带有分支结构 OpenMP 任务图的响应时间分析

孙景昊[1),2)] 张利威[1)] 池瑶瑶[1)] 曹 蕾[1)] 邓庆绪[1)]

[1)](东北大学计算机科学与工程学院 沈阳 110819)

[2)](大连理工大学计算机科学与技术学院 辽宁 大连 116024)

摘 要 随着多核技术在实时系统中广泛应用,实时程序的并行化成为当前的研究热点.在实时领域,有向无环图(DAG)是刻画并行实时程序的理论模型.然而,传统的 DAG 任务图并不能刻画并行程序的实际特征(例如 if-else 控制流结构).于是,能够同时反映程序的并行负载特征和 if-else 控制流结构的分支并行任务图(conditional DAG:con-DAG)应运而生.目前,实时领域通常假设 con-DAG 满足某种特殊结构,即图中的并行子结构和分支子结构必须是单入口和单出口(简称为"单入单出").在这种约束下,con-DAG 的响应时间分析问题具有多项式时间算法.然而,现实的 OpenMP 程序具有更加灵活的结构."单入单出"假设一旦失效,con-DAG 响应时间分析是否依然存在多项式时间算法为开放性问题.本文针对一般情况下 OpenMP 程序的分支结构开展研究.对于非"单入单出" con-DAG 图上响应时间分析问题,基于动态规划理论,提出了多项式时间的求解方法.实验表明,本文方法求得的 con-DAG 响应时间在之前方法的基础上能够提升 3%,为实时并行程序的可预测性提供更精确的理论支撑.

关键词 OpenMP;有向无环图;if-else 分支结构;响应时间;多项式时间

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2020.02166

Exactly Analyzing Response Time of OpenMP Tasks with Conditional Branches

SUN Jing-Hao[1),2)] ZHANG Li-Wei[1)] CHI Yao-Yao[1)] CAO Lei[1)] DENG Qing-Xu[1)]

[1)](*School of Computer Science and Engineering, Northeastern University, Shenyang* 110819)

[2)](*School of Computer Science and Technology, Dalian University of Technology, Dalian, Liaoning* 116024)

Abstract Multi-cores are becoming mainstream hardware platforms for embedded and real-time systems. To fully utilize the processing capacity of multi-cores, software should be parallelized. Recently much work has been done on real-time scheduling of parallel tasks modeled as directed acyclic graphs (DAG), motivated by the parallel task structures supported by popular parallel programming frameworks such as OpenMP. The DAG-based task models in existing real-time scheduling research cannot fully capture critical features of parallel programs, e.g., if-else conditional structure. Recently, the conditional DAG models are proposed to formulate the real-time systems composed by parallel workload and conditional components. Existing DAG-based task models in real-time scheduling research assume special structures recursively composed by single-source-single-sink parallel and conditional components. With this special assumption, polynomial-time response time analysis method is developed. However, realistic OpenMP task systems in general have more flexible structures that do not comply with this assumption. This paper studies general OpenMP task systems with general branching structures, and develops a polynomial-time dynamic programming algorithm to efficiently calculate response time bounds for

收稿日期:2019-08-30;在线发布日期:2020-07-01.本课题得到国家自然科学基金(61972076)、兴辽英才计划(XLYC1902017)、NSFC-辽宁联合基金(U1908212)资助.**孙景昊**,博士,副教授,中国计算机学会(CCF)会员,主要研究方向为实时系统调度、并行程序分析.E-mail:jhsun@dlut.edu.cn.**张利威**,硕士研究生,主要研究方向为 OpenMP 并行程序分析.**池瑶瑶**,硕士研究生,主要研究方向为实时并行系统调度.**曹 蕾**,学士,主要研究方向为实时系统、优化算法.**邓庆绪**(通信作者),博士,教授,主要研究领域为实时系统.E-mail:dengqx@mail.neu.edu.cn.

OpenMP task systems. First, this paper outlines OpenMP program semantics, and model the behaviors of general OpenMP task systems with general branching structures, including task models, execution models and scheduling models. Second, this paper deeply analyzes the related method in the existing work. By constructing a counterexample, we find that the existing method cannot accurately calculate the response time boundary of the task graphs, and there is a certain pessimism. The specific reason is that although the existing method considers OpenMP task graphs with multi-source-multi-sink conditional branching structures, it cannot correctly analyze some situations, e. g. , the parameters associated with the two branches in the conditional branching structure are quite different. Third, for the analysis problem of response time on multi-source-multi-sink con-DAG graphs, based on dynamic programming theory, this paper proposes an accurate solution method of polynomial time. The main idea of this method is as follows. The DAG graph is traversed topologically in reverse order, then the parameters associated with the predecessor nodes are calculated by using the calculation results of the successor nodes, and finally the response time bound of the DAG graph is calculated based on the values of the associated variables of the source node. Compared with the existing algorithm, the algorithm in this paper introduces more node-associated variables and considers more node types. In the experiment part, we implement the algorithm of existing work and the algorithm in this paper. The response time bounds of the two algorithms are evaluated by randomly generated DAGs. Experimental work shows that compared to the traditional method, our method can effectively improve the accuracy of the response time bound. Under average conditions, the accuracy of the response time bound can be increased by 3%. It is very important for real-time system to improve the accuracy of response time bound effectively. In real-time systems, the response time directly affects the schedulability of systems. For real-time systems whose schedulability is in critical state, the accuracy of response time is increased by 1%, which is likely to greatly increase the schedulability of systems.

Keywords　OpenMP; directed acyclic graph; if-else branch structure; response time; polynomial time

1　引　言

近年来，多核技术越来越广泛地应用于实时系统. 为了充分利用多核的计算资源，实时软件并行化成为当前的研究热点和未来的发展趋势. OpenMP[1]是多核系统最流行的并行编程框架之一. 近年来，在实时嵌入式领域 OpenMP 得到了广泛关注. 在实时领域中，有向无环图(Directed Acyclic Graph，DAG)是刻画 OpenMP 程序的天然模型，基于 DAG 图的实时调度和分析的问题中涌现出大量研究工作[2-12]. 然而，DAG 图并不能完全刻画 OpenMP 的程序特征. OpenMP 程序的一个关键特征是：OpenMP 任务间不仅具有并行性，而且在任务内还具有条件分支结构(例如 if-else 语句). 因此，文献[13-18]研究了带有条件分支结构的 con-DAG 任务图.

本文发现，现有的 con-DAG 任务图模型仍然不能刻画实际的 OpenMP 程序. 例如，在现有的 con-DAG 图中，并行结构和条件分支结构具有单入口单出口的特征(以下简称为“单入单出”). 如图 1 所示，该 con-DAG 图包含一个并行结构(黄色框)，并行结构内嵌一个条件分支结构(绿色框). 这些并行结构和条件分支结构都具有一个入口和一个出口，不允许出现图中红色边(从条件分支结构外/内指向条件分支结构内/外)的情况.

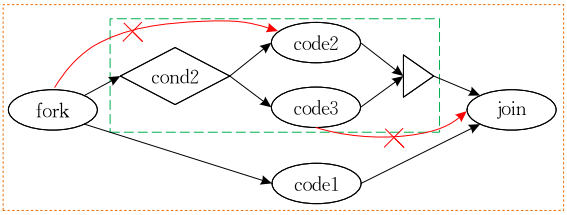


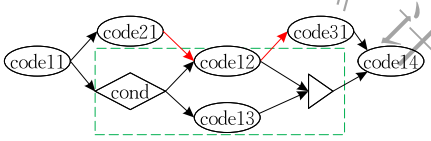
图 1　“单入单出”con-DAG 模型示例

然而，在实际的 OpenMP 程序中，并行和条件分支结构可能不满足“单入单出”条件. 图 2(a)为一个 OpenMP 程序，其中条件分支结构内所触发的负载(code31)与该条件分支结构外所触发的负载

(code14)同步.现有的实时调度研究并不能很好地描述和分析这种行为.如果用类似于图 1 的方法对图 2(a)中的 OpenMP 程序进行建模,将有两条边(在图 2(b)中标记为红色),从条件分支结构外/内部节点指向条件分支结构内/外部节点.也就是说,图 2(b)的 DAG 图中条件分支结构具有多个入口和出口.当前研究[14-18]并不适用于图 2 中的任务模型.本文将对此类一般情况下 OpenMP 任务进行建模与分析.

```
1 #pragma omp task{
2   code11;
3   #pragma omp task{code21;}
4   if(exp)
5     #pragma omp taskwait;
6     code12;
7     #pragma omp task{code31;}
8   else
9     code13;
10  #pragma omp taskwait;
11  code14;
12    }
```

(a) OpenMP程序



(b) 相应的DAG任务图

图 2 OpenMP 程序及相应的 DAG 任务图

本文旨在分析带有条件分支结构 OpenMP 程序的响应时间.文献[9]研究发现,不带条件分支结构 OpenMP 程序的响应时间,可由经典 Graham 界[19]来限定.由于 con-DAG 图的执行流是一个非条件 DAG 图,所以文献[9]的结论可以扩展到 con-DAG 图中.即首先枚举 con-DAG 图中所有可能的执行流,然后求解每个执行流的 Graham 界,并求取最大值.然而,这种枚举的方法,在面对指数规模的执行流时,其计算复杂度也是指数的.为了规避枚举方法,当前研究[14-18]基于动态规划给出了 con-DAG 图的响应时间界限的多项式时间计算方法.但是,这些方法均局限于带有"单入单出"约束的 con-DAG 图,不适用于带有"多入多出"条件分支结构的一般性 OpenMP 程序.

本文针对一般性 OpenMP 程序开展研究,提出具有"多入多出"条件分支结构 con-DAG 任务图的响应时间算法.在撰写本文之前,文献[20]也研究了"多入多出"条件分支结构的 OpenMP 程序响应时间分析问题.但是,本文发现,文献[20]提出的算法并不能精确求解 con-DAG 的响应时间界,具有相当的悲观性.相比较而言,本文提出的方法不但能够精确求解 con-DAG 的响应时间界,而且还具有多项式时间的计算复杂度.实验表明,本文的算法在求解精度上,优于文献[20]中的方法.与此同时,本文实验表明,文献[20]方法具有较少的计算时间,这说明在不要求较高求解精度的情况下,文献[20]方法具有易实现、鲁棒的优点.

本文第 2 节介绍 con-DAG 方面的相关研究工作;第 3 节建立 OpenMP 系统的形式化模型;第 4 节介绍响应时间界;第 5 节指出文献[20]中的方法并不能精确求解响应时间界;第 6 节提出伪多项式时间的精确算法;第 7 节是实验结果;最后是总结与展望.

2 相关工作

文献[10]首次将 OpenMP 程序建模为 DAG 图.文献[9]针对 OpenMP 框架中已有的调度算法(例如,宽度优先调度(Breath-First-Scheduling,BFS)和负载优先调度(Work-First-Scheduling,WFS)),基于文献[10]中 DAG 任务图模型,首次为 OpenMP 程序提出了响应时间界限分析.在文献[9]中,OpenMP 任务分为两种类型:绑定型任务和非绑定型任务.其中,绑定型任务一旦在一个线程上开始执行,就不能迁移到另一个线程.另外,非绑定型任务则不受此限制,能够在不同的线程上迁移执行.文献[9]指出,绑定型任务的时序分析具有内在的复杂性,这将导致响应时间分析的悲观性.之后,文献[11]的研究发现,绑定型任务不但会导致响应时间的悲观性,还可能引起整个 OpenMP 系统极差的时间行为(例如,原本并行的程序,由于绑定型任务的 runtime 约束,被强制在单核上串行执行).上述工作均没有考虑 OpenMP 中的条件分支结构.

文献[14-17]研究了 con-DAG 模型,主要提出了两类计算响应时间界限方法:一种方法将 con-DAG 转换为不包含条件节点的(非条件)DAG 图模型,然后直接应用非条件 DAG 图的分析和计算方法求解 con-DAG 的响应时间界限[14-15].另一种方法则是基于动态规划思想,在 con-DAG 图上直接计算响应时间[16-17].这两种方法均具有多项式时间的计算复杂性,也都局限于"单入单出"条件分支结构的 con-DAG 图模型.最近,文献[20]研究了带有"多入多出"条件分支结构 con-DAG 的响应时间计算方法.然而,本文发现,文献[20]中的方法并不能精确求解 con-DAG 图响应时间界限,存在一定的悲观

性. 本文工作扩展自文献[20]，并在实验部分将本文提出的方法与文献[20]的方法进行了定量分析和比对.

此外，文献[21-24]研究了 OpenMP 程序的平均意义上的性能分析. 这与本文针对最坏情况下 OpenMP 响应时间分析存在较大的差异：文献[21-24]基于度量的方法，不需要显式地构建条件分支结构. 与此相反，本文最坏情况的响应时间分析显式地构建了 OpenMP 程序条件分支结构和时序特性，并根据这些信息形式化地推出响应时间的上界.

3　系统模型

本节通过对 OpenMP 程序语义进行简单介绍，建立 OpenMP 系统的形式化模型，具体包括任务模型、执行模型和调度模型.

3.1　OpenMP 程序与线程

本文主要研究具有 task 语义的 OpenMP 程序. task 语义在 OpenMP 3.0[17] 及以上版本中出现. 图 3 给出了一个 OpenMP 程序示例. 该程序从 parallel 指令(第 1 行)开始. parallel 指令构造了一个并行域，其包括 parallel 指令后面花括号中的所有代码(第 2 行至第 21 行).

```
1  #pragma omp parallel num_threads (m)
2  {#pragma omp single
3    {#pragma omp task  //τ1
4      {code11;  //v1^1
5        #pragma omp task{code21;  //v2^1} //τ2
6        if(exp1)    //entry: v1^4; exit: v1^5
7          code12;   //v1^2
8          #pragma omp task //τ3
9          {code31;  //v3^1
10           #pragma omp task{ code41;  //v4^1} //τ4
11           code32;  //v3^2
12         } //end of τ3
13       else
14         #pragma omp taskwait;
15         code13;   //v1^3
16       #pragma omp taskwait;
17       code16;  //v1^6
18       code17;  //v1^7
19     }
20   }
21 }
```

图 3　OpenMP 程序示例

parallel 指令创建 m 个 OpenMP 线程，其中 m 由 num_threads 子句指定(第 1 行). OpenMP 线程是指能够执行并行域内代码的实体(例如，操作系统中的线程). 与之前工作[9-10]类似，本文假设每个线程独享一个内核的计算资源. 因此，本文中“线程”的概念相当于一个物理内核.

3.2　OpenMP 任务模型

并行域中的代码可以划分为一组并行任务 $\Gamma=\{\tau_1,\cdots,\tau_n\}$. 其中，任务 τ_i 对应第 i 个 task 指令后面括号中所包含的代码域. 如图 3 所示，τ_1 对应第 4 行的代码. 但是，第 9 行的代码不包含在 τ_1 的代码域中. 显然，第 9 行的代码属于 τ_3 的代码域. 在 OpenMP 任务系统 Γ 中，每个任务都有内部控制流结构，任务之间存在依赖关系.

本文将 OpenMP 任务系统 Γ 建模为 DAG 图 $G=(V,E)$. 其中，V 表示节点集，每个节点表示 task 指令中的顺序代码段. E 表示边集，每条边表示两个节点之间的前驱后继关系. 图 G 可以分为 n 个不相交的子图，即 $G=G_1\cup\cdots\cup G_n$. 其中，子图 $G_i=(V_i,E_i)$ 是任务 τ_i 的控制流图(Control-Flow Graph，CFG). 图 4 给出了图 3 中对应 DAG 图的任务模型.

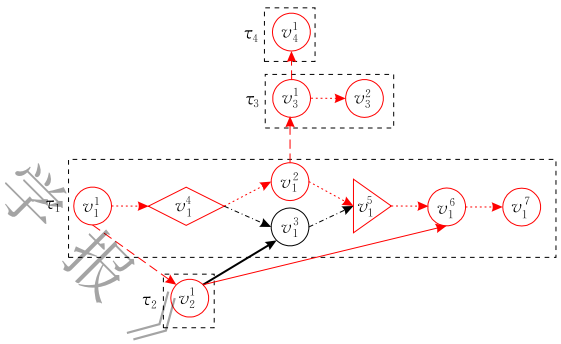


图 4　图 3 中 OpenMP 程序的 DAG 任务图

在每个 CFG $G_i=(V_i,E_i)$ 中，V_i 表示节点集，节点 v_i^x 表示 τ_i 的第 x 个节点，该节点的最坏情况执行时间(WCET)定义为 $c(v_i^x)$. E_i 表示 F-边集，每条 F-边 $(v_i^x,v_i^y)\in E_i$(在图 4 中用点虚线箭头表示)定义了节点 v_i^x 和 v_i^y 的时序关系，即在节点 v_i^x 完成执行之后，节点 v_i^y 才能开始执行. 特殊地，没有 F-入边的节点称为 G_i 的源点，没有 F-出边的节点称为 G_i 的汇点，分别定义为 v_i^{src} 和 v_i^{snk}. 本文假设每个 G_i 都有一个源点和一个汇点. 然而，本文的方法也可以处理具有多个源/汇点的控制流图的情况：只需在多个源点之前添加一个执行时间为零的虚拟源点，在多个汇点之后添加一个执行时间为零的虚拟汇点.

接下来，分别介绍 CFG 图的内部结构和 CFG 间的依赖关系.

3.2.1　CFG 图的内部结构

每个任务 τ_i 是一个具有单入口和单出口的程序

块(block).程序块中包含条件分支结构,如图 3 中第 6～15 行所示.一般情况下,每个条件分支结构包含一个进入点(if 指令)和一个退出点(endif 指令),以及 if 和 else 两个分支.条件分支结构中只有一个分支会得到执行,当该分支完成执行后,条件分支结构退出执行.本文允许嵌套条件分支结构,即一个条件分支结构可以包含在另一个条件分支结构中.

根据 OpenMP 任务内结构,可以把 CFG 中的节点 V_i 分为两类:条件节点和非条件节点.

(1)条件节点成对出现,在图中用菱形和三角形表示,它们分别表示条件分支结构的入口节点和出口节点.如图 4 所示,v_1^4 是条件节点.条件分支结构包含两个分支,每个分支都是 G_i 的一个子图.

(2)非条件节点是除条件节点外的所有其它节点.一个非条件节点最多有一条 F-出边,最多有一条 F-入边.非条件节点又分为以下三种类型:

① T-节点(V_i^{T})表示 task 指令前的代码段.如图 4 所示,v_1^1 是 T-节点,对应于图 3 的第 4 行处的代码段 code11.

② W-节点(V_i^{W})表示 taskwait 指令后的代码段.如图 4 所示,v_1^3 是 W-节点,对应于图 3 的第 15 行处的代码段 code13.

③ N-节点(V_i^{N}).既非 T 类型又非 W 类型的非条件节点.

3.2.2 CFG 间的依赖关系

任务之间有两种依赖关系:任务创建和任务同步,详细介绍如下:

(1)任务创建.task 指令用于创建 OpenMP 任务.如图 3 所示,任务 τ_1 是由第 3 行的 task 指令创建的.OpenMP 支持任务嵌套,例如,一个任务的代码域中还包含其它任务的 task 指令.一般地,如果 τ_i 的代码域包含 τ_j 的 task 指令,则称 τ_i 是 τ_j 的父任务,τ_j 是 τ_i 的子任务.如果 τ_i 是 τ_j 的父任务,或者 τ_i 是 τ_j 的父任务的父任务(可嵌套定义),则称 τ_i 是 τ_j 的祖先任务,τ_j 是 τ_i 的子孙任务.令 D_i 为 τ_i 的任务图以及 τ_i 子孙任务图的并.如图 3 所示,τ_2 是 τ_1 的孩子,τ_4 是 τ_1 的后代,τ_1 是 τ_4 的祖先.一般地,我们假设第一个任务 τ_1 没有父任务.任何其它任务都只有一个父任务.

相应地,CFG 之间也存在任务创建关系,并由 T-边来确定.

T-边,用半划线箭头表示.父任务通过 T-边指向其子任务.准确地说,如果(τ_i 的)T-节点 v_i^x 创建了 τ_i 的子任务 τ_j,则从 v_i^x 到 τ_j 的源点有一条 T-边.在这种情况下,只有 v_i^x 完成执行后,τ_j 才能够释放和执行.在图 4 中,(v_1^1, v_2^1) 是一条 T-边,表示节点 v_1^1 创建任务 τ_2.因为在 OpenMP 中,一个 task 指令创建一个任务,所以一个 T-节点关联一条 T-出边.

规则 1. 只有 T-边能从任务(的 T-节点)指向其子任务(的源点).

(2)任务同步.taskwait 指令定义任务同步点,即任务可以在 taskwait 指令处挂起并等待其它任务完成.准确地说,在 t 时刻,任务 τ_p 在它自身的 taskwait 指令处挂起,直到所有在 t 时刻之前创建的 τ_p 的子任务都完成,任务 τ_p 才能继续执行.如图 3 所示,任务 τ_1 在其 taskwait 子句处挂起,并等待其子任务 τ_2 的完成.

相应地,CFG 之间也存在任务同步关系,并由 W-边来确定.

W-边,用实线箭头表示.子任务通过 W-边指向其父任务.准确地说,对于任意 T-节点 v_i^x 以及 W-节点 v_i^y,其中,v_i^x 是 v_i^y 的前驱,τ_j 表示由 v_i^x 创建的 τ_i 的子任务,在这种情况下,有一条 W-边从 τ_j 的汇点指向 v_i^y.

规则 2. 只有 W-边能从任务(的汇点)指向其父任务(的 W-节点).

W-边 (v_j^{snk}, v_i^y) 定义 OpenMP 中的父任务 τ_i 与其子任务 τ_j 同步,即父任务 τ_i 在 W-节点 v_i^y 处挂起,等待其子任务 τ_j 完成后,可恢复执行.如图 4 所示,由于 τ_2 是由 v_1^8 的前驱节点 v_1^1 创建的且图中有一条从 v_1^1 到 v_1^3 不经过任何中间 W-节点的路径,所以 (v_2^1, v_1^3) 是一条 W-边,该边从 τ_2 的汇点指向 τ_1 中的 W-节点 v_1^3.从 τ_4 到 v_1^6 没有 W-边,因为 τ_4 不是由 v_1^6 的前驱节点创建的.从 τ_2 到 τ_1 有两条 W-边,即 (v_2^1, v_1^3) 和 (v_2^1, v_1^6),因为从 v_1^1(τ_1 的 T-节点)到 W-节点 v_1^3 和 v_1^6 存在路径(不经过任何中间 W-节点),例如 (v_1^1, v_1^4, v_1^3) 和 $(v_1^1, v_1^4, v_1^2, v_1^5, v_1^6)$.

3.3 OpenMP 执行流模型

G 从第一个任务 τ_1 开始执行,进而执行 τ_1 的子孙任务.G 中的每个任务 τ_i 从源点 v_i^{src} 开始执行,到汇点 v_i^{snk} 执行结束.在任务 τ_i 执行期间,有两种可能的情况.

(1)当遇到条件节点 v_i^x 时,有且只有 v_i^x 的一个后继节点得到执行.

(2)当遇到 N-节点或 T-节点 v_i^x 时,通过 F-边或 T-边连接的 v_i^x 的所有后继节点都得到执行.

根据以上执行语义,由于 G 中存在条件分支结

构，通常情况下，在 G 中只有部分任务和部分节点得到执行. 因此，G 上的一次执行对应 G 的一个子图 G'，其包含执行中所遍历的所有节点. 称 G' 为 G 的一条执行流 ε. 如图 4 所示，执行流 ε 中的节点和边用红色标记. 因为 ε 没有遍历 v_1^3，所以 W-边 (v_2^1, v_1^3) 不包含在 ε 中.

3.4　OpenMP 调度模型

OpenMP 系统合理的任务调度满足以下两个约束：

(1) 任务调度点(TSP)包括任务创建点(task 指令)、同步点(taskwait 指令)以及任务完成退出点. 任务的执行只能在 TSP 处中断和恢复.

(2) 绑定型任务. OpenMP 任务分为两种类型：绑定型任务和非绑定型任务. 其中，绑定型任务一旦在一个线程上开始执行，就不能迁移到另一个线程. 非绑定型任务则不受此限制，能够在不同的线程上迁移执行.

在 OpenMP 任务模型中，OpenMP 调度是指将执行流 ε 中的节点分配给线程，直到 ε 中的所有节点完成执行. 根据 OpenMP 系统合理调度任务时满足的约束，执行流 ε 的合理调度必须满足以下约束条件：

(1) 对于执行流 ε 中任意节点 v_i^x，只有当 v_i^x 在 ε 中的所有前驱节点(v_i^y 是 v_i^x 的前驱节点当且仅当 $(v_i^y, v_i^x) \in \varepsilon$)都完成执行，$v_i^x$ 才能执行. 特殊地，对于执行流 ε 中 W-节点 v_i^x，v_i^x 有两个前驱节点：v_i^y 和 v_j^z，分别由 F-边和 W-边连接，其中，v_j^z 是 τ_i 的子任务 τ_j 的汇点. 根据执行语义，在 v_i^x 执行之前，节点 v_i^y 和任务 τ_j 必须完成执行.

(2) 任务 τ_i 的执行在两个连续节点 v_i^y 和 v_i^x 之间发生中断，仅当下列条件中至少有一个满足：

① v_i^x 是 W-节点；

② v_i^y 是 T-节点；

③ ε 中的任务允许在线程之间迁移.

OpenMP 系统支持两种调度算法：负载优先调度(WFS)[25] 和宽度优先调度(BFS)[26]. WFS 优先执行新创建的任务，而 BFS 优先执行已经在线程上执行的任务. 在 WFS 和 BFS 的调度下，非绑定型任务系统满足 work-conserving 约束，其响应时间可由 Graham 界保证. 本文考虑非绑定型 OpenMP 任务，并采用 WFS 或 BFS 调度算法.

本文假设执行流 ε 由 WFS 或 BFS 算法调度(详见 3.3 节). 根据文献[9]，基于 WFS 和 BFS 算法，执行流 ε 的调度满足 work-conserving 约束. 根据文献[19]可知，满足 work-conserving 约束的调度具有良好的响应时间界，如第 4 节所示.

迁移开销. 由于本文考虑了非绑定型任务，因此任务可能会在线程之间迁移. 任务迁移会产生系统开销. 注意到，在 OpenMP 调度算法(例如，WFS 和 BFS)的调度下，任务迁移仅发生在某些特定类型的节点上，比如，T-节点和 W-节点. 因此，迁移开销可以限定如下. 我们用 O 表示单个节点的迁移开销，用 NC 表示 DAG 图中 T-节点的数目，用 NW 表示 DAG 图中 W-节点的数目. 在 WFS 的调度下，DAG 图的总迁移开销小于 $NC \times O$. 在 BFS 的调度下，DAG 图的总迁移开销小于 $NW \times O$.

4　响应时间界

在 m 个线程上，应用 WFS 或 BFS 算法调度，任务图 G 的最坏情况响应时间(Worst Case Response Time，WCRT) $R(G)$ 定义如下：

$$R(G) = \max\{R(\varepsilon) \mid \varepsilon \text{ 是 } G \text{ 上的一条执行流}\} \quad (1)$$

其中，$R(\varepsilon)$ 是执行流 ε 的 WCRT. 由于一条执行流是一个非条件 DAG 图，根据文献[9]，$R(\varepsilon)$ 可由以下引理限定.

引理 1[9]. 在 m 个线程上，应用 WFS 或 BFS 算法调度，执行流 ε 响应时间 $R(\varepsilon)$ 的界限如下：

$$R(\varepsilon) \leq len(\varepsilon) + \frac{vol(\varepsilon) - len(\varepsilon)}{m} \quad (2)$$

其中，$len(\varepsilon)$ 是 ε 的最长路径长度，$vol(\varepsilon)$ 表示 ε 中所有节点的最坏情况执行时间之和(即最大负载).

由于式(2)是 Graham 在文献[19]中首次提出的，所以式(2)又称为 Graham 界. 下面的定理表明，Graham 界可以推广到带有条件分支结构的 OpenMP 任务图.

定理 1. 在 m 个线程上，应用 WFS 或 BFS 算法调度，OpenMP 任务图 G 响应时间 $R(G)$ 的界限如下：

$$R(G) \leq \max_{\varepsilon \in G} \left\{ len(\varepsilon) + \frac{vol(\varepsilon) - len(\varepsilon)}{m} \right\} \quad (3)$$

其中，ε 代表 G 中的执行流.

证明. 根据引理 1 和式(1)，易知定理 1 成立. 证毕.

5　已有工作存在的问题

目前，与本文最相近的工作是文献[20]. 文献

[20]同样解决带有(非“单入单出”)条件分支结构 OpenMP 任务图的响应时间分析问题,给出了多项式时间算法.然而,文献[20]的方法并不能精确求解式(3)中的响应时间界,存在一定的悲观性.接下来,首先简单介绍文献[20]中计算方法的相关概念和关键步骤.然后,通过构造反例,揭示文献[20]中方法的悲观性.最后,比较本文算法和文献[20]中算法的异同点.

5.1 文献[20]中算法简介

文献[20]中的算法应用动态规划技术,按照图 G 的拓扑逆序对节点进行遍历.在遍历过程中,对于图中每个节点 v_i^x,直接利用 v_i^x 后继节点的计算结果,推导出 v_i^x 关联变量的值.最后,响应时间界即可由 v_1^{src} 的关联变量计算得出.接下来,首先介绍节点 v_i^x 关联变量的相关概念.然后,给出 v_i^x 关联变量的递推公式.

定义 1(S-图). 节点 v_i^x 引导的 S-图 $G_S(v_i^x)$ 包含节点 v_i^x 及 v_i^x 的后继节点,即 $G_S(v_i^x)=\{v_i^x\}\bigcup \mathrm{SUCC}(v_i^x)$.其中,$\mathrm{SUCC}(v_i^x)$ 为 v_i^x 的所有后继节点.

例如,在图 4 中,$G_S(v_2^1)=(v_2^1,v_1^3,v_1^5,v_1^6,v_1^7)$.

定义 2(S-执行流). 对于 G 中任意执行流 ε,以及 ε 中的节点 v_i^x,v_i^x 引导的 S-执行流 $\varepsilon_S(v_i^x)$ 为子图 $\varepsilon\bigcap G_S(v_i^x)$.

定义 3(D-图). 节点 v_i^x 引导的 D-图 $G_D(v_i^x)$ 包含 v_i^x 及 v_i^x 在 τ_i(和 τ_i 的子孙任务)中的后继节点,即 $G_D(v_i^x)=\{v_i^x\}\bigcup\overline{\mathrm{SUCC}}(v_i^x)$.其中,$\overline{\mathrm{SUCC}}(v_i^x)=\mathrm{SUCC}(v_i^x)\bigcap D_i$,$D_i$ 为 τ_i 的任务图以及 τ_i 子孙任务图的并.

例如,在图 4 中,$G_D(v_2^1)=(v_2^1)$.

定义 4(D-执行流). 对于 G 中任意的执行流 ε,以及 ε 中的节点 v_i^x,v_i^x 引导的 D-执行流 $\varepsilon_D(v_i^x)$ 为子图 $\varepsilon\bigcap G_D(v_i^x)$.

节点 v_i^x 关联的变量定义如下.

定义 5(S-图 $G_S(v_i^x)$ 的 Length). $G_S(v_i^x)$ 中最长路径的长度,即

$$len_G^S(v_i^x)=\max_{\varepsilon\in G} len_\varepsilon^S(v_i^x) \tag{4}$$

其中,$len_\varepsilon^S(v_i^x)$ 是 S-执行流 $\varepsilon_S(v_i^x)$ 中最长路径长度.

定义 6(D-图 $G_D(v_i^x)$ 的 Volume). $G_D(v_i^x)$ 中 D-执行流的最大负载,即

$$vol_G^D(v_i^x)=\max_{\varepsilon\in G} vol_\varepsilon^D(v_i^x) \tag{5}$$

其中,$vol_\varepsilon^D(v_i^x)$ 是 D-执行流 $\varepsilon_D(v_i^x)$ 中所有节点的响应时间之和,即

$$vol_\varepsilon^D(v_i^x)=\sum_{v_j^y\in\varepsilon_D(v_i^x)} c(v_j^y) \tag{6}$$

定义 7(S-图 $G_S(v_i^x)$ 的界). $G_S(v_i^x)$ 中 S-执行流的最大界,即

$$gra_G^S(v_i^x)=\max_{\varepsilon\in G} gra_\varepsilon^S(v_i^x) \tag{7}$$

其中,$gra_\varepsilon^S(v_i^x)$ 是 S-执行流 $\varepsilon_S(v_i^x)$ 的界,即

$$gra_\varepsilon^S(v_i^x)=len_\varepsilon^S(v_i^x)+\frac{vol_\varepsilon^D(v_i^x)-len_\varepsilon^S(v_i^x)}{m} \tag{8}$$

节点 v_i^x 关联变量的计算分以下两种情况:

(1)若 v_i^x 是条件分支结构的入口节点,设 $v_i^{y_1}$ 和 $v_i^{y_2}$ 是 v_i^x 的直接后继,那么,

$$gra_G^S(v_i^x)=c(v_i^x)+\max\{gra_G^S(v_i^{y_1}),gra_G^S(v_i^{y_2})\} \tag{9}$$

(2)若 v_i^x 是 T-节点,设 τ_j 是 v_i^x 所创建的 τ_i 的子任务,v_j^{src} 是 τ_j 的源点,v_i^y 是(通过 F-边连接的)v_i^x 的后继节点,则

$$gra_G^S(v_i^x)=c(v_i^x)+\max\{\Gamma_1,\Gamma_2\} \tag{10}$$

其中,

$$\Gamma_1=gra_G^S(v_j^{src})+\frac{vol_G^D(v_i^y)}{m},$$

$$\Gamma_2=gra_G^S(v_i^y)+\frac{vol_G^D(v_j^{src})}{m}.$$

5.2 文献[20]中算法的悲观性

本节通过构造反例(如图 5 所示),说明文献[20]中的算法不能精确求解式(3)中的响应时间界.在图 5 中,最外层的任务 τ_i 包含一个 T-节点 v_i^x 和一个以 v_i^y 为入口的条件分支结构.其中,T-节点 v_i^x 创建了 τ_i 的子任务 τ_j.另外,条件分支结构包含两个分支.其中,第一个分支包含一个执行时间为 L 的 W-节点,第二个分支包含 mL 个 T-节点,每个节点创建一个执行时间为 1 的任务,分别定义为 $\tau_1,\tau_2,\cdots,\tau_{mL}$.DAG 图中其它节点执行时间均为 0.

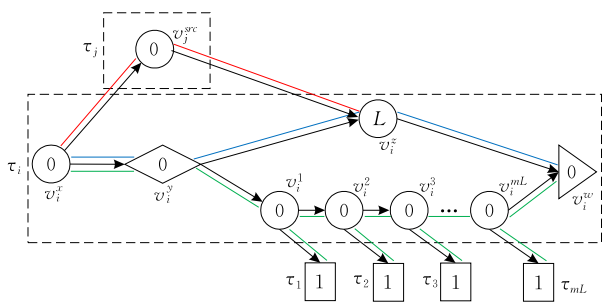


图 5 文献[20]的反例

根据式(10),为计算 v_i^x 的关联变量 $gra_G^S(v_i^x)$,需要预先计算出 v_j^{src} 的关联变量 $gra_G^S(v_j^{src})$ 和 $vol_G^D(v_j^{src})$ 以及 v_i^y 的关联变量 $gra_G^S(v_i^y)$ 和 $vol_G^D(v_i^y)$.其中,v_j^{src} 是 v_i^x 所创建任务 τ_j 的源点,v_i^y 是 v_i^x 的后继节点.如图 5 所示,根据定义 6,可以计算两个参数

$vol_G^{\mathrm{D}}(v_j^{src})=0$ 和 $vol_G^{\mathrm{D}}(v_i^{y})=mL$. 根据定义 7,可以计算两个参数 $gra_G^{\mathrm{S}}(v_j^{src})=L(1-1/m)$ 和 $gra_G^{\mathrm{S}}(v_i^{y})=L+(1-1/m)$. 进一步地,可计算得出式(10)中的中间变量 $\Gamma_1=L+L(1-1/m)$, $\Gamma_2=L+(1-1/m)$. 因此, $gra_G^{\mathrm{S}}(v_i^{x})=L+L(1-1/m)$.

另一方面,图 5 中 DAG 任务图实际包含两条执行流 ε_1 和 ε_2. 其中,执行流 ε_1 包含 $\{v_i^x, v_i^y, v_j^{src}, v_i^z, v_i^w\}$(如图 5 中红色和蓝色轨迹的组合). 执行流 ε_2 包含 $\{v_i^x, v_i^y, v_j^{src}, v_i^1, v_i^2, \cdots, v_i^{mL}, \tau_1, \tau_2, \cdots, \tau^{mL}, v_i^w\}$(如图 5 中红色和绿色轨迹的组合) ε_1 关联的变量 $gra_{\varepsilon_1}^{\mathrm{S}}(v_i^x)=L$, ε_2 关联的变量 $gra_{\varepsilon_2}^{\mathrm{S}}(v_i^x)=L+(1-1/m)$. 因此,以上两者取大,可得 $gra_G^{\mathrm{S}}(v_i^x)=L+(1-1/m)$. 文献[20]中算法的误差为 $(L-1)(1-1/m)$.

5.3　本文算法和文献[20]中算法的异同

由上述反例可以看出,文献[20]中的算法存在一定的悲观性. 本文在第 6 节将提出能够精确求解式(3)中响应时间界的算法. 本文算法与文献[20]中算法的异同如下:

相同点. 两个算法都采用了动态规划技术,为图中的节点关联一定数量的变量,按照 DAG 图的拓扑逆序对节点进行遍历,计算各节点关联的变量. 在计算过程中,图中每个节点 v_i^x 的关联变量,可直接利用 v_i^x 后继节点的计算结果得出. 最后,响应时间界即为源点关联的相关变量.

不同点. 本文的算法能够精确求解式(3)中的响应时间界,然而,文献[20]中的算法仅能得到近似解. 具体的原因是:文献[20]虽然考虑了带有“多入多出”条件分支结构的 DAG 图,但是不能正确分析条件分支结构中两个分支所关联的参数差异较大的情况. 例如,一个分支关联较小的 vol 和较长的 len,另一个分支关联较大的 vol 和较短的 len(详见图 5 所示). 在这种情况下,文献[20]中的算法把 DAG 任务图中实际没有包含的路径(如图 5 中 v_i^z)纳入了响应时间计算,从而导致了响应时间界的悲观估计. 为了解决这个问题,本文算法引入更多的节点关联变量(详见第 6.1 节表 1),并对节点关联变量的计算情况进行更加细致地划分(详见第 6.2 节).

6　多项式时间精确算法

由第 5 节可知,已有工作中的算法存在悲观性,不能精确求解式(3)中的响应时间界. 本节将给出能够精确求解式(3)中响应时间界的多项式时间算法. 与文献[20]中方法类似,本文依然采用动态规划技术. 如算法 1 所示,对图 G 进行拓扑逆序遍历,利用后继节点的计算结果来计算前驱节点关联的参数. 最后,由源点 v_i^{src} 的关联变量值计算出图 G 的响应时间界.

算法 1.　OpenMP 任务图响应时间界的精确算法.

输入:带有条件分支结构的有向无环图

输出:最坏情况响应时间界

1. 按照 G 的拓扑逆序 σ 遍历节点;
2. 对于 G 的拓扑逆序 σ 中的每个节点 v_i^x:
3. 　　按照 6.2 节方法计算 v_i^x 关联的变量;
4. 由 v_i^x 的关联变量计算 G 的响应时间界(详见式(3));

相较于文献[20]方法,本文方法为图 G 中的节点关联了更多的变量. 这些变量对于式(3)的精确求解起到关键作用. 接下来,首先介绍节点关联的变量. 然后,给出响应时间界的精确求解方法. 最后,比较文献[20]方法与本文方法的精确度.

6.1　节点关联的变量

图 G 中每个节点 v_i^x 的关联变量如表 1 所示.

表 1　图 G 中节点关联的变量定义

变量名称	变量的形式化定义
$vol_{\mathrm{D}}(v_i^x)$	$vol^{\mathrm{D}}(v_i^x)=\max_{\varepsilon\in G}\{vol_\varepsilon^{\mathrm{D}}(v_i^x)\}$
$len^{\mathrm{D}}(v_i^x)$	$len^{\mathrm{D}}(v_i^x)=\max_{\varepsilon\in G}\{len_\varepsilon^{\mathrm{D}}(v_i^x)\}$
$gra_{\mathrm{D}}(v_i^x)$	$gra_{\mathrm{D}}(v_i^x)=\max_{\varepsilon\in G}\{vol_\varepsilon^{\mathrm{D}}(v_i^x)/m+(1-1/m)\,len_\varepsilon^{\mathrm{D}}(v_i^x)\}$
$gra_e(v_i^x)$	$gra_e(v_i^x)=\max_{\varepsilon\in G}\{vol_\varepsilon^{\mathrm{D}}(v_i^x)/m+(1-1/m)\,len_\varepsilon^{e}(v_i^x)\}$
$gra_w(v_i^x)$	$gra_w(v_i^x)=\max_{\varepsilon\in G}\{vol_\varepsilon^{\mathrm{D}}(v_i^x)/m+(1-1/m)\,len_\varepsilon^{w}(v_i^x)\}$
$gra_u(v_i^x)$	$gra_u(v_i^x)=\max_{\varepsilon\in G}\{vol_\varepsilon^{\mathrm{D}}(v_i^x)/m+(1-1/m)\,len_\varepsilon^{u}(v_i^x)\}$

在表 1 的各个公式中,对于给定的执行流 ε,根据定义 4,节点 v_i^x 引导的 D-执行流为 $\varepsilon_{\mathrm{D}}(v_i^x)$. $vol_\varepsilon^{\mathrm{D}}(v_i^x)$ 为 D-执行流 $\varepsilon_{\mathrm{D}}(v_i^x)$ 的负载量. $len_\varepsilon^{\mathrm{D}}(v_i^x)$ 为 D-执行流 $\varepsilon_{\mathrm{D}}(v_i^x)$ 中最长路径的长度. $len_\varepsilon^{e}(v_i^x)$ 为 D-执行流 $\varepsilon_{\mathrm{D}}(v_i^x)$ 中以 v_i^x 为起点且以 v_i^{snk} 为终点的最长路径长度. $len_\varepsilon^{w}(v_i^x)$ 为 D-执行流 $\varepsilon_{\mathrm{D}}(v_i^x)$ 中以 W-节点为起点的最长路径长度. $len_\varepsilon^{u}(v_i^x)$ 为 D-执行流 $\varepsilon_{\mathrm{D}}(v_i^x)$ 中以 W-节点为起点且以 v_i^{snk} 为终点的最长路径长度.

6.2　节点变量的递推公式

下面介绍表 1 中节点关联变量的计算方法. 根据第 3 节的任务模型可知,一个节点 v_i^x 可能有零个后继、一个后继和两个后继三种情况. 因此,将按照这三种可能,分情况讨论:

(1) 如果节点 v_i^x 没有后继节点,则可按以下方法计算各参数.

$$vol^{\mathrm{D}}(v_i^x)=c(v_i^x) \tag{11}$$

$$len^{\mathrm{D}}(v_i^x)=c(v_i^x) \tag{12}$$

$$gra_{\mathrm{D}}(v_i^x)=c(v_i^x) \tag{13}$$

$$gra_e(v_i^x)=c(v_i^x) \tag{14}$$

$$gra_w(v_i^x)\begin{cases}c(v_i^x), & \text{若 } v_i^x \text{ 是 W-节点}\\ -\infty, & \text{否则}\end{cases} \tag{15}$$

$$gra_u(v_i^x)\begin{cases}c(v_i^x), & \text{若 } v_i^x \text{ 是 W-节点}\\ -\infty, & \text{否则}\end{cases} \tag{16}$$

其中,$-\infty$ 表示“负无穷大”,即对于任意实数 a,$-\infty+a=-\infty$.

(2) 如果节点 v_i^x 仅有一个直接后继节点,则考虑以下三种情况:

① 如果 v_i^x 的后继节点 v_i^y 与 v_i^x 同属于任务 τ_i,则可按以下方法计算各参数.

$$vol^{\mathrm{D}}(v_i^x)=c(v_i^x)+vol^{\mathrm{D}}(v_i^y) \tag{17}$$

$$len^{\mathrm{D}}(v_i^x)=c(v_i^x)+len^{\mathrm{D}}(v_i^y) \tag{18}$$

$$gra_{\mathrm{D}}(v_i^x)=c(v_i^x)+gra_{\mathrm{D}}(v_i^y) \tag{19}$$

$$gra_e(v_i^x)=c(v_i^x)+gra_e(v_i^y) \tag{20}$$

$$gra_w(v_i^x)\begin{cases}gra_{\mathrm{D}}(v_i^x), & \text{若 } v_i^x \text{ 是 W-节点}\\ gra_w(v_i^y), & \text{否则}\end{cases} \tag{21}$$

$$gra_u(v_i^x)\begin{cases}gra_e(v_i^x), & \text{若 } v_i^x \text{ 是 W-节点}\\ gra_u(v_i^y), & \text{否则}\end{cases} \tag{22}$$

图 6 给出了式(17)~(22)的解释.如图 6 所示,最外层的任务 τ_i 包含一个以 v_i^z 为入口的条件分支结构.该条件分支结构包含两个分支.其中,第一个分支包含一个 N-节点,第二个分支包含一个 T-节点 v_i^w,节点 v_i^w 创建了 τ_i 的子任务 τ_j.DAG 图中所有节点的执行时间均为 1.

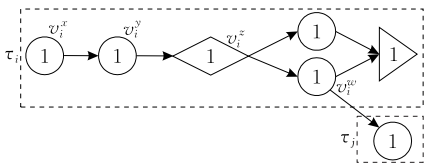


图 6 式(17)~(22)的解释示例

式(17)的含义为,v_i^x 的关联变量 $vol^{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 v_i^y 的关联变量 $vol^{\mathrm{D}}(v_i^y)$.其中,v_i^y 是 v_i^x 的后继节点.例如,在图 6 中,v_i^x 的关联变量 $vol^{\mathrm{D}}(v_i^x)=1+5=6$.

式(18)的含义为,v_i^x 的关联变量 $len^{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 v_i^y 的关联变量 $len^{\mathrm{D}}(v_i^y)$.其中,v_i^y 是 v_i^x 的后继节点.例如,在图 6 中,v_i^x 的关联变量 $len^{\mathrm{D}}(v_i^x)=1+4=5$.

式(19)的含义为,v_i^x 的关联变量 $gra_{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 v_i^y 的关联变量 $gra_{\mathrm{D}}(v_i^y)$.其中,v_i^y 是 v_i^x 的后继节点.例如,在图 6 中,v_i^x 的关联变量 $gra_{\mathrm{D}}(v_i^x)=1+4+1/m=5+1/m$.

式(20)的含义为,v_i^x 的关联变量 $gra_e(v_i^x)$ 等于 v_i^x 的执行时间加上 v_i^y 的关联变量 $gra_e(v_i^y)$.其中,v_i^y 是 v_i^x 的后继节点.例如,在图 6 中,v_i^x 的关联变量 $gra_e(v_i^x)=1+4+1/m=5+1/m$.

式(21)的含义为,若 v_i^x 是 W-节点,则 v_i^x 的关联变量 $gra_w(v_i^x)$ 等于 $gra_{\mathrm{D}}(v_i^x)$.否则,$gra_w(v_i^x)$ 等于 v_i^y 的关联变量 $gra_w(v_i^y)$.其中,v_i^y 是 v_i^x 的后继节点.例如,在图 6 中,v_i^x 的关联变量 $gra_w(v_i^x)=5/m$.

式(22)的含义为,若 v_i^x 是 W-节点,则 v_i^x 的关联变量 $gra_u(v_i^x)$ 等于 $gra_e(v_i^x)$.否则,$gra_u(v_i^x)$ 等于 v_i^y 的关联变量 $gra_u(v_i^y)$.其中,v_i^y 是 v_i^x 的后继节点.例如,在图 6 中,v_i^x 的关联变量 $gra_u(v_i^x)=5/m$.

② 如果 v_i^x 是 T-节点,则可按以下方法计算各参数.其中,τ_j 是 T-节点 v_i^x 所创建的任务,v_j^{src} 是任务 τ_j 的源点,也即 v_i^x 的后继节点.

$$vol^{\mathrm{D}}(v_i^x)=c(v_i^x)+vol^{\mathrm{D}}(v_j^{src}) \tag{23}$$

$$len^{\mathrm{D}}(v_i^x)=c(v_i^x)+len^{\mathrm{D}}(v_j^{src}) \tag{24}$$

$$gra_{\mathrm{D}}(v_i^x)=c(v_i^x)+gra_{\mathrm{D}}(v_j^{src}) \tag{25}$$

$$gra_e(v_i^x)=c(v_i^x) \tag{26}$$

$$gra_w(v_i^x)\begin{cases}gra_{\mathrm{D}}(v_i^x), & \text{若 } v_i^x \text{ 是 W-节点}\\ -\infty, & \text{否则}\end{cases} \tag{27}$$

$$gra_u(v_i^x)\begin{cases}gra_e(v_i^x), & \text{若 } v_i^x \text{ 是 W-节点}\\ -\infty, & \text{否则}\end{cases} \tag{28}$$

图 7 给出了式(23)~(28)的解释.如图 7 所示,最外层的任务 τ_i 包含一个 T-节点 v_i^x,v_i^x 创建了 τ_i 的子任务 τ_j.任务 τ_j 包含一个以 v_j^z 为入口的条件分支结构.该条件分支结构包含两个分支.DAG 图中所有节点的执行时间均为 1.

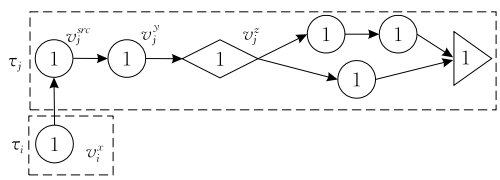


图 7 式(23)~(28)的解释示例

式(23)的含义为,v_i^x 的关联变量 $vol^{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 v_j^{src} 的关联变量 $vol^{\mathrm{D}}(v_j^{src})$.其中,$v_j^{src}$ 是节点 v_i^x 所创建任务 τ_j 的源点.例如,在图 7 中,

v_i^x的关联变量 $vol^{\mathrm{D}}(v_i^x)=1+6=7$.

式(24)的含义为，v_i^x的关联变量 $len^{\mathrm{D}}(v_i^x)$等于 v_i^x的执行时间加上 v_j^{src}的关联变量 $len^{\mathrm{D}}(v_j^{src})$. 其中，$v_j^{src}$是节点 v_i^x所创建任务 τ_j的源点. 例如，在图 7 中，v_i^x的关联变量 $len^{\mathrm{D}}(v_i^x)=1+6=7$.

式(25)的含义为，v_i^x的关联变量 $gra_{\mathrm{D}}(v_i^x)$等于 v_i^x的执行时间加上 v_j^{src}的关联变量 $gra_{\mathrm{D}}(v_j^{src})$. 其中，$v_j^{src}$是 v_i^x所创建任务 τ_j的源点. 例如，在图 7 中，v_i^x的关联变量 $gra_{\mathrm{D}}(v_i^x)=1+6=7$.

式(26)的含义为，v_i^x的关联变量 $gra_e(v_i^x)$等于 v_i^x的执行时间. 例如，在图 7 中，v_i^x的关联变量 $gra_e(v_i^x)=1$.

式(27)的含义为，若 v_i^x是 W-节点，则 v_i^x的关联变量 $gra_w(v_i^x)$等于 $gra_{\mathrm{D}}(v_i^x)$. 否则，$gra_w(v_i^x)$等于 $-\infty$. 例如，在图 7 中，v_i^x的关联变量 $gra_w(v_i^x)=-\infty$.

式(28)的含义为，若 v_i^x是 W-节点，则 v_i^x的关联变量 $gra_u(v_i^x)$等于 $gra_e(v_i^x)$. 否则，$gra_u(v_i^x)$等于 $-\infty$. 例如，在图 7 中，v_i^x的关联变量 $gra_u(v_i^x)=-\infty$.

③ 如果 v_i^x的后继是 W-节点 v_j^y(任务 τ_j即为 τ_i的父任务)，则可按以下方法计算各参数.

$$vol^{\mathrm{D}}(v_i^x)=c(v_i^x) \tag{29}$$

$$len^{\mathrm{D}}(v_i^x)=c(v_i^x)+len^{\mathrm{D}}(v_j^y) \tag{30}$$

$$gra_{\mathrm{D}}(v_i^x)=c(v_i^x)+gra_{\mathrm{D}}(v_j^y) \tag{31}$$

$$gra_e(v_i^x)=c(v_i^x) \tag{32}$$

$$gra_w(v_i^x)\begin{cases}gra_{\mathrm{D}}(v_i^x), & \text{若 } v_i^x \text{是 W-节点}\\ -\infty, & \text{否则}\end{cases} \tag{33}$$

$$gra_u(v_i^x)\begin{cases}gra_e(v_i^x), & \text{若 } v_i^x \text{是 W-节点}\\ -\infty, & \text{否则}\end{cases} \tag{34}$$

图 8 给出了式(29)～(34)的解释. 如图 8 所示，最外层的任务 τ_j包含一个 W-节点 v_j^y、一个 T-节点 v_j^w和一个以 v_j^z为入口的条件分支结构. 其中，T-节点 v_j^w创建了 τ_j的子任务 τ_i. DAG 图中所有节点的执行时间均为 1.

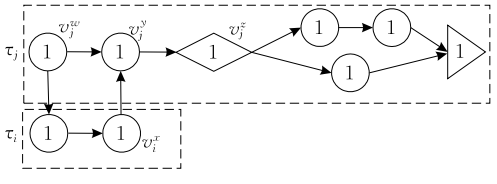


图 8　式(29)～(34)的解释示例

式(29)的含义为，v_i^x的关联变量 $vol^{\mathrm{D}}(v_i^x)$等于 v_i^x的执行时间. 例如，在图 8 中，v_i^x的关联变量 $vol^{\mathrm{D}}(v_i^x)=1$.

式(30)的含义为，v_i^x的关联变量 $len^{\mathrm{D}}(v_i^x)$等于 v_i^x的执行时间加上 v_j^y的关联变量 $len^{\mathrm{D}}(v_j^y)$. 其中，v_j^y是 W-节点且为 v_i^x的后继节点. 例如，在图 8 中，v_i^x的关联变量 $len^{\mathrm{D}}(v_i^x)=1+5=6$.

式(31)的含义为，v_i^x的关联变量 $gra_{\mathrm{D}}(v_i^x)$等于 v_i^x的执行时间加上 v_j^y的关联变量 $gra_{\mathrm{D}}(v_j^y)$. 其中，v_j^y是 W-节点且为 v_i^x的后继节点. 例如，在图 8 中，v_i^x的关联变量 $gra_{\mathrm{D}}(v_i^x)=1+5=6$.

式(32)的含义为，v_i^x的关联变量 $gra_e(v_i^x)$等于 v_i^x的执行时间. 例如，在图 8 中，v_i^x的关联变量 $gra_e(v_i^x)=1$.

式(33)的含义为，若 v_i^x是 W-节点，则 v_i^x的关联变量 $gra_w(v_i^x)$等于 $gra_{\mathrm{D}}(v_i^x)$. 否则，$gra_w(v_i^x)$等于 $-\infty$. 例如，在图 8 中，v_i^x的关联变量 $gra_w(v_i^x)=-\infty$.

式(34)的含义为，若 v_i^x是 W-节点，则 v_i^x的关联变量 $gra_u(v_i^x)$等于 $gra_e(v_i^x)$. 否则，$gra_u(v_i^x)$等于 $-\infty$. 例如，在图 8 中，v_i^x的关联变量 $gra_u(v_i^x)=-\infty$.

(3) 如果节点 v_i^x有两个直接后继节点，则考虑以下三种情况：

① 如果 v_i^x是条件分支结构的入口节点(假设 v_i^x的两个后继节点分别为 v_i^y和 v_i^z)，则可按以下方法计算各参数.

$$vol^{\mathrm{D}}(v_i^x)=c(v_i^x)+\max\{vol^{\mathrm{D}}(v_i^y),vol^{\mathrm{D}}(v_i^z)\} \tag{35}$$

$$len^{\mathrm{D}}(v_i^x)=c(v_i^x)+\max\{len^{\mathrm{D}}(v_i^y),len^{\mathrm{D}}(v_i^z)\} \tag{36}$$

$$gra_{\mathrm{D}}(v_i^x)=c(v_i^x)+\max\{gra_{\mathrm{D}}(v_i^y),gra_{\mathrm{D}}(v_i^z)\} \tag{37}$$

$$gra_e(v_i^x)=c(v_i^x)+\max\{gra_e(v_i^y),gra_e(v_i^z)\} \tag{38}$$

$$gra_w(v_i^x)\begin{cases}gra_{\mathrm{D}}(v_i^x), & \text{若 } v_i^x \text{是 W-节点}\\ \max\{gra_w(v_i^y),gra_w(v_i^z)\}, & \text{否则}\end{cases} \tag{39}$$

$$gra_u(v_i^x)=\max\{gra_u(v_i^y),gra_u(v_i^z)\} \tag{40}$$

图 9 给出了式(35)～(40)的解释. 如图 9 所示，最外层的任务 τ_i包含一个 T-节点 v_i^z和一个以 v_i^x为入口的条件分支结构. 其中，T-节点 v_i^z创建 τ_i的子任务 τ_j. 另外，条件分支结构的第一个分支包含一个

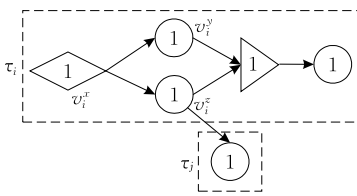


图 9　式(35)～(40)的解释示例

节点 v_i^y,第二个分支包含一个节点 v_i^z. DAG 图中所有节点的执行时间均为 1.

式(35)的含义为,节点 v_i^x 的关联变量 $vol^{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 $\max\{vol^{\mathrm{D}}(v_i^y),vol^{\mathrm{D}}(v_i^z)\}$. 其中,$v_i^y$ 和 v_i^z 是条件节点 v_i^x 的两个后继节点. 另外,$\max\{vol^{\mathrm{D}}(v_i^y),vol^{\mathrm{D}}(v_i^z)\}$ 表示在 v_i^y 的关联变量 $vol^{\mathrm{D}}(v_i^y)$ 和 v_i^z 的关联变量 $vol^{\mathrm{D}}(v_i^z)$ 中取最大值. 例如,在图 9 中,$vol^{\mathrm{D}}(v_i^y)=3$,$vol^{\mathrm{D}}(v_i^z)=4$. 因此,$v_i^x$ 的关联变量 $vol^{\mathrm{D}}(v_i^x)=1+4=5$.

式(36)的含义为,节点 v_i^x 的关联变量 $len^{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 $\max\{len^{\mathrm{D}}(v_i^y),len^{\mathrm{D}}(v_i^z)\}$. 其中,$v_i^y$ 和 v_i^z 是条件节点 v_i^x 的两个后继节点. 另外,$\max\{len^{\mathrm{D}}(v_i^y),len^{\mathrm{D}}(v_i^z)\}$ 表示在 v_i^y 的关联变量 $len^{\mathrm{D}}(v_i^y)$ 和 v_i^z 的关联变量 $len^{\mathrm{D}}(v_i^z)$ 中取最大值. 例如,在图 9 中,$len^{\mathrm{D}}(v_i^y)=3$,$len^{\mathrm{D}}(v_i^z)=3$. 因此,$v_i^x$ 的关联变量 $len^{\mathrm{D}}(v_i^x)=1+3=4$.

式(37)的含义为,v_i^x 的关联变量 $gra_{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 $\max\{gra_{\mathrm{D}}(v_i^y),gra_{\mathrm{D}}(v_i^z)\}$. 其中,$v_i^y$ 和 v_i^z 是条件节点 v_i^x 的两个后继节点. 另外,$\max\{gra_{\mathrm{D}}(v_i^y),gra_{\mathrm{D}}(v_i^z)\}$ 表示在节点 v_i^y 的关联变量 $gra_{\mathrm{D}}(v_i^y)$ 和 v_i^z 的关联变量 $gra_{\mathrm{D}}(v_i^z)$ 中取最大值. 例如,在图 9 中,$gra_{\mathrm{D}}(v_i^y)=3$,$gra_{\mathrm{D}}(v_i^z)=3+1/m$. 因此,$v_i^x$ 的关联变量 $gra_{\mathrm{D}}(v_i^x)=1+3+1/m=4+1/m$.

式(38)的含义为,v_i^x 的关联变量 $gra_e(v_i^x)$ 等于 v_i^x 的执行时间加上 $\max\{gra_e(v_i^y),gra_e(v_i^z)\}$. 其中,$v_i^y$ 和 v_i^z 是条件节点 v_i^x 的两个后继节点. 另外,$\max\{gra_e(v_i^y),gra_e(v_i^z)\}$ 表示在 v_i^y 的关联变量 $gra_e(v_i^y)$ 和 v_i^z 的关联变量 $gra_e(v_i^z)$ 中取最大值. 例如,在图 9 中,$gra_e(v_i^y)=3$,$gra_e(v_i^z)=3+1/m$. 因此,v_i^x 的关联变量 $gra_e(v_i^x)=1+3+1/m=4+1/m$.

式(39)的含义为,若 v_i^x 是 W-节点,则 v_i^x 的关联变量 $gra_w(v_i^x)$ 等于 $gra_{\mathrm{D}}(v_i^x)$. 否则,$gra_w(v_i^x)$ 等于 $\max\{gra_w(v_i^y),gra_w(v_i^z)\}$. 其中,$v_i^y$ 和 v_i^z 是条件节点 v_i^x 的两个后继节点. 另外,$\max\{gra_w(v_i^y),gra_w(v_i^z)\}$ 表示在节点 v_i^y 的关联变量 $gra_w(v_i^y)$ 和 v_i^z 的关联变量 $gra_w(v_i^z)$ 中取最大值. 例如,在图 9 中,v_i^x 的关联变量 $gra_w(v_i^x)=4/m$.

式(40)的含义为,v_i^x 的关联变量 $gra_u(v_i^x)$ 等于 $\max\{gra_u(v_i^y),gra_u(v_i^z)\}$. 其中,$v_i^y$ 和 v_i^z 是条件节点 v_i^x 的两个后继节点. 另外,$\max\{gra_u(v_i^y),gra_u(v_i^z)\}$ 表示在节点 v_i^y 的关联变量 $gra_u(v_i^y)$ 和 v_i^z 的关联变量 $gra_u(v_i^z)$ 中取最大值. 例如,在图 9 中,v_i^x 的关联变量 $gra_u(v_i^x)=4/m$.

② 如果 v_i^x 创建任务 τ_j 且有后继节点 v_i^y(如图 10 所示),则按以下方法计算各参数.

$$vol^{\mathrm{D}}(v_i^x)=c(v_i^x)+\max\{vol^{\mathrm{D}}(v_i^y),vol^{\mathrm{D}}(v_j^{src})\} \tag{41}$$

$$len^{\mathrm{D}}(v_i^x)=c(v_i^x)+\max\{len^{\mathrm{D}}(v_i^y),len^{\mathrm{D}}(v_j^{src})\} \tag{42}$$

$$gra_{\mathrm{D}}(v_i^x)=c(v_i^x)+\max\{\Gamma_1,\Gamma_2,\Gamma_3\} \tag{43}$$

其中,

$$\Gamma_1=vol^{\mathrm{D}}(v_j^{src})/m+gra_{\mathrm{D}}(v_i^y),$$

$$\Gamma_2=gra_{\mathrm{D}}(v_j^{src})+vol^{\mathrm{D}}(v_i^y)/m,$$

$$\Gamma_3=gra_e(v_j^{src})+gra_w(v_i^y),$$

$$gra_e(v_i^x)=c(v_i^x)+\max\{\Omega_1,\Omega_2\} \tag{44}$$

其中,

$$\Omega_1=vol^{\mathrm{D}}(v_j^{src})/m+gra_e(v_i^y),$$

$$\Omega_2=gra_e(v_j^{src})+gra_u(v_i^y),$$

$$gra_w(v_i^x)=\begin{cases}gra_{\mathrm{D}}(v_i^x), & \text{若 } v_i^x \text{ 是 W-节点}\\ gra_w(v_i^y), & \text{否则}\end{cases} \tag{45}$$

$$gra_u(v_i^x)=\begin{cases}gra_e(v_i^x), & \text{若 } v_i^x \text{ 是 W-节点}\\ gra_u(v_i^y), & \text{否则}\end{cases} \tag{46}$$

图 10 给出了式(41)~(46)的解释. 如图 10 所示,最外层的任务 τ_i 包含一个 T-节点 v_i^x 和一个以 v_i^z 为入口的条件分支结构. 其中,T-节点 v_i^x 创建了 τ_i 的子任务 τ_j. DAG 图中所有节点执行时间均为 1.

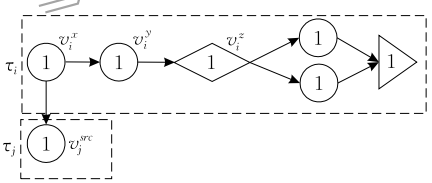


图 10 式(41)~(46)的解释示例

式(41)的含义为,v_i^x 的关联变量 $vol^{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 $\max\{vol^{\mathrm{D}}(v_i^y),vol^{\mathrm{D}}(v_j^{src})\}$. 其中,$v_i^y$ 是 v_i^x 的后继节点,v_j^{src} 是节点 v_i^x 所创建任务 τ_j 的源点. 另外,$\max\{vol^{\mathrm{D}}(v_i^y),vol^{\mathrm{D}}(v_j^{src})\}$ 表示在 v_i^y 的关联变量 $vol^{\mathrm{D}}(v_i^y)$ 和 v_j^{src} 的关联变量 $vol^{\mathrm{D}}(v_j^{src})$ 中取最大值. 例如,在图10 中,$vol^{\mathrm{D}}(v_i^y)=4$,$vol^{\mathrm{D}}(v_j^{src})=1$. 因此,$v_i^x$ 的关联变量 $vol^{\mathrm{D}}(v_i^x)=1+4=5$.

式(42)的含义为,v_i^x 的关联变量 $len^{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 $\max\{len^{\mathrm{D}}(v_i^y),len^{\mathrm{D}}(v_j^{src})\}$. 其中,$v_i^y$ 是 v_i^x 的后继节点,v_j^{src} 是节点 v_i^x 所创建任务 τ_j 的源点. 另外,$\max\{len^{\mathrm{D}}(v_i^y),len^{\mathrm{D}}(v_j^{src})\}$ 表示在 v_i^y

的关联变量 $len^{\mathrm{D}}(v_i^y)$ 和 v_j^{src} 的关联变量 $len^{\mathrm{D}}(v_j^{src})$ 中取最大值. 例如，在图 10 中，$len^{\mathrm{D}}(v_i^y)=4$，$len^{\mathrm{D}}(v_j^{src})=1$. 因此，$v_i^x$ 的关联变量 $len^{\mathrm{D}}(v_i^x)=1+4=5$.

式(43)的含义为，v_i^x 的关联变量 $gra_{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 $\max\{\Gamma_1,\Gamma_2,\Gamma_3\}$. 其中，$\max\{\Gamma_1,\Gamma_2,\Gamma_3\}$ 表示在 Γ_1、Γ_2 和 Γ_3 中取最大值. 令 v_j^{src} 是 v_i^x 所创建任务 τ_j 的源点，v_i^y 是 v_i^x 的后继节点，参数 $\Gamma_x(x=1,2,3)$ 的含义表示如下：Γ_1 等于 v_j^{src} 的关联变量 $vol^{\mathrm{D}}(v_j^{src})/m$ 加上 v_i^y 的关联变量 $gra_{\mathrm{D}}(v_i^y)$；Γ_2 等于 v_j^{src} 的关联变量 $gra_{\mathrm{D}}(v_j^{src})$ 加上 v_i^y 的关联变量 $vol^{\mathrm{D}}(v_i^y)/m$；$\Gamma_3$ 等于 v_j^{src} 的关联变量 $gra_e(v_j^{src})$ 加上 v_i^y 的关联变量 $gra_w(v_i^y)$. 例如，在图 10 中，$\Gamma_1=4+1/m$，$\Gamma_2=1+4/m$，$\Gamma_3=1+4/m$. 因此，v_i^x 的关联变量 $gra_{\mathrm{D}}(v_i^x)=1+4+1/m=5+1/m$.

式(44)的含义为，v_i^x 的关联变量 $gra_e(v_i^x)$ 等于 v_i^x 的执行时间加上 $\max\{\Omega_1,\Omega_2\}$. 其中，$\max\{\Omega_1,\Omega_2\}$ 表示在 Ω_1 和 Ω_2 中取最大值. 令节点 v_j^{src} 是 v_i^x 所创建任务 τ_j 的源点，节点 v_i^y 是 v_i^x 的后继节点，参数 $\Omega_x(x=1,2)$ 的含义表示如下：Ω_1 等于 v_j^{src} 的关联变量 $vol^{\mathrm{D}}(v_j^{src})/m$ 加上 v_i^y 的关联变量 $gra_e(v_i^y)$；Ω_2 等于 v_j^{src} 的关联变量 $gra_e(v_j^{src})$ 加上 v_i^y 的关联变量 $gra_u(v_i^y)$. 例如，在图 10 中，$\Omega_1=4+1/m$，$\Omega_2=1+4/m$. 因此，v_i^x 的关联变量 $gra_e(v_i^x)=1+4+1/m=5+1/m$.

式(45)的含义为，若 v_i^x 是 W-节点，则 v_i^x 的关联变量 $gra_w(v_i^x)$ 等于 $gra_{\mathrm{D}}(v_i^x)$. 否则，$gra_w(v_i^x)$ 等于 v_i^y 的关联变量 $gra_w(v_i^y)$. 其中，v_i^y 是 v_i^x 的后继节点. 例如，在图 10 中，v_i^x 的关联变量 $gra_w(v_i^x)=4/m$.

式(46)的含义为，若 v_i^x 是 W-节点，则 v_i^x 的关联变量 $gra_u(v_i^x)$ 等于 $gra_e(v_i^x)$. 否则，$gra_u(v_i^x)$ 等于 v_i^y 的关联变量 $gra_u(v_i^y)$. 其中，v_i^y 是 v_i^x 的后继节点. 例如，在图 10 中，v_i^x 的关联变量 $gra_u(v_i^x)=4/m$.

③ 如果 v_i^x 创建任务 τ_j 且有后继 W-节点 v_1^y（τ_1 是 τ_i 的父任务），如图 11 所示，则按以下方法计算各参数.

$$vol^{\mathrm{D}}(v_i^x)=c(v_i^x)+vol^{\mathrm{D}}(v_j^{src}) \tag{47}$$

$$len^{\mathrm{D}}(v_i^x)=c(v_i^x)+\max\{len^{\mathrm{D}}(v_j^{src}),len^{\mathrm{D}}(v_1^y)\} \tag{48}$$

$$gra_{\mathrm{D}}(v_i^x)=c(v_i^x)+\max\{\Gamma_1,gra_{\mathrm{D}}(v_j^{src})\} \tag{49}$$

其中，

$$\Gamma_1=vol^{\mathrm{D}}(v_j^{src})/m+len^{\mathrm{D}}(v_1^y),$$

$$gra_e(v_i^x)=c(v_i^x) \tag{50}$$

$$gra_w(v_i^x)\begin{cases}gra_{\mathrm{D}}(v_i^x), & \text{若 } v_i^x \text{ 是 W-节点}\\ -\infty, & \text{否则}\end{cases} \tag{51}$$

$$gra_u(v_i^x)\begin{cases}gra_e(v_i^x), & \text{若 } v_i^x \text{ 是 W-节点}\\ -\infty, & \text{否则}\end{cases} \tag{52}$$

图 11 给出了式(47)～(52)的解释. 如图 11 所示，最外层的任务 τ_1 包含一个 T-节点 v_1^z 和一个 W-节点 v_1^y. 其中，T-节点 v_1^z 创建了 τ_1 的子任务 τ_i. τ_i 包含一个 T-节点 v_i^x，T-节点 v_i^x 创建了 τ_i 的子任务 τ_j. τ_j 包含一个以 v_j^w 为入口的条件分支结构. DAG 图中所有节点的执行时间均为 1.

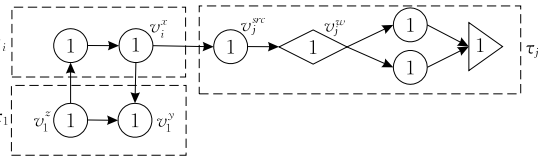


图 11　式(47)～(52)的解释示例

式(47)的含义为，v_i^x 的关联变量 $vol^{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 v_j^{src} 的关联变量 $vol^{\mathrm{D}}(v_j^{src})$. 其中，$v_j^{src}$ 是 v_i^x 所创建任务 τ_j 的源点. 例如，在图 11 中，v_i^x 的关联变量 $vol^{\mathrm{D}}(v_i^x)=1+4=5$.

式(48)的含义为，v_i^x 的关联变量 $len^{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 $\max\{len^{\mathrm{D}}(v_j^{src}),len^{\mathrm{D}}(v_1^y)\}$. 其中，$v_j^{src}$ 是 v_i^x 所创建任务 τ_j 的源点，v_1^y 是 W-节点且为 v_i^x 的后继节点. 另外，$\max\{len^{\mathrm{D}}(v_j^{src}),len^{\mathrm{D}}(v_1^y)\}$ 表示在 v_j^{src} 的关联变量 $len^{\mathrm{D}}(v_j^{src})$ 和 v_1^y 的关联变量 $len^{\mathrm{D}}(v_1^y)$ 中取最大值. 例如，在图 11 中，$len^{\mathrm{D}}(v_j^{src})=4$，$len^{\mathrm{D}}(v_1^y)=1$. 因此，$v_i^x$ 的关联变量 $len^{\mathrm{D}}(v_i^x)=1+4=5$.

式(49)的含义为，v_i^x 的关联变量 $gra_{\mathrm{D}}(v_i^x)$ 等于 v_i^x 的执行时间加上 $\max\{\Gamma_1,gra_{\mathrm{D}}(v_j^{src})\}$. 其中，$\max\{\Gamma_1,gra_{\mathrm{D}}(v_j^{src})\}$ 表示在 Γ_1 和节点 v_j^{src} 的关联变量 $gra_{\mathrm{D}}(v_j^{src})$ 中取最大值. 令 v_j^{src} 是 v_i^x 所创建任务 τ_j 的源点，v_1^y 是 W-节点且为 v_i^x 的后继节点，参数 Γ_1 等于 v_j^{src} 的关联变量 $vol^{\mathrm{D}}(v_j^{src})/m$ 加上 v_1^y 的关联变量 $len^{\mathrm{D}}(v_1^y)$. 例如，在图11中，$\Gamma_1=1+4/m$，$gra_{\mathrm{D}}(v_j^{src})=4$. 因此，$v_i^x$ 的关联变量 $gra_{\mathrm{D}}(v_i^x)=1+4=5$.

式(50)的含义为，v_i^x 的关联变量 $gra_e(v_i^x)$ 等于 v_i^x 的执行时间. 例如，在图 11 中，$gra_e(v_i^x)=1$.

式(51)的含义为，若 v_i^x 是 W-节点，则 v_i^x 的关联变量 $gra_w(v_i^x)$ 等于 $gra_{\mathrm{D}}(v_i^x)$. 否则，$gra_w(v_i^x)$ 等于 $-\infty$. 例如，在图 11 中，v_i^x 的关联变量 $gra_w(v_i^x)=-\infty$.

式(52)的含义为,若 v_i^x 是 W-节点,则 v_i^x 的关联变量 $gra_u(v_i^x)$ 等于 $gra_e(v_i^x)$. 否则,$gra_u(v_i^x)$ 等于 $-\infty$. 例如,在图 11 中,v_i^x 的关联变量 $gra_u(v_i^x)=-\infty$.

时间复杂度分析. 本节的算法具有多项式时间计算复杂度,其理由如下:首先,本节算法为图 G 中的每个节点计算 6 个参数,故需要计算的参数总个数为 $6n$(n 为图 G 中的节点个数). 其次,每个节点上参数的计算直接利用该节点后继节点的参数计算结果. 根据式(11)~(52),每个参数的计算时间为 $O(\delta)$,其中,δ 为节点的最大出度. 综上,本节算法总的计算复杂度为 $O(n\delta)$.

6.3 算法 1 的误差分析

本节介绍算法 1 在图 5 示例上响应时间的计算过程,并比较文献[20]算法与算法 1 的精确度.

如图 5 所示,v_i^x 有两个直接后继节点 v_j^{src} 和 v_i^y. 其中,v_j^{src} 是 v_i^x 所创建任务 τ_j 的源点. 根据式(43),为了计算 v_i^x 的关联变量 $gra_D(v_i^x)$,需要预先计算 v_j^{src} 的关联变量 $vol^D(v_j^{src})$、$gra_D(v_j^{src})$ 和 $gra_e(v_j^{src})$ 以及 v_i^y 的关联变量 $vol^D(v_i^y)$、$gra_D(v_i^y)$ 和 $gra_w(v_i^y)$. 如图 5 所示,v_j^{src} 仅有一个直接后继节点 v_i^z,并且 v_i^z 为 W-节点. 根据式(29)、(31)、(32),可以计算以下三个参数 $vol^D(v_j^{src})=0$、$gra_D(v_j^{src})=0$ 和 $gra_e(v_j^{src})=0$. 如图 5 所示,v_i^y 有两个直接后继节点且为条件分支结构的入口节点. 根据式(35)、(37)、(39),可以计算以下三个参数 $vol^D(v_i^y)=mL$、$gra_D(v_i^y)=L+(1-1/m)$ 和 $gra_w(v_i^y)=L$. 进一步地,可计算式(43)中的中间变量 $\Gamma_1=L+1-1/m$,$\Gamma_2=L$ 和 $\Gamma_3=L$. 因此,$gra_D(v_i^x)=L+(1-1/m)$.

根据 5.2 节可知,相较于图 5 的真实响应时间界($L+(1-1/m)$),文献[20]算法求得的响应时间界误差为 $(L-1)(1-1/m)$. 根据上文可知,本文算法求得的响应时间界为 $L+(1-1/m)$,即真实响应时间界. 因而,文献[20]算法仅能求得近似解,而本文算法能够得到精确解.

7 实验结果

本节实现了文献[20]的算法(记为 Alg0)和本文第 6 节的算法(记为 Alg1). 实验所使用的微型计算机为 DELL OptiPlex 3050 台式机,硬件配置为 Intel(R) Core(TM) i5-7500 CPU@3.40GHz,8GB DDR4 内存(7.89 GB 可用),操作系统为 Microsoft Windows10 64 位. 利用 Python 语言进行编程,程序运行环境为 Python3.7,并利用 Python 模块 matplotlib 绘制实验图. 通过随机生成的算例,对算法求得的响应时间界和计算时间进行了评估. 随机算例的生成程序描述如下.

随机生成 n 个 con-DAG 任务图 $\{G_1,\cdots,G_n\}$. 每个任务图在[10,40]范围内随机选择非条件节点个数. 每个非条件节点的最坏情况执行时间的取值范围为[1,100]. 此外,每个任务图中条件节点的执行时间为 0.

对于每个任务图,按照从源点到汇点的顺序生成图中的节点. 每个新生成的节点 v_i^x 以 P_{if} 的概率确定为条件节点. 如果 v_i^x 为条件节点,则生成 v_i^x 对应的条件分支结构. 该条件分支结构包含一个入口节点(即节点 v_i^x)、两个条件分支和一个出口节点. 如果 v_i^x 不是条件节点,则将其随机插入到任务图现有的条件分支中. 对于每个非条件节点 v_i^x,以 P_{cre} 的概率将 v_i^x 确定为 T-节点,以 P_{wait} 的概率将 v_i^x 确定为 W-节点,以 P_{nor} 的概率将 v_i^x 确定为 N-节点,并且保证 $P_{nor}+P_{cre}+P_{wait}=1$.

对于 τ_i 中任意的 T-节点 v_i^x,在任务集 $\{\tau_{i+1},\cdots,\tau_n\}$ 中选择一个尚未分配父任务的 τ_j($i<j\leq n$),确定 τ_i 是 τ_j 的父任务,即构造从 v_i^x 到 τ_j 源点 v_j^{src} 的 T-边. 如果不存在这样的 τ_j,则调整 v_i^x 为 N-节点. 对于 W-节点 v_i^x,设 v_i^y 为 v_i^x 的前驱 T-节点,假设 τ_j 是由 v_i^y 创建的任务,从 τ_j 的汇点到 W-节点 v_i^x 添加 W-边.

图 12~图 16 针对不同的参数组合进行实验,参数配置如图例所示. 其中,m 表示线程数,n 表示任务数,P_{if} 表示条件节点的生成概率,P_{wait} 表示 W-节点的生成概率,P_{cre} 表示 T-节点的生成概率. 图 12~图 16 的(a)中,灰色柱状图代表 Alg0 求得的响应时间界的平均值 R_0;白色柱状图代表 Alg1 求得的响应时间界的平均值 R_1;箱型图展示了 Alg0 和 Alg1 的响应时间界差值(Gap)的分布. 具体来说,箱型图对响应时间界的差值按照从小到大排序,箱型图的顶表示第 75%的数据;箱型图的底表示第 25%的数据;箱型图顶和底之间的区域表示 25%~75%的数据;中间浅线表示中位数,即第 50%的数据;上须线表示 Gap 的最大值;下须线表示 Gap 的最小值. 图 12~图 16 的(b)中,实折线代

表 Alg0 求得的计算时间；虚折线代表 Alg1 求得的计算时间；箱型图展示了 Alg1 和 Alg0 的计算时间比值的分布. 具体来说，箱型图对计算时间的比值按照从小到大排序，箱型图的顶表示第 75％的数据；箱型图的底表示第 25％的数据；箱型图顶和底之间的区域表示 25％～75％的数据；中间浅线表示中位数，即第 50％的数据；上须线表示计算时间比值的最大值；下须线表示计算时间比值的最小值. 对于每种参数组合，进行 1000 次随机实验. 实验表明，Alg1 能够改进响应时间界的精度，与 Alg0 的平均界差可达 176. 另外，针对所有算例，Alg1 的计算时间在 14.3 ms 之内，略高于 Alg0.

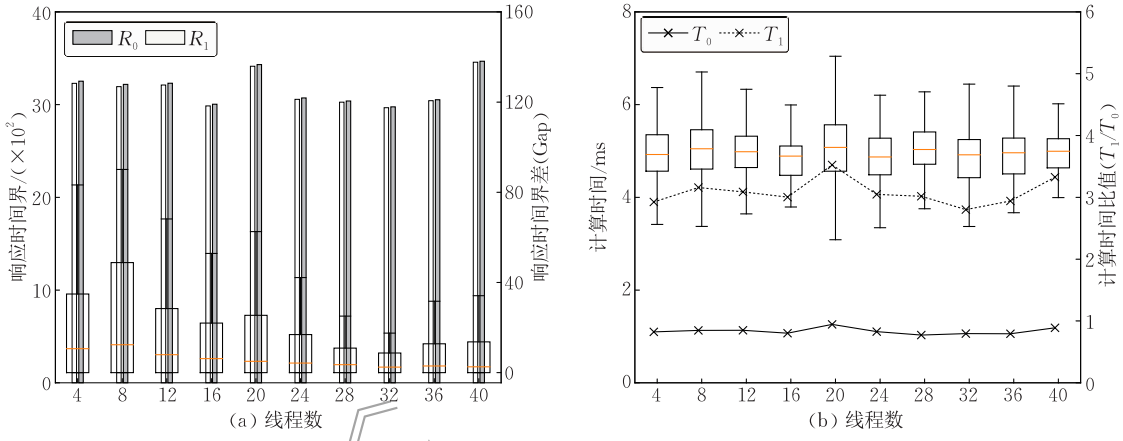


图 12　$n=10$，$P_{if}=0.3$，$P_{wait}=0.3$，$P_{cre}=0.3$

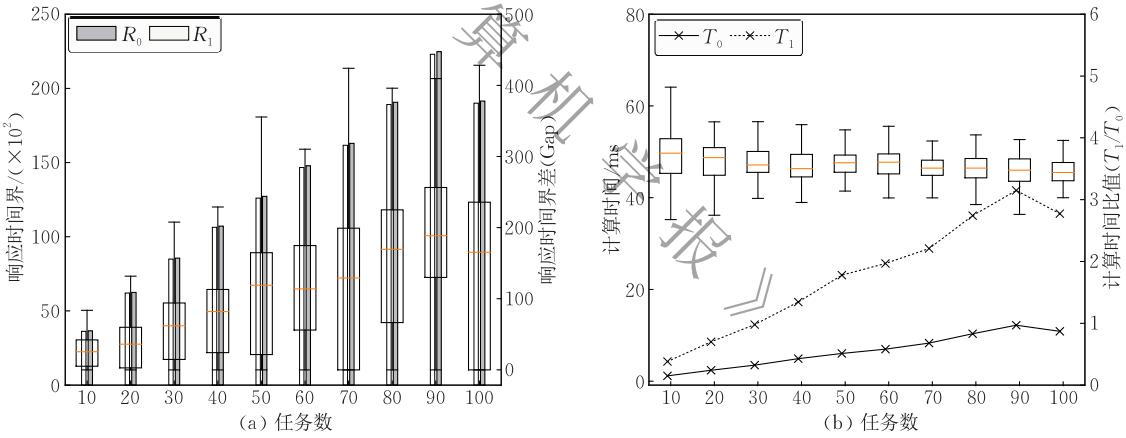


图 13　$m=4$，$P_{if}=0.3$，$P_{wait}=0.3$，$P_{cre}=0.3$

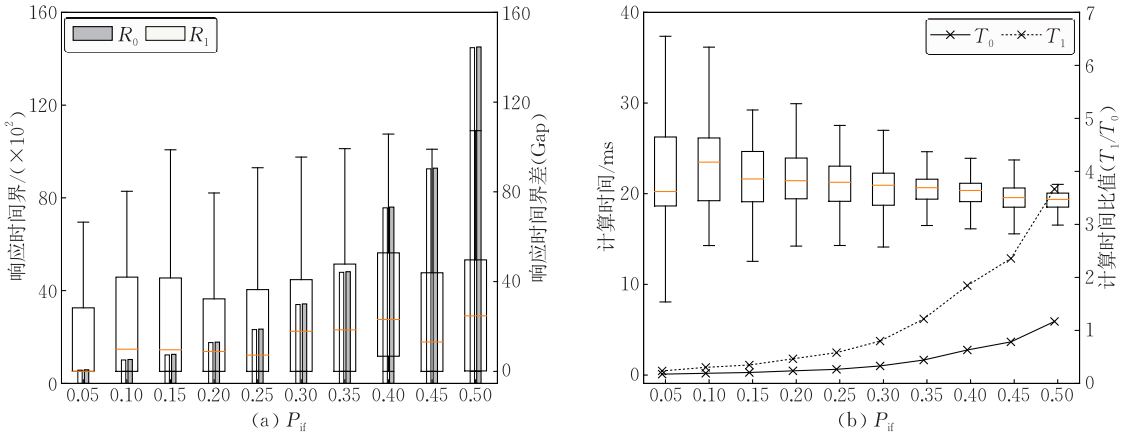


图 14　$m=4$，$n=10$，$P_{wait}=0.3$，$P_{cre}=0.3$

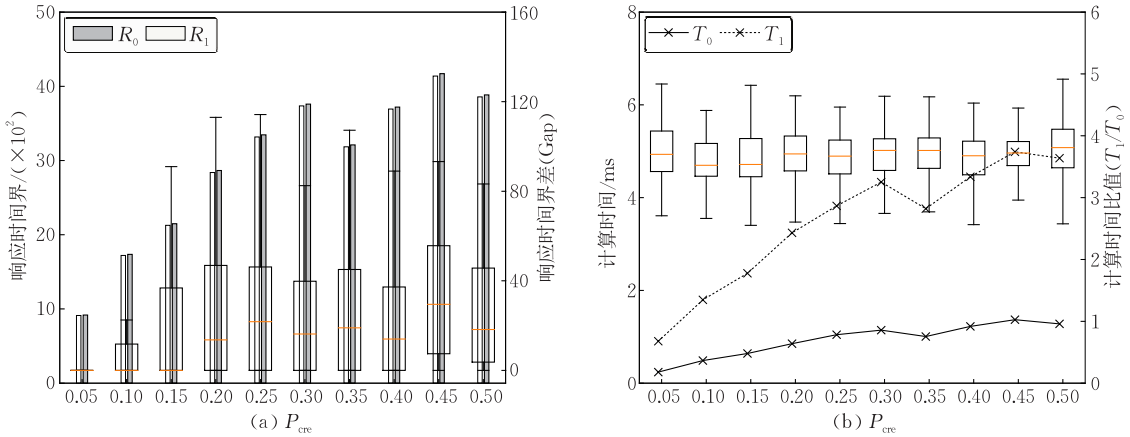


图 15 $m=4, n=10, P_{if}=0.3, P_{wait}=0.3$

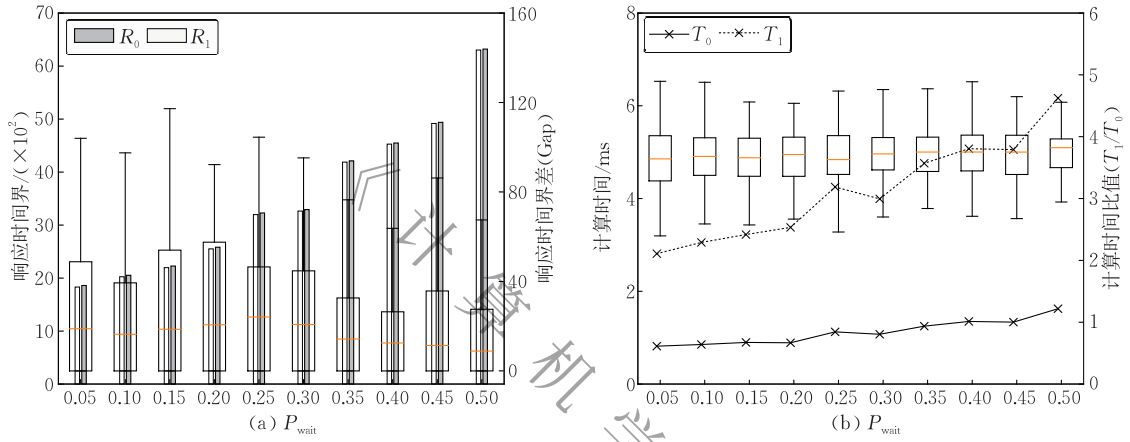


图 16 $m=4, n=10, P_{if}=0.3, P_{cre}=0.3$

如图 12 所示,实验数据表明,硬件平台上处理器核数会影响 Alg0 与 Alg1 求得的 Gap. 在一般情况下,随着核数的增多,两个算法求得的 Gap 越小. 根据 Graham 界式(3),核数越多,DAG 图的最大负载对于响应时间的影响越弱. 因此,该实验现象表明,相对于提升 DAG 图最长路径长度的评估精度而言,Alg1 能够更有效地改善 DAG 图最大负载的评估精度. 另外,Alg1 与 Alg0 的计算时间比值不超过 3.5.

如图 13 所示,Gap 随着任务数的增长而增大. 具体原因如下:随着任务数的增多,有向无环图的总负载增大. 由于 Alg1 能够更加精确地估计有向无环图的最大负载量(如图 12 所示),因此,Gap 随着任务数增长呈递增趋势. 另外,Alg1 与 Alg0 的计算时间的比值略微减少.

如图 14 所示,随着 P_{if}的增大,Gap 具有增大的趋势. 具体原因如下:P_{if}的值越大,有向无环图中条件分支嵌套结构越复杂,越有可能产生有利于 Alg1 的子结构(如 5.2 节中的反例). 另外,Alg1 与 Alg0 的计算时间比值呈递减趋势.

如图 15 所示,随着 P_{cre}的增大,Gap 呈现递增趋势. 具体原因如下:根据随机算例程序,P_{cre}越大,单个任务的子任务就越多,这导致了较大的任务并行度. 在高并行度的算例(如 5.2 节的反例)中更能体现 Alg1 的优越性. 另外,Alg1 与 Alg0 的计算时间呈现线性增长趋势.

如图 16 所示,随着 P_{wait}的增大,Gap 略微减少. 具体原因如下:P_{wait}的值越大,有向无环图的最长路径长度越长. 根据 Graham 界式(3),最长路径长度越长,DAG 图的最长路径长度对响应时间的影响越强. 因此,实验现象表明,对于响应时间精度的影响而言,提升 DAG 图最长路径长度的评估精度并没有改善 DAG 图的最大负载精度明显(如图 12 所示). 另外,Alg1 与 Alg0 的计算时间呈现线性增长的趋势.

实验数据表明,一方面,与 Alg1 相比,Alg0 需要更少的计算时间(平均意义下,Alg0 能够节省 2.5 倍的计算时间),并且能够尽可能高地保障响

应时间界的精度.这说明 Alg0 具有易实现、鲁棒的优点.

另一方面,相较于 Alg0,Alg1 能够有效提高响应时间界精度.在平均情况下,响应时间界精度能够提高 3%.能够有效改善响应时间界精度,对实时系统是至关重要的.原因如下:在实时系统里,响应时间直接影响着系统的可调度性.Alg1 在平均响应时间界精度上提高 3%,这会增强系统可调度的概率.值得注意的是,在实际系统中,通常存在多个 DAG 任务并行工作的情况.尽管对于单个 DAG 任务,Alg1 带来的精度提升有限(通常是原响应时间界的 3%),但是,对于包含多个 DAG 任务的实时系统而言,如果每个 DAG 任务的响应时间界精度提升 3%,其积累的精度也是很可观的.

另外,与 Alg0 相比,Alg1 的计算时间有所增加.但是,这种增加并不会破坏实时系统的实时性.具体原因如下:在实时系统里,系统计算性能和运行时间是非常重要的.其中,系统运行时间指的是在系统部署之后运行时的时间.本文方法应用于系统部署之前,对系统进行离线分析和验证.因此,Alg1 的计算时间与系统部署之后实际运行的时间没有必然联系.

综上所述,尽管 Alg1 增加了计算时间,但是提升了响应时间界的分析精度.这对于保障系统的安全性来说是值得的:对于可调度性处于临界状态的实时系统而言,响应时间精度有 1%的提升,都有可能极大地增加系统可调度的概率.

8　总结与展望

OpenMP 是实时系统最有希望的多核并行编程框架之一.近年来,实时系统领域的研究热点围绕 OpenMP 的 DAG 任务图展开.当前,DAG 任务图的研究很少考虑 OpenMP 程序的条件分支特征.尚未有研究能够精确计算出带有条件分支结构 OpenMP 任务图的响应时间界.本文致力于研究 OpenMP 的并行结构和分支结构特征,基于动态规划技术提出了能够求解带有条件分支 OpenMP 任务图响应时间界的精确算法.并且,本文方法具有多项式时间的计算复杂度.下一步的工作将围绕 OpenMP 程序更复杂的实际特征展开,例如,程序中的循环和递归特征.为带有条件分支和循环递归结构 OpenMP 程序的响应时间分析提供新的理论和方法.

参　考　文　献

[1] Board O A R. OpenMP application program interface version 3.0. Barcelona, Spain: Barcelona Supercomputing Center, Technology Report: 01, 2008

[2] Saifullah A, Li J, Agrawal K, et al. Multi-core real-time scheduling for generalized parallel task models. Real-Time Systems, 2013, 49(4): 404-435

[3] Li J, Luo Z, Ferry D, et al. Global EDF scheduling for parallel real-time tasks. Real-Time Systems, 2015, 51(4): 395-439

[4] Ferry D, Li J, Mahadevan M, et al. A real-time scheduling service for parallel tasks//Proceedings of the 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS). Philadelphia, USA, 2013: 261-272

[5] Saifullah A, Ferry D, Li J, et al. Parallel real-time scheduling of DAGs. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(12): 3242-3252

[6] Baruah S. Improved multiprocessor global schedulability analysis of sporadic DAG task systems//Proceedings of the 2014 26th Euromicro Conference on Real-Time Systems. Madrid, Spain, 2014: 97-105

[7] Qamhieh M, Fauberteau F, George L, Midonnet S. Global EDF scheduling of directed acyclic graphs on multiprocessor systems//Proceedings of the 21st International Conference on Real-Time Networks and Systems. New York, USA, 2013: 287-296

[8] Qamhieh M, George L, Midonnet S. A stretching algorithm for parallel real-time DAG tasks on multiprocessor systems//Proceedings of the 22nd International Conference on Real-Time Networks and Systems. New York, USA, 2014: 13

[9] Serrano M A, Melani A, Vargas R, et al. Timing characterization of OpenMP4 tasking model//Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES). Amsterdam, Netherlands, 2015: 157-166

[10] Vargas R, Quinones E, Marongiu A. OpenMP and timing predictability: A possible union?//Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition. San Jose, USA, 2015: 617-620

[11] Sun Jinghao, Guan Nan, Wang Yang, et al. Real-time scheduling and analysis of OpenMP task systems with tied tasks//Proceedings of the Real-Time Systems Symposium (RTSS). Pairs, France, 2017: 92-103

[12] Zhou Jia-Xiang, Zheng Wei-Min. A static scheduling algorithm on DAG partition-reconfiguration in the network of workstations. Journal of Software, 2000, 11(8): 1097-1104 (in Chinese)
(周佳祥,郑纬民. 基于 DAG 图解—重构的机群系统静态调度算法. 软件学报, 2000, 11(8): 1097-1104)

[13] Fonseca J C, Nélis V, Raravi G, Pinho L M. A multi-DAG model for real-time parallel applications with conditional execution//Proceedings of the 30th Annual ACM Symposium on Applied Computing. New York, USA, 2015: 1925-1932

[14] Baruah S, Bonifaci V, Marchetti-Spaccamela A. The global EDF scheduling of systems of conditional sporadic DAG tasks//Proceedings of the 2015 27th Real-Time Systems (ECRTS). Lund, Sweden, 2015: 222-231

[15] Baruah S. The federated scheduling of systems of conditional sporadic DAG tasks//Proceedings of the 12th International Conference on Embedded Software. NJ, USA, 2015: 1-10

[16] Melani A, Bertogna M, Bonifaci V, et al. Response-time analysis of conditional DAG tasks in multiprocessor systems//Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems. Lund, Sweden, 2015: 211-221

[17] Melani A, Bertogna M, Bonifaci V, et al. Schedulability analysis of conditional parallel task graphs in multicore systems. IEEE Transactions on Computers, 2017, 66(2): 339-353

[18] Sun J, Guan N, Wang Y, et al. Feasibility of fork-join real-time task graph models: Hardness and algorithms. ACM Transactions on Embedded Computing Systems, 2016, 15(1): 14

[19] Graham R L. Bounds for certain multiprocessing anomalies. Bell System Technical Journal, 1966, 45(9): 1563-1581

[20] Sun J, Guan N, Sun J, Chi Y. Calculating response-time bounds for OpenMP task systems with conditional branches //Proceedings of the 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). Montreal, Canada, 2019: 169-181

[21] Liu X, Mellor-Crummey J, Fagan M. A new approach for performance analysis of OpenMP programs//Proceedings of the 27th International ACM Conference on International Conference on Supercomputing. New York, USA, 2013: 69-80

[22] Eichenberger A E, Mellor-Crummey J, Schulz M, et al. OMPT: An OpenMP tools application programming interface for performance analysis//Proceedings of the International Workshop on OpenMP. Canberra, Australia, 2013: 171-185

[23] Huck K A, Malony A D, Shende S, Jacobsen D W. Integrated measurement for cross-platform OpenMP performance analysis//Proceedings of the International Workshop on OpenMP. Salvador, Brasil, 2014: 146-160

[24] Dietrich R, Schmitt F, Grund A, Schmidl D. Performance measurement for the OpenMP 4.0 offloading model//Proceedings of the European Conference on Parallel Processing. Porto, Portugal, 2014: 291-301

[25] Frigo M, Leiserson C E, Randall K H. The implementation of the Cilk-5 multithreaded language. ACM SIGPLAN Notices, 1998, 33(5): 212-223

[26] Narlikar G J. Scheduling threads for low space requirement and good locality. Theory of Computing Systems, 2002, 35(2): 151-187

SUN Jing-Hao, Ph. D., associate professor. His research interests include real-time system scheduling and parallel program analysis.

ZHANG Li-Wei, M. S. candidate. His research interests include OpenMP parallel program analysis.

CHI Yao-Yao, M. S. candidate. Her research interests include real-time parallel system scheduling.

CAO Lei, B. S. Her research interests include real-time system and optimization algorithm.

DENG Qing-Xu, Ph. D., professor. His research interest is real-time system.

Background

Nowadays, with the development of multicores, high-performance and real-time of software have drawn increasing interests in embedded and real-time computing systems. OpenMP can fully reflect the parallel nature of real-time computing, and is one of the most promising the de facto standard for building next-generation real-time embedded systems. OpenMP real-time scheduling theory breaks through they have not been resolved. This subject is to study the difficult problems in OpenMP real-time scheduling. In the real-time field, a directed acyclic graph (DAG) is the natural model for characterizing OpenMP programs. However, DAGs cannot accurately describe OpenMP programs because OpenMP programs are not only parallel, but also include conditional branches (such as if-else statement). Therefore, research on the conditional DAG (con-DAG) task graph is still open.

Existing DAG task models assume well-nested structures recursively composed by single-source-single-sink parallel and conditional components. However, con-DAGs in general do not comply with this assumption, and existing work

cannot fully describe and analyze the behaviors of realistic OpenMP programs. Therefore, this paper will model and analyze the behaviors of OpenMP task systems with general branching structures.

This paper aims to analyze the response time of OpenMP programs with conditional branch structure. A naive solution is calculating the response time by enumerating all possible execution flows in the con-DAGs, but it has exponential time complexity. Therefore, existing works develop polynomial-time dynamic programming algorithms to calculate response time bounds of con-DAGs. However, these methods assume structures recursively composed by single-source-single-sink parallel and conditional components, so they cannot analyze general OpenMP task systems that do not comply with this assumption. This paper studies general OpenMP task systems, and proposes an algorithm for calculating the response time of con-DAG task graph with "multi-source-multi-sink" conditional structures. Compared with existing methods, The algorithm in this paper not only can accurately calculate the response time bounds of con-DAG tasks, but also has polynomial time complexity.

This work was supported by the National Natural Science Foundation of China (No. 61972076), which aims to analyze the theory, models, algorithms, and applications of real-time scheduling for OpenMP task graph.

Our research group has been working on real-time embedded systems, multi-core real-time scheduling, network optimization, and parallel computing. Related works have been published in reputable journals and conferences, such as Chinese Journal of Computers, IEEE Trans on Computers, IEEE Trans on CAD, ACM Trans on ECS, RTSS, DAC, RTAS, EMSOFT, DATE, etc.