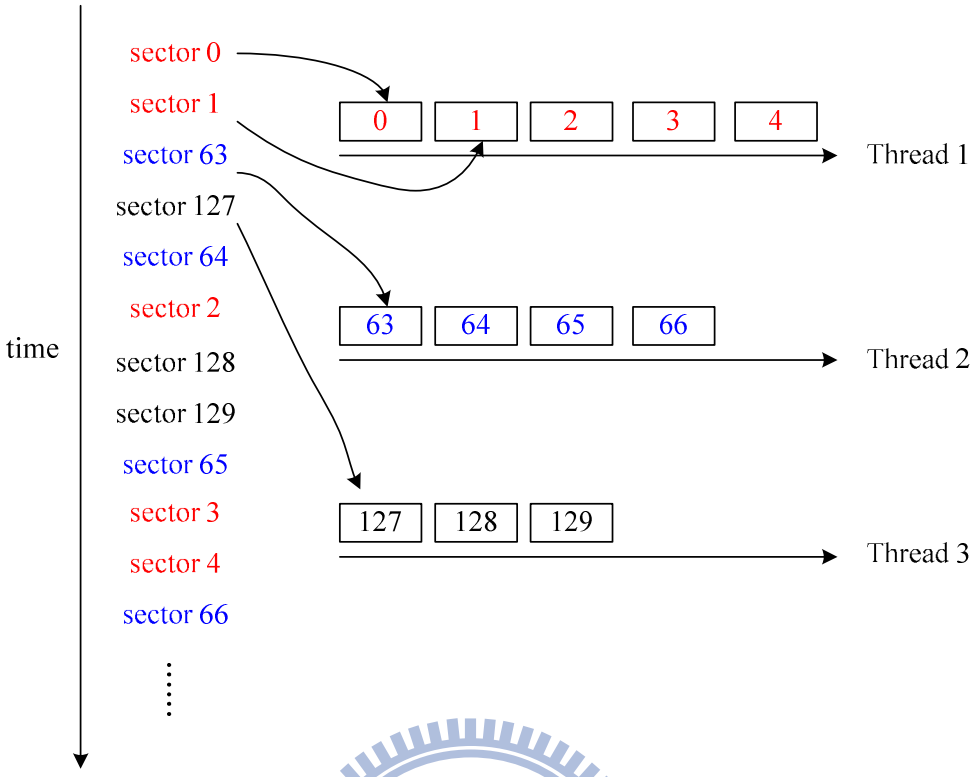


Figure 15. The example for the function of the thread

當一個 request 進來時，因爲無法判別 random read 或 sequential read，所以我們都同樣會建立 thread information，這邊存在一個技術議題，因爲作業系統中不可避免有許多 thread 交錯並發出比較隨機的讀取動作，這些隨機讀取會干擾我們紀錄 sequential read access 的 thread，所以我們提出用 2-level thread table 來管理這些 thread，分別爲 continuous thread table 跟 random thread table，並設定每個 table 中可記錄的 thread 個數，continuous thread table 主要是紀錄至少有兩個 request 相連接的 thread，random thread table 則是紀錄只有一個 request 構成的 thread，也就是目前被認定爲 random read request 的會記錄在 random thread table，用 2-level table 主要是避免 table 裡的 sequential read access 的 thread 太快被 random read request 替換掉。

每次要處理一個 read request 的時候，會先搜尋 continuous thread table 中的 thread，檢查此 request 的 logical address 是否有接續在當中某一個 thread 後面，若有則更新 thread 中的 end address 跟 sequential read sectors，並判斷有沒有達到 prefetch 的條件，如果沒有找到 thread 與此 request 的 address 相連接，則再搜尋 random thread table，同樣地，如果有接續在 thread 的後面，我們就更新此 thread 的資訊，並且將此 thread 移動到 continuous thread table，當 continuous thread table 中的 thread 超過設定值，我們就用 LRU 的策略移除掉最久沒被更新的 thread，如果在 random thread table 一樣沒有找到相連接的 thread，就建立一個新的 thread 放到 random thread table，如果 random thread table 中的 thread 超過所設定的個數，一樣是用 LRU 的方式移除

thread。

3.4 Buffer replacement

我們做 prefetch 之後，因爲不確定 prefetch 的資料是否會被使用，所以我們會先將 prefetch 的資料放到 prefetch buffer，除了放 prefetch 的資料外，爲了避免重複讀取，我們 prefetch buffer 還會存放那些因爲 multi-channel 的關係而一起讀取的資料，如果之後有 read request 需要的資料有在 prefetch buffer 裡，就會直接去 prefetch buffer 讀，省下再去從 flash memory chip 索取的時間，而當 prefetch buffer 空間不夠時，就必須將一些資料從 prefetch buffer 釋放出來，要選擇釋放那一個資料是此章節要探討的議題。如果被釋放的資料在短時間內仍然會再被讀取時，我們稱作 **buffer miss**，就是釋放掉錯誤的資料，使得 controller 要重新再去 flash memory chip 讀取而增加 response，一個好的 buffer replacement 要讓 buffer miss 的機率降到最低。

Prefetch buffer 在此主要的目的是提供空間給 prefetch 的資料，當執行 prefetch 時，一定是因爲判斷出它有可能是 sequential read access，所以放進 prefetch buffer 時也會是 sequential 的資料，因爲此種特性，當 prefetch buffer 空間不夠需要釋放資料時，偏好使用的策略是 First-In-First-Out(FIFO)，如果被選擇要釋放的資料仍然還沒被讀過的話，那麼那一次 prefetch 的資料也都還沒被使用過。不使用 LRU 的策略是因爲 LRU 的好處是在於對同位置的 logical address 做重複讀取，prefetch buffer 主要是存放 prefetch 的資料，而且大部分重複讀取的資料也會被 Operating system 內部的 buffer 所 buffer 住，又因爲 read 的特性使然，基本上讀過的資料就不會再讀了，所以 LRU 跟 FIFO 應用在 prefetch buffer 的效果是差不多的。

而在 4.3 章節的實驗部分，我們會觀察不同的 prefetch buffer size 和 FIFO 跟 LRU 這兩種不同的 buffer replacement policy 對於 prefetch 的影響。

四、 Experimental result

4.1 Experimental setup and performance metrics

實驗的 workload 是在 real-life UMPC 上的 disk I/O trace，disk size 是 18.75GB，而作業系統跟 file system 分別是 Windows XP 和 NTFS，我們將在這台 UMPC 上收集爲期一個月的 I/O trace replay 在一顆模擬的 20GB SSD 上，我們的實驗主要是觀察 prefetch 對 read request 的影響，所以會過濾掉 trace 裡所有的 write request，如果 prefetch 有效，就會增加 flash memory chips 的平行度而使得總執行時間減少，若 prefetch 的 overhead 過高就會增加執行時間，所以我們就以 read 的 throughput 來觀察 prefetch 的效果。而 SSD 的詳細規格見 table 1，所有的實驗沒有特別說明的話，

都是在這個設定下完成 。

Default system configuration	
Flash memory capacity	20GB
Number of channels	8
Number of CEs per channel	4
Page size	2KB
Block size	128KB
Read ahead buffer	4MB
Page read (setup)	1us
Page read (busy)	100us
Page read (data transfer)	30us
Number of continuous sectors to perform prefetch	256 sectors
Prefetch size	512 sectors
Thread table entry	20
Prefetch buffer replacement policy	FIFO

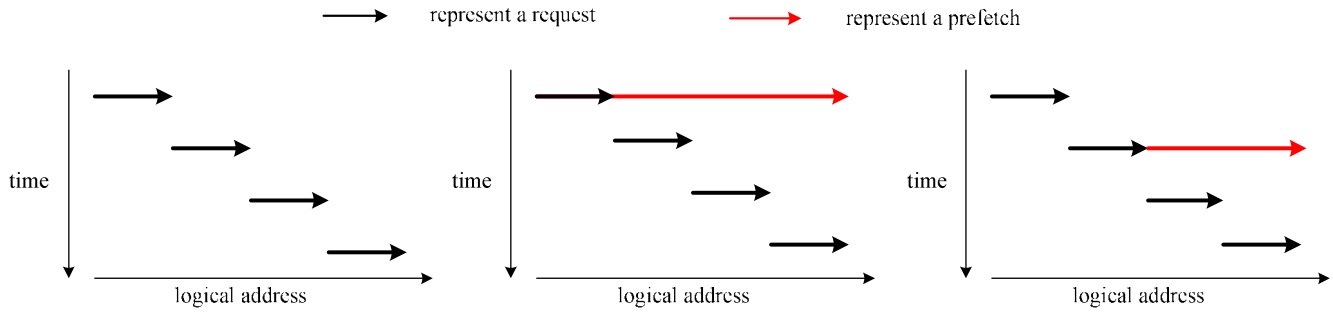
Table 1. Default system cofiguration

實驗中會有一個比較的對象是 OPT，指的是 optimal prefetch，它的方法是每處理一個 read request，就會往後搜尋 100 個 read request，看當中是否有 request 和目前的 request 是 sequential 的，搜尋 100 個 request 後，看此 request 之後會 sequential read 到那個 address，就 prefetch 到那裡，如果 prefetch size 大於 prefetch buffer 時，我們就最多就只 prefetch 到 prefetch buffer 的 size。

以 figure 16 來看，一個 sequential read access 被分成 4 個 request，figure 16.(a) 呈現的是 non-prefetch 的情形，完全根據 OS 對 device 下的 command 依序執行，在此我們稱爲 **by-requested**，而有些 request 沒有對齊 channel 邊界，因爲 multi-channel 的關係而使運作的 chips 所讀取的 idle slots 資料該不該留下，這個問題我們在 4.3 的時候觀察這兩者之間的影響，但在其他實驗沒有特別說明外，by-requested、prefetch 以及 OPT 都是會保留那些資料的，而這些資料也都是放到 prefetch buffer 當中。

在 figure 16.(b)中當第一個 request 來時就已經判斷出要 prefetch 到哪裡，即圖中紅色的線段，這麼一來，OPT 在 request 一進來就會馬上判斷是否要做 prefetch，並且 prefetch 的命中率是 100%，而這在現實上是不可能達成的，把 OPT 拿來做比較，主要目的是可以知道 prefetch 所能改善的效能的上限，而在本篇論文中，所使用的 prefetch 方法如 figure 16.(c)，當發現 sequential read 超過一個定值之後，我們才會去做 prefetch，跟 OPT 相比，我們 prefetch 的時機較晚而且 prefetch 的命中率無

法達到 100%。



(a) By-requested (b) OPT prefetch (c) prefetch

Figure 16. The concept for the different prefetch situation

實驗結果主要是觀察 by-requested、prefetch 跟 OPT 的 throughput，以及 prefetch 在不同的硬體架構跟 prefetch 機制下的影響，實驗會去做調整的包括 read ahead size、prefetch buffer size and replacement、number of channel 以及 number of CE per channel。

4.2 Read ahead size

在 section 3.2 所提到的 prefetch 需要考慮的是時機跟 prefetch size，而 prefetch 主要是加快 sequential read access，但要如何判斷說它是 sequential read 並且該開始做 prefetch，我們發現當 read request 連續讀了一定量的 sectors 後，就會有很大的機會接著讀下面連續的 sectors，所以我們用目前已 sequential read 多少 sectors 來判斷 prefetch 的時機，用 thread 爲單位來記錄此 sequential read access 已連續讀了多少 sectors。而 figure 17 是這份 trace 裡是 sequential read access 的統計，主要是觀察 sequential read 讀了多少 sectors 後，仍然會再繼續讀下去多少 sectors，若考慮在 sequential read 64 sectors 或 128 sectors 就做 prefetch 的話，因爲只會再讀 0 至 127 sectors 的 thread 佔很大的比例，所以此時做 prefetch，prefetch size 太大反而會造成更多的 overhead，而再來看 sequential read 256 sectors 跟 512 sectors 的部份，這兩個會再讀 0 至 127 sectors 的 thread 就明顯少了很多，而會再讀 256 sectors 以上的 thread 數也跟其他 prefetch 的時機差不多，在 3.2 的部份也有提到 prefetch size 的影響，所以 prefetch 的時機我們會選在 sequential read 256 sectors 或 512 sectors，這麼一來我們選較大的 prefetch size 所造成的 overhead 會比較小。

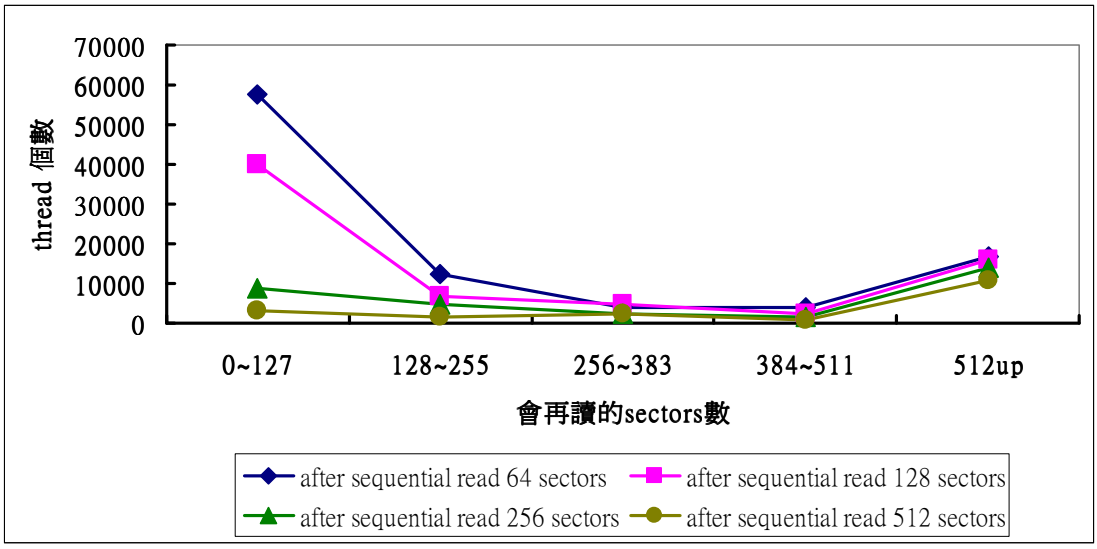


Figure 17. The statistics for the different sequential read sizes

figure 18 是在 8 channels 的架構下分別對不同的 prefetch size 跟不同的 prefetch 時機的實驗結果，發現每次 prefetch 512 sectors，在 sequential read 128 以上 sectors 時都有較好的 throughput，只有在 sequential read 64 sectors 時表現較差，主要是因爲 sequential read 64 sectors 之後有太多的 thread 只會再讀 0 至 128 個 sectors，造成太多的 overhead。從 figure 17 可得知不管在什麼時機做 prefetch，大多數的 thread 都只會再讀 128 以內的 sectors，但是從 figure 18 看來，若每次只 prefetch 128 sectors，它的 throughput 都不盡理想，就如同 section3.2 所提，prefetch size 越大，所能改善的效能就越大，而且 OS 將大的 sequential read access 大部分都會分割成數個 128 個 sectors 的 request，我們 prefetch 主要想改善的就是較大的 sequential read access，所以每次 prefetch 128 sectors 跟不做 prefetch 是差不多的。prefetch size 越大，如果 prefetch 命中率高，就能改善越多的 throughput，但是命中率低時，我們就必須額外的 overhead，就如同在 figure 18 中，在 sequential read 64 sectors 和 128 sectors 後就 prefetch 1024 sectors，這兩者的效能差了約 17%。

對於 prefetch 的時機，從 figure18 來看，sequential read 256 sectors 或 512 sectors 之後做 prefetch，而 prefetch size 從 128 sectors 到 512 sectors，這兩個時機所表現的 performance 其實都差不多，它們之間最多也只有 2%的差距而已，選擇哪一個 prefetch 的時機其實是差不多的，所以我們選擇 sequential read 256 sectors 之後就做 prefetch。而對於 prefetch size 的選擇，一樣從 figure 18 來看，已決定了 sequential read 256 sectors 之後做 prefetch，每次 prefetch 512 sectors 才可以達到最好的 performance。

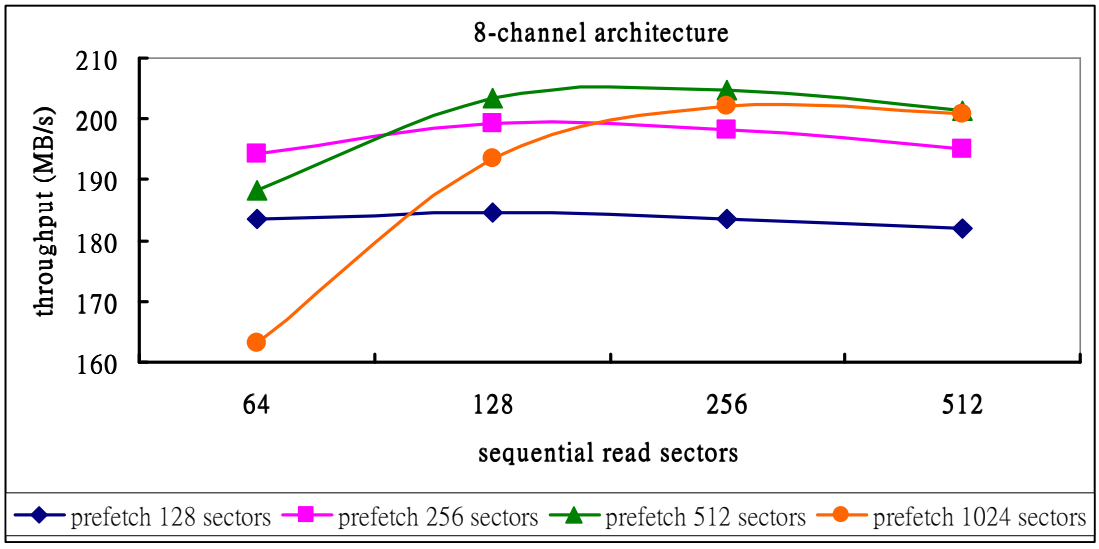


Figure 18. The different prefetch policies on the 8 channels architecture

figure 19 是在 2 channel 架構下的實驗結果，我們發現在這四個 prefetch 時機下每次只 prefetch 128 個 sectors，跟其他的 prefetch size 比較起來，仍然還有不錯的 throughput，主要的原因在於 overhead 的問題，因爲在 2 channels 的架構下，它將 data 傳回 host 的時間比 8channel 的架構慢了四倍，這樣當 prefetch 到不需要的資料時，跟 8 channel 相比 overhead 也就多了四倍，所以在 2 channel 的架構下，反而要比較注重 prefetch 的命中率，但是因爲是 2 channel 的架構，request 的 size 不需要太大就能夠使所有的 chip 都在 busy，關於 channel 數量的因素，我們會在 section 4.4 實驗結果的地方說明。

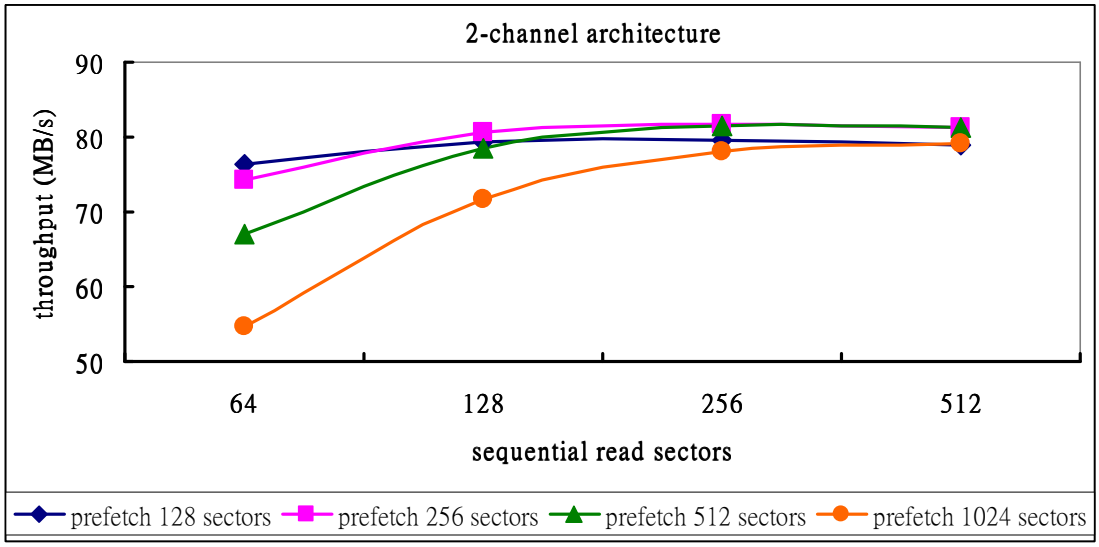


Figure 19. The different prefetch policies on the 2 channels architecture

另外，一般在 RAM 上做 prefetch 大部分也是觀察它 sequential read 的情形來做

prefetch，但是在 RAM 上是以硬體的方式來執行 prefetch，並沒有 thread table 這種架構來處理 multi-tasking 的問題，而我們不使用 thread table 來模擬這種情形，單純以 device 連續接收到的 requests 來判斷是否是 sequential read access，我們分別對不同的 prefetch 時機但 prefetch size 均爲 512 sectors 來做實驗，而 figure 20 是實驗結果，從圖中我們可以看出這兩者之間的差別，尤其當 prefetch 的時機越後面時差別越大，主要是因爲 request size 大部分都是在 128 sectors 以下，要 sequential read 256 sector 而不被其他的 request 所打斷比較困難，只有在 OS 只專心處理一個 task 時才容易判斷它是否是 sequential read acces，而使用 thread table 就能夠明確的從交錯的 sequential read requests 判斷出 sequential read access。

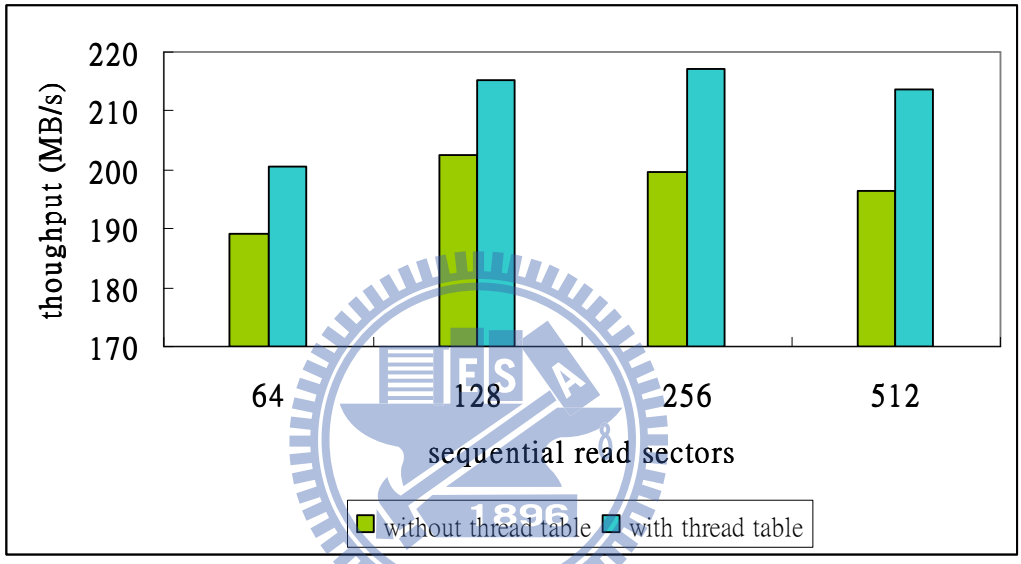


Figure 20. The experimental result for the effect of using thread table

4.3 Prefetch buffer size and replacement

Prefetch 的資料都會先放到 prefetch buffer 裡，這個章節要觀察不同的 prefetch buffer size 跟 buffer replacement 的影響，prefetch buffer size 分別爲 256KB、512KB、1MB 直到 32MB，figure 21 爲實驗結果，by-requested、prefetch 以及 OPT 都是 prefetch buffer size 越大 throughput 就越高，當中 OPT 成長速率最大，在 buffer size 是 256KB 時，OPT 的 improvement 爲 26%，而當 buffer size 增加到 32MB 時，OPT 的 improvement 增加爲 56%，這之間的差別主要是因爲 OPT 的 prefetch size 限制爲 buffer size，對 OPT 而言，prefetch size 直接影響 interleaving 的效能，當 prefetch buffer size 越大時，OPT 幾乎都可以把一個 sequential read request 全部 prefetch 完，反觀 prefetch 及 by-requested，prefetch buffer size 變大單純只增加空間放 prefetch 的資料，用以增加資料留在 buffer 的時間，所以導致它們的成長速率比 OPT 平緩很多，另外，因爲 prefetch policy 每次最多只 prefetch 256KB，所以 prefetch buffer size 從 256KB 到 1MB 之間 prefetch 的 throughput 增加的比較明顯，當 prefetch buffer 超過 1MB 後，增加

的速率就變的較平緩了。

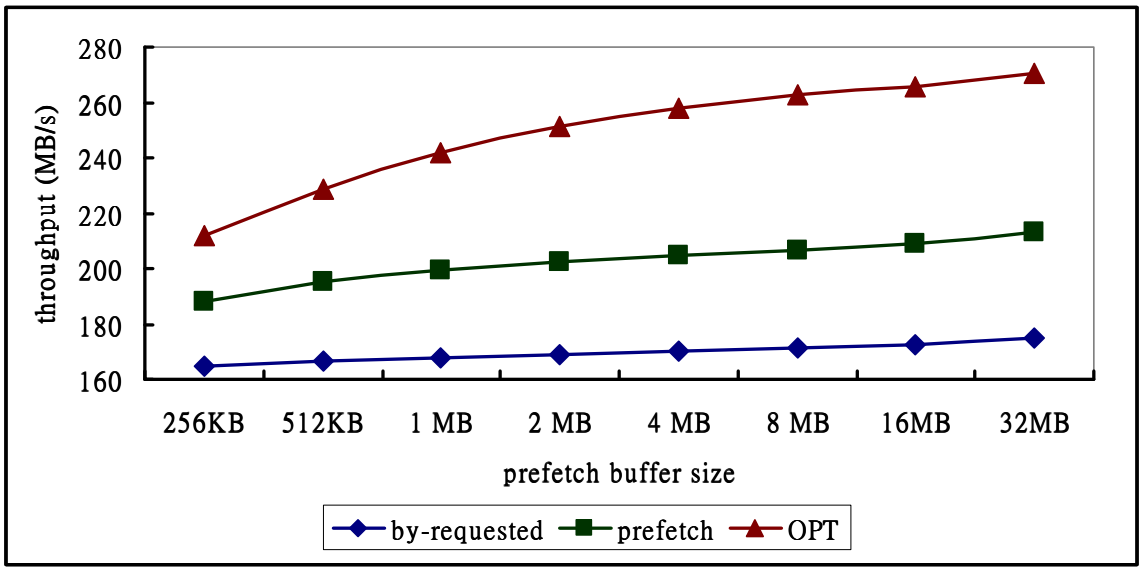
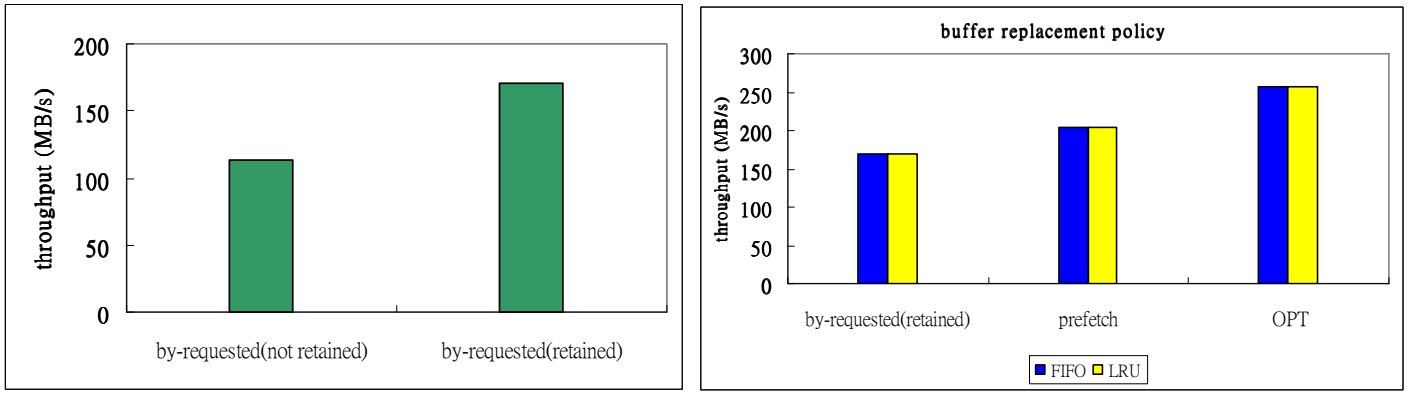


Figure 21. The experimental result for different prefetch buffer sizes

在 2.3 所提到的那些沒有對齊 channel 邊界的 request 而出現一些 idle slot，對於這些 idle slots 處理的方式有兩種，一種是這些 slot 所讀取的資料不留下來，全部遺棄掉，單純只會讀 request 所要的資料，另一種則是將 idle slot 的資料也讀取出來並放到 prefetch buffer，在此我們將這兩種處理方式應用在 **by-requested** 的情形下做實驗 (by-requested 定義請參考 4.1 部分)，而 figure 22(a)是實驗結果，by-requested (not retained)指的就是第一種情形，不會留下 idle slot 所讀取的資料，而 by-requested (retained)則是第二種，將那些 idle slot 讀取的資料儲存在 prefetch buffer 中，而這跟第一種情形相比，可以改善到約 50%的 performance，由此我們得知，有很多的 requests 都不會對齊 channel 邊界，而如果不對那些 slot 加以處理，會有很多重複讀取出現。所以之後不論是 by-requested、prefetch 還是 OPT，對於那些 idle slots，我們都是會把那些 idle slot 的資料讀出來放到 prefetch buffer 裡，以備不時之需，同樣的，在做 buffer replacement 時這些資料也會跟 prefetch data 一起進入 FIFO policy 所選擇釋放的資料裡面。

接下來 Figure 22(b)是對於 FIFO 跟 LUR 這兩種 buffer replacement policy 實驗的結果，就如同 section3.4 所說，因爲讀過的資料基本上就不會再讀了，以及 OS 內部的 read cache 會儲存一些讀過的資料，所以 FIFO 跟 LRU 這兩種 replacement policy 應用在 prefetch buffer 表現的效果是差不多的，我們所使用的 policy 也就以 FIFO 爲主。



(a) (b)

Figure 22. (a)The experimental result for two cases of by-requested
(b)The experimental result for the different buffer replacement policies

4.4 Number of channels

Channel 的數量影響到 logical unit size 的大小跟整體的 throughput，2 channels 跟 8 channels 的 logical unit 分別爲 2 pages 和 8 pages，而在此 multi-channel 下，一次的 read operation 單位就是一個 logical unit，同樣是一個 page read operation 的時間，在 2 channels 的架構下只能讀兩個 pages，而在 8 channels 就能讀 8 個 pages，throughput 就整整差了 4 倍，這也是現今 SSD 均採用 multi-channel 的原因。

而把 channel 數增加在 by-requested 也同樣會增加 throughput，在此章節我們要觀察在不同的 channel 數下 prefetch 對於 by-requested improvement 的影響，figure 23 爲此實驗結果，從圖可以發現，當 channel 數從 2 增加到 8 時，prefetch 的 improvement 逐漸增加，在 2 channels 時 prefetch 跟 OPT 僅改善了 7%和 28%的 throughput，而在 8 channels 時 prefetch 跟 OPT 改善了 22.5%和 49.5%的 throughput，這其中的原因在於 channel 數越多，一次 read operation 的 logical unit size 就越大，這麼一來即使 prefetch 到不需要的資料時，從 flash chip 傳送到 host 也會比較快，大大地減少了 prefetch 所造成的 overhead，而且當 prefetch 成功命中時，因爲 channel 數多，所以傳輸的速度較快，造就了在 8 channels OPT 那驚人的 improvement。

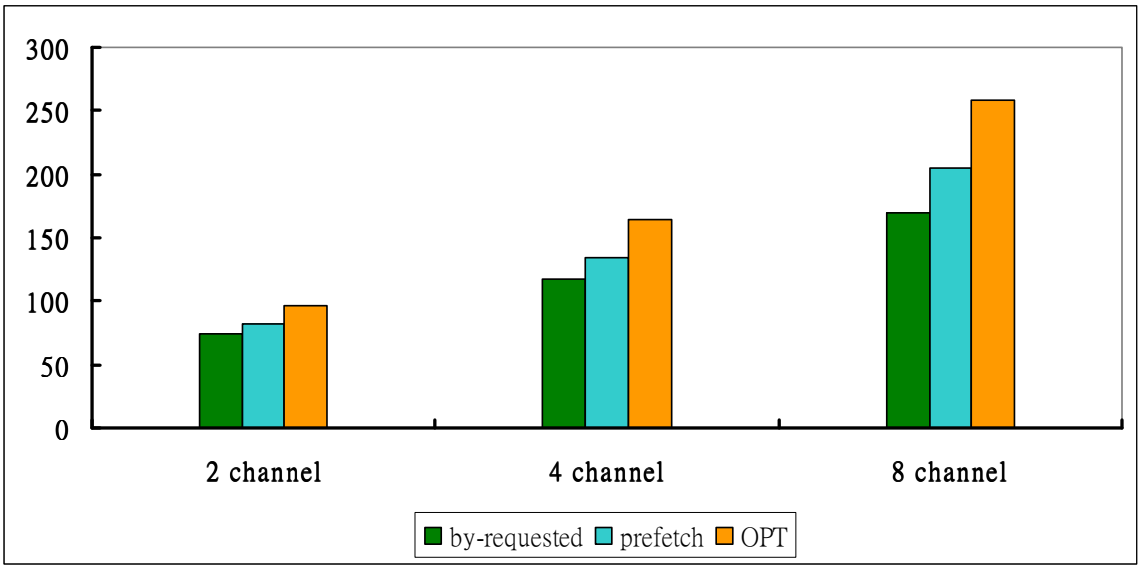


Figure 23. The experimental result for different number of channel

4.5 Interleaving degree

在 2.2 介紹的 interleaving 能改善的部份是因爲每個 CEs 在 busy phase 時所重疊的時間，以及利用其他 CE 在 busy 時趁機將資料回傳給 host，但並不是 channel 裡 CE 越多，所得到的 improvement 就會成比例的增加，實際上 interleaving 跟 CE 的個數和 read operation timing model 的比例息息相關，以表 1 裡的 timing model 爲例，假設分別在 1 channel 2CEs、4CEs 跟 8CEs 讀 16 個 pages，跟在 1 channel 1 CE 相比，它們的 response 分別減少了 48.5%、70.5%以及 72%，發現從 4 CEs 增加到 8 CEs 時，它所改善的 response 只有 1.5%，其主要原因是當 CE 數大於 4 時，CE1 在 busy phase 後要先等其他的 CEs 傳送完 data 給 host 之後它才會進入 data transfer phase，使得 CE 1 有一段時間是處於 idle 狀態如 figure 24(c)，反觀 figure 24(a)跟 figure 24(b)，在 4 CEs 時，在 CE1 busy phase 結束後其他 CE 的 data transfer phase 也剛好完成，就不會出現 CE idle time，但在 2 CEs 時，因爲 CE2 在 data transfer 後 bus 空了出來，但是 CE1 仍處在 busy 階段，使得 bus 在這段時間出現了 idle，這麼一來 interleaving 利用會不完全。當固定了 flash busy time 和 data transfer time 時，最佳的 interleaving degree 也就被確定了，大約會是 busy time 跟 data transfer time 的比例再加一，因爲我們主要是要將 busy 的時間給重疊起來，如果 CE 數量太多，如 figure 24(c)，這樣 busy time 重疊的部份只有連續的四個 CE，第五個 CE 的 busy time 就不會和第一個 CE 的 busy time 重疊，所以當 CE 數越多，在此 timing model 下，最多也只有 4 個 CE 的 busy time 會重疊，因此不會得到等比例的 improvement。

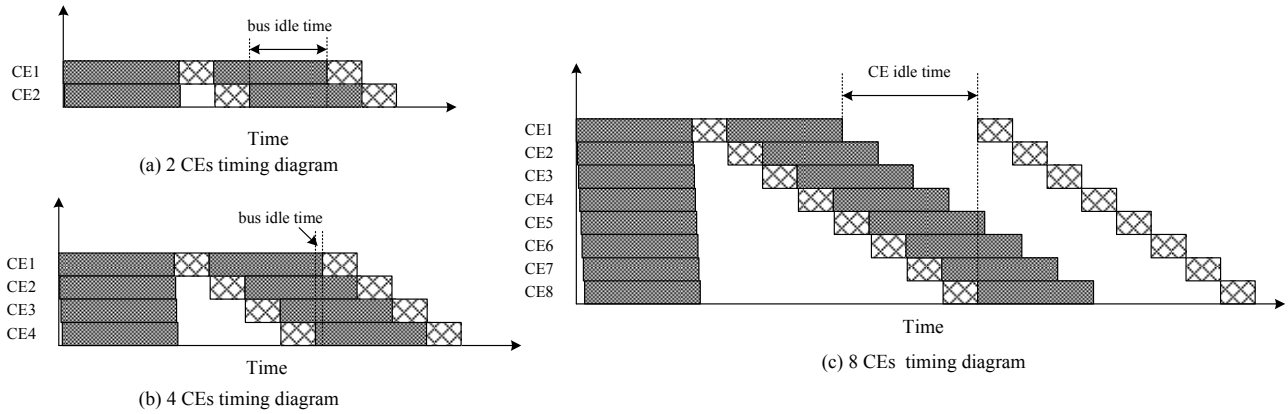


Figure 24. The interleaving diagrams of different number of CE

接下來分別對 2 CEs per channel、4 CEs per channel 以及 8 CEs per channel 做實驗，觀察在不同的 CE 數下 perfetch 對 by-requested 的影響，將 channel 數固定爲 8 且 read ahead buffer 爲 4MB，而實驗結果如 figure 25，在 2 CEs per channel 的架構下，發現 prefetch 幾乎沒有改善，因爲 bus idle time 太長，使得 interleaving 能改善的效能有限，剛好跟 prefetch 所造成的 overhead 互相抵消，而 OPT 也僅僅改善 11%的 performance。而在 4 CEs per channel 跟 8 CEs per channel 的架構下，它們只有些微的差距，其原因就如同上面所提的，CE 數太多會造成太多的 CE idle time，而就成本來考慮，在此 timing model 下 4 CEs per channel 的架構會優於 8 CEs per channel 的架構。

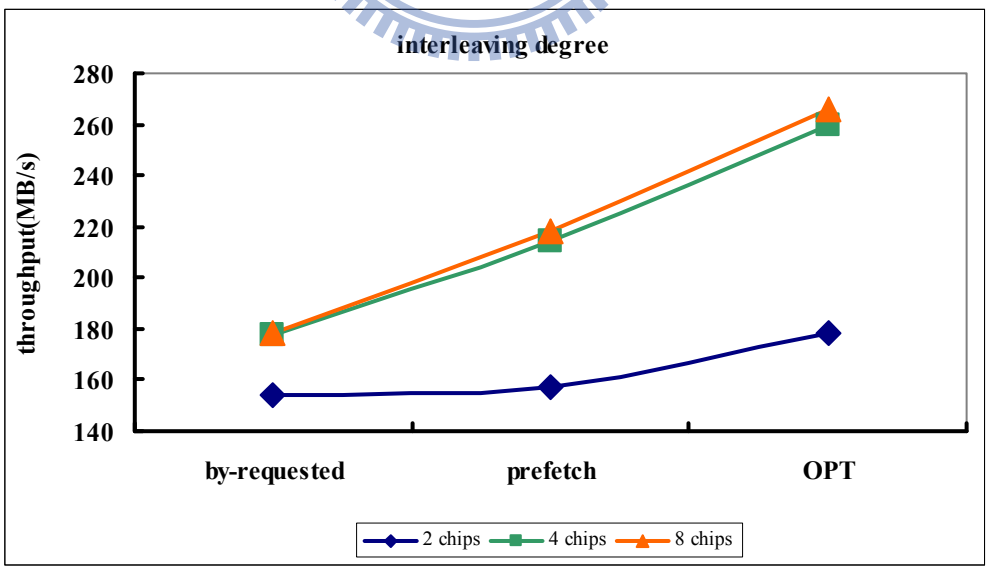


Figure 25. The experimental result for different number of CE

五、 Conclusion

由於 SSD 的容量越來越大，SSD 裡的 flash chip 越來越多，都會利用 multi-channel 跟 shared bus 的方式來組織 chips，而且因爲 device 每次只能接受一個讀取以及 OS 會把 sequential read access 分爲許多 request，導致 request 沒辦法讓所有的 chips busy，大部分的 chips 都在 idle，使得 interleaving 沒有辦法發揮作用。

我們提出一個 prefetch 的方法來預測 sequential read access，將這些 sequential read access 的 requests 串聯起來，充分利用 interleaving 來減少 response，因爲現今 OS 都支援 multi-tasking，使得在判斷 sequential read access 的時候有可能被另一個 sequential read access 或是 random read 所干擾，對於這個問題，我們利用 2-level thread table 來解決。我們額外提供一個 prefetch buffer，主要只用來存放 prefetch 的資料，而 prefetch buffer 的 size 從 section 4.3 可得知不需要太大，只需要能夠負擔數個 prefetch 的 size 即可，對於 prefetch buffer 的 replacement policy，因爲 read 的特性，基本上讀過的資料就不會再讀了，所以我們就用 FIFO 而不用 LRU。

我們也在不同的硬體架構下觀察 prefetch 的影響，在 2-channel 跟 4-channel 的架構下，prefetch 所能改進的效能不盡理想，主要是因爲在 2-channel 和 4-channel 下 bandwidth 較少，prefetch 到不需要的資料時傳輸出去的時間較長，所以 prefetch 所造成的 overhead 也較嚴重。而從 interleaving 的 degree 來看，interleaving degree 越大雖然 throughput 會越多，但是當 interleaving degree 大於一個定值，能改善的效能就急遽減少，而這個定值取決於現今 flash chip 的 timing model。而在 8-channel 跟 4-way interleaving 的 architecture 下，我們提出的 prefetch policy 可以改進 20%的 throughput。

現今 SSD 的容量仍然逐漸增加當中，若沒有解決 data bus bandwidth 的限制或是新的架構出現，shared bus 的情形是必須的，就會出現 request 無法使所有 chips 都 busy 的問題，所以需要利用 prefetch 將 sequential read access 連接起來，使 SSD 的效能提升。

References

1. Jeong-Uk Kang, Jin-Soo Kim, Chanik Park, Hyoungjun Park, Joonwon Lee, "A multi-channel architecture for high-performance NAND flash-based storage system" ,Received 9 December 2005; received in revised form 5 January 2007; accepted 8 January 2007;Available online 1 February 2007
2. Li-Pin Chang, Tei-Wei Kuo, "An Adaptive Striping Architecture for Flash Memory Storage Systems of Embedded Systems," The 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2002) September 24 -27, 2002. San Jose, California.
3. Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, Rina Panigrahy , " Design Tradeoffs for SSD Performance", USENIX '08 Refereed Paper Pp. 57–70 of the Proceedings.
4. Samsung Elec. 2Gx8 Bit NAND Flash Memory (K9WAG08U1A). 2006.
5. Samsung Elec. 2Gx8 Bit NAND Flash Memory (K9GAG08U0M-P).2006.
6. Cagdas Dirik, Bruce Jacob, " The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization ", Proceedings of the 36th annual international symposium on Computer architecture ,@ 978-1-60558-526-0, Austin, TX, USA, 279-289, 2009, http://doi.acm.org/10.1145/1555754.1555790, ACM