



(12)发明专利申请

(10)申请公布号 CN 110231978 A

(43)申请公布日 2019. 09. 13

(21)申请号 201910123211.7

(22)申请日 2019.02.19

(30)优先权数据

1851375 2018.02.19 FR

(71)申请人 布尔简易股份公司

地址 法国博伊斯

(72)发明人 昂盖朗·珀蒂 西里尔·马佐里克

邦雅曼·帕若 格扎维埃·维古鲁

(74)专利代理机构 成都超凡明远知识产权代理

有限公司 51258

代理人 魏彦 洪玉姬

(51)Int.Cl.

G06F 9/46(2006.01)

G06F 9/54(2006.01)

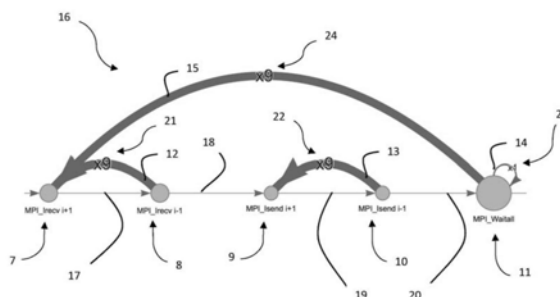
权利要求书2页 说明书5页 附图2页

(54)发明名称

用于分析在若干进程之间进行通信函数调用的应用程序的性能的计算机化方法和系统

(57)摘要

本发明涉及用于对在若干进程之间进行通信函数调用的应用程序的性能进行分析的计算机化方法和系统。该方法包括：收集与函数调用、形成模式(7至11)的调用或至少两个调用的系列相关的信息的步骤，以及对所收集的信息进行压缩的步骤。压缩步骤包括：-基于所收集的信息对模式(7至11)和系列模式(7至11)进行检测以形成通过唯一标识符标识每个模式(7至11)和每个系列模式(7至11)的语法的步骤，-在语法中检测模式(7至11)和系列模式(7至11)的重复的步骤，-生成其中存储有模式(7至11)和系列模式(7至11)以及它们各自的重复次数(12至15)的压缩跟踪文件的步骤。



1. 一种对应用程序的性能进行分析的计算机化方法,所述应用程序在若干进程之间进行通信函数的调用(1至6),一个调用(1至6)或至少两个调用(1至6)的系列形成模式(7至11),所述方法包括:

- 收集与所述函数调用(1至6)相关的信息,以及
- 将所收集的信息压缩,所述压缩包括:
 - 基于所收集的信息对模式(7至11)和系列模式(7至11)进行检测,以形成其中每个模式(7至11)和每个系列模式(7至11)均通过唯一的标识符被标识的语法,
 - 在所述语法中检测模式(7至11)和系列模式(7至11)的重复,
 - 生成压缩跟踪文件,所述压缩跟踪文件中存储有所述模式(7至11)和所述系列模式(7至11)以及它们各自的重复次数(12至15)。

2. 根据权利要求1所述的方法,包括:

基于所述压缩跟踪文件生成图形(16),所述图形的节点(7至11)表示所述模式(7至11),所述模式按照形成所述模式(7至11)的所述函数调用(1至6)的时序排序。

3. 根据权利要求2所述的方法,其中,所述图形(16)包括第一类边,所述第一类边中的每个边(17至20)按照形成两个连续模式(7至11)中的每个的所述函数调用(1至6)的时序顺序连接与所述两个连续模式(7至11)对应的两个节点(7至11)。

4. 根据权利要求2和3中任一项所述的方法,其中,所述图形(16)包括第二类边,所述第二类边中的每个边(12至15)连接分别与在所述跟踪文件中重复的系列模式(7至11)的起点和终点对应的两个节点(7至11),或者在所述跟踪文件中重复的节点(11)上形成环(14),所述第二类边中的每个边(12至15)均被分配有与对应系列模式(7至11)或对应模式(11)的重复次数(21至24)对应的权重(21至24)。

5. 根据前述权利要求中任一项所述的方法,其中,对模式(7至11)和系列模式(7至11)进行检测包括:在所收集的信息中为通信函数的每个调用(1至6)确定所调用的函数、接收进程、发送进程以及所述调用的时间。

6. 根据权利要求5所述的方法,其中,所述确定在所述应用程序被执行时进行。

7. 根据权利要求6所述的方法,其中,所述语法存储在存储器中,并且其中,一旦所述应用程序的执行被终止,就通过遍历所述语法来进行对所述模式(7至11)和系列模式(7至11)的重复进行检测的步骤。

8. 根据权利要求5所述的方法,其中,收集信息包括生成跟踪文件,对于通信函数的每个调用(1至6),所述跟踪文件中存储有所述调用的时间、所述所调用的函数、所述接收进程、所述发送进程,并按照所述调用的时间、所述所调用的函数、所述接收进程、所述发送进程的顺序存储,并且其中,一旦所述应用程序的执行被终止,就在所述跟踪文件中进行对每个调用(1至6)的所述所调用的函数、所述接收进程、所述发送进程和所述调用的时间的确定。

9. 根据前述权利要求中任一项所述的方法,其中,每个模式(7至11)通过该模式(7至11)中调用的第一函数的名称标识,并且每个系列模式(7至11)通过该系列模式(7至11)中的第一模式(7至11)中调用的第一函数的名称标识。

10. 一种用于对在若干进程之间进行通信函数调用(1至6)的应用程序的性能进行分析的计算机化系统,所述系统包括处理器和存储器,所述存储器包括程序指令,当所述程序指

令被所述处理器执行时,所述程序指令使所述系统执行根据前述权利要求中任一项所述的方法的步骤。

用于分析在若干进程之间进行通信函数调用的应用程序的性能的计算机化方法和系统

技术领域

[0001] 本发明涉及一种用于对在若干进程之间进行通信函数调用的应用程序的性能进行分析的计算机化方法和系统。其在并行和分布式计算环境中例如,在按照“消息传递接口”或MPI标准在进程间实现通信的情况下,特别地在性能的分析方面以及计算机化应用程序中的进程间通信的规划方面找到应用。

背景技术

[0002] 对并行和分布式计算环境中的应用程序的性能进行分析是个复杂的任务。人们对通过MPI标准进行的进程间通信特别感兴趣。以函数库的形式实现的MPI标准允许通过插入在应用程序与MPI库之间的特定库来与该应用程序对接。

[0003] 这使得可以收集诸如下述信息:

[0004] -每进程的时间以及总体通信时间,

[0005] -发送和接收的消息的数量以及消息的大小,

[0006] -各MPI函数调用,

[0007] -每MPI函数调用的持续时间,

[0008] -该调用发生的时间。

[0009] 这些各项信息对于获得每个进程的全局和局部统计以及对于跟踪函数调用的时间序列(通常称为时间线)都是有用的。然而,要生成、解译和使用该时间序列是复杂的。

发明内容

[0010] 因此,本发明的目标之一是解决特别是上述问题。因此,本发明的目的是特别地提出一种便于对在并行和分布式计算环境中使用进程间通信的应用程序的性能进行分析的方法和系统。

[0011] 因此,根据第一方面,本发明的主题是对应用程序的性能进行分析的计算机化方法,所述应用程序在若干进程之间进行通信函数调用,一个调用或至少两个调用的系列形成模式。

[0012] 该方法包括收集与函数调用相关的信息的步骤,以及将所收集的信息压缩的步骤。

[0013] 压缩步骤包括基于所收集的信息对模式和系列模式进行检测以形成其中每个模式和每个系列模式均通过唯一的标识符标识的语法的步骤。

[0014] 压缩步骤还包括在语法中检测模式和系列模式的重复的步骤。

[0015] 该压缩步骤还包括生成其中存储有模式和系列模式以及它们各自的重复次数的压缩跟踪文件的步骤。

[0016] 根据某些实现的模式,该方法还包括以下特征中的一个或多个,孤立地或根据所有技术上可能的组合来看待这些特征:

[0017] -方法包括基于压缩跟踪文件生成图形,该图形的节点表示模式,该模式按照形成所述模式的函数调用的时序排序。

[0018] -图形包括第一类边,所述第一类边中的每个边按照形成两个连续模式中的每个的函数调用的时序顺序连接与这两个连续模式对应的两个节点;

[0019] -图形包括第二类边,所述第二类边中的每个边连接分别与在跟踪文件中重复的系列模式的起点和终点对应的两个节点,或者在跟踪文件中重复的节点上形成环,所述第二类边中的每个边均被分配有与对应系列模式或对应模式的重复次数对应的权重;

[0020] -对模式和系列模式进行检测的步骤包括:根据所收集的信息为通信函数的每个调用确定所调用的函数、接收进程、发送进程以及该调用的时间;

[0021] -该确定在应用程序被执行期间进行;

[0022] -语法存储在存储器中,并且一旦应用程序的执行被终止就通过遍历语法来进行对模式和系列模式的重复进行检测的步骤;

[0023] -收集信息的步骤包括生成跟踪文件,对于通信函数的每个调用,该跟踪文件中存储有调用的时间、所调用的函数、接收进程、发送进程,并按照该调用的时间、所调用的函数、接收进程、发送进程的顺序存储,并且一旦应用程序的执行被终止,就在跟踪文件中进行对所调用的函数、接收进程、发送进程和调用的时间的确定;

[0024] -每个模式通过该模式中调用的第一函数的名称标识,并且每个系列模式通过该系列模式中的第一模式中调用的第一函数的名称标识;

[0025] -函数调用例如按照“消息传递接口”或MPI类型或“开放式多重处理”或OpenMP类型的协议进行。

[0026] 根据第二方面,本发明还涉及一种用于对在若干进程之间进行通信函数调用的应用程序的性能进行分析的计算机化系统,该系统包括处理器和存储器,该存储器包括程序指令,当该程序指令被处理器执行时,该程序指令使系统实现如上所述的方法。

[0027] 因此,通过检测进程间通信函数调用循环即重复的一系列分层调用,本发明的方法和系统使得可以容易地生成时间线并简化该时间线。

[0028] 为此,在应用程序被执行时对函数调用进行分析,例如进行即时分析,或者通过使用记录调用的跟踪文件进行分析。

[0029] 这些调用形成复杂模式,在不丢失信息的情况下对该复杂模式进行检测和记录。在每个进程上也在进程之间进行这种模式检测,从而使得可以在单个图形中更加直观清晰地以图形方式表示所有通信方案。

附图说明

[0030] 在参考以下附图阅读后面的仅以示例的方式给出的非限制性描述后,本发明的特征和优点将变得明显:

[0031] -图1:按照MPI标准进行的进程间通信函数调用的示例应用程序的算法表示;

[0032] -图2:在执行图1的应用程序期间生成的示例原始跟踪文件的表示;

[0033] -图3:根据本发明的方法生成的与图1的应用程序的执行对应的示例压缩跟踪文件的表示;

[0034] -图4:基于图1的应用程序根据本发明的方法生成的图形的示意表示。

具体实施方式

[0035] 在随后的描述中,在使用MPI标准的背景下呈现了本发明的方法的示例实现。

[0036] 在并行和分布式程序或应用程序中,通常在每个时间步长在进程与其相邻进程(距离等于1)之间进行通信。

[0037] 图1示出了以算法形式表示的示例应用程序。在该实施方式中,与MPI函数调用对应的数个操作重复N次。更精确地,这涉及向距离等于1的进程发送并从其接收的操作(发送进程、相应地接收进程的序号(rank,排名)分别加1或减1)。

[0038] 在该实施方式中,进行并重复六次通信函数调用1至6,即:

[0039] -对接收函数MPI_Irecv的调用1,以用于通过当前序号的进程从当前序号加一(距离等于一)的进程接收信息项,

[0040] -对接收函数MPI_Irecv的调用2,以用于通过当前序号的进程从当前序号减一(距离等于一)的进程接收信息项,

[0041] -对发送函数MPI_Isend的调用3,以用于通过当前序号的进程向当前序号加一(距离等于一)的进程发送信息项,

[0042] -对发送函数MPI_Isend的调用4,以用于通过当前序号的进程向当前序号减一(距离等于一)的进程发送信息项,

[0043] -对函数MPI_Waitall的两次调用5和6,以用于延时,直到进行中的通信均已终止。

[0044] 在该实施方式中,用于通信函数调用的句法是C语言MPI函数库的句法。

[0045] 因此,根据本发明的方法,对图1中以算法方式表示的应用程序的性能进行分析首先包括对模式进行检测,即,检测来自调用1至6中的调用,或者检测来自调用1至6中的至少两个调用的系列,并检测其可能的重复。

[0046] 因此,该方法包括检测模式和系列模式的重复的步骤,在实施方式中,这些模式在图3和图4中以附图标记7至11表示,后面将更详细地对其进行描述。

[0047] 被检测的每个模式7至11均通过唯一的标识符标识,该唯一的标识符可以例如包含在模式7至11中调用的第一函数的名称,并且每个被检测的系列模式均通过唯一的标识符标识,该唯一的标识符可以例如包含系列模式中的第一模式中调用的第一函数的名称。

[0048] 对模式7至11进行检测的该步骤使得可以形成语法,基于此,通过遍历该语法,又检测模式的重复。

[0049] 基于对调用1至6的分析,生成压缩跟踪文件,诸如图3所示,其中存储有模式7至11和系列模式7至11,及其各自的重复次数,重复次数用附图标记12至15表示。

[0050] 更准确地说,对于对模式7至11的重复和系列模式7至11的重复的检测,为通信函数的每个调用1至6确定所调用的函数、接收进程、发送进程以及该调用的时间。

[0051] 可以在应用程序的执行期间直接对这些调用1至6进行分析,例如,进行即时分析,即,通过调用一完成就对调用进行处理来进行分析。

[0052] 或者,可以使用记录这些调用1至6的跟踪文件,诸如图2所示。具体地,对于每个调用1至6,存储有调用的时间、所调用的函数、接收进程、发送进程,特别地按照调用的时间、所调用的函数、接收进程、发送进程的顺序存储。这样,随后就可以对该跟踪文件中的每个调用1至6进行分析。

[0053] 对于模式7至11及其重复的检测,可以使用用于数据压缩但不会丢失信息也不会

丢失序列性的算法类型,诸如改进的Sequitur型算法。

[0054] Sequitur是一种通过从一系列离散的符号中推出分层结构、通过替换由生成短语的语法规则检测到的重复短语并通过递归地继续从而允许对文本进行分层压缩的算法(参见:Nevill-Manning,C.G.and Witten,I.H.(1997) "Identifying Hierarchical Structure in Sequences:A linear-time algorithm,"Journal of Artificial Intelligence Research,7,67-82)。

[0055] 执行这种算法的结果可以以原始序列的分层表示的形式给出,其提供了关于其词汇结构的信息。

[0056] 然而,虽然Sequitur是基于简单的符号,诸如字母或数字,但是本发明的方法更为复杂,因为分层结构的对应于模式7至11的终端元素由单词、整数和实数构成。

[0057] 然而,针对本发明的方法的需要,仅限于一部分信息,即:函数(其名称)、调用方的顺序号、接收者、发送者、时间(例如由跟踪文件中的行号限定)。

[0058] 因此,重新集中于指令(或调用)的序列,同时通过跟踪文件的特有结构保存发送者、接收者、调用方的概念以及时间性的概念,因为调用按时间顺序排列。

[0059] 因此,对跟踪文件中或在执行应用程序期间得到的与函数调用对应的字符串进行分析,以形成图3所示的压缩跟踪文件,包括与模式7至11对应的终端符号7至11,以及表示一个或多个终端符号或模式7至11的重复12至15的环12至15。

[0060] 压缩跟踪文件还包括对各模式7至11之间的时间顺序链接17至20的定义。

[0061] 该压缩跟踪使得可以生成使将模式7至11和各个重复12至15串连起来形象化的图形。

[0062] 因此,基于与图1所示应用程序的执行对应的压缩跟踪,生成图4的图形16。

[0063] 图形16包含表示模式7至11的节点7至11,这些模式按照形成这些模式7至11的函数调用1至6的时间顺序排序。

[0064] 此外,图形16包括将节点7至11中的某些节点连接在一起的两类边12至20。

[0065] 在第一类边中,每个边17至20按照形成两个连续模式7至11中的每个的函数调用1至6的时间顺序连接与这两个连续模式7至11对应的两个节点7至11。

[0066] 因此:

[0067] -边17沿着节点8的方向连接节点7和8,

[0068] -边18沿着节点9的方向连接节点8和9,

[0069] -边19沿着节点10的方向连接节点9和10,

[0070] -边20沿着节点11的方向连接节点10和11。

[0071] 在第二类边中,每个边12至15要么连接分别与在应用程序的执行期间和/或在跟踪文件中重复的系列模式7至11的起点和终点对应的两个节点7至11,要么在应用程序的执行期间和/或在跟踪文件中重复的节点11上形成环14。

[0072] 该第二类边中的每个边12至15均被分配有与对应系列模式7至11或对应模式11的重复次数21至24对应的权重21至24。

[0073] 在图2所示的图形16的实施方式中,图1中所示的应用程序参数N等于9。

[0074] 因此:

[0075] -边12沿着节点7的方向连接节点7和8,并被分配有等于9的权重21,以显示模式7

和8中函数的连续调用重复9次，

[0076] 一边13沿着节点9的方向连接节点9和10，并被分配有等于9的权重22，以显示模式9和10中函数的连续调用重复9次，

[0077] 一边14连接节点11自身，并被分配有等于9的权重23，以显示模式11中函数的连续调用重复9次，

[0078] 一边15沿着节点7的方向连接节点7和11，并被分配有等于9的权重24，以显示模式7至11中函数的连续调用重复9次。

[0079] 为了由跟踪文件生成压缩跟踪文件，并随后能生成图形16，在深度上遍历与所得到的语法对应的分层结构，并将各种非终端符号进行比较，以将相等的非终端符号分为一组，同时计算其在语法的各个深度重复的次数。语法的深度对应于得到分层结构中的终端符号之前非终端符号的数量。

[0080] 因此，针对各种进程获得各种图形。然后对这些图形进行比较，以形成进程的组。一组由具有相同图形的所有进程组成。在该进程的结尾，每组保存单个图形，从而使要制作的模型能更加直观，并且每组进程仅显示一个图形。

[0081] 如上所述，图形16由节点7至11和定向边12至20组成，该定向边表示从一个调用到另一个调用或从一个系列调用到另一个系列调用的通道。与环12至15对应的边12至15被分配有表示分别通过这些边的通道的数量的权重21至24。

[0082] 一旦得到图形16，就恢复压缩期间排除的所有数据，以重新认识被传输消息的所有时间、持续时间和大小概念。通过恢复这些数据，可以对每个循环和/或对每个调用计算关于被传输消息的时间和/或大小的统计数据。还可以在同一个组内检测进程的不同步。

[0083] 基于这些数据，可以向每个节点分配与对应的函数调用或系列调用中所花费的时间成比例的权重。节点还可以与组中的各进程的对应调用或系列调用相关的所有统计数据相关联。

[0084] 因此可以根据图形和/或与分配给节点的相关联的统计数据 and/或分配给节点的权重和/或分配给边的权重对应用程序的性能进行分析。

[0085] 为了实现方法，使用计算机化系统，该计算机化系统包括处理器和存储器。存储器中存储有程序指令，当程序指令被处理器执行时，该程序指令实施本发明的方法。

[0086] 本说明书以示例方式给出，并且不限制本发明。

[0087] 特别地，本发明不限于使用MPI类型的进程间通信函数的应用程序。其适用于其他的函数库，诸如例如“开放式多重处理”或OpenMP类型的函数。其还适用于对输入/输出(读写)函数的调用进行分析。更一般地，其适用于对下述应用程序进行分析：该应用程序进行函数调用，根据该函数调用，可以按时间顺序从跟踪文件中提取信息或在执行相关应用程序期间得到信息。

```

1  for (int j=0;j<N;j++){
    for(int i=0; i<SIZE2; i+=len,id+=2){
        MPI_Irecv(&(a[i]),len,MPI_DOUBLE,recv[0],0,MPI_COMM_WORLD,&(req1[id]));
        MPI_Irecv(&(a[i]),len,MPI_DOUBLE,recv[1],0,MPI_COMM_WORLD,&(req1[id+1]));
    }
    id=0;
    for(int i=0; i<SIZE2;i+=len ,id+=2){
        MPI_Isend(&(a[i]),len,MPI_DOUBLE,send[1],0,MPI_COMM_WORLD,&(req2[id+1]));
        MPI_Isend(&(a[i]),len,MPI_DOUBLE,send[0],0,MPI_COMM_WORLD,&(req2[id]));
    }
    MPI_Waitall((2*SIZE2)/len,req1,stat);
    MPI_Waitall((2*SIZE2)/len,req2,stat);
    }
6

```

图1

```

1, 2 { "type": "MPI", "rank": 0, "function": " MPI_Irecv ", "start": 0.025754, "duration": 0.000001, "size": 80.000000, "sender": 5, "recv": 0 }
      { "type": "MPI", "rank": 0, "function": " MPI_Irecv ", "start": 0.025759, "duration": 0.000000, "size": 80.000000, "sender": 1, "recv": 0 }
      { "type": "MPI", "rank": 0, "function": " MPI_Irecv ", "start": 0.025762, "duration": 0.000000, "size": 80.000000, "sender": 5, "recv": 0 }
      { "type": "MPI", "rank": 0, "function": " MPI_Irecv ", "start": 0.025766, "duration": 0.000000, "size": 80.000000, "sender": 1, "recv": 0 }
      ...
      { "type": "MPI", "rank": 0, "function": " MPI_Irecv ", "start": 0.025795, "duration": 0.000000, "size": 80.000000, "sender": 5, "recv": 0 }
      { "type": "MPI", "rank": 0, "function": " MPI_Irecv ", "start": 0.025796, "duration": 0.000000, "size": 80.000000, "sender": 1, "recv": 0 }
3, 4 { "type": "MPI", "rank": 0, "function": " MPI_Isend ", "start": 0.025800, "duration": 0.000004, "size": 80.000000, "sender": 0, "recv": 1 }
      { "type": "MPI", "rank": 0, "function": " MPI_Isend ", "start": 0.025806, "duration": 0.000003, "size": 80.000000, "sender": 0, "recv": 5 }
      { "type": "MPI", "rank": 0, "function": " MPI_Isend ", "start": 0.025812, "duration": 0.000001, "size": 80.000000, "sender": 0, "recv": 1 }
      { "type": "MPI", "rank": 0, "function": " MPI_Isend ", "start": 0.025815, "duration": 0.000001, "size": 80.000000, "sender": 0, "recv": 5 }
      ...
      { "type": "MPI", "rank": 0, "function": " MPI_Isend ", "start": 0.025865, "duration": 0.000001, "size": 80.000000, "sender": 0, "recv": 1 }
      { "type": "MPI", "rank": 0, "function": " MPI_Isend ", "start": 0.025868, "duration": 0.000001, "size": 80.000000, "sender": 0, "recv": 5 }
5 { "type": "MPI", "rank": 0, "function": " MPI_Waitall ", "start": 0.025871, "duration": 0.000022, "size": 0.000000, "sender": -1, "recv": 0 }
6 { "type": "MPI", "rank": 0, "function": " MPI_Waitall ", "start": 0.025897, "duration": 0.000000, "size": 0.000000, "sender": -1, "recv": 0 }
      ...
1, 2 { "type": "MPI", "rank": 0, "function": " MPI_Irecv ", "start": 0.025900, "duration": 0.000000, "size": 80.000000, "sender": 5, "recv": 0 }
      { "type": "MPI", "rank": 0, "function": " MPI_Irecv ", "start": 0.025902, "duration": 0.000000, "size": 80.000000, "sender": 1, "recv": 0 }
      { "type": "MPI", "rank": 0, "function": " MPI_Irecv ", "start": 0.025904, "duration": 0.000000, "size": 80.000000, "sender": 5, "recv": 0 }
      { "type": "MPI", "rank": 0, "function": " MPI_Irecv ", "start": 0.025906, "duration": 0.000000, "size": 80.000000, "sender": 1, "recv": 0 }
      ...
      { "type": "MPI", "rank": 0, "function": " MPI_Irecv ", "start": 0.025943, "duration": 0.000000, "size": 80.000000, "sender": 5, "recv": 0 }
      { "type": "MPI", "rank": 0, "function": " MPI_Irecv ", "start": 0.025945, "duration": 0.000000, "size": 80.000000, "sender": 1, "recv": 0 }
3, 4 { "type": "MPI", "rank": 0, "function": " MPI_Isend ", "start": 0.025947, "duration": 0.000002, "size": 80.000000, "sender": 0, "recv": 1 }
      { "type": "MPI", "rank": 0, "function": " MPI_Isend ", "start": 0.025951, "duration": 0.000001, "size": 80.000000, "sender": 0, "recv": 5 }
      { "type": "MPI", "rank": 0, "function": " MPI_Isend ", "start": 0.025954, "duration": 0.000001, "size": 80.000000, "sender": 0, "recv": 1 }
      { "type": "MPI", "rank": 0, "function": " MPI_Isend ", "start": 0.025957, "duration": 0.000001, "size": 80.000000, "sender": 0, "recv": 5 }
      ...
      { "type": "MPI", "rank": 0, "function": " MPI_Isend ", "start": 0.026004, "duration": 0.000001, "size": 80.000000, "sender": 0, "recv": 1 }
      { "type": "MPI", "rank": 0, "function": " MPI_Isend ", "start": 0.026007, "duration": 0.000001, "size": 80.000000, "sender": 0, "recv": 5 }
5 { "type": "MPI", "rank": 0, "function": " MPI_Waitall ", "start": 0.026010, "duration": 0.000000, "size": 0.000000, "sender": -1, "recv": 0 }
6 { "type": "MPI", "rank": 0, "function": " MPI_Waitall ", "start": 0.026012, "duration": 0.000002, "size": 0.000000, "sender": -1, "recv": 0 }

```

图2

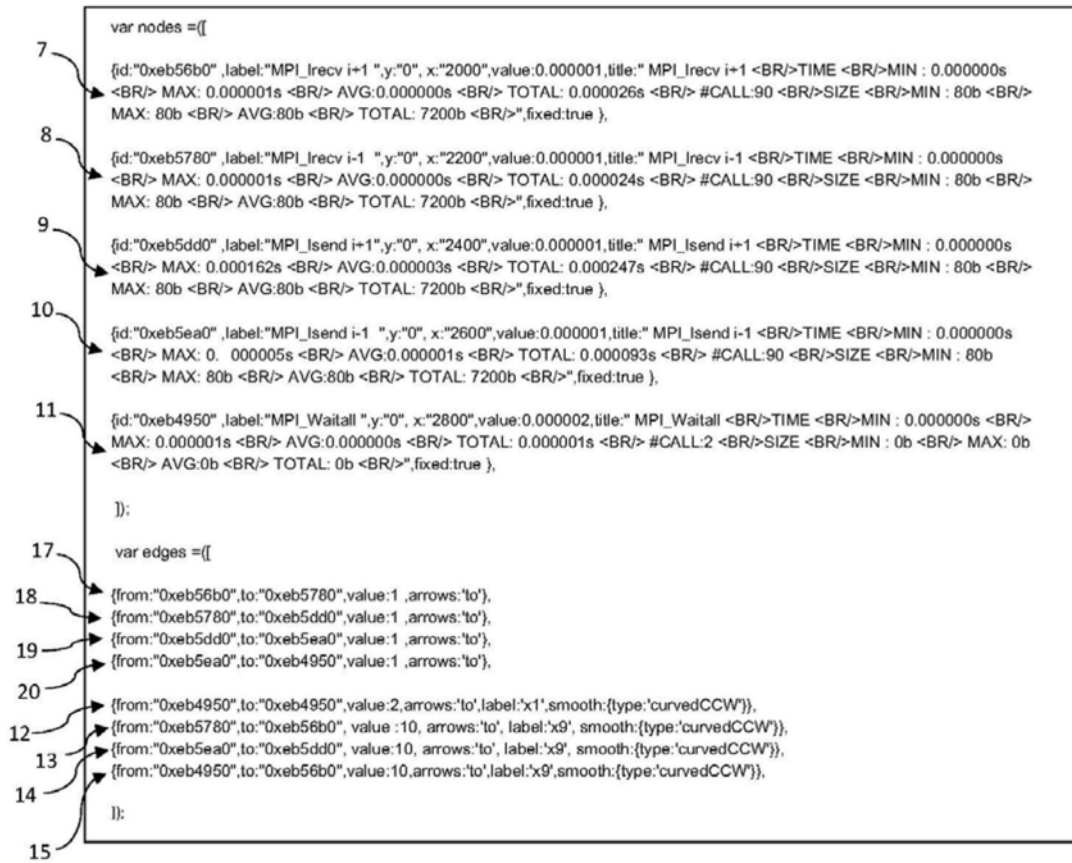


图3

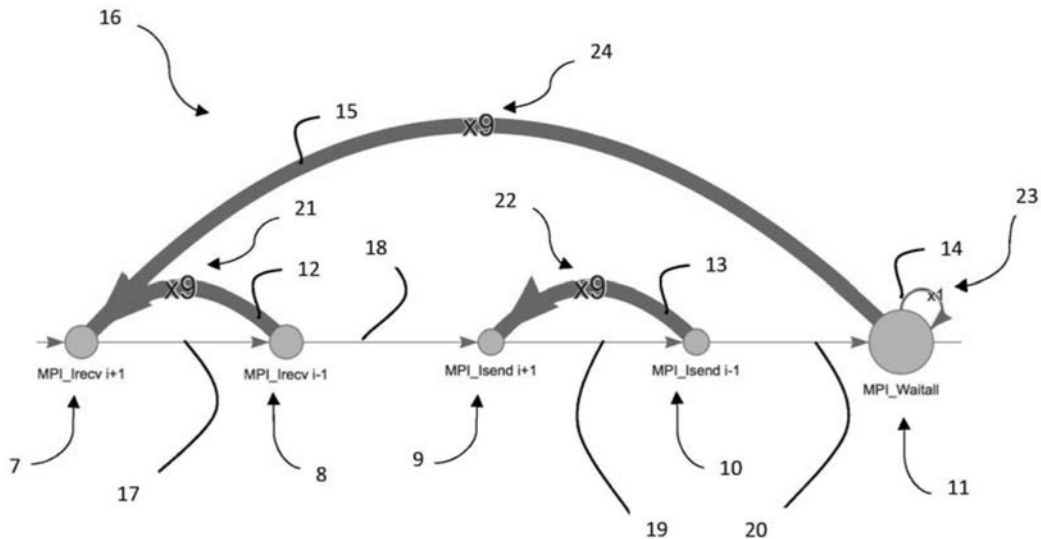


图4