(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 110262995 A
(43)申请公布日 2019.09.20

(21)申请号 201910633638.1

(22)申请日 2019.07.15

(71)申请人 北京一流科技有限公司
地址 100083 北京市海淀区王庄路1号B座6层7-C

(72)发明人 李新奇 牛冲 袁进辉

(74)专利代理机构 北京金讯知识产权代理事务所(特殊普通合伙) 11554
代理人 黄剑飞

(51)Int.Cl.
G06F 15/173(2006.01)
G06N 3/04(2006.01)
G06N 3/063(2006.01)

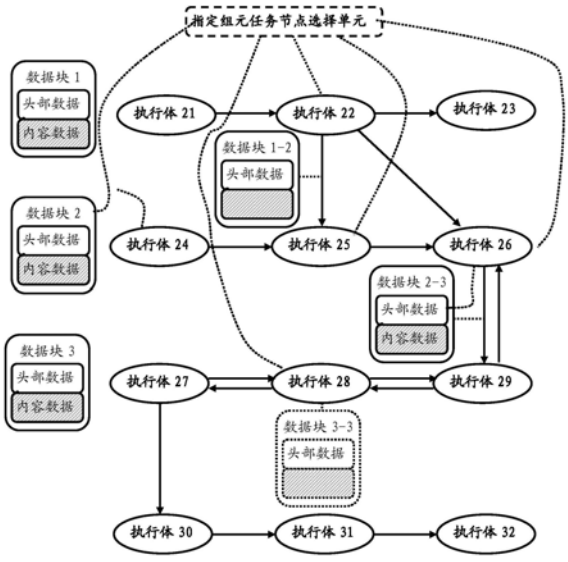
权利要求书1页 说明书13页 附图3页

(54)发明名称

执行体创建系统和执行体创建方法

(57)摘要

本公开公开了一种执行体创建系统，包括：任务拓扑图生成组件，用于将所需完成的作业分解为一系列由异构架构中执行体来执行的任务，并在作业分解的同时，基于所分解的任务之间的固有关系，生成任务关系拓扑图，所述任务拓扑图的任务节点含有执行对应任务所需的全部节点属性，所述任务拓扑图生成组件包括指定组元节点选择单元，用于在生成任务关系拓扑图过程中根据具体任务选择具有包含附加节点属性的指定组元的具体任务节点，所述附加节点属性改变该具体任务节点执行对应任务的方式；以及执行体创建组件，基于任务关系拓扑图以及每个任务节点的全部节点属性或含有指定组元的全部节点属性在计算资源中为每个任务节点创建对应的执行体，并使得与有附加节点属性的具体任务节点对应的具体执行体按照附加节点属性执行对应的任务。



CN 110262995 A

1.一种执行体创建系统,包括:

任务拓扑图生成组件,用于将所需完成的作业分解为一系列由异构架构中执行体来执行的任务,并在作业分解的同时,基于所分解的任务之间的固有关系,生成任务关系拓扑图,所述任务拓扑图的任务节点含有执行对应任务所需的全部节点属性,所述任务拓扑图生成组件包括指定组元节点选择单元,用于在生成任务关系拓扑图过程中根据具体任务选择具有包含附加节点属性的指定组元的具体任务节点,所述附加节点属性改变该具体任务节点执行对应任务的方式;以及

执行体创建组件,基于任务关系拓扑图以及每个任务节点的全部节点属性或含有指定组元的全部节点属性在计算资源中为每个任务节点创建对应的执行体,并使得与有附加节点属性的具体任务节点对应的具体执行体按照附加节点属性执行对应的任务。

2.如权利要求1所述的执行体创建系统,其中所述指定组元包含的附加节点属性为数据操作节点属性,用于使得所创建的具体执行体仅仅对被处理数据的头部数据执行操作。

3.如权利要求1所述的执行体创建系统,其中所述指定组元包含的附加节点属性为数据存储节点属性,用于使得所创建的具体执行体仅仅存储被处理数据的头部数据。

4.如权利要求1所述的执行体创建系统,其中所述指定组元包含的附加节点属性为数据修改节点属性,用于使得所创建的具体执行体能够修改被处理数据的头部数据。

5.一种执行体创建方法,包括:

将所需完成的作业分解为一系列由异构架构中执行体来执行的任务,并在作业分解的同时,基于所分解的任务之间的固有关系,生成任务关系拓扑图,所述任务拓扑图的任务节点含有执行对应任务所需的全部节点属性;

在生成任务关系拓扑图过程中,根据具体任务选择具有包含附加节点属性的指定组元的具体任务节点,所述附加节点属性改变该具体任务节点执行对应任务的方式;以及

基于任务关系拓扑图以及每个任务节点的全部节点属性或含有指定组元的全部节点属性在计算资源中为每个任务节点创建对应的执行体,并使得与有附加节点属性的具体任务节点对应的具体执行体按照附加节点属性执行对应的任务。

6.如权利要求5所述的执行体创建方法,其中所述指定组元包含的附加节点属性为数据操作节点属性,用于使得所创建的具体执行体仅仅对被处理数据的头部数据执行操作。

7.如权利要求5所述的执行体创建方法,其中所述指定组元包含的附加节点属性为数据存储节点属性,用于使得所创建的具体执行体仅仅存储被处理数据的头部数据。

8.如权利要求5所述的执行体创建方法,其中所述指定组元包含的附加节点属性为数据修改节点属性,用于使得所创建的具体执行体能够修改被处理数据的头部数据。

执行体创建系统和执行体创建方法

技术领域

[0001] 本公开涉及一种对数据处理网络中存储空间进行精细化管理和硬件优化的系统和方法,更具体而言,涉及一种数据处理网络中执行体创建系统和执行体创建方法。

背景技术

[0002] 随着机器学习的发展以及人工神经网络的研究的逐步深入,深度学习的概念得到广泛的关注和应用。深度学习是一种特殊的机器学习,它采用网状层级结构来表达所学习的对象,通过简单概念来组合成抽象概念,通过简单概念计算来实现抽象的概念表达。目前,深度学习在图像识别、语音识别以及自然语言处理领域有了长足的进展。深度学习涉及的模型参数多,导致计算量巨大,而且训练数据的规模大,因此需要消耗较多的计算资源等特点。

[0003] 随着大数据计算以及深度学习的兴起,各种协处理器通常被用于分担CPU的数据处理功能。例如GPU(Graphic Processing Unit)、APU等。GPU具有高并行结构(highly parallel structure),所以GPU在处理图形数据和复杂算法方面拥有比CPU更高的效率。CPU执行计算任务时,一个时刻只处理一个数据,不存在真正意义上的并行,而GPU具有多个处理器核,在一个时刻可以并行处理多个数据。与CPU相比,GPU拥有更多的ALU(Arithmetic Logic Unit,逻辑运算执行体)用于数据处理,而非数据高速缓存和流控制。这样的结构非常适合于对于类型高度统一的、相互无依赖的大规模数据和不需要被打断的纯净的计算环境。

[0004] 现有的大数据计算和深度学习网络系统通过预定各个运算执行体的操作功能来进行连续的数据处理,因此,网络一旦启动进入运行期,执行体就需要依据预定操作对数据块进行操作和处理,并且执行体运行以及数据块的操作都固定不变。这样会导致在某些执行体处的数据块的处理过程中的资源浪费,甚至会导致操作过程由于上下游关系对数据块的空置过慢影响下游执行体的操作。

[0005] 因此,人们希望有一种数据处理网络能够对特定的执行体和数据块之间的关系进行调节,从而提高整体数据处理网络的数据处理速度以及提高执行体的重复利用率。

发明内容

[0006] 由于大数据以及深度学习由于所处理的数据具有类型高度统一型,因此为提供一种能够消除现有技术中存在的上述问题提供了可能。本公开的目的在于提供一种执行体创建系统,包括:任务拓扑图生成组件,用于将所需完成的作业分解为一系列由异构架构中执行体来执行的任务,并在作业分解的同时,基于所分解的任务之间的固有关系,生成任务关系拓扑图,所述任务拓扑图的任务节点含有执行对应任务所需的全部节点属性,所述任务拓扑图生成组件包括指定组元节点选择单元,用于在生成任务关系拓扑图过程中根据具体任务选择具有包含附加节点属性的指定组元的具体任务节点,所述附加节点属性改变该具体任务节点执行对应任务的方式;以及执行体创建组件,基于任务关系拓扑图以及每个任

务节点的全部节点属性或含有指定组元的全部节点属性在计算资源中为每个任务节点创建对应的执行体，并使得与有附加节点属性的具体任务节点对应的具体执行体按照附加节点属性执行对应的任务。

[0007] 根据本公开的执行体创建系统，其中所述指定组元包含的附加节点属性为数据操作节点属性，用于使得所创建的具体执行体仅仅对被处理数据的头部数据执行操作。

[0008] 根据本公开的执行体创建系统，其中所述指定组元包含的附加节点属性为数据存储节点属性，用于使得所创建的具体执行体仅仅存储被处理数据的头部数据。

[0009] 根据本公开的执行体创建系统，其中所述指定组元包含的附加节点属性为数据修改节点属性，用于使得所创建的具体执行体能够修改被处理数据的头部数据。

[0010] 根据本公开的另一个方面，提供了一种执行体创建方法，包括：将所需完成的作业分解为一系列由异构架构中执行体来执行的任务，并在作业分解的同时，基于所分解的任务之间的固有关系，生成任务关系拓扑图，所述任务拓扑图的任务节点含有执行对应任务所需的全部节点属性；在生成任务关系拓扑图过程中，根据具体任务选择具有包含附加节点属性的指定组元的具体任务节点，所述附加节点属性改变该具体任务节点执行对应任务的方式；以及基于任务关系拓扑图以及每个任务节点的全部节点属性或含有指定组元的全部节点属性在计算资源中为每个任务节点创建对应的执行体，并使得与有附加节点属性的具体任务节点对应的具体执行体按照附加节点属性执行对应的任务。

[0011] 根据本公开的执行体创建方法，其中所述指定组元包含的附加节点属性为数据操作节点属性，用于使得所创建的具体执行体仅仅对被处理数据的头部数据执行操作。

[0012] 根据本公开的执行体创建方法，其中所述指定组元包含的附加节点属性为数据存储节点属性，用于使得所创建的具体执行体仅仅存储被处理数据的头部数据。

[0013] 根据本公开的执行体创建方法，其中所述指定组元包含的附加节点属性为数据修改节点属性，用于使得所创建的具体执行体能够修改被处理数据的头部数据。

[0014] 通过指定组元节点选择单元选择具有包含附加节点属性的指定组元的具体任务节点，能够使得基于包含节点附加属性的全部节点属性创建的执行体对数据块的操作范围、操作方式以及存储方式做出对应于节点属性的变化，使得执行体具有更为灵活的操作功能，从而实现静态数据处理网络中对存储空间的精细化管理，并使得各个节点的执行体硬件得到优化。通过指定组元节点选择单元选择具有指定组元所包含加节点属性的具体任务节点使得对应的具体执行体可以对数据块的头部数据进行修改，从而使得执行体能够超出数据块本身的固有属性对数据块进行更丰富多样的操作处理，从而能够扩展数据处理网络对数据块的多样化处理要求。

[0015] 本公开的其它优点、目标和特征的一部分将通过下面的说明得以体现，另一部分将通过对本公开的研究和实践而为本领域的技术人员所理解。

附图说明

[0016] 图1所示的是根据本公开的用于数据处理网络中的执行体创建系统的结构示意图；

[0017] 图2所示的是根据本公开的数据处理网络中执行体创建系统中指定组元节点选择单元对具体执行体影响的原理示意图。

[0018] 图3为所示的是根据本公开的数据处理网络中的执行体创建系统所创建的执行体网络的结构示意图。

具体实施方式

[0019] 下面结合实施例和附图对本公开做进一步的详细说明，以令本领域技术人员参照说明书文字能够据以实施。

[0020] 这里将详细地对示例性实施例进行说明，其示例表示在附图中。下面的描述涉及附图时，除非另有表示，不同附图中的相同数字表示相同或相似的要素。以下示例性实施例中所描述的实施方式并不代表与本公开相一致的所有实施方式。相反，它们仅是与如所附权利要求书中所详述的、本公开的一些方面相一致的装置和方法的例子。

[0021] 在本公开使用的术语是仅仅出于描述特定实施例的目的，而非旨在限制本开。在本公开和所附权利要求书中所使用的单数形式的“一种”、“所述”和“该”也旨在包括多数形式，除非上下文清楚地表示其他含义。还应当理解，本文中使用的术语“和/或”是指并包含一个或多个相关联的列出项目的任何或所有可能组合。

[0022] 应当理解，尽管在本公开可能采用术语第一、第二、第三等来描述各种信息，但这些信息不应限于这些术语。这些术语仅用来将同一类型的信息彼此区分开。例如，在不脱离本公开范围的情况下，在下文中，两个可能设备之一可以被称为第一执行体也可以被称为第二执行体，类似地，两个可能设备的另一个可以被称为第二执行体也可以被称为第一执行体。取决于语境，如在此所使用的词语“如果”可以被解释成为“在……时”或“当……时”或“响应于确定”。

[0023] 为了使本领域技术人员更好地理解本公开，下面结合附图和具体实施方式对本公开作进一步详细说明。

[0024] 图1所示的是根据本公开的用于数据处理网络中的执行体创建系统的结构示意图。如图1所示，执行体创建系统布置在异构架构，该异构架构由CPU00和CPU01分别与CPU00相连的GPU00、GPU01以及GPU2和与CPU01相连的GPU10、GPU11以及GPU12构成。尽管此处仅仅显示了两个CPU和六个GPU，但是异构架构可以包括更多的CPU，并且与每个CPU相连的GPU也可以更多或更少，这可基于实际需要来定。

[0025] 根据本公开的执行体创建系统100部署在图1所示的异构架构中。尽管图1显示的执行体创建系统的构成部分单独显示在各个CPU和GPU之外，这是为了突出显示和方便描述进行的处理。实际上该执行体创建系统的构成部分都分布在CPU和/或GPU之中。

[0026] 如图1所示，所述执行体创建系统100包括任务拓扑图生成组件120以及执行体创建组件130。执行体网络组件140是执行体创建组件130的创建结果网络组件。

[0027] 如图1所示，任务拓扑图生成组件120用于将所需完成的作业分解为一系列由异构架构中执行体来执行的任务，并在进行作业分解的同时，基于所分解的任务之间的固有关系，生成任务关系拓扑图。执行体创建系统100是为了处理预定类型的作业数据而设立的，为了连续对同类型的数据进行连续的处理，需要将作业分解成为适合CPU或GPU的运算单元执行运算或其他操作的简单任务。具体而言，就是将作业分解成为彼此相关联的任务。所述任务拓扑图生成组件120对作业的分解包括对数据块的分解以及对数据处理模型的分解，即对作业的分解是设置对将要处理的作业数据的分解方式。具体而言，按照对作业任务要

求的描述，对作业按照将要处理的过程，进行分层分解成多层神经网络结构。一个作业(Job)被分解成一系列互相依赖的任务(Task)，这种依赖关系通常用有向无环图(Directed acyclic graph,DAG)来描述，每个节点表示一个任务，节点之间的连接线表示一个数据依赖关系(生产者和消费者关系)。在此不具体描述作业分解后任务关系图的情形。

[0028] 在逐步分解作业的同时，任务拓扑图生成组件120也逐层形成任务关系拓扑图。由于作业分解形成的各个任务之间的存在固有的逻辑关系，因此，随着作业被分解为不同的任务，在不同的任务层上，任务拓扑图生成组件120也随之形成任务关系拓扑图，这些任务关系拓扑图形成了所分解的任务之间的神经网络。在作业复杂的情况下，任务关系拓扑图包含有多层，因此也形成了多层任务神经网络。每层神经网络既包含对应具体任务的神经元节点，也包含各个神经元之间关系，而且既包含针对未来将被用于分片数据的处理的任务的数据并行网络，也包含将被用于分片模型的任务的模型并行网络。可选择地，这些神经网络中也可仅仅包括数据并行网络。是否同时包含数据并行网络与模型并行网络，可以根据实际需要进行配置。

[0029] 为了在随后的执行体创建组件一次性对任务拓扑图的任意节点创建执行体，根据本公开的任务拓扑图生成组件120在生成任务拓扑图的每个节点同时，赋予每个节点执行对应任务所需的全部节点属性。所述全部节点属性包含了诸如指明节点所对应的任务所需的资源的资源属性以及触发任务执行的触发条件的条件属性等等。所述全部节点属性也包含了后面将提到的附加节点属性。正是由于本公开的任务拓扑图中的每个节点包含了全部节点属性，因此其在随后创建执行体时立即自动具备了执行任务所有的资源和所有属性，处于完全配置状态，不需要在对具体数据执行具体任务时进行诸如对环境资源进行动态分配以及动态配置触发条件等等。对于基于本公开的任务拓扑图以及含有全部节点属性的节点所创建的每个执行体而言，其在对具体数据进行处理过程中，本身处于静态状态，变化的仅仅是输入数据的不同。现有用于深度学习的执行体创建系统的神经网络的节点所包含的节点属性非常少或基本没有，因此节点对应任务的执行需要在具体任务执行过程中临时推导出所需的属性从而动态获取相应的属性来完成对应的任务。并且这种临时推导的属性针对同样的任务每次都需要临时推导，因此会需要大量的运算开销。

[0030] 需要指出的是，任务拓扑图生成组件120存在逐层形成任务关系拓扑图同时，需要对已经形成的任务关系拓扑图进行优化。因此根据本公开的任务拓扑图生成组件120还包括拓扑图优化组件121。所述拓扑图优化组件121包括各种优化单元，例如冗余节点消除单元1211、阻塞节点消除单元1212以及指定组元节点选择单元这样的等效子图变换单元1213以及其他用于优化拓扑图的单元1214。尽管在本公开的图1中显示包含了上述三个单元，但是并不表示本公开必须都包含这些单元。本公开的实现并不需要一定包含上述拓扑优化组件121。拓扑优化组件121的存在，将使得任务拓扑图生成组件120所生成的任务拓扑图更为合理，并且将在随后的数据处理过程中运行的更为流畅，处理效率更高。

[0031] 具体而言，在任务拓扑图生成组件120生成任务拓扑图的过程中，可能会存在针对某一任务，出现重复生成对应节点的情况。例如，在一个神经网络子图中，可能会出现两个并列的节点，两者具有相同的上游节点和相同的下游节点，并且对应同一个任务。这样的节点就是冗余节点情形。这种冗余节点的存在会重复消耗异构架构中的运算资源，使得神经网络复杂化。因此这种冗余节点是需要去掉的节点。在任务拓扑图生成组件120生成任务拓

扑图的过程中如果发现了这种重复节点，冗余节点消除单元1211会获知这种节点的存在并直接删除该冗余节点，使得与该冗余节点的上下游节点仅仅关联到与给被删除冗余节点相同的节点(与冗余节点执行相同任务的节点)的上下游节点。此外，在任务拓扑图生成组件120生成任务拓扑图的过程中，可能会存在针对某些任务彼此之间的交互，会由于任务处理不及时出现下游节点堵塞情况，从而将导致被阻塞的节点的阻塞情况向前传导。为此，在任务拓扑图生成组件120生成任务拓扑图的过程中如果发现了这种阻塞节点，阻塞节点消除单元1212会消除任务拓扑图中的造成运算阻塞的节点。具体而言，就是改变阻塞节点与上游节点之间的连接边，增加一个或多个节点，消除阻塞节点处阻塞对上游的传导。

[0032] 在任务拓扑图生成组件120生成任务拓扑图的过程中，为了执行体具有更为灵活的操作功能，从而实现静态数据处理网络中对存储空间的精细化管理，并使得各个节点的执行体硬件得到优化，指定组元节点选择单元1213在生成任务关系拓扑图过程中根据具体任务选择具有包含附加节点属性的指定组元的具体任务节点，所述附加节点属性改变该具体任务节点执行对应任务的方式，例如，改变对应任务节点的执行体处理任务数据的范围、改变输出数据存储的范围、消除一次不必要的后向运算、改变输出数据的可修改属性以及改变执行体执行任务的频率等。

[0033] 尽管前面仅仅详细列出了如冗余节点消除单元1211、阻塞节点消除单元1212以及指定组元节点选择单元1213的三种拓扑图优化的单元，但是用于本公开的拓扑图优化单元非常多，在此不一一描述。另外，在任务拓扑图生成组件120生成任务拓扑图的过程中，可能会存在针对某些关联任务生成的网络子图比较复杂或低效的情形。为了获得更高效的任务拓扑图，任务拓扑图生成组件120会对对某些关联任务生成的多个网络子图，由此，需要在拓扑图优化组件121中对各种子图进行等效变换，从而从多个能够完成相同运算功能的子图网络中选择最高运算效率的子图网络来替代当前子图网络。尽管公开阐述了上述拓扑图优化组件121的各种优化单元，但是还可以包括其他任何优化单元，例如图1中所示的其他单元1214。

[0034] 在任务拓扑图生成组件120生成各层任务神经网络拓扑图之后，执行体创建组件130基于任务关系拓扑图在异构架构所包含的计算资源中为每个任务创建对应的执行体。具体而言，根据对任务描述中所需的硬件资源，基于每个节点的全部节点属性，为每个任务在异构架构中指定相应数量的运算单元和相应的存储单元构成执行体来执行对应的任务。所创建的执行体包括了异构架构中的计算资源中的各种资源，例如存储单元、消息发送或接收单元、运算单元等等。执行体的运算单元可能为一个也可能包含多个，只要其能够完成指定的任务即可。在执行体在被创建之后，将始终执行所指定的任务不会改变，除非所需执行的任务消失，例如该执行体所属的异构架构再次被应用于其他类型作业的处理。所创建的执行体之间形成的网络关系与任务拓扑图中各个神经网络节点彼此之间的关系对应从而形成了图1中所示的执行体网络组件140。构成执行体网络组件140的各个执行体分布在构成异构架构的一个或多个CPU和与CPU相连的协处理器中。协处理器诸如GPU、TPU等。如图1示意性所示，在执行体网络组件140的各个执行体采用各种微小的圆圈代替。一些小圆圈彼此通过虚线串联在一起形成数据处理路径。一条数据处理路径中会存在一些分支。两条或更多条数据处理路径之间会存在彼此交叉形成一条更为复杂关系的数据处理路径。这些数据处理路径在该异构架构上将始终保持不变。

[0035] 执行体网络组件140在接收实际作业数据时，将实际作业数据分片成为任务数据，所述任务数据被连续输入数据处理路径中，以便完成任务数据的处理。形象而言，连续输入的数据中的同类数据分片将被固定输入到同一数据处理路径中，所输入的数据分片向流水一样输入，依次进入同一数据处理路径的数据入口，并且经过处理后生成的数据也自动发送到数据处理路径中紧接着的下游执行体，直到流过整个数据处理路径为止。

[0036] 图2所示的是根据本公开的数据处理网络中执行体创建系统中指定组元节点选择单元对具体执行体影响的原理示意图。如图2所示，在用于大数据计算和深度学习的系统中，数据处理网络由各种执行体构成，为了方便描述，在图2中仅仅显示了十二个，分别标记为21、22、…32。在实际应用场景中，执行体基于需要可以为任意数量。这些执行体对输入的数据块执行预定的操作。尽管图2中进行显示了三个数据块，实际场景中，数据块的数量是海量的。

[0037] 如图2所示，在椭圆形的执行体21-29中，对外部输入的数据块1-3执行预定的操作处理。执行体通常为数据处理设备中的运算单元，例如GPU中的运算单元，或者是一个运算组件。执行体21-29之一可以只接收有一个数据输入，也可以接收多个数据输入。有些执行体可以不需要输入任何内容数据。

[0038] 在深度学习数据处理网络中，执行体在初始阶段基于确定的数据处理类型分别被赋予预定的处理任务。随着数据块的流入，每个执行体在固定接收来自上游执行体输出的数据块并将所产生的数据块输出或输出到其下游执行体。

[0039] 在本公开的数据处理网络中，数据块包括头部数据和内容数据，下面表1给出了一种数据块结构表。

[0040] 表1

[0041]

头部数据	data_type
	data_shape
	dim0_valid_num
	dim1_valid_num
	dim2_valid_num
内容数据	

[0042] 如上表1所示，头部数据确定了头部数据所含有的元数据描述了数据块的内容数据的具体内容，并且指引了内容数据的在所给定的存储空间中的具体位置。在默认情况下，执行体对数据块的操作是对数据块中的头部数据和内容数据整体的操作。但是，在具体的数据处理过程中，为了实现各种不同的目的和消除一些静态数据处理网络中问题，需要对数据块进行选择性操作、改变对数据块的操作方式或者对数据块本身进行修改。

[0043] 在具体的数据处理过程中，由于指定组元节点选择单元1213会为预定的数据块或预定的执行体20所对应的拓扑图中的具体任务节点从节点库选择含有指定组元的任务节

点，因此这些任务节点就具备了一些指定的节点属性。下文中，在不具体指明的情况下，任务节点通常与执行体对应。而且如果在不需要明确执行体具体是执行体21-29中的哪一个执行体时，统一采用标记20代替，除非需要具体指明。

[0044] 在一种场景中，指定组元节点选择单元1213可以为某一执行体的任务节点选择具有指定组元或附加节点属性的任务节点，从而使得该任务节点对应的执行体指定该执行体使用数据块的范围。例如，本公开所使用的数据块包括头部数据或内容数据。指定组元节点选择单元1213所选择的任务节点所具备使得预定执行体25仅仅使用执行体22产生的数据块1-2的头部数据。在数据处理网络中，由于指定组元节点选择单元1213选择了与执行体25对应的任务节点具有仅仅使用头部数据的指定组元，从而执行体25将从预定的常规操作方式改变成在获得指定组元后的新操作方式，例如仅仅使用执行体22产生的数据块1-2的头部数据。换句话说，由于对应任务节点具有指定组元，因此执行体25消费的是数据块1-2的头部数据。因此，执行体25只需数据块1-2的头部数据的具体值，没有必要获取执行体22保存数据块1-2的内容数据。这显然会加快数据从执行体22向执行体25的流动。

[0045] 在静态数据处理网络中，执行体是数据的消费者，可能同时也是数据的生产者。因此多个执行体构成的数据处理网络中，数据块在不同执行体之间流动，经过执行体处理或消费后变成一个新的数据块，从而供下一个执行体进行预订的操作。如下面针对图3所述，下游执行体在消费或使用完上游执行体输出的数据块之后，会向上游执行体反馈使用完成的消息，以便上游执行体空置所输出的数据块所占据的存储空间，以便接收新的需要处理的数据块。在实际场景中，由于有些执行体执行的操作可能需要多个输入数据块才能执行完成预定的操作。这很可能会出现，在接收到第一输入数据块之后，由于第二或第三数据块还没有接收到因此，该执行体不能立即执行操作的情况，因此第一输入数据块的输出执行体将不能很快获得多输入执行体的反馈消息，从而第一输入数据块的输出执行体不能空置用于存储该第一输入数据块的存储空间(例如执行体的输出数据缓存)，从而也就不能进一步向上一级执行体发出反馈消息。这会导致数据流的堵塞情形一直向前一级传导。具体而言，如图2所示，对于执行体29而言，其执行操作时需要获得来自执行体26输出的数据块2-2和经由执行体27和执行体28输出的数据块3-3。因此，在执行操作过程中，可能会存在执行体29在第一时间收到了执行体28输出的数据块3-3，但是一直没有接收到上游执行体26输出的数据块2-3，因此不能执行预定的操作。因此即使执行体29获得了数据块3-3，由于执行体29没有执行预定操作并消费完数据块3-3，因此上游执行体28无法获得执行体29的反馈消息，也就不能空置其所拥有的数据块3-3所占据的存储空间(不可以被改写状态)。这样，导致执行体28也始终不能向其上游执行体27发出执行完毕的确认消息，这也导致执行体27中数据块3所占据的存储空间无法被空置，因此最终导致执行体27所绑定的固定存储空间不能接收新的数据块进行连续的数据处理。这样即使在执行体27的下游执行体30、31以及32已经完成预定的操作的情况下，也无法从执行体27获得新的数据块进行后续处理。从而导致执行体30、31以及32也处于停滞等待状态。因此，由于执行体29处在接收来之执行体26的输出数据块的延迟导致其操作处于等待状态，因此导致了数据处理网络在一些与执行体29相关的任务节点上出现数据堵塞的情况。为了消除在数据处理过程中存在的这种数据流阻塞情况。指定组元节点选择单元1213或堵塞任务节点消除单元为此会为预计会出现数据流阻塞的执行体29的对应任务节点前选择一个具有保持头部数据的指定组元的任务节点，

例如执行体28所对应的任务节点，使得该执行体28在具有该指定组元的情况下改变所对处理的数据块之间的操作关系，从而改变执行体28对数据块的预定的操作方式，例如，将处理后的数据块的数据存储方式改变为新的数据存储方式，举例而言，对于所获得数据块仅仅保持其头部数据。同样，为配合所选择的与执行体28对应的任务节点具有仅仅保持头部数据的指定组元，需要指定组元节点选择单元1213所选择的与执行体29对应的任务节点具有仅仅使用执行体28所生成的数据的头部数据的指定组元。这样由于执行体28仅仅保存头部数据，从而在执行体29未执行操作的情况下也由于存在“仅仅使用头部数据”的指定组元，因此，执行体28的输出数据缓存中用于存储内容数据的部分处于空置状态，使得执行体28能够向执行体27反馈消息，使得执行体27可以使其自身的输出数据缓存处于空置状态以便处于能够执行下一轮操作获得新的数据块的状态。因此，执行体27的下游执行体30-33能够及时进入下一个数据块的处理，从而消除了执行体29处的数据流堵塞对相关执行体的数据处理操作的影响。

[0046] 如图2所示，为了使得数据块在被操作过程中符合一些执行体的操作要求，例如，指定组元节点选择单元1213会向数据块2发出指定组元，从而修改其数据属性。例如该属性显示，数据块2的头部数据告知对其进行操作的执行体，其可以被随后接收到的执行体修改，例如可以被执行体24所修改，从而满足执行体24的操作需求。概括而言，这是对输入数据块的头部数据的改变。

[0047] 指定组元节点选择单元1213可选择具有指定组元的任务节点，从而改变执行体对数据块的操作范围、操作方式以及存储方式，这消除了执行体在运行时需要进行临时改操作方式。即通过在任务节点选择阶段，采用先验方式选择完整任务节点(不需要运行时进行修改)，来消除运行时的额外开销。另一方面，如图2所示，为了使得输出的数据块满足下游执行体的操作需要，指定组元节点选择单元1213可以选择与执行体对应的具有可修改头部数据的指定组元的任务节点，使得对应执行体和数据块受到指定组元的影响，例如使得执行体26输出的数据块2-3改变其头部数据内容，以满足下游执行体29的操作需要。

[0048] 此外，如图2所示，为了消除不必要的后向操作，指定组元节点选择单元1213可以选择与执行体22对应的具有消除后向操作的指定组元的任务节点，从而使得执行体22在该指定组元的影响下，消除在执行体23处针对执行体22的后向操作。在涉及深度学习系统的数据处理网络中，大多存在前向操作和逆向操作，这些操作都默认需要执行体来执行。对于确定不需要逆向操作的前向操作，例如由于本公开的指定组元节点选择单元1213选择与执行体22对应的具有消除后向操作的指定组元的任务节点，消除执行体23进行的逆向操作。从而极大节省了处理过程和开销。

[0049] 根据本公开的指定组元节点选择单元1213可以通过选择特定任务节点使得执行体具有更为灵活的操作功能，从而实现静态数据处理网络中对存储空间的精细化管理，并使得各个任务节点的执行体硬件得到优化。通过指定组元节点选择单元1213为对应执行体选择能够具有对数据块的头部数据进行修改的指定组元的任务节点，使得执行体可以对数据块的头部数据进行修改，从而使得执行体能够超出数据块本身的固有属性对数据块进行更丰富多样的操作处理，从而能够扩展数据处理网络对数据块的多样化处理要求。由于指定组元节点选择单元1213的存在，对应用于数据处理网络中运行的程序的编写也带来了更多的便利性。

[0050] 尽管在本公开的描述中将数据处理网络中执行体创建系统所创建的一些执行体描述为两个独立的个体，但是可选择地，并不意味着两者分离存在是实现本公开所必要的安排，而是可以将两者结合起来。

[0051] 当根据本公开的数据处理网络中执行体创建系统用于大数据技术以及深度学习领域并且构成分布式系统时，数据处理的流畅性显得非常重要。当一个任务节点的数据处于阻塞状态时，会导致其他部分的数据处理会出现暂停，从而会导致整个系统在数据处理方面暂停，从而使得数据流动处理处于等待状态。在大数据计算以及深度学习中采用根据本公开的数据处理网络中执行体创建系统，由于数据块包含内容数据与头部数据，因此通过指定组元节点选择单元1213选择具有使得的执行体的操作关系以及执行体与数据块的操作关系收到指定组元的任务节点，可以改变执行体与数据块之间的预定操作关系，从而能够对与执行体相关联的存储空间进行精细化管理，使得一些存储空间能够被高效利用，优化执行体的硬件性能提高执行体连续处理数据块的效率。

[0052] 图3为所示的是根据本公开的数据处理网络中的执行体创建系统所创建的执行体网络的结构示意图。如图3所示，大虚线框代表一个执行体。在图3所示的执行体网络组件140中，为了图示方便，仅仅显示了五个执行体。实际上，对应于任务拓扑图，神经网络有多少任务节点，在执行体网络组件140中就存在多少执行体，因此在图3的左下侧采用连续的小实心方块来代表未示出的其他执行体。图3原理性显示了构成本公开的每个执行体的构成，其包含有消息仓、有限状态机、处理组件以及输出数据缓存。从图3中可以看出，每个执行体似乎都包含有一个输入数据缓存，但是采用了虚线标识。实际上这是一个想象的构成部件，这将在后面详细解释。处于数据处理路径中的每个执行体，例如图3中的第二执行体，基于任务拓扑图的神经网络中的一个任务节点建立，并且基于完全节点属性，形成该第二执行体与其上下游执行体的拓扑关系、消息仓、有限状态机以及(处理组件)处理方式以及生成数据的缓存位置(输出数据缓存)。具体而言，第二执行体在执行数据处理时，其任务需要其上游的第三执行体的输出数据。当第三执行体生成将要输出到第二执行体的数据，例如生成第三数据时，第三执行体将向第二执行体发出数据准备好的消息到第二执行体的消息仓，告知第二执行体第三数据已经处于第三执行体的输出数据缓存中并且处于可获取的状态，从而第二执行体可以随时执行该第三数据的读取。第二执行体的有限状态机在消息仓获得第三执行体的消息后修改其状态。如果第二执行体的处理组件在上一次执行完运算任务之后产生了数据，例如第二数据，并缓存在其输出数据缓存中，并向第二执行体的下游执行体，例如两个第一执行体(第一执行体A和第一执行体B)发出可以读取第二数据的消息。

[0053] 当第一执行体A和第一执行体B在读取第二数据并使用完成之后，会分别向第二执行体反馈消息，告知第二执行体使用完该第二数据，因此，第二执行体的输出数据缓存处于空置状态。此时第二执行体的有限状态机也会修改其状态。

[0054] 这样，当有限状态机的状态变化达到预定的状态时，例如第二执行体的执行运算所需的输入数据(例如第三数据)处于可获取状态以及其输出数据缓存处于空置状态时，则告知处理组件读取第三执行体的输出数据缓存中的第三数据，并执行指定的运算任务，从而生成该执行体的输出数据，例如新的第二数据，并存储在第二执行体的输出数据缓存中。

[0055] 同样，当第二执行体完成指定的运算任务之后，有限状态机回归到其初始状态，等

待下一次状态改变循环，同时第二执行体向第三执行体反馈对第三数据使用完成的消息到第三执行体的消息仓以及向第一执行体A和第一执行体B发送已经生成第二数据的消息，告知第一执行体A和第一执行体B，第二数据已经处于可以被读取的状态。

[0056] 当第三执行体获得第二执行体使用完第三数据的消息后，使得第三执行体的输出数据缓存处于空置状态。同样，第二执行体获得第一执行体A和第一执行体B使用完第二数据的消息后，使得第二执行体的输出数据缓存处于空置状态。

[0057] 第二执行体的上述执行任务的过程在其他执行体中同样发生。因此，在每个执行体中的有限状态机的控制下，基于上游执行体的输出结果，循环处理同类任务。从而每个执行体犹如一条数据处理路径上的固定任务的岗位人员，从而形成数据的流水线处理，不需要任何其他的外在指令。

[0058] 需要指出的是，如图3所示，第一执行体A和第一执行体B、第二执行体以及第三执行体中都被指定组元节点选择单元1213所选择任务节点中的指定组元(例如第一指定组元、第二指定组元以及第三指定组元)所影响，从而针对指定数据执行指定的操作。举例而言，第一执行体A和第一执行体B(尽管在此举例为两个第一执行体，但是其可以仅仅只有一个第一执行体或更多个第一执行体)中的第一指定组元为使得第一执行体仅仅使用某些上游执行体的输出数据的头部数据的指定组元，第三执行体中的第三指定组元为使得第三执行体可以修改数据的头部数据的指定组元以及第四执行体中的第四指定组元为使得第四执行体不要求后向运算的指定组元。需要特别指明的是，尽管此处采用第二指定组元来解释第二执行体的特性，其实此处的第二执行体的指定功能并不一定是由指定组元节点选择单元1213直接确定的，而是由指定组元节点选择单元1213所选择的具有指定组元的任务节点所确定的，即在形成任务节点过程中直接形成的一种特殊任务节点，该任务节点自身的属性包含了其仅仅保存头部数据的功能，从而使得该任务节点对应的执行体执行仅仅保持头部数据的功能。其他执行体亦然。换句话说，执行体中包含了对应任务节点所包含的指定组元所描述的节点属性之一。但是为了描述方便，在每个执行体中依然采用指定组元的方式来描述这种特殊执行体本身所含有的特定属性或功能。

[0059] 在本公开的数据存储方式中，由于头部数据和内容数据是分离存储的。如上针对图3所描述的，由于第一执行体A和第一执行体B具有仅仅使用数据的头部数据的指定组元，因此，其上游的第二执行体仅仅保持数据的头部数据。由于第二执行体仅仅搬运了第三执行体的输出数据缓存中数据的头部数据(该头部数据并不存储在该输出数据缓存中而是存储在协处理器所连接的CPU中，即该头部数据与输出数据缓存中的内容数据分离存储)，因此第二执行体会向第三执行体反馈对第三数据的内容数据使用完毕的消息，因此第三执行体输出数据缓存中用于内容数据的空间被空置。第三执行体的输出数据缓存中只需要为第二执行体保持其第二数据的头部数据。从而第三执行体的上游执行体也会获得第三执行体的反馈消息，从而消除第三执行体的上游执行体出现堵塞的现象。

[0060] 第一执行体A和第一执行体B中所具有的第一指定组元为仅仅使用某些上游执行体的输出数据的头部数据的指定组元，从而其针对第二执行体的输出数据缓存中的数据，形成仅仅使用头部数据的消费关系。当第一执行体A和第一执行体B都读取了第二执行体的输出缓存中的头部数据之后，向第二执行体反馈消息，以便第二执行体在下一个周期将另一个头部数据搬运到其输出数据缓存中。

[0061] 第三执行体中的第三指定组元为可以修改数据的头部数据的指定组元，为了使得输出的数据块满足第三执行体的下游执行体(例如第二执行体)的操作需要，指定组元节点选择单元1213可以使得第三执行体和数据块之间具有第三指定组元，使得数据块的头部数据中包含“可使用”元数据，从而使得第三执行体能够对其上游执行体输出的数据进行修改。

[0062] 尽管上面针对图3描述了执行体的基本构成以及直接上下游执行体之间的操作关系，但是有些执行体的处理组件也许根本不执行实际的运算，而是仅仅对数据进行移动，改变数据的位置，也就是一种简单的搬运执行体。例如第二执行体的处理组件仅仅将其从其第三执行体获取的数据的头部数据搬运到其输出数据缓存中而不会对第三执行体所获取的数据进行任何变换(transformation)。这种搬运执行体的存在能够消除一些执行体导致执行堵塞导致的堵塞向上游传导从而导致的整条数据处理路径上游的堵塞以及其他支路的处理暂停。

[0063] 通过在第一执行体A和第一执行体B之前设置只保存头部数据的第二执行体，第一执行体A和第一执行体B就可以基于来自第二执行体的消息知晓，在对该数据进行操作时，仅仅读取第二执行体所搬运并存储在其输出数据缓存中的头部数据即可。当第一执行体A和第一执行体B之一处于等待其他输入数据的情况下，第二执行体仅仅需要保持其头部数据，而不需要实际存储内容数据。因此第三执行体可以在接到第二执行体的反馈消息后将第三执行体的输出数据缓存置于空置状态，从而第三执行体可以在其下游第一执行体A和第一执行体B处于等待阻塞状态时依然可以对新的数据进行处理。如果不存在第二执行体，由第一执行体A和第一执行体B直接与第三执行体相连，则第三执行体在第一执行体A和第一执行体B等待期间，将不能使得其输出数据缓存处于空置状态(内容数据很大，会占用很大的内存)，这回导致其上游执行体也将处于停滞状态不能执行运算。因此，通过指定组元节点选择单元1213选择与第一执行体A和第一执行体B的对应的其中包含仅仅使用头部数据的第一指定组元的任务节点，以及在第一执行体A和第一执行体B的对应的任务节点之前增加与第二执行体对应的包含有仅仅保持头部数据的第二指定组元的任务节点，从而使得第一执行体A和第一执行体B在进行实际数据处理时，仅仅获得直接上游执行体输出数据的头部数据，从而使得第二执行体的所有上游数据都能够及时使得输出数据缓存空置，处于可以执行数据处理的状态。通过根据本公开的这种加速数据处理的构件，消除数据处理路径中的堵塞，使得在路径中存在处于等待状态的执行体任务节点时，其上游执行体的正常运行不受影响，从而加快的数据流动。

[0064] 在第一执行体A和第一执行体B可以通过其他执行体获得与第三执行体所生成的数据相似的内容数据来完成其最终的计算。如图3所示，第四执行体作为第一执行体A和第一执行体B的上游执行体，可以输出与第三执行体所生成的数据相似的内容数据，这样可以在第一执行体A和第一执行体B需要内容数据时提供解决方案。在深度学习神经网络中经常会出现一些队友型的任务节点，基于该任务节点所创建和描述的执行体是一模一样，图3中所示的第三执行体和第四执行体就属于深度学习神经网络中的队友型执行体任务节点。这属于本领域现有技术，在此不进行具体描述。

[0065] 如前所述，图3中每个执行体显示都包含有一个输入数据缓存，实际上是不包含的，因为，每个执行体并不需要任何缓存来存储将被使用的数据，而是仅仅获取所需使用的

数据处于能够被读取的状态即可。因此，每个执行体所要使用的数据在执行体未处于具体执行的状态时，数据依然保存在其上游执行体的输出数据缓存中。因此，为了形象显示，每个执行体中的输入数据缓存采用虚线表示，其实际上并不真的存在于执行体中。或者说，上游执行体的输出数据缓存就是下游执行体的虚拟输入数据缓存。因此，在图3中，对输入数据缓存采用了虚线标识。

[0066] 返回参见图1。如图1所示，根据本公开的用于异构架构的执行体创建系统还包括作业描述组件110，其用于基于作业类型描述作业神经网络模型、神经网络层数以及每层神经网络中神经元的数量。具体而言，作业描述组件110描述作业所需的运算资源以及需要执行的何种运算。例如，作业描述用于说明该作业是用于图像分类还是语音识别分类，其所需的神经网络的层数，每层的任务节点数量，层与层之间的联系，在执行数据处理过程中输入数据的存放地点。描述作业是一种现有技术。本公开的作业描述组件110采用了分离描述方法，将所需要描述的对象分拆成几个相关的维度，从几个方面或维度进行区分描述，并描述几个维度之间的正交关系。由于所描述的维度之间按照彼此区分分离方式从不同维度来描述作业，彼此处于正交关系，因此各个维度彼此不相互干扰，对任务的描述不需要考虑维度之间的关联，因此能够显著降低在本公开的异构架构的执行体创建系统中运行的程序代码的复杂度，因此也显著减轻编写这些程序代码的程序员的智力负担。尽管在图1中显示了作业描述组件110。但是采用现有的作业描述组件也可以实现本公开的目的。

[0067] 尽管如图1显示的根据本公开的用于异构架构包括一个或多个中央处理单元以及至少一个与其相连的协处理器设备端，但是在图1所示的系统中可以包括CPU之间的网关组件，也可包括协处理器，例如GPU，之间的直接通信组件，例如图1中采用虚线连接在两个GPU之间的较大的圆圈。

[0068] 此外，根据本公开的另一个方面，如上所述的构件也执行了用于加速协处理器的数据处理路径中的数据流动的方法。所述数据处理路径中多个执行体中的具有第二指定组元的第二执行体只将作为其上游执行体的第三执行体的输出数据缓存中所存储的输出数据块对应的头部数据搬运到本地的输出数据缓存，并向所述多个执行体中的一个或多个具有第一指定组元的第一执行体发出可以读取数据的消息。所述数据处理路径中多个执行体中的多个执行体中的一个或多个具有第一指定组元的第一执行体基于来自第二执行体的消息，从第二执行体的输出数据缓存中读取所述头部数据，并执行预定的操作。所述第二执行体在将头部数据搬运到本地的输出数据缓存的同时，向第二执行体的第三执行体发送消息，使得第三执行体将其自身的输出数据缓存置于空置状态。所述第一执行体从位于其上游的第四执行体的输出数据缓存中获取与第三执行体所生成的数据块的内容数据相似的数据以便执行预定的操作。第二执行体在接收到所有第一执行体的反馈消息后将其自身的输出数据缓存置于空置状态。

[0069] 以上结合具体实施例描述了本公开的基本原理，但是，需要指出的是，对本领域的普通技术人员而言，能够理解本公开的方法和装置的全部或者任何步骤或者部件，可以在任何计算装置(包括处理器、存储介质等)或者计算装置的网络中，以硬件、固件、软件或者它们的组合加以实现，这是本领域普通技术人员在阅读了本公开的说明的情况下运用他们的基本编程技能就能实现的。

[0070] 因此，本公开的目的还可以通过在任何计算装置上运行一个程序或者一组程序来

实现。所述计算装置可以是公知的通用装置。因此，本公开的目的也可以仅仅通过提供包含实现所述方法或者装置的程序代码的程序产品来实现。也就是说，这样的程序产品也构成本公开，并且存储有这样的程序产品的存储介质也构成本公开。显然，所述存储介质可以是任何公知的存储介质或者将来所开发出来的任何存储介质。

[0071] 还需要指出的是，在本公开的装置和方法中，显然，各部件或各步骤是可以分解和/或重新组合的。这些分解和/或重新组合应视为本公开的等效方案。并且，执行上述系列处理的步骤可以自然地按照说明的顺序按时间顺序执行，但是并不需要一定按照时间顺序执行。某些步骤可以并行或彼此独立地执行。

[0072] 上述具体实施方式，并不构成对本公开保护范围的限制。本领域技术人员应该明白的是，取决于设计要求和其他因素，可以发生各种各样的修改、组合、子组合和替代。任何在本公开的精神和原则之内所作的修改、等同替换和改进等，均应包含在本公开保护范围之内。

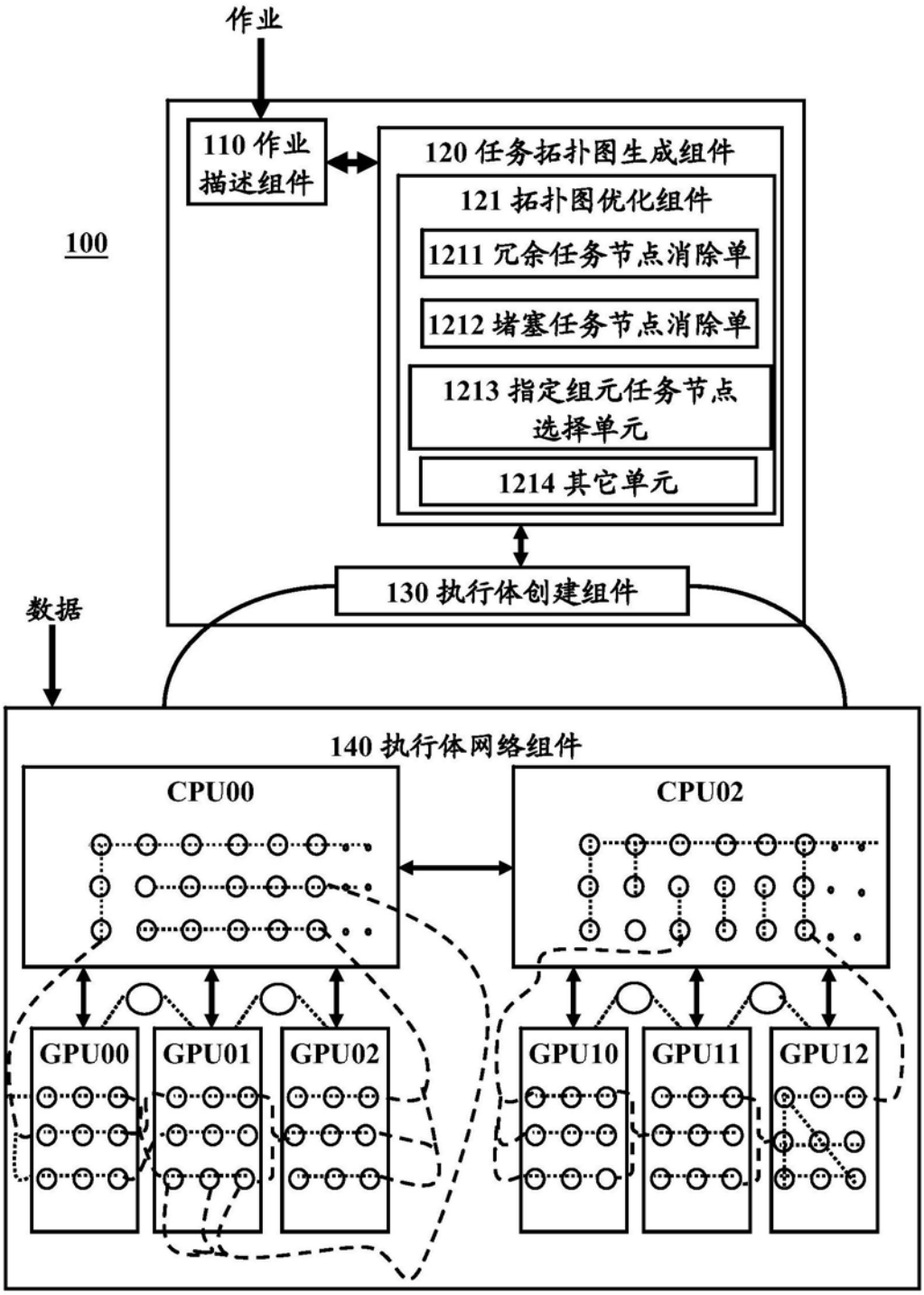

作业
110 作业描述组件
120 任务拓扑图生成组件
121 拓扑图优化组件
100
1211 冗余任务节点消除单
1212 堵塞任务节点消除单
1213 指定组元任务节点选择单元
1214 其它单元
130 执行体创建组件
数据
140 执行体网络组件
CPU00
CPU02
GPU00
GPU01
GPU02
GPU10
GPU11
GPU12

图1

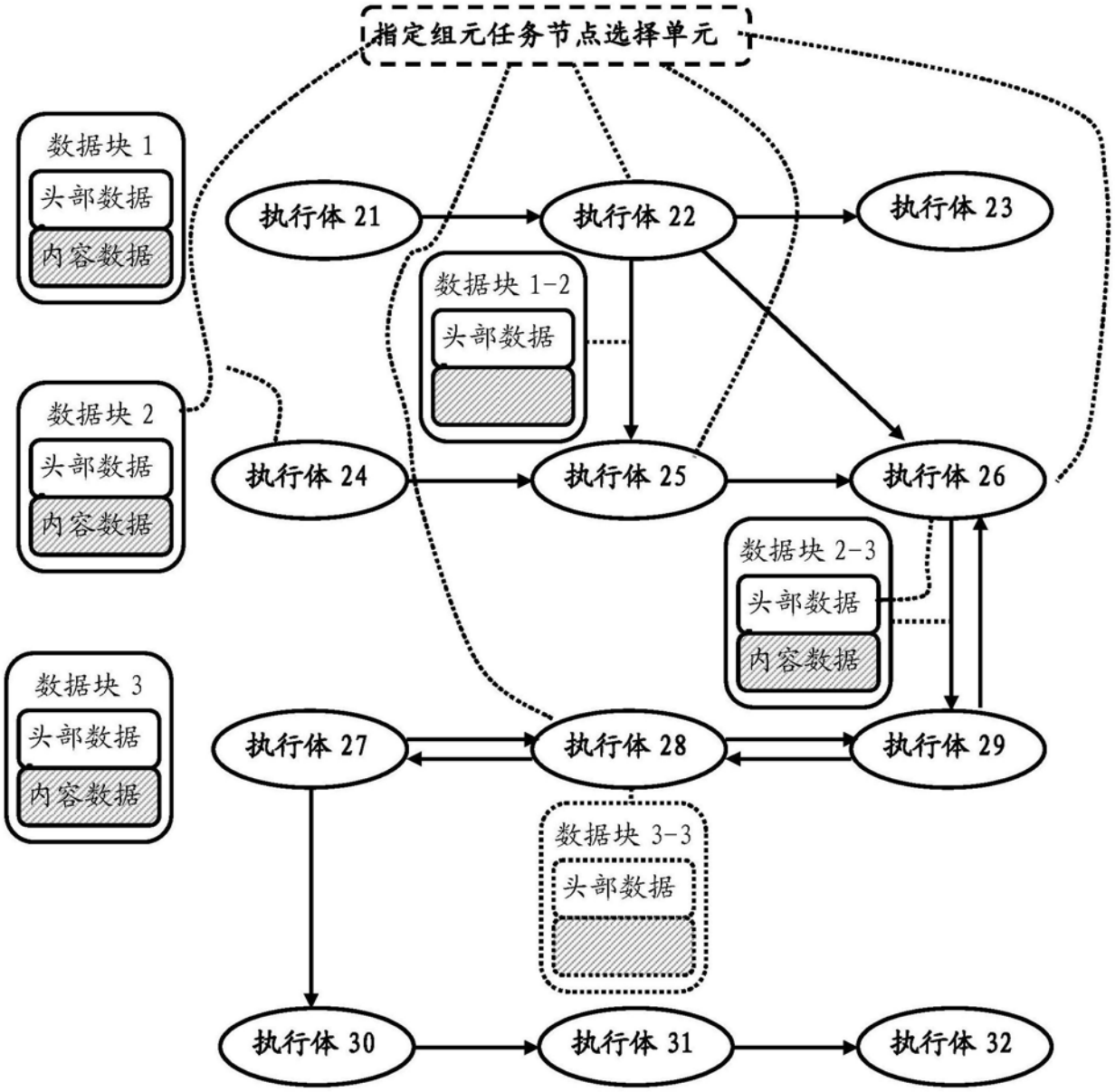

指定组元任务节点选择单元
数据块 1
头部数据
内容数据
执行体 21
执行体 22
执行体 23
数据块 1-2
头部数据
数据块 2
头部数据
内容数据
执行体 24
执行体 25
执行体 26
数据块 2-3
头部数据
内容数据
数据块 3
头部数据
内容数据
执行体 27
执行体 28
执行体 29
数据块 3-3
头部数据
执行体 30
执行体 31
执行体 32

图2

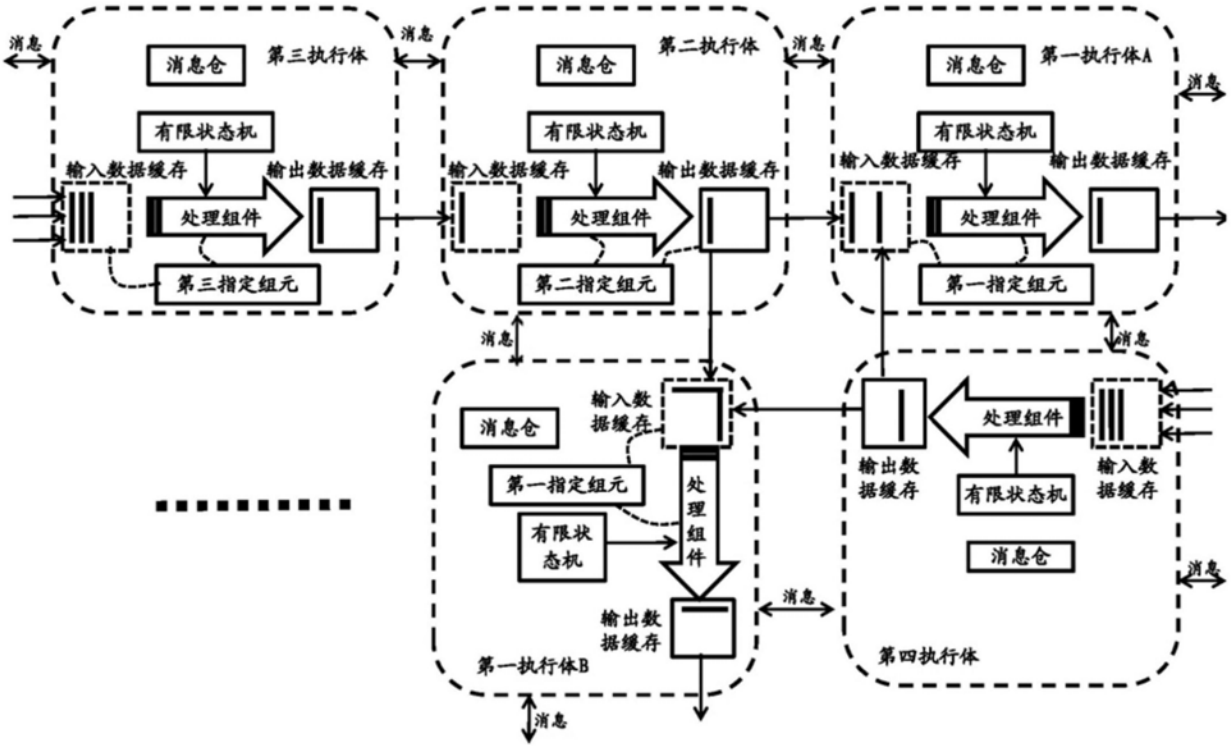

消息
消息仓
第三执行体
有限状态机
输入数据缓存
输出数据缓存
处理组件
第三指定组元
第二执行体
第二指定组元
第一执行体A
第一指定组元
第一执行体B
第四执行体

图3