(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 109614170 B
(45)授权公告日 2022.02.25

(21)申请号 201811207891.2

(22)申请日 2014.04.23

(65)同一申请的已公布的文献号
申请公布号 CN 109614170 A

(43)申请公布日 2019.04.12

(30)优先权数据
61/815,052 2013.04.23 US

(62)分案原申请数据
201480023408.9 2014.04.23

(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

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

(51)Int.Cl.
G06F 9/448(2018.01)

审查员 陈晓燕

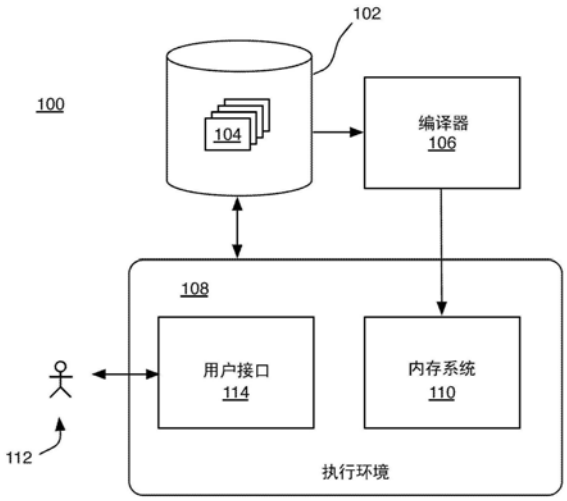
权利要求书3页 说明书14页 附图7页

(54)发明名称

控制由计算系统执行的任务

(57)摘要

对任务进行控制包括:接收指定多个任务中至少部分排序的排序信息(104);并且至少部分基于所述排序信息产生用于执行至少一些所述任务的指令。存储用于执行对应于第一任务的第一子程序的指令,所述第一子程序包括第一控制部分,所述第一控制部分控制至少对应于第二任务的第二子程序的执行,所述第一控制部分包括被配置为改变与所述第二任务相关的状态信息以及基于改变后的状态信息确定是否初始化所述第二子程序的执行的函数。存储用于执行所述第二子程序的指令,所述第二子程序包括用于执行所述第二任务的任务部分和控制对应于第三任务的第三子程序的执行的第二控制部分。



CN 109614170 B

1.一种执行任务的方法,所述方法包括:

在至少一个存储器中存储用于执行多个任务的指令,所述指令对于至少一些所述任务中的每一个任务包括相应的子程序,该子程序包括用于执行该任务的任务部分和控制后续任务的子程序执行的控制部分;并且

由至少一个处理器执行至少一些所存储的子程序,所述执行包括:

生成用于执行第一任务的第一子程序的第一进程,所述第一子程序包括第一任务部分和第一控制部分;并且

在所述第一任务部分返回之后,调用所述第一控制部分中所包括的至少一个函数,该函数确定是否生成用于执行第二子程序的第二进程,所述第二子程序不同于所述第一子程序。

2.根据权利要求1所述的方法,其中所述函数被配置为:

减小与所述第二子程序相关的计数器,并且基于所述计数器的值确定是否初始化所述第二子程序的执行。

3.根据权利要求1或2所述的方法,其中当使用识别对应于所述第二子程序的第二任务的参数来调用函数时,所述函数被配置为基于状态信息来确定是否初始化所述第二子程序的执行,所述状态信息捕捉之前以识别所述第二任务的参数来调用该函数的历史。

4.根据权利要求3所述的方法,其中所述函数是多个不同函数中的一个,并且所述状态信息捕捉之前以识别所述第二任务的参数来调用所述多个不同函数中任意函数的历史。

5.根据权利要求1所述的方法,其中所述函数被配置为:在所述第一进程中初始化所述第二子程序的执行,并且响应于所述第二子程序的执行时间超过预定阈值,生成所述第二进程以继续执行所述第二子程序。

6.根据权利要求5所述的方法,其中所述函数被配置为将与所述第一进程相关的栈框架提供给所述第二进程。

7.根据权利要求5或6所述的方法,其中所述函数被配置为在生成所述第二进程之后返回以使得所述第一进程继续与所述第二进程同时执行。

8.根据权利要求1所述的方法,其中所述第一控制部分包括确定所述第一任务部分是否被调用的逻辑。

9.根据权利要求8所述的方法,其中所述逻辑基于与所述第一任务相关的标志的值来确定所述第一任务部分是否被调用。

10.根据权利要求1所述的方法,其中所述第二子程序对应于第二任务,并且所述第一控制部分至少部分基于指定多个任务中至少部分排序的排序信息,所述多个任务包括所述第一任务和所述第二任务。

11.根据权利要求10所述的方法,其中所述排序信息包括控制流图,所述控制流图包括在表示相应任务的节点的各对之间的有向边,其中从上游节点到下游节点的有向边表明在所述部分排序中由所述上游节点表示的所述任务先于由所述下游节点表示的所述任务。

12.根据权利要求1所述的方法,其中调用所述第一控制部分中所包括的至少一个函数包括调用所述第一控制部分中所包括的、确定是否生成所述第二进程的至少一个函数,所述第二进程用于在没有调用中央任务调度进程的情况下执行所述第二子程序。

13.一种计算机可读存储介质,具有存储于其上的计算机程序,用于执行任务,所述计

算机程序包括用于使计算系统执行下述步骤的指令：

存储用于执行多个任务的指令，所述指令对于至少一些所述任务中的每一个任务包括相应的子程序，该子程序包括用于执行该任务的任务部分和控制后续任务的子程序执行的控制部分；并且

执行至少一些所存储的子程序，所述执行包括：

生成用于执行第一任务的第一子程序的第一进程，所述第一子程序包括第一任务部分和第一控制部分；并且

在所述第一任务部分返回之后，调用所述第一控制部分中所包括的至少一个函数，该函数确定是否生成用于执行第二子程序的第二进程，所述第二子程序不同于所述第一子程序。

14.一种用于执行任务的计算系统，所述计算系统包括：

至少一个存储器，用于存储执行多个任务的指令，所述指令对于至少一些所述任务中的每一个任务包括相应的子程序，该子程序包括用于执行该任务的任务部分和控制后续任务的子程序执行的控制部分；以及

至少一个处理器，被配置为执行至少一些所存储的子程序，所述执行包括：

生成用于执行第一任务的第一子程序的第一进程，所述第一子程序包括第一任务部分和第一控制部分；并且

在所述第一任务部分返回之后，调用所述第一控制部分中所包括的至少一个函数，该函数确定是否生成用于执行第二子程序的第二进程，所述第二子程序不同于所述第一子程序。

15.根据权利要求14所述的计算系统，其中所述至少一个函数被配置为：

减小与所述第二子程序相关的计数器，并且基于所述计数器的值确定是否初始化所述第二子程序的执行。

16.根据权利要求14所述的计算系统，其中当使用识别对应于所述第二子程序的第二任务的参数来调用所述至少一个函数时，所述至少一个函数被配置为基于状态信息来确定是否初始化所述第二子程序的执行，所述状态信息捕捉之前以识别所述第二任务的参数来调用所述至少一个函数的历史。

17.根据权利要求14所述的计算系统，其中所述至少一个函数被配置为：在所述第一进程中初始化所述第二子程序的执行，并且响应于所述第二子程序的执行时间超过预定阈值，生成所述第二进程以继续执行所述第二子程序。

18.根据权利要求17所述的计算系统，其中所述至少一个函数被配置为在生成所述第二进程之后返回以使得所述第一进程继续与所述第二进程同时执行。

19.根据权利要求14所述的计算系统，

其中所述第一控制部分包括确定所述第一任务部分是否被调用的逻辑，并且

其中所述逻辑还基于与所述第一任务相关的标志的值来确定所述第一任务部分是否被调用。

20.根据权利要求14所述的计算系统，

其中所述第二子程序对应于第二任务，并且所述第一控制部分至少部分基于指定多个任务中至少部分排序的排序信息，所述多个任务包括所述第一任务和所述第二任务，并且

其中所述排序信息包括控制流图，所述控制流图包括在表示相应任务的节点的各对之间的有向边，其中从上游节点到下游节点的有向边表明在所述部分排序中由所述上游节点表示的所述任务先于由所述下游节点表示的所述任务。

21.根据权利要求14所述的计算系统，其中调用所述第一控制部分中所包括的至少一个函数包括调用所述第一控制部分中所包括的、确定是否生成所述第二进程的至少一个函数，所述第二进程用于在没有调用中央任务调度进程的情况下执行所述第二子程序。

22.一种用于执行任务的计算系统，所述计算系统包括：

存储用于执行多个任务的指令的装置，所述指令对于至少一些所述任务中的每一个任务包括相应的子程序，该子程序包括用于执行该任务的任务部分和控制后续任务的子程序执行的控制部分；以及

用于执行至少一些所存储的子程序的装置，所述执行包括：

生成用于执行第一任务的第一子程序的第一进程，所述第一子程序包括第一任务部分和第一控制部分；并且

在所述第一任务部分返回之后，调用所述第一控制部分中所包括的至少一个函数，该函数确定是否生成用于执行第二子程序的第二进程，所述第二子程序不同于所述第一子程序。

23.根据权利要求22所述的计算系统，其中所述至少一个函数被配置为：

减小与所述第二子程序相关的计数器，并且基于所述计数器的值确定是否初始化所述第二子程序的执行。

24.根据权利要求22所述的计算系统，其中所述至少一个函数被配置为：在所述第一进程中初始化所述第二子程序的执行，并且响应于所述第二子程序的执行时间超过预定阈值，生成所述第二进程以继续执行所述第二子程序。

25.根据权利要求24所述的计算系统，其中所述至少一个函数被配置为在生成所述第二进程之后返回以使得所述第一进程继续与所述第二进程同时执行。

26.根据权利要求22所述的计算系统，

其中所述第一控制部分包括确定所述第一任务部分是否被调用的逻辑，并且

其中所述逻辑还基于与所述第一任务相关的标志的值来确定所述第一任务部分是否被调用。

27.根据权利要求22所述的计算系统，

其中所述第二子程序对应于第二任务，并且所述第一控制部分至少部分基于指定多个任务中至少部分排序的排序信息，所述多个任务包括所述第一任务和所述第二任务，并且

其中所述排序信息包括控制流图，所述控制流图包括在表示相应任务的节点的各对之间的有向边，其中从上游节点到下游节点的有向边表明在所述部分排序中由所述上游节点表示的所述任务先于由所述下游节点表示的所述任务。

28.根据权利要求22所述的计算系统，其中调用所述第一控制部分中所包括的至少一个函数包括调用所述第一控制部分中所包括的、确定是否生成所述第二进程的至少一个函数，所述第二进程用于在没有调用中央任务调度进程的情况下执行所述第二子程序。

控制由计算系统执行的任务

[0001] 本申请是申请号为201480023408.9(对应于PCT国际申请号PCT/US2014/035094)、发明名称为“控制由计算系统执行的任务”、申请日为2014年04月23日的发明专利申请的分案申请。

[0002] 相关申请的交叉引用

[0003] 本申请要求享有2013年4月23日提交的61/815,052号美国专利申请的优先权。

技术领域

[0004] 本申请涉及控制由计算系统执行的任务。

背景技术

[0005] 在控制由计算系统执行的任务的一些技术中,由为一项任务生成的进程或线程来执行该单独任务,并且在完成该任务后,此进程或线程结束。计算系统的操作系统、或利用操作系统的特征的其他集中控制实体可以被用来调度不同的任务或者管理不同任务之间的通讯。可以使用控制流图来限定多个任务的部分排序,该控制流图表明在其他下游任务(例如,任务B)开始之前某些上游任务(例如,任务A)必须完成。可能存在管理新进程的生成的控制进程,用于根据控制流图执行任务。在控制进程生成用于执行任务A的进程A之后,控制进程等待操作系统发出进程A已经结束的通知。在进程A结束之后,操作系统通知控制进程,然后控制进程生成用于执行任务B的进程B。

发明内容

[0006] 在一个方案中,通常,一种控制由计算系统执行的任务的方法,所述方法包括:接收指定多个任务中至少部分排序的排序信息;并且使用至少一个处理器至少部分基于所述排序信息产生用于执行至少一些所述任务的指令。所述产生包括:存储用于执行对应于第一任务的第一子程序的指令,所述第一子程序包括第一控制部分,所述第一控制部分控制至少对应于第二任务的第二子程序的执行,所述第一控制部分包括被配置为改变与所述第二任务相关的状态信息以及基于改变后的状态信息确定是否初始化所述第二子程序的执行的函数;并且存储用于执行所述第二子程序的指令,所述第二子程序包括用于执行所述第二任务的任务部分和控制对应于第三任务的第三子程序的执行的第二控制部分。

[0007] 这些方案可包括一个或多个以下特征。

[0008] 所述排序信息包括控制流图,所述控制流图包括在表示相应任务的节点的各对之间的有向边,其中从上游节点到下游节点的有向边表明在所述部分排序中由所述上游节点表示的所述任务先于由所述下游节点表示的所述任务。

[0009] 所述控制流图包括在表示所述第一任务的第一节点与表示所述第二任务的第二节点之间的有向边、以及在所述第二节点与表示所述第三任务的第三节点之间的有向边。

[0010] 所述函数被配置为:减小或增大与所述第二任务相关的计数器,并且基于所述计数器的值确定是否初始化所述第二子程序的执行。

[0011] 所述函数被配置为执行原子地减小或增大所述计数器且读取所述计数器的值的原子操作。

[0012] 所述改变后的状态信息包括之前以识别所述第二任务的参数调用所述函数的调用历史。

[0013] 所述函数是多个不同函数中的一个，并且所述状态信息捕捉之前以识别所述第二任务的参数来调用所述多个不同函数中任意函数的调用历史。

[0014] 所述第二控制部分包括确定用于执行所述任务的所述任务部分是否被调用的逻辑。

[0015] 所述逻辑基于与所述第二任务相关的标志的值来确定用于执行所述任务的所述任务部分是否被调用。

[0016] 所述第一控制部分控制至少以下子程序的执行：对应于所述第二任务的所述第二子程序和对应于第四任务的第四子程序。

[0017] 所述排序信息表明在所述部分排序中所述第一任务先于所述第二任务，并且表明在所述部分排序中所述第一任务先于所述第四任务，并且不约束所述部分排序中所述第二任务和所述第四任务相对于彼此的顺序。

[0018] 所述第一控制部分包括：确定是否生成用于执行所述第二子程序的新进程的第一函数，以及使用与执行所述第一子程序相同的进程来初始化所述第四子程序的执行的第二函数。

[0019] 在另一个方案中，通常，一种存储在计算机可读存储介质上的计算机程序，用于控制任务。所述计算机程序包括用于使计算系统执行下述步骤的指令：接收指定多个任务中至少部分排序的排序信息；并且至少部分基于所述排序信息产生用于执行至少一些所述任务的指令。所述产生包括：存储用于执行对应于第一任务的第一子程序的指令，所述第一子程序包括第一控制部分，所述第一控制部分控制至少对应于第二任务的第二子程序的执行，所述第一控制部分包括被配置为改变与所述第二任务相关的状态信息以及基于改变后的状态信息确定是否初始化所述第二子程序的执行的函数；并且存储用于执行所述第二子程序的指令，所述第二子程序包括用于执行所述第二任务的任务部分和控制对应于第三任务的第三子程序的执行的第二控制部分。

[0020] 在另一个方案中，通常，一种用于控制任务的计算系统包括：输入设备或端口，被配置为接收指定多个任务中至少部分排序的排序信息；以及至少一个处理器，被配置为至少部分基于所述排序信息产生用于执行至少一些所述任务的指令。所述产生包括：存储用于执行对应于第一任务的第一子程序的指令，所述第一子程序包括第一控制部分，所述第一控制部分控制至少对应于第二任务的第二子程序的执行，所述第一控制部分包括被配置为改变与所述第二任务相关的状态信息以及基于改变后的状态信息确定是否初始化所述第二子程序的执行的函数；并且存储用于执行所述第二子程序的指令，所述第二子程序包括用于执行所述第二任务的任务部分和控制对应于第三任务的第三子程序的执行的第二控制部分。

[0021] 在另一个方案中，通常，一种用于控制任务的计算系统包括：用于接收排序信息的装置，所述排序信息指定多个任务中的至少部分排序；以及用于产生指令的装置，所述指令至少部分基于所述排序信息来执行至少一些所述任务。所述产生包括：存储用于执行对应

于第一任务的第一子程序的指令，所述第一子程序包括第一控制部分，所述第一控制部分控制至少对应于第二任务的第二子程序的执行，所述第一控制部分包括被配置为改变与所述第二任务相关的状态信息以及基于改变后的状态信息确定是否初始化所述第二子程序的执行的函数；并且存储用于执行所述第二子程序的指令，所述第二子程序包括用于执行所述第二任务的任务部分和控制对应于第三任务的第三子程序的执行的第二控制部分。

[0022] 在另一个方案中，通常，一种执行任务的方法包括：在至少一个存储器中存储用于执行多个任务的指令，所述指令对于至少一些所述任务中的每一个任务包括相应的子程序，该子程序包括用于执行该任务的任务部分和控制后续任务的子程序执行的控制部分；并且由至少一个处理器执行至少一些所存储的子程序。所述执行包括：生成用于执行第一任务的第一子程序的第一进程，所述第一子程序包括第一任务部分和第一控制部分；并且在所述第一任务部分返回之后，调用包含在所述第一控制部分中的至少一个函数，该函数确定是否生成用于执行第二子程序的第二进程。

[0023] 这些方案可包括一个或多个以下特征。

[0024] 所述函数被配置为：减小与所述第二子程序相关的计数器，并且基于所述计数器的值确定是否初始化所述第二子程序的执行。

[0025] 当使用识别对应于所述第二子程序的第二任务的参数来调用函数时，所述函数被配置为基于状态信息来确定是否初始化所述第二子程序的执行，所述状态信息捕捉之前以识别所述第二任务的参数来调用该函数的调用历史。

[0026] 所述函数是多个不同函数中的一个，并且所述状态信息捕捉之前以识别所述第二任务的参数来调用所述多个不同函数中任意函数的调用历史。

[0027] 所述函数被配置为：初始化所述第一进程中所述第二子程序的执行，并且响应于所述第二子程序的执行时间超过预定阈值，生成所述第二进程以继续执行所述第二子程序。

[0028] 所述函数被配置为将与所述第一进程相关的栈框架提供给所述第二进程。

[0029] 所述函数被配置为在生成所述第二进程之后返回以使得所述第一进程继续与所述第二进程同时执行。

[0030] 所述第一控制部分包括确定所述第一任务部分是否被调用的逻辑。

[0031] 所述逻辑基于与所述第一任务相关的标志的值来确定所述第一任务部分是否被调用。

[0032] 所述第二子程序对应于第二任务，并且所述第一控制部分至少部分基于指定多个任务中至少部分排序的排序信息，所述多个任务包括所述第一任务和所述第二任务。

[0033] 所述排序信息包括控制流图，所述控制流图包括在表示相应任务的节点的各对之间的有向边，其中从上游节点到下游节点的有向边表明在所述部分排序中由所述上游节点表示的所述任务先于由所述下游节点表示的所述任务。

[0034] 在另一个方案中，通常，一种计算机程序存储在计算机可读存储介质上，用于执行任务。所述计算机程序包括用于使计算系统执行下述步骤的指令：存储用于执行多个任务的指令，所述指令对于至少一些所述任务中的每一个任务包括相应的子程序，该子程序包括用于执行该任务的任务部分和控制后续任务的子程序执行的控制部分；并且执行至少一些所存储的子程序。所述执行包括：生成用于执行第一任务的第一子程序的第一进程，所述

第一子程序包括第一任务部分和第一控制部分;并且在所述第一任务部分返回之后,调用包含在所述第一控制部分中的至少一个函数,该函数确定是否生成用于执行第二子程序的第二进程。

[0035] 在另一个方案中,通常,一种用于执行任务的计算系统包括:至少一个存储器,存储用于执行多个任务的指令,所述指令对于至少一些所述任务中的每一个任务包括相应的子程序,该子程序包括用于执行该任务的任务部分和控制后续任务的子程序执行的控制部分;以及至少一个处理器,被配置为执行至少一些所存储的子程序。所述执行包括:生成用于执行第一任务的第一子程序的第一进程,所述第一子程序包括第一任务部分和第一控制部分;并且在所述第一任务部分返回之后,调用包含在所述第一控制部分中的至少一个函数,该函数确定是否生成用于执行第二子程序的第二进程。

[0036] 在另一个方案中,通常,一种用于执行任务的计算系统包括:存储用于执行多个任务的指令的装置,所述指令对于至少一些所述任务中的每一个任务包括相应的子程序,该子程序包括用于执行所述任务的任务部分和控制后续任务的子程序执行的控制部分;以及用于执行至少一些所存储的子程序的装置。所述执行包括:生成用于执行第一任务的第一子程序的第一进程,所述第一子程序包括第一任务部分和第一控制部分;并且在所述第一任务部分返回之后,调用包含在所述第一控制部分中的至少一个函数,该函数确定是否生成用于执行第二子程序的第二进程。

[0037] 这些方案可包括一个或多个以下优点。

[0038] 当由计算系统执行任务时,在生成用于执行任务的新进程时需要花费处理时间,并且在任务进程与调度器或保持任务相关性和排序的其他中央进程之间来回交换时需要花费时间。本文描述的技术能够以计算有效的方式来选择性地生成新进程,或者选择性地再用运行进程。基于被添加至用于执行任务的子程序的相对少量代码,编译器能够避免唯一地依赖于具有分散调度机制的集中调度器。以允许并发性和条件逻辑的方式,完成任务自然使得计算系统根据输入约束(诸如,控制流图)执行其他任务。编译器生成的与任务相关的代码在运行期间调用函数以基于存储在计数器和标志中的状态信息来确定是否执行其他任务。由此,在运行期间,编译器生成的所述代码有效地实施了控制任务子程序的调用的状态机。免于切换至调度器和从调度器切换的额外开销,计算系统可以更有效地执行细粒度的潜在同时发生的任务。

[0039] 通过以下说明书和权利要求书,本发明的其它特征和优点将变得显而易见。

附图说明

[0040] 图1是计算系统的方框图。

[0041] 图2A是控制流图的图表。

[0042] 图2B-图2D是与执行图2A的控制流图的节点的子程序相关的进程生命周期的图表。

[0043] 图3和图4是控制流图的图表。

具体实施方式

[0044] 图1示出其中可以使用任务控制技术的计算系统100的示例。系统100包括用于存

储任务说明104的存储系统102、用于将任务说明编译成任务子程序以执行任务的编译器106、以及用于执行被加载至存储器系统110的任务子程序的执行环境108。每个任务说明104对将要执行的任务进行编码，并且当可以执行上述任务时进行约束(包括不同任务之间的排序约束)。任务说明104中的一些可以由通过执行环境108的用户接口114相互作用的用户112来构建。所述执行环境108可能被托管在例如受一个合适的操作系统(诸如UNIX操作系统的一个版本)控制的一个或多个通用计算机。例如，所述执行环境108还可以包括多节点并行计算环境，该多节点并行计算环境包括使用多个中央处理器(CPU)或处理器内核的计算机系统的配置，这些CPU或处理器内核可以是本地(例如多处理器系统，诸如对称多处理(symmetric multi-processing,SMP)计算机)，或本地分布式(例如多个处理器耦合为集群或大规模并行处理(MPP)系统)，或者远程，或远程分布式(例如通过局域网(LAN)和/或广域网(WAN)来耦合的多个处理器)，或其组合。提供所述存储系统102的存储设备可能是所述执行环境108本地的存储设备，例如，存储在被连接到托管所述执行环境108的计算机的存储介质(例如，硬盘驱动器)上;或可能是所述执行环境108远程的存储设备，例如，被托管在通过远程连接(例如，由云计算基础架构提供)与托管所述执行环境108的计算机进行通讯的远程系统上。

[0045] 图2A示出控制流图200的示例，其限定了将被施加于由计算系统100执行的一组任务上的部分排序。由控制流图200限定的部分排序被编码成存储的任务说明104。在一些实施方式中，用户112选择将被包含在控制流图中的各类节点，并且使用表示被连接节点之间的排序约束的链路来连接这些节点中的一些。一类节点是任务节点，由图2A中的直角方块表示。每个任务节点表示待执行的不同任务。从第一任务节点(在有向链路的开始处)连接至第二任务节点(在有向链路的终点处)的有向链路施加这样的排序约束:在第二节点的任务可以开始之前第一节点的任务必须完成。另一类节点是结节点，由图2A中的圆角方块表示。除非控制流图包括条件行为，否则结节点仅仅用于施加排序约束。具有单个输入链路和多个输出链路的结节点施加排序约束，使得在由输出链路连接的任务节点中的任何任务可以开始之前由输入链路连接的任务节点中的任务必须完成。具有多输入链路和单个输出链路的结节点施加排序约束，使得在由输出链路连接的任务节点中的任务可以开始之前由输入链路连接的任务节点中的所有任务必须完成。任务节点还可以是多个输入链路的终点，其施加排序约束，使得在该任务节点中的任务可以开始之前由输入链路连接的任务节点中的所有任务必须完成。利用条件行为，具有多个输入链路的任务节点还提供与具有多个输入的结节点不同的逻辑行为，在下文进行更详细地的描述。

[0046] 在构建控制流图之后，编译器106编译任务说明104(其中任务说明104对控制流图表示的任务信息和排序信息进行编码)，并且产生用于执行任务的指令。该指令可以是准备好被执行的低级机器代码的形式，或是更高级代码的形式(其被进一步编译为提供最终将被执行的低级机器代码)。所生成的指令包括每个任务节点的子程序(“任务子程序”)、以及每个结节点的子程序(“结点子程序”)。每个任务子程序包括用于执行相应任务的任务部分(也称为任务体)。任务节点包括待执行的相应任务的一些描述，使得编译器能够生成适当的任务部分。例如，在一些实施方式中，任务节点识别将被调用的特定函数、将要运行的程序、或包含在任务部分中的可执行代码。一些任务子程序还可以包括控制部分，所述控制部分控制控制流图中另一个节点的后续子程序的执行。因为在控制完成之后不需要将其传至

任何后续任务，所以未被连接至任何下游节点的任务节点的任务子程序可以不需要控制部分。因为结节点的目的是对控制流指定约束，所以每个结点子程序包括控制部分作为其主体。

[0047] 包含在控制部分中的函数的示例是“chain(链)”函数，其基于与控制流图的节点相关的状态信息确定是否生成用于执行后续节点的子程序的新进程。chain函数的参数(argument)识别后续节点。下表示出被包含在由编译器为控制流图200的每个节点编写的子程序中函数的示例，其中任务子程序的任务部分由函数调用T#()表示，并且子程序的剩余部分被视为表示控制部分。(在其他示例中，任务部分可以包括多个函数调用，且在最后一个函数返回之后表示任务完成)。结点子程序不包括任务部分，并且由此完全由控制部分组成。在这个示例中，单独的函数调用以它们被调用的顺序由分号隔开。

[0048]

节点	子程序
任务节点 T1	T1(); chain(J1)
结节点 J1	chain (T2); chain(T3)

[0049]

任务节点 T2	T2(); chain(J2)
任务节点 T3	T3(); chain(J2)
结节点 J2	chain(T4)
任务节点 T4	T4()

[0050] 表1

[0051] 在任务说明104被编译之后，计算系统100将生成的子程序加载至执行环境108的存储器系统110中。当调用特定子程序时，程序计数器将被设置为在存储有子程序的存储器系统110的一部分地址空间起点处的对应地址。

[0052] 在被调度的时刻，或响应于用户输入或预定事件，计算系统100开始执行表示控制流图的根的至少一个所加载子程序。例如，对于控制流图200，计算系统100生成用于执行任务节点T1的任务子程序的进程。当子程序开始执行时，该进程将首先调用用于执行任务节点T1的任务的任务部分，然后在任务部分返回之后(表明任务节点T1的任务已经完成)，进程将调用子程序的控制部分中的chain函数。chain函数所使用的用以确定是否生成用于执行特定节点的子程序的新进程的状态信息是这样的信息，该信息捕捉(capture)使用该特定节点作为参数调用的之前chain函数的历史。在下文进行更详细地的描述。

[0053] 这个历史信息可以被保持在与不同节点相关联的激活计数器中。计数器的值可以被存储在，例如，一部分存储器(memory)系统110中或者存储在其他工作贮存器(storage)中。在生成第一进程之前，每个节点的激活计数器的值被初始化为到该节点的输入链路的数目。因此，对于控制流图200，具有被初始化为以下值的六个激活计数器。

[0054]

节点	激活计数器值
任务节点T1	0
结节点J1	1

任务节点T2	1
任务节点T3	1
结节点J2	2
任务节点T4	1

[0055] 表2

[0056] 由于任务节点T1不具有任何输入链路，其激活计数器被初始化为零。可替代地，对于不具有任何输入链路的初始节点，无需具有与该节点相关联的激活计数器。通过输入链路连接的不同节点的控制部分将减小下游链接节点的激活计数器并且将基于减小的值来确定动作。在一些实施方式中，在减小操作(例如，原子“减小-和-测试”操作)之前或之后，访问计数器的函数可以利用原子地减小计数器且读取计数器的值的原子操作(atomic operation)。在一些系统中，由系统的本地指令来支持这种操作。可替代地，取代将计数器的值减小至零，计数器可以在零处开始，并且函数可以增大计数器的值直到达到被初始化为到节点的输入链路的数目的预定阈值(例如，使用原子“增大-和-测试”操作)。

[0057] 对chain函数“chain(N)”的调用减小了节点N的激活计数器，并且如果减小后的值为零，则chain函数通过新生成的进程来触发节点N的子程序的执行，然后返回。如果减小后的值大于零，则chain函数简单地返回而不触发新子程序的执行或生成新进程。如对于表1的结节点J1的结点子程序一样，子程序的控制部分可以包括对chain函数的多个调用。在控制部分中最后一个函数返回之后，执行子程序的进程可以退出，或者对于一些函数调用(例如，对于下文描述的“chainTo(链至)”函数)，该进程继续执行另一个子程序。新进程的有条件生成使得能够根据期望的部分排序来(潜在同时地)执行任务子程序，而不需要切换至调度器进程或从调度器切换以管理新进程的生成。

[0058] 对于表1的子程序，在任务子程序T1的任务部分之后对chain函数“chain(J1)”的调用返回节点J1的激活计数器中从1减小至0的结果，使得执行结点子程序，其包括对chain函数“chain(T2)”和“chain(T3)”的调用。这些调用中的每个导致节点T2和T3的相应激活计数器从1减小到0，从而能够执行节点T2和T3的任务子程序。两个任务子程序都包括调用“chain(J2)”的控制部分，其减小了节点J2的激活计数器。不论节点T2和T3的任务体中哪一个首先结束，都将导致调用将节点J2的激活计数器从2减小到1的chain函数。其次结束的任务部分将调用使节点J2的激活计数器从1减小到0的chain函数。由此，只有要完成的最后一个任务将导致执行节点J2的结点子程序，其导致对chain函数“chain(T4)”的最后一次调用并且将节点T4的激活计数器从1减小到0，这初始化了节点T4的任务子程序的执行。在节点T4的任务部分返回之后，因为节点T4的任务子程序不存在控制部分，所以控制流完成。

[0059] 在表1的子程序的示例中，为控制流图200中每个节点的子程序生成新进程。尽管具有控制部分的每个进程(其中控制部分自身确定是否生成新进程而不需要中央任务监控或调度进程)的子程序获得了一些效率，通过对控制部分进行一定的编译器优化可以得到更高的效率。例如，在一种编译器优化中，如果存在对第一子程序的控制部分中chain函数的单次调用，则在执行第一子程序的相同进程中可以执行(当激活计数器达到零时)下一个子程序(即，该chain函数的参数)-无需生成新进程。实现上述的一个方式是对于节点的最后一个输出链路，编译器明确地生成不同的函数调用(例如，“chainTo”函数而不是“chain”函数)。除了取代当激活计数器是零时生成新进程来执行其参数的子程序之外，chainTo函

数与chain函数类似，chainTo函数使得相同进程来执行其参数的子程序。如果节点具有单个输出链路，则其编译后的子程序将具有单次调用chainTo函数的控制部分。如果节点具有多个输出链路，则其编译后的子程序将具有一次或多次调用chain函数且单次调用chainTo函数的控制部分。这降低了在独立进程中生成的子程序的数目和它们相关的启动开销。表3示出使用这个编译器优化，对于控制流图200将产生的子程序的示例。

[0060]

节点	子程序
任务节点T1	T1();chainTo(J1)
结节点J1	chain(T2);chainTo(T3)
任务节点T2	T2();chainTo(J2)
任务节点T3	T3();chainTo(J2)
结节点J2	chainTo(T4)
任务节点T4	T4()

[0061] 表3

[0062] 在表3的子程序的示例中，第一进程执行节点T1和J1的子程序，然后生成新进程来执行节点T2的子程序，同时第一进程继续执行节点T3的子程序。不论这两个进程中的哪一个首先从它们相应任务部分返回，则其首先减小(从2到1)结节点J2的激活计数器，然后退出。从其任务部分返回的第二进程将结节点J2的激活计数器从1减小到0，然后通过执行结节点J2的子程序继续，其是函数调用“chainTo(T4)”，并且最后为任务节点T4的子程序。图2B示出在节点T3的任务在节点T2的任务之前结束的情况下，当第一和第二进程执行控制流图200中不同节点的子程序时，第一和第二进程的生命周期的示例。沿着表示进程的线的点对应于不同节点(由虚线连接至点)子程序的执行。点之间线段的长度不一定与流逝的时间成比例，而只是旨在示出执行不同子程序和生成新进程的相对顺序。

[0063] 可以潜在地提高效率的另一修改示例是延迟生成新进程，直到满足阈值，这表明特定子程序可能受益于并发性。如果每个子程序都需要花费大量时间来完成，则由不同进程来同时执行多个子程序是尤其有利的。另外，如果子程序中的任意子程序与其他子程序性相比花费相对少量的时间来完成，则可以利用另一个子程序来连续地执行子程序，而不浪费大量效率。上述延迟生成机制允许由同时运行进程来执行花费大量时间且可以一起执行的多个任务，而且尝试阻止为较短任务生成新进程。

[0064] 在使用延迟生成的chain函数的可替代实施方式中，chain函数(像chainTo函数)使得由相同的进程来开始执行其参数的子程序。但是，与chainTo函数不同，计时器跟踪它花费在执行子程序上的时间，并且如果超过阈值时间，则chain函数生成新进程来继续执行子程序。第一进程可以继续，就像子程序已经完成一样，并且第二进程可以在第一进程中断处接管的子程序的执行。可以被用来完成上述的一个机制是使得第二进程继承第一进程的子程序栈框架(stack frame)。被执行的子程序的栈框架包括指向特定指令的程序计数器、以及包含与子程序执行相关的其他值(例如，局部变量和寄存器值)。在这个示例中，T2的任务子程序的栈框架将包括返回指针，返回指针使得在完成T2的任务子程序之后，进程返回至J1的结点子程序。当延迟生成计时器溢出时，生成新进程且将其与T2的任务子程序的当前栈框架相关联，并且第一进程立即返回至J1的结点子程序(以调用“chainTo(T3)”)。因为在完成T2的任务子程序之后新进程不需要返回至J1的结点子程序，所以在继承的栈框架中

的返回指针被移除(即使其变为null(空))。所以,当任务是快(相对于可配置阈值)的情况,延迟生成使得能够执行后续任务的子程序而没有生成新进程的开销,并且对于通过继承现有栈框架而使得任务更长的情况,延迟生成降低生成新进程的开销。

[0065] 图2C示出在节点T2的任务比延迟生成阈值长的情况下,当第一和第二进程执行控制流图200中不同节点的子程序时,第一和第二进程的生命周期的示例。当达到生成阈值时,进程1生成进程2,其继承执行节点T2的任务的子程序的栈框架且继续执行子程序。在这个示例中,在节点T2的任务(由进程1开始且由进程2完成)结束之前,节点T3的任务(由进程1执行)结束。所以,在这个示例中,进程1将J2的激活计数器从2减小到1(并且然后退出),且进程2将J2的激活计数器从1减小到0,导致进程2执行任务节点T4的任务。在这个示例中,在确定这种并发性将有益于整体效率之后,允许同时执行节点T2的任务和节点T3的任务。

[0066] 图2D示出在节点T2的任务比延迟生成阈值短的情况下,当单个进程执行控制流图200中不同节点的子程序时,单个进程的生命周期的示例。在这个示例中,在节点T3的任务(由进程1执行)结束之前,节点T2的任务(也由进程1执行)结束。所以,在这个示例中,进程1在完成节点T2的任务之后将J2的激活计数器从2减小到1,并且进程1在完成节点T3的任务之后将J2的激活计数器从1减小到0,使得进程1执行任务节点T4的任务。在这个示例中,在确定可以快速完成节点T2的任务之后,牺牲节点T2和节点T3任务的同时执行,避免生成第二进程,从而获得效率。

[0067] 可以包含在控制流图中的另一类节点是条件节点,由图3示出的控制流图300中的圆圈表示。条件节点限定用于确定是否执行任务节点(该任务节点被连接至条件节点的输出)的任务的条件。如果在运行期间所限定的条件为真,则控制流跳过该条件节点继续向下游进行,但是如果在运行期间所限定的条件为假,则控制流不会跳过该条件节点继续进行。如果条件为假,且如果控制流图存在到条件节点下游那些任务节点(且它们自身没有被其它假条件节点阻挡)的其他路径,则只执行条件节点下游那些任务节点的任务。

[0068] 编译器为每个条件节点产生“条件子程序”,且还使用由条件节点限定的条件来修改条件节点下游的其他一些节点的子程序。例如,编译器可以为将在运行期间应用的“跳过机制”生成指令以遵循由控制流图限定的控制流。在跳过机制中,每个节点都有相关联的“跳过标志”,其控制是否执行相应的任务部分(如果有的话)。如果设置了跳过标志,则任务部分的执行被抑制(节点处于“抑制”状态),并且可以通过适当的控制代码(其由编译器放置在控制部分中)将这个抑制传播至其他任务。在前面的示例中,任务子程序的任务部分先于控制部分。在以下示例中,一些任务子程序的控制部分包括在任务部分之前出现的控制代码(也称为“序言(prologue)”)以及在任务部分之后出现的控制代码(也称为“后记(epilogue)”)。例如,为了实现这个跳过机制,在序言中(即,在任务部分之前执行的代码),编译器包括条件指令(例如if(如果)语句),以及对使用识别下游节点的参数来调用的“跳过函数”的调用。在后记中(即,在任务部分之后执行的代码),编译器包括对chain或chainTo函数的调用。在一些情况下,可以只执行序言,并且由于由跳过标志的值表示的存储状态,任务部分和后记可以被跳过。表4示出为控制流图300生成子程序的示例。

[0069]

节点	子程序
任务节点 T1	T1(); chainTo(J1)
结节点 J1	chain(C1); chain(C2); chainTo(J3)

条件节点 C1	if (<条件 1>)
	chainTo(T2)
	else
	skip(T2)

条件节点 C2	if (<条件 2>)
	chainTo(T3)
	else
	skip(T3)

任务节点 T2	if (skip)
	skip(J2)
	else
	T2(); chainTo(J2)

任务节点 T3	if (skip) skip(J2)

[0070]

	else T3(); chainTo(J2)

结节点 J2	if (skip)
	skip(T4)
	else
	chainTo(T4)

任务节点 T4	if (skip)
	skip(J3)
	else
	T4(); chainTo(J3)

结节点 J3	if (skip)
	skip(T5)
	else
	chainTo(T5)

任务节点 T5	if (skip)
	return
	else
	T5()

[0071] 表4

[0072] 与chain和chainTo函数类似,跳过函数“skip(N)”减小其参数(节点N)的激活计数器,并且如果减小后的值为0,则执行相应的子程序。在这个示例中,通过继续使用相同的进程而不生成新的进程来遵循chainTo函数的行为,然而,编译器可以使用表现为chain和chainTo函数的两个版本跳过函数来以相似的方式控制任务生成。编译器为条件节点的下游节点生成子程序,使得如果设置了正在执行其子程序的节点的跳过标志(即,评估为布尔真值),则其对下游节点调用跳过而不调用任务部分,并且如果跳过标志被清除(即,评估为布尔假值),则其调用任务部分(如果该节点是任务节点)且对下游节点调用chain。可替代地,编译器可以默认包括子程序的控制部分中的条件语句,而无需确定哪个节点是条件节点的下游。特别地,对于每个节点的子程序,可以默认包括用以检查跳过标志的“if”语句,无需编译器确定是否包括该“if”语句(尽管这可能导致对跳过标志的不必要检查)。

[0073] 如果控制流图中存在条件节点,则具有多个输入的节点在运行期间获得基于节点类型的逻辑行为。具有多个输入链路和单个输出链路的结节点对应于逻辑“或”操作,使得如果由输出链路连接的输出节点使它的子程序为chain调用(且不是跳过调用)的参数,则由输入链路连接的至少一个输入节点必须使它的子程序执行chain调用(且不是跳过调

用)。具有多个输入链路和一个输出链路的任务节点对应于逻辑“与”操作,使得如果该任务节点的子程序为chain调用(且不是跳过调用)的参数,则由输入链路连接的所有输入节点必须使它们的子程序执行chain调用(且不是跳过调用)。

[0074] 为了确保这个逻辑行为,设置与节点相关的跳过标志,并且在运行期间根据预定规则将其清除。在执行控制流图中节点的任何子程序之前出现的初始化阶段,提供跳过标志的初始值,并且该初始值取决于节点的类型。编译器还使用不同版本的跳过函数和chain函数,其根据节点的类型具有不同的行为。如下示出用以改变节点N的跳过标志的预定规则组、以及编译器使用的不同版本函数的行为的一个示例。

[0075] • 对于多输入结节点(OR(或)操作):初始设置跳过标志,skip_OR(N)不改变跳过标志,chain_OR(N)清除跳过标志

[0076] • 对于多输入任务节点(与操作):初始清除跳过标志,skip_AND(N)设置跳过标志,chain_AND(N)不改变跳过标志

[0077] • 对于单输入节点:初始设置跳过标志,skip(N)不改变跳过标志,chain(N)清除跳过标志

[0078] 关于跳过标志,chainTo函数的行为与chain函数是相同的。对于单输入节点,或操作和与操作的行为是等价的,并且两者都可以被使用(诸如这个示例中或操作的行为)。对于此组规则,(多个)起始节点(即,不具有任何输入链路的节点)使得它们的跳过标志被清除(如果其初始值尚未被清除)。

[0079] 对于控制流图300,考虑节点C1的条件为真、节点C2的条件为假、并且在节点C2条件检查完成之前节点T3的任务结束的情况:节点T3的子程序将遵循chain逻辑(与跳过逻辑相反),其清除节点J2的跳过标志,并且减小节点J2的激活计数器(从2到1);然后节点T4的子程序遵循跳过逻辑(其不改变跳过标志),并且减小节点J2的激活计数器(从1到0),由于节点J2的跳过标志被节点T3的子程序清除了,所以这导致chain(T5)。

[0080] 其他规则也是可能的。如下示出用以改变节点N的跳过标志的预定规则组、以及编译器使用的不同版本函数的行为的另一个示例。

[0081] • 对于结节点:初始设置跳过标志,skip_J(N)不改变跳过标志,chain_J(N)清除跳过标志

[0082] • 对于任务节点或条件节点:初始清除跳过标志,skip(N)设置跳过标志,chain(N)不改变跳过标志

[0083] 对于此组规则,(多个)起始节点(即,不具有任何输入链路的节点)也将使得它们的跳过标志被清除(如果其初始值尚未被清除)。

[0084] 编译器可以基于控制流图的分析可选地执行子程序的控制部分中条件语句或其他指令的多种优化。例如,根据控制流图300,可以确定不管条件节点C1和C2的条件是真还是假,因为在结节点J1与结节点J3之间存在将最终导致执行任务节点T5的任务的链路,所以任务节点T5的任务不会被跳过。所以,编译器能够为任务节点T5生成子程序,其避免了检查其跳过标志,从而简单地调用其任务部分T5()。编译器还可以进行其他优化,例如,在条件节点之后跳过控制流图的整个部分的情况下,只要在适当地处理被跳过的部分之后有任何其他输入了下游节点,就省去中间跳过标志检查和跳过函数调用(即,多次减小下游节点的计数器,该次数为在包含对被跳过部分的中间调用情况下该下游节点的计数器将被减小

的次数)。

[0085] 图4示出包括多输入任务节点T3的简单控制流图400,具有来自分别沿袭条件节点(分别为C1和C2)的任务节点T1和T2的输入链路。在这个示例中,任务节点T3对应于逻辑与操作,使得当节点T3的任务被执行时,节点T1和T2的任务也必须被执行(而不是跳过)。表5示出为控制流图400生成的子程序的示例。

[0086]

节点	子程序
结节点 J1	chain(C1); chainTo(C2)

条件节点 C1	if (<条件 1>)
	chainTo(T1)
	else
	skip(T1)

条件节点 C2	if (<条件 2>)
	chainTo(T2)
	else
	skip(T2)

任务节点 T1	if (skip)
	skip(T3)
	else
	T1(); chainTo(T3)

任务节点 T2	if (skip)
	skip(T3)
	else
	T2(); chainTo(T3)

任务节点 T3	if (skip)
	return
	else
	T3()

[0087]

[0088] 表5

[0089] 在一些实施方式中,结节点(或其它节点)可以被配置为根据到节点的输入的特性来提供各种逻辑操作。例如,当所有的输入被指定为“必需”输入时,节点可以被配置为提供逻辑与操作,以及当所有的输入被指定为“可选”输入时,节点可以被配置为提供逻辑或操

作。如果一些输入被指定为“必需”输入且一些输入被指定为“可选”输入，则可以使用一组预定规则来说明将由节点执行的逻辑操作的组合。

[0090] 上述任务控制技术可以使用执行适当软件的计算系统来实施。例如，软件包括在一个或多个已编程或可编程计算机系统(可以具有各种架构，诸如分布式、客户端/服务器、或网格式)上执行的一个或多个计算机程序中的程序，每个计算机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)、以及至少一个用户接口(用于使用至少一个输入设备或端口来接收输入，以及用于使用至少一个输出设备或端口来提供输出)。该软件可以包括大型程序的一个或多个模块，例如，该大型程序提供与数据流图的设计、配置和执行相关的服务。该程序的模块(例如，数据流图的元件)可以被实施为数据结构或者符合在数据库中存储的数据模型的其它经过组织的数据。

[0091] 该软件可以被提供在诸如CD-ROM或其他计算机可读介质之类的有形永久存储介质(例如，其可以被通用或专用计算机系统或设备读取)上，或者通过网络的通信介质递送(例如，被编码成传播信号)到执行该软件的计算机系统的有形永久介质。一些或全部处理可以在专用计算机上执行，或者使用诸如协处理器或现场可编程门阵列(FPGA)或专用集成电路(ASIC)之类的专用硬件来执行。该处理可以以分布方式实施，在该分布方式中，由该软件指定的不同的计算部分由不同的计算元件执行。每个这样的计算机程序被优选地存储在或下载到可由通用或专用可编程计算机读取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、或者磁或光介质)，用于在计算机读取该存储介质或设备时配置和操作该计算机，以执行此处所描述的处理。也可以考虑将本发明的系统实施为有形永久存储介质，其配置有计算机程序，其中，如此配置的存储介质使得计算机以特定和预定义的方式操作以执行此处所描述的一个或多个处理步骤。

[0092] 已经对本发明的多个实施例进行了描述。然而，应当理解，前面的描述旨在说明而非限制本发明的范围，本发明的范围由以下权利要求书的范围来限定。因此，其它实施例也落在以下权利要求书的范围内。例如，在不脱离本发明的范围的情况下可进行各种修改。此外，上述的一些步骤可以是顺序独立的，因此可以以不同于所述的顺序来执行。

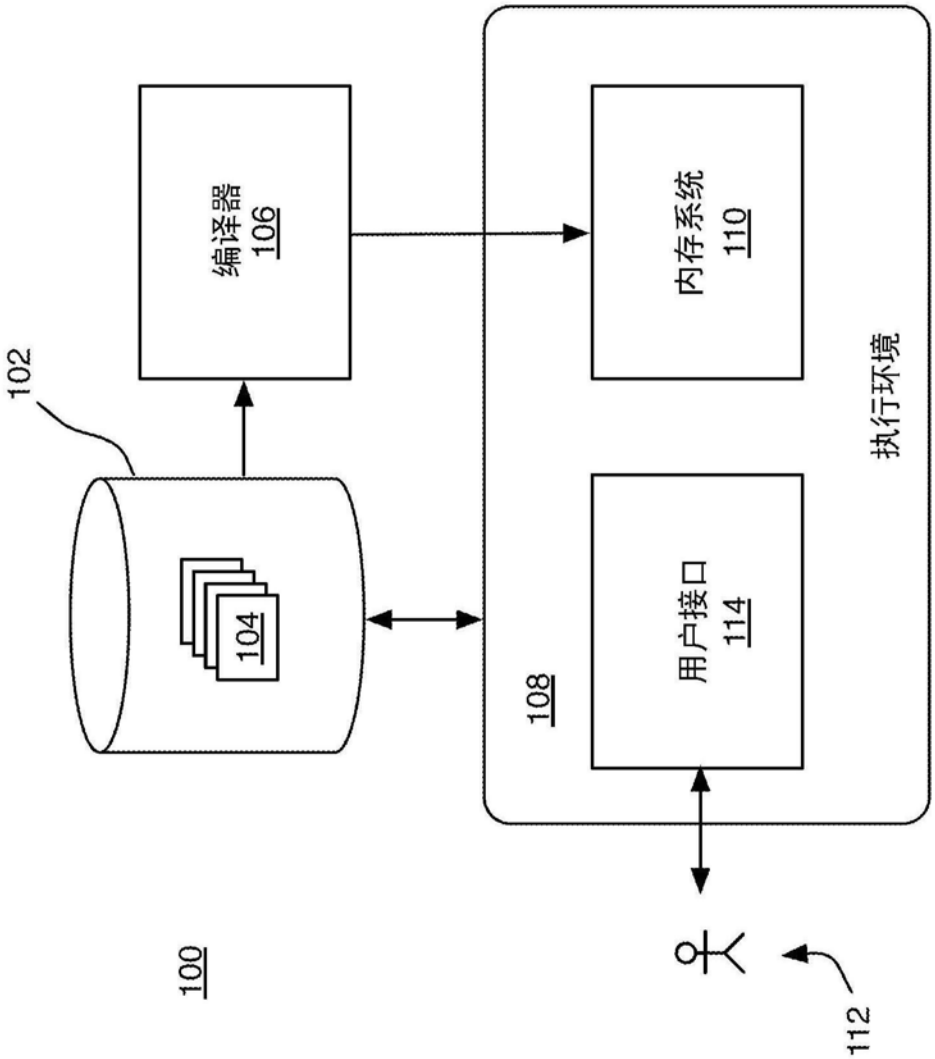


图1

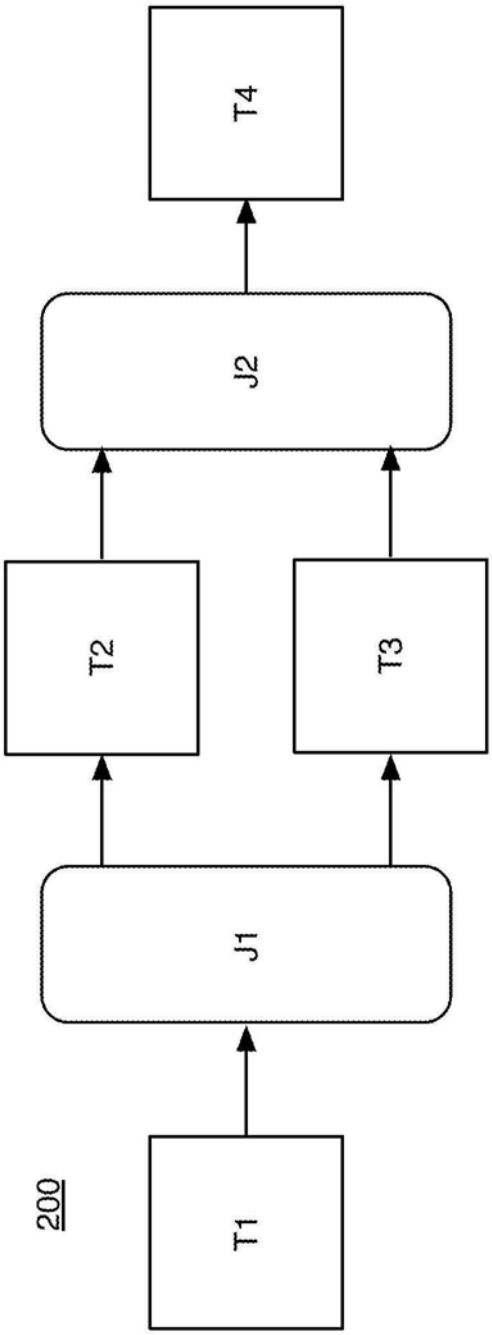


图2A

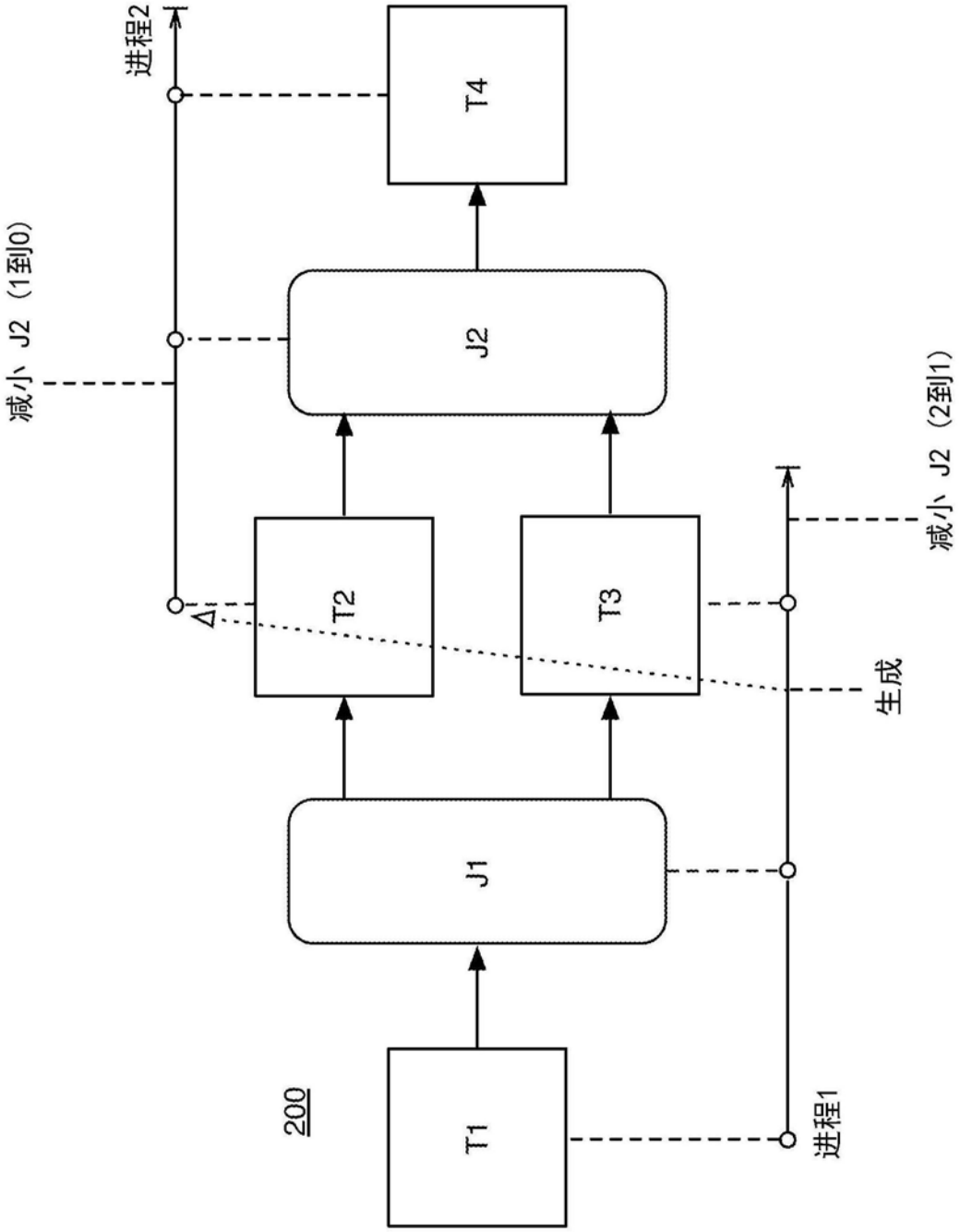


图2B

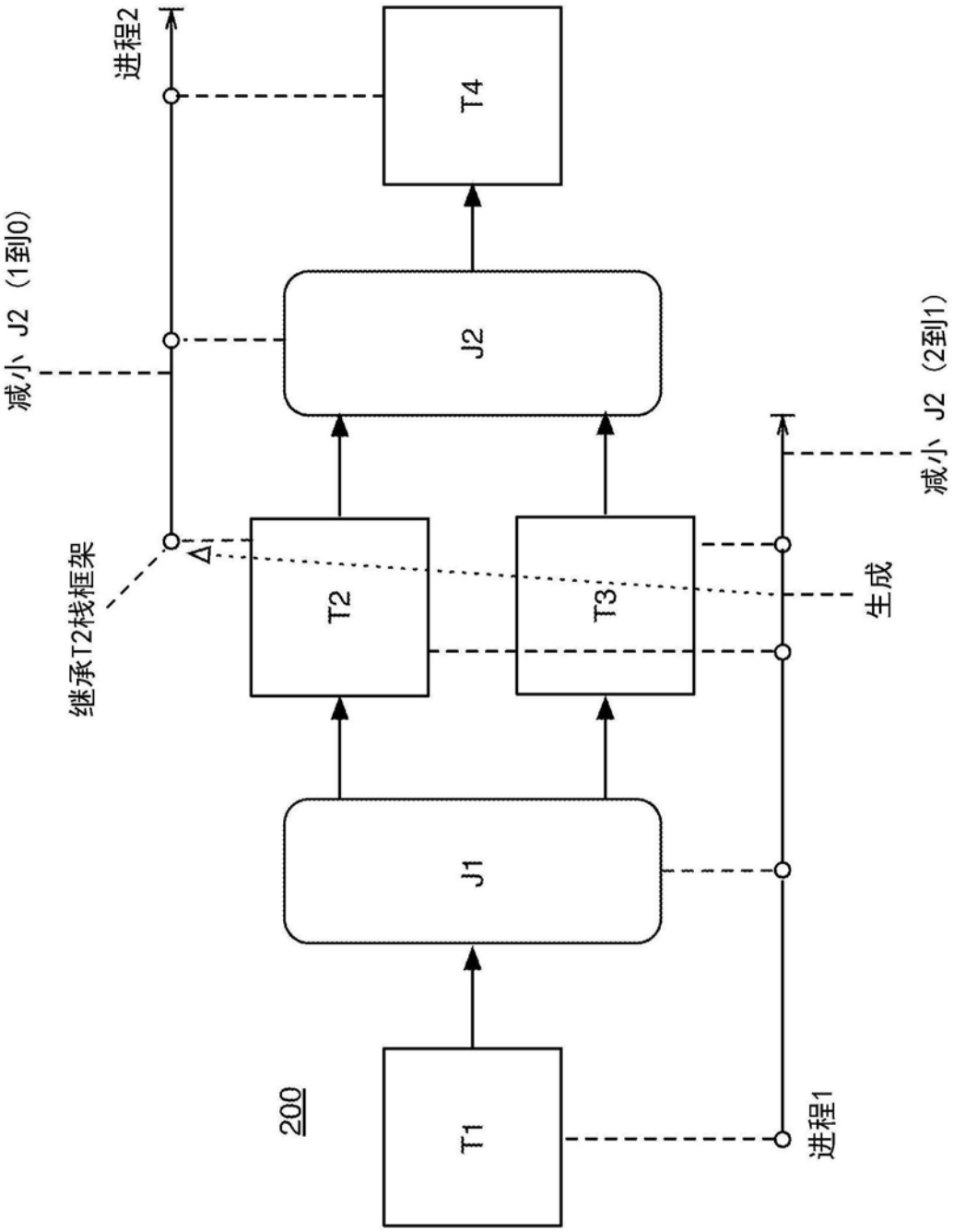


图2C

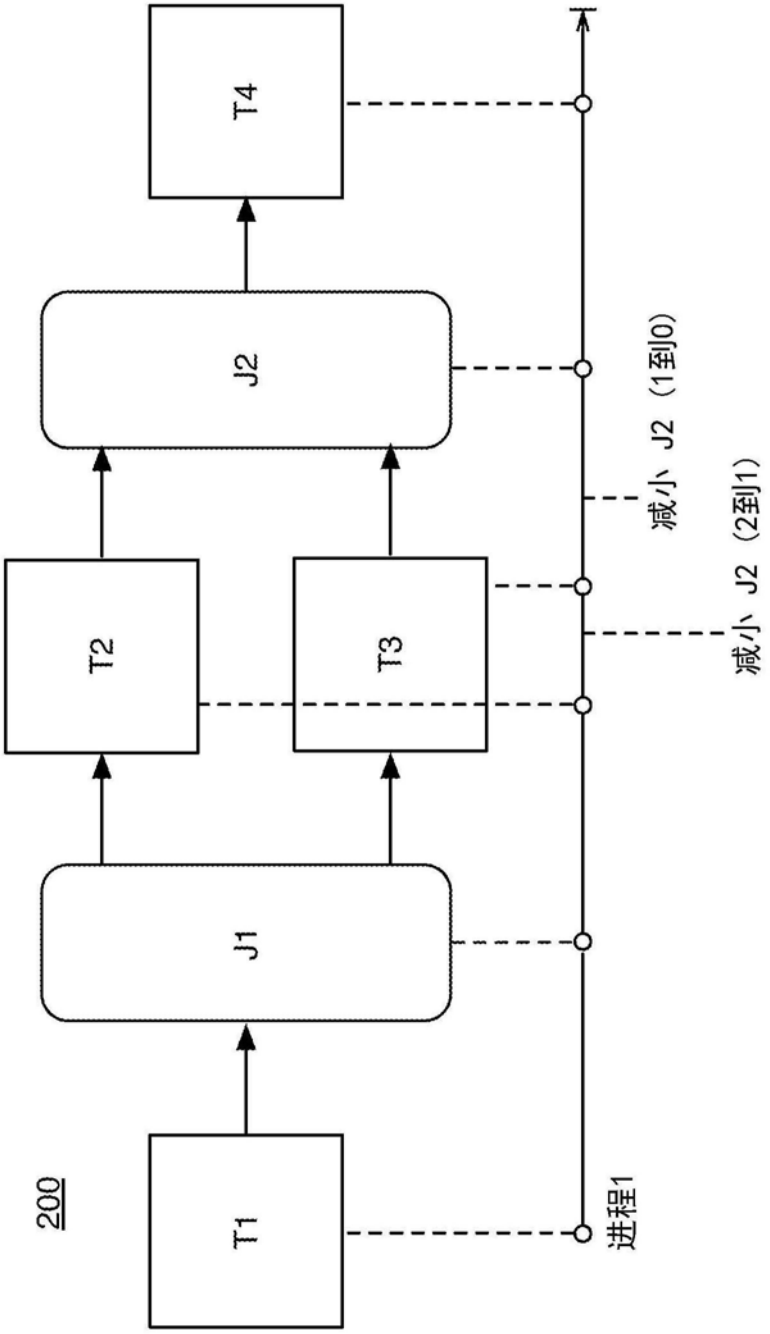


图2D

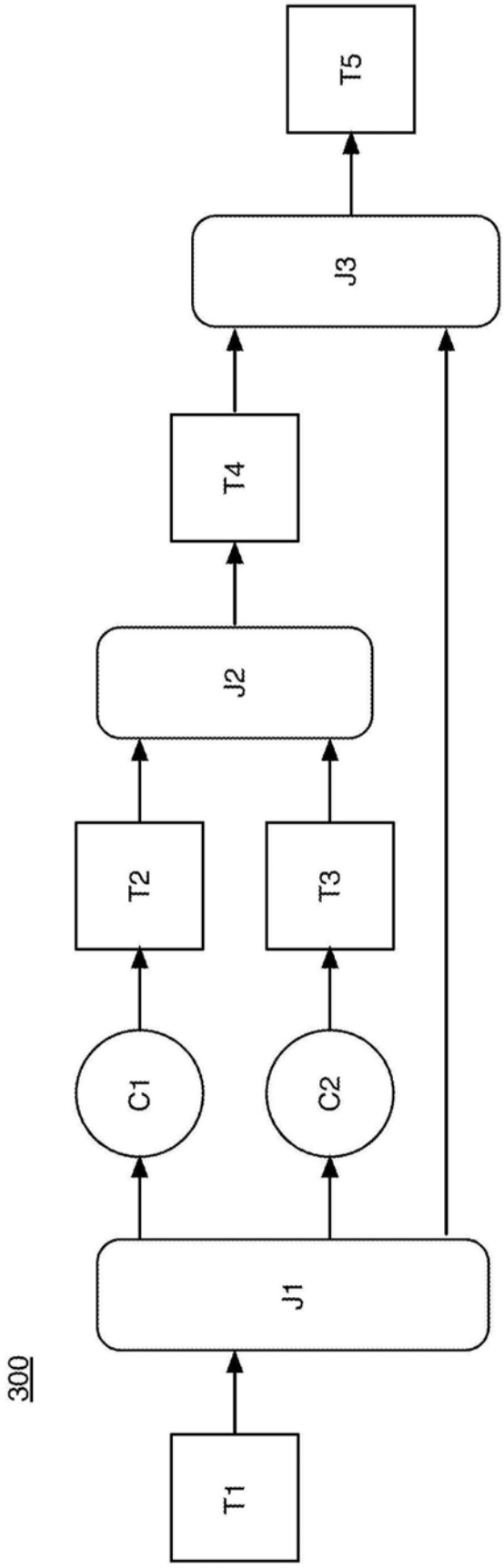


图3

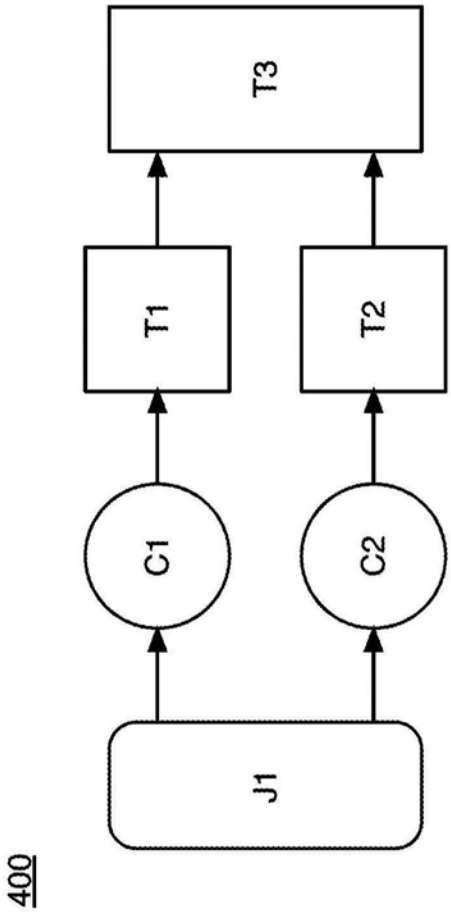


图4