

2003 年 7 月

Serial ATA

原生指令排序

Serial ATA 令人驚喜的新效能特性

白皮書作者：

Intel Corporation 和 Seagate Technology



摘要

「Serial ATA II : Serial ATA 1.0 規格延伸」(可從 www.serialata.org 下載)中所推出的先進特性相當多,而且相當令人期待,其中一項是原生指令排序(Native Command Queuing, NCQ)。NCQ 是威力強大的介面/磁碟技術,可以讓硬碟機內部的工作負荷執行順序最佳化,以加強效能和持久性。將硬碟機內部指令佇列內的指令順序作有智慧的重新排序,可將硬碟機上的機械定位等待時間盡量縮短,以協助增進佇列工作負荷的效能。本文將提供基本概念,讓讀者了解如何套用 NCQ 特性來完成儲存解決方案,以及軟體開發人員如何加強其應用程式,以利用 Serial ATA NCQ 藉此建立效能更強的應用程式。

簡介

在硬碟機(HDD)等大量儲存裝置上存取媒體,對整體系統效能可能會有負面影響。硬碟機和現代系統中的其他純粹電子元件不同,它主要仍然是機械裝置。硬碟機因為機械元件速度遲緩而受限,這種情況實際上限制了媒體存取和資料擷取的速度。只有達到某個點,才可以實際改進機械效能,而且要增進效能,通常會使機械元件關鍵的成本增加。不過,以智慧的方式對機械程序的順序進行內部管理,可以大幅增進整個工作流程的效率。這其中的關鍵詞彙是「智慧型」和「內部」,表示硬碟機本身必須存取目標邏輯區塊位址(LBA)的位置,然後作適當的決策,亦即決定需要以哪種順序來執行指令,以達到最高效能。

原生指令排序是 Serial ATA 的指令通訊協定,可讓多重指令同時在硬碟機內等待處理。支援 NCQ 的硬碟機具有內部的佇列,在此佇列中,可以依照必要的追蹤機制,看看工作負荷中有哪些未完成和完成的部分,然後動態地重新排定指令的順序。NCQ 另外還有一項機制,可讓主機在硬碟機為另一項指令搜尋資料時,對硬碟機發出其他的指令。

Microsoft Windows* 和 Linux* 之類的作業系統愈來愈常利用多重執行緒軟體或以處理器為主的超執行緒技術(Hyper Threading Technology)。當硬碟機有許多指令同時等待處理時,這些特性也能夠建立工作負荷。利用 NCQ,就可以對這些工作負荷大幅提高潛在的磁碟效能。

硬碟機基礎

硬碟機是電子機械裝置,因此混合了各種電子機械元件。硬碟機的機械部分容易磨損,而且也是影響效能的重要因素。以下針對如何在硬碟機上放置資料作簡短的討論,以協助讀者了解機械限制。

資料寫入硬碟機的方式,是將資料記錄到稱為「磁軌」的一圈圈同心圓中,依序往圓心累計編號,磁碟最外緣的磁軌為零軌,第一個讀寫磁頭為零號磁頭。在磁碟的一面記錄完一圈(零號磁頭上的零軌)時,硬碟機開始寫入磁碟另一面的下一個磁頭(零軌和一號磁頭)。磁軌在一號磁頭上完成時,碟機開始寫入磁碟另一面的下一個磁頭(零軌和二號磁頭)。這項程序會持續下去,直到最後一片磁碟最後一面上的最後一個磁頭完成第一個磁軌為止。之後,硬碟機會開始記錄第二個磁軌(一號磁軌和零號磁頭),繼續進行它在記錄零號磁軌時所作的相同程序。這個程序會形成一圈圈的同心圓,因為在持續記錄資料時,資料的移動會愈來愈靠近磁碟的內圈。所有磁頭上相同圓周的特定磁軌,或是磁碟兩面上的特定磁軌,統稱為磁柱。因此,資料是運用從硬碟機外圈開始的磁柱,循序放置在磁碟上。

Serial ATA 原生指令排序

目前在機械方面的一大挑戰是，應用程式很少以資料寫入磁碟的順序來搜尋資料，而是搜尋散佈在硬碟機各個部分的資料。將適當的讀寫磁頭放置在正確旋轉位置中的正確磁軌上，所需要的機械動作相當重要。

對搜尋等待時間影響最大的機械指令執行時間，是搜尋等待時間和旋轉等待時間。搜尋和旋轉等待時間方面的需求，都必須以緊密的最佳化演算法來處理。

可盡量縮短搜尋和旋轉等待時間的演算法，最出名的一種稱為「旋轉位置排序」(Rotational Position Ordering)。旋轉位置排序可讓硬碟機選擇在媒體上執行指令的順序，以便盡量縮短存取時間，進而充分提高效能。存取時間包含了搜尋時間（將驅動臂定位）以及等待時間（等待資料在磁頭下旋轉），搜尋時間和旋轉等待時間都可能花費幾毫秒。

早期的演算法只是盡量縮短搜尋距離以便盡量縮短搜尋時間，但若目標位置需要較長的旋轉等待時間，才可等待旋轉到磁頭底下的位置，則搜尋時間太短可能會造成整體存取時間加長。當考慮到執行指令的順序時，旋轉位置排序會考慮磁碟的旋轉位置和搜尋距離。如果要提高效能，執行指令的順序一定要盡可能縮短整體存取時間（搜尋時間加上旋轉等待時間）。

原生指令排序可讓硬碟機利用旋轉位置排序，以最佳的方式將指令重新排序，以充分提高效能。

搜尋等待時間最佳化

讀寫磁頭停放到包含目標「邏輯區塊位址」(LBA) 的正確磁軌所需要的時間，即為搜尋等待時間。為了滿足多項指令，硬碟機將需要存取所有的目標 LBA。如果沒有佇列作業，硬碟機就必須依據發出指令的順序來存取目標 LBA。但若在同一個時候，硬碟機的所有指令都在等待處理，硬碟機就可以用最佳的順序來滿足指令。縮短搜尋等待時間的最佳順序，即為盡量縮小機械移動量的順序。

升降電梯是一個相當簡單的譬喻。如果依按下按鈕的順序到每一個樓層，就很沒有效率，而且在不同目標位置之間來回，很浪費時間。這聽來似乎很瑣碎，不過目前大部分硬碟機還是用這種方式操作。現在升降電梯已經進步到知道要將目標重新排序，使作業模式更具經濟效益、更快速。有了 Serial ATA，不僅可以從特定起點重新排序，而且重新排序的設計是動態的，換言之，您隨時都可以將其他指令加入佇列。這些新指令會納入進行中的執行緒，或是延後到下一系列的指令來執行，主要視指令與未完成工作負荷的配合情況而定。

如果將升降電梯的譬喻轉換成硬碟技術，只要接受已排成佇列的指令（按下欲前往之樓層），並將之重新排序，就可以有效率地提供主機所要求的資料，縮短硬碟機的機械指令執行時間。硬碟機執行一個指令時，新的指令可能進入佇列，並且被整合到未完成的工作負荷中。如果新指令在程序上剛好是最符合機械效率，它就會是下一個待完成的指令。

請記住，透過最後完成的指令之邏輯區塊位址 (LBA)，完全根據磁頭的最終位置將擱置中的指令重新排序，並不是最有效率的解決方案。就像某個人按了剛剛經過的樓層按鈕，升降電梯也不會停到那層樓一樣，硬碟機會使用複雜的演算法來決定接下來最好執行哪個指令。所謂複雜，是指包括可能會切換磁頭、搜尋不同磁軌的時間、不同的作業模式（例如安靜搜尋）。納入考量的參數包括：搜尋長度、開始位置和方向、驅動臂的加速設定檔、旋轉定位（括讀取和寫入完成次數之間的差異）、讀取快取命中和讀取快取未命中、啟用寫入快取和停用寫入快取、處理相同 LBA 的 I/O 處理，以及避免指令匱乏的公平 (fairness) 演算法等等。

旋轉等待時間

旋轉等待時間是指，磁頭停在正確的磁軌之後，啟動 LBA 以便在磁頭底下旋轉所需要的時間。以最糟糕的情況來說，這可能表示，磁碟會先平白旋轉一次，才能夠存取起始的 LBA，然後繼續從其餘的目標 LBA 讀取。旋轉等待時間視主軸馬達轉速 (RPM) 而定，7200-RPM 硬碟機最多需要 8.3 msec 的旋轉等待時間，5400-RPM 硬碟機最多需要 11.1 msec 的旋轉等待時間，而 10K-RPM 硬碟機最多需要 6 msec 的旋轉等待時間。在起始 LBA 相對於磁頭角度位置的隨機分佈圖中，平均旋轉等待時間會是最糟情況時的一半。

比起任何現有系統的整體效能，I/O 延遲（以毫秒為單位）相當嚴重，如果現有作業系統利用多重執行緒，或是超執行緒技術容許幾乎同時執行獨立的工作負荷時，情況更是如此，因為這一切都需要幾乎同時從相同的硬碟機取得資料。

提高主軸馬達轉速 (RPM)，是縮短旋轉等待時間的一個方法。但是提高主軸馬達轉速會使成本大幅提高。另外兩個方法也可以將旋轉等待時間減至最少，第一是將指令重新排序，以盡量縮短旋轉等待時間，這種最佳化類似可縮短搜尋等待時間的線性最佳化，但它是將硬碟機磁頭旋轉位置納入考量，以決定接下來最好處理哪個指令。一個次階 (second-order) 最佳化是使用稱為「無順序資料遞送」(out-of-order) 資料遞送。「無順序資料遞送」是指磁頭不需要先存取起始的 LBA，但是可以開始讀取目標 LBA 內任何位置的資訊。硬碟機若要返回所搜尋資料的第一個 LBA，必須旋轉一部分，但是硬碟機並未如此做，而是一停放到正確的磁軌，就開始讀取所要求的資料，並且在同一次迴轉結束時將遺漏的資料加入。

以最糟糕的情況來說，利用無順序資料遞送，只要磁碟迴轉一次，就可以完成整體的傳輸。如果不用無順序資料遞送，則完成傳輸所需的時間，最多只是旋轉一次再加上旋轉到所有目標 LBA 所需要的時間。

原生指令佇列的好處

為了縮短機械指令執行時間以縮短 I/O 等待時間，顯然有必要將未執行的指令重新排序。但同樣顯而易見的是，儲存指令所使用的磁，只用來收集佇列中的指令，實在不划算。有效率地將演算法重新排序，會考量到目標資料的直線和角度位置，並且會為這兩者最佳化，以盡量縮短整體執行時間。這個程序稱為「根據搜尋和旋轉最佳化的指令重新排序」或標記指令排序。指令排序和縮減機械工作負荷，可使機械磨損的情況減少，使機械更為耐久。Serial ATA II 提供有效率的通訊協定，以導入稱為原生指令排序的標記指令排序 (tagged command queuing)。

原生指令排序透過有效地重新排序指令來取得高效能和效率，此外，為強化 NCQ 效能而建置到 Serial ATA 通訊協定的新功能有三種，其中包括 race-free status return、interrupt aggregation 和 First Party DMA。

- Race-Free Status Return 機制

這項特性可允許隨時通知任何指令的狀態。要使這項狀態回傳 (status return) 發生，主機並不需要進行「訊息交換」。硬碟機可能會緊接著或是同時針對多重指令發出指令完成訊息。

- Interrupt Aggregation

一般而言，硬碟機會在每次完成一項指令時岔斷主機。岔斷次數愈多，主機處理的負擔就愈大，但是有了 NCQ 之後，每項指令的平均岔斷次數就會減至不到一次。如果硬碟機在很短的期間內完成多項指令（高度排列的工作負荷，出現次數頻繁），個別的岔斷次數就可能很集中。在這種情況下，主機控制器只需針對多重指令處理一項岔斷。

- First Party DMA (FPDMA)

原生指令排序擁有一項機制，可讓硬碟機針對資料傳輸設定「直接記憶體存取」(DMA) 作業，而不需要主機軟體介入。這項機制稱為 First Party DMA。硬碟機傳送 DMA Setup FIS (Frame Information Structure) 到主機控制器，藉此選取 DMA 環境定義。這個 FIS 指定了設定 DMA 的指令之標記。根據標記值，主機控制器將該指令的 PRD 表格指標載入 DMA 引擎，傳輸便可持續進行，而不需要任何軟體介入。利用這個方法，硬碟機就可以將指令有效地重新排序，因為它可以根據自己的方法選擇緩衝區來傳輸。

NCQ 的詳細說明

NCQ 有三個主要部分：

1. 在硬碟機中建置指令佇列
2. 針對每個指令傳輸資料
3. 針對已完成的指令回傳狀態

下列幾節將會詳細說明每一個機制如何運作。

建置佇列

硬碟機必須知道它何時接收到特定的指令，不論它應該將指令排序，或是該直接執行該指令。此外，硬碟機必須了解，接收到的指令後，要使用哪種通訊協定；指令通訊協定可以是 NCQ、DMA、PIO 等。硬碟機依據發出的特定指令 opcode 來判斷這項資訊，因此，為了利用 NCQ，會定義專用於 NCQ 的指令。有兩種 NCQ 指令已加入 Serial ATA II 中的 NCQ 定義，亦即 Read FPDMA Queued 和 Write FPDMA Queued。圖 1 顯示 Read FPDMA Queued 指令輸入；Write FPDMA Queued 的輸入也類似。指令為延伸的 LBA 和磁扇計數指令，以便在目前的硬碟機中納容大容量。

指令也包含針對高可用性應用的 force unit access (FUA) 位元。針對 Write FPDMA Queued 指令設定 FUA 位元時，硬碟機會先對媒體確定資料，然後再回傳指令執行順利。視需要在寫入時使用 FUA 位元，主機就可以管理尚未在硬碟機內部快取記憶體內對媒體確定的資料量

暫存器	7	6	5	4	3	2	1	0
特性	磁扇計數 7:0							
特性 (exp)	磁扇計數 15:8							
磁扇計數	標記				已保留			
磁扇計數 (exp)	已保留							
磁扇號碼	LBA 7:0							
磁扇號碼 (exp)	LBA 31:24							
磁柱低	LBA 15:8							
磁柱低 (exp)	LBA 39:32							
磁柱高	LBA 23:16							
磁柱高 (exp)	LBA 47:40							
裝置/磁頭	FUA	1	Res	0	已保留			
指令	60h							

圖 1 Read FPDMA Queued 指令

一個有趣的欄位是「磁扇計數」暫存器中的「標記」欄位。每一個發出的佇列指令都有一個相關的標記，這個標記是速記機制，在主機和裝置之間使用，以識別特定的未執行的指令。標記值可以介於 0 到 31，雖然硬碟機可以報告對佇列深度低於 32 的佇列提供支援。在這種情況下，標記值限於硬碟機支援的最高標記值。將標記值限制在 0 到 31 之間，有一些優點，其中包括可用一個 32 位元值來報告所有指令的狀態。每一個未執行的指令必須有一個獨特的標記值。

發出 Read 和 Write FPDMA Queued 指令的方式，就和任何其他指令一樣，亦即是以特定暫存器值寫入工作檔 (taskfile)，然後再以指令 opcode 來寫入指令暫存器。佇列和非佇列指令之間的差異，是發出指令之後所發生的情況。如果發出非佇列指令，硬碟機會針對該指令傳輸資料，然後清除「狀態」暫存器中的 BSY 位元，告訴主機已完成指令。發出佇列指令時，硬碟機會先立即清除 BSY，之後才會將資料傳輸到主機。在佇列作業中，不會使用 BSY 位元來傳送指令完成訊息，而是使用 BSY 位元來傳送硬碟機是否已備妥接受新指令的訊息。一旦清除 BSY 位元，主機就會對硬碟機發出另一個佇列指令。以此方式，就可以在硬碟機中建置指令佇列。

傳輸資料

NCQ 利用稱為 First Party DMA 的特性在硬碟機和主機之間傳輸資料。First Party DMA 可讓硬碟機控制資料傳輸的 DMA 引擎程式設計。這是一項重要的加強功能，因為只有硬碟機知道硬碟機磁頭目前的角度和旋轉位置。之後，硬碟機可以選取下一個資料傳輸，將搜尋和旋轉等待時間盡量縮短。First Party DMA 機制可以有效讓硬碟機以最佳方式將指令重新排序。

在進行其他的最佳化作業時，硬碟機也可以回傳資料無排序，以進一步縮短旋轉等待時間。如果這是完成資料傳輸最有效的方法，First Party DMA 可讓硬碟機針對指令回傳部分資料，並對另一項指令傳送局部資料，然後針對第一項指令完成傳送資料。

Serial ATA 原生指令排序

若要針對資料傳輸將 DMA 引擎程式化，硬碟機會對主機發出 DMA Setup FIS（如圖 2 所示）。在 DMA Setup FIS 中有一些關鍵欄位，對於將 DMA 引擎程式化相當重要。

0	已保留 (0)	已保留 (0)	A	I	D	已保留 (0)	FIS 類型 (41h)
1	0						標記
2	0						
3	已保留 (0)						
4	DMA 緩衝區偏差量						
5	DMA 傳輸計數						
6	已保留 (0)						

圖 2 DMA Setup FIS

「標記」(TAG) 欄位指出 DMA 傳輸的指令標記之目的。為了避免主機記憶體受到不良裝置之干擾，一定要防止硬碟機隨意指定實際位址來傳輸資料到主機記憶體，以及從主機記憶體來傳送。標記的作用是作為主機中實體記憶體緩衝區的控點，如此一來，主機就不需要知道有實際實體記憶體位址存在，相反地，主機會使用標記來指出要針對資料傳輸使用 PRD 表格，並因此將 DMA 引擎程式化。

使用 DMA 緩衝區偏差量 (DMA Buffer Offset) 欄位來支援無順序資料遞送，它又稱為在規格中的非零值緩衝偏差量。非零值緩衝偏差量可讓硬碟機無順序地傳輸資料或是依順序但是以多重項目來傳輸。

DMA 傳輸計數 (DMA Transfer Count) 欄位指出要傳輸的位元組數目。D 位元指定傳輸方向（不論是讀取或寫入）。A 位元是稱為 Auto-Activate（自動啟動）的寫入最佳化，可以在寫入指令執行時免除一項 FIS 傳輸。

HBA 設計人員必須記住的一項重點是，不能在 DMA Setup FIS 以及該 DMA Setup FIS 傳輸資料完成之間發出新指令。重點是，在主動傳輸資料時不要岔斷硬碟機，因為執行一項新指令可能會造成資料傳輸中斷。所以，目前已在 NCQ 定義中明白加入這項限制。同理，硬碟機不能先傳送 Set Device Bits FIS，然後才完成該 DMA Setup FIS 的資料傳輸。這項限制有一個例外，那就是：如果在傳輸所有的資料之前發現一項錯誤，硬碟機可能會傳送 Set Device Bits，以錯誤狀態來終止傳輸。

硬碟機發出 DMA Setup FIS 之後，會使用在非佇列 DMA 資料傳輸作業中使用的相同 FISes 來傳輸資料。

狀態回傳

指令狀態回傳為 race-free（無需競爭），可容許多重指令的岔斷集中起來。主機和硬碟機搭配作業，以達到 race-free 狀態回傳，而不需要主機和硬碟機之間的訊息交換。關於哪些指令仍未完成，主機和硬碟機之間會進行通訊，而這項通訊是透過 SActive 主機中的 32 位元暫存器來處理。SActive 暫存器有一個位元配置到每一個標記，亦即位元 x 以標記 x 顯示指令狀態。如果設定 SActive 暫存器中的一個位元，就表示附有該標記的指令在硬碟機中尚未完成（或附有該標記的指令即將被發送至硬碟機）。如果清除 SActive 暫存器中的一個位元，就表示附有該標記的指令在硬碟機中尚未完成。主機和硬碟機搭配作業，以確定 SActive 暫存器隨時都是正確的。

主機可以在 SActive 暫存器中設定位元，而裝置可以在 SActive 暫存器中清除位元。這可確保，對 SActive 暫存器的更新並不需要在主機和硬碟機之間進行同步化。主機發出指令之前會設定位元，此位元對應於它即將發出的指令之標記。硬碟機順利完成指令時會清除位元，此位元對應於它剛剛完成的指令之標記。

硬碟機使用 Set Device Bits FIS 來清除 SActive 暫存器中的位元（如圖 3 所示），Set Device Bits FIS 的 SActive 欄位用來傳送成功狀態給主機。在 FIS 的 SActive 欄位中設定位元時，表示附有對應標記的指令已順利完成。主機控制器會清除 SActive 暫存器中的位元，這些位元所對應的位元，已在所接收到的 Set Device Bits FIS 之 SActive 欄位中設定成一個位元。

0	錯誤	R	狀態高	R	狀態低	R	I	R	已保留 (0)	FIS 類型 (A1h)	
1	Status Hi					SActive 31:0					

圖 3 Set Device Bits FIS

另一項關鍵特性是，Set Device Bits FIS 可以傳達多重指令已同時完成的訊息。這可確定主機只會對多重指令完成訊息收到一項岔斷。比方說，如果硬碟機幾乎同時完成附有標記 3 的指令和附有標記 7 的指令，硬碟機會選擇傳送一個 Set Device Bits FIS，此 Set Device Bits FIS 已把位元 3 和位元 7 同時設定為一個位元。如此即可順利完成兩個指令，而且保證只會產生一項岔斷。

由於硬碟機可以回傳 Set Device Bits FIS，而不會有主機訊息交換，因此可能幾乎同時收到兩項 Set Device Bits FISes。如果第二個 Set Device Bits FIS 在主機軟體已經初次處理岔斷之前到達，則岔斷會自動集中，這表示主機實際上只會處理一項岔斷而非兩項，因此可縮短指令執行時間。

應用程式如何利用佇列作業

只有在硬碟機上建置要求佇列，才可以獲得佇列作業的優點。目前桌上型電腦工作負荷中的一個主要問題是，許多應用程式一次會要求一項資料，等接收到前一項資料，才會要求下一項資料。在這種情況下，硬碟機一次只會接收一項未執行的指令。如果一次只有一項指令等待完成，硬碟機就不會執行重新排序，如此一來，排序的所有好處也不會實現。

請注意，隨著超執行緒技術問世，即使應用程式一次只發出一項要求，還是有可能建置佇列。超執行緒技術可以大幅提高多重執行緒的數量，如此一來，多重應用程式就可能讓 I/O 要求同時擱置。但是，如果稍微修改應用程式以利用佇列作業，就可以達到最佳的效能強化。

用佇列作業，實際上幾乎不需要修改做法，目前撰寫大部份應用程式都是使用同步 I/O，又稱為 blocking I/O（阻塞的 I/O）。在同步 I/O，從檔案讀取或寫入檔案的呼叫功能，一直到實際讀取或寫入作業完成時才會回傳。未來，應該會撰寫應用程式來使用非同步 I/O。非同步 I/O 是 non-blocking（無阻塞的 I/O），亦即在要求完成之前，從檔案讀取或寫入檔案的函數呼叫才會實際回傳。應用程式會檢查要提供信號的事件或是接收一項回呼，藉此判斷 I/O 是否已完成。由於呼叫立即回傳，應用程式可以繼續執行實用的工作，包括發出更多讀取或寫入的檔案函數。

需要進行數種不同檔案存取來寫入應用程式的方法，最常用的是使用 non-blocking I/O 呼叫來發出所有的檔案存取。之後，應用程式可以使用事件或回呼來判斷個別的呼叫何時完成。如果 I/O 數目很多，大約四到八個，藉由同時發出所有的 I/O，擷取全部資料的總次數可以減半。

在 Windows* 使用非同步 I/O

在 Windows* 應用程式中，存取稱為 ReadFile 和 WriteFile 的檔案所使用的主要函數有兩項。在一般應用程式中，這些函數是以 blocking 或同步方式來使用。以下範例顯示使用 ReadFile 從檔案讀取 1KB：

```
bStatus = ReadFile(  
    hFile,                // Handle to file to read from  
    pBuffer,              // Pointer to buffer to place data in  
    1024,                 // Want to read 1024 bytes from the file  
    &numBytesRead,        // Number of bytes read from the file  
    NULL);                // Synchronous so overlapped parameter is NULL  
  
//  
// Code for checking the status value and ensuring there were not any errors.  
//  
...
```

進行這項呼叫時，在從檔案讀取所有資料之前，它不會回傳。此時，應用程式無法執行任何有用的作業，也無法在這個執行緒內發出任何更多的 I/O 呼叫。

從檔案讀取資料的常用方法是使用旗標 FILE_FLAG_OVERLAPPED 開啟檔案，藉此以非同步方式使用 ReadFile 和 WriteFile。以下使用非同步機制來顯示相同的範例。

```
//  
// Fill in the OVERLAPPED structure used for asynchronous IO.  
//  
overlap.offset = 0;      // Offset in the file to start reading from  
overlap.offsetHigh = 0;  // Upper 32-bits of offset  
overlap.hEvent = hEvent; // Event to trigger when complete, initialized  
                        // using CreateEvent()  
  
//  
// Make the call to start reading the file.  
//  
bStatus = ReadFile(  
    hFile,                // Handle to file to read from  
    pBuffer,              // Pointer to buffer to place data in  
    1024,                 // Want to read 1024 bytes from the file  
    &numBytesRead,        // Number of bytes read from the file
```

Serial ATA 原生指令排序

```
&overlap);          // Contains event and offset for asynchronous
                    // operation

//
// Make sure there was not an error.
//
if ((ERROR_SUCCESS != bStatus) || (ERROR_IO_PENDING != bStatus))
{
    fprintf(stderr, "ReadFile call failed.\n");
    ExitProcess();
}

//
// Make more IO calls or do other useful work.
//
...

//
// Check that the IO has been completed. Note that multiple completions can be
// checked for at the same time.
//
dwResult = WaitForMultipleObjects(
    n,                // Number of IOs issued
    handleArray,      // Array of handles to the events in the
                    // overlapped structure for each IO
    TRUE,             // Wait for all IOs to be completed, could
                    // wait for first to complete and then act on
                    // that data, then check for more
    50);              // Wait up to 50 milliseconds for completion,
                    // could be infinity.

//
// Check the value of dwResult and also call GetOverlappedResult to ensure
// that each IO that completed was with good status.
//
...
```

從這個範例可以看出，非同步 I/O 涉及更多程式碼，而且有點麻煩，但是結合佇列作業和非同步 I/O 所產生的效能潛力，就算多寫幾行程式碼也值得。

結論

原生指令排序有潛力提供重大的效能優勢，在硬碟機中建置指令佇列時，可以實現 NCQ 的好處，因此，硬碟機可以將用最好的方式將指令重新排序，以縮短搜尋和旋轉等待時間。NCQ 透過 Serial ATA 通訊協定中的特性提供有效率的解決方案，其中包括 race-free status return、interrupt aggregation 和 First Party DMA。

Serial ATA 原生指令排序

只有在硬碟機中建置佇列，才可以落實 NCQ 效能優勢，因此，應用程式和作業系統必須盡可能使用非同步 I/O，並且讓硬碟機一次保有多重指令的佇列。在 2003 年底和 2004 年初，業界開始廣泛運用 NCQ，因此，建議獨立軟體供應商 (ISV) 和作業系統供應商立即利用非同步 I/O，以充分利用 NCQ 的好處。

作者

Amber Huffman，Intel 主要設計師

Amber Huffman 是 Intel 公司研發事業群主要設計師，主要負責儲存效能和架構。Amber 目前從事之專案側重於定義和開發 Serial ATA II 先進的特性以及尖端的進階主機控制卡介面 (Advanced Host Controller Interface, AHCI) 定義。Amber 擁有密西根大學電腦工程學位，在 Intel 已任職 6 年。

Joni Clark，Seagate 產品行銷經理

Joni Clark 是 Seagate 桌上型電腦介面產品行銷經理，主要負責對系統建置商乃至於一般使用者的 Serial ATA 技術行銷，目前從事 Serial ATA 推廣和串列式介面技術定位的銷售與行銷訓練。她也是業界 Serial ATA Working Group 行銷團隊主席，負責領導推動 Serial ATA 介面。

除外聲明

Copyright © 2003, Intel Corporation and Seagate Technology LLC. 版權所有。Intel 標誌為 Intel Corporation 或其子公司在美國及其他國家的商標或註冊商標。Seagate、Seagate Technology 與 Wave 標誌為 Seagate Technology, LLC 的註冊商標或商標。其他產品名稱為其擁有者的註冊商標或商標。

Serial ATA 原生指令排序