(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2014-154155
(P2014-154155A)
(43)公開日　平成26年8月25日(2014.8.25)

(51)Int.Cl.		F I		テーマコード（参考）
G06F 3/06	(2006.01)	G06F	3/06	301F
G06F 13/10	(2006.01)	G06F	13/10	340A
G06F 3/08	(2006.01)	G06F	3/08	H
G06F 13/12	(2006.01)	G06F	3/06	540
		G06F	13/12	340B

審査請求 未請求　請求項の数 19　ＯＬ　外国語出願　（全 36 頁）　最終頁に続く

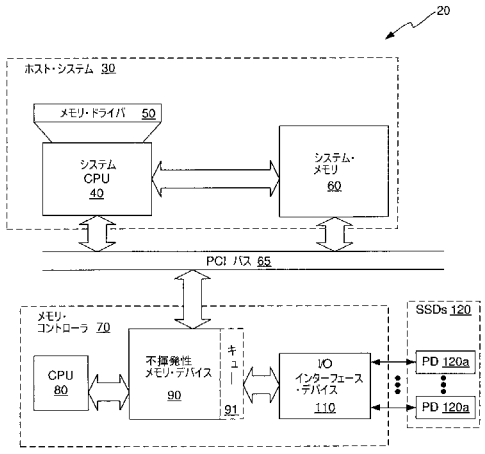
(21)出願番号　特願2014-18979(P2014-18979)
(22)出願日　平成26年2月4日(2014.2.4)
(31)優先権主張番号　13/758,853
(32)優先日　平成25年2月4日(2013.2.4)
(33)優先権主張国　米国(US)

(71)出願人　508243639
エルエスアイ　コーポレーション
アメリカ合衆国カリフォルニア州９５１３１，サンノゼ，リッダー・パーク・ドライヴ　１３２０
(74)代理人　100094112
弁理士　岡部　讓
(74)代理人　100106183
弁理士　吉澤　弘司
(74)代理人　100170601
弁理士　川崎　孝
(74)代理人　100187964
弁理士　新井　剛

最終頁に続く

(54)【発明の名称】コマンド・プッシュ・モデルの使用によりデータ・ストレージ・システムにおける書込みレイテンシを低減させるための方法およびシステム

(57)【要約】
【課題】レイテンシを低減させるコマンド・プッシュ・モデルを実施するデータ・ストレージ・システムを提供すること。
【解決手段】本ホスト・システムは、同システムがＮＶＭ装置の中に位置するコマンド・キューにコマンドをプッシュすることを可能にするために、メモリ制御部の不揮発性メモリ（ＮＶＭ）装置に対してアクセスを有する。ホスト・システムは、メモリ制御部からの介入を必要とせずに各ＩＯを完了し、それによってホスト・システムとメモリ制御部との間の同期化又は初期接続手順の必要が無くなる。書込コマンドでは、ホスト・システムは、書込コマンドがメモリ制御部の待ち行列の中にプッシュされる時点では書込みコマンドが完了しているとみなすので、メモリ制御部は、コマンドの完了のすぐ後に完了割込みをホスト・システムに対して発行する必要はない。これらの特徴を組み合わせることにより、全体のレイテンシが大きく低減されることになる。
【選択図】図２



【特許請求の範囲】
【請求項１】
システム・プロセッサと、システム・メモリ・デバイスと、を備えるホスト・システムと、
コントローラ・プロセッサと、不揮発性メモリ（ＮＶＭ）デバイスと、入出力（Ｉ／Ｏ）インターフェース・デバイスと、を備えるメモリ・コントローラであって、該ＮＶＭデバイスの一部分はコマンド・キューとして使用される、メモリ・コントローラと、
前記Ｉ／Ｏインターフェース・デバイスに接続された少なくとも１つのソリッド・ステート・ドライブ（ＳＳＤ）であって、該少なくとも１つのＳＳＤは、物理ディスク・ドライブ（ＰＤ）のアレイとして構成されている、少なくとも１つのソリッド・ステート・ドライブ（ＳＳＤ）と、
前記ホスト・システムを前記メモリ・コントローラと相互接続するバスであって、前記ホスト・システムは、該バスを経由して前記ＮＶＭデバイスにアクセスし、前記バスを経由して前記ＮＶＭデバイスの前記コマンド・キューの中にコマンドをプッシュする、バスと、
を備える、データ・ストレージ・システム。
【請求項２】
前記システム・プロセッサは、前記バスを経由して前記ＮＶＭデバイスにアクセスし、前記バスを経由して前記ＮＶＭデバイスの前記コマンド・キューの中に前記コマンドをプッシュするメモリ・ドライバ・プログラムを実行する、請求項１に記載のデータ・ストレージ・システム。
【請求項３】
前記メモリ・コントローラは、前記ＮＶＭデバイスに直接アクセスするダイレクト・メモリ・アクセス（ＤＭＡ）エンジンをさらに備え、前記システム・プロセッサは、前記バスを経由して前記ＤＭＡエンジンに対して前記コマンドをプッシュするメモリ・ドライバ・プログラムを実行し、前記ＤＭＡエンジンは、前記ＮＶＭデバイスの前記コマンド・キューに前記コマンドを記憶する、請求項１に記載のデータ・ストレージ・システム。
【請求項４】
前記ホスト・システムは、前記ＮＶＭデバイスの前記コマンド・キューの中にコマンドがプッシュされたときに前記メモリ・コントローラに通知する、請求項１に記載のデータ・ストレージ・システム。
【請求項５】
前記ホスト・システムと前記ＮＶＭデバイスとは、同じコマンドとデータ構造とを使用する、請求項２に記載のデータ・ストレージ・システム。
【請求項６】
前記バスは、周辺相互接続高速（ＰＣＩｅ）バスであり、前記メモリ・ドライバ・プログラムは、前記ＰＣＩｅバスのベース・アドレス・レジスタ（ＢＡＲ）を使用することにより、前記ＮＶＭデバイスにアクセスする、請求項５に記載のデータ・ストレージ・システム。
【請求項７】
前記ＮＶＭデバイスの一部分がコマンド・キューとして使用される、入出力（Ｉ／Ｏ）インターフェース・デバイスと、
前記Ｉ／Ｏインターフェース・デバイスに接続された少なくとも１つのソリッド・ステート・ドライブ（ＳＳＤ）であって、該少なくとも１つのＳＳＤは、物理ディスク・ドライブ（ＰＤ）のアレイとして構成されている、少なくとも１つのソリッド・ステート・ドライブ（ＳＳＤ）と、
コントローラ・プロセッサと、
一部分がコマンド・キューとして割り付けられた、不揮発性メモリ（ＮＶＭ）デバイスであって、ホスト・システムを前記メモリ・コントローラと相互接続するバスを経由して該ホスト・システムによってアクセスされるように構成されており、該ホスト・システム

が該バスを経由して前記ＮＶＭデバイスの前記コマンド・キューの中にコマンドをプッシュすることを可能にするように構成されている、不揮発性メモリ（ＮＶＭ）デバイスと、を備える、メモリ・コントローラ。

【請求項８】

前記ＮＶＭデバイスは、前記ホスト・システムのプロセッサによって実行されるメモリ・ドライバ・プログラムを経由して前記ホスト・システムによってアクセスされ、前記バスは、周辺相互接続高速（ＰＣＩｅ）バスであり、前記ＮＶＭデバイスは、前記ＰＣＩｅバスのベース・アドレス・レジスタ（ＢＡＲ）を通して前記メモリ・ドライバ・プログラムによってアクセス可能である、請求項７に記載のメモリ・コントローラ。

【請求項９】

前記メモリ・コントローラは、

前記バスを経由して前記ホスト・システムと通信し、前記ＮＶＭデバイスと直接に通信するダイレクト・メモリ・アクセス（ＤＭＡ）エンジンをさらに備え、前記ＤＭＡエンジンは、前記バスを経由して前記ホスト・システムによってアクセスされ、前記ＤＭＡエンジンは、前記メモリ・ドライバ・プログラムによって前記ＤＭＡエンジンに対してプッシュされるコマンドを受信し、前記ＮＶＭデバイスの前記コマンド・キューに前記コマンドを記憶する、請求項７に記載のメモリ・コントローラ。

【請求項１０】

前記ＮＶＭデバイスは、前記ホスト・システムと同じコマンドとデータ構造とを使用する、請求項８に記載のメモリ・コントローラ。

【請求項１１】

データ・ストレージ・システムにおいてレイテンシを低減させるための方法であって、

コントローラ・プロセッサと、不揮発性メモリ（ＮＶＭ）デバイスと、物理ディスク・ドライブ（ＰＤ）のアレイとして構成された少なくとも１つのソリッド・ステート・ドライブ（ＳＳＤ）に接続された入出力（Ｉ／Ｏ）インターフェース・デバイスとを備えるメモリ・コントローラにおいて、該ＮＶＭデバイスの一部分をコマンド・キューとして構成すること、

バスを経由して前記メモリ・コントローラと相互接続されたホスト・システムにより、該バスを経由して該メモリ・コントローラの中にコマンドをプッシュすること、および

前記メモリ・コントローラにおいて、前記ＮＶＭデバイスの前記コマンド・キューに前記コマンドを記憶すること、

を含む、方法。

【請求項１２】

前記ホスト・システムのシステム・プロセッサは、前記バスを経由して前記ＮＶＭデバイスにアクセスし、前記バスを経由して前記ＮＶＭデバイスの前記コマンド・キューの中に前記コマンドをプッシュするためのメモリ・ドライバ・プログラムを実行する、請求項１１に記載の方法。

【請求項１３】

前記メモリ・コントローラは、ダイレクト・メモリ・アクセス（ＤＭＡ）エンジンをさらに備え、前記ホスト・システムのシステム・プロセッサは、前記バスを経由して前記ホスト・システムから前記ＤＭＡエンジンにコマンドをプッシュするためのメモリ・ドライバ・プログラムを実行し、前記ＤＭＡエンジンは、前記バスを経由して前記ＮＶＭデバイスの前記コマンド・キューに前記コマンドを保存する、請求項１１に記載の方法。

【請求項１４】

前記ホスト・システムは、前記ＮＶＭデバイスの前記コマンド・キューの中にコマンドがプッシュされたときに前記メモリ・コントローラに通知する、請求項１２に記載の方法。

【請求項１５】

前記ホスト・システムと前記ＮＶＭデバイスとは、同じコマンドとデータ構造とを使用する、請求項１２に記載の方法。

【請求項１６】
前記バスは、周辺相互接続高速（ＰＣＩｅ）バスであり、前記メモリ・ドライバ・プログラムは、前記ＰＣＩｅバスのベース・アドレス・レジスタ（ＢＡＲ）を使用することにより、前記ＮＶＭデバイスにアクセスする、請求項１５に記載の方法。
【請求項１７】
バスを経由してホスト・システムに接続されたメモリ・コントローラの不揮発性メモリ（ＮＶＭ）デバイスのコマンド・キューの中にコマンドをプッシュするための前記ホスト・システムのプロセッサにより実行されるコンピュータ・プログラムをその上に記憶しており、該コンピュータ・プログラムは、
物理ディスク・ドライブ（ＰＤ）のアレイとして構成されたソリッド・ステート・ドライブ（ＳＳＤ）の１つまたは複数のアドレスを読み取り、または書き込むコマンドを受信するための第１のコード部分と、
前記メモリ・コントローラの中に前記コマンドをプッシュするための第２のコード部分と、
を含む、非一時的コンピュータ読取り可能媒体。
【請求項１８】
前記第２のコード部分は、前記ＮＶＭデバイスの前記コマンド・キューの中に前記コマンドをプッシュし、前記コンピュータ・プログラムは、
前記メモリ・コントローラの前記ＮＶＭデバイスの前記コマンド・キューの中に前記コマンドがプッシュされたときに、前記メモリ・コントローラのプロセッサに通知するための第３のコード部分、
をさらに含む、請求項１７に記載の非一時的コンピュータ読取り可能媒体。
【請求項１９】
前記第２のコード部分は、前記メモリ・コントローラのダイレクト・メモリ・アクセス（ＤＭＡ）エンジンの中に前記コマンドをプッシュし、前記ＤＭＡエンジンは、前記ＤＭＡエンジンが前記コマンド・キューに前記コマンドを保存することを可能にするために、前記ＮＶＭデバイスに直接接続され、また前記コンピュータ・プログラムは、
前記メモリ・コントローラの前記ＮＶＭデバイスの前記コマンド・キューの中に前記コマンドがプッシュされたときに前記メモリ・コントローラのプロセッサに通知するための第３のコード部分、
をさらに含む、請求項１７に記載の非一時的コンピュータ読取り可能媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、一般にデータ・ストレージ・システムに関し、より詳細には、データ・ストレージ・システムにおいてコマンド・プッシュ・モデル（ｃｏｍｍａｎｄ－ｐｕｓｈ ｍｏｄｅｌ）を使用して、書込みレイテンシを低減させるための方法およびシステムに関する。
【背景技術】
【０００２】
ストレージ・アレイまたはディスク・アレイは、複数のハード・ディスク・ドライブ（ＨＤＤ：ｈａｒｄ ｄｉｓｋ ｄｒｉｖｅｓ）または類似した永続性のストレージ・ユニットを含むデータ・ストレージ・デバイスである。ストレージ・アレイは、大量のデータを効率的なやり方で記憶することを可能にすることができる。サーバまたはワークステーションは、ストレージ・アレイがサーバまたはワークステーションに対してローカルであるように、ストレージ・アレイと直接に接続されることもある。サーバまたはワークステーションが、ストレージ・アレイに直接に接続される場合には、ストレージ・アレイは、一般的に、ダイレクト・アタッチド・ストレージ（ＤＡＳ：ｄｉｒｅｃｔ－ａｔｔａｃｈｅｄ ｓｔｏｒａｇｅ）システムと称される。代わりに、サーバまたはワークステーションは、ストレージ・アレイ・ネットワーク（ＳＡＮ：ｓｔｏｒａｇｅ ａｒｒａｙ ｎｅ

ｔｗｏｒｋ）を経由してストレージ・アレイにリモートに取り付けられることもある。ＳＡＮシステムにおいては、ストレージ・アレイは、サーバまたはワークステーションに対してローカルではないが、アレイのディスク・ドライブは、サーバまたはワークステーションのオペレーティング・システム（ＯＳ：ｏｐｅｒａｔｉｎｇ　ｓｙｓｔｅｍ）に対して、ローカルに取り付けられているように見える。

【０００３】

図１は、コマンド・プル・モデル（ｃｏｍｍａｎｄ－ｐｕｌｌ　ｍｏｄｅｌ）を実施する典型的なデータ・ストレージ・システム２のブロック図を示すものである。システム２は、ホスト・システム３と、メモリ・コントローラ４と、周辺相互接続（ＰＣＩ：ｐｅｒｉｐｈｅｒａｌ　ｉｎｔｅｒｃｏｎｎｅｃｔ）バスまたはＰＣＩ高速（ＰＣＩｅ）バス５とを含む。コントローラ４は、中央演算処理装置（ＣＰＵ：ｃｅｎｔｒａｌ　ｐｒｏｃｅｓｓｉｎｇ　ｕｎｉｔ）６と、メモリ・デバイス７と、Ｉ／Ｏインターフェース・デバイス８とを含む。Ｉ／Ｏインターフェース・デバイス８は、シリアル・アタッチドＳＣＳＩ（ＳＡＳ：Ｓｅｒｉａｌ　Ａｔｔａｃｈｅｄ　ＳＣＳＩ）規格、シリアル・アドバンスド・テクノロジー・アタッチメント（ＳＡＴＡ：Ｓｅｒｉａｌ　Ａｄｖａｎｃｅｄ　Ｔｅｃｈｎｏｌｏｇｙ　Ａｔｔａｃｈｍｅｎｔ）規格、不揮発性メモリ・ホスト・コントローラ・インターフェース高速（ＮＶＭｅ：Ｎｏｎｖｏｌａｔｉｌｅ　Ｍｅｍｏｒｙ　Ｈｏｓｔ　Ｃｏｎｔｒｏｌｌｅｒ　Ｉｎｔｅｒｆａｃｅ　Ｅｘｐｒｅｓｓ）規格など、知られているデータ転送プロトコル規格に準拠してデータ転送を実行するように構成されている。Ｉ／Ｏインターフェース・デバイス８は、複数の物理ディスク（ＰＤ：ｐｈｙｓｉｃａｌ　ｄｉｓｋｓ）９へのデータの転送、及び複数の物理ディスク９からのデータの転送を制御する。メモリ・コントローラ４は、ＰＣＩバス５を経由してシステムＣＰＵ　１１、およびシステム・メモリ・デバイス１２と通信する。システム・メモリ・デバイス１２は、システムＣＰＵ　１１により実行されるソフトウェア・プログラムと、データとを記憶する。システム・メモリ・デバイス１２の一部分は、コマンド・キュー１３として使用される。

【０００４】

典型的な書込みアクション中に、システムＣＰＵ　１１は、コマンドとデータとをコマンド・キュー１３に保存するメモリ・ドライバ・ソフトウェア・スタック１４を実行する。メモリ・ドライバ１４がコマンド・キュー１３にコマンドを保存するときに、メモリ・ドライバ１４は、コマンドが実行されるための準備ができていることをメモリ・コントローラ４に通知する。コントローラＣＰＵ　６がコマンドを実行するための準備ができているときに、コントローラＣＰＵ　６は、１つまたは複数のコマンドと、関連するデータとをバス５を経由してシステム・キュー１３からプルし、またホスト・システム３に対して完了割込み（ｃｏｍｐｌｅｔｉｏｎ　ｉｎｔｅｒｒｕｐｔ）を発行する。それらのコマンドが、メモリ・コントローラ４によって実行されるときに、コントローラＣＰＵ　６は、コマンドに関連するデータが、コントローラ・メモリ・デバイス７に一時的に記憶され、また次いでその後にＩ／Ｏインターフェース・デバイス８を経由して１つまたは複数のＰＤ　９に書き込まれるようにさせる。

【０００５】

従来、図１に示されるタイプのＨＤＤ－ベースのシステムの性能は、秒当たりの入出力（ＩＯ）（ＩＯＰＳ）と、いくつかの場合には、秒当たりのメガバイト（ＭＢ／ｓ）との観点から測定されてきている。そのようなストレージ・システムのレイテンシは、一般的に、レイテンシ＿全体＝レイテンシ＿ＳＷ＿スタック＋レイテンシ＿コントローラ＋レイテンシ＿ＨＤＤ＝１／ＩＯＰＳとして与えられ、ここでレイテンシ＿全体は、システムの全体のレイテンシであり、レイテンシ＿ＳＷ＿スタックは、メモリ・ドライバ１４に関連するレイテンシであり、レイテンシ＿コントローラは、メモリ・コントローラ４に関連するレイテンシであり、またレイテンシ＿ＨＤＤは、ＰＤ　９に関連するレイテンシである。レイテンシ＿ＳＷ＿スタックは、一般的に、およそマイクロ秒（10^{-6}秒）の程度である。同様に、レイテンシ＿コントローラは、一般的に、およそ１０マイクロ秒の程度で

ある。しかしながら、レイテンシ＿ＨＤＤは、一般的に、およそミリ秒（10^{-3}秒）または１０ミリ秒の程度である。全体のレイテンシのほぼ９９％は、それらのＨＤＤの極端に低速な機械部品に起因している。それゆえに、実用的な目的のためには、レイテンシ＿ＳＷ＿スタックと、レイテンシ＿コントローラとは、システム性能を決定するときに無視することができる。言い換えれば、システム性能は、レイテンシ＿ＨＤＤに等しいものとして推定することができる。

【０００６】

最近になって、ＰＤ　９として磁気ＨＤＤを使用することから、ＰＤ　９としてソリッド・ステート・ドライブ（ＳＳＤ：ｓｏｌｉｄ　ｓｔａｔｅ　ｄｒｉｖｅｓ）またはＳＳＤとＨＤＤとの組合せを使用することへの移り変わりが起こってきている。産業界においては、ＳＳＤ－ベースのソリューションの使用は、ＨＤＤ－ベースのソリューションの進化と見なされる。しかしながら、ＳＳＤ－ベースのソリューションは、ＨＤＤ－ベースのソリューションよりも約１００倍高速であり、またＨＤＤ－ベースのソリューションよりもずっと低い電力しか消費しない。ＳＳＤ－ベースのソリューションにおけるこの観点は、ＳＳＤ－ベースのソリューションにおいても、コマンド・キューからメモリ・コントローラにコマンドをプルするために、以前から存在する上記で説明されたプル手法（ｐｕｌｌ　ｍｅｔｈｏｄｏｌｏｇｙ）を使用し続けるように産業界を導いてきている。また、ＳＳＤ－ベースのソリューションは、産業界において単にＨＤＤ－ベースのソリューションの進化にすぎないものとして見なされているので、ＩＯＰＳは、ＳＳＤ－ベースのソリューションを実施するストレージ・システムにおいてシステム性能を測定するための性能測定基準として使用されてきている。

【０００７】

しかしながら、ＳＳＤ－ベースのソリューションと、ＨＤＤ－ベースのソリューションとの間の違いは、それが目に見えるよりもずっと大きく、また伝統的な測定基準を使用して、ＳＳＤ－ベースのソリューションを実施するシステムの性能を測定するべきではない。ＳＳＤ－ベースのソリューションを実施するシステムにおいては、ストレージ・システムの全体のレイテンシは、レイテンシ＿全体＝レイテンシ＿ＳＷ＿スタック＋レイテンシ＿コントローラ＋レイテンシ＿ＳＳＤ＝１／ＩＯＰＳとして与えられ、ここでレイテンシ＿ＳＷ＿スタックは、メモリ・ドライバ１４に関連するレイテンシであり、レイテンシ＿コントローラは、メモリ・コントローラ４に関連するレイテンシであり、またレイテンシ＿ＳＳＤは、ＰＤ　９として使用されるＳＳＤに関連するレイテンシである。ＨＤＤのレイテンシとは違って、ＳＳＤのレイテンシは、およそ１０から１００マイクロ秒の程度にあり、例えば、一般に１００から３００マイクロ秒範囲にあり、またメモリ・ドライバ１４と、メモリ・コントローラ４とに関連するレイテンシは、全体のレイテンシに対してそれよりもずっと多くのレイテンシを追加する。したがって、ＳＳＤ－ベースのソリューションを実施するストレージ・システムの全体のレイテンシを算出する際に、レイテンシ＿ＳＷ＿スタックと、レイテンシ＿コントローラとは、もはや無視されるべきではない。

【０００８】

コマンド・プル・アプローチは、メモリ・ドライバ１４と、メモリ・コントローラ４との間にかなりの相互作用を必要とする。これは、それがホスト・システム３の高速なオペレーティング・システム（ＯＳ）側が、より低速のＨＤＤ－ベースのコントローラ側とはほとんど完全に独立していることを可能にし、その結果、ＯＳ側は、より大きなキューの深さ（ＱＤ：ｑｕｅｕｅ　ｄｅｐｔｈ）を提供するためにキュー１３の中でできるだけ多くのコマンドを積み重ねることができるようになり、このより大きなキューの深さは、ＨＤＤ－ベースのソリューションにおいては非常に望ましく、また一般的であるという点で、ＨＤＤ－ベースのシステムにおいては便利なアプローチである。次いで、メモリ・コントローラ４は、それ自体のペースで、キュー１３からコマンドをプルすることができる。この方法は、ＨＤＤ－ベースのソリューションにおいては非常に便利である一方、それは、メモリ・コントローラ４と、ホスト・システム３との間で必要とされる同期化に起因して、及び、メモリ・コントローラ４が、それらが発行されたときよりもずっと後の時点で

コマンドをピックアップすることができるという事実に起因して、大量の余分なレイテンシを加える。さらに、多くの場合にそうであるように、メモリ・コントローラ４によって行われるべきたくさんの作業がある場合には、コマンド処理のすべては、メモリ・コントローラ４の作業負荷の残りと必ず競合することになり、これは、全体のレイテンシに対して、より多くのレイテンシを追加する。

【０００９】

上記で説明されたコマンド・プル・モデルは、ＨＤＤ－ベースのソリューションの場合に非常によく機能し、ここでは、完了するために一般的に約１０，０００マイクロ秒かかり得るコマンドに対して５０から５００マイクロ秒を追加することは、本方法が提供する他の利点を仮定すると、無視できる。しかしながら、コマンド・プル・モデルは、アクセス・タイムが、１００マイクロ秒ほどの短い時間であり得るＳＳＤ－ベースのソリューションを実施するストレージ・システムの中で使用されるときに、または１マイクロ秒から５マイクロ秒ほどであるダイナミック・ランダム・アクセス・メモリ（ＤＲＡＭ：ｄｙｎａｍｉｃ　ｒａｎｄｏｍ　ａｃｃｅｓｓ　ｍｅｍｏｒｙ）ライト・バック（ＷＢ：ｗｒｉｔｅ　ｂａｃｋ）バッファがメモリ・コントローラ４の中で使用される場合には、許容可能な結果を生ずることはない。

【００１０】

それにもかかわらず、上記で示されるように、ＳＳＤ－ベースのソリューションを実施するストレージ・システムの全体のレイテンシは、レイテンシ＿ＳＷ＿スタックと、レイテンシ＿コントローラとを推定から除外しながら、依然として、レイテンシ＿ＳＳＤに等しいものとして推定されている。この理由のためにＳＳＤ－ベースのソリューションを実施するストレージ・システムの全体のレイテンシを低減させることにおける試みは、レイテンシ＿ＳＷ＿スタックまたはレイテンシ＿コントローラを低減させることではなく、レイテンシ＿ＳＳＤを低減させることに主として焦点を当てている。

【００１１】

レイテンシ＿ＳＷ＿スタックと、レイテンシ＿コントローラとのうちの一方または両方を低減させることにより、ＳＳＤ－ベースのソリューションを実施し、そして全体のレイテンシを低減させるストレージ・システムについての必要性が存在する。

【発明の概要】

【課題を解決するための手段】

【００１２】

本発明は、データ・ストレージ・システムと、メモリ・コントローラと、方法と、コンピュータ読取り可能媒体とを提供しており、これらのすべては、レイテンシを低減させるコマンド・プッシュ・モデルを実施する。データ・ストレージ・システムは、ホスト・システムと、メモリ・コントローラと、少なくとも１つのＳＳＤと、ホスト・システムとメモリ・コントローラとを相互接続するバスとを備える。ホスト・システムは、システム・プロセッサと、システム・メモリ・デバイスとを備える。メモリ・コントローラは、コントローラ・プロセッサと、不揮発性メモリ（ＮＶＭ：ｎｏｎｖｏｌａｔｉｌｅ　ｍｅｍｏｒｙ）デバイスと、入出力（Ｉ／Ｏ：ｉｎｐｕｔ／ｏｕｔｐｕｔ）インターフェース・デバイスとを備える。ＮＶＭデバイスの一部分は、コマンド・キューとして使用される。ＳＳＤは、Ｉ／Ｏインターフェース・デバイスに接続され、またＰＤのアレイとして構成されている。ホスト・システムは、バスを経由してＮＶＭデバイスにアクセスし、バスを経由してＮＶＭデバイスのコマンド・キューの中に、コマンドをプッシュする。

【００１３】

本方法は、

コントローラ・プロセッサと、ＮＶＭデバイスと、ＰＤのアレイとして構成された少なくとも１つのＳＳＤに接続されたＩ／Ｏインターフェース・デバイスとを備えるメモリ・コントローラにおいて、ＮＶＭデバイスの一部分をコマンド・キューとして構成すること、

バスを経由してメモリ・コントローラと相互接続されたホスト・システムにより、バス

を経由してメモリ・コントローラの中にコマンドをプッシュすること、および
メモリ・コントローラにおいて、ＮＶＭデバイスのコマンド・キューにコマンドを記憶すること、
を含む。
【００１４】
コンピュータ読取り可能媒体は、バスを経由してホスト・システムに接続されるメモリ・コントローラのＮＶＭデバイスのコマンド・キューの中に、コマンドをプッシュするためのホスト・システムのプロセッサにより実行されるコンピュータ・プログラムをその上に記憶している。コンピュータ・プログラムは、第１のコード部分と、第２のコード部分とを含む。第１のコード部分は、ＰＤのアレイとして構成されるＳＳＤの１つまたは複数のアドレスを読み取り、または書き込むコマンドを受信する。第２のコード部分は、メモリ・コントローラの中に、コマンドをプッシュする。実例となる一実施形態に従って、第２のコード部分は、メモリ・コントローラのＮＶＭデバイスのコマンド・キューの中に、コマンドをプッシュする。別の実例となる実施形態に従って、第２のコード部分は、メモリ・コントローラのＮＶＭデバイスのコマンド・キューにコマンドを記憶するメモリ・コントローラのダイレクト・メモリ・アクセス（ＤＭＡ：ｄｉｒｅｃｔ　ｍｅｍｏｒｙ　ａｃｃｅｓｓ）エンジンの中に、コマンドをプッシュする。
【００１５】
本発明のこれらおよび他の特徴および利点は、以下の説明と、図面と、特許請求の範囲とから明らかになるであろう。
【図面の簡単な説明】
【００１６】
【図１】コマンド・プル・モデルを実施する、既知のデータ・ストレージ・システムのブロック図である。
【図２】コマンド・プッシュ・モデルを実施する実例となる一実施形態によるデータ・ストレージ・システムのブロック図である。
【図３】コマンド・プッシュ・モデルを実施する別の実例となる実施形態によるデータ・ストレージ・システムのブロック図である。
【図４】図２に示されるデータ・ストレージ・システムによって実行されるコマンド・プッシュ方法を表すフローチャートである。
【図５】図３に示されるデータ・ストレージ・システムによって実行されるコマンド・プッシュ方法を表すフローチャートである。
【発明を実施するための形態】
【００１７】
本発明に従って、メモリ・ドライバと、メモリ・コントローラと、に関連するレイテンシを低減させるコマンド・プッシュ・モデルを実施する、データ・ストレージ・システムが提供される。コマンド・プッシュ・モデルは、書込みコマンドＩＯのための、ホスト・システムと、メモリ・コントローラとの間の同期化についての必要性をなくし、それによって同期化に関連するレイテンシをなくす。コマンドＩＯは、ホスト・システムにおいて発行されるので、ホスト・システムは、メモリ・コントローラのメモリ・デバイスにアクセスし、メモリ・デバイスのコマンド・キューの中に、コマンドＩＯと、関連するデータとをプッシュする。それゆえに、ホスト・システムは、メモリ・コントローラからのどのような介入もなしに各書込みコマンドＩＯを完了し、それによってホスト・システムとメモリ・コントローラとの間の同期化、または初期接続手順（ｈａｎｄｓｈａｋｉｎｇ）についての必要性を取り除いている。
【００１８】
メモリ・コントローラのメモリ・デバイスは、電源障害と、典型的なメモリ・エラーとに対して保護され、したがって、永続性ストレージ要素と考えられることもある。それゆえに、ホスト・システムは、ホスト・システムがメモリ・コントローラのメモリ・デバイスのコマンド・キューの中に、書込みコマンドと、任意の関連するデータとをプッシュす

る時間には、書込みコマンドＩＯが安全に完了されていると考えることができる。言い換えれば、ホスト・システムの観点から、ホスト・システムが、メモリ・コントローラのメモリ・デバイスのコマンド・キューの中に、書込みコマンドをプッシュするときに、それは、まるでホスト・システムがＰＤ　９に対して書込みを完了しているかのようである。それゆえに、ホスト・システムは、メモリ・コントローラからのどのような介入もなしに各書込みコマンドＩＯを完了し、それによってホスト・システムと、メモリ・コントローラとの間の同期化、または初期接続手順についての必要性を取り除いている。

【００１９】

さらに、図１を参照して上記で説明されたコマンド・プル・モデルとは対照的に、ホスト・システムが、それがメモリ・コントローラのコマンド・キューの中に書込みコマンドをプッシュする時点で書込みコマンドが完了していると考えるので、本発明のメモリ・コントローラは、ひとたびメモリ・コントローラが書込みコマンドの処理を完了した後に、ホスト・システムに対して完了割込みを発行する必要はない。この特徴により、ホスト・システムと、メモリ・コントローラとは、書込みコマンドに関して互いに独立して動作することができるようになる。これらの特徴のすべての組合せは、レイテンシ＿ＳＷ＿スタックと、レイテンシ＿コントローラとの両方に対する大きな低減をもたらし、これは、全体のレイテンシにおける大きな低減をもたらす。

【００２０】

したがって、コマンドが、システム・メモリ１２のコマンド・キュー１３において用意され、また後でメモリ・コントローラ４の中にプルされ、またメモリ・コントローラ４によって処理される、図１を参照して上記で説明したコマンド・プル・モデルを使用するのではなくて、ホスト・システムは、メモリ・コントローラのメモリ・デバイス内に位置するコマンド・キューの中に、コマンドと、任意の関連するデータとをプッシュする。ホスト・システムが、メモリ・コントローラのメモリ・デバイスのコマンド・キューの中に、コマンドと、任意の関連するデータとをプッシュする方法は、いくつかのやり方で行われる可能性がある。第１の実例となる実施形態によると、ホスト・システムのメモリ・デバイスは、メモリ・コントローラのメモリ・デバイスのコマンド・キューの中に、コマンドと、関連するデータとをプログラムでプッシュする。本明細書において説明される第２の実例となる実施形態によると、メモリ・コントローラのダイレクト・メモリ・アクセス（ＤＭＡ）エンジンは、メモリ・コントローラのメモリ・デバイスのコマンド・キューの中に、コマンドと、任意の関連するデータとをプッシュする。これらの実施形態の両方は、ホスト・システムと、メモリ・コントローラとの間の同期化、または初期接続手順についての必要性を取り除いている。

【００２１】

図２は、コマンド・プッシュ・モデルを実施する第１の実例となる実施形態によるデータ・ストレージ・システム２０のブロック図を示すものである。システム２０は、ホスト・システム３０と、メモリ・コントローラ７０と、周辺相互接続（ＰＣＩ）またはＰＣＩ高速（ＰＣＩｅ）バス６５と、ＰＤ　１２０ａを備える少なくとも１つのＳＳＤ　１２０とを含む。ホスト・システム３０は、システムＣＰＵ　４０と、システム・メモリ・デバイス６０とを含んでいる。メモリ・コントローラ７０は、ＣＰＵ　８０と、不揮発性メモリ（ＮＶＭ）デバイス９０と、Ｉ／Ｏインターフェース・デバイス１１０とを含む。ＮＶＭデバイス９０は、一般的にＤＲＡＭデバイスである。ＮＶＭデバイス９０の一部分は、コマンド・キュー９１として使用される。Ｉ／Ｏインターフェース・デバイス１１０は、ＳＡＳ規格、ＳＡＴＡ規格、および／またはＮＶＭｅ規格など、知られているデータ転送プロトコル規格に準拠してデータ転送を実行するように構成されている。Ｉ／Ｏインターフェース・デバイス１１０は、ＳＳＤ（単数または複数）１２０の複数のＰＤ　１２０ａへのデータの転送と、ＳＳＤ（単数または複数）１２０の複数のＰＤ　１２０ａからのデータの転送とを制御する。メモリ・コントローラ７０は、ＰＣＩバス６５と、メモリ・ドライバ５０とを経由してシステムＣＰＵ　４０と通信する。システム・メモリ・デバイス６０は、システムＣＰＵ　４０により実行されるソフトウェア・プログラムと、データと

を記憶する。

【００２２】

この実例となる実施形態に従って、メモリ・ドライバ５０は、バス６５を経由してＮＶＭデバイス９０と直接に通信する。典型的な書込みアクション中に、システムＣＰＵ　４０は、メモリ・ドライバ５０を含むソフトウェア・スタックを実行し、このメモリ・ドライバは、ＮＶＭデバイス９０のコマンド・キュー９１の中に、コマンドと、データとをプッシュし、またそのコマンドと、データとが、キュー９１の中にあることをコントローラＣＰＵ　８０に対して通知する。ホスト・システム３０と、メモリ・コントローラ７０との間の同期化、または初期接続手順は、プッシュ・アクションを実行するためには必要とされない。ひとたびメモリ・ドライバ５０が、コマンド・キュー９１の中に、書込みコマンドと、関連するデータとをプッシュした後に、ホスト・システム３０のＯＳは、書込みコマンドが完了しているとみなす。それゆえに、メモリ・コントローラ７０が、ホスト・システム３０に対して完了割込みを発行して、書込みコマンドが完了されていることをホストＣＰＵ　４０に通知することは、必要ではない。

【００２３】

コントローラＣＰＵ　８０がコマンドを実行する用意ができているときに、コントローラＣＰＵ　８０は、ＮＶＭデバイス９０からコマンドとデータとを取り出し、またコマンドを実行する。書込みコマンドの場合には、コントローラＣＰＵ　８０は、関連するデータが、Ｉ／Ｏインターフェース・デバイス１１０を経由してＳＳＤ　ＰＤ　１２０ａに対して書き込まれるようにさせる。読取りコマンドの場合には、コントローラＣＰＵ　８０は、ＮＶＭ　９０のキャッシュ（明確にするために示されていない）をチェックして、読み取られるべきデータがキャッシュの中にあるかどうかを決定し、またそうである場合には、キャッシュからデータを読み取り、またそれをホスト・システム３０に対して転送する。読み取られるべきデータが、キャッシュの中にない場合には、ＣＰＵ　８０は、ＳＳＤ　ＰＤ　１２０ａからデータが読み取られ、そしてホスト・システム３０に対して転送されるようにさせる。

【００２４】

図２のコマンド・プッシュ・モデルと、図１のコマンド・プル・モデルとの間の重要な違いのうちの１つは、コントローラＣＰＵ　８０がコマンドについて知りさえする前に、コマンド所有権が、ホスト・システム３０からＮＶＭデバイス９０のキュー９１へと転送されることである。図１のコマンド・プル・モデルにおいては、書込みコマンドに関連するコマンド所有権は、ホスト・システム３が、メモリ・コントローラ４から完了割込みを受信するまで、転送されない。さらに、図１のコマンド・プル・モデルにおいては、メモリ・コントローラ４の中に、システム・メモリ・デバイス１２のキュー１３からのコマンドをプルするために、コントローラＣＰＵ　６と、ホスト・システム３との間で、同期化、または初期接続手順が発生する。これらのレイテンシを取り除くことは、一般的に、図１に示されるコマンド・プル・モデルと比べて１桁から２桁の大きさの書込みレイテンシにおける低減をもたらす。

【００２５】

ひとたびそれらのコマンドがＮＶＭ　９０のキュー９１の中にプッシュされた後に、メモリ・コントローラ７０がコマンドの所有権を取得するので、ＮＶＭ　９０が電源障害に対して保護されることが重要である。そのような保護は、いくつかのやり方で達成される可能性があるが、１つのやり方は、ＮＶＭ　９０についてバッテリ・バックアップ（ＢＢＵ：ｂａｔｔｅｒｙ　ｂａｃｋｕｐ）のスーパー・キャパシタ（Ｓｕｐｅｒｃａｐ）ダイナミック・ランダム・アクセス・メモリ（ＤＲＡＭ）デバイスを使用することである。ＢＢＵのＳｕｐｅｒｃａｐ　ＤＲＡＭデバイスまたは他の適切なメモリ・デバイスが、この目的のために選択され得る方法は、当業者によって理解されるであろう。

【００２６】

メモリ・ドライバ５０が、キュー９１の中に、コマンドを効率的にプッシュするために、ＮＶＭデバイス９０は、バス６５を経由してメモリ・ドライバ５０によって直接にアク

セス可能であるべきである。このアクセス可能性は、いくつかのやり方で達成されることもある。これを達成すべき１つのやり方は、ホスト・システム３０において使用されるコマンド及びデータ構造と同一のコマンド及びデータ構造を、ＮＶＭデバイス９０においても使用することである。これは、当業者によって理解されることになるように、ベース・アドレス・レジスタ（ＢＡＲ：Ｂａｓｅ　Ａｄｄｒｅｓｓ　Ｒｅｇｉｓｔｒｅｒｓ）の便利なプログラミングを通してＰＣＩｅにおいて可能である。ホスト・システム３からＮＶＭデバイス９０へのデータ転送は、メモリ・ドライバ５０によってプログラムで達成される可能性がある。

【００２７】

メモリ・ドライバ５０はまた、ＮＶＭデバイス９０におけるどの入力がコマンド情報を用いてそれらをマッピングし／タグ付けするために使用可能であるかを決定する方法とともに提供されるべきである。各コマンドは、データ・セットと、コマンド情報とから構成されている。コマンド情報は、読取り／書込み、ブロック・アドレス、ポインタ、フラグなどの情報を含んでいる。コマンド情報は、コマンド・タイプによって異なり、またデータ・セットとは独立に生成される。データ・セットと、コマンド情報との両方は、ＮＶＭ９０の中に移動される必要があり、またデータ・セットは、コマンド情報と、データ・セットとの間の整合性を保証するために、対応するコマンド情報を用いてマッピングされ／タグ付けされる必要がある。これが達成される方法は、使用されるメモリ・コントローラ７０のタイプまたは構成に応じて異なる。すべてのメモリ・コントローラは、この機能を提供するために構成可能であるが、ソリューションの効率は、メモリ・コントローラのハードウェア、ソフトウェアおよび／またはファームウェアの実装形態に依存することになる。それゆえに、この問題は、当業者によって理解されるように、ケース・バイ・ケースで対処される必要があることになる。

【００２８】

図３は、ＤＭＡエンジンを使用して、メモリ・コントローラのメモリ・デバイスのキューの中に、コマンドと、データとをプッシュする、第２の実例となる実施形態によるデータ・ストレージ・システム２００のブロック図を示すものである。ＤＭＡエンジンは、ホスト・システムの一部分、またはメモリ・コントローラの一部分とすることができるが、例証の目的のために、それは、メモリ・コントローラの一部分であるように示され、また説明されている。本システム２００は、ホスト・システム２２０と、メモリ・コントローラ２７０と、ＰＣＩバスまたはＰＣＩｅバス２６５と、ＰＤ　３２０ａを備える少なくとも１つのＳＳＤ　３２０とを含む。ホスト・システム２２０は、システムＣＰＵ　２３０と、システム・メモリ・デバイス２６０とを含む。メモリ・コントローラ２７０は、ＤＭＡエンジン２４０と、ＣＰＵ　２８０と、ＮＶＭデバイス２９０と、Ｉ／Ｏインターフェース・デバイス３１０とを含む。ＮＶＭデバイス２９０は、一般的にＤＲＡＭデバイスであり、またそれは、ＮＶＭ　９０を参照して上記で説明されたものと同じ保護を有する。ＮＶＭデバイス２９０の一部分は、コマンド・キュー２９１として使用される。Ｉ／Ｏインターフェース・デバイス３１０は、ＳＡＳ規格、ＳＡＴＡ規格、および／またはＮＶＭｅ規格など知られているデータ転送プロトコル規格に準拠してデータ転送を実行するように構成されている。Ｉ／Ｏインターフェース・デバイス３１０は、ＳＳＤ（単数または複数）３２０の複数のＰＤ　３２０ａへのデータの転送と、ＳＳＤ（単数または複数）３２０の複数のＰＤ　３２０ａからのデータの転送とを制御する。メモリ・コントローラ２７０は、ＰＣＩバス２６５と、メモリ・ドライバ２５０とを経由してシステムＣＰＵ　２３０と通信する。システム・メモリ・デバイス２６０は、システムＣＰＵにより実行されるソフトウェア・プログラムと、データとを記憶する。

【００２９】

実例となる実施形態に従って、ＤＭＡエンジン２４０は、バス２６５を経由してメモリ・ドライバ２５０と通信し、ＮＶＭデバイス２９０と直接に通信する。典型的な書込みアクション中に、メモリ・ドライバ２５０は、バス２６５を経由してＤＭＡエンジン２４０に対して、コマンドと、任意の関連するデータとをプッシュする。ＤＭＡエンジン２４０

は、ＮＶＭデバイス２９０のコマンド・キュー２９１に、コマンドと、データとを記憶する。ホスト・システム２２０と、メモリ・コントローラ２７０との間の同期化または初期接続手順は、プッシュ・アクションを実行するためには必要とされない。ひとたびＤＭＡエンジン２４０が、コマンド・キュー２９１に書込みコマンドと、関連するデータとを記憶した後に、それは、一般的には割込みを発行することにより、コマンド及びデータがＮＶＭ　２９０の中にあることをコントローラＣＰＵ　２８０に通知することが好ましい。ホスト・システム２２０のＯＳは、コマンド及びデータがＤＭＡエンジン２４０に対して転送された時点で書込みコマンドが完了されたとみなす。それゆえに、メモリ・コントローラ２７０が、ホスト・システム２２０に対して完了割込みを発行して、書込みコマンドが完了していることを通知することは、必要ではない。

【００３０】

コントローラＣＰＵ　２８０がコマンドを実行する用意ができているときに、コントローラＣＰＵ　２８０は、ＮＶＭデバイス２９０からコマンドとデータとを取り出し、またコマンドを実行する。書込みコマンドの場合には、コントローラＣＰＵ　２８０は、関連するデータを、Ｉ／Ｏインターフェース・デバイス３１０を経由してＳＳＤ　ＰＤ　３２０ａに対して書き込ませる。読取りコマンドの場合には、コントローラＣＰＵ　２８０は、ＮＶＭ　２９０のキャッシュ（明確にするために示されていない）をチェックして、読み取られるべきデータがキャッシュの中にあるかどうかを決定し、読み取られるべきデータがキャッシュの中にある場合に、キャッシュからデータを読み取り、またそれをホスト・システム２２０に対して転送する。読み取られるべきデータがキャッシュの中にない場合には、ＣＰＵ　２８０は、ＳＳＤ　ＰＤ　３２０ａからデータが読み取られ、またホスト・システム２２０に対して転送されるようにさせる。

【００３１】

図２を参照して上記で説明されるコマンド・プッシュ・モデルと同様に、図３のコマンド・プッシュ・モデルを用いて、コマンド所有権は、コントローラＣＰＵ　２８０がコマンドの存在を認識する前に、ホスト・システム２２０からＮＶＭデバイス２９０のキュー２９１へと転送される。図２を参照して上記で説明されるコマンド・プッシュ・モデルと同様に、図３のコマンド・プッシュ・モデルを用いて、キュー２９１の中にコマンドをプッシュするために、コントローラＣＰＵ　２８０と、ホスト・システム２２０との間には、同期化または初期接続手順が必要とされることはない。それゆえに、図１を参照して上記で説明されるレイテンシは、取り除かれる。

【００３２】

図４は、図２に示されるホスト・システム３０によって部分的に、またメモリ・コントローラ７０によって部分的に実行される実例となる実施形態によるコマンド・プッシュ方法を表すフローチャートを示すものである。本方法は、単一のコマンドを参照して説明されることになるが、本方法のステップは、複数のコマンドについて同時に実行される可能性もある。システムＣＰＵ　４０は、ブロック４０１によって示されるように、読取りコマンドまたは書込みコマンドを発行する。ホスト・システム３０のメモリ・ドライバ５０は、ＮＶＭデバイス９０のキュー９１の中に、コマンドをプッシュし、またブロック４０２によって示されるように、コマンドがＮＶＭデバイス９０の中にあることをメモリ・コントローラ７０に通知する。

【００３３】

書込みコマンドの場合には、コマンドに関連するデータは、ＮＶＭデバイス９０の中にプッシュされる。その後に、ＮＶＭデバイス９０から読み取られたコマンド及びデータは、ＳＳＤ（単数または複数）１２０のＰＤ　１２０ａに書き込まれる。ＮＶＭデバイス９０の中にプッシュされている読取りコマンドの場合には、メモリ・コントローラ７０は、一般的に、データがＮＶＭデバイス９０のキャッシュに記憶されるかどうかを判定することになる。メモリ・コントローラ７０が、キャッシュにデータが記憶されている（すなわち、キャッシュ・ヒット）と判定する場合、メモリ・コントローラ７０は、キャッシュからデータを読み取り、またそれをホスト・システム３０に対して転送する。メモリ・コン

トローラ７０が、キャッシュにデータが記憶されていない（すなわち、キャッシュ・ミス）と判定する場合、メモリ・コントローラ７０は、ＳＳＤ（単数または複数）１２０のＰＤ　１２０ａからデータを読み取り、またデータをホスト・システム３０に対して転送する。

【００３４】

書込みコマンドが、ＮＶＭデバイス９０のキュー９１の中にプッシュされる場合には、ひとたびメモリ・ドライバ５０がキュー９１の中に書込みコマンドをプッシュすれば、ＩＯは、ホスト・システム３０においては完了していることになる。読取りコマンドが、キュー９１の中にプッシュされる場合には、読取りデータがホスト・システム３０に対して戻されるまで、ＩＯは、ホスト・システム３０において完了していることにはならず、これは、読取りコマンドが、図１に示された既知のストレージ・システム２において処理されるやり方である。本発明の方法およびシステムによって達成されるレイテンシにおける低減は、書込みコマンドに関して最も顕著である。これは、ホスト・システム３０からメモリ・コントローラ７０へと書込みコマンドを転送するための同期化がもはや必要とされないこと、及び、書込みコマンドが完了していると見なされるまで、ホスト・システム３０が完了割込みを待つ必要がもはやないという事実に起因する。

【００３５】

図５は、図３に示されるホスト・システム２２０によって部分的に、またメモリ・コントローラ２７０によって部分的に実行される実例となる実施形態によるコマンド・プッシュ方法を表すフローチャートを示すものである。本方法は、単一コマンドを参照して説明されることになるが、本方法のステップはまた、複数のコマンドについて同時に実行される可能性もある。システムＣＰＵ　２３０は、ブロック５０１によって示されるように、読取りコマンド、または書込みコマンドを発行する。メモリ・コントローラ２７０は、ブロック５０２によって示されるように、ＤＭＡエンジン２４０に対してコマンドをプッシュする。ＤＭＡエンジン２４０は、ブロック５０３によって示されるように、ＮＶＭデバイス２９０のキュー２９１にコマンドを記憶し、またコマンドがＮＶＭデバイス２９０の中にあることをメモリ・コントローラ２７０に通知する。本発明は、コマンドがキュー２９１の中に配置されていることをメモリ・コントローラ２７０に通知する手法によって制限されない。これは、当業者によって理解されることになるように、いくつかのやり方で、例えば、割込みを発行すること、レジスタの中の値をアップデートすることなどで達成される可能性がある。

【００３６】

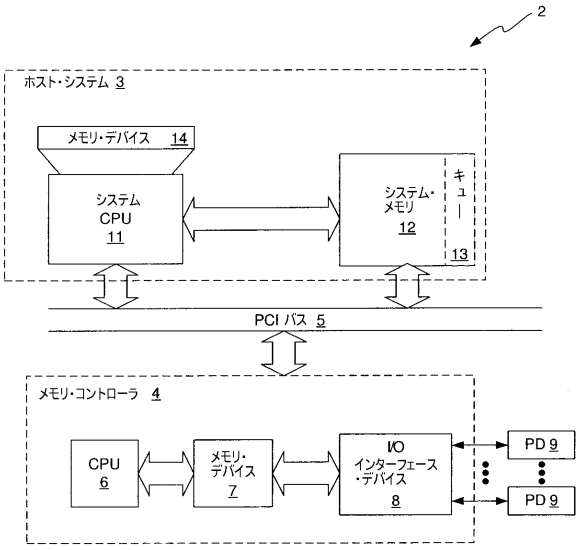
ホスト・システム３０および２２０の機能と、メモリ・コントローラ７０および２７０の機能とは、ハードウェア、ソフトウェア、ファームウェア、またはそれらの組合せの形で実施されることもある。ソフトウェアまたはファームウェアの形で機能を実施するためのコンピュータ・コードは、システム・メモリ・デバイス６０および２６０、ＮＶＭデバイス９０または２９０、何らかの他のメモリ・デバイスなどのコンピュータ読取り可能媒体（ＣＲＭ：ｃｏｍｐｕｔｅｒ－ｒｅａｄａｂｌｅ　ｍｅｄｉｕｍ）の上に記憶される。ＣＲＭは、それだけには限定されないが、磁気ストレージ・デバイスと、ソリッド・ステート・ストレージ・デバイスと、フラッシュ・メモリ・デバイスと、光ストレージ・デバイスとを含めて、任意のタイプのメモリ・デバイスとすることができる。ＣＰＵ　４０、８０、２３０および２８０のうちの各々は、一般的に、少なくとも１つのマイクロプロセッサを備えるが、例えば、マイクロコントローラと、デジタル信号プロセッサ（ＤＳＰ：ｄｉｇｉｔａｌ　ｓｉｇｎａｌ　ｐｒｏｃｅｓｓｏｒ）と、特定用途向け集積回路（ＡＳＩＣ：ａｐｐｌｉｃａｔｉｏｎ　ｓｐｅｃｉｆｉｃ　ｉｎｔｅｇｒａｔｅｄ　ｃｉｒｃｕｉｔ）と、システム・オン・チップ（ＳＯＣ：ｓｙｓｔｅｍ　ｏｎ　ａ　ｃｈｉｐ）とを含めて、関連するタスクを実行するために、必要であり、または望ましい機能を提供することができる任意のタイプのプロセッサを備えることができる。用語「プロセッサ」は、その用語が、本明細書において使用されるとき、上記で説明されるタスクと、ＣＰＵ　４０、８０、２３０および２８０がそれらの役割を実行することを可能にするために必要と

見なされる任意の追加のタスクとを実行するようにプログラムされ、または構成されていることもあるこれらおよび他のタイプの計算デバイスを示すことを意図している。

【００３７】

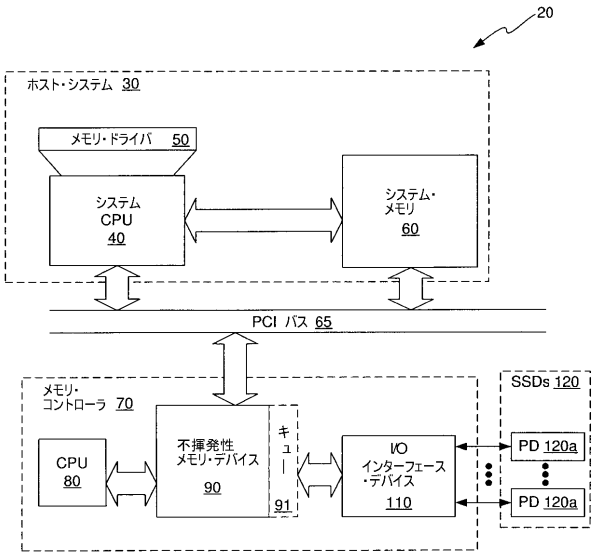
本発明は、本発明の原理および概念を実証する目的のために少数の実例となる、また例示の実施形態に関連して説明されてきていることに注意すべきである。当業者によって理解されることになるように、多数の変形形態が、本発明の範囲を逸脱することなく、上記で説明される実例となる実施形態に対して行われることもある。すべてのそのような変形形態は、本発明の範囲内にある。例えば、ＰＤ　１２０ａおよび３２０ａは、それぞれ図２および３に示されているが、それぞれ少なくとも１つのＳＳＤ　１２０または３２０の形でただ実施されているので、ＰＤ　１２０ａおよび３２０ａは、１つまたは複数のＳＳＤと、１つまたは複数のＨＤＤとの組合せとして実施される可能性がある。

【図１】

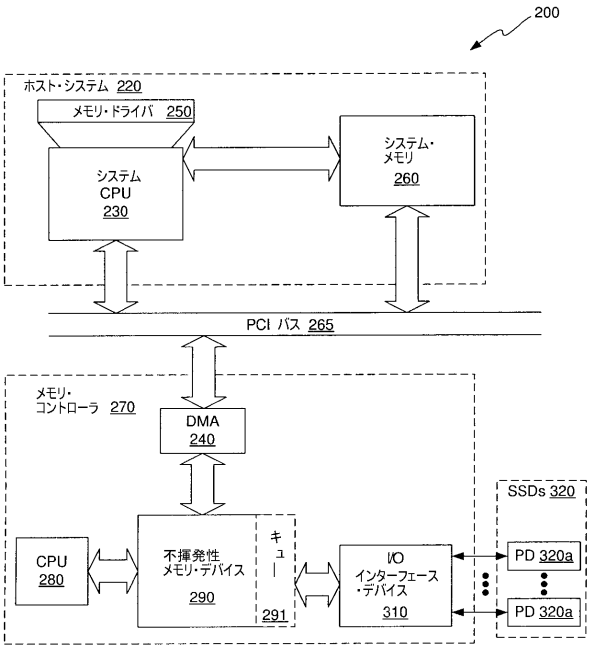


（先行技術）

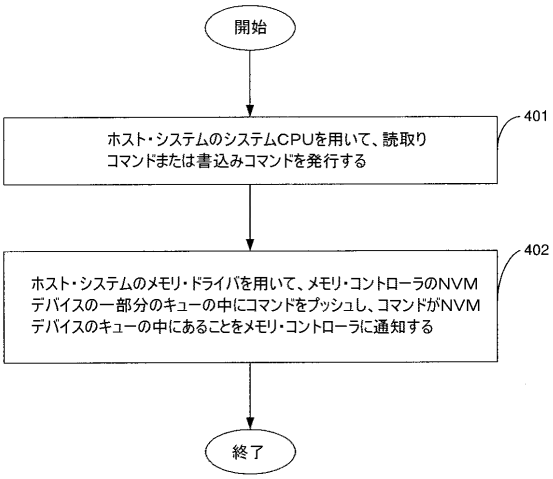
【図２】



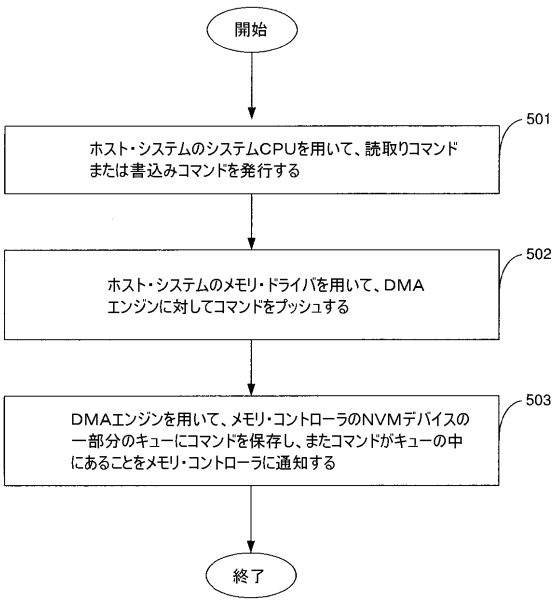
【図３】


200
ホスト・システム 220
メモリ・ドライバ 250
システム CPU 230
システム・メモリ 260
PCI バス 265
メモリ・コントローラ 270
DMA 240
CPU 280
不揮発性メモリ・デバイス 290
キュー 291
I/O インターフェース・デバイス 310
SSDs 320
PD 320a
PD 320a

【図４】


開始
ホスト・システムのシステムCPUを用いて、読取りコマンドまたは書込みコマンドを発行する
401
ホスト・システムのメモリ・ドライバを用いて、メモリ・コントローラのNVMデバイスの一部分のキューの中にコマンドをプッシュし、コマンドがNVMデバイスのキューの中にあることをメモリ・コントローラに通知する
402
終了

【図５】


開始
ホスト・システムのシステムCPUを用いて、読取りコマンドまたは書込みコマンドを発行する
501
ホスト・システムのメモリ・ドライバを用いて、DMAエンジンに対してコマンドをプッシュする
502
DMAエンジンを用いて、メモリ・コントローラのNVMデバイスの一部分のキューにコマンドを保存し、またコマンドがキューの中にあることをメモリ・コントローラに通知する
503
終了

フロントページの続き

(51)Int.Cl.	FI	テーマコード（参考）
	G06F 3/06 302Z	

(72)発明者 ルカ バート
アメリカ合衆国 ３００４１ ジョージア，カミング，マウントクレア ドライヴ ８２０
(72)発明者 アナン バデルディーニ
アメリカ合衆国 ３００４３ ジョージア，ローレンスヴィル，ハンプトン ウッズ ウェイ １８１７
(72)発明者 ホリア シミオネスク
アメリカ合衆国 ９４４０４ カリフォルニア，フォスター シティ，ピザロ レーン ９５６
(72)発明者 マーク イシュ
アメリカ合衆国 ３０３２８ ジョージア，サンディ スプリングス，ストーン ミル トレイル エヌ．イー．４０５

【外国語明細書】

[Title of the Invention]

METHOD AND SYSTEM FOR REDUCING WRITE LATENCY IN A DATA STORAGE SYSTEM BY USING A COMMAND-PUSH MODEL

TECHNICAL FIELD OF THE INVENTION

[0001] The invention relates generally to data storage systems and, more particularly, to a method and system for using a command-push model in a data storage system to reduce write latency.

BACKGROUND OF THE INVENTION

[0002] A storage array or disk array is a data storage device that includes multiple hard disk drives (HDDs) or similar persistent storage units. A storage array can allow large amounts of data to be stored in an efficient manner. A server or workstation may be directly attached to the storage array such that the storage array is local to the server or workstation. In cases in which the server or workstation is directly attached to the storage array, the storage array is typically referred to as a direct-attached storage (DAS) system. Alternatively, a server or workstation may be remotely attached to the storage array via a storage array network (SAN). In SAN systems, although the storage array is not local to the server or workstation, the disk drives of the array appear to the operating system (OS) of the server or workstation to be locally attached.

[0003] Fig. 1 illustrates a block diagram of a typical data storage system 2 that implements a command-pull model. The system 2 includes a host system 3, a memory controller 4, and a peripheral interconnect (PCI) or PCI Express (PCIe) bus 5. The controller 4 includes a central processing unit (CPU) 6, a memory device 7, and an I/O interface device 8. The I/O interface device 8 is configured to perform data transfer in compliance with known data transfer protocol standards, such as the Serial Attached SCSI (SAS) standard, the Serial Advanced Technology Attachment (SATA) standard, or the Nonvolatile Memory Host Controller Interface Express (NVMe) standard. The I/O interface device 8 controls the transfer of data to and from multiple physical disks (PDs) 9. The memory controller 4 communicates via the PCI bus 5 with a system CPU 11 and a system memory device 12. The system memory device 12 stores software programs for execution by the system CPU 11 and data. A portion of the system memory device 12 is used as a command queue 13.

[0004] During a typical write action, the system CPU 11 runs a memory driver software stack 14 that stores commands and data in the command queue 13. When the memory driver 14 stores a command in the command queue 13, it notifies the memory controller 4 that a command is ready to be executed. When the controller CPU 6 is ready to execute a command, it pulls the command, or multiple commands, and the associated data from the system queue 13 via the bus 5 and issues a completion interrupt to the host system 3. When the commands are executed by the memory controller 4, the controller CPU 6 causes the data associated with the commands to be temporarily stored in the controller memory device 7 and then subsequently written to one or more of the PDs 9 via the I/O interface device 8.

[0005] Historically, the performance of an HDD-based system of the type shown in Fig. 1 has been measured in terms of input/output (IO) per second (IOPS), and in some cases, megabytes per second (MB/s). Latency of such a storage system is typically given as: Latency_Overall = Latency_SW_Stack + Latency_Controller + Latency_HDD = 1/IOPS, where Latency_Overall is the overall latency of the system, Latency_SW_Stack is the latency associated with the memory driver 14, Latency_Controller is the latency associated with the memory controller 4, and Latency_HDD is the latency associated with the PDs 9. Latency_SW_Stack is typically on the order of microseconds (10^{-6} seconds). Likewise, Latency_Controller is typically on the order of tens of microseconds. However, Latency_HDD is typically on the order of milliseconds (10^{-3} seconds) or tens of milliseconds. Approximately 99% of overall latency is due to extremely slow mechanical parts of the HDDs. Therefore, for practical purposes, Latency_SW_Stack and Latency_Controller can be ignored when determining system performance. In other words, system performance can be estimated as being equal to Latency_HDD.

[0006] Recently, there has been a transition from using magnetic HDDs as the PDs 9 to using solid state drives (SSDs), or a combination of SSDs and HDDs, as the PDs 9. In the industry, the use of SSD-based solutions is viewed as an evolution of HDD-based solutions. However, SSD-based solutions are approximately one hundred times faster and consume much less power than HDD-based solutions. This view of SSD-based solutions has led the industry to continue using the pre-existent, above-described pull

methodology in SSD-based solutions to pull commands from the command queue into the memory controller. Also, because SSD-based solutions have been viewed in the industry as merely an evolution of HDD-based solutions, IOPS have been used as the performance metric for measuring system performance in storage systems that implement SSD-based solutions.

[0007] However, the differences between SSD-based solutions and HDD-based solutions are much greater than it appears, and traditional metrics should not be used to measure the performance of systems that implement SSD-based solutions. In a system that implements an SSD-based solution, the overall latency of the storage system is given as: Latency_Overall = Latency_SW_Stack + Latency_Controller + Latency_SSD = 1/IOPS, where Latency_SW_Stack is the latency associated with the memory driver 14, Latency_Controller is the latency associated with the memory controller 4, and Latency_SSD is the latency associated with the SSDs that are used as the PDs 9. Unlike the latency of the HDDs, the latency of the SSDs is on the order of tens to hundreds of microseconds, e.g., generally in the range of 100 to 300 microseconds, and the latencies associated with the memory driver 14 and the memory controller 4 add much more than that to the overall latency. Therefore, in calculating the overall latency of the storage system that implements an SSD-based solution, Latency_SW_Stack and Latency_Controller should no longer be ignored.

[0008] The command-pull approach requires quite a bit of interaction between the memory driver 14 and the memory controller 4. This is a convenient approach in HDD-based systems in that it allows the fast operating system (OS) side of the host system 3 to be almost completely independent of the slower HDD-based controller side so that the OS side can pile up as many commands as possible in the queue 13 to provide greater queue depth (QD), which is very desirable and common in HDD-based solutions. The memory controller 4 can then pull the commands from the queue 13 at its own pace. While this method is very convenient in HDD-based solutions, it adds a large amount of extra latency due to the synchronization that is required between the memory controller 4 and the host system 3, and due to the fact that the memory controller 4 may pick up commands at times much later than when they were issued. In addition, if there is lot of work to be done by the memory controller 4, as is often the case, all of the command

processing must compete with the rest of workload of the memory controller 4, which adds more latency to the overall latency.

[0009] The above-described command-pull model works very well for HDD-based solutions where adding 50 to 500 microseconds to a command that typically may take about 10,000 microseconds to complete is negligible given the other advantages that the method provides. However, the command-pull model does not produce acceptable results when used in a storage system that implements an SSD-based solution where the access time may be as low as 100 microseconds, or, in cases in which a dynamic random access memory (DRAM) write back (WB) buffer is used in the memory controller 4, as low as 1 to 5 microseconds.

[0010] Nevertheless, as indicated above, the overall latency of storage systems that implement SSD-based solutions is still being estimated as being equal to Latency_SSD, while leaving Latency_SW_Stack and Latency_Controller out of the estimation. For this reason, attempts at reducing the overall latency of storage systems that implement SSD-based solutions have focused primarily on reducing Latency_SSD rather than on reducing Latency_SW_Stack or Latency_Controller.

[0011] A need exists for a storage system that implements an SSD-based solution and that significantly reduces overall latency by significantly reducing one or both of Latency_SW_Stack and Latency_Controller.

SUMMARY OF THE INVENTION

[0012] The invention provides a data storage system, a memory controller, a method and a computer-readable medium, all of which implement a command-push model that reduces latency. The data storage system comprises a host system, a memory controller, at least one SSD, and a bus that interconnects the host system and the memory controller. The host system comprises a system processor and a system memory device. The memory controller comprises a controller processor, a nonvolatile memory (NVM) device and an input/output (I/O) interface device. A portion of the NVM device is used as a command queue. The SSD is connected to the I/O interface device and is configured as an array of PDs. The host system accesses the NVM device via the bus and pushes commands into the command queue of the NVM device via the bus.

[0013] The method comprises:

in a memory controller comprising a controller processor, a NVM device and an I/O interface device connected to at least one SSD configured as an array of PDs, configuring a portion of the NVM device as a command queue; and

with a host system interconnected with the memory controller via a bus, pushing a command into the memory controller via the bus; and

in the memory controller, storing the command in the command queue of the NVM device.

[0014] The computer-readable medium has a computer program stored thereon for execution by a processor of a host system for pushing commands into a command queue of a NVM device of a memory controller that is connected to the host system via a bus. The computer program comprises first and second code portions. The first code portion receives a command to read or write one or more addresses of a SSD configured as an array of PDs. The second code portion pushes the command into the memory controller. In accordance with one illustrative embodiment, the second code portion pushes the command into the command queue of the NVM device of the memory controller. In accordance with another illustrative embodiment, the second code portion pushes the command into a direct memory access (DMA) engine of the memory controller that stores the command in the command queue of the NVM device of the memory controller.

[0015] These and other features and advantages of the invention will become apparent from the following description, drawings and claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] Fig. 1 illustrates a block diagram of a known data storage system that implements a command-pull model.

[0017] Fig. 2 illustrates a block diagram of a data storage system in accordance with an illustrative embodiment that implements a command-push model.

[0018] Fig. 3 illustrates a block diagram of a data storage system in accordance with another illustrative embodiment that implements a command-push model.

[0019] Fig. 4 illustrates a flowchart that represents the command-push method performed by the data storage system shown in Fig. 2.

[0020] Fig. 5 illustrates a flowchart that represents the command-push method performed by the data storage system shown in Fig. 3.

DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

[0021] In accordance with the invention, a data storage system is provided that implements a command-push model that reduces the latencies associated with the memory driver and the memory controller. The command-push model eliminates the need for synchronization between the host system and the memory controller for write command IOs, thereby eliminating the latency associated with synchronization. As command IOs are issued in the host system, the host system accesses a memory device of the memory controller and pushes the command IOs and the associated data into a command queue of the memory device. Therefore, the host system completes each write command IO without any intervention from the memory controller, thereby obviating the need for synchronization, or handshaking, between the host system and the memory controller.

[0022] The memory device of the memory controller is protected against power failures and typical memory errors, and as such, may be considered a permanent storage element. Therefore, the host system may consider a write command IO as safely completed as of the time that it pushes the write command and any associated data into the command queue of the memory device of the memory controller. In other words, from the perspective of the host system, when it pushes the write command into the command queue of the memory device of the memory controller, it is as if it had completed a write to the PDs 9. Therefore, the host system completes each write command IO without any intervention from the memory controller, thereby obviating the need for synchronization, or handshaking, between the host system and the memory controller.

[0023] Furthermore, in contrast to the command-pull model described above with reference to Fig. 1, the memory controller of the invention does not need to issue a completion interrupt to the host system once the memory controller has completed processing of the write command because the host system considers the write command completed at the time that it pushes the write command into the command queue of the

memory controller. This feature allows the host system and the memory controller to operate independently of one another with respect to write commands. The combination of all of these features results in a large reduction in both Latency_SW_Stack and Latency_Controller, which leads to a large reduction in overall latency.

[0024] Thus, rather than using the command-pull model described above with reference to Fig. 1 in which commands are prepared in the command queue 13 of the system memory 12 and later pulled in and processed by the memory controller 4, the host system pushes commands and any associated data into the command queue located in the memory device of the memory controller. The manner in which the host system pushes the commands and any associated data into the command queue of the memory device of the memory controller can be done in a number of ways. In accordance with a first illustrative embodiment, the memory driver of the host system programmatically pushes commands and associated data into the command queue of the memory device of the memory controller. In accordance with a second illustrative embodiment described herein, a direct memory access (DMA) engine of the memory controller pushes commands and any associated data into the command queue of the memory device of the memory controller. Both of these embodiments obviate the need for synchronization, or handshaking, between the host system and the memory controller.

[0025] Fig. 2 illustrates a block diagram of a data storage system 20 in accordance with the first illustrative embodiment that implements a command-push model. The system 20 includes a host system 30, a memory controller 70, a peripheral interconnect (PCI) or PCI Express (PCIe) bus 65, and at least one SSD 120 comprising PDs 120a. The host system 30 includes a system CPU 40 and a system memory device 60. The memory controller 70 includes a CPU 80, a nonvolatile memory (NVM) device 90, and an I/O interface device 110. The NVM device 90 is typically a DRAM device. A portion of the NVM device 90 is used as a command queue 91. The I/O interface device 110 is configured to perform data transfer in compliance with known data transfer protocol standards, such as the SAS, SATA and/or NVMe standards. The I/O interface device 110 controls the transfer of data to and from multiple PDs 120a of the SSD(s) 120. The memory controller 70 communicates with the system CPU 40 via the PCI bus 65 and the

memory driver 50. The system memory device 60 stores software programs for execution by the system CPU 40 and data.

[0026] In accordance with this illustrative embodiment, the memory driver 50 communicates directly with the NVM device 90 via bus 65. During a typical write action, the system CPU 40 runs a software stack that includes the memory driver 50, which pushes commands and data into the command queue 91 of the NVM device 90 and notifies the controller CPU 80 that the command and data are in the queue 91. No synchronization or handshaking between the host system 30 and the memory controller 70 is needed to perform the push action. Once the memory driver 50 has pushed a write command and the associated data into the command queue 91, the OS of the host system 30 considers the write command completed. Therefore, it is not necessary for the memory controller 70 to issue a completion interrupt to the host system 30 to inform the host CPU 40 that the write command has been completed.

[0027] When the controller CPU 80 is ready to execute the command, it retrieves the command and data from the NVM device 90 and executes the command. In the case of a write command, the controller CPU 80 causes the associated data to be written to the SSD PDs 120a via the I/O interface device 110. In the case of a read command, the controller CPU 80 checks cache (not shown for clarity) of the NVM 90 to determine whether the data to be read is in cache, and if it is, reads the data from cache and transfers it to the host system 30. If the data to be read is not in cache, the CPU 80 causes the data to be read from the SSD PDs 120a and transferred to the host system 30.

[0028] One of the key differences between the command-push model of Fig. 2 and the command-pull model of Fig. 1 is that command ownership is transferred from the host system 30 to the queue 91 of the NVM device 90 before the controller CPU 80 is even aware of the command. In the command-pull model of Fig. 1, command ownership associated with a write command is not transferred until the host system 3 receives the completion interrupt from the memory controller 4. In addition, in the command-pull model of Fig. 1, synchronization, or handshaking, takes place between the controller CPU 6 and the host system 3 in order to pull commands from the queue 13 of the system memory device 12 into the memory controller 4. Eliminating these latencies will

typically result in a reduction in write latency of one to two orders of magnitude compared to the command-pull model shown in Fig. 1.

[0029] Because the memory controller 70 takes ownership of commands once they have been pushed into the queue 91 of the NVM 90, it is important for the NVM 90 to be protected against power failures. Such protection can be achieved in a number of ways, but one way is to use a battery backup (BBU), super-capacitor (Supercap) dynamic random access memory (DRAM) device for the NVM 90. The manner in which a BBU, Supercap DRAM device or other suitable memory device may be selected for this purpose will be understood by those of skill in the art.

[0030] In order for the memory driver 50 to efficiently push commands into the queue 91, the NVM device 90 should be directly accessible by the memory driver 50 via the bus 65. This accessibility may be accomplished in a number of ways. One way to accomplish this is to use the same command and data structure in the NVM device 90 that is used in the host system 30. This is possible in PCIe through a convenient programming of Base Address Registers (BARs), as will be understood by persons of skill in the art. The data transfer from the host system 3 to the NVM device 90 can be accomplished programmatically by the memory driver 50.

[0031] The memory driver 50 should also be provided with a way of determining which entries in the NVM device 90 are available in order to map/tag them with command information. Each command is made up of a data set and command information. The command information includes information such as Read/Write, Block address, pointers, and flags. The command information varies in accordance with the command type and is created independently of the data set. Both the data set and the command information need to be moved into the NVM 90 and the data set has to be mapped/tagged with the corresponding command information to ensure consistency between the command information and the data set. The manner in which this is accomplished will vary depending on the type or configuration of the memory controller 70 that is used. All memory controllers are configurable to provide this functionality, although the efficiency of the solution will depend on the hardware, software and/or firmware implementation of the memory controller. Therefore, this issue will need to be addressed on a case-by-case basis, as will be understood by those of skill in the art.

[0032] Fig. 3 illustrates a block diagram of a data storage system 200 in accordance with the second illustrative embodiment, in which a DMA engine is used to push commands and data into the queue of the memory device of the memory controller. Although the DMA engine may be part of the host system or part of the memory controller, for illustrative purposes it is shown and described as being part of the memory controller. The system 200 includes a host system 220, a memory controller 270, a PCI or PCIe bus 265, and at least one SSD 320 comprising PDs 320a. The host system 220 includes a system CPU 230 and a system memory device 260. The memory controller 270 includes a DMA engine 240, a CPU 280, an NVM device 290, and an I/O interface device 310. The NVM device 290 is typically a DRAM device, and it has the same protections described above with reference to NVM 90. A portion of the NVM device 290 is used as a command queue 291. The I/O interface device 310 is configured to perform data transfer in compliance with known data transfer protocol standards, such as the SAS, SATA and/or NVMe standards. The I/O interface device 310 controls the transfer of data to and from multiple PDs 320a of the SSD(s) 320. The memory controller 270 communicates with the system CPU 230 via the PCI bus 265 and the memory driver 250. The system memory device 260 stores software programs for execution by the system CPU 230 and data.

[0033] In accordance with this illustrative embodiment, the DMA engine 240 communicates with the memory driver 250 via the bus 265 and communicates directly with the NVM device 290. During a typical write action, the memory driver 250 pushes the command and any associated data to the DMA engine 240 via bus 265. The DMA engine 240 stores the command and data in the command queue 291 of the NVM device 290. No synchronization or handshaking between the host system 220 and the memory controller 270 is needed to perform the push action. Once the DMA engine 240 has stored a write command and the associated data into the command queue 291, it preferably notifies the controller CPU 280 that the command and data are in the NVM 290, typically by issuing an interrupt. The OS of the host system 220 considers the write command completed as of the time that the command and data are transferred to the DMA engine 240. Therefore, it is not necessary for the memory controller 270 to issue a

completion interrupt to the host system 220 to inform it that the write command has been completed.

[0034] When the controller CPU 280 is ready to execute the command, it retrieves the command and data from the NVM device 290 and executes the command. In the case of a write command, the controller CPU 280 causes the associated data to be written to the SSD PDs 320a via the I/O interface device 310. In the case of a read command, the controller CPU 280 checks cache (not shown for clarity) of the NVM 290 to determine whether the data to be read is in cache, and if it is, reads the data from cache and transfers it to the host system 220. If the data to be read is not in cache, the CPU 280 causes the data to be read from the SSD PDs 320a and transferred to the host system 220.

[0035] As with the command-push model described above with reference to Fig. 2, with the command-push model of Fig. 3, command ownership is transferred from the host system 220 to the queue 291 of the NVM device 290 before the controller CPU 280 is even aware of the command. As with the command-push model described above with reference to Fig. 2, with the command-push model of Fig. 3, no synchronization, or handshaking, is required between the controller CPU 280 and the host system 22 in order to push commands into the queue 291. Therefore, the latencies described above with reference to Fig. 1 are eliminated.

[0036] Fig. 4 illustrates a flowchart that represents the command-push method in accordance with an illustrative embodiment performed in part by the host system 30 and in part by the memory controller 70 shown in Fig. 2. The method will be described with reference to a single command, although the steps of the method could also be performed simultaneously on multiple commands. The system CPU 40 issues a read or write command, as indicated by block 401. The memory driver 50 of the host system 30 pushes the command into the queue 91 of the NVM device 90 and notifies the memory controller 70 that the command is in the NVM device 90, as indicated by block 402.

[0037] In the case of a write command, the data associated with the command is also pushed into the NVM device 90. Subsequently, the command and data read from the NVM device 90 are written to the PDs 120a of the SSD(s) 120. In the case of a read command that has been pushed into the NVM device 90, the memory controller 70 will typically determine whether the data is stored in the cache of the NVM device 90. If the

memory controller 70 determines that the data is stored in cache (i.e., a cache hit), it reads the data from cache and transfers it to the host system 30. If the memory controller 70 determines that the data is not stored in cache (i.e., a cache miss), it reads the data from the PDs 120a of the SSD(s) 120 and transfers the data to the host system 30.

[0038] In the case in which a write command is pushed into the queue 91 of the NVM device 90, the IO is complete in the host system 30 once the memory driver 50 pushes the write command into the queue 91. In the case in which a read command is pushed into the queue 91, the IO is not complete in the host system 30 until the read data is returned to the host system 30, which is the way read commands are handled in the known storage system 2 shown in Fig. 1. The reduction in latency achieved by the methods and systems of the invention is most significant with respect to write commands due to the fact that synchronization is no longer required to transfer write commands from the host system 30 into the memory controller 70 and due to the fact that the host system 30 no longer has to wait for a completion interrupt before it considers the write command completed.

[0039] Fig. 5 illustrates a flowchart that represents the command-push method in accordance with an illustrative embodiment performed in part by the host system 220 and in part by the memory controller 270 shown in Fig. 3. The method will be described with reference to a single command, although the steps of the method could also be performed simultaneously on multiple commands. The system CPU 230 issues a read or write command, as indicated by block 501. The memory controller 250 pushes the command to the DMA engine 240, as indicated by block 502. The DMA engine 240 stores the command in the queue 291 of the NVM device 290 and notifies the memory controller 270 that the command is in the NVM device 290, as indicated by block 503. The invention is not limited with respect to the way in which the memory controller 270 is notified that a command has been placed in the queue 291. This could be accomplished in a number of ways, e.g., issuing an interrupt, updating a value in a register, etc., as will be understood by those of skill in the art.

[0040] The functionality of the host systems 30 and 220 and of the memory controllers 70 and 270 may be implemented in hardware, software, firmware, or a combination thereof. The computer code for implementing functionality in software or firmware is stored on a computer-readable medium (CRM), such as system memory

devices 60 and 260 or NVM devices 90 or 290 or some other memory device. The CRM may be any type of memory device including, but not limited to, magnetic storage devices, solid state storage devices, flash memory devices, and optical storage devices. Each of the CPUs 40, 80, 230 and 280 typically comprises at least one microprocessor, but may comprise any type of processor that is capable of providing the functionality that is necessary or desired to perform the associated tasks, including, for example, a microcontroller, a digital signal processor (DSP), an application specific integrated circuit (ASIC), and a system on a chip (SOC). The term "processor," as that term is used herein, is intended denote these and other types of computational devices that may be programmed or configured to perform the tasks described above and any additional tasks that are deemed necessary to allow the CPUs 40, 80, 230 and 280 to perform their roles.

[0041] It should be noted that the invention has been described with reference to a few illustrative, or exemplary, embodiments for the purposes of demonstrating the principles and concepts of the invention. As will be understood by persons of skill in the art, many variations may be made to the illustrative embodiments described above without deviating from the scope of the invention. All such variations are within the scope of the invention. For example, although the PDs 120a and 320a are shown in Figs. 2 and 3, respectively, as being implemented solely in at least one SSD 120 or 320, respectively, the PDs 120a and 320a may be implemented as a combination of one or more SSDs and one or more HDDs.

CLAIMS

What is claimed is:

1. A data storage system comprising:

a host system comprising a system processor and a system memory device;

a memory controller comprising a controller processor, a nonvolatile memory (NVM) device and an input/output (I/O) interface device, wherein a portion of the NVM device is used as a command queue;

at least one solid state drive (SSD) connected to the I/O interface device, wherein said at least one SSD is configured as an array of physical disk drives (PDs); and

a bus interconnecting the host system with the memory controller, wherein the host system accesses the NVM device via the bus and pushes commands into the command queue of the NVM device via the bus.

2. The data storage system of claim 1, wherein the system processor executes a memory driver program that accesses the NVM device via the bus and pushes the commands into the command queue of the NVM device via the bus.

3. The data storage system of claim 1, wherein the memory controller further comprises a direct memory access (DMA) engine that directly accesses the NVM device, and wherein the system processor executes a memory driver program that pushes the commands to the DMA engine via the bus, and wherein the DMA engine stores the commands in the command queue of the NVM device.

4. The data storage system of claim 1, wherein the host system notifies the memory controller when a command has been pushed into the command queue of the NVM device.

5. The data storage system of claim 2, wherein the host system and the NVM device use a same command and data structure.

6. The data storage system of claim 5, wherein the bus is a peripheral interconnect express (PCIe) bus, and wherein the memory driver program accesses the NVM device by using base address registers (BARs) of the PCIe bus.

7. A memory controller comprising:

an input/output (I/O) interface device, wherein a portion of the NVM device is used as a command queue;

at least one solid state drive (SSD) connected to the I/O interface device, wherein said at least one SSD is configured as an array of physical disk drives (PDs);

a controller processor; and

a nonvolatile memory (NVM) device, a portion of the NVM device being allocated as a command queue, wherein the NVM device is configured to be accessed by a host system via a bus that interconnects the host system with the memory controller, wherein the NVM device is configured to allow the host system to push commands into the command queue of the NVM device via the bus.

8. The memory controller of claim 7, wherein the NVM device is accessed by the host system via a memory driver program being executed by a processor of the host system, and wherein the bus is a peripheral interconnect express (PCIe) bus, and wherein the NVM device is accessible by the memory driver program through base address registers (BARs) of the PCIe bus.

9. The memory controller of claim 7, wherein the memory controller further comprises:

a direct memory access (DMA) engine that communicates with the host system via the bus and that communicates directly with the NVM device, and wherein the DMA engine is accessed by the host system via the bus, and the DMA engine receives commands pushed to the DMA engine by the memory driver program and stores the commands in the command queue of the NVM device.

10. The memory controller of claim 8, wherein the NVM device uses a same command and data structure as the host system.

11. A method for reducing latency in a data storage system, the method comprising:

in a memory controller comprising a controller processor, a nonvolatile memory (NVM) device and an input/output (I/O) interface device connected to at least one solid state drive (SSD) configured as an array of physical disk drives (PDs), configuring a portion of the NVM device as a command queue;

with a host system interconnected with the memory controller via a bus, pushing a command into the memory controller via the bus; and

in the memory controller, storing the command in the command queue of the NVM device.

12. The method of claim 11, wherein a system processor of the host system executes a memory driver program that accesses the NVM device via the bus and pushes the commands into the command queue of the NVM device via the bus.

13. The method of claim 11, wherein the memory controller further comprises a direct memory access (DMA) engine, and wherein a system processor of the host system executes a memory driver program that pushes commands from the host system to the DMA engine via the bus, and wherein the DMA engine stores the commands into the command queue of the NVM device via the bus.

14. The method of claim 12, further comprising:

wherein the host system notifies the memory controller when a command has been pushed into the command queue of the NVM device.

15. The method of claim 12, wherein the host system and the NVM device use a same command and data structure.

16. The method of claim 15, wherein the bus is a peripheral interconnect express (PCIe) bus, and wherein the memory driver program accesses the NVM device by using base address registers (BARs) of the PCIe bus.

17. A non-transitory computer-readable medium having a computer program stored thereon for execution by a processor of a host system for pushing commands into a command queue of a non-volatile memory (NVM) device of a memory controller that is connected to the host system via a bus, the computer program comprising:

a first code portion for receiving a command to read or write one or more addresses of a solid state drive (SSD) configured as an array of physical disk drives (PDs); and

a second code portion for pushing the command into the memory controller.

18. The non-transitory computer-readable medium of claim 17, wherein the second code portion pushes the command into the command queue of the NVM device, and wherein the computer program further comprises:

a third code portion for notifying a processor of the memory controller when the command has been pushed into the command queue of the NVM device of the memory controller.

19. The non-transitory computer-readable medium of claim 17, wherein the second code portion pushes the command into a direct memory access (DMA) engine of the memory controller, wherein the DMA engine is directly connected to the NVM device to allow the DMA engine to store the command in the command queue, and wherein the computer program further comprises:

a third code portion for notifying a processor of the memory controller when the command has been pushed into the command queue of the NVM device of the memory controller.

[Abstract]

ABSTRACT OF THE DISCLOSURE

A data storage system is provided that implements a command-push model that reduces latencies. The host system has access to a nonvolatile memory (NVM) device of the memory controller to allow the host system to push commands into a command queue located in the NVM device. The host system completes each IO without the need for intervention from the memory controller, thereby obviating the need for synchronization, or handshaking, between the host system and the memory controller. For write commands, the memory controller does not need to issue a completion interrupt to the host system upon completion of the command because the host system considers the write command completed at the time that the write command is pushed into the queue of the memory controller. The combination of all of these features results in a large reduction in overall latency.

[Representative Drawing]

Fig. 2

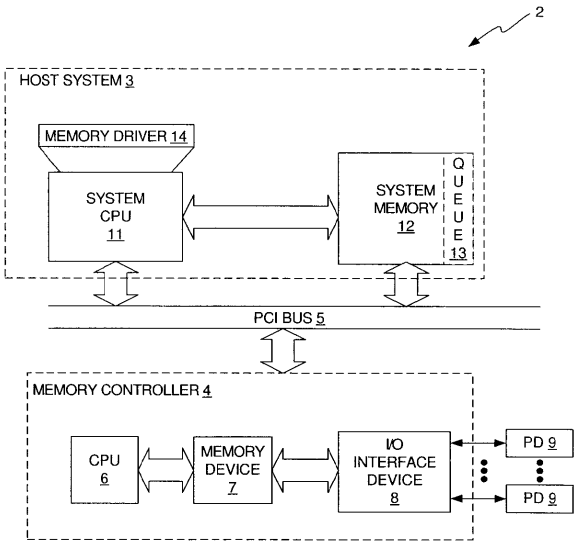


FIG. 1
(PRIOR ART)

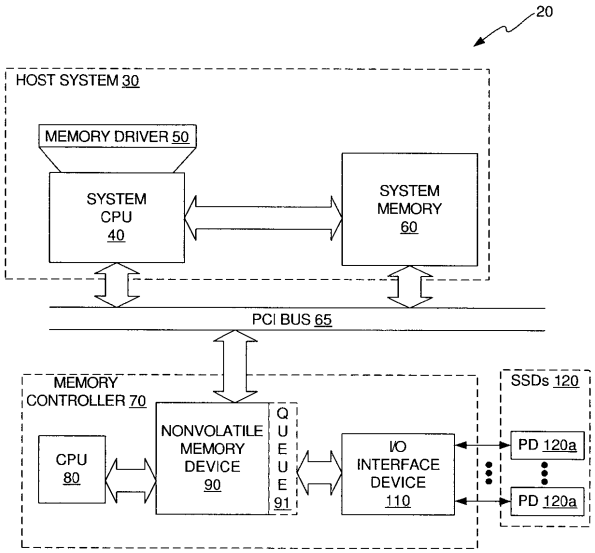


FIG. 2

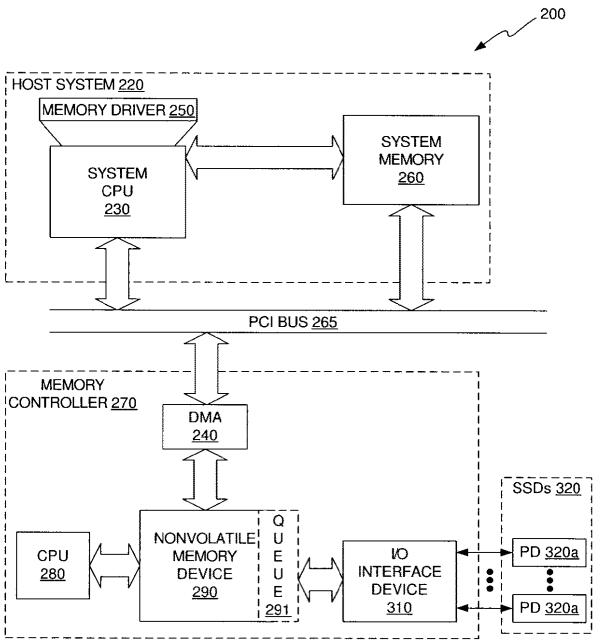


FIG. 3

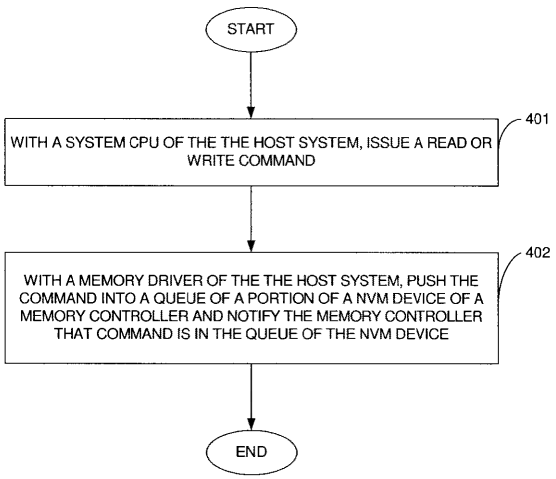


FIG. 4

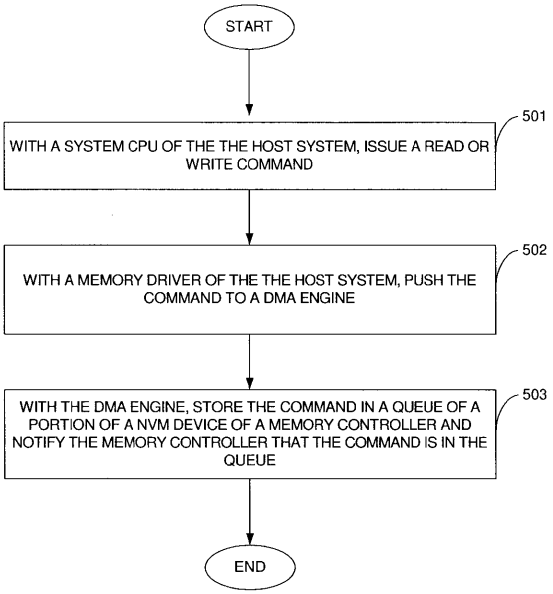

START
WITH A SYSTEM CPU OF THE THE HOST SYSTEM, ISSUE A READ OR WRITE COMMAND
501
WITH A MEMORY DRIVER OF THE THE HOST SYSTEM, PUSH THE COMMAND TO A DMA ENGINE
502
WITH THE DMA ENGINE, STORE THE COMMAND IN A QUEUE OF A PORTION OF A NVM DEVICE OF A MEMORY CONTROLLER AND NOTIFY THE MEMORY CONTROLLER THAT THE COMMAND IS IN THE QUEUE
503
END

FIG. 5