[19] Patents Registry
The Hong Kong Special Administrative Region
香港特別行政區
專利註冊處

[11] 1237463 B
CN 106687918 B

[12] **STANDARD PATENT (R) SPECIFICATION**
轉錄標準專利說明書

[21] Application no. 申請編號
17111227.3

[22] Date of filing 提交日期
02.11.2017

[51] Int. Cl.
G06F 8/34 (2018.01) G06F 8/41 (2018.01)
G06F 9/448 (2018.01) G06F 9/46 (2006.01)
G06F 9/48 (2006.01) G06F 9/50 (2006.01)
G06F 13/36 (2006.01) G06F 16/22 (2019.01)
G06F 16/901 (2019.01)

[54] COMPILING GRAPH-BASED PROGRAM SPECIFICATIONS
編譯基於圖的程序規範

[30] Priority 優先權
02.09.2014 US 62/044,645
20.05.2015 US 62/164,175

[43] Date of publication of application 申請發表日期
13.04.2018

[45] Date of publication of grant of patent 批予專利的發表日期
09.04.2021

[86] International application no. 國際申請編號
PCT/US2015/048096

[87] International publication no. and date 國際申請發表編號及日期
WO2016/036826 10.03.2016

CN Application no. & date 中國專利申請編號及日期
CN 201580047113.X 02.09.2015

CN Publication no. & date 中國專利申請發表編號及日期
CN 106687918 17.05.2017

Date of grant in designated patent office 指定專利當局批予專利日期
28.08.2020

[73] Proprietor 專利所有人
Ab Initio Technology LLC
起元科技有限公司
201 Spring Street
Lexington, MA02421
UNITED STATES OF AMERICA

[72] Inventor 發明人
Craig W. STANFILL C．W．斯坦菲尔
Richard SHAPIRO R．夏皮罗
Stephen A. KUKOLICH S．A．库克利希

[74] Agent and / or address for service 代理人及/或送達地址
LUNG TIN PATENT TRADEMARK AGENT LIMITED
FLAT 11, 10/F KOWLOON PLAZA
485 CASTLE PEAK ROAD, CHEUNG SHA WAN
HONG KONG

(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 106687918 B
(45)授权公告日 2020.08.28

(21)申请号 201580047113.X

(22)申请日 2015.09.02

(65)同一申请的已公布的文献号
申请公布号 CN 106687918 A

(43)申请公布日 2017.05.17

(30)优先权数据
62/044,645 2014.09.02 US
62/164,175 2015.05.20 US

(85)PCT国际申请进入国家阶段日
2017.03.02

(86)PCT国际申请的申请数据
PCT/US2015/048096 2015.09.02

(87)PCT国际申请的公布数据
WO2016/036826 EN 2016.03.10

(73)专利权人 起元科技有限公司
地址 美国马萨诸塞州

(72)发明人 C·W·斯坦菲尔 R·夏皮罗
S·A·库克利希

(74)专利代理机构 隆天知识产权代理有限公司 72003
代理人 张浴月 金鹏

(51)Int.Cl.
G06F 8/34(2018.01)
G06F 8/41(2018.01)
G06F 9/448(2018.01)
G06F 9/46(2006.01)
G06F 9/48(2006.01)
G06F 9/50(2006.01)
G06F 13/36(2006.01)
G06F 16/22(2019.01)
G06F 16/901(2019.01)

(56)对比文件
US 2003014500 A1,2003.01.16,
CN 103778015 A,2014.05.07,
CN 103729330 A,2014.04.16,

审查员 郭俊

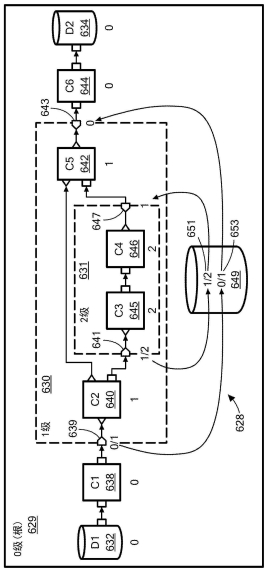
权利要求书3页 说明书36页 附图20页

(54)发明名称

编译基于图的程序规范

(57)摘要

基于图的程序规范(628)包括:多个组件,每个对应处理任务并包括发送或接收一个或多个数据元素的一个或多个端口;和一个或多个链路,每个将多个组件的上游组件的输出端口连接到多个组件的下游组件的输入端口。生成表示多个组件的子集(630,631)的准备好的代码包括:至少部分地基于链接的组件的特性,识别不同子集中组件之间的多个子集边界;基于所识别的子集边界形成子集;和为每个形成的子集生成准备好的代码,当用于运行时系统执行时,使得根据为该形成的子集的准备好的代码中嵌入的信息来执行与该形成的子集中的组件对应的处理任务。



CN 106687918 B

1.一种用于处理基于图的程序规范的方法,所述方法包括:

接收所述基于图的程序规范,所述基于图的程序规范包括:

多个组件,每个组件对应于处理任务并且包括用于发送或接收一个或多个数据元素的一个或多个端口;以及

一个或多个链路,所述一个或多个链路中的每个链路将所述多个组件中的上游组件的输出端口连接到所述多个组件中的下游组件的输入端口;以及

处理所述基于图的程序规范,以生成表示所述基于图的程序规范的所述多个组件的子集的准备好的代码,所述处理包括:

至少部分地基于链接的组件的特性,在不同子集中识别组件之间的多个子集边界;

基于所识别的子集边界形成子集;以及

为每个形成的子集生成准备好的代码,当用于运行时系统执行时,所述准备好的代码使得根据为该形成的子集的准备好的代码中嵌入的信息来执行与该形成的子集中的组件对应的处理任务,其中所述运行时系统实例化该形成的子集的一定数量的实例,并且在运行时间期间至少部分地基于所述运行时系统在处理多个数据元素的集时的性能来动态地确定所述数量。

2.根据权利要求1所述的方法,其中形成子集包括:遍历所述基于图的程序规范的组件,同时保持所遍历的子集边界的记录,以及将所述基于图的程序规范的每个组件与从所遍历的子集边界的记录中确定的单个子集标识符相关联。

3.根据权利要求2所述的方法,其中与所述多个组件的所识别的子集相关联的每个子集标识符是唯一的。

4.根据权利要求2所述的方法,其中所遍历的子集边界的记录被维护为标识符值的路径。

5.根据权利要求4所述的方法,其中所述标识符值的路径包括通过分隔符彼此分隔的标识符值的字符串。

6.根据权利要求1所述的方法,其中形成子集包括:

将所述基于图的程序规范的第一组件与一子集标识符相关联;

将该子集标识符传播到所述第一组件的下游的组件;以及

基于所识别的子集边界,在该子集标识符的传播过程中修改该子集标识符。

7.根据权利要求6所述的方法,其中在该子集标识符的传播过程中修改该子集标识符包括:

在遍历第一子集边界时,将该子集标识符的值从第一子集标识符值改变为与所述第一子集边界相关联的第二子集标识符值;以及

在遍历与所述第一子集边界相关联的第二子集边界时,将该子集标识符的值改变为所述第一子集标识符值。

8.根据权利要求1所述的方法,其中至少部分地基于链接的组件的特性识别一个或多个子集边界包括:基于上游组件上的第一类型的端口与下游组件上的第二类型的端口之间的链路来识别子集边界。

9.根据权利要求1所述的方法,其中至少部分地基于链接的组件的特性识别一个或多个子集边界包括:基于上游组件与下游组件之间的确定类型的链路来识别子集边界,其中

所述确定类型的链路是组件之间多种不同类型的链路之一。

10.根据权利要求1所述的方法，其中为每个形成的子集生成准备好的代码包括：对于至少一个形成的子集，将指示与该形成的子集中的组件对应的处理任务之间的允许的并发性的信息嵌入到所述准备好的代码中。

11.根据权利要求1所述的方法，其中为每个形成的子集生成准备好的代码包括：对于至少一个形成的子集，将指示相对于其它形成的子集的优先级的信息嵌入到所述准备好的代码中。

12.根据权利要求1所述的方法，其中为每个形成的子集生成准备好的代码包括：对于至少一个形成的子集，将指示与该形成的子集中的组件对应的一个或多个处理任务的事务性的信息嵌入到所述准备好的代码中。

13.根据权利要求1所述的方法，其中为每个形成的子集生成准备好的代码包括：对于至少一个形成的子集，将指示在所述准备好的代码的执行过程中要锁定的至少一个资源的信息嵌入到所述准备好的代码中。

14.根据权利要求1所述的方法，其中为每个形成的子集生成准备好的代码包括：对于至少一个形成的子集，将指示与该形成的子集的组件对应的一个或多个处理任务所处理的数据元素之间的排序特性的信息嵌入到所述准备好的代码中。

15.根据权利要求1所述的方法，其中为每个形成的子集生成准备好的代码包括：对于至少一个形成的子集，将指示使用所述准备好的代码执行该形成的子集的每个实例所操作的数据元素的数量的信息嵌入到所述准备好的代码中。

16.根据权利要求1所述的方法，其中如果存在多个实例，则将所述多个实例中的每一个应用于所述多个数据元素的集中的数据元素的不同的相应子集。

17.一种以非暂时形式存储有软件的计算机可读介质，所述软件用于处理基于图的程序规范，所述软件包括用于使得计算系统执行以下操作的指令：

接收所述基于图的程序规范，所述基于图的程序规范包括：

多个组件，每个组件对应于处理任务并且包括用于发送或接收一个或多个数据元素的一个或多个端口；以及

一个或多个链路，所述一个或多个链路中的每个链路将所述多个组件中的上游组件的输出端口连接到所述多个组件中的下游组件的输入端口；以及

处理所述基于图的程序规范，以生成表示所述基于图的程序规范的所述多个组件的子集的准备好的代码，所述处理包括：

至少部分地基于链接的组件的特性，在不同子集中识别组件之间的多个子集边界；

基于所识别的子集边界形成子集；以及

为每个形成的子集生成准备好的代码，当用于运行时系统执行时，所述准备好的代码使得根据为该形成的子集的准备好的代码中嵌入的信息来执行与该形成的子集中的组件对应的处理任务，其中所述运行时系统实例化该形成的子集的一定数量的实例，并且在运行时间期间至少部分地基于所述运行时系统在处理多个数据元素的集时的性能来动态地确定所述数量。

18.一种用于处理基于图的程序规范的计算系统，所述计算系统包括：

至少一个输入设备或端口，被配置为接收所述基于图的程序规范，所述基于图的程序

规范包括：

多个组件，每个组件对应于处理任务并且包括用于发送或接收一个或多个数据元素的一个或多个端口；以及

一个或多个链路，所述一个或多个链路中的每个链路将所述多个组件中的上游组件的输出端口连接到所述多个组件中的下游组件的输入端口；以及

至少一个处理器，被配置为处理所述基于图的程序规范，以生成表示所述基于图的程序规范的所述多个组件的子集的准备好的代码，所述处理包括：

至少部分地基于链接的组件的特性，在不同子集中识别组件之间的多个子集边界；

基于所识别的子集边界形成子集；以及

为每个形成的子集生成准备好的代码，当用于运行时系统执行时，所述准备好的代码使得根据为该形成的子集的准备好的代码中嵌入的信息来执行与该形成的子集中的组件对应的处理任务，其中所述运行时系统实例化该形成的子集的一定数量的实例，并且在运行时间期间至少部分地基于所述运行时系统在处理多个数据元素的集时的性能来动态地确定所述数量。

编译基于图的程序规范

[0001] 相关申请的交叉引用

[0002] 本申请要求于2014年9月2日提交的序列号为62/044,645的美国申请和2015年5月20日提交的序列号为62/164,175的美国申请的优先权。

技术领域

[0003] 本说明书涉及一种编译基于图的程序规范的方法。

背景技术

[0004] 一种用于数据流计算的方法利用基于图的表示,其中对应于图的节点(顶点)的计算组件通过与图的链路(有向边)对应的数据流(称为“数据流图”)耦合。通过数据流链路连接到上游组件的下游组件接收输入数据元素的有序流,并且以所接收的顺序处理输入数据元素,可选地生成一个或多个对应的输出数据元素流。用于执行这种基于图的计算的系统描述于发明名称为“EXECUTING COMPUTATIONS EXPRESSED AS GRAPHS(执行表达为图的计算)”的在先美国专利5,966,072中,其通过引用并入本文。在与该在先专利中描述的方法相关的实现方式中,每个组件被实现为驻留在通常多个计算机服务器之一上的进程。每个计算机服务器可以具有在任何一个时间活动的多个这样的组件进程,并且操作系统(例如,Unix)调度器在该服务器上托管的组件之间共享资源(例如,处理器时间和/或处理器内核)。在这种实现方式中,组件之间的数据流可以使用操作系统的数据通信服务和连接服务器的数据网络(例如,命名管道,TCP/IP会话等)来实现。组件的子集通常用作来自整体计算的数据的源和/或数据宿(或数据接收器,sink),例如,去往和/或来自数据文件、数据库表和外部数据流。然后,在例如通过协调进程建立组件进程和数据流之后,数据流过整个计算系统,该计算系统实现表达为通常由每个组件处的输入数据的可用性控制的图的计算并且为每个组件调度计算资源。因此,至少通过使不同的组件能够由不同的进程(托管在相同或不同的服务器计算机或处理器内核上)并行地执行,可以实现并行性,其中不同组件在通过数据流图的不同路径上并行地执行在本文中被称为组件并行性,并且不同组件通过数据流图在相同路径的不同部分上并行执行在本文中被称为流水线并行性。

[0005] 其他形式的并行性也得到这种方法的支持。例如,输入数据集可以例如根据数据集的记录中字段的值的分区来分割,其中每个部分被发送到处理数据集的记录的组件的单独副本。组件的这种单独的副本(或“实例”)可以在单独的服务器计算机或服务器计算机的单独的处理器内核上执行,从而实现本文所称的数据并行性。可以合并单独组件的结果以再次形成单个数据流或数据集。用于执行组件实例的计算机或处理器内核的数量将由开发人员在开发数据流图时指定。

[0006] 可以使用各种方法来提高这种方法的效率。例如,组件的每个实例不一定必须在其自己的操作系统进程中托管,例如使用一个操作系统进程来实现多个组件(例如,形成更大图的连接子图的组件)。

[0007] 上述方法的至少一些实现方式受到与在底层计算机服务器上所得到的进程的执

行效率有关的限制。例如，这些限制可能涉及重新配置图的运行实例以改变数据并行度、改变到托管各种组件的服务器和/或平衡不同计算资源上的负载的困难。现有的基于图的计算系统也具有的缺陷是启动时间慢，这通常是因为太多的进程被不必要地启动，浪费了大的存储量。通常，进程在启动图的执行时开始，并在图的执行完成时结束。

[0008] 已经使用了用于分布计算的其他系统，其中将整体计算划分为更小的部分，并且将这些部分从一个主计算机服务器分发到各个其他(例如，“从属”)计算机服务器，从属计算机服务器各自独立地执行计算并将其结果返回到主服务器。一些这样的方法被称为“网格计算”。然而，除了经由调用那些部分的主计算机服务器之外，这样的方法通常依赖于每个计算的独立性，而不提供用于在计算部分之间传递数据或调度和/或对这些部分的执行进行排序的机制。因此，对于托管涉及多个组件之间的交互的计算，这样的方法没有提供直接有效的解决方案。

[0009] 用于大型数据集的分布式计算的另一种方法使用MapReduce框架，例如，如在Apache Hadoop®系统中体现的。通常，Hadoop具有分布式文件系统，其中分发针对每个命名文件的部分。用户根据两个函数指定计算：以分布式方式在命名输入的所有部分上执行的映射函数，以及在映射函数执行的输出的部分上执行的归约(reduce)函数。映射函数执行的输出被分割并再次存储在分布式文件系统中的中间部分中。然后以分布式方式执行归约函数以处理这些中间部分，从而产生整体计算的结果。尽管可以有效地执行可以在MapReduce框架中表示并且其输入和输出可修改以用于在MapReduce框架的文件系统中存储的计算，但是许多计算不匹配该框架和/或不容易适应于使得它们的所有输入和输出都在分布式文件系统中。

[0010] 总体而言，与组件(或组件的并行执行副本)托管在不同服务器上的上述方法相比，需要提高计算的底层规范是图的计算的计算效率(例如，增加给定计算资源的每单位处理的记录的数量)。此外，期望能够适应变化的计算资源和需求。还需要提供一种计算方法，其允许适应在一个或多个基于图的计算的执行期间可用的计算资源的变化，和/或允许适应例如由于正在处理的数据的特性导致的这样的计算的不同组件的计算负载的变化或者负载的时间变化。还需要提供能够有效地利用具有不同特性的计算资源的计算方法，例如使用具有每个服务器不同数量的处理器、每个处理器不同数量的处理器内核等的服务器，以有效地支持同质和异质环境。还期望使基于图的计算快速启动。提供这种效率和适应性的一个方面是在创建图时(在设计时)开发人员做出的选择、(在编译时)编译器所采取的动作和(在运行时)运行时系统所采取的动作之间提供适当的分离和抽象屏障。

发明内容

[0011] 在一个方面中，一般来说，一种用于处理基于图的程序规范的方法包括：接收所述基于图的程序规范，所述基于图的程序规范包括：多个组件，每个组件对应于处理任务并且包括用于发送或接收一个或多个数据元素的一个或多个端口；以及一个或多个链路，所述一个或多个链路中的每个链路将所述多个组件中的上游组件的输出端口连接到所述多个组件中的下游组件的输入端口；以及处理所述基于图的程序规范以生成表示所述基于图的程序规范的所述多个组件的子集(对应于“执行集”)的准备好的代码。如本文所使用的，“准备好的代码”包括当对基于图的程序规范的解析后的元素进行转换时由编译器或解释器使

用的以任何目标语言编写的代码，其可以包括可执行代码或者可以被进一步编译或解释成可执行代码的代码。所述处理包括：至少部分地基于链接的组件的特性在不同子集中识别组件之间的多个子集边界；基于所识别的子集边界形成子集；以及为每个形成的子集生成准备好的代码，当用于运行时系统执行时，所述准备好的代码使得根据为该形成的子集的准备好的代码中嵌入的信息来执行与该形成的子集中的组件对应的处理任务。

[0012] 各个方面可以包括以下特征中的一个或多个。

[0013] 形成子集包括：遍历所述基于图的程序规范的组件，同时保持所遍历的子集边界的记录，以及将所述基于图的程序规范的每个组件与从所遍历的子集边界的记录中确定的单个子集标识符相关联。

[0014] 与所述多个组件的所识别的子集相关联的每个子集标识符是唯一的。

[0015] 所遍历的子集边界的记录被维护为标识符值的路径。

[0016] 所述标识符值的路径包括通过分隔符彼此分隔的标识符值的字符串。

[0017] 形成子集包括：将所述基于图的程序规范的第一组件与一子集标识符相关联；将该子集标识符传播到所述第一组件的下游的组件；以及基于所识别的子集边界，在该子集标识符的传播过程中修改该子集标识符。

[0018] 在该子集标识符的传播过程中修改该子集标识符包括：在遍历所述第一子集边界时将该子集标识符的值从第一子集标识符值改变为与第一子集边界相关联的第二子集标识符值；以及在遍历与所述第一子集边界相关联的第二子集边界时将该子集标识符的值改变为所述第一子集标识符值。

[0019] 至少部分地基于链接的组件的特性识别一个或多个子集边界包括：基于上游组件上的第一类型的端口与下游组件上的第二类型的端口之间的链路来识别子集边界。

[0020] 至少部分地基于链接的组件的特性识别一个或多个子集边界包括：基于上游组件和下游组件之间的确定类型的链路来识别子集边界，其中所述确定类型的链路是组件之间多种不同类型的链路之一。

[0021] 为每个形成的子集生成准备好的代码包括：对于至少一个形成的子集，将指示与该形成的子集中的组件对应的处理任务之间的允许的并发性的信息嵌入到所述准备好的代码中。

[0022] 为每个形成的子集生成准备好的代码包括：对于至少一个形成的子集，将指示相对于其他形成的子集的优先级的信息嵌入到所述准备好的代码中。

[0023] 为每个形成的子集生成准备好的代码包括：对于至少一个形成的子集，将指示与该形成的子集中的组件对应的一个或多个处理任务的事务性的信息嵌入到所述准备好的代码中。

[0024] 为每个形成的子集生成准备好的代码包括：对于至少一个形成的子集，将指示在所述准备好的代码的执行过程中要锁定的至少一个资源的信息嵌入到所述准备好的代码中。

[0025] 为每个形成的子集生成准备好的代码包括：对于至少一个形成的子集，将指示与该形成的子集的组件对应的一个或多个处理任务所处理的数据元素之间的排序特性的信息嵌入到所述准备好的代码中。

[0026] 为每个形成的子集生成准备好的代码包括：对于至少一个形成的子集，将指示使

用所述准备好的代码执行形成的子集的每个实例所操作的数据元素的数量的信息嵌入到所述准备好的代码中。

[0027] 在另一方面中，一般来说，一种以非暂时形式存储在计算机可读介质上的软件，用于处理基于图的程序规范，所述软件包括用于使得计算系统执行以下操作的指令：接收所述基于图的程序规范，所述基于图的程序规范包括：多个组件，每个组件对应于处理任务并且包括用于发送或接收一个或多个数据元素的一个或多个端口；以及一个或多个链路，所述一个或多个链路中的每个链路将所述多个组件中的上游组件的输出端口连接到所述多个组件中的下游组件的输入端口；以及处理所述基于图的程序规范以生成表示所述基于图的程序规范的所述多个组件的子集的准备好的代码，所述处理包括：至少部分地基于链接的组件的特性在不同子集中识别组件之间的多个子集边界；基于所识别的子集边界形成子集；以及为每个形成的子集生成准备好的代码，当用于运行时系统执行时，所述准备好的代码使得根据为该形成的子集的准备好的代码中嵌入的信息来执行与该形成的子集中的组件对应的处理任务。

[0028] 在另一方面中，一般来说，一种用于处理基于图的程序规范的计算系统包括：至少一个输入设备或端口，被配置为接收所述基于图的程序规范，所述基于图的程序规范包括：多个组件，每个组件对应于处理任务并且包括用于发送或接收一个或多个数据元素的一个或多个端口；以及一个或多个链路，所述一个或多个链路中的每个链路将所述多个组件中的上游组件的输出端口连接到所述多个组件中的下游组件的输入端口；以及至少一个处理器，配置为处理所述基于图的程序规范以生成表示所述基于图的程序规范的所述多个组件的子集的准备好的代码，所述处理包括：至少部分地基于链接的组件的特性在不同子集中识别组件之间的多个子集边界；基于所识别的子集边界形成子集；以及为每个形成的子集生成准备好的代码，当用于运行时系统执行时，所述准备好的代码使得根据为该形成的子集的准备好的代码中嵌入的信息来执行与该形成的子集中的组件对应的处理任务。

[0029] 各个方面可以具有一个或多个以下优点。

[0030] 本文所描述的技术还便于在其层级结构的各个层级处使用非常规技术特征来高效处理计算系统中的大量数据。这些技术特征在计算系统的各种操作阶段(包括设计时，编译时和运行时)一起工作。编程平台使得基于图的程序规范能够在设计时指定期望的计算。编译时编译器准备目标程序规范，以便在运行时在计算系统的多个服务器之间有效地分配细粒度任务。例如，根据基于图的程序规范中的任何控制流和数据流约束来配置多个任务。运行时系统支持以提高计算效率(例如，以给定计算资源的每单位被处理记录的数量为单位)的方式对并发执行的这些任务进行动态分配。各种技术特征一起工作以实现相对于常规系统的效率增益。

[0031] 例如，计算系统能够使用与数据处理图(或其他基于图的程序规范)的组件对应的任务，以便于以这些任务的灵活的运行时执行的方式来处理数据元素，而不需要对编程器添加不适当的负担。图形用户界面允许在执行期望的数据处理计算的组件上不同类型的端口之间的连接，并且计算系统能够自动地识别包括一个或多个组件的子集和/或嵌套的组件子集，用于稍后在处理程序规范时使用。例如，该执行集发现预处理过程可以识别潜在的嵌套的组件执行集的层级结构，这对于人们来说是非常难以识别的，然后系统可以确定底层系统架构中的资源分配，以执行那些子集，用于有效的并行数据处理。通过自动地识别组

件的这些子集(“执行集”),计算系统能够确保数据处理图满足某些一致性要求,如下面更详细描述的,并且允许底层计算系统以高度可扩展的并行度来操作执行集,因为执行集的并行度可以在运行时确定,并且仅受运行时可用的计算资源的限制,因此有助于数据处理图的有效执行。此外,通过将某些信息嵌入到识别执行集的准备好的代码中,这些执行集最终可以由底层计算系统作为特定任务来处理,并且计算系统可以通过例如并行化任务来确保以提高计算系统的内部功能的效率的方式执行处理任务。

[0032] 当执行本文所描述的方法时,这些技术还表现出对计算系统的内部功能的进一步的技术效果,诸如减少对存储器和其他计算资源的需求,以及减少系统在处理各个数据元素时的延迟。特别地,这些优点有助于数据处理图的高效执行。例如,由于在执行图时由其他进程(例如,Unix进程)启动多个进程,因而传统的基于图的计算系统可能具有相对较高的延迟(例如,几十毫秒的量级),导致这些进程的累积启动时间。然而,本文描述的技术通过允许单个进程内的程序代码直接启动其他程序代码而不需要进程启动开销,从而有利于实现相对较低的延迟(例如,几十微秒的数量级)以及每秒处理的数据的更高的吞吐量。有助于数据处理图的高效执行的其他方面在以下描述中将显而易见。

[0033] 从下面的描述和权利要求书中,本发明的其它特征和优点将变得显而易见。

附图说明

[0034] 图1是基于任务的计算系统的框图。

[0035] 图2A是具有控制和数据端口的数据处理图的一部分的示例。

[0036] 图2B-图2C是具有控制端口和数据端口的数据处理图的示例。

[0037] 图3A是包括多个标量输出端口到标量输入端口连接的数据处理图。

[0038] 图3B是包括多个集合输出端口到集合输入端口连接的数据处理图。

[0039] 图3C是包括集合输出端口到标量输入端口连接和标量输出端口到集合输入端口连接的数据处理图。

[0040] 图4A是两个组件之间的标量端口到标量端口连接。

[0041] 图4B是两个组件之间的集合端口到集合端口连接。

[0042] 图4C是两个组件之间的集合端口到标量端口连接,包括执行集入口点。

[0043] 图4D是两个组件之间的标量端口到集合端口连接,包括执行集出口点。

[0044] 图5是应用基于堆栈的分配算法的数据处理图。

[0045] 图6是应用基于全局映射的分配算法的数据处理图。

[0046] 图7是具有用户定义的执行集的数据处理图。

[0047] 图8A和图8B示出了数据处理图中的“相同集”关系。

[0048] 图9是具有复制数据元素的入口点的数据处理图。

[0049] 图10A-图10C示出了用户界面工作流。

[0050] 图11A是具有非法执行集的数据处理图。

[0051] 图11B是具有非法执行集循环的数据处理图。

[0052] 图12A-图12B是数据处理图和对应的控制图的示例的图。

[0053] 图13A-图13B是示例性执行状态机的状态转换图。

[0054] 图14是一组处理引擎的图。

具体实施方式

[0055] 参考图1,基于任务的计算系统100使用高级程序规范110来控制计算平台150的计算资源和存储资源,以执行由程序规范110指定的计算。编译器/解释器120接收高级程序规范110并生成可由基于任务的运行时接口/控制器140执行的形式的基于任务的规范130。编译器/解释器120识别一个或多个“组件”的一个或多个“执行集”,其可以单独地或作为单元被实例化为要应用于多个数据元素中每一个元素的细粒度任务。如下面更详细描述的,编译或解释进程的一部分涉及识别这些执行集并且准备所述执行集用于执行。应当理解,编译器/解释器120可以使用包括诸如解析高级程序规范110、验证语法、类型检查数据格式、生成任何错误或警告以及准备基于任务的规范130之类的步骤的各种算法中的任何算法,并且例如编译器/解释器120可以利用各种技术来优化在计算平台150上执行的计算的效率。由编译器/解释器120生成的目标程序规范本身可以是将由系统100的另一部分进一步处理(例如,进一步编译、解释等)以产生基于任务的规范130的中间形式。下面的讨论概述了这种变换的一个或多个示例,但是当然,例如由编译器设计的技术人员将理解的,用于变换的其他方法也是可能的。

[0056] 通常,计算平台150由多个计算节点152(例如,提供分布式计算资源和分布式存储资源这二者的个体服务器计算机)组成,从而实现高度并行性。如下面进一步详细讨论的,在高级程序规范110中表示的计算在计算平台150上被执行为相对细粒度的任务,进一步实现指定计算的高效并行执行。

[0057] 1数据处理图

[0058] 在一些实施例中,高级程序规范110是称为“数据处理图”的一种基于图的程序规范,其包括一组“组件”,每个组件指定要对数据执行的总体数据处理计算的一部分。组件例如在编程用户界面和/或计算的数据表示中表示为图中的节点。与一些基于图的程序规范(例如在上述背景技术中描述的数据流图)不同,数据处理图可以包括代表数据传送或控制传送中的任何一个或两者的节点之间的链路。指示链路特性的一种方式是在组件上提供不同类型的端口。链路是从上游组件的输出端口耦合到下游组件的输入端口的有向链路。端口具有表示如何从链路写入和读取数据元素和/或如何控制组件以处理数据的特性的指示符。

[0059] 这些端口可以具有多个不同的特性。端口的一个特性是其作为输入端口或输出端口的方向性。有向链路表示从上游组件的输出端口传送到下游组件的输入端口的数据和/或控制。允许开发人员将不同类型的端口链接在一起。数据处理图的一些数据处理特性取决于不同类型的端口如何链接在一起。例如,如下面更详细描述的,不同类型的端口之间的链路可以导致提供层级形式的并行性的不同“执行集”中的组件的嵌套子集。某些数据处理特性由端口的类型所暗示。组件可能具有的不同类型的端口包括:

[0060] • 集合输入或输出端口,意味着组件的实例将分别读取或写入将通过耦合到该端口的链路的集合(collection)的所有数据元素。对于在它们的集合端口之间具有单个链路的一对组件,通常允许下游组件在上游组件正在写入数据元素时读取这些数据元素,从而实现上游组件和下游组件之间的流水线并行性。如下面更详细描述的,数据元素也可以被重新排序,这使得并行化具有效率。在一些图形表示中,例如在编程图形界面中,这样的集合端口通常由组件处的正方形连接符指示。

[0061] • 标量输入或输出端口，意味着组件的一个实例将分别从耦合到端口的链路读取或写入至多一个数据元素。对于在其标量端口之间具有单个链路的一对组件，在上游组件已经完成执行之后通过使用作为控制传送的单个数据元素的传送来强制下游组件的串行执行。在一些图形表示中，例如在编程图形界面中，这样的标量端口通常由该组件处的三角形连接符指示。

[0062] • 控制输入或输出端口，类似于标量输入或输出，但是不需要发送数据元素，并且用于在组件之间通信控制的传送。对于具有在它们的控制端口之间的链路的一对组件，在上游组件已经完成执行之后强制下游组件的串行执行(即使那些组件也具有集合端口之间的链路)。在一些图形表示中，例如在编程图形界面中，这样的控制端口通常由组件处的圆形连接符指示。

[0063] 这些不同类型的端口实现了数据处理图的灵活设计，允许数据流和控制流与端口类型的重叠属性的强大组合。特别地，存在两种类型的端口(称为“数据端口”)：集合端口和标量端口，用于传送一些形式的数据；还存在两种类型的端口(称为“串行端口”)：标量端口和控制端口，用于强制串行执行。数据处理图通常将具有作为“源组件”而没有任何连接的输入数据端口的一个或多个组件，以及作为“宿(sink)组件”而没有任何连接的输出数据端口的一个或多个组件。一些组件将具有连接的输入和输出数据端口。在一些实施例中，不允许图具有循环，因此必须是有向无环图(DAG)。该特征可以用于利用DAG的某些特性，如下面更详细描述的。

[0064] 在数据处理图的组件上使用专用控制端口还使得能够灵活控制计算的不同部分，这种灵活控制使用某些其他控制流技术是不可能实现的。例如，能够在数据流图之间应用依赖性约束的作业控制解决方案无法提供由定义单个数据流图内的组件之间的依赖性约束的控制端口实现的精细粒度控制。此外，将组件分配到顺序运行的不同阶段的数据流图不允许对单个组件排序方面的灵活性。例如，使用简单阶段不可能的嵌套控制拓扑可以使用本文描述的控制端口和执行集来定义。这种更大的灵活性也可以潜在地通过允许更多的组件尽可能地并发运行来提高性能。

[0065] 通过以不同的方式连接不同类型的端口，开发人员能够在数据处理图的组件的端口之间指定不同类型的链路配置。例如，一种类型的链路配置可以对应于连接到相同类型的端口的特定类型的端口(例如，标量到标量链路)，另一种类型的链路配置可以对应于连接到不同类型的端口的特定类型的端口(例如，集合到标量链路)。这些不同类型的链接配置既用作开发人员在视觉上识别与数据处理图的一部分相关联的预期行为的方式，又用作向编译器/解释器120指示启用该行为所需的对应类型的编译进程的方式。虽然本文描述的示例使用用于不同类型的端口的独特形状以可视地表示不同类型的链路配置，但是系统的其他实现方式可以通过提供不同类型的链路并且为每种类型的链路分配独特的视觉指示符(例如，厚度，线型，颜色等)来区分不同类型的链路配置的行为。然而，为了用上面列出的三种类型的端口使用链路类型而不是端口类型来表示相同种类的链路配置，将存在多于三种类型的链路(例如，标量到标量、集合到集合、控制到控制、集合到标量、标量到集合、标量到控制等)。其他示例可以包括不同类型的端口，但是没有在数据处理图内可视地明确指示端口类型。

[0066] 编译器/解释器120执行准备数据处理图用于执行的过程。第一过程是执行集发现

预处理过程，用于识别组件的潜在的嵌套执行集的层级结构。第二过程是控制图生成过程，用于为每个执行集生成对应的控制图，编译器/解释器120将使用该控制图来形成控制代码，该控制代码将在运行时有效地实现状态机，用于控制每个执行集内的组件的执行。下面将更详细地描述这些过程中的每一个。

[0067] 具有至少一个输入数据端口的组件指定要对每个输入数据元素或集合(或其多个输入端口上的一元组的数据元素和/或集合)执行的处理。这种规范的一种形式是作为将要对一个或一元组输入数据元素和/或集合执行的过程。如果组件具有至少一个输出数据端口，则其可以产生对应的一个或一元组输出数据元素和/或集合。可以以高级的基于语句的语言(例如，使用Java源语句或例如在美国专利8,069,129“Editing and Compiling Business Rules(编辑和编译业务规则)”中使用的数据操作语言(DML))来指定这样的过程，或者可以以一些完全或部分编译的形式提供(例如，如Java Bytecode(Java字节码))。例如，组件可以具有工作过程，所述工作过程的自变量包括其输入数据元素和/或集合及其输出数据元素和/或集合，或更一般地，包括对这样的数据元素或集合或对过程或数据对象的引用(本文称为“句柄”)，所述引用用于获取输入并提供输出数据元素或集合。

[0068] 工作过程可以有各种类型。本文不希望限制可以指定的过程的类型，一种类型的工作过程根据记录格式指定对数据元素的离散计算。单个数据元素可以是来自表格(或其他类型的数据集)的记录，并且记录的集合可以是表中的所有记录。例如，具有单个标量输入端口和单个标量输出端口的组件的一种类型的工作过程包括接收一个输入记录，对该记录执行计算，以及提供一个输出记录。另一种类型的工作过程可以指定从多个标量输入端口接收的一元组输入记录如何被处理以形成在多个标量输出端口上发出的输出记录的一元组。

[0069] 由数据处理图指定的计算的语义定义固有地是平行的，因为它表示对由图定义的计算的处理的顺序和并发性的约束和/或无约束。因此，计算的定义不要求结果等同于计算步骤的某种顺序排序。另一方面，计算的定义确实提供了需要对计算的部分进行排序的特定约束以及对计算的部分的并行执行的限制。

[0070] 在数据处理图的讨论中，假定将组件的多个实例实现为运行时系统中的分开的“任务”，作为表示排序和并行约束的方式。在讨论基于图的规范本身的特性之后，将更全面地进行将数据处理图实现为基于任务的规范的更具体讨论，其实现与语义定义一致的计算。

[0071] 通常，数据处理图中的每个组件将在图的执行期间在计算平台中被多次实例化。每个组件的实例的数量可以取决于该组件被分配给多个执行集中的哪一个。当实例化组件的多个实例时，多于一个实例可以并行执行，并且不同的实例可以在系统中的不同计算节点中执行。组件的互连(包括端口的类型)决定由指定的数据处理图允许的并行处理的性质。

[0072] 尽管如下文所述，在组件的不同实例的执行之间通常不保持状态，但在系统中提供某些规定用于明确地引用可以跨越组件的多个实例的执行的持久存储。

[0073] 在工作过程指定如何处理单个记录以产生单个输出记录并且端口被指示为集合端口的示例中，可以执行组件的单个实例，并且迭代工作过程以处理连续的记录以生成连续的输出记录。在这种情况下，可能从迭代到迭代在组件内维持状态。

[0074] 在工作过程指定如何处理单个记录以产生单个输出记录并且端口被指示为标量端口的示例中，可以执行组件的多个实例，并且在工作过程的多次执行之间没有为不同的输入记录保持状态。

[0075] 此外，在一些实施例中，系统支持不遵循上面介绍的最细粒度规范的工作过程。例如，工作过程可以在内部实现迭代，例如，其通过标量端口接受单个记录并通过集合端口提供多个输出记录。

[0076] 如上所述，存在两种类型的数据端口：集合端口和标量端口，用于传送一些形式的数据；还存在两种类型的串行端口：标量端口和控制端口，用于强制串行执行。在一些情况下，一种类型的端口可以通过链路连接到另一类型的端口。这些情况中的一些将在下面描述。在某些情况下，一种类型的端口将链接到同一类型的端口。两个控制端口之间的链路(称为“控制链路”)在链接的组件之间施加串行执行排序，而不需要通过链路发送数据。两个数据端口之间的链路(称为“数据链路”)提供数据流，并且还在标量端口的情况下强制串行执行排序约束，并且在集合端口的情况下不需要串行执行排序。典型的组件通常具有至少两种端口，包括输入和输出数据端口(集合端口或标量端口)以及输入和输出控制端口。控制链路将上游组件的控制端口连接到下游组件的控制端口。类似地，数据链路将上游组件的数据端口连接到下游组件的数据端口。

[0077] 开发人员可以使用图形用户界面从一组组件中指定特定的数据处理计算，每个组件执行特定的任务(例如，数据处理任务)。开发人员通过在显示屏幕上显示的画布区域上组装数据处理图来进行这样的指定。这涉及将组件放置在画布上，用适当的链路连接它们的各个端口，以及适当地配置组件。以下简单示例说明了在具有单对集合端口和单对控制端口的组件的情况下的某些行为。

[0078] 图2a示出了被组装的数据处理图的一部分包括第一组件210A的示例，第一组件210A具有输入控制端口212A和输出控制端口214A以及输入集合端口216A和输出集合端口218A。控制链路220A，222A将输入控制端口212A和输出控制端口214A连接到数据处理图中其他组件的控制端口。类似地，数据链路224A，226A将输入集合端口216A和输出集合端口218A连接到数据处理图中其他组件的端口。集合端口216A，218A在图中表示为矩形，而控制端口212A，214A用圆形表示。

[0079] 通常，输入集合端口216A接收要由组件210A处理的数据，并且输出集合端口214提供已经由组件210A处理的数据。在集合端口的情况下，所述数据通常是未指定数量的数据元素的无序集合。在整体计算的特定实例中，集合可以包括多个数据元素，或单个数据元素，或没有数据元素。在一些实现方式中，集合与确定集合中的元素是无序的还是有序的(并且如果有序的话，什么决定顺序)的参数相关联。如下面将更详细地描述的，对于无序集合，数据元素由数据链路的接收侧的组件处理的顺序可以不同于数据链路的发送侧的组件提供这些数据元素的顺序。因此，在集合端口的情况下，它们之间的数据链路用作数据元素的“包”，从其中可以以任意顺序拉取数据元素，而不同于以特定顺序将数据元素从一个组件移动到另一个组件的“传送带”。

[0080] 控制链路用于在控制端口之间传送控制信息，其确定组件是否开始执行以及何时开始执行。例如，控制链接222A指示组件210B将在组件210A已经完成之后开始执行(即，以串行顺序)，或者指示组件210B将不开始执行(即，被“禁止”)。因此，虽然没有数据通过控制

链路发送，但是它可以被视为向接收侧的组件发送信号。发送该信号的方式可以根据实现方式而变化，并且在一些实现方式中可以涉及在组件之间发送控制消息。其他实现方式可以不涉及发送实际的控制消息，而是可以改为涉及直接调用进程或调取与由接收侧的组件表示的任务相关联的函数的进程(或者在禁止的情况下省略这种调用或函数调取)。

[0081] 链接控制端口的能力因此使得开发人员能够控制由数据处理图的不同组件表示的数据处理计算的不同部分之间的相对排序。另外，使用组件上的控制端口提供这种排序机制使得能够混合与数据流和控制流相关联的逻辑。实际上，这使得数据能够用于做出关于控制的决定。

[0082] 在图2A所示的示例中，控制端口连接到其他控制端口，并且数据端口连接到其他数据端口。然而，数据端口上的数据固有地携带两种不同种类的信息。第一种是数据本身，第二种是数据的存在。该第二种信息可以用作控制信号。结果，通过使标量数据端口能够连接到控制端口，可以提供额外的灵活性。

[0083] 图2B示出了示例性数据处理图230，其利用了由将标量端口连接到控制端口的能力所赋予的灵活性。

[0084] 数据处理图230的特征如下：标记为“计算日期信息”的第一组件231，标记为“制作每月报告?”的第二组件232，标记为“制作每周报告”的第三组件233，标记为“每月报告”的第四组件234，标记为“制作每周报告?”的第五组件235和标记为“每周报告”的第六组件236。数据处理图230执行过程，该过程总是产生每日报告、每日报告和每周报告、或所有三种报告。关于这些结果中的哪些将发生的决定取决于对由第一组件231提供的某些日期信息的评估。因此，图2B示出了有效地控制执行的数据的示例。

[0085] 当第一组件231将日期信息从输出标量端口提供到第二组件232的输入标量端口和第三组件233的输入标量端口时，执行开始。没有连接的输入控制端口的第二组件232立即工作。包括第三组件233的所有其他组件具有连接的输入控制端口，并且必须等待由适当的正控制信号激活。

[0086] 第二组件232检查该日期信息并确定是否适合制作每月报告。有两种可能的结果：需要每月报告，或者不需要。第二组件232和第三组件233都具有两个输出标量端口，并且都被配置为执行选择函数，该选择函数提供数据元素，该数据元素在一个输出标量端口(即，所选端口)上用作正控制信号并且在另一个输出标量端口上用作负控制信号。

[0087] 如果基于日期信息，第二组件232确定不需要每月报告，则第二组件232将数据元素从其底部输出标量端口发送到第三组件233的输入控制端口。该数据元素被解释为正控制信号，其向第三组件233指示第二组件232已经完成了对由第一组件231提供的数据的处理，并且第三组件233现在可以开始处理其接收的日期信息数据。

[0088] 另一方面，如果第二组件232基于由第一组件231提供的日期信息确定需要每月报告，则第二组件232改为将被解释为正控制信号的数据元素从其输出标量端口发送到第四组件234的输入控制端口。虽然数据元素不仅仅是控制信号，但是第四组件234将其视为正控制信号，因为它正被提供给其输入控制端口。第四组件234忽略数据元素中的实际数据，并且仅使用数据元素的存在作为正控制信号。

[0089] 第四组件234继续创建每月报告。在完成时，第四组件234从其输出控制端口输出控制信号到第三组件233的输入控制端口。这告诉第三组件233它(即第三组件233)现在可

以开始处理第一组件231提供给它的日期信息。

[0090] 因此，第三组件233将总是最终经由其输入标量端口处理由第一组件231提供的数据。唯一的区别在于哪个组件触发它开始处理：是第二组件232还是第四组件234。这是因为第三组件233上的两个输入控制端口将使用OR逻辑进行组合，使得在任一端口(或两者)接收的正控制信号将触发处理。

[0091] 图230的其余部分以基本上相同的方式操作，只是第三组件233接管第二组件232的角色，第六组件236接管第四组件234的角色。

[0092] 在被其输入控制端口处来自第二组件232或第四组件234的控制信号激活时，第三组件233通过将第一组件231连接到第三组件233的数据链路检查由第一组件231提供的日期信息。如果第三组件233基于日期信息确定不需要每周报告，则第三组件233将被解释为正控制信号的数据元素从其输出标量端口之一发送到第五组件235的输入控制端口。

[0093] 另一方面，如果第三组件233确定需要每周报告，则第三组件233将被解释为正控制信号的数据元素从其另一输出标量端口发送到第六组件236的输入控制端口。第六组件236继续创建每周报告。在完成时，第六组件236将被解释为正控制信号的数据元素从其输出标量端口发送到第五组件235的输入控制端口。

[0094] 因此，第五组件235将总是最终执行，唯一的区别是第三组件233还是第六组件236最终触发第五组件235开始执行。在从第三组件233或第六组件236接收到控制信号时，第五组件235创建每日报告。

[0095] 图2C还示出了标量和集合数据端口的使用。

[0096] 图2C示出了数据处理图240，其具有标记为“输入文件”的第一组件241，标记为“从请求获取文件名”的第二组件242，标记为“读取文件”的第三组件243，标记为“是坏记录?”的第四组件244，标记为“无效记录”的第五组件245，标记为“生成坏记录文件名”的第六组件246，标记为“任何验证错误?”的第七组件247和标记为“发送警报”的第八组件248。此图旨在将坏记录写入文件，并在检测到这种坏记录时发送警报。

[0097] 组件241和243是用作数据的源的组件的示例，并且组件245是用作数据的宿(接收器)的组件的示例。组件241和243使用可以以文件系统(诸如本地文件系统或分布式文件系统)中的各种格式中的任一种存储的输入文件作为其源。输入文件组件读取文件的内容，并从该文件产生记录的集合。标量输入端口(如组件243所示)提供指定要读取的文件的位置(例如，路径或统一资源定位符)和要使用的记录格式的数据元素。在一些情况下，位置和记录格式可以作为参数提供给输入文件组件，在这种情况下，输入标量端口不需要连接到任何上游组件，并且不需要显示(对于组件241也是如此)。集合输出端口(如组件241和243两者上的所示)提供记录的集合。类似地，输出文件组件(诸如组件245)将通过输入集合端口接收的记录的集合写到输出文件(其位置和记录格式可以可选地由输入标量端口指定)。输入文件组件或输出文件组件还可以包括链接到另一组件(诸如组件245)的控制端口的控制输入或输出端口。

[0098] 在所示的数据处理图240中，较大的虚线矩形内的组件是执行集(execution set)的一部分。此执行集包含嵌套在其中的另一个执行集。该嵌套执行集也在虚线矩形内示出，其仅包含第四组件244。下面更详细地讨论执行集。

[0099] 在操作中，第一组件241读取输入文件。当第一组件241正在执行时，第一组件241

经由从输出集合数据端口到第二组件242的输入集合数据端口的数据链路将输入文件内的记录的集合提供给第二组件。第二组件242的不同实例和其他下游组件(它们在同一执行集中)可以针对集合中的每个记录执行,如下面将更详细描述的。由于第二组件242没有连接到其控制输入的任何东西,所以第二组件242立即开始处理。在完成时,第二组件242在其输出标量端口上提供文件名。该文件名在相应输入标量端口处由第三组件243和第六组件246两者接收。

[0100] 第三组件243立即读取由文件名标识的文件,并且在输出集合端口上提供文件的内容以用于递送到第四组件244的实例的输入标量端口。同时,第六组件246接收相同的文件名并输出另一文件名,其提供到连接于第五组件245和第七组件247的对应输入标量端口的输出标量端口上。

[0101] 在从第六组件246接收到文件名和从第四组件244接收到坏记录时,第五组件245将坏记录写入文件名由第六组件246标识的输出文件。

[0102] 第七组件247是唯一一个在其数据输入端口接收数据时没有准备好被启动执行的组件。当第五组件245完成对输出文件的写入时,第五组件245将控制信号从其控制输出端口发送到第七组件247的输入控制端口。如果第七组件247确定存在错误,则第七组件247提供数据到第八组件248的输入标量端口。这使得第八组件248生成警报。这提供了控制端口用于限制数据处理图中的某些组件的执行的示例。

[0103] 应当清楚的是,当一组多个上游组件都已达到特定状态时,基于一个组件的状态控制另一个组件中的处理的能力使得该组件可以控制处理。例如,数据处理图可以支持去往或来自同一控制端口的多个控制链路。可替代地,在一些实现方式中,组件可以包括多个输入和输出控制端口。默认逻辑可以被编译器/解释器120应用。开发人员还可以提供用于确定如何组合控制信号的定制逻辑。这可以通过适当地布置组合逻辑以应用于上游组件的各种控制链路,并且仅当达到某个逻辑状态时(例如,在默认OR逻辑的情况下,当所有上游组件已经完成时,并且当至少一个上游组件已经发送激活控制信号时)触发组件的启动来实现。

[0104] 通常,控制信号可以是触发处理开始或触发处理禁止的信号。前者是“正控制信号”,后者是“负控制信号”。然而,如果使用组合逻辑来确定是否应该调用任务(触发处理的开始),则逻辑可以“反转”通常的解释,使得仅当所有输入提供负控制信号时调用任务。通常,组合逻辑可以提供用于确定与控制图对应的状态机中的下一状态的任意“真值表”,下面将更详细描述。

[0105] 未连接的控制端口可以被分配默认状态。在一个实施例中,默认状态对应于正控制信号。如下面更详细地描述的,这可以通过在表示数据处理图的控制图中使用隐式开始组件和结束组件来实现。

[0106] 各种组件上的不同类型的数据端口允许数据以不同的方式在组件之间通过链路传递,这取决于链接这些组件的输入和输出端口的类型。如上所述,标量端口表示至多单个数据元素(即,0或1个数据元素)的生产(对于标量输出端口)或消耗(对于标量输入端口)。而集合端口表示一组潜在的多个数据元素的生产(对于集合输出端口)或消耗(对于集合输入端口)。通过在单个数据处理图中支持两种类型的数据端口,可以更有效地分配计算资源,并且可以在任务之间生成更复杂的控制流和数据流,从而允许开发人员容易地指示期

望的行为。

[0107] 参考图3A,数据处理图300包括一连串三个连接的组件:第一组件(A1)302,第二组件(B1)304和第三组件(C1)306。第一组件包括集合型输入端口308以及标量型输出端口310。第二组件304包括标量型输入端口312和标量型输出端口314。第三组件包括标量型输入端口316和集合型输出端口318。

[0108] 第一链路320将第一组件302的标量输出端口310连接到第二组件304的标量输入端口312,这允许数据在第一组件302和第二组件304之间传递,并且同时强制第一组件302和第二组件304的串行执行。类似地,第二链路322将第二组件304的标量输出端口314连接到第三组件306的标量输入端口316,这允许数据在第二组件304和第三组件306之间传递,并且强制第二组件304和第三组件306的串行执行。

[0109] 由于图3A中标量端口的互连,第二组件304仅在第一组件302完成之后开始执行(并且在第一链路320上传递单个数据元素),并且第三组件306仅在第二组件304完成之后开始执行(并且在第二链路322上传递单个数据元素)。也就是说,数据处理图中三个组件中的每一个以严格的顺序A1/B1/C1运行一次。

[0110] 在一些示例中,一个或多个组件可以被置于禁止状态,这意味着一个或多个组件不执行,并因此不将任何数据元素从其输出端口传递出去。例如,通过确保将不执行任何有用处理的组件不需要专用于它们的计算资源(例如,进程或存储器),使得能够禁止组件,避免浪费资源。具有仅连接到被禁止组件的输出端口的标量输入端口的任何组件不会执行,因为它们不接收数据。例如,如果第一组件302被置于禁止状态,则第二组件304的标量输入端口312不从第一组件302的标量输出端口310接收数据,因此不执行。由于第二组件304不执行,因此第三组件306的标量输入端口316不接收来自第二组件304的标量输出端口314的数据,并且也不执行。因此,在两个标量端口之间传递的数据也用作正控制信号,类似于在两个链接的控制端口之间发送的信号。

[0111] 在图3A的示例性数据处理图中,第一组件302的输入端口308和第三组件318的输出端口恰好是集合端口,其对连接它们的标量端口强加的第一组件302、第二组件304和第三组件306的串行执行行为没有影响。

[0112] 通常,集合端口用于在组件之间传递数据元素的集合(collection),并且同时可以向运行时系统授予重新排序该集(set)中的数据元素的许可。允许对无序集合的数据元素重新排序,因为对从一个数据元素到另一个数据元素的计算的状态没有依赖性,或者如果存在处理每个数据元素时被访问的全局状态,则最终状态独立于处理这些数据元素的顺序。此重新排序的许可提供了将关于并行化的决策推迟到运行时的灵活性。

[0113] 参考图3B,数据处理图324包括一连串三个连接的组件:第一组件(A2)326、第二组件(B2)328和第三组件(C2)330。第一组件326包括集合型输入端口332和集合型输出端口334。第二组件328包括集合型输入端口336和集合型输出端口338。第三组件330包括集合型输入端口340和集合型输出端口342。

[0114] 三个组件326、328、330中的每一个指定如何处理一个或多个输入元素的集合以生成一个或多个输出元素的集合。在特定输入元素和特定输出元素之间并不必然存在一一对应关系。例如,第一组件326和第二组件328之间的数据元素344的第一集合中的多个数据元素可以不同于第二组件328和第三组件330之间的数据元素346的第二集合中的多个数据元

素。对集合端口之间的连接施加的唯一约束是集合中的每个数据元素从一个集合端口传递到另一个集合端口，同时允许第一组件326和第二组件328之间以及第二组件328和第三组件330之间相对于它们被处理的顺序任意重新排序。可替代地，在其他示例中，集合端口可以可选地被配置为保持顺序。在该示例中，三个组件326、328、330一同启动并且并发运行，从而允许流水线并行性。

[0115] 参照图1描述的编译器/解释器120被配置为识别集合端口到集合端口连接，并且以适合于正在执行的计算的方式将计算转换为可执行代码。集合数据链路的无序性质使得编译器/解释器120在如何实现这一点方面具有灵活性。例如，如果对于第二组件328，恰好基于单个输入元素计算每个输出元素(即，没有跨数据元素维持的状态)，则编译器/解释器120可以允许运行时系统通过对每个数据元素实例化多达一个组件的实例来动态地并行化数据元素的处理(例如，取决于运行时可用的计算资源)。可选地，在特殊情况下，可以在具有多个输入集合端口的组件中跨数据元素维护状态。但在一般情况下，可以允许运行时系统对组件的任务并行化。例如，如果运行时系统检测到没有被维持的全局状态，则可以允许对任务并行化。一些组件还可以被配置为支持维持状态，在这种情况下可以不允许并行化。如果集合是无序的，则不需要在数据元素之间保持顺序意味着第二组件328的每个实例一旦可用就可以向第三组件330提供其输出数据元素，并且第三组件330可以在第二组件328的所有实例已经完成之前开始处理这些数据元素。

[0116] 在一些示例中，图开发者可以明确地指示可以通过将一个组件的集合型输出端口连接到另一个组件的标量型输入端口来动态地并行化数据集合中数据元素的处理。这种指示还要求在集合的不同元素的处理之间无需维持状态。参考图3C，数据处理图348包括一连串三个连接的组件：第一组件(A3)350、第二组件(B3)352和第三组件(C3)354。第一组件350包括集合型输入端口356和集合型输出端口358。第二组件352包括标量型输入端口360和标量型输出端口362。第三组件354包括集合型输入端口364和集合型输出端口366。

[0117] 第一组件的集合型输出端口358通过第一链路368连接到第二组件352的标量型输入端口360，并且第二组件352的标量型输出端口362通过第二连接370连接到集合型输入端口364。如下面更详细描述的，从集合型输出端口到标量型输入端口的链路意味着进入执行集的入口点，并且从标量型输出端口到集合型输入端口的链路意味着执行集的出口点。非常普遍的情况是，如下面更详细地描述的，包括在执行集中的多个组件可以由运行时控制器动态地并行化以处理来自数据元素集合的数据元素。

[0118] 在图3C中，第一组件350的集合型输出端口358和第二组件352的标量型输入端口360之间的链路368意味着进入执行集的入口点。第二组件352的标量型输出端口362和第三组件354的集合型输入端口364之间的链路370意味着执行集的出口点。也就是说，第二组件352是执行集中的唯一组件。

[0119] 由于第二组件352包括在执行集中，因此针对从第一组件350的集合型输出端口358接收的每个数据元素启动第二组件352的单独实例。至少一些单独的实例可以并行地运行，这取决于直到运行时才能做出的决策。在该示例中，第一组件(350)和第三组件(354)一同启动并且并发运行，而第二组件(352)针对通过链路368接收的集合内的每个数据元素运行一次。可替代地，针对集合中每多个数据元素的元组，第二组件352运行一次。

[0120] 2执行集

[0121] 如上面参照图1所描述的，编译器/解释器120对数据处理图实施执行集发现预处理过程，以准备用于执行的数据处理图。在一般意义上，如本文所使用的，术语“执行集”是指可以作为单元被调用并应用于数据的一部分(诸如输出集合端口的数据元素的一部分)的一个或多个组件的集合。因此，对于每个输入数据元素(或呈现给执行集的一个或多个输入端口的多个输入数据元素的元组)实施执行集中的每个组件的至多一个实例。在执行集内，通过到标量和控制端口的链路强加排序约束，只要不违反排序约束，就允许执行集中组件的并行执行。由编译器/解释器120为执行集准备的代码可以包括指示在执行代码时如何实施与组件相对应的任务(例如，并行度)的嵌入信息(例如，注释或修改符)。在对接收到的集合中多个数据元素的元组执行执行集的一个实例的示例中，所述元组可包括例如固定数量的数据元素或共享一些特性(例如，公共键值)的多个数据元素。在存在被允许并行执行的至少一些组件的示例中，可以使用多个任务来实现执行集，例如，用于作为整体的执行集的任务，以及用于一个或多个组件的实例的并发执行的一个或多个子任务。因此，表示执行集的不同实例的任务本身可以被分解成甚至更细粒度的任务，例如具有可以并发执行的子任务。用于不同执行集的任务通常可以独立并行地执行。因此，如果大型数据集有例如一百万条记录，则可能有一百万个独立的任务。一些任务可以在计算平台150的不同节点152上执行。任务可以使用可以被有效地并发执行的轻量线程来执行，甚至在单个节点152上也是如此。

[0122] 通常，由分配算法标识的执行集(即，除根执行集之外的执行集)通过在执行集的边界处的“驱动”标量数据端口来接收数据元素。对于在执行集的驱动输入标量数据端口处接收的每个数据元素，执行集内的每个组件执行一次(如果激活)或根本不执行(如果禁止)。执行集的多个实例可以被实例化并且并行地执行以处理从上游集合端口对执行集可用的多个数据元素。可以在运行时确定执行集的并行度(并且包括不使执行集并行化的可能决策)，并且并行度仅受到运行时可用的计算资源的限制。执行集的独立实例的各个输出在执行集的(多个)输出端口处被聚集，而不管顺序如何，并且被提供给下游组件。可替代地，在其他实施例中，可以识别不同于根执行集的不需要驱动输入标量数据端口的执行集(在一些情况下，基于用户输入)。这样的没有驱动输入标量数据端口的执行集如果合适的话可以使用本文所述的过程在单个实例中执行(例如，下面描述的锁定的执行集)或在多个实例中并行地执行。例如，可以设置确定执行集将被执行的次数和/或将被执行的执行集的并行实例的数量的参数。

[0123] 非常常见的情况是，执行集发现过程使用确定数据处理图内的组件的子集的分配算法，所述子集将被应用为对数据元素的无序集合的输入元素的集。分配算法遍历数据处理图并基于分配规则将每个组件分配给子集。如在以下示例中清楚可见的，给定数据处理图可以包括嵌套在执行集层级结构的不同级别中的多个执行集。

[0124] 在本文所述的数据处理图中，存在两种类型的数据端口：标量数据端口和集合数据端口。通常，如果一对链接的组件(即，图4A到图4D的上游组件A402和下游组件B 404)通过相同类型的端口之间的链路连接，则在默认情况下它们将在相同的执行集中(除非它们由于另一个原因在不同的执行集中)。在图4A中，组件A402具有输出端口406，该端口具有标量类型，并且组件B 404具有输入端口408，该端口具有标量类型。由于组件A 402和组件B 404之间的链路410连接两个标量型端口，因此在该示例中组件A 402和组件B 404处于相同

的执行集中。在图4A中，由于组件A 402和组件B 404之间的链路是标量到标量链路，所以0个数据元素或1个数据元素通过链路410在上游组件A 402和下游组件B 404之间传递。在上游组件A 402的处理完成时，通过链路410传递数据元素，除非上游组件A 402被禁止(如上所述)，在这种情况下没有数据元素通过链路410传递。

[0125] 参考图4B，组件A 402具有输出端口412，该端口具有集合类型，并且组件B 404具有输入端口414，该端口具有集合类型。由于组件A 402和组件B 404之间的链路410连接两个集合型端口，因此在该示例中组件A 402和组件B 404也处于相同的执行集中。在图4B中，由于组件A 402和组件B 404之间的链路410是集合到集合链路，所以通过链路410在上游组件和下游组件之间传递一组数据元素。

[0126] 当链路两端的端口类型之间存在不匹配时，执行集层级结构的级别中存在隐性更改。具体地，不匹配的端口表示在执行集层级结构的特定级别处的执行集的入口点或出口点。在一些示例中，执行集入口点被定义为集合型输出端口和标量型输入端口之间的链路。在图4C中，在组件A 402和组件B 404之间的链路410处示出了执行组入口点424的一个示例，这是因为组件A 402的输出端口416是集合型端口，而组件B 404的输入端口418是标量型端口。

[0127] 在一些示例中，执行集出口点被定义为标量型输出端口和集合型输入端口之间的链路。参考图4D，在组件A 402与组件B 404之间的链路410处示出了执行组出口点426的一个示例，这是因为组件A 402的输出端口420是标量型端口，而组件B 404的输入端口422是集合型端口。

[0128] 在编译器/解释器120进行编译和/或解释之前实现的分配算法使用执行集入口点和执行集出口点来发现数据处理图中存在的执行集。

[0129] 2.1基于堆栈的分配算法

[0130] 为了说明的目的，在第一示例中，数据处理图具有简单的一维图结构，并且使用基于堆栈的算法示出更简单的分配算法。在基于堆栈的分配算法中，数据处理图中的每个组件用一个或多个“ID字符串”标记，该“ID字符串”由以分隔字符‘/’分隔的整数组成。分隔字符‘/’出现在给定组件的ID字符串中的次数确定执行集层级结构中的组件的级别。在一些示例中，组件可以具有多个输入链路，并且因此可以具有多个ID字符串。在这种情况下，算法具有下面更详细描述的规则，用于确定使用哪个ID字符串。

[0131] 在基于堆栈的分配算法的一个示例中，编译器/解释器120根据以下过程在上游到下游方向上遍历数据处理图。最初，最上游的组件用ID字符串‘0’标记，指示它是执行集层级结构中根级别上的组件。

[0132] 然后遍历从最上游组件到最下游组件的路径上的链路和组件。如果遇到上游组件的集合型输出端口和下游组件的集合型输入端口之间的链路，则上游组件的ID字符串被传播到下游组件。类似地，如果遇到上游组件的标量型输出端口和下游组件的标量型输入端口之间的链路，则上游组件的ID字符串被传播到下游组件。

[0133] 如果遇到上游组件的集合型输出端口和下游组件的标量型输入端口之间的链路，则向下游组件分配包括上游组件的标记其末尾附加‘/n’的标记，其中n是<所有现有ID字符串整数中最大的整数>+1。如果遇到上游组件的标量型输出端口和下游组件的集合型输入端口之间的链路，则向下游组件分配包括上游组件的标记去除其最右边的ID字符串整数

(以及其分隔字符‘/’)的标记。

[0134] 在一些示例中,各种条件可以被认为是非法的,并且将导致算法中的错误(例如,如果组件在执行集层级结构中相同级别上具有两个不同的ID字符串,或者在执行集中存在循环)。

[0135] 参考图5,将上述基于堆栈的分配算法应用于示例性数据处理图550,导致发现两个执行集(除了根以外,“0级”执行集551):第一“1级”执行集570和嵌套在第一“1级”执行集670内的第二“2级”执行集572。为了实现两个执行集570、572的发现,基于堆栈的分配算法首先标记最上游组件,具有ID字符串‘0’的第一数据集656。然后基于堆栈的分配算法遍历通过数据处理图550的一维路径的组件。在遍历该路径时,基于堆栈的分配算法首先遍历从第一数据集556到第一组件558的链路。由于第一数据集556的输出端口是集合型输出端口,并且第一组件558的输入端口是标量型输入端口,所以第一组件558被分配ID字符串‘0/1’,其是第一数据集556的ID字符串在其末尾附加‘/1’的标记,其中1是所有现有ID字符串整数的最大值+1。通常,将‘/1’附加到第一组件558的ID字符串是从根“0级”执行集551到“1级”执行集570的转换的指示。在一些示例中,该转换使用第一执行集入口点指示符557表示。

[0136] 然后,分配算法遍历从第一组件558到第二组件560的链路。由于第一组件558的输出端口是集合型输出端口并且第二组件560的输入端口是标量型输入端口,所以第二组件560被分配ID字符串‘0/1/2’,其是第一组件558的ID字符串其末尾附加‘/2’,其中2是所有现有ID字符串整数的最大值+1。通常,将‘/2’附加到第二组件560的ID字符串是从“1级”执行集570到“2级”执行集572的转换的指示。在一些示例中,该转换使用第二执行集入口点指示符559表示。

[0137] 然后,分配算法遍历从第二组件560到第三组件562的链路。由于第二组件560的输出端口是标量型输出端口,并且第三组件562的输入端口是标量型输入端口,所以第二组件560的ID字符串(即,‘0/1/2’)被传播到第三组件562。

[0138] 然后,分配算法遍历从第三组件562到第四组件564的链路。由于第三组件562的输出端口是标量型输出端口,并且第四组件564的输入端口是集合型输入端口,所以第四组件被分配ID字符串‘0/1’,其是第三组件562的ID字符串去除其最右边ID字符串‘2’(及其分隔字符‘/’)的ID字符串。通常,从第三组件562的ID字符串中去除‘/2’是从“2级”执行集572到“1级”执行集570的转换的指示。在一些示例中,该转换使用第一执行集出口点指示符563表示。

[0139] 然后,分配算法遍历从第四组件564到第五组件566的链路。由于第四组件564的输出端口是标量型输出端口,并且第五组件566的输入端口是集合型输入端口,所以第五组件566被分配ID字符串‘0’,其是第四组件564的ID字符串去除其最右边的ID字符串整数(及其分隔字符‘/’)。通常,从第四组件564的ID字符串中去除‘/1’是从“1级”执行集570到根“0级”执行集551的转换的指示。在一些示例中,该转换使用第二执行集出口点指示符565来表示。

[0140] 最后,分配算法遍历从第五组件566到第二数据集568的链路。由于第五组件566的输出端口是集合型输出端口,并且第二数据集568的输入端口是集合型输入端口,所以第五组件566的ID字符串(即,‘0’)被传播到第二数据集568。

[0141] 在一些示例中,除了入口点指示符和出口点指示符之外,可以使用用户界面内的

附加视觉提示在视觉上表示数据元素的集合的流和个体标量数据元素之间的变化。例如，表示链路的线在集合端口和指示符之间可以更粗，而在指示符和标量端口之间更细。

[0142] 基于堆栈的分配算法的结果包括数据处理图550的版本，其中每个组件用ID字符串标记。在图5的示例中，第一数据集556、第二数据集568和第五组件566都用ID字符串‘0’标记。第一组件558和第四组件564用ID字符串‘0’标记。第二组件560和第三组件562各自用ID字符串‘0/1/2’标记。

[0143] 每个唯一ID字符串表示执行集层级结构中的唯一执行集。具有ID字符串‘0’的那些组件被分组到执行层级中的根“0级”执行集551中。具有ID字符串‘0/1’的那些组件被分组到嵌套在根执行集651中的“1级”执行集670中(其中‘0/1’可以被读取为嵌套在执行集0中的执行集1)。具有ID字符串‘0/1/2’的那些组件被分组到“2级”执行集572，其嵌套在根“0级”执行集551和“1级”执行集570内。

[0144] 2.2基于全局映射的分配算法

[0145] 在一些示例中，对于更一般的数据处理图，基于堆栈的分配算法可能不足以正确地确定执行集的层级结构。例如，在一般的数据处理图中，任何给定的组件可以具有多个输入端口和/或多个输出端口，使一般的数据处理图与基于堆栈的方法不兼容。在这样的示例中，使用基于全局映射的分配算法来确定执行集层级结构。

[0146] 基于全局映射的分配算法利用数据处理图被限制为有向无环图的事实。可以使用拓扑排序后的顺序来处理有向无环图，以确保图的每个组件仅在紧邻该组件的上游的所有组件被处理之后得到处理。由于已知已经处理了组件的紧接上游的所有组件，所以可以通过选择该组件直接上游的(在执行集层级结构中)最深嵌套组件的ID字符串来确定该组件的ID字符串。

[0147] 在一些示例中，基于全局映射的分配算法使用诸如Kahn算法的标准拓扑排序算法来获得给定数据处理图的拓扑排序后的顺序。Kahn算法通过以下伪代码概述：

[0148] L←将要包含被排序元素的空列表

[0149] S←没有进入边的所有节点的集合

[0150] 只要S是非空的，进行以下动作：

[0151] 从S去除节点n

[0152] 将n添加到L的尾部

[0153] 对于具有n到m的边e的每个节点m，进行以下动作：

[0154] 从图中去除边e

[0155] 如果m没有其它进入边，则

[0156] 将m插入S

[0157] 如果图具有多个边，则

[0158] 返回错误(图具有至少一个循环)

[0159] 否则

[0160] 返回L(拓扑排序后的顺序)

[0161] 在确定拓扑排序后的顺序之后，基于全局映射的分配算法以拓扑排序后的顺序遍历数据处理图的组件，以确定每个组件的适当ID字符串(或简称ID号)。特别地，当遍历组件时，数据处理图的每个组件将其ID字符串复制到其输出端口。直接位于上游组件下游且没

有由于执行集入口点或执行集出口点而与上游组件分离的组件从上游组件的输出端口读取ID字符串，并使用该ID字符串作为其ID字符串。

[0162] 对于由于执行集入口点而与下游组件分离的上游组件，在执行集入口点处分配新的ID字符串，并将其提供给下游组件以用作其ID字符串。上游组件的ID字符串到下游组件的ID字符串的映射(即，父/子映射)被存储在全局映射数据存储中以供以后使用。

[0163] 对于由于执行集出口点而与下游组件分离的上游组件，上游组件的输出端口处的ID字符串由执行集出口点读取。然后查询全局映射数据存储以确定输出端口处的ID字符串的父ID字符串。将父ID字符串提供给下游组件以用作其ID字符串。

[0164] 参考图6，使用上述基于全局映射的分配算法来分析示例性一般二维数据处理图628的一个示例。数据处理图628包括第一数据集(D1)632，第一组件(C1)638，第二组件(C2)640，第三组件(C3)645，第四组件(C4)646，第五组件(C5)642，第六组件(C6)644和第二数据集(D2)634。在将ID字符串分配给数据处理图表628的各个组件之前，将拓扑排序算法(例如，Kahn算法)应用于数据处理图，得到以下拓扑排序后的顺序:D1，C1，C2，C3，C4，C5，C6，D2。

[0165] 利用确定的拓扑排序后的顺序，基于全局映射的分配算法以拓扑排序后的顺序遍历数据处理图的组件，以确定每个组件的适当ID字符串，从而导致发现“1级”执行集630和“2级”执行集631(除了根“0级”执行集以外)。为了实现两个执行集630、631的发现，基于全局映射的分配算法首先标记最上游组件，具有ID字符串‘0’的第一数据集(D1)632。然后基于堆栈的分配算法以拓扑排序后的顺序遍历数据处理图628的组件和链路。

[0166] 基于全局映射的分配算法首先遍历从第一数据集(D1)632到第一组件(C1)638的链路。由于第一数据集(D1)632的输出端口是集合型输出端口，并且第一组件(C1)638的输入端口是集合型输入端口，所以没有标识执行集入口点或出口点，并且从第一数据集(D1)632的输出端口读取第一数据集(D1)632的ID字符串(即‘0’)并且将该ID字符串分配给第一组件(C1)638。

[0167] 然后，分配算法遍历第一组件(C1)638和第二组件(C2)640之间的链路。由于第一组件(C1)638的输出端口是集合型输出端口，并且第二组件(C2)640的输入端口是标量型输入端口，所以在两个组件638、640之间标识第一执行集入口点639。在第一执行集入口点639处，分配新的ID字符串(即，‘1’)，并且该ID字符串被分配作为第二组件(C2)640的ID字符串。第一执行集入口点639的父ID字符串(即‘0’)到第一执行集入口点639的子ID字符串(即‘1’)的映射653被存储在全局映射数据存储649中以供以后使用。

[0168] 然后，分配算法遍历从第二组件(C2)640到第三组件(C3)645的链路。由于第二组件(C2)640的输出端口是集合型输出端口，并且第三组件645的输入端口是标量型输入端口，所以在两个组件640、645之间标识第二执行集入口点641。在第二执行集入口点641处，分配新的ID字符串(即，‘2’)并将其分配作为第三组件(C3)645的ID字符串。第二执行集入口点641的父ID字符串(即，‘1’)到第二执行集入口点641的子ID字符串(即‘2’)的映射651被存储在全局映射数据存储649中以供以后使用。

[0169] 然后，分配算法遍历从第三组件(C3)645到第四组件(C4)646的链路。由于第三组件(C3)645的输出端口是集合型输出端口，并且第四组件(C4)646的输入端口是集合型输入端口，所以未识别执行集入口点或出口点，并且从第三组件(C3)645的输出端口读取第三组

件(C3)645的ID字符串(即，‘2’)645并且将其分配给第四组件(C4)646。

[0170] 然后，分配算法遍历从第四组件(C4)646到第五组件(C5)642的链路。由于第四组件(C4)646的输出端口是标量型输出端口，并且第五组件(C5)642的输入端口是集合型输入端口，所以在两个组件646、642之间识别第一执行集出口点647。在第一执行集出口点647处，从第四组件(C4)646的输出端口读取第四组件(C4)646的ID字符串并且该ID字符串用于查询全局映射数据存储649。全局映射数据存储649返回与第二执行集入口点641相关联地存储的父子关系651(即，‘1/2’)。父/子关系651中的父ID字符串(即，‘1’)被分配作为第五组件(C5)642的ID字符串。

[0171] 然后，分配算法遍历从第五组件(C5)642到第六组件(C6)644的链路。由于第五组件(C5)642的输出端口是标量型输出端口，并且第六组件(C6)644的输入端口是集合型输入端口，所以在两个组件642、644之间识别第二执行集出口点643。在第二执行集出口点643处，从第五组件(C5)642的输出端口读取第五组件(C5)642的ID字符串并且该ID字符串用于查询全局映射数据存储649。全局映射数据存储649返回与第一执行集进入点639相关联存储的父/子关系653(即，‘0/1’)。父/子关系653中的父ID字符串(即，‘0’)被分配作为第六组件(C6)644的ID字符串。

[0172] 最后，分配算法遍历从第六组件(C6)644到第二数据集(D2)634的链路。由于第六组件(C6)644的输出端口是集合型输出端口，并且第二数据集(D2)634的输入端口是集合型输入端口，所以没有识别执行集入口点或出口点，并且从第六组件(C6)644的输出端口读取第六组件(C6)644的ID字符串(即‘0’)并且将其分配给第二数据集(D2)634。

[0173] 基于全局映射的分配算法的结果包括数据处理图628的版本，其中每个组件用ID字符串标记。在图6的示例中，第一数据集(D1)632，第一组件(C1)638，第六组件(C6)644和第二数据集(D2)634都用ID字符串‘0’标记。第二组件(C2)640和第五组件(C5)642都用ID字符串‘1’标记。第三组件(C3)645和第四组件(C4)646都用ID字符串‘2’标记。

[0174] 每个唯一的ID字符串表示执行集层级结构中的唯一执行集。具有ID字符串‘0’的那些组件被分组到执行层级结构中的根“0级”执行集629中。具有ID字符串‘1’的那些组件被分组到嵌套在根执行集629内的“1级”执行集630中。具有ID字符串‘2’的那些组件被分组到“2级”执行集631，“2级”执行集631嵌套在根“0级”执行集629内，并且进一步嵌套在“1级”执行集630内。

[0175] 2.3用户定义的执行集

[0176] 在上述示例中，(多个)分配算法用于自动发现存在于数据处理图中的执行集，而无需任何用户干预。然而，在一些示例中，用户可能需要除了由分配算法提供的功能之外的功能。在这种情况下，用户可以明确地添加执行集入口点和出口点，以明确定义执行集开始和/或结束的位置。参考图7，数据处理图776包括第一数据集774，第一组件778，第二组件780和第二数据集790。将上述分配算法应用于数据处理图776将导致发现包括第一组件778和第二组件780的单个执行集。然而，在这种情况下，用户已明确定义了用于数据处理图776的两个执行集(即，第一执行集782和第二执行集786)。具体地，用户已经将执行集出口点组件784插入到从第一组件778的输出端口出来的链路中，并且已经将执行集入口点788插入到进入第二组件780的输入端口的链路中。通过将执行集出口点784和执行集入口点788添加到第一组件778和第二组件780之间的链路，用户实质上已经将原本单个执行集分解为两

个单独的执行集782、786。

[0177] 在一些示例中，用户为数据处理图定义所有执行集入口点和出口点。在其他示例中，用户定义一些执行集入口点和出口点，然后将其留给分配算法以发现数据处理图的其余执行集入口点和出口点。

[0178] 2.4相同集关系

[0179] 在一些示例中，用户可能希望明确地指定给定组件属于哪个执行集。例如，如图8A所示，数据处理图892包括从创建数据组件896和读表组件898接收数据元素的第一执行集894。这些组件类似于输入文件组件，不同之处在于它们具有用于它们提供的数据元素的集合的不同的源。对于创建数据组件896，没有用来指定文件位置的标量输入端口，而是存在用来指定多个记录数据元素将如何产生的(可选的)标量输入端口，并且还存在用来指定每个数据元素将如何生成的参数。对于读表组件898，没有用来指定文件位置的标量输入端口，而是存在用来指定数据库中的表的(可选的)标量输入端口。第一执行集894包括第一组件891和第二组件893，它们一起处理来自创建数据组件896和读表组件898的数据元素，以生成提供给第一数据集899的输出。

[0180] 在图8A中，读表组件898在第一执行集894外部，意味着它运行一次，并从其集合型输出端口输出数据元素的集合。数据元素的集合遍历第一执行集894的边界，并被提供给第一组件891上的集合型输入端口。对于执行集894中的组件的每个并行实例，在第一组件891上的集合型输入端口处创建数据元素集合的副本。通常，无论链路来自集合端口、标量端口还是控制端口，被分配给不同执行集的组件之间的链路将数据元素或控制元素复制到流入执行集的链路的所有实例，并且从流出执行集的链路的所有实例聚集数据元素或控制元素。数据元素被聚集成集合，并且控制元素被聚集成向量，该向量可以根据下游组件的控制逻辑被适当地处理(包括可能将其标示为错误)。

[0181] 参考图8B，在一些示例中，用户可以要求对执行集894中的组件的每个并行实例执行读表组件898。为了实现该功能，用户可以在读表组件898和第一组件891之间指定“相同集”关系。作为用户指定“相同集”关系的结果，读表组件898被移动到与第一组件891相同的执行集(即，第一执行集894)。由于读表组件898被包括在第一执行集894中，所以第一执行集894中的组件的每个并行实例执行读表组件898的实例。

[0182] 在一些示例中，用户可以通过从与源执行集相关联的菜单选择目标执行集，或者通过将组件从源执行集拖动到目标执行集(例如，经由下面更详细描述的用户界面)来指定“相同集”关系。在一些示例中，实施错误检查以验证被拖拽的组件可以合法地位于目标执行集中。例如，可以在要彼此具有“相同集”关系的任何两个组件上强加的一个可能的要求是，必须具有通过数据处理图的包括这些组件的至少一个路径。

[0183] 2.5集合数据复制

[0184] 在一些示例中，执行集中的多个组件可以各自具有经由执行集入口点连接到上游组件的单个集合输出端口的标量输入端口。类似地，执行集中的多个组件可以各自具有连接到执行集下游的组件的单个集合输入端口的标量输出端口。

[0185] 在一些示例中，为了从多个组件的集合型输出端口向标量输入端口提供相同的数据，执行集入口点从集合为每个标量输入端口创建每个数据元素的复制本，并且将复制本提供到它们相应的标量输入端口。类似地，为了合并由多个组件的标量输出端口(来自执行

集的不同的相应迭代)输出的数据,执行集出口点可以从多个标量输出端口接收输出数据元素,合并输出数据元素,然后将合并的输出数据元素提供到下游组件的集合输入端口。通常,下游组件的集合输入端口被配置为处理合并的数据元素。

[0186] 如图9所示,数据处理图923包括第一数据集924,第二数据集926和执行集928。执行集928包括两个组件:第一组件930和第二组件932。第一数据集924具有集合输出端口934,其连接到执行集928的执行集入口点936并向其提供数据元素的集合。第二数据集926具有集合输入端口938,集合输入端口938连接到执行集合928的执行集出口点940并从其接收数据元素的集合。

[0187] 在执行集928内,第一组件930具有第一标量输入端口942,并且第二组件932具有第二标量输入端口944。第一标量输入端口942和第二标量输入端口944均连接到执行集入口点936并从该执行集入口点936接收各个数据元素。如上文所述,执行集入口点936复制从集合输出端口934接收的数据元素,以向连接到执行集入口点936的每个标量输入端口提供数据元素集合的每个数据元素的复制本。在图9中,执行集入口点936创建每个数据元素的两个复制本,并将其中一个复制本提供给第一标量输入端口942,将另一个复制本提供给第二标量输入端口944。从图中可以看出,在一些示例中,图形用户界面中的执行集入口点936的可视表示提供了由执行集入口点936创建了数据元素的多少复制本的表示。此外,在其他示例中,表示复制本的不同副本的不同入口点指示符可以被分离,并且围绕执行集的边界分布为与执行集内需要从馈送给该执行集的集合输出端口提供的每个复制的数据元素的副本的组件一样多的组件。

[0188] 第一组件930和第二组件932处理它们各自的数据元素,并且经由标量输出端口946、948将它们各自处理后的数据元素提供给执行集出口点940。在一些示例中,执行集出口点940将处理后的数据元素分组成对,将处理后的数据元素对输出到第二数据集926的集合输入端口938。从图中可以看出,在一些示例中,图形用户界面中的执行集出口点940的可视表示提供了关于由执行集入口点936分组了多少数据元素的复制本的表示。

[0189] 2.6资源锁定

[0190] 在一些示例中,给定执行集中的组件可以在并行实例中运行多次。在一些示例中,并行运行实例的组件可能需要访问共享资源。为了防止竞争情况以及与访问共享资源的多个进程相关的其他问题,可以使用锁定机制(latching mechanism)。通常,锁定机制允许执行集中的组件的一个实例在共享资源上获得时长为该实例完成运行时间的运行时锁定。在实例使共享资源锁定期间,只有实例中的组件才能访问该共享资源,并且其他实例的组件必须等待锁定被解除。实例完成后,它会解除运行时锁定,从而允许其他实例访问共享资源。锁定机制必须在单个执行集内锁定和解锁共享资源(例如,使用上游端处的显式锁定组件和下游端处的显式解锁组件)。在一些实施例中,这样的“锁定的执行集”不能被嵌套,也不能彼此重叠。

[0191] 2.7其他

[0192] 注意,虽然基于全局映射的分配算法是关于二维数据处理图描述的,但是也可以使用该算法来发现一维数据处理图的执行集。

[0193] 通常,执行集可以任意嵌套。

[0194] 通常,执行集具有为执行集的每个实例从链接的输出集合端口接收到的至多一个

驱动数据元素。然而，如果跨越执行集的边界显式地或隐式地复制相同的数据元素，则多个标量输入端口可以接收该相同的数据元素。

[0195] 通常，具有跨越执行集的边界的链路的所有输出标量端口具有来自执行集的多个实例中的每一个的所有数据元素，这些数据元素被聚集到提供给链接的输入采集端口的同一集合中。但是，如果执行集仅具有单个实例，则具有跨越执行集的边界的链路的输出标量端口可以链接到输入标量端口。

[0196] 通常，相同类型的两个端口之间的链路可以遍历执行集边界，假设执行集的遍历不会导致数据处理图中的任何循环。

[0197] 在一些示例中，默认地为每个执行集分配唯一标识符(例如，‘1’)。在其他示例中，每个执行集可以被分配执行集ID路径(例如，‘1/3/6’)。在一些示例中，用户明确地提供执行集ID字符串。执行集ID字符串不一定是唯一的。在执行集ID字符串不是唯一的情况下，执行集ID字符串可以与其父、祖父节点等的执行集ID字符串组合，从而形成唯一的ID字符串。

[0198] 在一些示例中，基于全局映射的分配算法导致组件被分配对应于最深嵌套执行集的ID字符串。在一些示例中，当为执行集被分配了执行集ID路径时，执行集ID路径不一定是唯一的。为了补偿执行集ID路径不唯一的情况，对执行集ID路径施加约束，要求给定执行集上游的执行集ID路径必须是“兼容的”，其中两个执行集ID路径是兼容当且仅当它们是相同的，或者一个是另一个的适当前缀。例如：

[0199] • /1/2/3和/1/2/3兼容

[0200] • /1/2/3和/1/2兼容

[0201] • /1/2和/1/2/3兼容

[0202] • 1/2/3和/1兼容

[0203] • 1/2/3和1/4不兼容

[0204] • /1/2/3和/1/4/5不兼容

[0205] 上述实施例基本上没有对标量块的实例的执行强加排序/并发约束。但是，在一些实施例中，提供其他输入以控制从馈送给执行集的集合接收的数据元素的子集的允许并发性和所需串行化。在一些实施例中，可以对数据元素的一些子集强加根据部分排序的顺序处理。

[0206] 在默认情况下，执行集的实例可以完全并行地运行。然而，在一些情况下，用户可能期望不同的行为。例如，如果正在处理的数据是帐户级数据，则用户可能希望对每个帐户内的数据强制某些限制。例如，用户可能想强制串行执行。在这种情况下，可以允许跨帐户的任何并行度，但是不能同时(即，并发地)处理同一帐户的两个数据元素。可选地，附加限制可以是有序处理，使得相同帐户的两个数据元素不得根据由键限定的顺序或者通过例如接收的顺序来进行无序处理。

[0207] 为了实现这一点，可以为执行集提供串行化键。具有相同串行化键值的所有数据元素必须串行处理，并且在某些情况下必须按照明确定义的顺序处理。运行时系统对具有相同串行化键的数据元素强制串行执行的一种方式是通过串行化键来对执行集实例进行分区：将其驱动数据元素具有特定串行化键(或串行化键的哈希值)的实例分配为在特定计算节点152上执行。在运行时，系统可以通过扫描数据元素的集合来确保工作均匀分布在多个计算节点152上，以确保可运行任务的队列保持充满。在不需要明确定义的顺序(例如在

集合中)的情况下,顺序可以是与它们从输出端口(甚至集合输出端口)产生的顺序相同的顺序,或者与控制串行化键组内的处理顺序的不同的校对键相关联的顺序相同的顺序。在一些情况下,可以通过提供预定义的值作为串行化键来强制执行集完全串行运行。

[0208] 在一些实施例中,可以表面保持顺序被保留,即使没有严格根据该顺序实施处理也是如此。如果执行集的输入和输出两者处的数据与特定顺序(例如,向量内的元素的顺序)相关联,则用户可能希望保留该顺序。即使在数据元素的处理中没有串行化,也可以例如使用处理数据元素时与数据元素一起携带的排序键来对输出数据元素进行排序,以恢复与对应的一组输入数据元素相关联的排序。可替代地,并行产生的输出数据元素可以以与它们进入执行集的顺序相同的顺序合并,而不一定需要实施显式排序操作。

[0209] 与为执行集准备的执行代码相关联的各种计算特性可以由编译器/解释器120来配置,可以有或没有来自用户的输入。例如,上述用于指示如何实施与特定执行集内的组件相对应的任务的嵌入信息可以包括以下任何内容。所述信息可以包括指示任务将完全串行执行(即,没有并行性)的编译器注释。所述信息可以包括指示以与排序约束所允许的并行性一样多的并行性来实施任务的编译器注释。所述信息可以包括指示与相同键值相关的任务被串行执行并且与不同键值相关的任务被并行执行的编译器注释(即,如上所述按键串行化)。

[0210] 编译器注释或修改符可以用于指示各种计算特性中的任何一种:

[0211] • 并发性(例如,如上所述的并行、串行、按键串行)

[0212] • 不同执行集之间的优先性(precedence)(例如,一个执行集的所有任务发生在另一执行集的所有任务之后)

[0213] • 事务性(例如,执行集的任务作为数据库事务(database transaction)处理)

[0214] • 资源锁定(例如,执行集的任务通过特定资源(例如共享变量)实施,锁住,允许任务将资源作为原子单元访问)

[0215] • 排序(例如,保留数据元素之间的排序)

[0216] • 元组大小(例如,将由执行集的每个实例操作的数据元素的数量)

[0217] 编译器/解释器120可以基于自动分析作为整体的执行集或数据处理图的属性,和/或基于接收来自用户的输入(例如,图中的用户注释)确定这些特性。例如,如果在执行集中引用键值,则编译器注释可以指示按键串行化。如果在执行集内使用资源,则编译器修改符可以在该执行集之前/之后允许锁定/解锁该资源。如果在执行集内存在数据库操作,则执行集的每个实例可以被配置为作为数据库事务执行。如果可以在编译时确定可用的核的数量,则编译器注释可以指示每个核将对由数量为等于集合的总大小除以核的数量的数据项组成的数据项元组执行执行集的实例。

[0218] 编译器注释和修改符可以被添加到以目标语言准备的代码,诸如合适的高级语言(例如,DML)或低级可执行代码,或数据处理图的目标中间形式。例如,编译器/解释器120可以将组件插入到明确指示到执行集的入口点或出口点的数据处理图中,或者用来开始/结束事务的组件可以放置在用于处理事务的组件集的入口/出口点,或者组件可以用于锁定/解锁资源。可替代地,编译器/解释器120可以将修改符添加为修改类型的数据流链路。

[0219] 3数据处理图的用户界面

[0220] 在一些示例中,用户界面允许用户通过将组件拖动到画布上并使用链路将组件的

端口连接在一起来开发数据处理图。在一些示例中，用户界面在用户开发数据处理图时重复地将上述分配算法应用于数据处理图。例如，当用户向正在开发的数据处理图中添加组件时，分配算法可以应用于具有被添加组件的图。然后可以将由分配算法发现的结果执行集显示为例如围绕用户界面中的组件绘制的框，或者显示为包围组件的任意形状的区域，其可以通过用于在同一执行集中呈现包含多个组件的区域的唯一的颜色、阴影、纹理或标记来区分。在一些示例中，用户然后可以通过向执行集添加组件或从执行集中去除组件来修改由分配算法发现的执行集。在一些示例中，分配算法验证修改后的执行集是合法的。例如，各种端口之间可以存在组件和链路的一些配置，其可以潜在地以各种合法方式中的任一种划分为多个执行集。在这样的模糊情况下，分配算法可以默认地选择执行集的一个分配，但是用户可能已经期望进行执行集的不同分配，在这种情况下，用户可以修改分配(例如，通过插入出口点以在组件链中更早地关闭执行集)。可替代地，分配算法可以被配置为识别其中可能有多个合法分配的模糊配置，并且提示用户输入以选择一个配置。

[0221] 参考图10A，用户已将三个组件(第一数据集1022，第一计算组件1024和第二数据集1026)拖动到数据处理图开发用户界面的画布1028上。用户还没有使用链路将组件1022、1024、1026的端口连接在一起，并且分配算法尚未在数据处理图中发现任何执行集(除了根执行集)。

[0222] 参考图10B，当用户用链路将组件1022、1024、1026的端口连接在一起时，分配算法自动发现第一执行集1030，第一执行集1030包括第一计算组件1024。通过用户界面将第一执行集1030显示给用户。当用户继续向图中添加组件和链路时，分配算法自动发现执行集，并通过用户界面显示执行集。

[0223] 参考图10C，在一些示例中，用户可能需要打破链路(例如，将另一组件插入到链路中)。在这样的示例中，如果分配算法被允许重新分析数据处理图，则第一执行集1030将被去除，可能导致用户的工作中断和丢失。

[0224] 为了避免这种中断，当用户从数据处理图中去除流或组件时，可以不执行分配算法，而是保持其余组件及它们的执行集关联未受影响。例如，在图10C中，其输入和输出端口断开，第一组件1024仍然包括在第一执行集1030中。在一些示例中，当断开的组件被重新连接时，则允许分配算法自动地发现和显示与重新连接的组件相关联的任何执行集。

[0225] 在一些示例中，如果数据处理图的组件不具有明确的(例如，用户定义的)执行集指定，则允许分配算法发现组件属于哪个执行集。否则，如果组件具有明确的用户定义的执行集指定，则不允许分配算法选择该组件包括在哪个执行集中。例如，如果用户将组件手动移动到给定的执行集中，则不允许分配算法将该组件包括在除用户指定的执行集之外的任何执行集中。也就是说，对数据处理图的任何用户修改不能被分配算法重写(overridden)。

[0226] 在一些示例中，用户界面允许用户使用手势或通过输入设备进行的其他交互来将组件提升到给定执行集中和/或将组件从给定执行集中降级。在一些示例中，用户可以使用菜单选项或其他提示来提升或降级组件。在其他示例中，用户可以简单地将组件拖动到用户界面中的期望执行集中。

[0227] 在一些示例中，用户界面允许用户为数据处理图中的执行集指定一个或多个约束。例如，用户可以将执行约束为在给定时间并行运行不超过N次。

[0228] 在一些示例中，编译器/解释器120接收包括手动定义的执行集和由分配算法发现

的执行集的混合的数据处理图的表示。

[0229] 在一些示例中，用户可以使用界面定义另一类型的执行集，称为启用/禁止执行集。例如，用户可以围绕他们希望被包括在启用/禁止执行集中的一个或多个组件绘制框。启用/禁止执行集包括一个或多个组件并且具有标量输入端口。如果上游组件的标量输出端口向启用/禁止执行集的标量输入端口提供一个数据元素，则允许启用/禁止执行集中的组件执行。如果上游组件的标量输出端口向启用/禁止执行集的标量输入端口提供零数据元素，则启用/禁止执行集中包括的组件被禁止。任何执行集(包括启用/禁止执行集)可以包括控制输入和输出端口，其可用于确定整个执行集是否将被执行以及是否将控制信号传播到其他组件或执行集。如果执行集被并行化(即具有多个实例)，则在执行任何实例之前必须激活输入控制端口，并且在所有实例完成执行之后激活输出控制端口。在一些示例中，通过将端口的可视表示放置在执行集的边界上来提供这些输入和输出控制端口。在其他示例中，通过将这些输入和输出控制端口放置在执行集前方的附加组件上来提供这些输入和输出控制端口。例如，该附加的“对于所有组件”可以(例如，由用户界面自动地或由用户手动地)插入在上游集合输出数据端口和入口点指示符之间，或者代替入口点指示符(即，在上游集合输出数据端口和驱动输入标量数据端口之间)。

[0230] 如上文参照图7注意到的，在一些示例中，用户可以通过沿着数据处理图的流放置执行集入口点和出口点组件来明确地定义执行集入口点和出口点。

[0231] 在一些示例中，用户界面提供实时反馈以在他们的图包括非法操作时通知用户。例如，如果存在由在用户指定的执行集中的组件引起的冲突，则分配算法可以通过用户界面向用户发出警告。为了提供实时反馈，分配算法将验证规则应用于数据处理图以通知用户数据处理图是否合法。参考图11A，非法数据处理图配置1195的一个示例包括两个数据源：将数据元素的第一集合馈送到第一执行集1197中的第一组件1102的标量端口的第一数据源1191，以及将数据元素的第二集合馈送到第二执行集1199中的第二组件1104的标量端口的第二数据源1198。第二执行集1199输出数据元素的第三集合，然后该第三集合被输入到第一执行集1197中的第三组件1106的标量数据端口。由于两个不同的数据元素的集合连接到第一执行集1197中的不同标量端口，所以没有办法知道应该实例化第一执行集1197中的组件的多少并行实例(因为针对存在于第一执行集1197的边界处的每个数据元素生成组件的一个实例)。在一些示例中，通过在例如第二组件1104上显示错误指示符1108来通知用户此冲突。

[0232] 参考图11B，非法数据处理配置1110的另一示例包括将数据元素的集合馈送到第一执行集1116中的第一组件1114的标量输入端口的数据源1112。第一组件1114的标量输出将其输出作为数据的集合提供到第一执行集1116外部的第二组件1118的集合端口。第二组件1118从集合型输出端口将数据元素的集合提供到第一执行集1116中的第三组件1120的标量数据端口。

[0233] 通过从第一执行集1116外的第一组件1114的集合型输出端口传递数据元素的集合，在第二组件1118处处理数据元素的集合，然后将处理后的数据元素的集合传递回第三组件1120的标量端口，定义了“执行集循环”。

[0234] 通常，执行集循环是非法的，因为它们不利于执行排序。例如，通常允许具有进入执行集或离开执行集的附加流，因为对于输入而言，输入数据可以在执行集被执行之前得

到缓存，对于输出而言，输出数据可以在执行集完成执行之后被聚集。但是，如果需要外部组件在执行集之前和之后运行，则这是不可能的。

[0235] 在一些示例中，通过在一个或多个组件上显示错误指示符1108来通知用户执行集循环。

[0236] 在一些示例中，如果每个执行集入口点与至少一个对应的执行集出口点不匹配，则数据处理图被认为是非法的。可替代地，具有入口点但没有对应出口点的执行集可以被允许作为用户定义的执行集，即使其不会被分配算法自动识别。在这些情况下，在(一个或多个)最下游组件完成执行之后，执行集可以结束(而不提供任何输出数据元素)。在一些示例中，如果每个锁定操作不与对应的解锁操作匹配，则数据处理图被认为是非法的。可替代地，如果没有明确指定，则可以推断解锁操作，并且如果推断的解锁操作将需要在与锁定操作不同的执行集中，则仅指示为非法。在一些示例中，如果锁定操作及其对应的解锁操作不存在于同一执行集中，则数据处理图被认为是非法的。

[0237] 4控制图的状态机

[0238] 在为执行准备数据处理图的过程中，编译器/解释器120还在控制图生成过程中生成控制图。在一些实现方式中，生成控制图包括生成用于执行与各个组件相对应的任务的可执行代码和与确定任务之间的数据流和控制流的各种组件间链路相对应的代码。这包括由编译器/解释器120发现的执行集的层级结构中的数据传送和控制传送。

[0239] 生成这样的可执行代码的一部分包括在一些数据结构表示中为每个执行集生成对应的控制图，包括任何启用/禁止执行集。执行集内的任何嵌套执行集被视为表示该嵌套执行集的单个组件，用于生成控制图。此代表性组件的端口对应于嵌套执行集内连接到跨越嵌套执行集边界的链路的组件的端口。编译器/解释器120然后将使用该控制图来生成控制代码。这个生成的控制代码有效地实现了在运行时控制执行的状态机。特别地，一旦执行开始，则该生成的控制代码控制组件或端口何时从该状态机的一个状态转换到另一个状态。

[0240] 图12A示出了编译器/解释器120如何将根执行集的第一组件对1202和第二组件对1204组合成控制图1206的示例。在该示例中，第一组件对1202包括由相应集合数据端口1212、1214连接的第一组件1208和第二组件1210。第二组件对1204包括由相应标量数据端口1220、1222连接的第三组件1216和第四组件1218。

[0241] 编译器/解释器120通过添加开始组件1224和完成组件1226并且按照数据处理图的拓扑的指示将组件连接到开始组件1224和完成组件1226来创建控制图。开始组件和完成组件不执行任何计算任务，但编译器/解释器120将使用开始组件和完成组件来管理控制信号，该控制信号将用于开始某些组件的执行并确定执行集中的所有组件何时已完成执行。

[0242] 为了确定特定组件是否需要连接到开始组件1224，编译器/解释器120检查至该组件的输入，以基于到上游串行端口的现有链路确定该组件是否没有被指定开始执行，如上所述，上游串行端口包括控制端口和标量端口。

[0243] 例如，如果组件没有至其控制输入端口的链路，则它可能永远不会开始执行，因为将永远不会有控制信号来指示它开始。另一方面，即使没有控制输入，也可能取决于组件具有的数据输入的类型，针对数据的到达而触发该组件的执行。例如，如果组件具有标量输入端口，则即使在其控制输入端口处没有控制信号，一旦该组件在其标量输入端口处看到数

据，该组件仍将开始执行。另一方面，如果组件只有集合数据输入，那么这种情况不会发生。如果这样的组件不具有用来触发执行的控制输入或标量数据输入，则它将需要至开始组件1224的连接。

[0244] 在图12A的上下文中，第一组件1208既不具有控制输入也不具有标量数据输入。因此，第一组件1208将无法自己开始执行。因此，第一组件1208必须链接到开始组件1224。第三组件1216同样既不具有控制输入也不具有标量数据输入。因此，第三组件1216也必须链接到开始组件1224。

[0245] 第四组件1218没有控制输入。但是它被连接以从第三组件1216接收标量数据输入。因此，它将在通过其输入标量端口1222接收到数据时开始执行。因此，第四组件1218不需要连接到开始组件1224。

[0246] 第二组件1210被配置为从第一组件1208接收数据。然而，在输入集合端口1214而不是在输入标量端口处接收该数据。结果，类似于第一组件，第二组件1210也必须连接到开始组件1224。

[0247] 编译器/解释器120还需要识别哪些组件将需要连接到完成组件1226。

[0248] 通常，当组件缺少控制输出链路或(任何类型的)数据输出链路时，将组件连接到完成组件1226。在图12A的左侧的图中，仅第二组件1210和第四组件1218满足该条件。因此，如图12A的右侧所示，仅这两个组件连接到完成组件1226。

[0249] 图12B类似于图12A，不同的是在图的左侧的第一组件1208和第三组件1216之间存在控制链路。与规则一致，不再需要在得到的替代控制图1206'中将第三组件1216连接到开始组件1224。

[0250] 控制图有效地定义分布式状态机，其中组件及其串行端口响应于上游组件和串行端口发生的转换从一个状态转换到另一个状态。通常，上游组件将从一个状态转换到另一个状态，导致其输出串行端口转换，这使得下游组件的链接串行输入端口转换，这使得这些下游组件转换，等等。下面参照用于组件及其串行端口的状态转换图更详细地描述用于实现该行为的特定类型的状态机的一个示例。

[0251] 为了提供对状态机的转换的控制，编译器/解释器120移植额外的控制代码到用于执行由特定组件表示的任务的代码中。如本文所使用的，“移植(graft)”意指前附、后附或既前附又后附的控制代码。前附的控制代码在本文被称为“前序(prologue)”代码，而后附的控制代码被称为“结束(epilogue)”代码。组件的前序代码在组件执行其任务之前执行。组件的结束代码在组件610A已经完成执行其任务之后执行。

[0252] 移植的控制代码检查所存储的状态信息，例如累加器的值(例如，计数器倒计数到指示输入已为调用组件准备就绪的值)或标志的状态(例如，设置为指示组件已被禁止的值的标志)，以确定是否使一个或多个下游组件执行其相应的任务。

[0253] 在一个实施例中，前序代码监视上游输出串行端口的状态，并更新组件的输入串行端口的状态和组件的状态，而结束代码在组件完成实施其任务之后更新组件的输出串行端口。

[0254] 在另一个实施例中，代替监视上游输出串行端口的下游组件的前序代码，上游组件的结束代码更新下游输入串行端口的集合状态，并监视该集合状态以触发下游组件的前序代码在适当的时间(例如当初始化为输入串口数量的计数器达到零时)执行。可替代地，

代替计数器从输入端口的数量倒计数(或向上计数到输入端口的数量),可以使用另一形式的累加器来存储用于触发组件的状态信息,例如存储表示不同组件的不同端口的状态的位的位图。

[0255] 作为该移植的控制代码的结果,基于在特定组件的执行开始和结束时一个或多个上游逻辑状态的集合的发生,任务的完成自动导致以与由控制图表示的数据控制依赖性一致的方式、并且以允许多个组件的并发操作并且允许使用条件控制逻辑来控制的方式自动执行其他任务。

[0256] 图13A和图13B示出了可用于组件(图13A的状态转换图1300)和其串行端口(图13B的状态转换图1310)的示例性状态机的状态转换图。这两个状态转换图是类似的,不同之处在于,由于活动状态1304与正在进行的执行相关联,并且由于只有组件而不是端口实施执行,所以只有组件可以处于活动状态1304。

[0257] 将描述两个状态转换图的所有可能状态,以及遵循状态之间的每个转换所需的条件,需要参考图13A和图13B。在状态转换图的此描述中涉及的所有输入和输出端口均是串行端口,因为控制图中的组件仅需要链接串行端口(而不是集合端口)。控制图中的特定组件可以处于状态转换图1300中的四个逻辑状态之一。第一状态是未决(pending)状态1302。这是当与控制图相关联的执行集开始执行时,组件开始的状态。如果组件的任何输入端口处于未决状态1312,则组件保持未决状态1302。如果组件恰好没有输入端口,则它在未决状态1302中开始,但是立即有资格从未决状态1302转换出。

[0258] 从未决状态1302,组件可以转换到活动状态1304或禁止状态1306。

[0259] 如果组件没有输入端口处于未决状态1312并且并非其所有输入端口都处于禁止状态1316(即,至少一个输入端口处于完成状态1314),则组件转换到活动状态1304。端口默认为“必需”,但可标记为“可选”。可选端口可以保持未连接到另一个端口,而不会导致错误(虽然可能有警告)。未连接的任何可选端口自动地处于完成状态1314。只要组件仍在执行其任务,该组件就保持在活动状态1304。当组件处于活动状态1304时,其多个输出端口可以在不同时间转换或一起从未决状态1312转换到完成状态1314或禁止状态1316。在完成其任务的执行时,组件转换从活动状态1304转换到完成状态1308。

[0260] 如果组件的任务已经完成执行,并且其所有输出端口被“决断(resolved)”,即不再未决,则组件转换到完成状态1308。

[0261] 如果由于定制控制逻辑,或由于其所有输入端口被禁止,或由于禁止了其所需输入端口中的至少一个,或由于组件中的未处理错误而导致该组件的前序已触发至禁止状态1306的转换,则该组件处于禁止状态1306。组件的所有输出端口也决断到禁止状态1316以向下游传播这种禁止。

[0262] 对于端口,状态转换规则取决于端口是输入端口还是输出端口。

[0263] 端口的初始状态是未决状态1312。输入端口通常跟随它所链接到的上游输出端口的状态。因此,当上游输出端口转换时,在控制图中链接到该输出端口的输入端口转换到相同的状态。输出端口保持未决,直到组件在其活动状态期间确定输出端口应决断到什么状态。

[0264] 如上所述,输入端口跟随它们所链接的上游输出端口。因此,对于链接到单个上游输出端口的输入端口,当其所链接的上游输出端口转换到完成状态1314时,该输入端口转

换到完成状态1314。如果输入端口通过多个链路链接到多个上游输出端口，则输入端口在其上游输出端口中的至少一个转换到完成状态1314之后转换到完成状态1314。否则，如果所有上游输出端口转换到禁止状态1316，则输入端口转换到禁止状态1316。一些实施例使用与该默认“或逻辑”不同的其他逻辑来确定是将输入端口转换到完成状态1314还是禁止状态1316(例如，“与逻辑”，其中仅当所有上游输出端口处于完成状态1314时输入端口才转换到完成状态1314)。如果组件的输入数据端口决断为完成状态1314，则数据元素准备就绪供该组件处理。如果组件的输出数据端口决断为完成状态1314，则数据元素准备就绪从该组件向下游发送。

[0265] 与输入端口跟随它们所链接到的上游输出端口的状态的规则一致，当与其链接的上游输出端口决断为禁止状态1316时，输入端口决断为禁止状态1316。输出端口决断为禁止状态1316，或者因为活动的组件计算出的结果确定输出端口应当被禁止，或者为了使得禁止从上游被禁止组件向下游传播，或者如果组件中存在未处理的错误。在一些实施例中，编译器可以通过禁止根在被禁止组件处的下游组件的树来优化执行，而不必一个组件接一个组件地向下游传播禁止。

[0266] 在其他实施例中，可以使用各种替代状态机，其中集合端口之间的链路也可以包括在控制图中。在一些这样的实施例中，集合端口的状态转换图可以包括除未决状态、完成状态和禁止状态之外的活动状态，诸如在组件的状态转移图1300中。当集合端口(作为输出端口)产生数据或(作为输入端口)消耗数据时，集合端口处于活动状态。对于输入集合端口，例如，一旦确定不是所有输入端口都将被禁止，就可以在上游产生第一数据元素时触发活动状态。在一些实施例中，没有集合端口的禁止状态。包括集合端口的状态转换的控制图中的组件所遵循的转换规则可以以与为输入标量端口或控制端口处理完成状态相同的方式来处理输入集合端口的活动状态。

[0267] 5计算平台

[0268] 回看图1，数据处理图的组件的实例在执行数据处理图的情境中被派生(spawn)为任务，并且通常在计算平台150的多个计算节点152中执行。如下面更详细讨论的，控制器140提供这些任务的调度和执行轨迹的监督控制方面，以便实现例如与计算负荷的分配，通信或输入/输出开销的减少以及存储器资源的使用相关的系统的性能目标。

[0269] 通常，在由编译器/解释器120翻译之后，整个计算被表达为可由计算平台150执行的目标语言的过程的基于任务的规范130。这些过程利用原语，诸如“派生(spawn)”和“等待”，并且可以在过程中包括原语，或调用由程序员为高级(例如，基于图的)程序规范110中的组件指定的工作过程。

[0270] 在许多情况下，组件的每个实例被实现为任务，其中一些任务实现单个组件的单个实例，一些任务实现执行集的多个组件的单个实例，以及一些任务实现组件的连续实例。来自组件及其实例的特定映射取决于编译器/解释器的特定设计，使得所得到的执行保持与计算的语义定义一致。

[0271] 通常，运行时环境中的多个任务被分层级排列，例如，一个顶级任务派生多个任务，例如，一个任务用于数据处理图的每个顶级组件。类似地，执行集的计算可以具有用于处理整个集合的一个任务，其中多个(即许多)子任务分别用于处理集合的元素。

[0272] 在运行时环境中，已经派生的每个任务可以处于一组可能状态之一。在第一次派

生时，任务在被初始执行之前处于派生状态。在执行时，它处于执行状态。任务时而可能处于暂停状态。例如，在某些实现方式中，调度器可以在任务已经超过处理器利用的量、正在等待资源等时将任务置于暂停状态。在一些实现方式中，任务的执行不被抢占，并且任务必须放弃控制。有三个暂停子状态：可运行，被阻止和已做完。例如，如果任务在完成其计算之前放弃控制，则任务是可运行的。当例如在父任务检索到任务的返回值之前该任务完成其处理时，该任务是已做完的。如果任务正在等待该任务外部的事件，例如另一个任务的完成(例如，因为它已经使用了“等待”原语)或数据记录的可用性(例如，阻止in.read()或out.write()函数的一次执行)，则任务是被阻止的。

[0273] 再次参考图1，每个计算节点152具有一个或多个处理引擎154。在至少一些实现方式中，每个进程引擎与在计算节点150上执行的单个操作系统进程相关联。取决于计算节点的特性，在单个计算节点上执行多个处理引擎可能是有效的。例如，计算节点可以是具有多个单独处理器的服务器计算机，或者服务器计算机可以具有带多个处理器内核的单个处理器，或者可以是具有多个内核的多个处理器的组合。在任何情况下，执行多个处理引擎都可能比在计算节点152上仅使用单个处理引擎更有效。

[0274] 处理引擎的一个示例托管在虚拟机的情境中。一种类型的虚拟机是Java虚拟机(JVM)，其提供在其中可以执行以Java Bytecode(Java字节码)的编译形式指定的任务的环境。但是也可以使用其他形式的处理引擎，其可以使用或可以不使用虚拟机架构。

[0275] 参考图14，计算节点152的每个处理引擎154具有一个或多个运行器1450。每个运行器1450使用一个或多个进程或进程线程来执行可运行任务。在一些实现方式中，每个运行器具有相关联的进程线程，但是运行器与线程的这种关联不是必需的。在任何时候，每个运行器正在执行计算的最多一个可运行任务。每个运行器具有单独的可运行队列1466。计算的每个可运行任务在系统的运行器1450的一个可运行队列1466中。每个运行器1450具有调度器/解释器1460，其监视当前运行的任务，并且当该任务将状态改变为已做完，被阻止或暂停时，从运行队列1466中选择另一个任务并执行它。任务与运行器相关联，并且不可运行的运行器的任务被保持在可运行队列1466之外，例如，如图中所示，在被阻止和已做完队列1468中。

[0276] 例如，当初始化处理引擎154时，可以创建运行器1450，为每个引擎创建预配置数量的运行器。如下所述，在一些实现方式中，可以向处理引擎添加或去除运行器，并且甚至在数据处理图的执行期间，可以从计算平台150添加和去除处理引擎本身。然而，对于下面的初始描述，我们假设处理引擎的数量以及每个处理引擎内运行器的数量保持恒定。

[0277] 作为示例，对数据处理图的处理开始于在顶层任务中执行主过程。例如，基于任务的控制器140指示与处理引擎1450之一的监视器1452通信的计算节点中的一个开始执行主过程。在该示例中，监视器1452将用于执行主过程的任务放置在处理引擎之一的可运行队列1466中。在该示例中，运行器是空闲的(即，此时没有其他任务在运行，并且在可运行队列中没有其他可运行任务)，所以该运行器的调度器/解释器1460从可运行队列中取出该任务并且开始执行任务。当以需要解释的语言表达过程时，调度器/解释器1460解释过程的连续语句。

[0278] 在该示例中，主过程的第一语句为支持无序集合的流的链路创建链路缓存器1470(即，分配存储器)，其在该示例中包括无序无界缓存器，即缓存器1，缓存器2和缓存器3。可

以使用各种方法创建这种类型的组件间链路，并且管理用于这些链路的相关联的计算资源(包括链路缓存器1470)，其包括其上游端口是集合端口的任何链路。在一些示例中，链路缓存器1470包括用于表示集合的源的输出集合端口的缓存器和用于表示集合的目标的输入集合端口的单独缓存器。这些缓存器可以在开始处理集合之前在运行时分配，并且在对集合的处理结束之后被解除分配(即，释放用于缓存器的内存)。在该示例中，这些链路缓存器1470被分配在其中任务的运行器正在执行的处理引擎154的存储器中。通常，其中创建缓存器的存储器在半导体随机存取存储器(RAM)中，尽管在一些实现方式中，诸如磁盘之类的其他存储设备可以用于存储至少一些缓存数据。注意，在其他方法中，缓存器对运行器本身而言可以是本地的。在实践中，如果处理引擎154被实现为操作系统进程，则缓存器被创建为该进程的地址空间中的存储器区域。因此，直接基于硬件地址对缓存器的访问限于在该进程内执行的指令。注意，在这种方法中，如果多个运行器将能够读取或写入缓存器，则可能对缓存器需要至少进行一些同步和访问控制，例如使用锁(lock)或信号量(semaphore)。在其中每个运行器在操作系统进程内被实现为单个线程的方法中，缓存器可以与特定运行器相关联，并且所有访问可以被限制到该运行器，从而避免来自多个线程的潜在争用。在下面的讨论中，我们假设可以从处理引擎中的任何运行器访问缓存器，并且实现适当的访问控制以允许这样的共享访问。

[0279] 主进程的后续步骤涉及派生或主过程调用的forall(用于所有)原语。通常，至少在默认情况下，任务或子任务的派生使得这些任务最初在与父进程相同的运行器中形成。例如，派生的Work_Read_External_Data(工作读取外部数据)任务在同一个运行器上派生。在任务正在访问外部数据的程度上，任务可以利用至该外部数据的I/O接口1464。例如，该接口可以包括到外部数据库、网络数据连接的端点等的开放连接。这样的I/O接口可以绑定到特定的运行器，因此使用该接口的任务可能需要访问仅来自该运行器的接口，如下面在运行器之间任务的潜在迁移的情境中进一步讨论的。在该示例中，我们假设任务以合理计量的方式填充缓存器1，并且不会例如通过使缓存器1增长超过处理引擎的容量而“淹没(overwhelm)”系统。下面还讨论对控制方面的方法，例如，以避免资源的拥塞或耗尽。

[0280] 与Work_Read_External_Data任务的执行并发地，forall_Work A导致针对从缓存器1读取的每个记录派生任务。特别地，“forall”原语引起由要执行原语的自变量标识的任务的多个实例，其中实例的数量通常由在运行时接收的数据元素的数量确定，并且其中它们被执行的位置和它们被调用的顺序可以不受编译器限制而用于在之后的运行时确定。如上所述，在默认情况下，这些任务也在相同的运行器1450上创建，并且在没有其他控制的情况下，与从缓存器1可得到的数据一样快地派生这些任务。Work_B和Work_Read_External_Data的任务类似地在相同的运行器上创建。

[0281] 注意，基于任务的规范使用“forall”原语，而没有明确地指定运行时控制器将如何实现任务的分布，以导致所有数据待处理。如上所述，运行时控制器可以使用的一种方法是在同一计算节点上派生单独的任务，然后依赖于迁移特征以使得任务在分开的节点上执行，从而平衡负载。可以使用其他方法，其中“forall”原语导致在多个节点上直接执行多个任务。在游标(cursor)定义内存数据库的表的行的基于索引的子集的情况下，游标“forall”原语的实现可以使得游标被分割成多个部分，每个部分与存储在不同节点上的记录相关联，并且为不同节点上的游标的单独部分派生任务，从而导致处理和数据存储的局

部性。但是应当理解，可以在运行时控制器和分布式计算平台的一个或多个实施例中实现多种方法来执行在作为编译器120输出的基于任务的规范130中使用的“forall”原语。在一些示例中，方法的选择可以取决于例如基于记录的数量，计算节点上的数据分布，节点上的负载等的运行时决策。在任何情况下，用于实现“forall”原语的方法不一定对于数据处理图的开发人员或编译器的设计者是已知的。

[0282] 系统的特征在于任务可以在创建运行器之后在运行器之间传送。非常一般地，这种任务传送的一种方式是通过“挪用”或“拉动”机制来实现的，其中空闲或至少轻微加载的运行器使得来自另一运行器的任务被传送给它。虽然可以使用各种标准，但是运行器可运行队列1466中的多个可运行任务可以基于本地标准(诸如在其可运行队列中是否有少于阈值数量的任务)来确定该运行器是否应当寻求从其他运行器挪用的任务。在一些实现方式中，更全局性的决策进程可用于在多个运行器上重新平衡任务队列，但总体效果是类似的。

[0283] 在至少一些实施例中，将任务从一个运行器挪用到另一个运行器不是必然涉及转移该任务的所有数据。例如，只有当前执行“框架”中可访问的数据(例如，用于从当前程序范围可访问的局部和全局变量的数据，例如当前子例程调用)与引用一起打包回任务“主(home)”运行器。此数据足以在迁移的目标运行器处创建任务的可运行副本，并且目标可运行队列中的条目准备好在在该运行器中执行。

[0284] 当迁移的运行器完成执行时，或者通过从局部变量可用的程序范围返回该迁移的运行器耗尽了传送到该运行器的数据时，任务被传送回主运行器，其中用于任务的数据被合并，并且任务再次在其主运行器处可运行。

[0285] 注意，在单个处理引擎内传送任务期间，运行器之间的通信可以通过本地存储器(即，避免磁盘或网络通信)，从而消耗相对少的资源。在允许处理引擎之间进行挪用和迁移的实现方式中，在从一个运行器转换到另一个运行器时，任务消耗相对较少的资源，例如，主要消耗处理引擎之间的通信资源而不是计算资源。此外，这种通信的等待时间相对不显著，因为主运行器和目标运行器被假定为在传送期间忙于计算，主运行器是因为其可运行队列被大量填充并且因此不可能空，而目标运行器是因为挪用是在预期在目标处的可运行队列被清空的情况下进行的。

[0286] 在与图2A-图2B中所示的计算相关联的任务的执行的示例中，任务挪用机制将用于计算的负载分布在一个或多个处理引擎的运行器上。然而，注意，某些数据访问被限于特定运行器(或可能限于特定处理引擎)。例如，如上文所述，缓存器2的数据可以由单个运行器(或可能一组运行器)访问，然而可能需要写入缓存器2的Work_A任务可能已经被无法写入缓存器2的运行器挪用。在这样的情况下，当任务需要采取必须在与当前正在执行的任务所在的运行器不同的运行器上执行的动作时，以“迁移”或“推动”方式将任务迁移到合适的运行器。

[0287] 在至少一些示例中，计算平台150支持一组全局变量对(键，值)的全局数据存储。该数据存储可以分布在多个计算节点(或处理引擎)上的存储器(例如，RAM或磁盘)上。键的名称空间是全局的，因为键的规范在所有计算节点152及其运行器1450处具有相同的含义。这些变量的值在任务被实例化、执行和终止时持续，从而提供了一种在任务之间传递信息的方式，而不需要经由共同的父任务将这种信息从一个任务传递到另一个任务。如下所述，根据键对进行的访问被控制，使得值的使用和更新不会导致任务之间的冲突。在一些示例

中，任务对于它们的一些或全部执行而获得对特定(键，值)对的独占访问。

[0288] 通常，(键，值)对的存储是分布的，并且任何特定(键，值)对与特定计算节点152相关联。例如，(键，值)对存储在该计算节点处的分布式表存储1480中。在一些实现方式中，派生原语允许指定键和从相关联的变量到任务的局部变量的映射。当指定一个键时，派生的任务在其执行的持续时间内获得对键的独占访问。在执行开始之前，将值从存储传递到任务的本地上下文中，并且在执行完成后，本地上下文中的值将被传递回全局存储。如果一个派生原语指定另一个正在执行的任务使用的键，则这个新派生的任务将被阻止，直到它可以获得对该键的独占访问。在一些实现方式中，每个计算节点可以确定特定键的主节点，并且当请求派生任务时，该请求由(键，值)对驻留的计算节点处理，并且任务的执行将最初在该节点开始。在替代实施例中，用于获得对这种全局共享(键，值)对的类似的独占访问的其他方法不是必然涉及在与存储相同的位置发起任务，例如通过传送独占访问的请求并且随后使用更新的键值来传送独占访问的释放。任务可以创建新的(键，值)对，在默认情况下，在创建新(键，值)对时新的(键，值)对存储在任务运行的节点上。

[0289] 全局状态变量的一种用途是在执行集合的连续记录的函数期间进行聚集。例如，全局存储维护分配给键的值的窗口，而不是作为单项的值。因此，在编程模型中，可以将值添加到与键相关联地维护的历史中，并且可以提供先前添加的值的函数。值的窗口可以根据项目的数量(即，最后的100项)定义，通过时间窗口定义(即，例如在最后10分钟中添加的项目，由值被添加时的时间定义或由每个值被添加时提供的显式时间戳定义)。注意，编程模型不需要明确删除落在窗口外的旧值，窗口的定义允许实现自动执行这样的删除。编程模型包括用于创建这种基于窗口的键控全局变量的多个原语(例如，定义窗口的性质和范围)，将值添加到键以及值的窗口的计算函数(例如，最大值，平均值，不同值的数量，等等)。一些原语将针对键的新值的添加和窗口函数的返回(例如，将新值添加到键并返回添加的最后100个值的平均值)组合。

[0290] 在至少一些示例中，全局存储还包括经由称为句柄的标识符访问的面向记录的共享数据。例如，句柄可以标识数据记录的源或宿，或者作为另一示例，句柄可以标识数据集中的特定记录。通常，句柄被键入，因为句柄点提供了访问数据的方式，并且还提供被访问的数据结构的定义。例如，句柄可以具有与其相关联的数据记录的字段(列)结构。

[0291] 在至少一些示例中，全局存储器(例如，在计算节点的存储器中)包括针对多行类型化数据的一个或多个表的表存储，其中该表格或表格的特定记录经由称为句柄的标识符被访问。表的行类型可以是具有向量、记录向量等的层级记录类型。在一些示例中，表可以具有提供对行的哈希或B树(有序)访问的一个或多个索引，并且游标可以基于表、索引或索引和键值创建。行可以单独插入、更新或删除。为了支持事务处理，任务可以锁定一个或多个表的一行或多行，例如，用于在对数据处理图的组件的处理期间读取或更新访问。表可以被视为用于数据并行操作的集合，例如，作为数据处理图中的数据的源或目标。通常，对表进行索引，并且可以基于产生游标的索引来选择表的行的子集，然后使用该游标来提供所选择的行作为数据源。在一些示例中，另外的原语对任务可用，用于诸如分割游标和估计与句柄相关联的记录的数量的动作。当提供游标作为用于执行集的数据源时，游标可以被分割成多个部分，每个部分将表的一些行提供给执行集的对应实例，从而提供并行性并且适当地分割游标使得能够在存储行的节点上执行。数据表还可以由实现事务的任务访问，使

得维护数据表的修改，以便在任务之外不可见，直到由任务明确地提交这些修改。在一些示例中，可以通过锁定表的一个或多个行来实现这样的事务支持，而在其他示例中，可以实现涉及行的多个版本的更复杂的方法，以比可以仅使用锁定提供更高的潜在并发性。

[0292] 文件、数据流和内存表都是被称为集合的示例。读取器任务从集合中读取记录，并且写入器任务将记录写入集合。一些任务既是读取器又是写入器。

[0293] 如上所述，表示集合的流可以使用内存缓存器在运行时系统中实现。可替代地，在各种实现方式中可以使用任何形式的存储，包括数据库内的表或分布式存储系统。在一些实现方式中，使用内存中分布式数据库。在一些实现方式中，编译器以不是必然暴露给数据处理图的开发人员的方式使用内存表来实现这样的流。例如，编译器可以使上游组件填充表的多个行，并且下游组件读取先前填充的行，从而实现无序数据流。运行时间控制器可以调用与执行集相对应的任务的多个实例，从而通过以数据元素被接收到存储中的顺序不同的顺序从存储中取出这些数据元素，对来自上游集合端口的驱动数据元素进行处理，防止某些形式的阻塞。例如，可以调用任务的实例而不阻止任何特定的其他实例调用任何实例(即，直到任何特定的其他实例完成处理一个或多个数据元素之后)。

[0294] 通常，集合中的记录可以在该记录中的数据被首次写入之前具有句柄。例如，可以将表设置为索引的一组记录的目标，并且即使在写入那些记录的数据之前，各个记录也可以具有句柄。

[0295] 6实现方式

[0296] 上述方法可以例如使用执行合适的软件指令的可编程计算系统来实现，或者可以在诸如现场可编程门阵列(FPGA)或一些混合形式的合适的硬件中实现。例如，在编程方法中，软件可以包括在一个或多个编程或可编程计算系统(其可以是诸如分布式、客户端/服务器或网格的各种架构)上执行的一个或多个计算机程序中的过程，每个计算系统包括至少一个处理器，至少一个数据存储系统(包括易失性和/或非易失性存储器和/或存储元件)，至少一个用户界面(用于使用至少一个输入设备或端口接收输入，并且用于使用至少一个输出设备或端口提供输出)。软件可以包括例如提供与数据处理图的设计、配置和执行相关的服务的更大程序的一个或多个模块。程序的模块(例如，数据处理图的组件)可以被实现为符合存储在数据仓库中的数据模型的数据结构或其他有组织的数据。

[0297] 软件可以使用持续一段时间(例如，动态存储器装置(例如动态RAM)的刷新周期之间的时间)的介质的物理特性(例如，表面凹坑和平台、磁畴或电荷等)以非暂时性形式存储，例如被实施在易失性或非易失性存储介质或任何其它非暂时性介质中。在准备加载指令时，软件可以提供在有形、非暂时性介质上，例如CD-ROM或其他计算机可读介质(例如，可由通用或专用计算系统或设备读取)，或者可以通过网络的通信介质被递送(例如，被编码成传播信号)到其被执行的计算系统的有形、非暂时性介质。可以在专用计算机上或使用诸如协处理器或现场可编程门阵列(FPGA)或特定的专用集成电路(ASIC)的专用硬件来执行处理中的一些或全部。处理可以以分布式方式实现，其中由软件指定的计算的不同部分由不同的计算元件执行。每个这样的计算机程序优选地存储在或下载到可由通用或专用可编程计算机访问的存储设备的计算机可读存储介质(例如，固态存储器或介质，或磁介质或光介质)上，用于当计算机读取存储设备介质以执行本文所述的处理时，配置和操作计算机。本发明的系统还可以被认为可实现为配置有计算机程序的有形的、非暂时性介质，其中如

此配置的介质使得计算机以特定和预定义的方式操作以执行本文描述的一个或多个处理步骤。

[0298] 已经描述了本发明的多个实施例。然而，应当理解，前述描述旨在说明而不是限制本发明的范围，本发明的范围由所附权利要求的范围限定。因此，其他实施例也在所附权利要求的范围内。例如，在不脱离本发明的范围的情况下可以进行各种修改。另外，上述的一些步骤可以是与顺序无关的，并且因此可以以与所描述的顺序不同的顺序来执行。

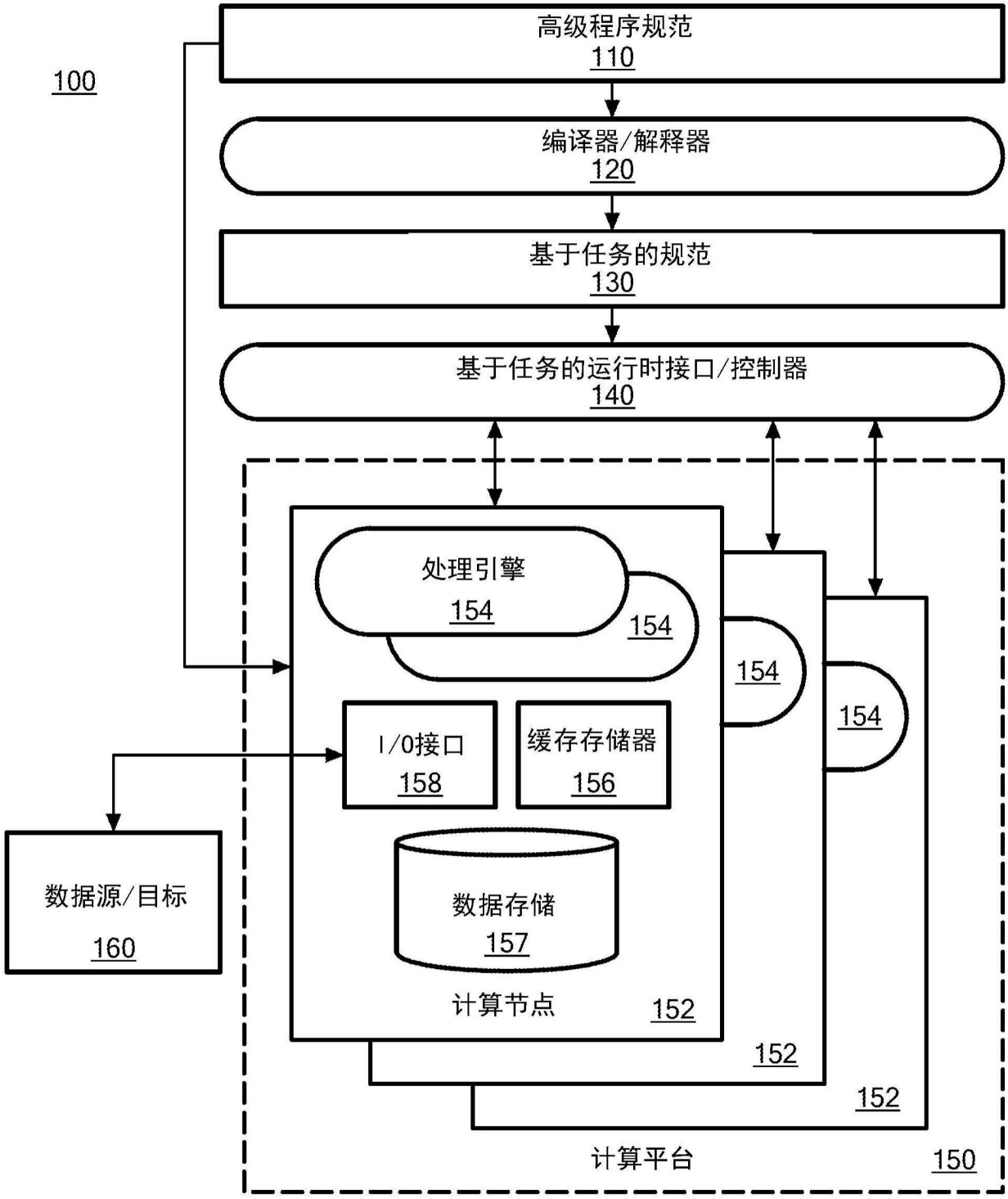

100
高级程序规范
110
编译器/解释器
120
基于任务的规范
130
基于任务的运行时接口/控制器
140
处理引擎
154
154
154
154
I/O接口
158
缓存存储器
156
数据存储
157
数据源/目标
160
计算节点
152
152
152
计算平台
150

图1

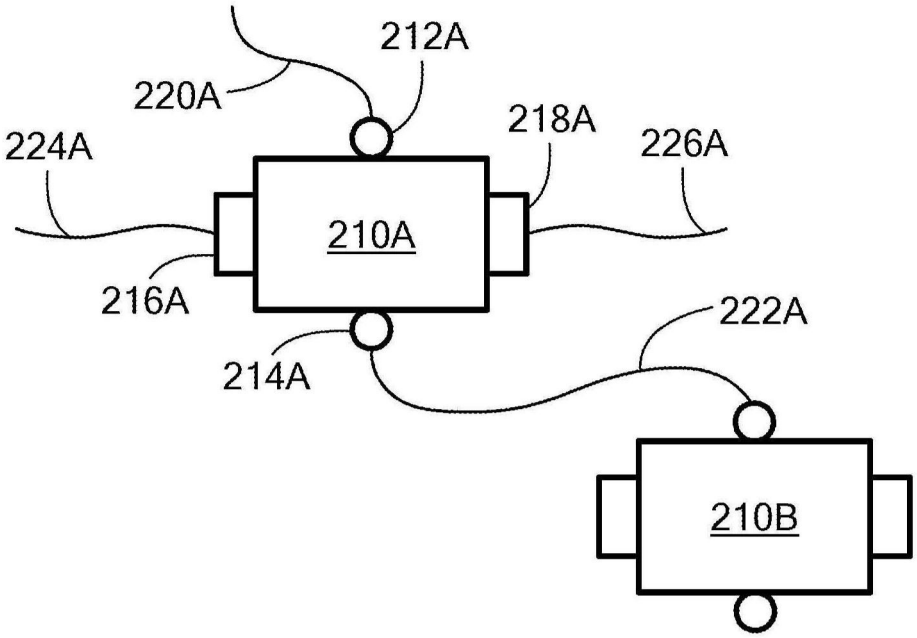


图2A

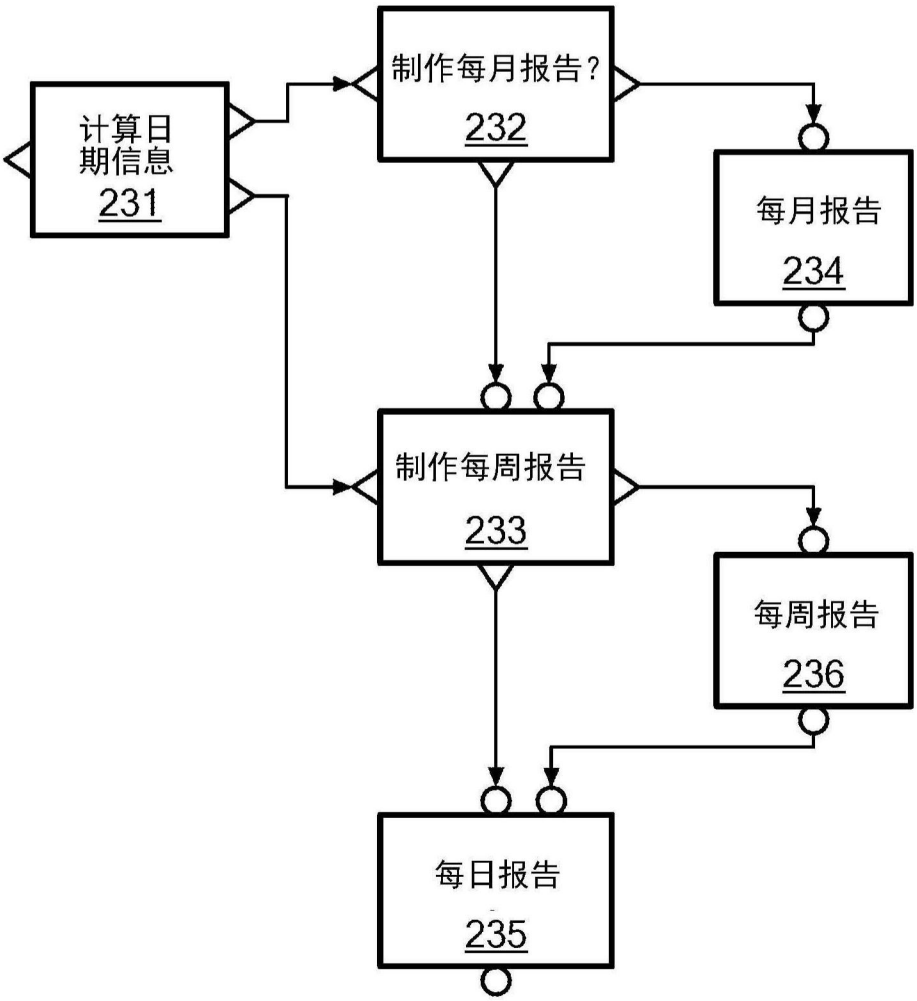


图2B

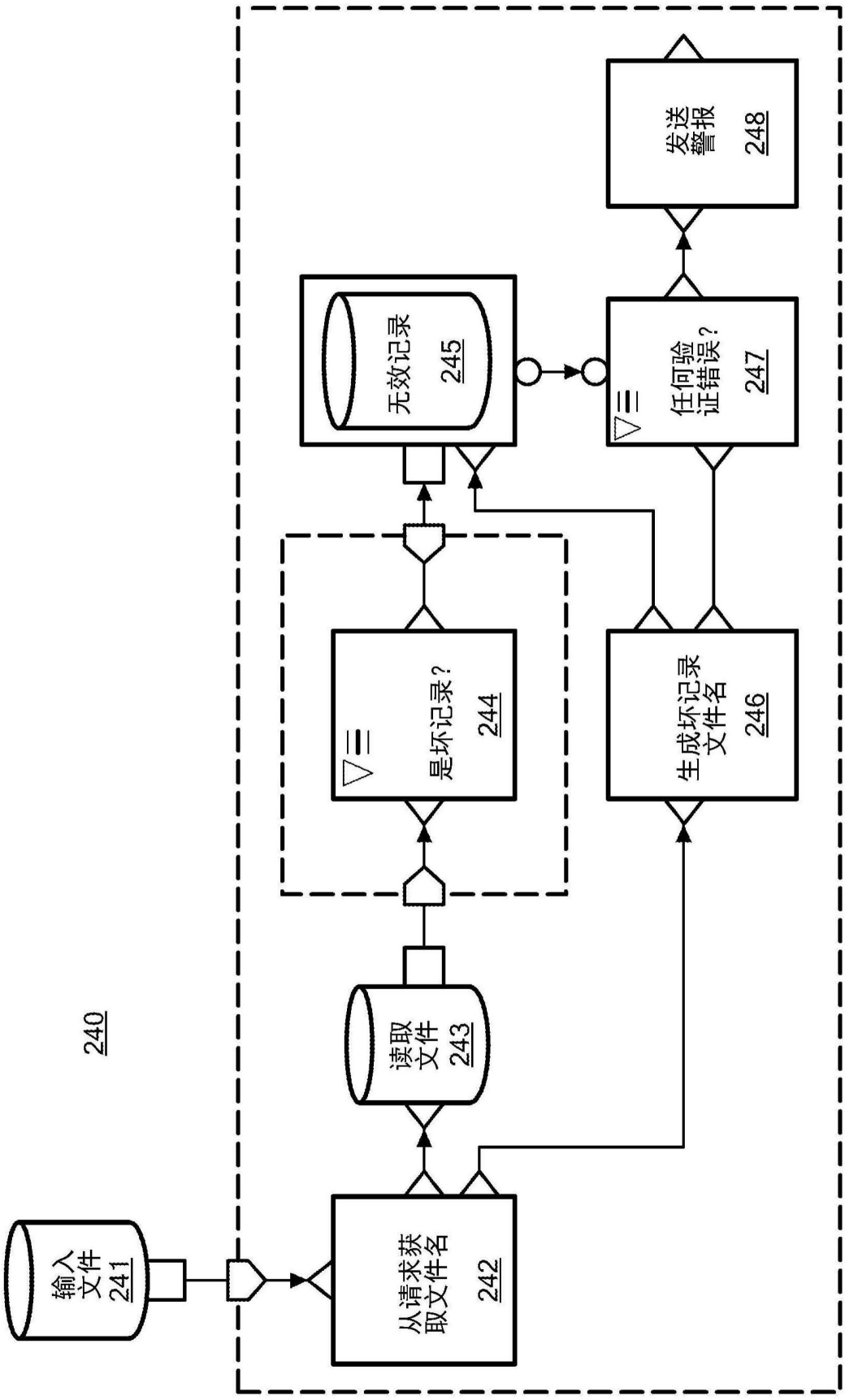

240
输入文件 241
从请求获取文件名 242
读取文件 243
是坏记录? 244
无效记录 245
生成坏记录文件名 246
任何验证错误? 247
发送警报 248

图2C

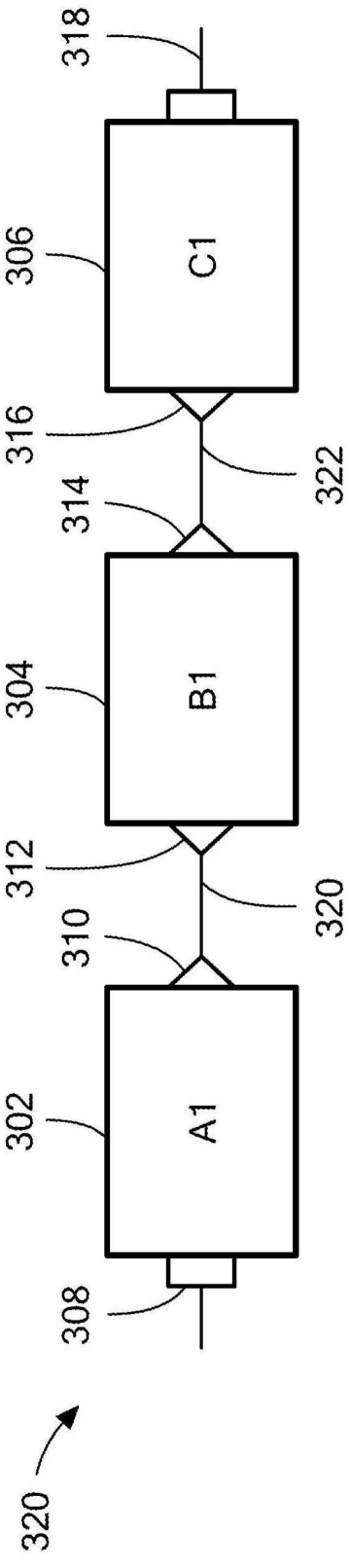


图3A

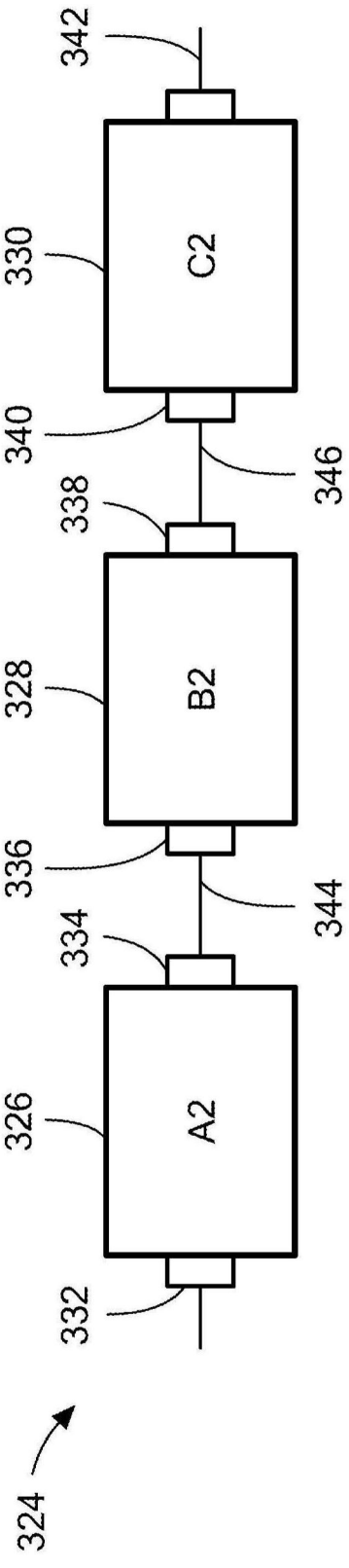


图3B

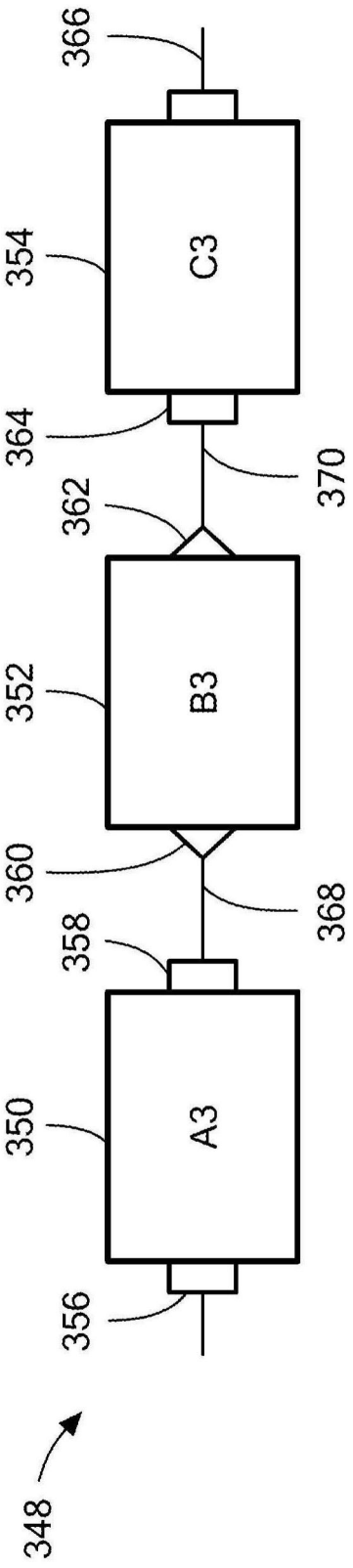


图3C

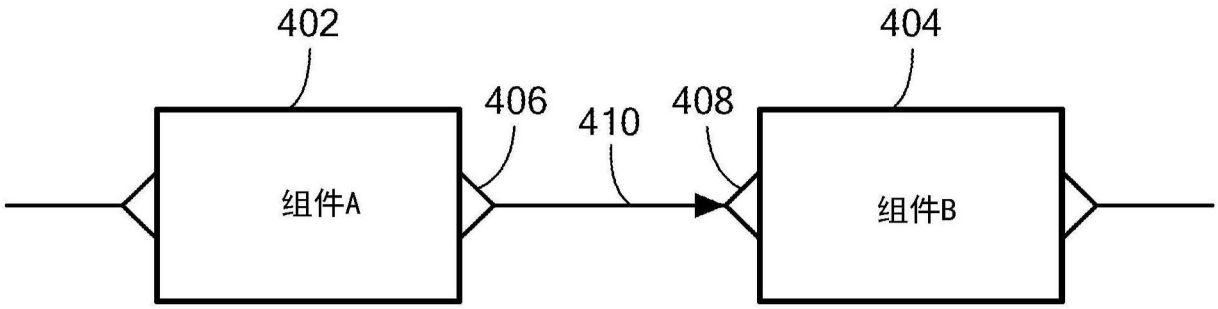


图4A

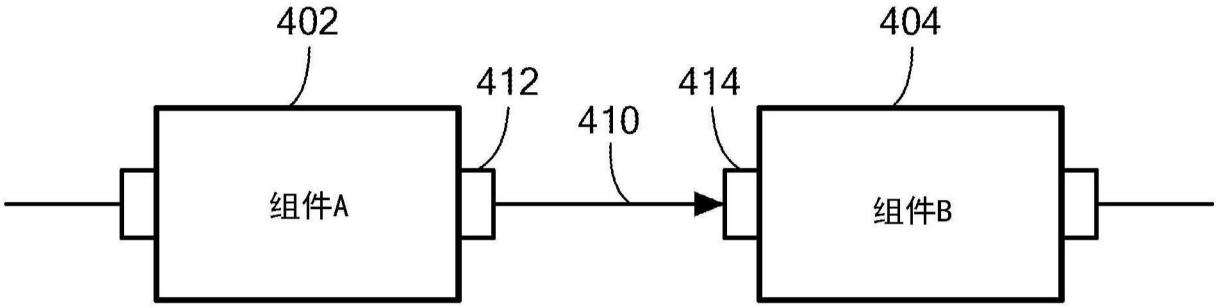


图4B

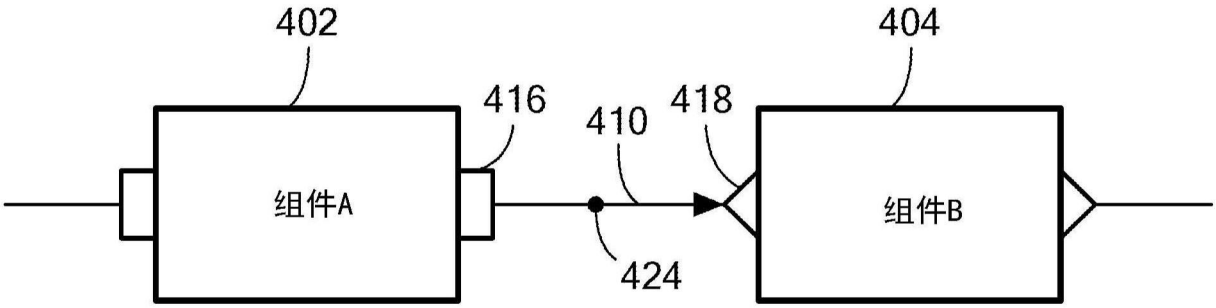


图4C

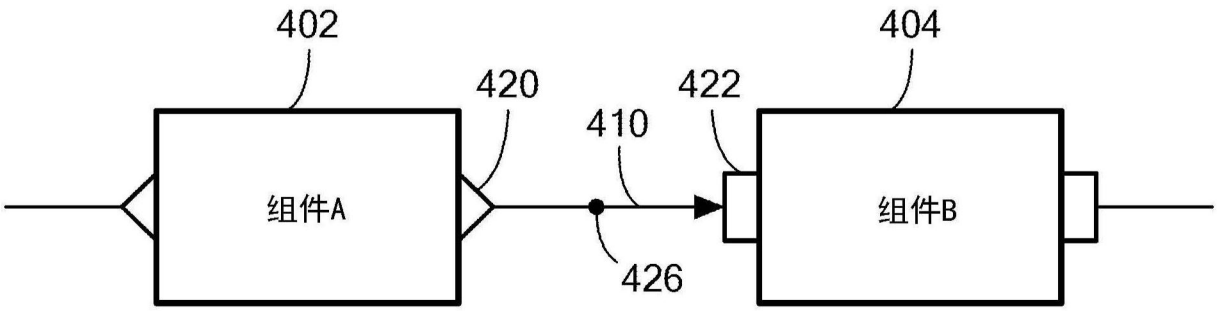


图4D

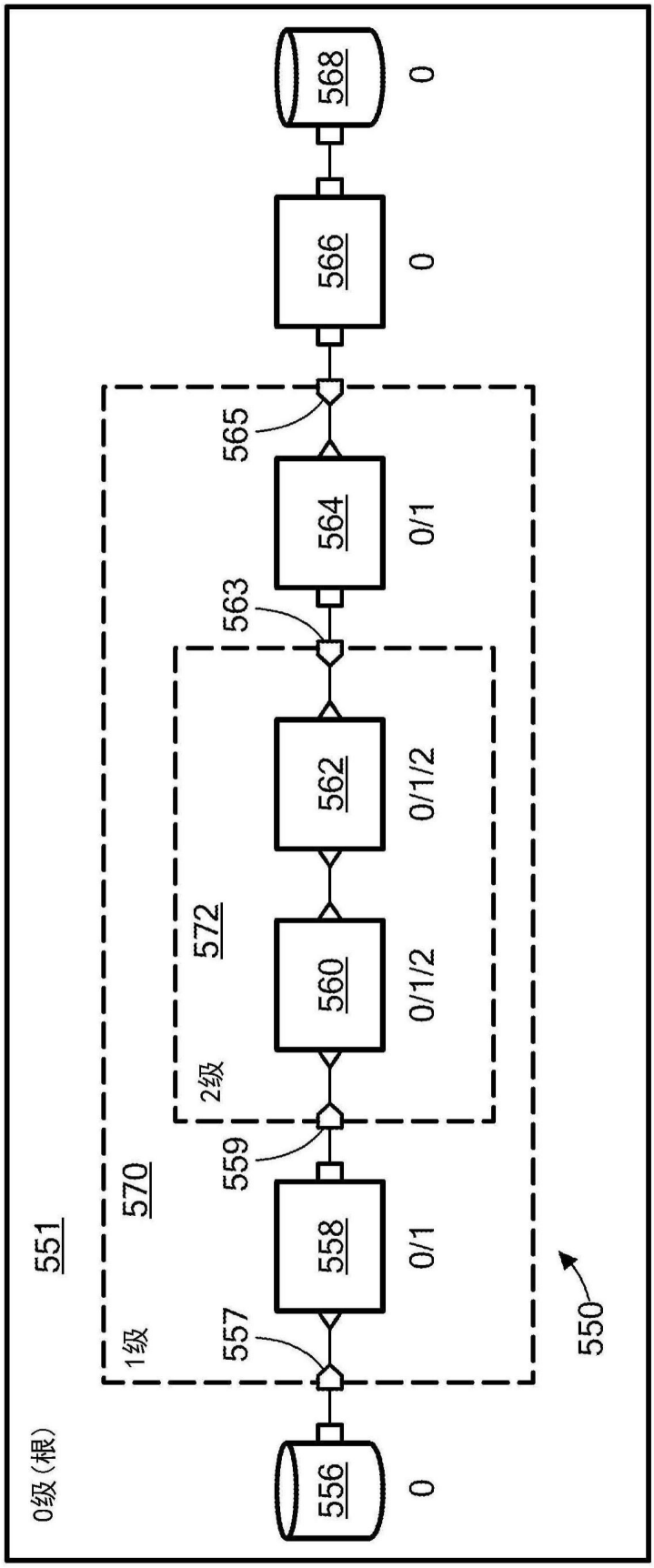

0级(根)
551
1级
570
2级
572
556
557
558
559
560
562
563
564
565
566
568
0
0/1
0/1/2
0/1/2
0/1
0
0
550

图5

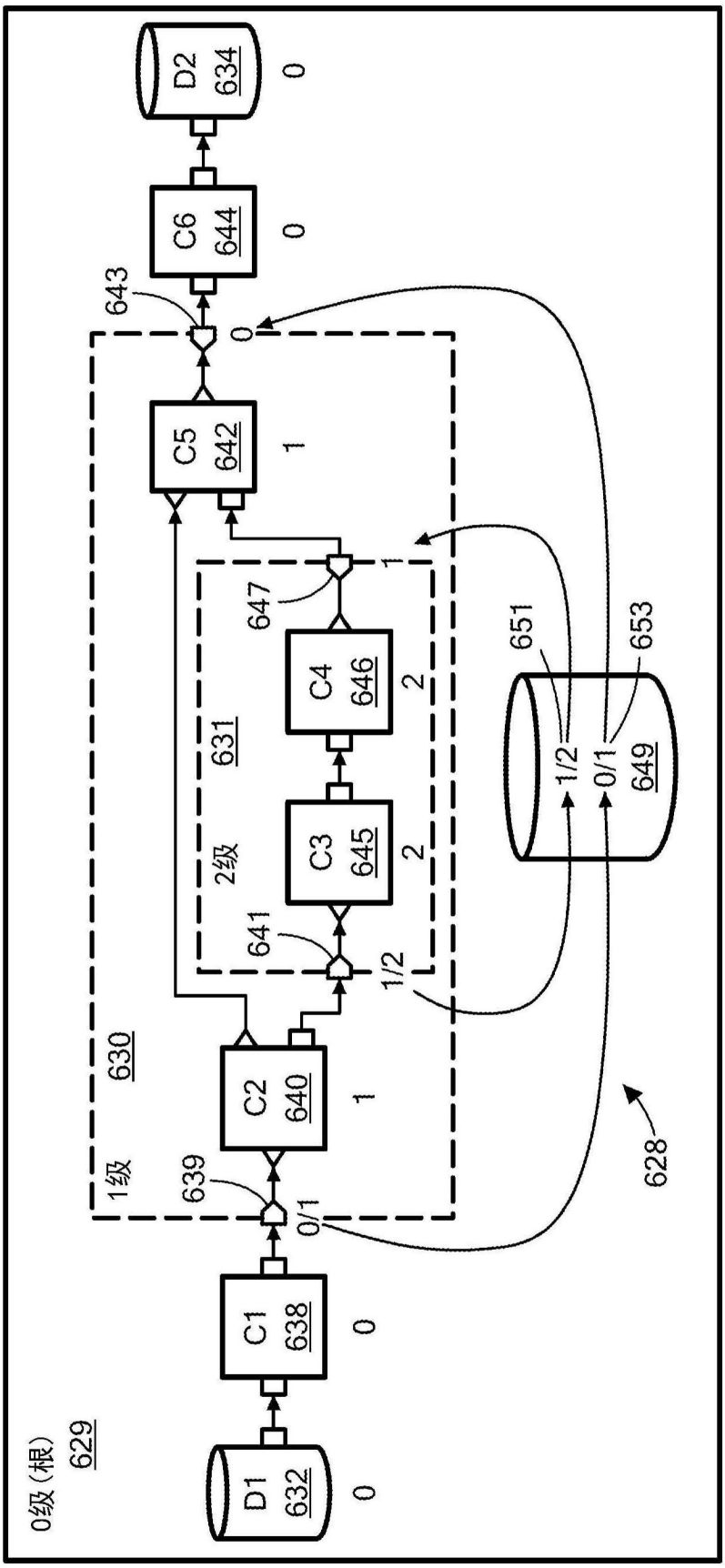

D2
634
0
C6
644
0
643
0
C5
642
1
647
1
C4
646
2
631
2级
C3
645
2
641
1/2
630
C2
640
1
1级
639
0/1
C1
638
0
D1
632
0
0级(根)
629
651
653
1/2
0/1
649
628

图6

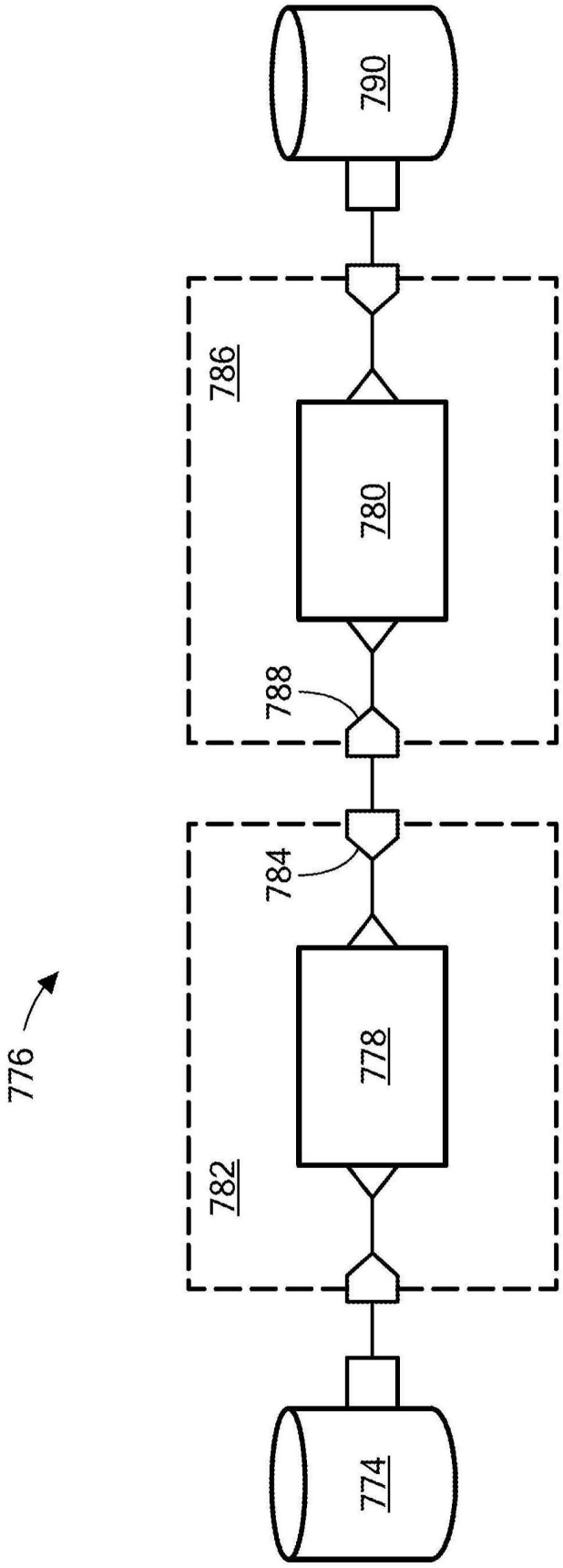


图7

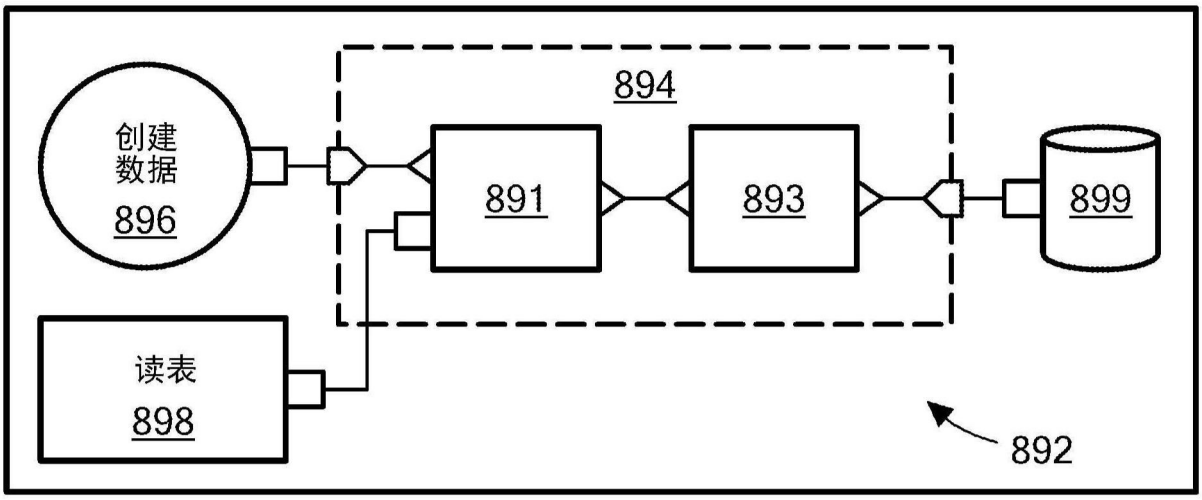

894
创建
数据
896
891
893
899
读表
898
892

图8A

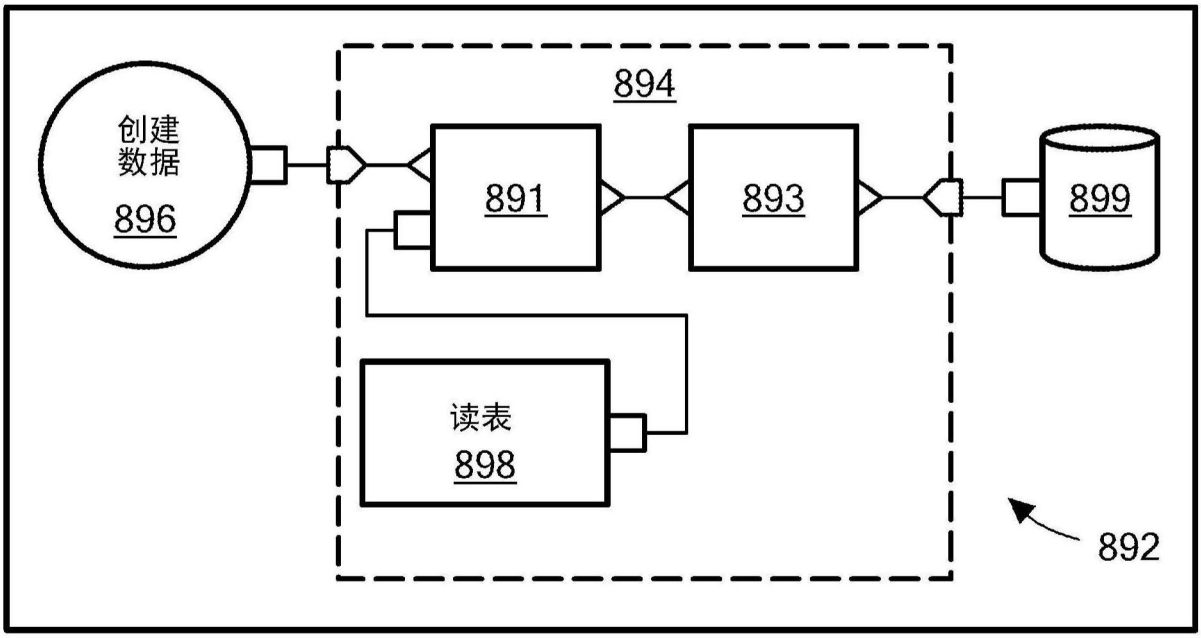

894
创建
数据
896
891
893
899
读表
898
892

图8B

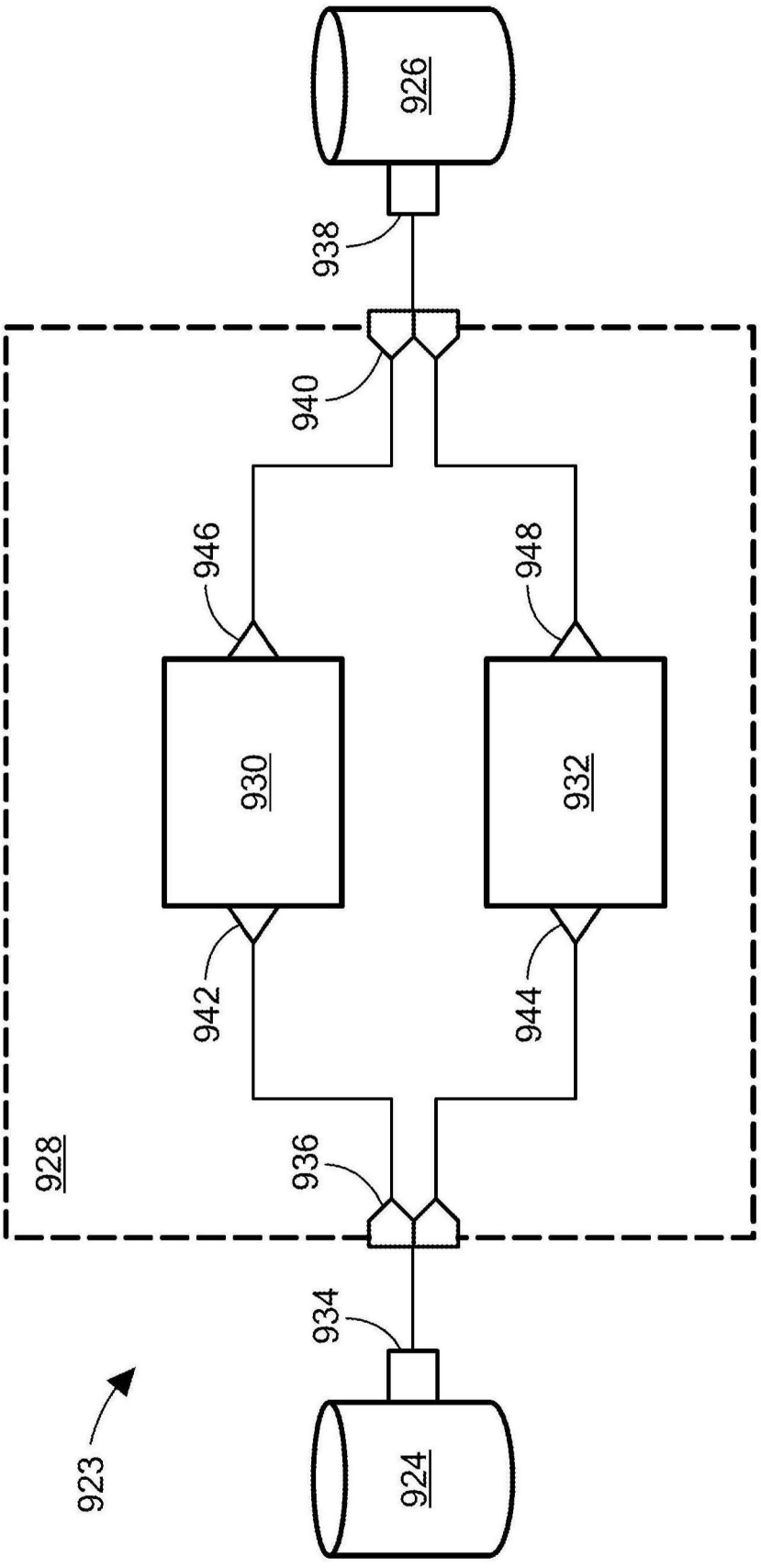


图9

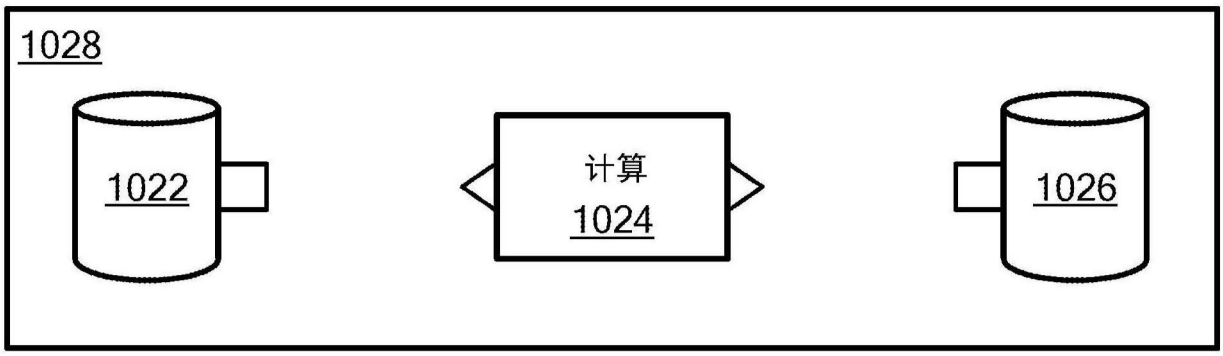


图10A

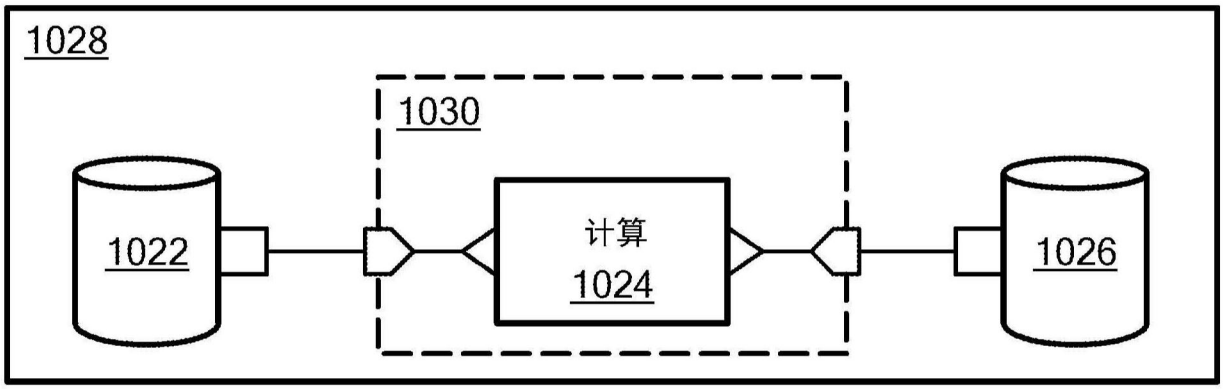


图10B

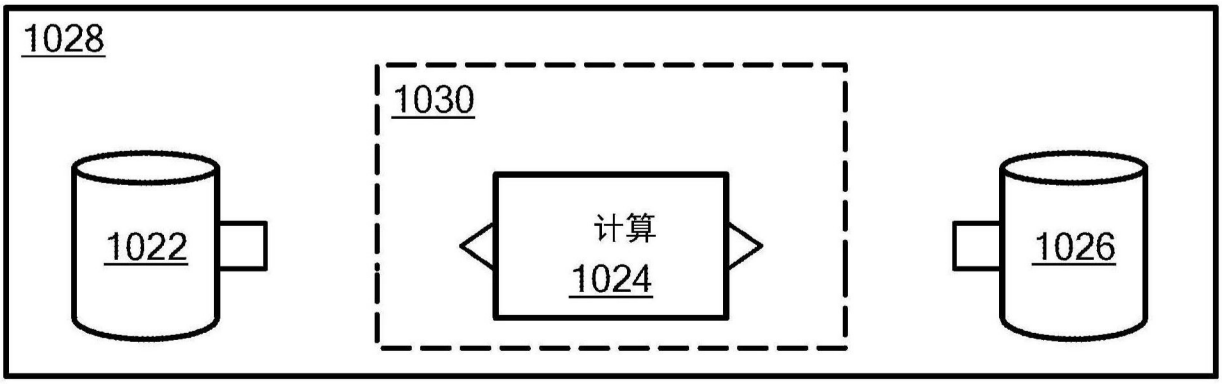


图10C

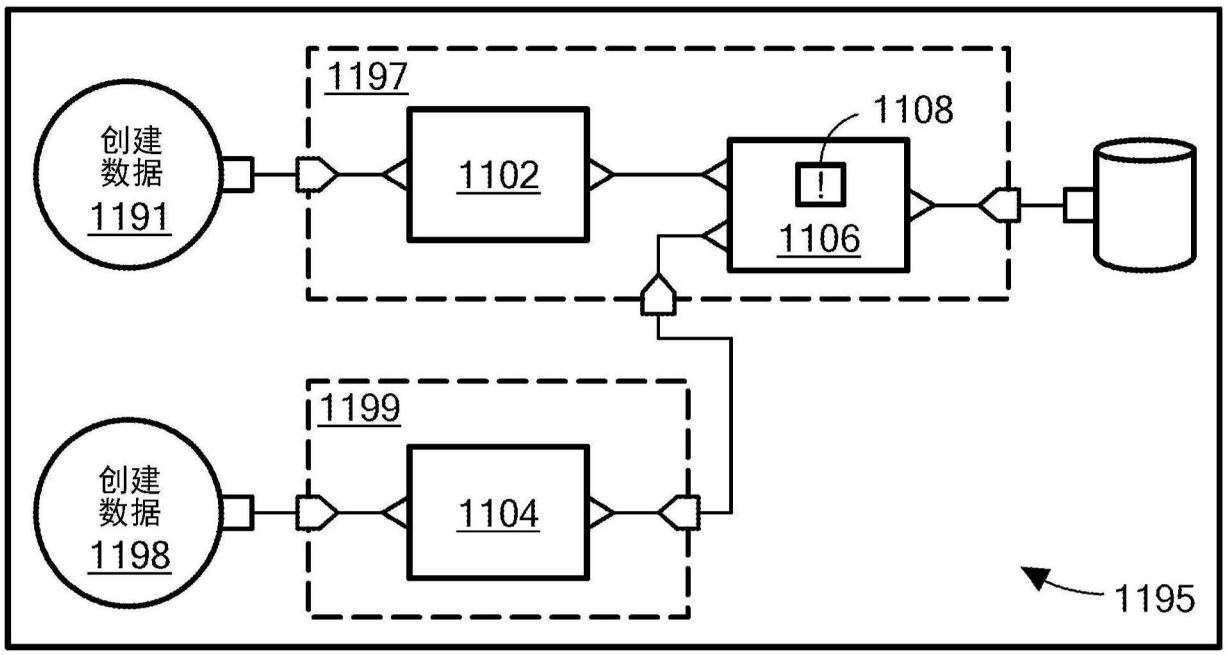

1197
1108
创建
数据
1191
1102
!
1106
1199
创建
数据
1198
1104
1195

图11A

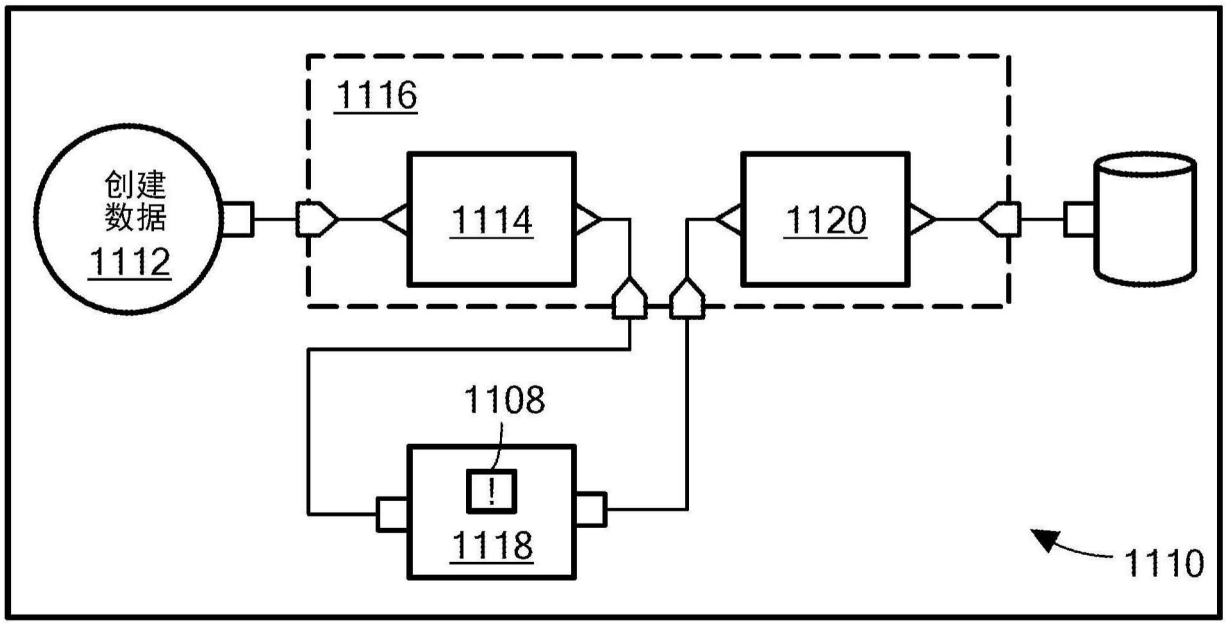
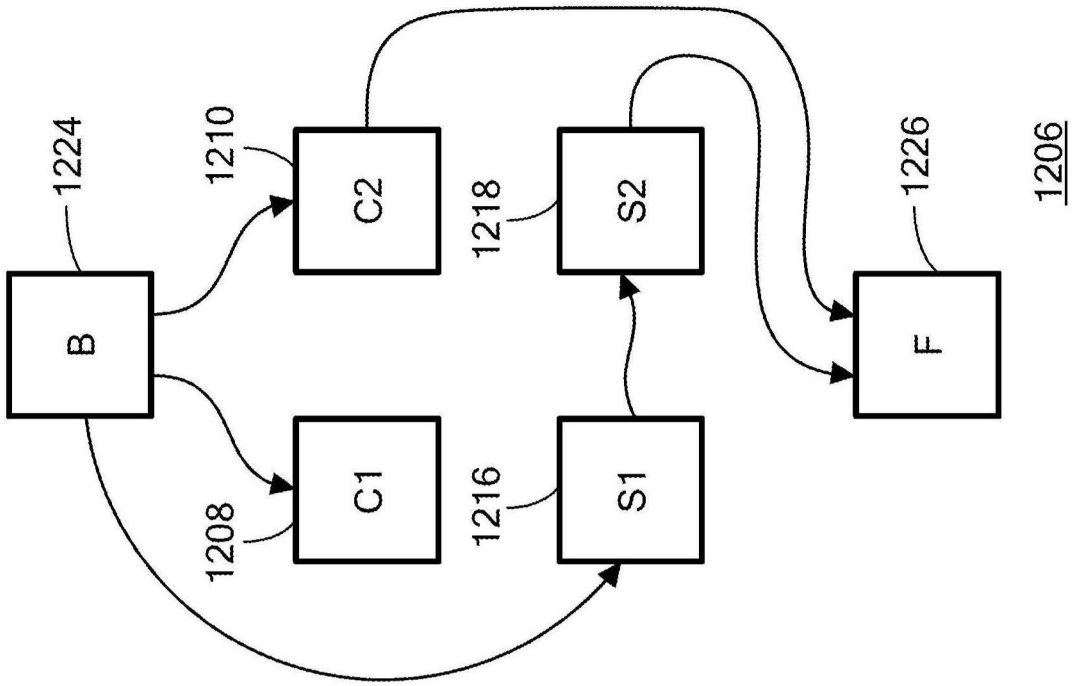

1116
创建
数据
1112
1114
1120
1108
!
1118
1110

图11B


1224
B
1210
C2
1208
C1
1218
S2
1216
S1
1226
F
1206

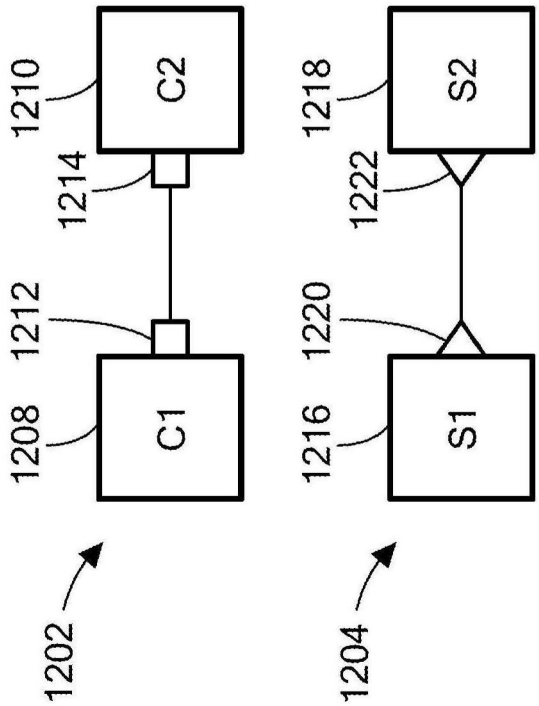

1210
C2
1214
1212
1208
C1
1202
1218
S2
1222
1220
1216
S1
1204

图12A

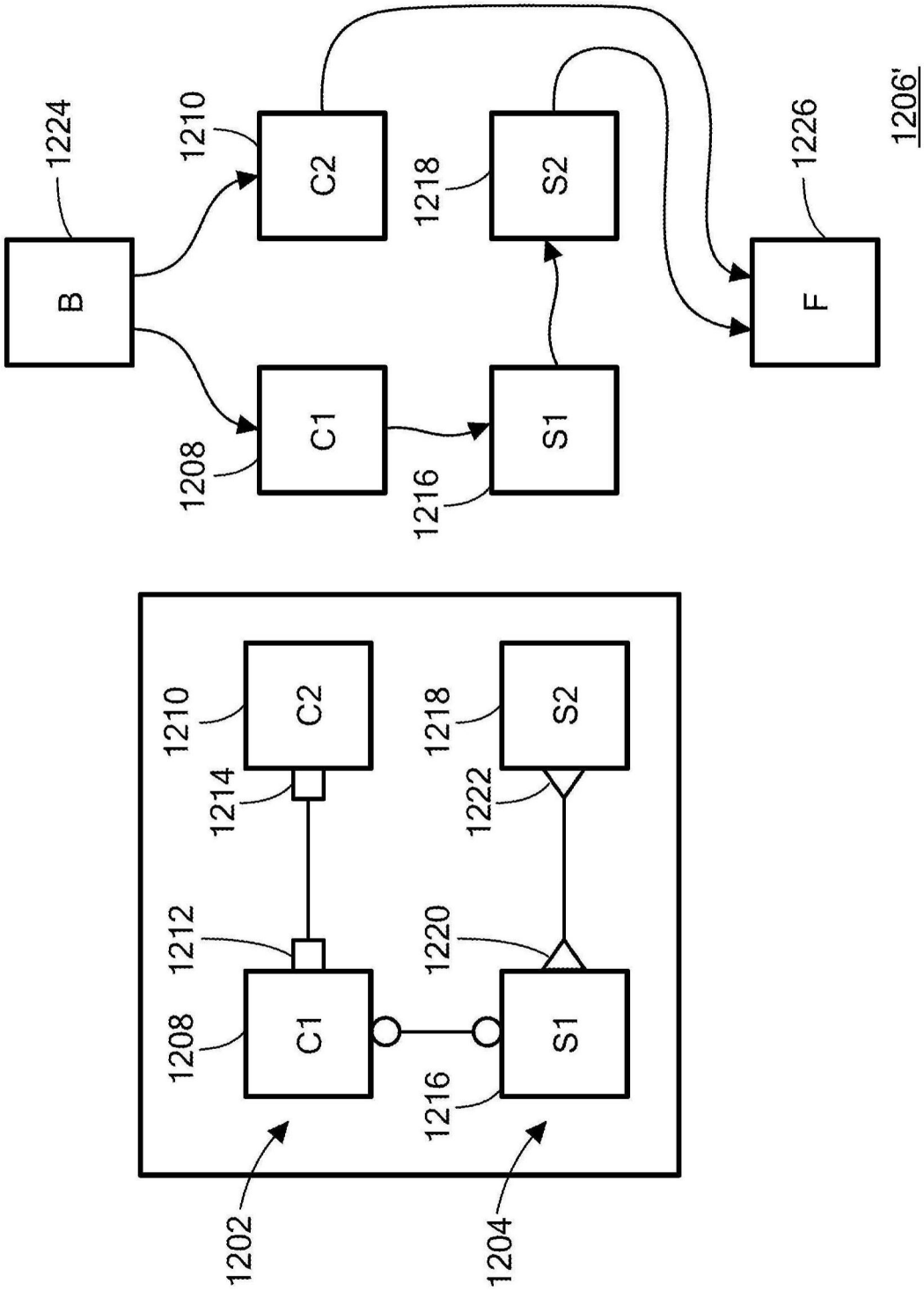

1224
B
1210
C2
1218
S2
1226
F
1206'
1208
C1
1216
S1
1210
C2
1218
S2
1214
1222
1212
1220
1208
C1
S1
1216
1202
1204

图12B

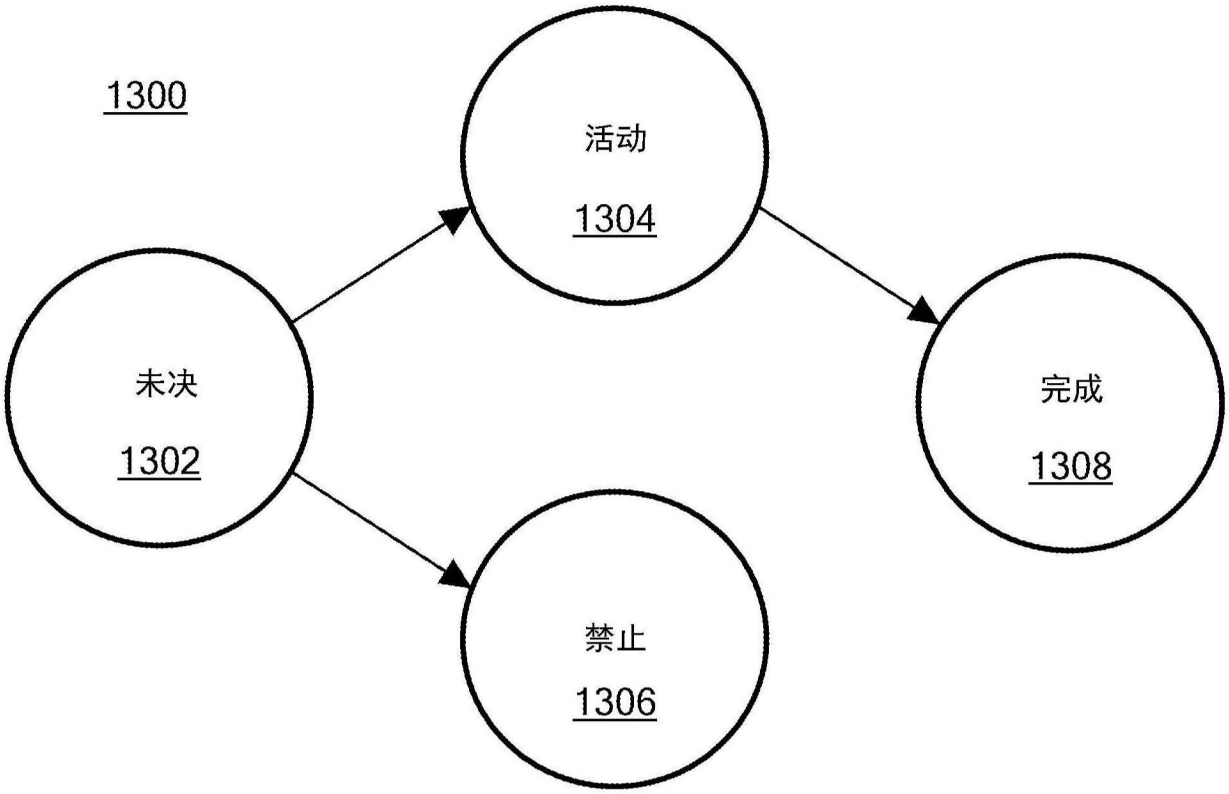


图13A

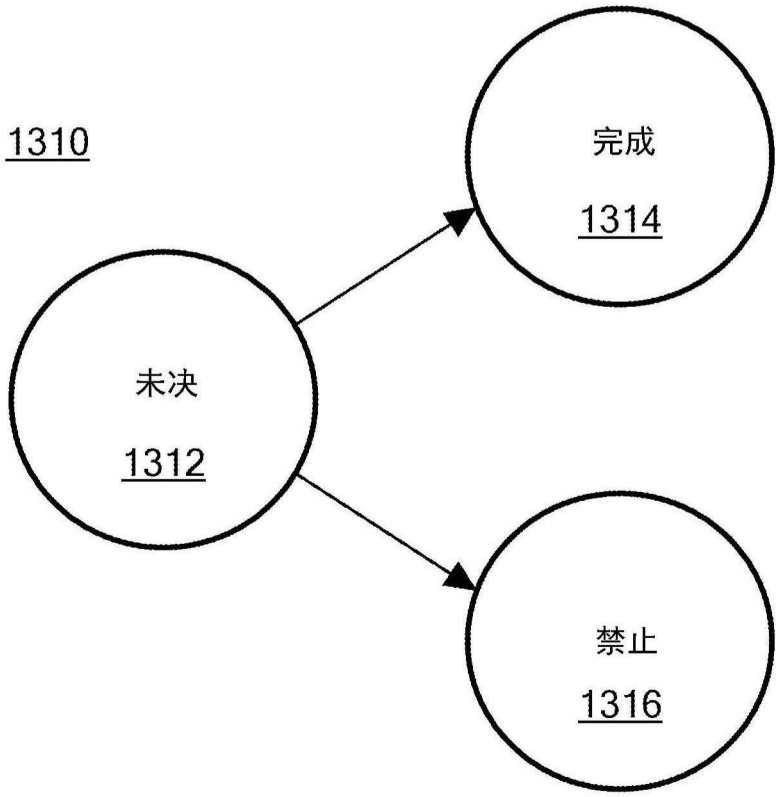


图13B

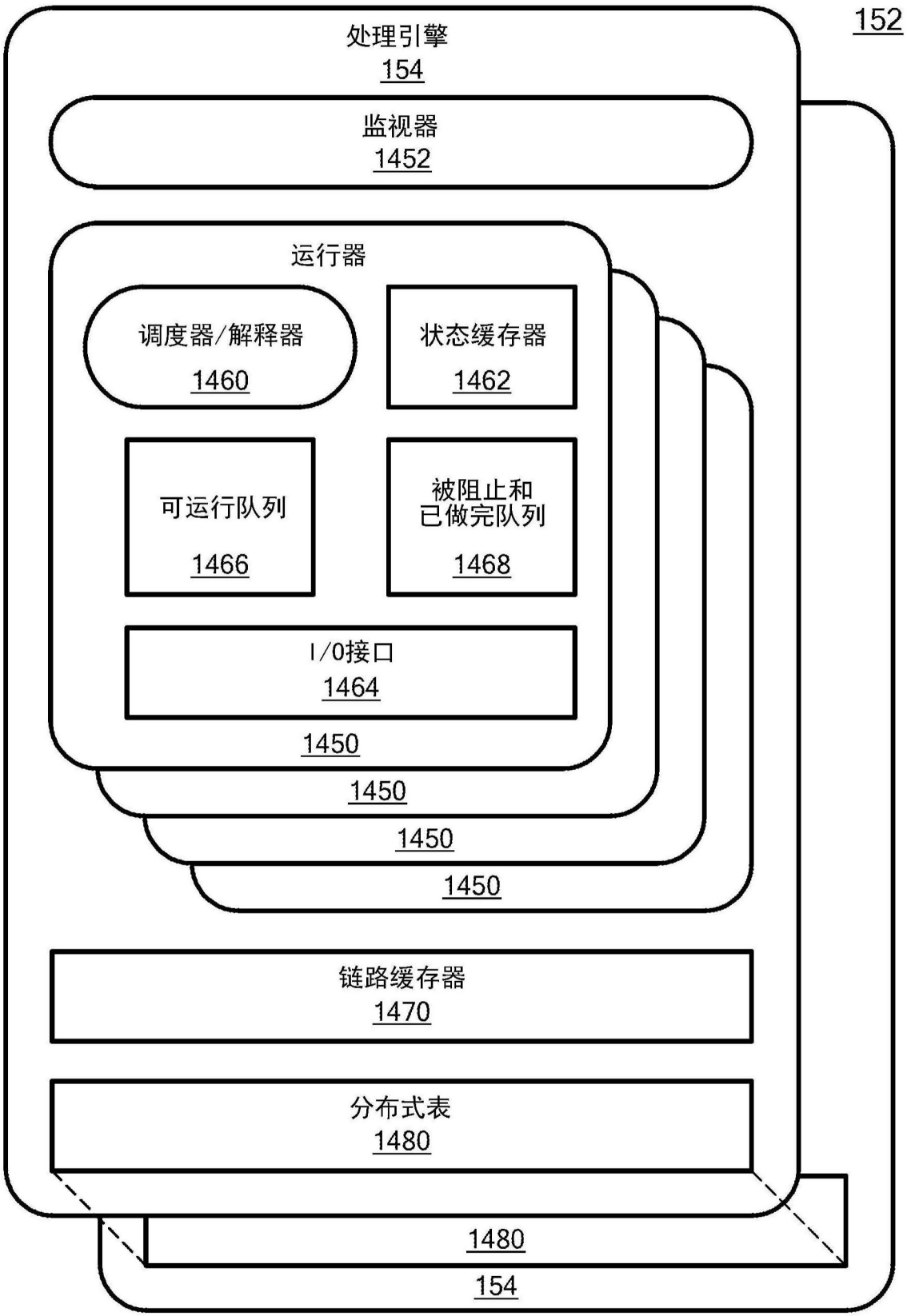

152
处理引擎
154
监视器
1452
运行器
调度器/解释器
1460
状态缓存器
1462
可运行队列
1466
被阻止和
已做完队列
1468
I/O接口
1464
1450
1450
1450
1450
链路缓存器
1470
分布式表
1480
1480
154

图14