(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 101751356 A
(43)申请公布日 2010.06.23

(21)申请号 200910222436.4

(22)申请日 2009.11.13

(30)优先权数据
12/337,703 2008.12.18 US

(71)申请人 国际商业机器公司
地址 美国纽约阿芒克

(72)发明人 吕纪竹 M·P·佩龙

(74)专利代理机构 北京市金杜律师事务所 11256
代理人 酆迅

(51)Int.Cl.
G06F 13/28(2006.01)

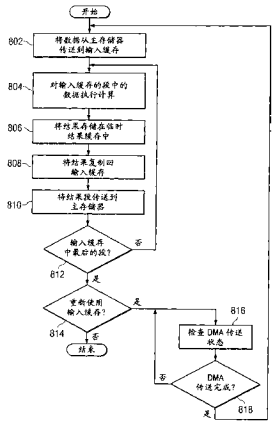
权利要求书 2 页 说明书 8 页 附图 9 页

(54)发明名称

用于改进直接存储器存取传送效率的方法、系统和装置

(57)摘要

本发明涉及用于改进直接存储器存取传送效率的方法、系统和装置。提供了一种用于改进多个较小直接存储器存取传送的效率的机制。该机制使用一个输入缓存和小结果缓存或一些临时变量来临时地存储计算结果。该机制对输入缓存中的数据段执行计算并且将结果存储在临时结果缓存中。该机制随后将结果复制回输入缓存中。这样，该机制将输入缓存既作为输入缓存也作为结果缓存使用。该机制随后对包含计算结果的输入缓存的段执行直接存储器存取传送并且随后对输入缓存的下一段执行计算。该机制随后重复该过程直至整个输入缓存已经被处理为止。



CN 101751356 A

1. 一种在数据处理系统中用于改进的直接存储器存取传送的方法，所述方法包括：

执行直接存储器存取传送以将数据从主存储器加载到输入缓存；

对所述输入缓存内的第一数据段执行至少一个计算以产生第一结果数据段；

将所述第一结果数据段存回到所述输入缓存中；并且

发起直接存储器存取传送以将所述第一结果数据段从所述输入缓存存储到所述主存储器。

2. 根据权利要求 1 所述的方法，其中将所述第一结果数据段存储回所述输入缓存包括：

将所述第一结果数据段存储在临时结果缓存中；并且

将所述第一结果数据段从所述临时结果缓存复制到所述输入缓存中。

3. 根据权利要求 2 所述的方法，其中所述临时结果缓存和所述输入缓存存储在与协同处理单元相关联的本地储存器中。

4. 根据权利要求 2 所述的方法，其中所述临时结果缓存包括一个或多个寄存器。

5. 根据权利要求 1 所述的方法，进一步包括：

与将前一结果数据段从所述输入缓存存储到所述主存储器的直接存储器存取传送并行地，对当前数据段执行至少一个计算以产生当前结果数据段。

6. 根据权利要求 5 所述的方法，进一步包括：

将所述当前结果数据段存回到所述输入缓存中；并且

发起直接存储器存取传送以将所述当前结果数据段从所述输入缓存存储到所述主存储器。

7. 根据权利要求 6 所述的方法，进一步包括：

响应于所述当前数据段是所述输入缓存中的最后一个数据段，确定是否要重新使用所述输入缓存。

8. 根据权利要求 7 所述的方法，进一步包括：

响应于确定要重新使用所述输入缓存，检查用于存储此前结果数据段的直接存储器存取传送的状态；并且

响应于所有用于存储此前结果数据段的直接存储器存取传送已完成，执行直接存储器存取传送以将后续数据从所述主存储器加载到所述输入缓存。

9. 一种包括计算机可读介质的计算机程序产品，所述介质上记录有计算机可读程序，其中当所述计算机可读程序在计算装置上被执行时，使所述计算装置实现权利要求 1 至 8 的任一方法。

10. 一种计算机系统，包括用于实现权利要求 1 至 8 的任一方法的装置。

11. 一种装置，包括：

处理器；以及

连接到所述处理器的存储器，其中所述存储器包括指令，当所述指令由所述处理器执行时，使处理器：

执行直接存储器存取传送以将数据从主存储器加载到输入缓存；

对所述输入缓存中的第一数据段执行至少一个计算以产生第一结果数据段；

将所述第一结果数据段存回到所述输入缓存中；并且

发起直接存储器存取传送以将所述第一结果数据段从所述输入缓存存储到所述主存储器。

12. 根据权利要求 11 所述的装置，其中将所述第一结果数据段存储回所述输入缓存包括：

将所述第一结果数据段存储在临时结果缓存中；并且

将所述第一结果数据段从所述临时结果缓存复制到所述输入缓存中。

13. 根据权利要求 11 所述的装置，其中所述指令进一步使处理器：

与将前一结果数据段从所述输入缓存存储到所述主存储器的直接存储器存取传送并行地，对当前数据段执行至少一个计算以产生当前结果数据段；

将所述当前结果数据段存回所述输入缓存中；并且

发起直接存储器存取传送以将所述当前结果数据段从所述输入缓存存储到所述主存储器。

14. 根据权利要求 13 所述的装置，其中所述指令进一步使处理器：

响应于所述当前数据段是所述输入缓存中的最后一个数据段，确定是否要重新使用所述输入缓存；

响应于确定要重新使用所述输入缓存，检查用于存储此前结果数据段的直接存储器存取传送的状态；并且

响应于所有用于存储此前结果数据段的直接存储器存取传送已完成，执行直接存储器存取传送以将后续数据从所述主存储器加载到所述输入缓存。

用于改进直接存储器存取传送效率的方法、系统和装置

技术领域

[0001] 本发明一般地涉及改进的数据处理装置和方法并且更特别地涉及改进多个较小直接存储器存取传送的效率以便在 Cell 宽带引擎 (CellBroadband Engine™) 处理器上编程的方法和装置。

背景技术

[0002] Cell 宽带引擎体系结构将例如性能有限的通用目的 PowerArchitecture® 内核和极大地加速多媒体和矢量处理应用的流线型共处理单元结合起来。共处理单元被称为协同处理单元 (SPE)，其显著加速许多形式的专门计算。Cell 宽带引擎 (Cell/B.E.™) 处理器是单芯片多核体系结构，其为从计算机游戏到在各种应用领域里的很多计算密集型应用，如地震数据处理、生物信息学、图形处理等广大范围的应用提供低成本和高性能。Cell 宽带引擎和 Cell/B.E. 是索尼计算机娱乐公司在美国、其他国家或两者的商标。POWER ARCHITECTURE 是 IBM 公司在美国、其他国家或两者的注册商标。

[0003] Cell/B.E.™ 体系结构提供的一个特性是例如使用八个协同处理单元 (SPE)，每个单元使用其自己的本地储存器 (LS) 而不是高速缓存。这要求 Cell/B.E.™ 程序员明确地使用直接存储器存取 (DMA) 指令以在主共享存储器和本地储存器之间传送数据和 / 或结果。该特性为 Cell/B.E.™ 程序员提供了更高的灵活性，也带来了更多的复杂性。

[0004] 于是，Cell/B.E.™ 程序员必须考虑使用 LS 的效率，这大大影响应用的实际性能。如何有效地使用 LS 是 Cell/B.E.™ 程序员应当考虑的一个十分重要的问题。由于每个 LS 空间有限，在主存储器和 LS 之间频繁往复地执行 DMA 传送是寻常的。因此，用于执行 DMA 传送的策略对应用性能来说可能是关键的。

[0005] 目前一种普遍使用的技术被称为“双缓冲”。双缓冲的基本思想是通过在两个缓存之间切换计算而隐藏 DMA 传送延迟，由此当在其中一个缓存上执行 DMA 传送的同时在另一个缓存上执行计算。对于受计算限制的应用，此技术能够有效地隐藏 DMA 传送延迟。

[0006] 然而，有很多应用是受存储器限制的，并且在那些情况下应用的性能不是受到每个 SPE 的计算能力的限制，而是受到 DMA 传送带宽的限制。因此，任何耗费在 DMA 传送上的时间可能对应用的性能有不利的影响。为了获得更好的性能和效率，程序员可以尽量以尽可能大的传送大小执行尽可能少的 DMA 传送。然而，由于很多计算算法的性质，可能无法避免多个较小的 DMA 传送，使得双缓冲成为改进性能的唯一技术。

发明内容

[0007] 在一个示例性实施例中，提供一种在数据处理系统中用于改进的直接存储器存取传送的方法。方法包括执行直接存储器存取传送以将数据从主存储器加载到输入缓存，对输入缓存中的第一数据段执行至少一个计算以产生第一结果数据段，将第一结果数据段存回到输入缓存中，并且发起直接存储器存取传送以将第一结果数据段从输入缓存存储到主存储器。

[0008] 在另一个示例性实施例中，提供一种包括计算机可用或可读介质的计算机程序产品，该介质具有计算机可记录程序。当计算机可读程序在计算装置上被执行时，使计算装置执行上述关于方法示例性实施例的各种操作之一或其组合。

[0009] 在另一个示例性实施例中，提供一种系统 / 装置。系统 / 装置可以包括一个或多个处理器以及连接到该一个或多个处理器的存储器。存储器可以包括指令，当该指令被一个或多个处理器执行时，使一个或多个处理器执行上述关于方法示例性实施例的各种操作之一或其组合。

[0010] 本发明的这些以及其他特性和优点将在以下对本发明示例性实施例的详述中描述，或者本领域的普通技术人员在阅读之后将理解本发明的这些以及其他特性和优点。

附图说明

[0011] 本发明，以及优选使用方式及其进一步的目的和优点，将可以通过参照附图并参照以下对本发明示例性实施例的详述而得到最好的理解。

[0012] 图 1 是其中可以实现本发明的各个方面的数据处理系统的示例性方框图；

[0013] 图 2 示出了直接存储器存取传送技术；

[0014] 图 3 示出了使用双缓冲的直接存储器存取传送技术；

[0015] 图 4 示出了根据一个示例性实施例的使用临时结果缓存的直接存储器存取传送技术；

[0016] 图 5A 和图 5B 示出了使用直接存储器存取传送的代码的例子，其中直接存储器存取传送使用双缓冲；

[0017] 图 6 示出了根据一个示例性实施例的使用直接存储器存取传送的代码的例子，其中直接存储器存取传送使用临时变量；

[0018] 图 7 是示出根据一个示例性实施例的对各种矩阵大小和计算的性能改进的表；以及

[0019] 图 8 是示出根据一个示例性实施例利用临时结果缓存的直接存储器存取传送操作的流程图。

具体实施方式

[0020] 示例性实施例提供用于改进多个较小直接存储器存取传送的效率的机制。该机制可以用于双缓冲技术通常所应用的任何应用。该机制使用一个输入缓存和小结果缓存或一些临时变量以临时地存储计算结果。该机制对输入缓存中的数据段执行计算并且将结果或中间结果存储在结果缓存或临时变量中。该机制随后将结果复制回输入缓存或直接地将结果存储在输入缓存中。这样，该机制将输入缓存既作为输入缓存也作为结果缓存加以使用。

[0021] 该机制随后对包含计算结果的输入缓存的段执行直接存储器存取传送并且随后对输入缓存的下一段执行计算。该机制随后重复该过程直至整个输入缓存已经被处理为止。该机制在计算和直接存储器存取传送过程期间不执行直接存储器存取状态检查。该机制仅当其需要重新使用输入缓存时才检查是否直接存储器存取传送已完成。

[0022] 该机制在本地储存器内执行额外的复制操作；然而，在本地储存器内进行复制的开销小于对直接存储器存取传送状态进行检查并等待直接存储器存取传送完成。实际上，

该机制可以不执行额外的复制，因为对于一些负杂计算，该机制可以简单地使用临时缓存以存储中间结果并且直接地将最终结果放回输入缓存而不是临时结果缓存中。因此，该机制不仅使用有限的较小的本地储存器空间，也带来更好的性能。

[0023] 示例性实施例可以在许多不同类型的数据处理环境中利用，包括分布式数据处理环境，单个数据处理设备等。为了给示例性实施例的特定单元和功能性的描述提供背景，在此作为示例性环境提供图 1，在其中可以实现示例性实施例的各个示例性方面。对图 1 的描述将主要地集中于单个数据处理设备实施方式，但这只是示例性的并且并不意在声明或暗示对有关本发明特性的任何限制。相反，示例性实施例意在包括分布式数据处理环境、网格计算环境等等。

[0024] 图 1 是其中可以实现本发明的各个方面的数据处理系统的示例性方框图。图 1 中示出的示例性数据处理系统是 Cell 宽带引擎 (Cell/B.E.™) 数据处理系统的例子。尽管 Cell/B.E.™处理器将在对本发明优选实施例的描述中使用，但本领域的普通技术人员在阅读以下描述后即可理解，本发明并不限于此。

[0025] 如图 1 所示，Cell/B.E.™处理器 100 包括主处理器单元 (PPE)110，其具有处理器 (PPU)116 及其 L1 高速缓存 112 和 L2 高速缓存 114；以及多个协同处理器单元 (SPE)120 至 134，其分别具有自己的协同处理器单元 (SPU)140 至 154、存储器流控制 155 至 162、本地存储器或储存器 (LS)163 至 170 以及总线接口单元 (BIU 单元)180 至 194，总线接口单元可以是例如组合直接存储器存取 (DMA)、存储器管理单元 (MMU) 和总线接口单元。并提供高带宽内部单元互联总线 (EIB)196、总线接口控制器 (BIC)197 以及存储器接口控制器 (MIC)198。

[0026] 本地存储器或本地储存器 (LS)163 至 170 是大存储器映射的非相干可寻址部分，其物理地可以作为连接到 SPU 140 至 154 的小存储器而提供。本地储存器 163 至 170 可以被映射到不同的地址空间。这些地址区域在非混叠配置中是连续的。本地储存器 163 至 170 通过其地址位置例如经由下文更详细讨论的 SPU 识别寄存器与其相应的 SPU 149 至 154 以及 SPE 120 至 134 相关联。只要本地储存器并不处于操作安全模式下，系统中的任何资源都有能力从 / 向本地储存器 163 至 170 读 / 写，在操作安全模式下仅本地储存器相关联的 SPU 可以访问本地储存器 163 至 170 或本地储存器 163 至 170 的指定的安全部分。

[0027] Cell/B.E.™处理器 100 可以是系统芯片，使得可以在单个微处理器芯片上提供图 1 中示出的每个单元。而且，Cell/B.E.™处理器 100 是异构处理环境，其中每个 SPU 可以从系统中的每个其他 SPU 接收不同指令。而且，SPU 的指令集与 PPU 的指令集不同，例如，PPU 可以执行基于精简指令集计算机 (RISC) 的指令而 SPU 执行矢量化指令。

[0028] SPE 120 至 134 经由 EIB 196 彼此连接并且连接到 L2 高速缓存 114。另外，SPE 120 至 134 经由 EIB 196 连接到 MIC 198 和 BIC 197。MIC 198 向共享存储器 199 提供通信接口。BIC 197 在 Cell/B.E.™处理器 100 和其他外部总线和装置之间提供通信接口。

[0029] PPE 110 是双线程 PPE 110。双线程 PPE 110 和八个 SPE 120 至 134 的组合使 Cell/B.E.™ 100 能够处理 10 个并发线程和超过 128 个未完成存储器请求。PPE 110 作为处理大部分计算工作量的其他八个 SPE 120 至 134 的控制器。例如 PPE 110 可以被用于运行传统操作系统而 SPE 120 至 134 执行矢量化浮点代码的执行。

[0030] SPE 120 至 134 包括协同处理单元 (SPU)140 至 154、存储器流控制 155 至 162、本地存储器或储存器 163 至 170 以及接口单元 180 至 194。在一个示例性实施例中，本地存储

器或储存器 163 至 170 包括 256KB 指令和数据存储器，其对于 PPE 110 可见并且能够被软件直接地寻址。

[0031] PPE 110 可以对 SPE 120 至 134 加载小程序或线程，将 SPE 链接到一起以处理复杂操作中的每个步骤。例如，结合 CBE 100 的机顶盒可以加载用于读 DVD、视频和音频解码以及显示的程序，并且数据可以从 SPE 传递给 SPE 直至其最终到达输出显示器。在 4GHz，每个 SPE 120 至 134 提供理论的 32GFlops 的性能，而 PPE 110 具有类似的性能水平。

[0032] 存储器流控制单元 (MFC) 155 至 162 作为 SPU 到系统其余和其他单元的接口。MFC 155 至 162 为主储存器和本地储存器 163 至 170 之间数据传送、保护和同步提供基本机制。逻辑上在处理器上每个 SPU 都有 MFC。一些实施例能够在多个 SPU 之间共享一个 MFC 的资源。在这种情况下，为 MFC 定义的全部工具和命令必须表现出独立于每个 SPU 的软件。共享 MFC 的效果被限于依赖于实现的工具和命令。

[0033] 图 2 示出了直接存储器存取传送技术。协同处理单元 (SPE) 210 执行直接存储器存取 (DMA) 传送以将数据从主存储器 250 加载到输入缓存 220。SPE 210 随后对缓存 220 中的数据段执行计算并且将结果存储在结果缓存 230 中。例如，SPE 210 对数据段 222 执行计算以产生结果数据段 232 并且对数据段 224 执行计算以产生结果数据段 234。输入缓存 220 和结果缓存 230 可以是与 SPE 210 相关联的本地储存器的部分。例如，SPE 210 可以是 SPE 120 至 134 之一，并且输入缓存 220 和结果缓存 230 包括在图 1 中的本地储存器 (LS) 163 至 170 之一中。

[0034] 在图 2 示出的例子中，SPE 210 随后执行 DMA 传送以将结果从结果缓存 230 传送到主存储器 250 中。SPE 210 可以将结果数据段从结果缓存 230 传送到在主存储器中的不同位置以便为下一步骤的计算准备数据。因此，SPE 210 可以执行多个较小 DMA 传送或者使用包括从结果缓存 230 到主存储器 250 的多个较小 DMA 传送的 DMA 列表执行一个 DMA 操作。在两种情况下都将使用相对较大的结果缓存。

[0035] Cell/B.E.™ 程序员必须考虑使用 LS 的效率，这大大影响应用的实际性能。如何有效地使用 LS 是 Cell/B.E.™ 程序员应当考虑的一个十分重要的问题。由于每个 LS 的空间有限，在主存储器和 LS 之间频繁往复地执行 DMA 传送是寻常的。因此，用于执行 DMA 传送的策略对应用性能来说可能是关键的。

[0036] 目前普遍使用的技术称为“双缓冲”。双缓冲的基本思想是通过在两个缓存之间切换计算而隐藏 DMA 传送延迟，由此当在其中一个缓存上执行 DMA 传送的同时在另一个缓存上执行计算。对于受计算限制的应用，此技术能够有效地隐藏 DMA 传送延迟。

[0037] 图 3 示出了使用双缓冲的直接存储器存取传送技术。SPE 310 执行 DMA 传送以将数据从主存储器 350 加载到输入缓存 320。SPE 310 随后对输入缓存 320 中的数据段执行计算。在示出的例子中，SPE 310 交替将结果存储到结果缓存 A 330 和结果缓存 B 340 中。例如，SPE 310 对数据段 322 执行计算以产生在输入缓存 A 330 中的结果数据段 332 并且对数据段 324 执行计算以产生在输入缓存 B 340 中的结果数据段 334。

[0038] 在图 3 示出的例子中，SPE 310 随后执行 DMA 传送以将结果从结果缓存 A 330 和结果缓存 B 340 存储到主存储器 350 中。SPE 310 可以将结果数据段从结果缓存 A 330 和结果缓存 B 340 存储到主存储器 350 的不同位置以便为下一步骤的计算准备数据。SPE 310 在结果缓存 A 330 和结果缓存 B 340 之间交替，对数据段执行计算并且将结果存储在一个

结果缓存中而同时执行 DMA 传送以存储来自另一结果缓存的结果数据段。例如，SPE 310 可以对数据段 324 执行计算并且将结果数据段 344 存储在结果缓存 B 340 中而同时执行 DMA 传送以将来自结果缓存 A 330 的结果数据段 332 存储到主存储器 350。随后，SPE 310 可以对下一数据段执行下一计算并且将结果存储在结果缓存 A 中而同时执行 DMA 传送将来自结果缓存 B 340 的结果数据段 344 存储到主存储器 350。由此，SPE 310 从结果缓存 A 330 和结果缓存 B 340 到主存储器执行多个较小 DMA 传送。

[0039] 有很多应用是受存储器限制的，并且在那些情况下应用的性能并不受限于每个 SPE 的计算能力，而是受限于 DMS 传送带宽。此外，双缓冲的一个缺点是 SPE 必须检查 DMA 传送的状态并且仅当 DMA 传送完成时才能重新使用缓存。这造成同步的额外开销。因此，根据示例性实施例，提供一种用于改进多个较小直接存储器存取传送的效率的机制。该机制可以用于双缓冲技术通常所应用的任何应用。

[0040] 图 4 示出了根据一个示例性实施例的使用临时结果缓存的直接存储器存取传送技术。SPE 410 包括输入和结果缓存 420 以及临时结果缓存 430。SPE 410 执行 DMA 传送以将数据从主存储器 450 加载到输入缓存 420。SPE 410 随后对输入缓存 420 中的数据段执行计算。在示出的例子中，SPE 410 对数据段 424 执行计算并且临时地将结果或中间结果存储在临时结果缓存 430 的结果数据段 434 中。临时结果缓存 430 可以是本地储存器的一小部分或在 SPE 410 中的一个或多个寄存器。SPE 410 随后将结果数据段 434 从临时结果缓存 430 复制回输入和结果缓存 420 中或对包含中间结果的数据段 434 执行另一计算并且将结果放入输入和结果缓存 420。

[0041] 在图 4 示出的例子中，SPE 410 随后执行 DMA 传送以将结果从输入和结果缓存 420 存储到主存储器 450 中。由于所实现的算法或为了将准备用于下一步骤的计算的数据，SPE 410 可以将结果数据段从输入和结果缓存 420 传送到主存储器 450 中的不同位置。SPE 410 可以对一个数据段如数据段 424 执行计算，并且将结果存储回输入和结果缓存 420，而同时执行 DMA 传送以将前一结果数据段如数据段 422 从输入和结果缓存 420 存储到主存储器 450。由此，SPE 410 可以从输入和结果缓存 420 到主存储器 450 执行多个较小 DMA 传送。

[0042] SPE 410 随后重复该过程直至整个输入缓存已经被处理为止。SPE410 在计算和直接存储器存取传送过程期间不执行直接存储器存取状态检查。SPE 410 仅当其需要重新使用输入和结果缓存 420 时才检查是否直接存储器存取传送已完成。SPE 410 在本地储存器内执行额外的复制操作；然而，在本地储存器内复制的开销小于对直接存储器存取传送状态进行检查并等待直接存储器存取传送完成。实际上，SPE 410 可以不必执行额外的复制，因为对于一些复杂计算，SPE 410 可以简单地将结果放回到输入缓存中而不是临时结果缓存中。因此，如图 4 示出的使用临时结果缓存的 DMA 传送不仅使用有限的较小的本地储存器空间，也带来更好的性能。

[0043] 本领域的普通技术人员可以理解，本发明可以作为系统、方法或计算机程序产品实现。相应地，本发明可以采取完全的硬件实施例、完全的软件实施例（包括固件、驻留软件、微码等）或在本说明书中一般地可以被称为“电路”、“模块”或“系统”的组合了软件和硬件方面的实施例的形式。进而，本发明可以采取在表达的任何有形介质中包含的计算机程序产品的形式，其具有在介质中包含的计算机可读程序代码。

[0044] 可以采用一个或多个计算机可用或计算机可读介质的任意组合。计算机可用或计

算机可读介质可以是，例如但不限于：电子的、磁的、光学的、电磁的、红外的或半导体的系统、装置、设备或传播介质。计算机可读介质的更多特定的例子（非穷尽的列举）可以包括：具有一个或多个接线的电连接、便携式计算机磁盘、硬盘、随机存取存储器（RAM）、只读存储器（ROM）、可擦除可编程只读存储器（EPROM 或闪存）、光纤、便携式压缩盘只读存储器（CDROM）、光学存储设备、传送媒体如那些支持因特网或内网的介质或磁存储设备。注意计算机可用或计算机可读介质甚至可以是纸或另一种适当的介质，其上印刷有程序，因为如果需要，程序可以经由例如对纸或其他介质的光学扫描被电子地获得，随后被编辑、翻译或以适当的方式做其他处理，随后被存储在计算机存储器中。在本说明书背景下，计算机可用或计算机可读介质可以是可以包含、存储、传送、传播或传输程序供指令执行系统、装置或设备使用或与其联合使用的任何介质。计算机可用介质可以包括传播数据信号，信号中在基带或载波的部分中包含计算机可用程序代码。计算机可用程序代码可以使用任何适当介质被传送，包括但不限于：无线、有线、光缆、射频（RF）等。

[0045] 用于实现本发明的操作的计算机程序代码可以被写成一种或更多程序设计语言的任意组合，包括面向对象的程序设计语言如 Java™、Smalltalk™、C++ 等以及传统的过程程序设计语言如“C”程序设计语言或类似的程序设计语言。程序代码可以完全地在用户计算机上执行、部分地在用户计算机上执行、作为独立软件包执行、部分地在用户计算机上并且部分地在远程计算机上执行或者完全地在远程计算机上或服务器上执行。在后一种情况下，远程计算机可以通过任何类型的网络（包括局域网（LAN）或广域网（WAN））连接到用户计算机，或者可以连接到外部计算机（例如，通过因特使用因特网服务提供商）。

[0046] 以下参照根据本发明的示例性实施例的方法、装置（系统）和计算机程序产品的流程图和/或方框图描述示例性实施例。可以理解，流程图和/或方框图中的每个框或流程图和/或方框图中的框的组合能够被计算机程序指令实现。这些计算机程序指令可以被提供给通用目的计算机、特定目的计算机或其他可编程数据处理装置的处理器以产生机器，使得指令在经由计算机或其他可编程数据处理装置的处理器执行时，产生用于实现流程图和/或方框图中的一个框或多个框所规定的功能/动作的装置。

[0047] 这些计算机程序指令也可以被存储在能够指导计算机或其他可编程数据处理装置以特定的方式发挥功能的计算机可读介质中，使得存储在计算机可读介质中的指令产生一种制品，其包括实现流程图和/或方框图中的一个框或多个框所规定的功能/动作的指令装置。

[0048] 计算机程序指令也可以被加载到计算机或其他可编程数据处理装置上，以使在计算机或其他可编程数据处理装置上执行一系列可操作的步骤，以产生计算机实现的过程，使得在计算机或其他可编程数据处理装置执行的指令提供用于实现流程图和/或方框图中的一个框或多个框所规定的功能/动作的过程。

[0049] 附图中的流程图和方框图示出根据本发明的各种实施例的系统、方法和计算机程序产品的可能实施方式的体系结构、功能性和操作。就此而言，流程图和/或方框图中每个框可以代表代码的模块、段或部分，其包括用于实现特定逻辑功能的一个或多个可执行指令。应当注意，在一些替代施例中，框中注明的功能可以按照与图示不同的顺序发生。例如，顺序示出的两个框根据涉及的功能性实际上可以基本上同时地执行，或者有时候可以以相反的顺序执行。也应当注意，流程图和/或方框图中的每个框，以及流程图和/或方框图中

的框的组合，能够通过执行特定的功能或动作的基于硬件的特定目的系统或特定目的硬件和计算机指令的组合实现。

[0050] 图 5A 和图 5B 示出了使用直接存储器存取传送的代码的例子，其中直接存储器存取传送使用双缓冲。图 5A 和图 5B 中示出的代码示出了用于 Cell/B.E.™ 体系结构的并行素因子分析快速傅立叶变换 (PFAFFT)。图 6 示出了根据一个示例性实施例的使用直接存储器存取传送的代码的例子，其中直接存储器存取传送使用临时缓存。图 6 所示的代码执行用于 Cell/B.E.™ 体系结构的同样的并行 PFAFFT 功能，而图 6 中的代码不仅更有效而且更简短。输入缓存越长并且小 DMA 传送数量越大，性能改进越大。

[0051] 图 7 是示出根据一个示例性实施例的各种矩阵大小和计算的性能改进的表。对于每种矩阵大小和计算，图 7 示出的表比较了用于使用双缓冲计算的时间以及相应的“新”时间，其中“新”时间使用例如图 4 示出的输入 / 结果缓存以及临时结果缓存。

[0052] 在 Cell/B.E.™ 数据处理系统上有两种 PFAFFT 的实现，它们使用不同的策略 ：一种是“3 步骤 PFAFFT”并且另一种是“2 步骤 PFAFFT”。而且，也有两种应用 PFAFFT 的情况 ：一种是采用矢量化格式的输入数据并且另一种是采用非矢量化格式的输入数据。对于后一种情况，PFAFFT 需要将输入数据从非矢量化格式转换为矢量化格式，随后执行 FFT 计算并且在 FFT 计算之后需要将输出数据从矢量化格式转换为非矢量化格式。“3SW”代表采用矢量化输入数据的 3 步骤 PFAFFT。“3SWO”代表不采用矢量化输入数据的 3 步骤 PFAFFT。“2SW”代表采用矢量化输入数据的 2 步骤 PFAFFT 并且类似地“2SWO”代表不采用矢量化输入数据的 2 步骤 PFAFFT。

[0053] 图 8 是示出根据一个示例性实施例采用临时结果缓存器的直接存储器存取传送操作的流程图。操作开始，该机制执行直接存储器存取 (DMA) 传送将数据从主存储器加载到输入缓存 (框 802)。该机制随后对输入缓存中的数据段执行计算 (框 804) 并且将结果存储在临时结果缓存中 (框 806)。随后，该机制将结果复制回输入缓存中 (框 808)。因此，该机制将输入缓存既作为输入缓存也作为结果缓存使用。在一个替代实施例中，或者当计算允许时，该机制也可以将结果直接地存回到输入 / 结果缓存中而不使用临时结果缓存。

[0054] 接下来，该机制将结果段传送到主存储器 (框 810)。该机制确定是否被传送的结果段是输入缓存中的最后一个段 (框 812)。如果该段不是最后一个段，操作回到框 804 以对下一个数据段执行计算。框 810 中的传送是 DMA 传送。因此，当框 810 中的 DMA 传送在执行的同时，该机制可以并行地执行下一个计算。仅对最后的计算，该机制必须检查回到主存储器的 DMA 传送的状态。

[0055] 如果在框 812 中，输入缓存中的段是最后一个段，该机制确定其是否需要重新使用输入缓存 (框 814)。如果该机制不需要重新使用输入缓存，操作结束。在框 814，如果该机制确定其需要重新使用输入缓存，该机制检查 DMA 传送的状态 (框 816) 并且确定是否从输入 / 结果缓存到主存储器的所有 DMA 传送已经完成 (框 818)。如果所有 DMA 传送已经完成，操作回到框 802 以执行从主存储器到输入缓存的下一个 DMA 传送。否则，操作回到框 816 检查 DMA 传送的状态。

[0056] 因此，示例性实施例提供用于改进多个较小直接存储器存取传送的效率的机制。该机制可以用于双缓冲技术通常所应用的任何应用。该机制使用一个输入缓存和小结果缓存或一些临时变量以临时地存储计算结果。该机制对输入缓存中的数据段执行计算并且将

结果存储在临时结果缓存中。该机制随后将结果复制回输入缓存中。这样，该机制将输入缓存既作为输入缓存也作为结果缓存使用。

[0057] 该机制随后对包含计算结果的输入缓存的段执行直接存储器存取传送并且随后对输入缓存的下一段执行计算（计算和直接存储器存取传送将被同时地执行）。该机制随后重复该过程直至整个输入缓存已经被处理为止。该机制在计算和直接存储器存取传送过程期间不执行直接存储器存取状态检查。该机制仅当其需要重新使用输入缓存时才检查是否直接存储器存取传送已完成。该机制在本地储存器内执行额外的复制操作；然而，在本地储存器内复制的开销小于对直接存储器存取传送状态进行检查并等待直接存储器存取传送完成。

[0058] 如上所述，应当理解，示例性实施例可以采取完全的硬件实施例、完全的软件实施例或既包括硬件单元也包括软件单元的实施例的形式。在一个实施例的例子中，示例性实施例的机制以软件或程序代码实现，其包括但不限于固件、驻留软件、微码等。

[0059] 适于存储和／或执行程序代码的数据处理系统将包括直接地或通过系统总线间接地连接到存储器单元的至少一个处理器。存储器单元可以包括在程序代码实际执行期间利用的本地存储器、大容量存储器以及高速缓存，高速缓存是为了降低在执行期间必须从大容量缓存检索代码的次数而为至少一些程序代码提供的临时存储空间。

[0060] 输入／输出或I/O设备（包括但不限于键盘、显示器、指示设备等）能够直接地或通过居间I/O控制器连接到系统。网络适配器也可以连接到系统以允许数据处理系统通过居间私人或公共网络连接到其他数据处理系统或远程打印机或存储设备。调制解调器、线缆调制解调器以及以太网卡只是目前可用的网络适配器类型中的几种。

[0061] 对本发明的描述是为了解释和描述之目的而提供，并不意在以所公开的形式穷尽或限制本发明。本领域的普通技术人员可以理解诸多变更和变型。对实施例的选择和描述是为了对本发明原则、实际应用做出最佳解释，并且为了使本领域的普通技术人员理解对本发明各种实施例的适于所预期之特定用途的各种变更。

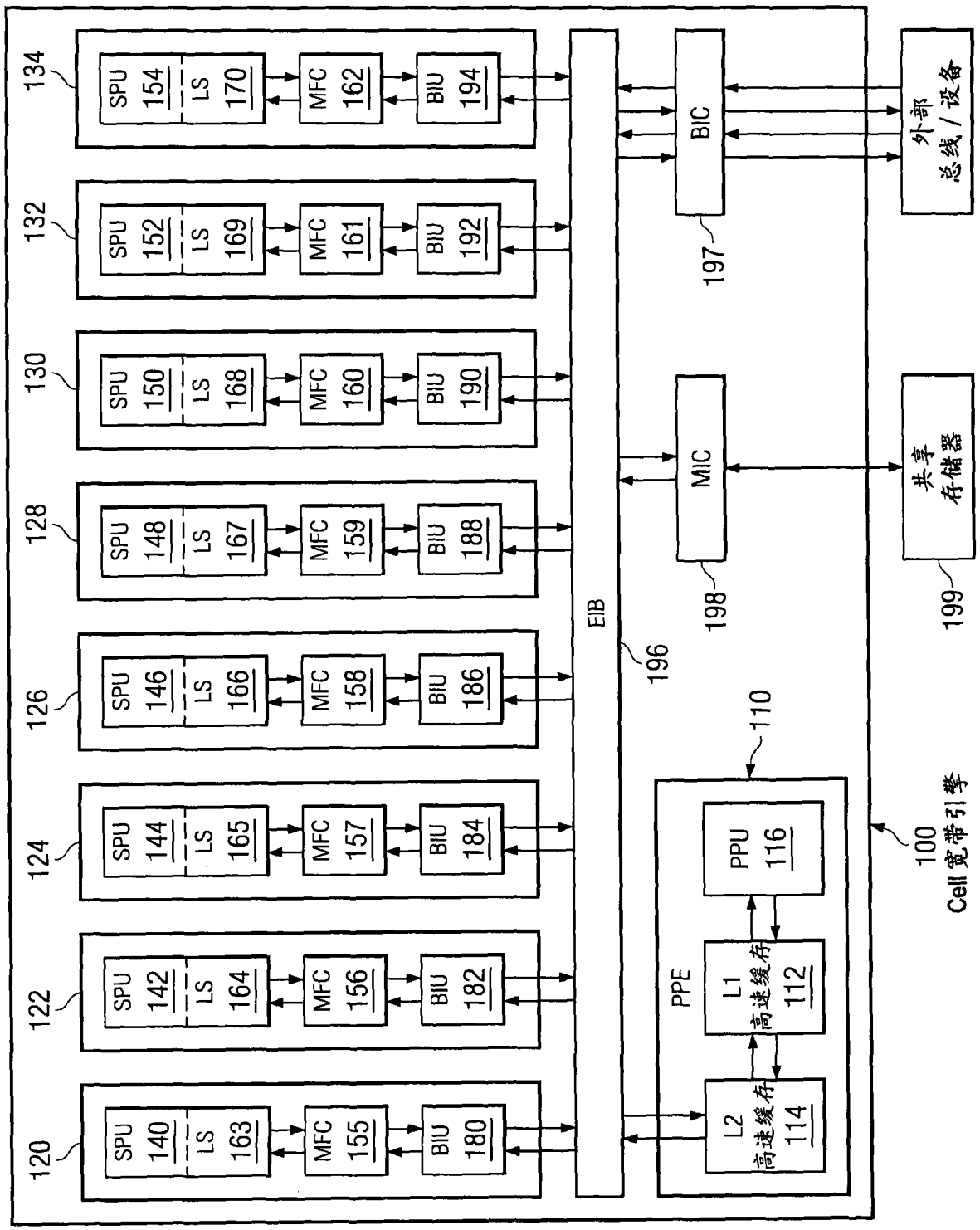

Cell 宽带引擎
100
PPE
PPU 116
L1 高速缓存 112
L2 高速缓存 114
110
EIB
196
BIC
197
外部总线 / 设备
MIC
198
共享存储器
199
120 SPU 140 LS 163 MFC 155 BIU 180
122 SPU 142 LS 164 MFC 156 BIU 182
124 SPU 144 LS 165 MFC 157 BIU 184
126 SPU 146 LS 166 MFC 158 BIU 186
128 SPU 148 LS 167 MFC 159 BIU 188
130 SPU 150 LS 168 MFC 160 BIU 190
132 SPU 152 LS 169 MFC 161 BIU 192
134 SPU 154 LS 170 MFC 162 BIU 194

图 1

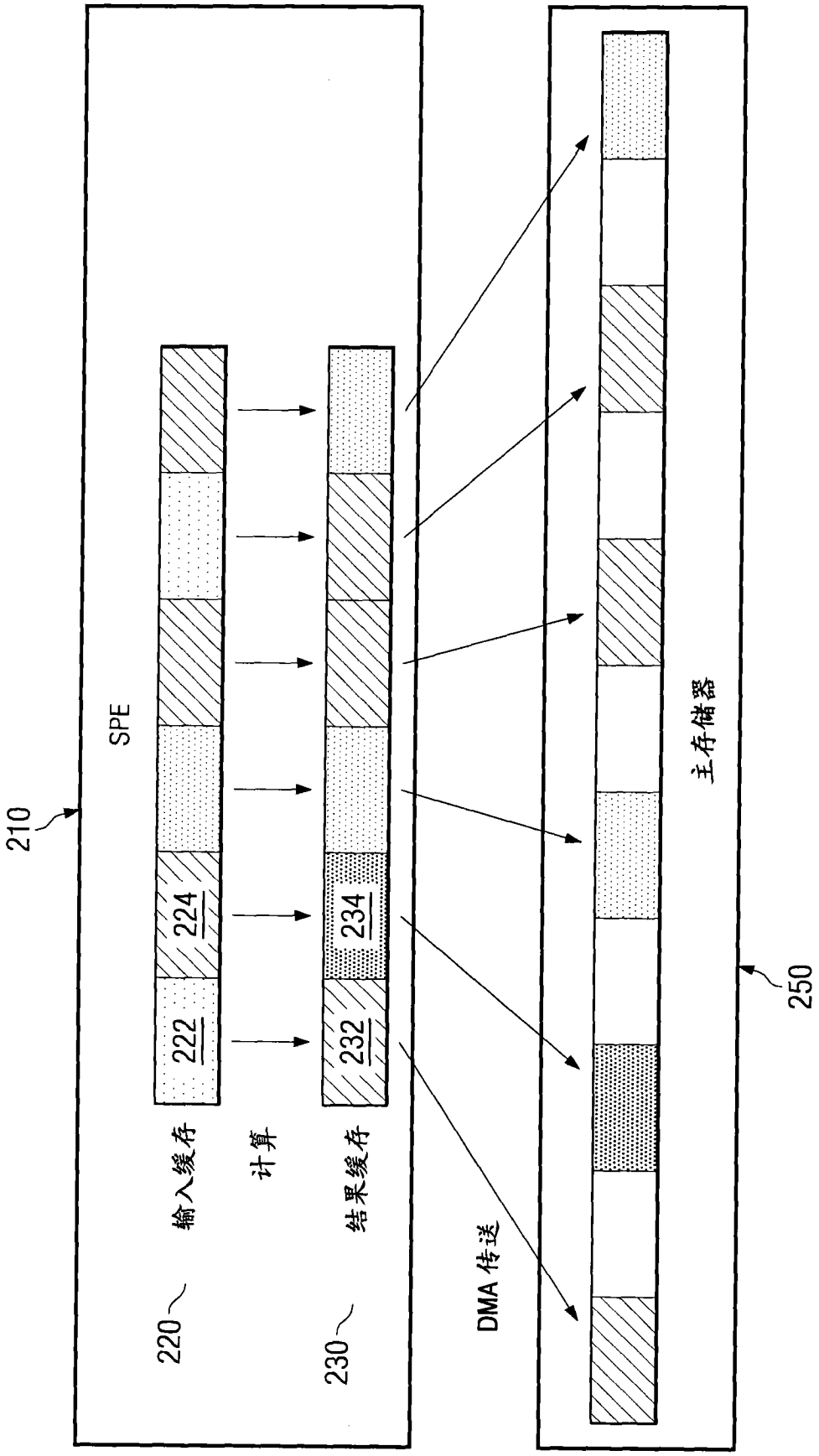


图 2

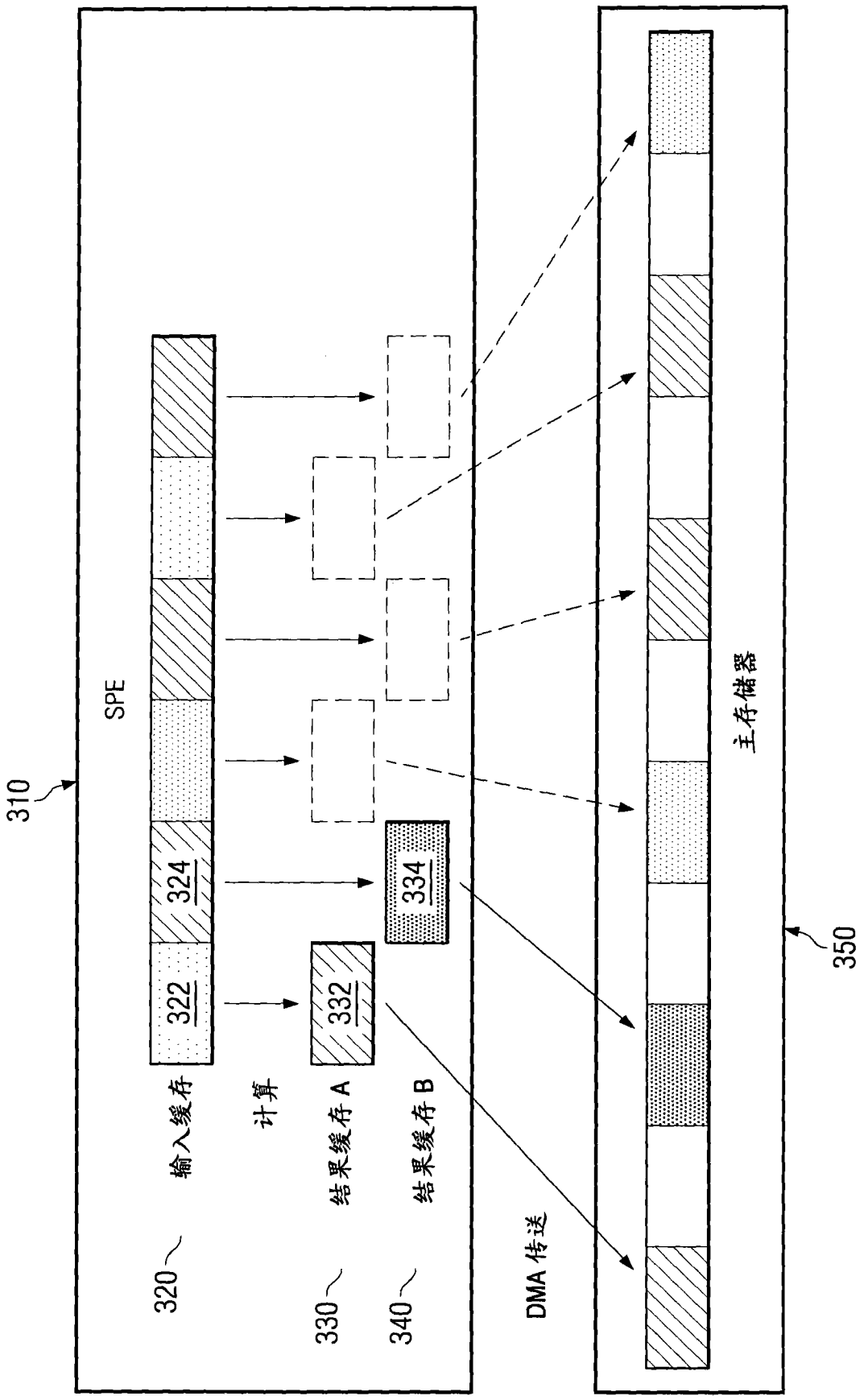

310
SPE
322
324
334
332
输入缓存
计算
结果缓存 A
结果缓存 B
320
330
340
DMA 传送
主存储器
350

图 3

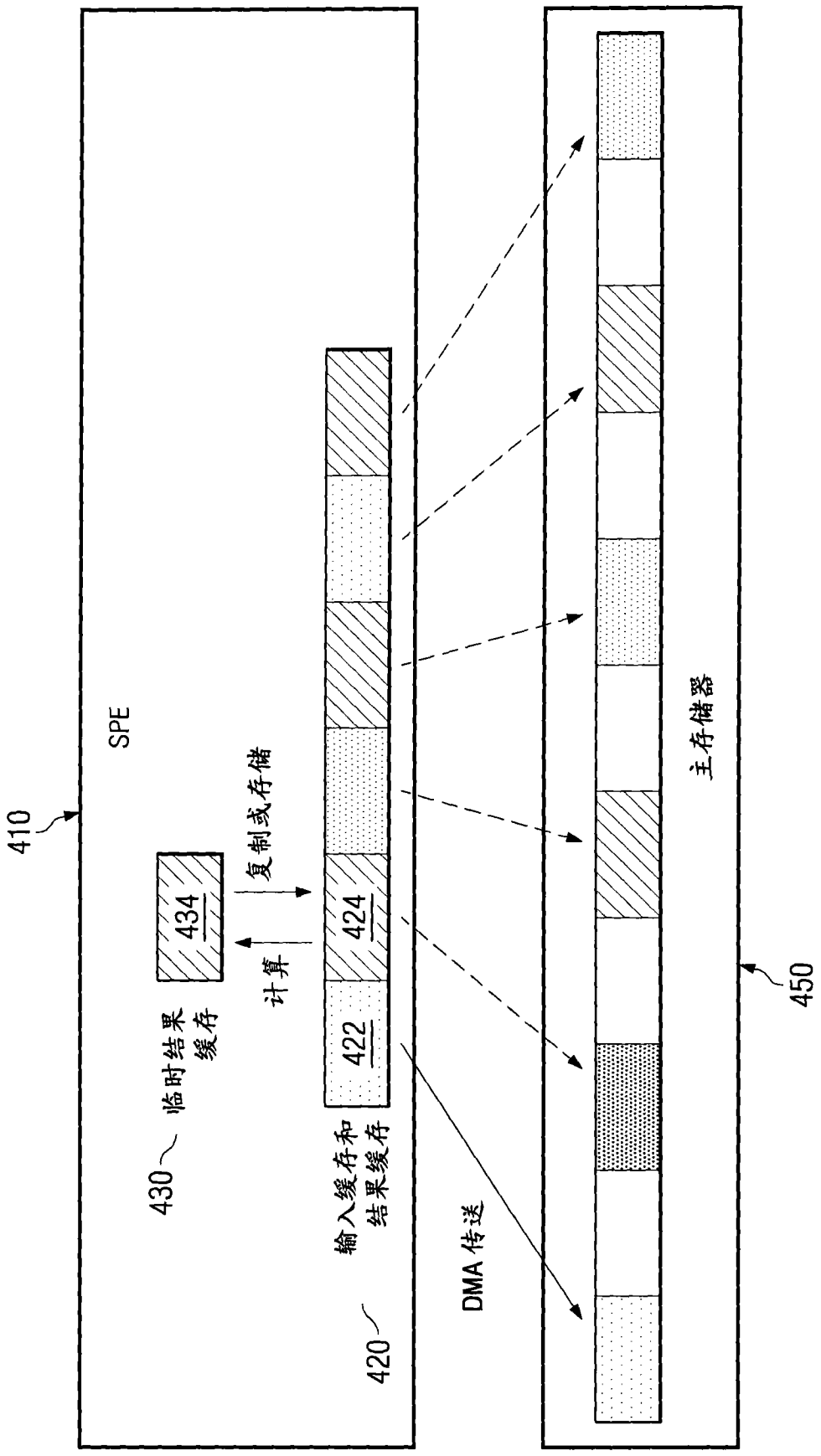

410
SPE
434
复制或存储
计算
424
422
临时结果缓存
430
输入缓存和结果缓存
420
DMA 传送
主存储器
450

图 4

```
ibuf = 0;
  ftme = 1;
  ii = 0;
  jj = 0;
  abuf = tbuf0;
    for(j=0; j<rlen; j+=8)  {
     /* Real */
     wk1 = spu_shuffle(z[j], z[j+2],pshf01);
     wk2 = spu_shuffle(z[j+4],z[j+6],pshf01);
     wk3 = spu_shuffle(z[j], z[j+2],pshf02);
     wk4 = spu_shuffle(z[j+4],z[j+6],pshf02);
     abuf[jj] = spu_shuffle(wk1,wk2,pshf03);
     abuf[jj+2] = spu_shuffle(wk1,wk2,pshf04);
     abuf[jj+4] = spu_shuffle(wk3,wk4,pshf03);
     abuf[jj+6] = spu_shuffle(wk3,wk4,pshf04);
     /* Imaginary */
     wk1 = spu_shuffle(z[j+1],z[j+3],pshf01);
     wk2 = spu_shuffle(z[j+5],z[j+7],pshf01);
     wk3 = spu_shuffle(z[j+1],z[j+3],pshf02);
     wk4 = spu_shuffle(z[j+5],z[j+7],pshf02);
     abuf[jj+1] = spu_shuffle(wk1,wk2,pshf03);
     abuf[jj+3] = spu_shuffle(wk1,wk2,pshf04);
     abuf[jj+5] = spu_shuffle(wk3,wk4,pshf03);
     abuf[jj+7] = spu_shuffle(wk3,wk4,pshf04);
     /* List PUT COLSEG column sections */
     ii++;
     if(ii == COLSEG) { // Time to put COLSEG column sections
     if(ftme){ // Don't wait first time, to overlap the PUT
       ftme = 0;
     } else {
       mfc_write_tag_mask(1<<26);
       mfc_read_tag_status_all();
     }
     dmalst[ii-1] = (unsigned int) curz2+coset+(128*(dbl-1))+(bsize*ibuf);
     /* Put COLSEG column slices */
     for(i=0; i<COLSEG; i++) {
       mfc_put(abuf+(8*i), dmalst[i], 128, 26, 0, 0);
     }i
     f(abuf == tbuf0) { // Swap buffers
       abuf = tbuf1;
     } else {
       abuf = tbuf0;
     }
     jj = 0;
```

图 5A

```
        } else {
          mfc_write_tag_mask(1<<26);
          mfc_read_tag_status_all();
        }
        dmalst[ii-1] = (unsigned int) curz2+coset+(128*(dbl-1))+(bsize*ibuf);
        /* Put COLSEG column slices */
        for(i=0; i<COLSEG; i++) {
          mfc_put(abuf+(8*i), dmalst[i], 128, 26, 0, 0);
        }i
        f(abuf == tbuf0) { // Swap buffers
          abuf = tbuf1;
        } else {
          abuf = tbuf0;
        }
        jj = 0;
        ii = 0;
      } else { // Build another list entry
        jj += 8;
        dmalst[ii-1] = (unsigned int) curz2+coset+(128*(dbl-1))+(bsize*ibuf);
      }i
      buf++;
  } // End of loop through one vector array
      /* Wait on the last put in the loop (if any) */
      mfc_write_tag_mask(1<<26);
      mfc_read_tag_status_all();
  /* Put any remaining column sections */
  if(ii) {
      for(i=0; i<ii; i++) {
        mfc_put(abuf+(8*i), dmalst[i], 128, 26, 0, 0);
      }
      mfc_write_tag_mask(1<<26);
      mfc_read_tag_status_all();
}
```

图 5B

```
ii = 0;
outdmaadr = (unsigned int) curz2+coset+128*(dbl-1);
for(j=0; j<rlen; j+=8) {
   /* Real */
   wk1 = spu_shuffle(z[j], z[j+2],pshf01);
   wk2 = spu_shuffle(z[j+4],z[j+6],pshf01);
   wk3 = spu_shuffle(z[j], z[j+2],pshf02);
   wk4 = spu_shuffle(z[j+4],z[j+6],pshf02);
   wk5 = spu_shuffle(z[j+1],z[j+3],pshf01);
   wk6 = spu_shuffle(z[j+5],z[j+7],pshf01);
   wk7 = spu_shuffle(z[j+1],z[j+3],pshf02);
   wk8 = spu_shuffle(z[j+5],z[j+7],pshf02);
   z[j] = spu_shuffle(wk1,wk2,pshf03);
   z[j+1] = spu_shuffle(wk5,wk6,pshf03);
   z[j+2] = spu_shuffle(wk1,wk2,pshf04);
   z[j+3] = spu_shuffle(wk5,wk6,pshf04);
   z[j+4] = spu_shuffle(wk3,wk4,pshf03);
   z[j+5] = spu_shuffle(wk7,wk8,pshf03);
   z[j+6] = spu_shuffle(wk3,wk4,pshf04);
   z[j+7] = spu_shuffle(wk7,wk8,pshf04);
   /* List PUT COLSEG column sections */
   ii++;
   if(ii == COLSEG) { // Time to put COLSEG column sections
     dmalst[ii-1] = outdmaadr;

     /* Put COLSEG column slices */
     for(i=0; i<COLSEG; i++){
       mfc_put(z+(8*i+j-56), dmalst[i], 128, 26, 0, 0);
     }
     ii = 0;
   } else { // Build another list entry
     dmalst[ii-1] = outdmaadr;
   }
   outdmaadr += bsize;
} // End of loop through one vector array

/* Put any remaining column sections */
if (ii) {
   for (i=0; i<ii; i++){
     mfc_put(z+(8*(i-ii)+j), dmalst[i], 128, 26, 0, 0);
   }
}
   } else { // Build another list entry
     dmalst[ii-1] = outdmaadr;
   }
   outdmaadr += bsize;
} // End of loop through one vector array

/* Put any remaining column sections */
if (ii) {
   for (i=0; i<ii; i++) {
     mfc_put(z+(8*(i-ii)+j), dmalst[i], 128, 26, 0, 0);
   }
}
```

图 6

矩阵大小	3SW	新3SW	3SWO	新3SWO	2SW	新2SW	2SWO	新2SWO
364x240	6.63	6.18	4.74	4.42	5.56	5.46	5.31	5.22
616x308	11.86	11.14	8.21	7.58	9.5	9.29	9.05	8.91
840x462	24.3	22.87	16.96	15.73	18.71	18.26	17.83	17.34
1008x616	34.72	32.19	23.07	21.12	27.58	27.05	26.29	25.91
1260x840	59.71	55.58	39.94	36.62	50.84	50.06	48.38	47.16
1540x1008	86.16	80.26	57.65	52.86	79.1	77.44	75.66	73.68

图 7

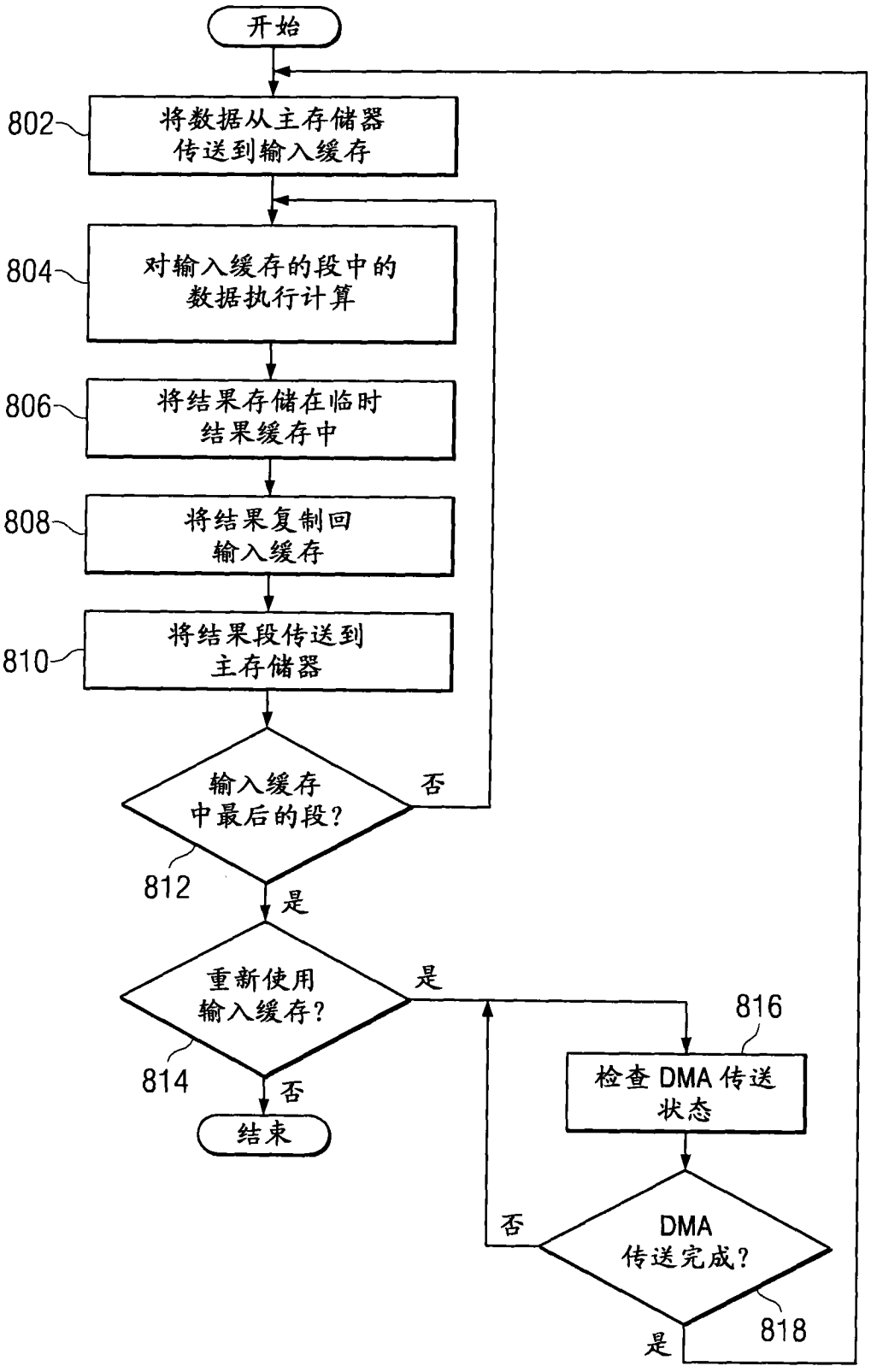

开始
802
将数据从主存储器传送到输入缓存
804
对输入缓存的段中的数据执行计算
806
将结果存储在临时结果缓存中
808
将结果复制回输入缓存
810
将结果段传送到主存储器
输入缓存中最后的段?
否
812
是
重新使用输入缓存?
是
814
否
结束
816
检查 DMA 传送状态
否
DMA 传送完成?
818
是

图 8