带优先级DAG实时任务图模型的响应时间分析

李 峰[1)] 毕 冉[2),3)] 马 野[1)] 孙景昊[1)] 李西盛[1)] 邓庆绪[4)]

[1)](大连理工大学计算机科学与技术学院 辽宁 大连 116024)

[2)](东北大学秦皇岛分校计算机与通信工程学院 河北 秦皇岛 066004)

[3)](东北大学秦皇岛分校河北省海洋感知网络与数据处理重点实验室 河北 秦皇岛 066004)

[4)](东北大学计算机科学与工程学院 沈阳 110819)

摘 要 随着多核技术在实时嵌入式系统中的广泛应用，多核处理器已经成为主流的硬件平台，充分发挥多核处理器的计算能力需要实现对实时程序进行全面的并行化.有向无环图(DAG)是用于描述并行实时程序的理论模型，可描绘复杂任务的细粒度并行性.任务内优先级分配可以减少DAG任务运行时行为的不确定性，获得更小的最坏情况响应时间(WCRT).现有优先级DAG任务的响应时间分析都是关于DAG任务最坏情况响应时间界限的研究，因其与实际的最坏情况响应时间存在较大差距而存在悲观性，限制了实时嵌入式系统的计算性能，使其占用更多计算资源以确保任务在截止时间内完成.本文针对具有优先级的DAG任务的响应时间分析问题，提出了一种基于可满足性模理论(SMT)的方法来计算DAG任务精确的最坏情况响应时间.尽管已有研究给出关于DAG任务精确的WCRT，但并不适用于具有优先级的DAG.本文将带有优先级DAG任务的响应时间分析问题形式化为混合逻辑公式的可满足性问题，从而获得精确的最坏情况响应时间.实验结果表明，本文提出的方法不仅能够保证WCRT的精度，而且与现有DAG任务精确WCRT的计算方法相比，本文方法的计算效率平均提升了50%.

关键词 响应时间；可满足性模理论；优先级调度；有向无环图；并行调度

中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2024.02909

Response Time Analysis for Prioritized DAG Task

LI Feng[1)] BI Ran[2),3)] MA Ye[1)] SUN Jing-Hao[1)] LI Xi-Sheng[1)] DENG Qing-Xu[4)]

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

[2)](*School of Computer and Communication Engineering, Northeastern University at Qinhuangdao, Qinhuangdao, Hebei* 066004)

[3)](*Hebei Key Laboratory of Marine Perception Network and Data Processing, Northeastern University at Qinhuangdao, Qinhuangdao, Hebei* 066004)

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

Abstract With the widespread application of multi-core technology in real-time embedded systems, multi-core processors have become the mainstream hardware platform. To fully utilize the powerful computational capabilities of multi-cores, comprehensive parallelization must be implemented in real-time programs. Directed Acyclic Graph (DAG) describes the parallelism of real-time programs which can effectively depict the fine-grained parallel characteristics of complex tasks. Intra-task Priority Assignment can reduce the uncertainty in the runtime behavior of DAG tasks, resulting in a smaller Worst-Case Response Time (WCRT). Existing response time

收稿日期：2024-02-06；在线发布日期：2024-09-12.本课题得到国家自然科学基金(62472063，62072085)和河北省自然科学基金(F2024501037)资助.**李 峰**，博士研究生，主要研究领域为实时系统调度.E-mail：lifeng@mail.dlut.edu.cn. **毕 冉**，博士，副教授，主要研究领域为实时系统调度、调度优化.E-mail：biranhit@163.com. **马 野**，博士，助理教授，主要研究领域为实时系统调度. **孙景昊**(通信作者)，博士，副教授，中国计算机学会(CCF)会员，主要研究领域为实时系统调度、并行程序分析.E-mail：jhsun@dlut.edu.cn. **李西盛**，硕士研究生，主要研究领域为实时系统调度、算法优化. **邓庆绪**，博士，教授，主要研究领域为实时系统调度.

analyses for priority-based DAG tasks focus on the upper bound of the WCRT. However, these analyses tend to be pessimistic due to the gap between the upper bound and the exact WCRT. This pessimism limits the computational performance of real-time embedded systems, causing them to occupy more computational resources to ensure tasks are completed within their deadlines. This paper focuses on response time analysis of prioritized DAG tasks and proposes an improved method that employs Satisfiability Modulo Theories (SMT) to compute a more accurate upper bound for DAG task response time. Although previous studies have provided exact WCRT analysis for DAG tasks, these findings do not extend to DAGs with priorities, and existing methods still exhibit pessimism. This paper formalizes the response time analysis of priority-based DAG tasks into a satisfiability problem of mixed logical formulas and gets the exact WCRT. Experimental work shows that the method proposed in this paper not only guarantees the precision of the obtained bounds but also improves the average computational efficiency by 50% compared to existing methods for accurately calculating the WCRT of DAG tasks.

Keywords response time; satisfiability modulo theories; priority scheduling; directed acyclic graph; parallel scheduling

1 引 言

多核处理器已成为实时嵌入式系统的主流硬件平台. 为了充分发挥多核处理器的计算能力，需要对实时软件进行全面的并行化，包括任务间并行和任务内并行. 任务内并行的实现能够让单个任务(并行程序的抽象)在多个内核上同时执行. 对于计算性能和响应时间有严格要求的实时嵌入式系统，任务内并行化尤为重要. 许多现代并行编程框架，如Cilk[1]、OpenMP[2]和Intel的TBB[3]，通常支持任务内并行. 然而，目前的软件和调度算法尚不能充分利用多核处理器的性能.

有向无环图(Directed Acyclic Graph，DAG)用于描述任务间的依赖关系，是刻画复杂任务细粒度并行性的天然模型. DAG被广泛应用于任务内并行建模，涌现出大量基于DAG模型实时调度和分析的研究工作[4-10]. 响应时间分析是实时嵌入式系统中判定任务可调度性的重要手段. DAG任务响应时间分布如图1所示，其中BCRT代表最好情况响应时间，WCRT代表最坏情况响应时间.

Graham首次提出了单DAG任务的WCRT界限，称为Graham界[4]. 大量多核调度方法直接或间接地受到了Graham界的启发，并将Graham界拓展到更多应用中，如OpenMP程序[11-15]和异构多核平台[16-17]. 优先级技术是指将为DAG任务节点分配优先级，以控制节点的执行顺序和任务的运行时行为. 研究表明，任务内优先级的分配可以缩短任务的响应时间，有利于系统的可调度性[18-20]. 文献[18]通过对BOTS[21]、SPEC2012[22]、Dash[23-25]等基准的评估，进一步证明了任务内优先级分配在现实应用中的有效性. 近年来，许多研究工作应用优先级技术调度DAG任务，旨在减小WCRT界限. 文献[18]通过为DAG任务的节点分配不同的优先级来决定节点的执行顺序，给出了一个优于Graham界的WCRT界限. WCRT界限为DAG任务的响应时间提供了一个安全的估计，但如图1所示，WCRT界限可能被估计过高，且过于悲观. 为了保证在截止时间内完成计算任务，悲观的WCRT界限导致DAG任务占用较多的计算资源，降低了实时嵌入式系统的计算性能. 文献[26]给出DAG任务精确的响应时间界限，但并未考虑节点之间的优先级，仍具有相当的悲观性.

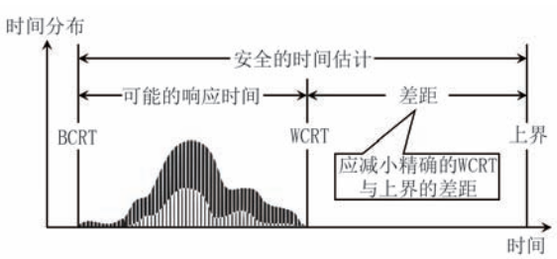

图1 DAG任务的响应时间分布示例

本文旨在分析带有优先级DAG实时任务的响应时间，利用可满足性模理论技术(Satisfiability Modulo Theories，SMT)计算DAG精确的最坏情况响应时间，将优先级DAG任务响应时间求解问

题形式化为混合逻辑公式的可满足性判定问题. 本文方法优势在于，针对带有优先级DAG任务，在保证WCRT计算精度的前提下，大大缩短了方法的计算时间. 实验结果表明，本文方法求解的WCRT界限在保证精度的同时，所需计算时间也更少.

2　相关工作

优先级技术是改善DAG任务响应时间界限的有效方法. 通过为任务节点分配适当的优先级，可以有效优化任务调度并减少任务的响应时间. 近年来，涌现出许多优先级DAG任务的研究，提出了大量调度算法和优化方法. 本文旨在探讨优先级DAG任务的响应时间分析问题，采用基于任务内优先级分配的方法，结合SMT技术计算DAG任务的精确WCRT. 优先级DAG任务的代表性研究总结如下.

文献[27-30]研究的是全局固定优先级调度算法下的DAG任务. 2015年，Burmyakov[27]在全局固定优先级调度算法下对多核平台上多DAG任务可调度性测试的性能进行了评估，通过为节点分配优先级证实了其具有局限性. 然后，为了克服现有DAG任务精确可调度性测试的主要缺点，即计算时间和内存消耗大，又提出了一种改进的精确测试. 但该算法在时间上是指数级的，在空间上是多项式级别的. 2017年，Pathan[28]提出了一种DAG任务调度的两级抢占式全局固定优先级策略，利用任务内优先级分配提高资源利用率，降低了任务内节点间的干扰. 同年，Voudouris[29]提出了一种无异常动态调度方法，通过给任务分配固定优先级来优化多核架构中的并行处理，从而避免了因负载不均衡或时间异常导致的响应时间分析上的悲观估计问题. 2019年，Yalcinkaya[30]在全局固定优先级调度算法下，针对多核平台上的偶发周期性任务，提出了一种精确的可调度性分析方法.

文献[31-33]研究的是划分固定优先级调度算法下的DAG任务. Fonseca[31]和Casini[32]分别针对DAG任务，研究了划分固定优先级调度算法下DAG任务的可调度性分析问题，Fonseca的研究对象是多核平台上的自悬挂DAG任务，Casini假设的是偶发式DAG任务具有不可抢占的性质. 通过为每个节点分配固定的优先级，并行处理和计算任务的WCRT. Slim[33]利用划分固定优先级调度算法研究了多核平台上DAG任务的概率响应时间分析问题，提出了一种子任务优先级分配算法，在调度时不仅计算了不同内核上子任务的通信开销，而且考虑了DAG图不同节点之间的执行顺序，获得了更小的DAG任务的响应时间界限.

以上研究重点集中于固定优先级DAG任务的响应时间分析与可调度性分析方面. 接下来介绍任务内优先级分配的相关研究.

文献[18-20]研究的是带有优先级的一般类型DAG任务的响应时间分析问题. 2019年，He[18]提出了带有优先级DAG任务的第一个WCRT界限，并给出了一种任务内优先级分配方法，利用该方法可以为DAG任务的节点分配优先级，控制节点的执行顺序和运行时行为，并在多项式时间内计算出DAG任务的WCRT界限. 然而，He的方法受到任务内优先级必须符合图的拓扑顺序的限制(即节点的优先级不能高于其任何前驱节点). 对于没有此限制的大范围优先级分配，He方法可能会得到错误的界限. 2020年，Zhao[19]进一步改进和扩展了文献[18]的结果，探讨了DAG任务结构中的并行性和依赖性，并提出了基于并发provider和consumer模型的优先级分配策略和响应时间界限. 但该模型和算法依然未突破图拓扑顺序的限制. 2021年，He[20]放宽了文献[18]中的约束，提出了一种处理任意优先级分配的多项式时间WCRT界限计算方法，用于任意节点的排序规则，无须受限于图的拓扑顺序，进一步增加了任务的可调度性. 该模型和算法是任意优先级DAG响应时间分析的经典参考. 本文选择该模型及算法做对比实验.

文献[34-36]研究的是带有优先级的特殊类型DAG任务的响应时间分析问题. 2022年，Bi[34]提出了一种在伪多项式时间内计算具有互斥节点的优先级DAG任务模型WCRT界限的动态规划算法，并证明了相应的WCRT界限计算问题是强NP困难的. 2023年，Chen[35]针对多DAG任务，提出了一种任务内任意优先级分配的响应时间分析方法，该方法通过全面探索每个DAG及其节点间的并行性来精确处理内部和任务间干扰. 2023年，He[36]首次研究了条件DAG任务的任务内优先级分配问题，并提出了一种算法来计算任意优先级条件DAG任务的响应时间界限.

以上所有研究均是关于优先级DAG任务最坏情况响应时间界限的工作，与实际最坏情况响应时间存在较大差距. 本文针对带有优先级的一般单

DAG任务的响应时间分析问题，利用DAG任务节点任意优先级分配方法，结合SMT技术，计算DAG任务的精确WCRT.

3 系统模型

本文主要研究的是多核平台上基于优先级列表调度算法的单DAG非复发任务的响应时间分析问题. 本节主要介绍系统模型，包括任务模型和调度模型.

3.1 任务模型

本文主要研究带有优先级的DAG任务模型$G=(N, E, P)$，其中，N代表任务节点集；E代表节点之间的边集；P代表节点优先级集合，用于存储分配给各节点的优先级. 每个节点n_a对应于一段连续的执行代码，其执行时间定义为e_a，最坏情况执行时间(Worst-Case Execution Time，WCET)定义为w_a，且$e_a \leqslant w_a$，表示节点的执行时间不能超过WCET. 每个节点n_a的开始执行时间定义为s_a，完成时间定义为f_a. 每个节点n_a都有固定的优先级，用整数p_a来表示，p_a越大，节点优先级越低. 每条边(n_a, n_b)表示的是节点n_a和n_b之间的依赖关系，即节点n_a完成执行后，n_b才能开始执行. 此时，称n_a为n_b的“前驱”，n_b是n_a的“后继”，定义Pred(n_a)表示n_a的前驱集合，Succ(n_a)表示n_a的后继集合. 若n_a是n_b的(前驱的)前驱，则称n_a为n_b的“祖先”，n_b是n_a的“后代”，定义Ance(n_a)表示n_a的祖先集合，Desc(n_a)表示n_a的后代集合. 此外，定义Para(n_a)表示与n_a可以并行执行的节点集，Para(n_a)定义如下，

$$\text{Para}(n_a)=\{n_l | n_l \notin \text{Ance}(n_a) \wedge n_a \notin \text{Desc}(n_a) \wedge n_l \neq n_a\}.$$

特别地，没有前驱的节点称为G的源点，用n_{sc}表示；没有后继的节点称为G的汇点，用n_{sk}表示. 本文假设DAG G只有一个源点和汇点. 本文的方法也适用于有多个源点和汇点的DAG：只需要在多个源点前添加一个执行时间为0的虚拟源点，在多个汇点后添加一个执行时间为0的虚拟汇点.

图2(a)给出的是一个简单的DAG任务G的示例. 圆圈内的数字表示每个节点的WCET. 括号内的数字表示每个节点分配的优先级. G的源点为n_0，其优先级为$p_0=0$. G的汇点为n_8，其优先级$p_8=4$. 节点n_3有两个前驱n_1和n_2，一个后继n_6. n_3的祖先集合Ance$(n_3)=\{n_0, n_1, n_2\}$，后代集合Desc$(n_3)=\{n_6, n_8\}$，并行节点集合Para$(n_3)=\{n_4, n_5, n_7\}$.

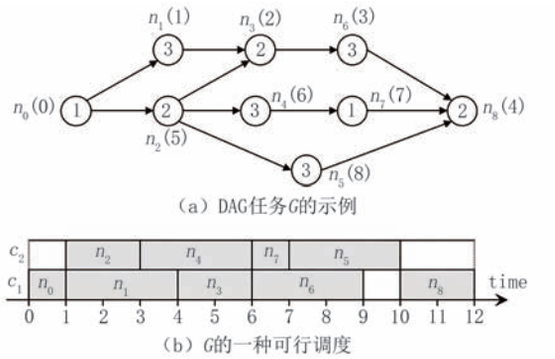

图2 DAG任务G的示例和G的可能调度

3.2 调度模型

本文使用文献[20]中的方法为DAG中的每个节点分配优先级. 节点的调度基于多核平台C进行，C由m个相同的内核构成，即$C=\{c_1, c_2, \cdots, c_m\}$. 可行调度需要满足如下约束：

(1) 在任意时刻t，同一个节点只能在一个内核上执行. 一个内核不能同时执行两个节点.

(2) 如果一个节点n_a的所有前驱都已经完成执行，则称n_a处于“可执行状态”，表示一旦存在空闲内核，n_a就可以执行.

(3) DAG图G中节点的执行顺序由给节点的优先级决定. 在调度时，节点的执行是不可抢占的，即在节点n_a的执行过程中，节点n_b被释放且处于可执行状态，其优先级也高于n_a，但n_b不能打断n_a的执行. 如果两个具有相同优先级的节点在同一时刻变为可执行状态，则随机选择其中一个执行.

4 调度算法

本节将详细介绍优先级列表调度算法(PLS)的调度过程. PLS算法在调度过程中考虑了DAG图G中节点的优先级，其调度过程如下.

在PLS算法下，对于给定DAG图G节点的列表L，所有m个内核在0时刻同时遍历列表L，搜索优先级最高且处于可执行状态的节点. 搜索到列表L中第一个满足条件的节点n_a会立即由某个内核(如c_i)执行，执行时间为e_a个时间单位. 通常，在任意时间点，内核c_i一旦完成一个节点的执行，就会立即遍历列表L，搜索下一个优先级最高且处于可执行状态的节点. 如果当前不存在这样的节点，c_i会保持空闲状态. 当某一个内核c_j完成一个节点n_b

的执行时，内核c_i(以及内核c_j)会立即重新遍历列表L，搜索符合条件的节点来执行(因为节点n_b的完成，所以此时可能会存在处于可执行状态的节点).如果两个或多个内核同时尝试执行同一个节点，规定将该节点分配给序号小的内核上执行.

图2(b)给出的是图2(a)中DAG任务在内核数量为2的多核平台上的一种可能的调度.假设所有节点都在最坏情况下执行.该调度从$t=0$时开始执行，到$t=12$时结束执行.在$t=3$时，节点n_5的前驱节点n_2完成执行，所以n_5变为“可执行状态”，但n_5并未立即执行，而是延迟到了$t=7$时执行，造成这种现象的原因是：在$t=3$时，n_4、n_5均处于“可执行状态”，且n_4是n_5的并行节点，即$n_4\in\mathrm{Para}(n_5)$，但n_5的优先级较低，所以优先级较高的n_4优先执行，同理，节点n_3、n_6、n_7的优先级均高于n_5，在$t=7$时，内核c_1完成了n_7的执行，处于空闲状态，此时才开始执行n_5.

定义1. 关键前驱.如果一个节点n_a满足以下条件，则称节点n_a为n_b的关键前驱，

$$n_a=\arg\max\left\{f_k|n_k\in\mathrm{Pred}(n_b)\right\} \tag{1}$$

其中，f_k表示节点n_k的完成执行时间.定义$\mathrm{CP}(n_b)$表示n_a的关键前驱，即$n_a=\mathrm{CP}(n_b)$.

直观上，节点n_b的关键前驱指的是，所有前驱中完成时间最晚的节点.例如，在图2(b)中，节点n_3的关键前驱是n_1，即$n_1=\mathrm{CP}(n_3)$；节点n_8的关键前驱为n_5，即$n_5=\mathrm{CP}(n_8)$.

引理1. 在基于不可抢占调度模型的PLS算法中，对于任意节点n_b以及$n_a=\mathrm{CP}(n_b)$，如果$s_b>f_a$，那么在时间区间$[f_a,\ s_b]$内，所有内核都是忙碌的.

证明. 采用反证法证明.不妨设存在时刻$t\in[f_a,\ s_b]$以及内核c_i，在时刻t内核c_i是空闲的.由PLS算法可知，在时刻t不存在可执行节点；否则内核c_i在t时刻执行该节点.

由于$n_a=\mathrm{CP}(n_b)$，即n_a为n_b的关键前驱，且n_a在f_a时刻完成，通过关键前驱和可执行状态定义可知，在时间区间$[f_a,\ s_b]$内节点n_b处于“可执行状态”，即在t时刻n_b处于“可执行状态”，这与t时刻不存在可执行节点矛盾.　　证毕.

引理2. 对任意节点n_b以及$n_a=\mathrm{CP}(n_b)$，如果$s_b>f_a$，那么在时间区间$[f_a,\ s_b]$内，多核平台C的工作负载为$(s_b-f_a)m$.

证明. 由引理1可知，对任意节点n_b以及$n_a=\mathrm{CP}(n_b)$，如果$s_b>f_a$，则所有内核都是忙碌的.由于多核平台C由m个相同的内核构成，因此在时间区间$[f_a,\ s_b]$内，多核平台C的工作负载为$(s_b-f_a)m$.

证毕.

在多核平台上以PLS算法调度DAG任务G，其响应时间(Response Time)$R(G)$定义如下，

$$R(G)=f_{sk}-s_{sc} \tag{2}$$

其中，s_{sc}表示G的源点n_{sc}的开始执行时间，f_{sk}表示G的汇点n_{sk}的完成执行时间.本文研究DAG任务G的最差情况响应时间(Worst Case Response Time，WCRT)$R_w(G)$，定义如下，

$$R_{\mathrm{w}}(G)=\max\left(f_{sk}-s_{sc}\right) \tag{3}$$

5　可满足性模理论模型

本节将介绍能够精确求解带有优先级DAG任务WCRT的方法.与文献[26]相似，本文采用可满足性模理论(SMT)技术计算多核平台上DAG任务的WCRT. SMT技术是一种用于解决在特定理论下判定一阶逻辑公式可满足性问题的技术.在DAG任务响应时间分析中，SMT通过将任务的执行逻辑、资源约束以及优先级关系等转化为一系列逻辑公式，并利用SMT求解器来判断这些公式是否可满足，从而计算DAG任务的WCRT. SMT能够处理任务的依赖关系和优先级约束等复杂的逻辑约束，使得SMT在DAG任务响应时间分析中具有较高的灵活性和准确性.

文献[26]中的SMT方法主要关注节点之间的依赖关系，没有考虑节点优先级对响应时间的影响.优先级的分配可以缩短任务的响应时间，有利于系统的可调度性.本文在SMT模型中加入与节点优先级相关的约束，确保DAG任务WCRT的计算能够准确反映高优先级节点对低优先级节点执行时间的影响.具体做法是在SMT模型中添加与优先级相关的约束，这些约束反映了不同优先级节点在调度和执行时的优先顺序.通过这些约束，能够在计算WCRT时更准确地模拟实际系统中，特别是在高负载和复杂任务依赖关系下的任务调度行为.接下来将首先介绍SMT模型涉及的常量和变量，然后介绍SMT模型的约束.

(1) 模型参数

SMT模型的常量设置如下.

① N：DAG任务G的节点集合.

② E：DAG任务G的边集合.

③ w_a：节点n_a的WCET.

④ p_a：节点n_a的优先级.

⑤ m：多核平台C的内核数量.

⑥ Pred(n_a)：节点n_a的前驱集合.

⑦ Succ(n_a)：节点n_a的后继集合.

⑧ Para(n_a)：节点n_a的并行节点集合.

SMT模型中变量设置如下.

① r_a：实数型变量，节点n_a的释放时间.

② s_a：实数型变量，节点n_a的开始执行时间.

③ f_a：实数型变量，节点n_a的完成执行时间.

④ e_a：实数型变量，节点n_a的执行时间.

⑤ $Z_{ab}(k)$：布尔型变量，表示节点n_k的执行时间是否与时间区间$[f_a, s_b]$相交.

⑥ $F_{ab}(k)$：布尔型变量，表示在时刻f_a之前开始执行的节点n_k，其执行时间区间$[s_k, f_k]$是否与时间区间$[f_a, s_b]$相交.

⑦ $B_{ab}(k)$：布尔型变量，表示在时刻s_b之后完成执行的节点n_k，其执行时间区间$[s_k, f_k]$是否与时间区间$[f_a, s_b]$相交.

变量$Z_{ab}(k)$、$F_{ab}(k)$和$B_{ab}(k)$的详细介绍见后文，在此不再赘述.

(2) 模型构建

本文构建的SMT模型采用了文献[26]中的部分约束，用于描述DAG任务G模型的前驱后继关系. 下面首先给出SMT模型的目标函数，然后详细介绍SMT模型中的约束.

目标函数. SMT模型致力于DAG任务G的最差情况响应时间，其目标函数如下.

$$\max(f_{sk}-s_{sc}) \tag{4}$$

其中，s_{sc}表示G的源点n_{sc}的开始执行时间，f_{sk}表示G的汇点n_{sk}的完成执行时间.

约束. SMT模型包含了以下几类约束：WCET约束、依赖约束、不可抢占约束、源点约束、节点释放时间约束、并行节点优先级约束以及负载保持约束. 其中，负载保持约束的实现是构建SMT模型的关键，“负载保持”属性对DAG任务WCRT的求解至关重要. 本节的重点在于加入优先级后如何实现负载保持约束.

① **WCET约束**. DAG任务G的节点执行时间不能大于其最坏情况执行时间WCET，约束如下：

$$0<e_a\leqslant w_a,\quad \forall n_a\in N \tag{5}$$

通常情况下，DAG任务中的节点代表着有意义的任务，即节点的执行时间大于0. 本文假设，除了源点和汇点之外的所有节点的执行时间均大于0.

②**依赖约束**. G中的边(n_a, n_b)表示的是一种依赖关系，即节点n_b的执行依赖n_a，在n_a完成执行后，n_b才能开始执行，约束如下：

$$f_a\leqslant s_b,\quad \forall(n_a,n_b)\in E \tag{6}$$

③**不可抢占约束**. 本文考虑的是带有优先级的不可抢占调度，对于G中的每个节点n_a，一旦n_a开始执行，就不能被打断，约束如下：

$$f_a=s_a+e_a,\quad \forall n_a\in N \tag{7}$$

④**源点约束**. 对G的源点n_{sc}来说，规定其在0时刻开始执行，约束如下：

$$s_{sc}=0,\quad n_{sc}\in N \tag{8}$$

⑤**节点释放时间约束**. 在本文模型中，每个节点n_b的释放时间r_b要保证不早于其关键前驱n_a的完成时间f_a，对任意节点n_b及其关键前驱n_a，有

$$r_b\geqslant f_a \tag{9}$$

根据定义1，公式(9)进一步可改写成如下形式：

$$r_b\geqslant\max\{f_k|n_k\in\mathrm{Pred}(n_b)\} \tag{10}$$

⑥**并行节点优先级约束**. 对任意节点n_b及其并行节点n_k，若n_k的优先级p_k高于n_b的优先级p_b(即$p_k<p_b$)，且n_k的释放时间不晚于n_b，则n_k的开始执行时间不能晚于n_b，其表达式表示如下：

$$r_k\leqslant r_b\wedge p_k<p_b\rightarrow s_k\leqslant s_b \tag{11}$$

为实现公式(11)，其进一步可改写成如下约束：

$$r_k>r_b\vee p_k\geqslant p_b\vee s_k\leqslant s_b \tag{12}$$

⑦**负载保持约束**. 由引理1可知，对任意节点n_b以及$n_a=\mathrm{CP}(n_b)$，如果在时间区间$[f_a, s_b]$内，列表L存在可执行节点，则所有内核都是忙碌的. 定义以下逻辑表达式来表示负载保持约束.

$$\Phi_1:\left(\bigwedge_{n_k\in\mathrm{Pred}(n_b)}f_a\geqslant f_k\right)\wedge s_b>f_a \tag{13}$$

引理3. $\Phi_1=1$表示节点n_a是节点n_b的关键前驱，即$n_a=\mathrm{CP}(n_b)$，并且在时间区间$[f_a, s_b]$内所有内核都是忙碌的.

证明. $\bigwedge_{n_k\in\mathrm{Pred}(n_b)}f_a\geqslant f_k$为真表示节点$n_a$是节点$n_b$所有前驱中最晚完成执行的节点之一，节点$n_a$具有最大的完成时间. 由定义1可知，节点$n_a$是$n_b$的关键前驱.

当$n_a=\mathrm{CP}(n_b)$且$s_b>f_a$时，由引理1可知在时

间区间$[f_a, s_b]$内所有内核都是忙碌的．　　证毕．

对于在$[f_a, s_b]$内执行的节点，给出以下定义．

定义2. 关系节点．　对任意节点n_b及其关键前驱n_a，且$s_b > f_a$. 若节点n_k满足以下条件，则称n_k为时间区间$[f_a, s_b]$的关系节点，

$$s_k < s_b \wedge f_k > f_a \tag{14}$$

节点n_k为时间区间$[f_a, s_b]$的关系节点意味着n_k的执行区间$[s_k, f_k]$与区间$[f_a, s_b]$相交不为空集．如图3所示，n_k的执行区间$[s_k, f_k]$包含于区间$[f_a, s_b]$分为2种情况：(1) $[s_k, f_k]$部分包含于$[f_a, s_b]$，如图3(a)、图3(b)和图3(c)所示；(2) $[s_k, f_k]$完全包含于$[f_a, s_b]$，如图3(d)所示．

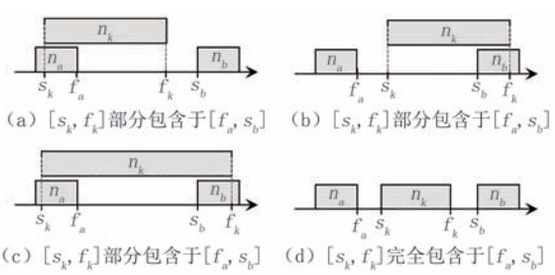

图3　区间$[s_k, f_k]$包含于区间$[f_a, s_b]$的情况示意

定义U_{ab}为$[f_a, s_b]$包含的所有关系节点的集合．例如在图2(b)所示的调度中，节点n_2是n_5的关键前驱，时间区间$[f_2, s_5]$的关系节点集合$U_{25}=\{n_1, n_3, n_4, n_6, n_7\}$.

引理4.　任意节点n_b及其关键前驱n_a，且$s_b > f_a$，下式一定成立，

$$\begin{aligned} &s_k < s_b \wedge f_k > f_a \leftrightarrow \\ &(s_k < f_a \wedge f_k > f_a) \vee (s_k \geqslant f_a \wedge s_k < s_b) \end{aligned} \tag{15}$$

证明．(1) 当公式(15)左部，即$s_k < s_b \wedge f_k > f_a$成立时，节点$n_k$是区间$[f_a, s_b]$的关系节点．节点$n_k$的开始执行时间$s_k$可分为以下两种情况，$s_k < f_a$，或$s_k \geqslant f_a \wedge s_k < s_b$，即当$s_k < s_b \wedge f_k > f_a$为真时，$(s_k < f_a) \vee (s_k \geqslant f_a \wedge s_k < s_b)$也为真，进而可得$(s_k < f_a \wedge f_k > f_a) \vee (s_k \geqslant f_a \wedge s_k < s_b \wedge f_k > f_a)$为真．由WCET约束(5)及不可抢占约束(7)可知，$f_k > s_k$，故当$s_k \geqslant f_a$时，$f_k > s_a$恒成立，所以$(s_k < f_a \wedge f_k > f_a) \vee (s_k \geqslant f_a \wedge s_k < s_b)$为真．

(2) 当公式(15)右部，即$(s_k < f_a \wedge f_k > f_a) \vee (s_k \geqslant f_a \wedge s_k < s_b)$成立时，分以下两种情况讨论．

① 当$s_k < f_a \wedge f_k > f_a$为真时，可知$s_k < f_a$且$f_k > f_a$，又$s_b > f_a$，所以$s_k < s_b$，故$s_k < s_b \wedge f_k > f_a$为真．

② 当$s_k \geqslant f_a \wedge s_k < s_b$为真时，可知$s_k \geqslant f_a$且$s_k < s_b$. 由WCET约束(5)及不可抢占约束(7)可知，$f_k > s_k$，所以$f_k > f_a$，故$s_k < s_b \wedge f_k > f_a$为真．

综合以上两种情况可知，当$(s_k < f_a \wedge f_k > f_a) \vee (s_k \geqslant f_a \wedge s_k < s_b)$为真时，$s_k < s_b \wedge f_k > f_a$也为真．

证毕．

根据关系节点n_k的开始执行时间，本文考虑两类关系节点，并给出不同的逻辑表示式，

$$\begin{aligned} \Phi_2:&(s_k < f_a \wedge f_k > f_a) \vee \\ &(s_k \geqslant f_a \wedge s_k < s_b \wedge p_k < p_b) \end{aligned} \tag{16}$$

引理5.　对任意节点n_b以及$n_a = \mathrm{CP}(n_b)$，如果节点n_k满足$\Phi_2 = 1$，则n_k为时间区间$[f_a, s_b]$的关系节点，即$n_k \in U_{ab}$.

证明．(1) 如果公式(16)左部满足，说明节点n_k的开始执行时间s_k早于节点n_a的完成时间f_a. 由于正在执行任务的内核是不可抢占的，因此n_k执行的一部分包含于区间$[f_a, s_b]$.

(2) 如果公式(16)右部满足，说明下述3种情况同时为真：①节点n_k的开始执行时间s_k恰好落在区间$[f_a, s_b]$内；②节点n_k的释放时间r_k不晚于节点n_b的释放时间r_b；③节点n_k的优先级p_k高于节点n_b的优先级p_b(即$p_k < p_b$). 因此，由PLS算法可知节点n_k执行的一部分必定包含于区间$[f_a, s_b]$.　证毕．

引理6.　对任意节点n_b以及$n_a = \mathrm{CP}(n_b)$，如果节点$n_k \in U_{ab}$，则$n_k \in \mathrm{Para}(n_b)$.

证明．通过反证法证明该引理．不妨设对于给定的节点n_b及其关键前驱n_a，即$n_a = \mathrm{CP}(n_b)$，节点$n_k \in U_{ab}$，并且$n_k \notin \mathrm{Para}(n_b)$. 由并行节点定义可知存在下述三种情况．

(1)节点n_k为节点n_b的祖先节点，即$n_k \in \mathrm{Ance}(n_b)$. 由于节点$n_a$是节点$n_b$的关键前驱，由公式(1)可知下式成立，

$$f_k \leqslant f_a \tag{17}$$

由WCET约束可知$s_k < f_k$，可以推出，

$$s_k < f_a \tag{18}$$

根据公式(17)和公式(18)，$(s_k < f_a \wedge f_k > f_a) = 0$，并且$(f_a \leqslant s_k < s_b) = 0$，即公式(15)右部为0. 由关系节点定义可知，$n_k$不是时间区间$[f_a, s_b]$的关系节点．这与节点$n_k \in U_{ab}$矛盾．

(2)节点n_k为节点n_b的后代节点，即$n_k \in \mathrm{Desc}(n_b)$. 由依赖关系可知$s_k \geqslant f_b$. 由执行时间$e_b > 0$可知$f_b > s_b$. 因此，可推出下式成立，

$$s_k > s_b \tag{19}$$

由节点n_a是节点n_b的关键前驱可知$s_b \geqslant f_a$，容易推出下式成立，

$$s_k > f_a \tag{20}$$

根据公式(19)和公式(20)，$(s_k < f_a \wedge f_k > f_a) = 0$，并且$(f_a \leqslant s_k < s_b) = 0$. 与情况1)类似，公式(15)右部为0. 由关系节点定义可知，n_k不是时间区间$[f_a, s_b]$的关系节点. 这与节点$n_k \in U_{ab}$矛盾.

(3) 节点n_k等于节点n_b，即$n_k = n_b$. 显然，在此情况下，$n_k \notin U_{ab}$，与节点$n_k \in U_{ab}$矛盾. 证毕.

对任意边(n_a, n_b)及任意节点$n_k \in \mathrm{Para}(n_b)$，布尔变量$Z_{ab}(k)$的定义如下，

$$Z_{ab}(k)=\begin{cases}1: & n_a=\mathrm{CP}(n_b), \text{且} n_k \in U_{ab}\\ 0: & \text{其他情况}\end{cases} \tag{21}$$

根据公式(21)和引理6，关系节点集合U_{ab}可通过以下表达式来表示，

$$U_{ab}=\{n_k | Z_{ab}(k)=1, \forall n_k \in \mathrm{Para}(n_b)\} \tag{22}$$

引理7. 任意节点n_b以及$n_a = \mathrm{CP}(n_b)$，对于n_b的任意并行节点n_k ($n_k \in \mathrm{Para}(n_b)$)，$n_k \in U_{ab}$，当且仅当$\Phi_1$和$\Phi_1$同时为真，即

$$\Phi_1 \wedge \Phi_2 \leftrightarrow Z_{ab}(k)=1 \tag{23}$$

证明. 必要性. 由引理3可知，$\Phi_1 = 1$表示节点n_a是节点n_b的关键前驱，并且$s_b > f_a$。由公式(16)可知$\Phi_2 = 1$存在两种情况：

(1) 对任意$n_k \in \mathrm{Para}(n_b)$，当$s_k < f_a \wedge f_k > f_a$为真时，由公式(14)可知，节点$n_k$是时间区间$[f_a, s_b]$的关系节点.

(2) 对任意$n_k \in \mathrm{Para}(n_b)$，$s_k \geqslant f_a \wedge s_k < s_b \wedge p_k < p_b$为真时，易知$s_k \geqslant f_a \wedge s_k < s_b$为真. 由公式(14)可知，节点$n_k$是时间区间$[f_a, s_b]$的关系节点.

当Φ_1和Φ_1同时为真时，可知对任意并行节点$n_k \in \mathrm{Para}(n_b)$有$n_k \in U_{ab}$.

充分性. 当$Z_{ab}(k) = 1$时，由公式(21)可知$n_a = \mathrm{CP}(n_b)$且$n_k \in U_{ab}$，并且节点n_k是区间$[f_a, s_b]$的关系节点。由关系节点的定义可知，$s_b > f_a$. 由公式(13)可知，$\Phi_1 = 1$.

由于节点n_k是区间$[f_a, s_b]$的关系节点，由引理4可知，$(s_k < f_a \wedge f_k > f_a) \vee (s_{k \geqslant} f_a \wedge s_k < s_b) = 1$，存在下述两种情况.

① 当$s_k < f_a \wedge f_k > f_a$为真时，由公式(16)可知$\Phi_2 = 1$.

② 当$s_k \geqslant f_a \wedge s_k < s_b$为真时，可知在时间区间$[f_a, s_b]$中，节点$n_k$和节点$n_b$都是可执行节点. 由PLS算法可知，内核$c_i$一旦完成某个节点的执行，立即遍历列表$L$，搜索下一个优先级最高且处于可执行状态的节点. 因为$s_k < s_b$，因此可知节点n_k的优先级高于节点n_b的优先级，即$p_k < p_b$. 由公式(16)可知$\Phi_2 = 1$.

因此，对于n_b的任意并行节点n_k，如果$n_k \in U_{ab}$为真，则可知Φ_1和Φ_1同时为真. 证毕.

由引理3可知，$\Phi_1 = 1$表示节点n_a是节点n_b的关键前驱，$s_b > f_a$并且在时间区间$[f_a, s_b]$内所有内核都是忙碌的. 由引理5可知，$\Phi_2 = 1$表示节点n_k为时间区间$[f_a, s_b]$的关系节点. 公式(23)可以进一步改写成以下约束. 对任意边(n_a, n_b)及任意节点$n_k \in \mathrm{Para}(n_b)$，有，

$$\neg \Phi_1 \vee \neg \Phi_2 \vee (Z_{ab}(k)=1) \tag{24}$$

$$(Z_{ab}(k)=0) \vee \Phi_1 \tag{25}$$

$$(Z_{ab}(k)=0) \vee \Phi_2 \tag{26}$$

约束(24)到(26)皆为真才能保证布尔型变量$Z_{ab}(k) = 1$的条件成立.

为了分析节点n_k的执行区间$[s_k, f_k]$与时间区间$[f_a, s_b]$的交集，对于任意边(n_a, n_b)的关系节点集合U_{ab}，进一步定义关于U_{ab}的两种类型的子集，

(1)左子集S_L：满足如下条件的U_{ab}中的节点属于U_{ab}的左子集S_L，

$$\Phi_3: \quad s_k < f_a \tag{27}$$

(2)右子集S_R：满足如下条件的U_{ab}中的节点属于U_{ab}的右子集S_R，

$$\Phi_4: \quad f_k > s_b \tag{28}$$

例如，在图2中，节点n_5的关键前驱为节点n_2，时间区间$[f_2, s_5]$的关系节点集合$U_{25} = \{n_1, n_3, n_4, n_6, n_7\}$，其中，$n_1$属于$U_{25}$的左子集$S_L$(即$S_L = \{n_1\}$)，$n_6$属于$U_{25}$的右子集$S_R$(即$S_R = \{n_6\}$).

对于任意边(n_a, n_b)及任意节点n_k，定义布尔变量$F_{ab}(k)$来表示n_k是否包含在U_{ab}的左子集S_L中. 同样，定义$B_{ab}(k)$来表示n_k是否包含在U_{ab}的右子集S_R中，定义如下，

$$F_{ab}(k)=\begin{cases}1, & n_k \in S_L\\ 0, & \text{其他情况}\end{cases} \tag{29}$$

$$B_{ab}(k)=\begin{cases}1, & n_k \in S_R\\ 0, & \text{其他情况}\end{cases} \tag{30}$$

为了实现公式(29)和(30)，对任意边(n_a, n_b)及任意节点$n_k \in \mathrm{Para}(n_b)$，给出下面的逻辑表达式，

$$(Z_{ab}(k)=1) \wedge \Phi_3 \rightarrow (F_{ab}(k)=1) \tag{31}$$

$$(Z_{ab}(k)=1) \wedge \Phi_4 \rightarrow (B_{ab}(k)=1) \tag{32}$$

公式(31)保证了对于区间$[f_a, s_b]$的任意关系节点n_k(即当$Z_{ab}(k)=1$时)，如果n_k属于U_{ab}的左子集S_L(即Φ_3为真)，则$F_{ab}(k) = 1$. 相似地，公式(32)保

证了，若n_k属于U_{ab}的右子集S_R，那么$B_{ab}(k)=1$. 根据公式(31)和(32)，给出如下两个表达式作为SMT模型中的约束，

$$(Z_{ab}(k)=0)\vee\neg\Phi_3\vee(F_{ab}(k)=1) \tag{33}$$

$$(Z_{ab}(k)=0)\vee\neg\Phi_4\vee(B_{ab}(k)=1) \tag{34}$$

对任意实数$x\in\mathrm{R}$，函数$\sigma(x)$的定义如下，

$$\sigma(x)=\begin{cases}x, & x>0\\ 0, \text{其他情况}\end{cases}$$

例如，当区间$[f_a, s_b]$的长度大于0时，$\sigma(s_b-f_a)=s_b-f_a$. 当区间$[f_a, s_b]$的长度为0时，$\sigma(s_b-f_a)=0$.

引理8. 对任意满足负载保持的调度算法，任意节点n_b及其关键前驱n_a，时间区间$[f_a, s_b]$内的执行负载满足下述公式，

$$m(s_b-f_a)=\sum_{n_k\in\mathrm{Para}(n_b)}(e_kZ_{ab}(k)-\sigma(f_a-s_k)F_{ab}(k)-\sigma(f_k-s_b)B_{ab}(k)) \tag{35}$$

证明. 由于节点n_a是节点n_b的关键前驱，由依赖约束公式(6)可知，$s_b\geqslant f_a$，存在下述两种情况.

第一种情况：$s_b=f_a$. 对于任意$n_k\in\mathrm{Para}(n_b)$，由公式(13)可知Φ_1为假(即$\Phi_1=0$). 由公式(21)可知$Z_{ab}(k)=0$. 由公式(22)可知U_{ab}为空集合，根据公式(29)和(30)可知，$F_{ab}(k)=0$且$B_{ab}(k)=0$. 因此，容易推出公式(35)成立.

第二种情况：$s_b>f_a$. 对于任意$n_k\in\mathrm{Para}(n_b)$，假如n_k不是时间区间$[f_a, s_b]$的关系节点(即不满足公式(14))时，可知$s_k\geqslant s_b$为真，或者$f_k\leqslant f_a$为真. 接下来分两种情况讨论.

(1)当$s_k\geqslant s_b$时，$s_k\geqslant s_b>f_a$成立，由公式(16)可知$\Phi_2=0$. 根据公式(21)可推出$Z_{ab}(k)=0$.

(2)当$f_k\leqslant f_a$时，由WCET约束(5)及不可抢占约束(7)可知$s_k<f_k\leqslant f_a$，由公式(16)可知$\Phi_2=0$. 根据公式(21)可推出$Z_{ab}(k)=0$. 由公式(22)可知$n_k\notin U_{ab}$，根据公式(29)和(30)可知，$F_{ab}(k)=0$且$B_{ab}(k)=0$. 因此，n_k不会对公式(35)右部产生影响.

接下来，仅针对属于U_{ab}的节点n_k展开证明，即节点n_a是节点n_b的关键前驱，$s_b>f_a$，且$Z_{ab}(k)=1$成立，分三种情况讨论.

(1)当n_k包含在U_{ab}的左子集S_L中，即满足$s_k<f_a$时，可知$F_{ab}(k)=1$，分下述两种情况.

①当$f_k\leqslant s_b$时，由公式(28)和(30)可知$B_{ab}(k)=0$，且节点n_k的执行区间$[s_k, f_k]$与区间$[f_a, s_b]$的交集为$[f_a, f_k]$. 将$Z_{ab}(k)=1$、$F_{ab}(k)=1$以及$B_{ab}(k)=0$代入公式(35)右部可推出，

$$(f_k-s_k)-(f_a-s_k)=f_k-f_a$$

②当$f_k>s_b$时，由公式(28)和(30)可知$B_{ab}(k)=1$，且节点n_k的执行区间$[s_k, f_k]$与区间$[f_a, s_b]$的交集为$[f_a, s_b]$. 将$Z_{ab}(k)=1$、$F_{ab}(k)=1$以及$B_{ab}(k)=1$代入公式(35)右部可推出，

$$(f_k-s_k)-(f_a-s_k)-(f_k-s_b)=s_b-f_a$$

(2)当n_k包含在U_{ab}的右子集S_R中，即满足$f_k>s_b$时，可知$B_{ab}(k)=1$，分下述两种情况.

①当$s_k<f_a$时，可知$F_{ab}(k)=1$，与1)中第②种情况一致.

②当$s_k\geqslant f_a$时，可知$F_{ab}(k)=0$，且节点n_k的执行区间$[s_k, f_k]$与区间$[f_a, s_b]$的交集为$[s_k, s_b]$. 将$Z_{ab}(k)=1$、$F_{ab}(k)=0$以及$B_{ab}(k)=1$代入公式(35)右部可推出，

$$(f_k-s_k)-(f_k-s_b)=s_b-s_k$$

(3)当n_k即不属于U_{ab}的左子集S_L也不属于其右子集S_R时，即$s_k\geqslant f_a\wedge f_k\leqslant s_b$时，可得$F_{ab}(k)=0$以及$B_{ab}(k)=0$，且节点$n_k$的执行区间$[s_k, f_k]$与区间$[f_a, s_b]$的交集为$[s_k, f_k]$. 将$Z_{ab}(k)=1$、$F_{ab}(k)=0$以及$B_{ab}(k)=0$代入公式(35)右边可推出，

$$e_kZ_{ab}(k)=f_k-s_k$$

由引理1可知，在时间区间$[f_a, s_b]$内，所有内核都是忙碌的.

综合上述3种情况，可知公式(35)成立. 证毕.

引理9. 基于m核平台上的调度S具有“负载保持”属性当且仅当满足下述约束条件，

$$\Phi_1\rightarrow m(s_b-f_a)=\sum_{n_k\in\mathrm{Para}(n_b)}(e_kZ_{ab}(k)-\sigma(f_a-s_k)F_{ab}(k)-\sigma(f_k-s_b)B_{ab}(k)) \tag{36}$$

证明. 充分性：若公式(36)成立，可知节点n_a是节点n_b的关键前驱，且$f_a<s_b$. 公式(35)成立说明在时间区间$[f_a, s_b]$内，所有内核都是忙碌的. 由负载保持约束可知，该调度具有负载保持属性.

必要性：由引理8可知必要性成立. 证毕.

将公式(36)改写为下式作为SMT模型的约束条件.

$$\neg\Phi_1\vee m(s_b-f_a)=\sum_{n_k\in\mathrm{Para}(n_b)}(e_kZ_{ab}(k)-\sigma(f_a-s_k)F_{ab}(k)-\sigma(f_k-s_b)B_{ab}(k)) \tag{37}$$

基于PLS算法在m核上调度DAG任务，求解DAG任务的WCRT可形式化为下述SMT优化问题.

$$\max(f_{sk}-s_{sc}) \tag{38}$$

s.t. 公式(5)～(9)、(12)、(24)～(26)、(33)、(34)、(37).

下面的定理1说明了，基于优先级列表调度算法(PLS)，SMT模型能够得到DAG任务精确的WCRT.

定理1. SMT模型能够精确地求解基于PLS算法DAG任务的WCRT.

证明. 定理1成立等价于：SMT模型给出的解对应了PLS算法的可行调度中最大的响应时间. 证明过程分为两部分：(1) 证明基于PLS算法的任意可行调度均是SMT模型的解，即SMT模型的最优解不小于DAG任务的WCRT；(2) 证明SMT模型的最优解对应PLS算法的一个可行调度.

首先，证明SMT模型的最优解不小于DAG任务的WCRT. 对m核上的任意DAG任务G，将该任务G的WCRT记为W^*，将优化问题(38)的最优目标值记为W^S. 注意到，SMT优化问题(38)中的约束条件未包含PLS可行调度的所有约束条件. 因此，基于PLS算法的任意可行调度均是优化问题(38)的可行解. 易知，有$W^S \geqslant W^*$成立.

然后，通过反证法证明SMT模型的最优解对应PLS算法的一个可行调度. 假设SMT模型的最优解对应了一个不可行调度. 即SMT模型的最优解违背了第3.2节可行调度模型的某一条约束. 由依赖约束(6)可知SMT模型的解满足第3.2节调度模型的约束2). 由不可抢占约束(7)和并行节点优先级约束(11)可知SMT模型的解满足第3.2节调度模型的约束3). 此外，由引理3可知SMT模型的解满足负载保持约束. 所以，SMT模型的解不满足第3.2节调度模型的约束1)，即存在某个时刻t满足，在时刻t有超过m个核同时执行DAG中的节点. 将这些节点按照开始执行时间依次在m核平台上执行，得到一个新的可行调度，其响应时间W^N满足$W^N > W^S$. 由于任务G的WCRT(W^*)满足$W^* \geqslant W^N$，因此可推出$W^* \geqslant W^N > W^S$，又$W^S > W^*$，产生矛盾. 所以，SMT模型的最优解对应了一个可行调度.

综上可得，SMT模型给出的解对应了PLS算法的可行调度中最大的响应时间，即SMT模型能够精确地求解基于PLS算法DAG任务的WCRT.

证毕.

易知，SMT优化问题至多包含$|N|*|E|$个约束. 为了加快SMT模型的求解，本文进一步分析优化问题(39)的约束条件. 在给定的调度S下，定义$W_I(S)$为内核的空闲负载量. 可推出下式成立.

$$W_I(S)+\sum_{i=1}^{|N|}e_i = m(f_{sk}-s_{sc}) \tag{39}$$

由WCET约束知$e_a \leqslant w_a$，因此将约束WCET约束(即公式(5))改写为下式，不会导致响应时间变小.

$$e_a = w_a, \quad \forall n_a \in N \tag{40}$$

类似地，不可抢占约束(即公式(7))改写为下式：

$$f_a = s_a + w_a, \quad \forall n_a \in N \tag{41}$$

综上所述，基于PLS算法求解DAG任务的WCRT可形式化为下述SMT优化问题.

$$\max(f_{sk}-s_{sc})$$

s.t. 公式(6)，(8)，(9)，(12)，(24)～(26)，(33)，(34)，(37)，(40)，(41).

6 实验结果

通过两部分实验深入探讨本文算法和模型在优先级DAG任务WCRT计算中的效率与精度. 首先，与文献[26]进行对比，该文献是当前非优先级DAG任务WCRT精确计算的最新研究成果. 通过该组实验证明本文算法和模型能够在保证计算精度的前提下，提升优先级DAG任务的WCRT计算效率. 其次，与文献[20]进行比较，该文献是研究任意优先级DAG任务WCRT界限的经典参考. 通过该组实验验证本文算法在优先级DAG任务WCRT计算精度上的改进.

实验所用的微型计算机型号为DELL OptiPlex 5090台式机，配置为11th Gen Intel(R) Core(TM) i5-11500 @ 2.70GHz，16.0 GB DDR4内存(15.7 GB可用)，操作系统为Microsoft Windows 11 64位，编程语言使用Python，程序运行环境为Python 3.10.8. 实验使用可配置任务图生成工具TGFF[37]随机生成DAG任务图.

本文实验使用TGFF(Task Graphs for Free，TGFF)工具随机生成任务图，TGFF是一种用户主动控制、使用简便、易理解的伪随机任务生成工具，专门用于生成有向无环图(DAG)任务. TGFF具有很高的灵活性，用户只需要了解简单的任务生成规则就可以通过命令行参数生成对应的任务. 根据TGFF工具生成的任务图，本文详细对比和分析了不同参数组合下算法的实验结果.

6.1　SMT模型计算效率的实验评估

文献[26]的算法及模型记为M_1，本文提出模型记为M_2. 图4～图7分别给出了不同参数配置的实验结果，参数组合如图例所示. 其中m表示内核数量，n表示每个DAG任务图的节点数量，e表示每个节点的执行时间，p表示每个节点的出度(即该节点的后继节点数量).

本文定义响应时间界差Gap来描述实验结果，

$$\mathrm{Gap}=\frac{R_G-R_x}{R_G} \tag{42}$$

R_G为Graham界，其计算公式如下，

$$R_G=len(G)+\frac{vol(G)-len(G)}{m} \tag{43}$$

其中，其中$len(G)$指的是DAG的最长路径长度，$vol(G)$指的是DAG任务G所有节点的执行时间之和，m为内核数量.

M_1的计算结果记为R_1，计算时间记为t_1，相应地，其界差记为$\mathrm{Gap}_1(R_x=R_1)$. M_2的计算结果记为R_2，计算时间记为t_2，其界差记为$\mathrm{Gap}_2(R_x=R_2)$. 在图4～图7的子图(a)中，箱型图展示了M_1和M_2的响应时间界差分布. 左侧点状箱形图为M_1的界差平均值，右侧斜线箱形图为M_2的界差平均值. 具体来说，箱型图按照界差结果从小到大排序，箱型图底部表示第25%的数据，图顶表示第75%的数据. 图底和图顶之间的区域表示25%～75%的数据. 上边缘线表示界差Gap的最大值，中间横线表示中位数，即第50%的数据，下边缘线表示Gap的最小值. 在图4～图7的子图(b)中，左侧点状柱形图表示M_1计算时间t_1的平均值，右侧斜线柱形图则给出的是M_2计算时间t_2的平均值，折线表示M_2相较于M_1的计算效率提升率.

对于每种参数组合，进行1000次随机实验. 实验结果表明，在考虑节点优先级的情况下，M_2依然能够得到较好的响应时间精度，且计算时间优于M_1，在最好情况下本文算法的计算效率提升了近80%. 计算时间减少的原因是SMT模型中引入的优先级相关的约束，减少了求解器的搜索空间. 优先级约束能够排除不满足实际调度约束的解，从而减少求解器需要尝试的候选解的数量. SMT模型在保证计算结果准确性的同时，降低计算复杂度和提高计算效率. 尤其是在处理节点复杂的依赖关系和高负载情况时，本文算法能够更快地收敛到正确的WCRT值，从而进一步减少计算时间.

如图4所示，M_1和M_2的Gap都随着任务节点数量的增长而增大. 具体原因如下，随着任务节点数量的增多，有向无环图的总负载增大. 由于M_1和M_2都能够更加精确地估计DAG的最大响应时间，因此，Gap随着节点数的增加呈递增趋势. 此外，M_1和M_2的计算时间均随着节点数量的增加而增大. 在节点数量较少时，两种方法的平均计算时间差距并不明显，这是因为节点数量越少，模型约束也越少，对计算时间的影响就越小. 随着节点数量的增多，M_2的计算时间明显少于M_1. 平均计算效率提升了近50%. 最好情况下，M_2计算效率提升了近80%.

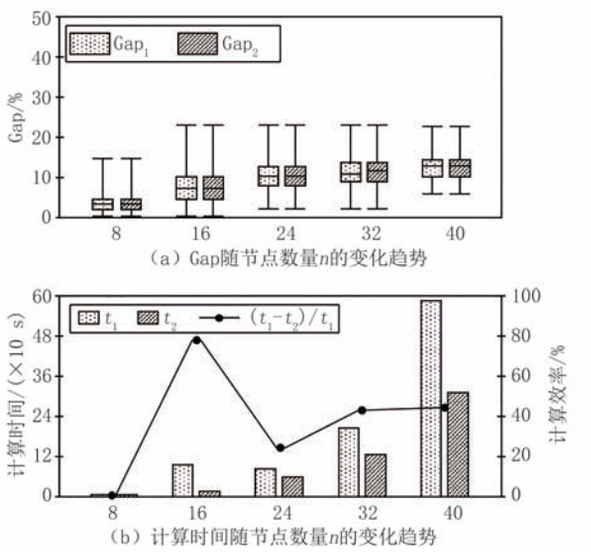

图4　实验参数$e=10$, $m=6$, $p=3$

如图5所示，随着节点执行时间的增大，界差Gap的变化并不显著，表明节点执行时间的大小对界差Gap的影响并不明显，即在节点执行时间变化时，M_1和M_2对Graham界的改进效果相似. 但随着节点执行时间的增加，M_2的计算时间有着明显的改进，M_2的计算效率平均提升了75%.

如图6所示，实验数据表明，硬件平台上的内核数量会影响M_1和M_2所求得的Gap. 在一般情况下，随着内核数量的增加，两种方法求得的Gap越小，根据Graham界的计算公式(43)，内核数量越多，DAG任务图的最大负载对响应时间的影响就越小. 此外，随着内核数量的增加，两者的计算时间均呈递减趋势，但M_2的计算时间始终保持在较低水平，平均计算效率提升了近35%. 即使在内核数量较少时，M_2的计算效率相较于M_1也提升了近20%.

如图7所示，随着节点并行度的增大，Gap以

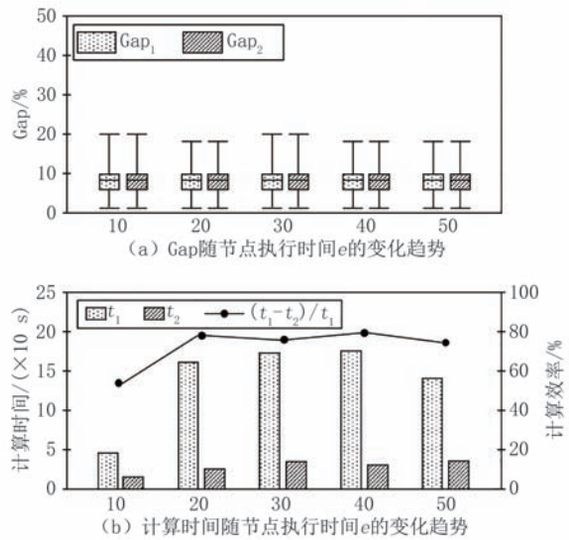

图5 实验参数$n=20$, $m=6$, $p=3$

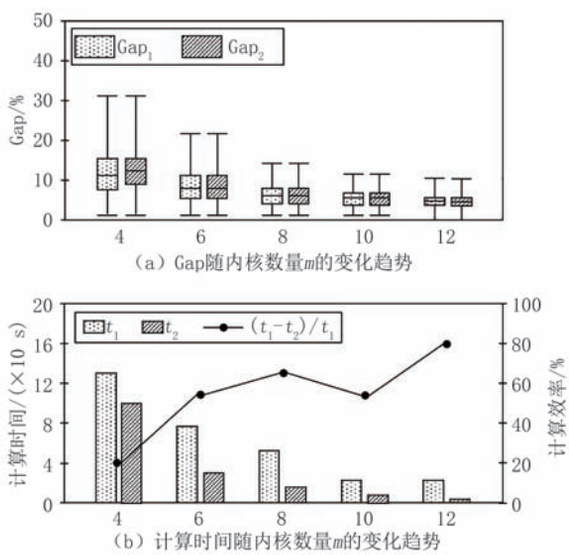

图6 实验参数$n=20$, $e=10$, $p=3$

较为缓慢的趋势增加. 原因如下，当节点的并行度变大时，任务中的可并行执行的节点就越多，所以在内核数量固定时，会发生更多节点等待执行的情况. 所以，M_1和M_2的计算时间也随着并行度的增大而增加，但M_2的计算时间明显优于M_1，平均计算效率提升了近40%.

综合上述实验结果可以得出如下结论. 一方面，与M_1相比，在加入节点优先级后，M_2的计算效率更高(平均情况下提升了近50%)，且能够保证响应时间的精度，这说明M_2具有易实现、鲁棒的优势. 另一方面，在考虑优先级的情况下，M_2依旧能够保证较高的响应时间精度，为优先级DAG任务图响应时间界限的计算提供了一种有效的计算方法.

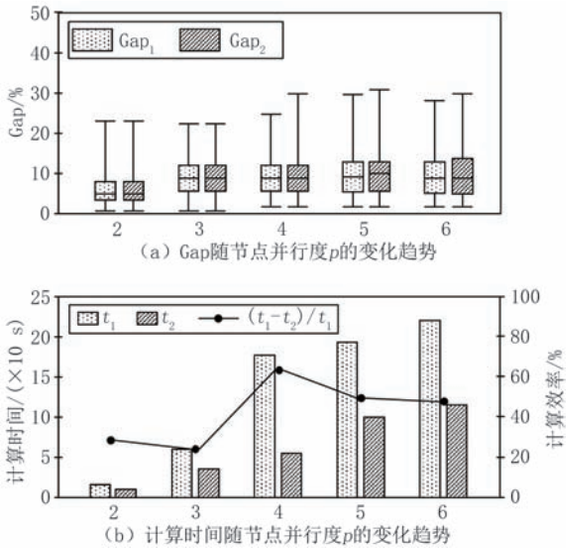

图7 实验参数$n=20$, $e=10$, $m=6$

6.2 SMT模型计算精度的实验评估

文献[20]的算法及模型记为M_3，其WCRT界限的计算结果记为R_3. 图8～图11给出了不同参数配置的实验结果，参数组合如图例所示. 利用响应时间界差Gap展示算法的有效性. M_2与M_3响应时间界差Gap_3的计算方法如下，

$$Gap_3=\frac{R_3-R_2}{R_2}$$

采用箱型图展示本次的实验结果. 对于每种参数组合，进行1000次随机实验. 实验结果表明，在考虑节点优先级的情况下，算法M_2能够改进响应时间界限的精度，在最好情况下，精度提高17.2%.

如图8所示，Gap_3随着任务节点数量的增多而缓慢增大，表明节点数量越多，现有方法的计算结果越不精确，本文算法的改进效果越明显. 原因如下，M_2方法(SMT)具有在细粒度水平上进行优化的能力，能够精确考虑任务节点之间的依赖关系和并发执行情况. 当节点数量增加时，这种细粒度优化的优势变得更加显著，因为任务调度的决策空间扩大，SMT方法可以在更大的解空间中找到更优的调度方案. 在最好情况下，M_2的精度提高了近15%.

如图9所示，随着节点执行时间的增大，Gap_3的只有微小的变化，表明节点执行时间对响应时间界差的影响并不明显，即节点执行时间不是影响响应时间精度的决定性因素. 在最好情况下，M_2的精度提高了近10%.

图10所示的实验表明，硬件平台上的内核数量

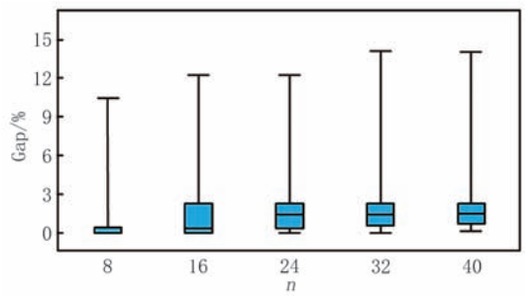

图8　实验参数$e=10$, $m=6$, $p=3$

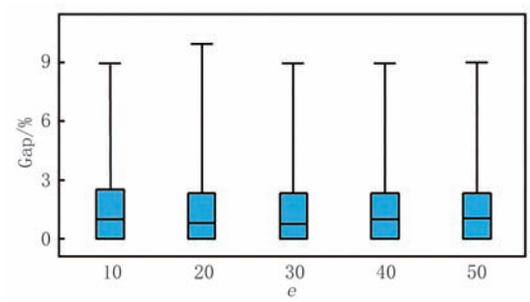

图9　实验参数$n=20$, $m=6$, $p=3$

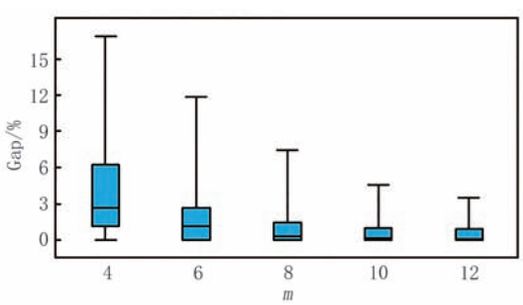

图10　实验参数$n=20$, $e=10$, $p=3$

对响应时间精度的影响巨大. 随着内核数量的增加，响应时间界差Gap_3逐渐减小，其原因在于，M_2虽然能够精确计算复杂任务的WCRT，但计算复杂度也较高. 内核数量增加，节点的并行执行减少了节点间的依赖和冲突，M_3方法能够更快地给出近似解，表现出与M_2方法接近的效果. 在最好情况下，M_2的精度提高了17.2%.

如图11所示，随着节点并行度的增大，Gap_3以较为缓慢的趋势增加. 原因如下，M_2方法在计算优先级DAG任务的WCRT时，通过构建可满足性约束，精准地描述节点间的依赖和优先级，其捕捉并发行为和冲突的能力，确保了高并行度情况下WCRT的精确度. 在最好情况下，M_2的精度提升了近15%.

上述实验结果表明，与M_3方法相比，M_2能够有效提升响应时间界限的精度. 在平均情况下，响应时间界限的精度提升了2%，最好情况下，精度

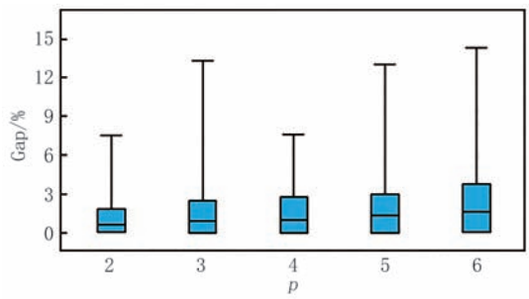

图11　实验参数$n=20$, $e=10$, $m=6$

提升近15%. 响应时间界限精度的提升，对实时系统至关重要. 原因如下：实时系统中的响应时间直接影响着系统的可调度性，响应时间精度平均提升2%，大大增强了系统可调度的能力. 在实际系统中，通常是多DAG任务并行工作的情况. 尽管对于单DAG任务，M_2方法带来的精度提升有限，但对于包含多个DAG任务的实时系统而言，如果每个DAG任务的响应时间精度都提升2%，其累计的精度也是很可观的.

7　总结与展望

响应时间分析(针对DAG任务)是实时领域中最重要的问题之一. 现有优先级DAG任务的响应时间分析方法都是关于WCRT界限的研究，与实际的WCRT存在较大差距. 尽管已有研究给出关于DAG任务精确WCRT的计算方法，但并不适用于具有优先级的DAG. 本文针对带有优先级的DAG任务，使用SMT技术给出了优先级列表调度算法下DAG任务的精确WCRT. 实验结果表明，本文提出的方法在保证WCRT计算精度前提下，大大提高了计算效率. 但在某些情况下，计算效率提升并不明显，仍有可提升空间. 在未来，计划提出更有效的技术提升SMT模型的计算效率.

作者贡献声明　李峰、毕冉对本文贡献相同，为共同第一作者.

参　考　文　献

[1] CilkPlus, https://software. intel. com/en-us/intel-cilk-plus-support

[2] OpenMPA. OpenMP application program interface version 5.2 (2021)

[3] Pheatt C. Intel® threading building blocks. Journal of Computing Sciences in Colleges, 2008, 23(4): 298-298

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

[5] Nasri M, Nelissen G, Brandenburg B. Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling//Proceedings of the 31st Conference on Real-Time Systems. Stuttgart, Germany, 2019: 1-23

[6] Chang S, Sun J, Hao Z, et al. Computing exact WCRT for typed DAG tasks on heterogeneous multi-core processors. Journal of Systems Architecture, 2022, 124: 102385

[7] Liang H, Jiang X, Guan N, et al. Response time analysis and optimization of DAG tasks exploiting mutually exclusive execution//Proceedings of the 60th ACM/IEEE Design Automation Conference. San Francisco, USA, 2023: 1-6

[8] He Q, Guan N, Lv M, et al. The shape of a DAG: Bounding the response time using long paths. Real-Time Systems, 2023, 60(2): 1-40

[9] Ueter N, Günzel M, von der Brüggen G, et al. Parallel path progression DAG scheduling. IEEE Transactions on Computers, 2023, 72(10): 3002-3016

[10] Li F, Bi R, Wang J, et al. VPSS: A DAG scheduling heuristic with improved response time bound. Journal of Systems Architecture, 2024, 148: 103084

[11] Vargas R, Quinones E, Marongiu A. OpenMP and timing predictability: a possible union? //Proceedings of the Design, Automation and Test in Europe Conference and Exhibition. Grenoble, France, 2015: 617-620

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

[13] Sun J, Guan N, Wang Y, et al. Real-time scheduling and analysis of OpenMP task systems with tied tasks//Proceedings of the IEEE Real-Time Systems Symposium. Paris, France, 2017: 92-103

[14] Sun J, Guan N, Wang X, et al. Real-time scheduling and analysis of synchronous OpenMP task systems with tied tasks// Proceedings of the 56th Annual Design Automation Conference. Las Vegas, USA, 2019: 1-6

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

[16] Serrano M, Quinones E. Response-time analysis of DAG tasks supporting heterogeneous computing//Proceedings of the 55th Annual Design Automation Conference. San Francisco, USA, 2018: 1-6

[17] Han M, Guan N, Sun J, et al. Response time bounds for typed DAG parallel tasks on heterogeneous multi-cores. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(11): 2567-2581

[18] He Q, Guan N, Guo Z. Intra-task priority assignment in real-time scheduling of DAG tasks on multi-cores. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(10): 2283-2295

[19] Zhao S, Dai X, Bate I, et al. DAG scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency//Proceedings of the IEEE Real-Time Systems Symposium, Houston, USA, 2020: 128-140

[20] He Q, Lv M, and Guan N. Response time bounds for DAG tasks with arbitrary intra-task priority assignment//Proceedings of the Euromicro Conference on Real-Time Systems. Online, 2021: 1-21

[21] Duran A, Teruel X, Ferrer R, et al. Barcelona OpenMP tasks suite: A set of benchmarks targeting the exploitation of task parallelism in openmp//Proceedings of the International Conference on Parallel Processing. Vienna, Austria, 2009: 124-131

[22] Müller M, Baron J, Brantley W, et al. SPEC OMP2012—an application benchmark suite for parallel systems using OpenMP//Proceedings of the 8th International Workshop on OpenMP. Rome, Italy, 2012: 223-236

[23] Gajinov V, Stipić S, Erić I, et al. Dash: a benchmark suite for hybrid dataflow and shared memory programming models// Proceedings of the 11th ACM Conference on Computing Frontiers. Cagliari, Italy, 2014: 1-11

[24] Bull J, Enright J, Ameer N. A microbenchmark suite for mixed-mode OpenMP/MPI//Proceedings of the 5th International Workshop on OpenMP. Dresden, Germany, 2009: 118-131

[25] Bull J, Reid F, McDonnell N. A microbenchmark suite for OpenMP tasks// Proceedings of the 8th International Workshop on OpenMP. Rome, Italy, 2012: 271-274

[26] Sun J, Li F, Guan N, et al. On computing exact WCRT for DAG tasks//Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC). Online, 2020: 1-6

[27] Burmyakov A, Bini E, Tovar E. An exact schedulability test for global FP using state space pruning//Proceedings of the 23rd International Conference on Real Time and Networks Systems. Lille, France, 2015: 225-234

[28] Pathan R, Voudouris P, Stenström P. Scheduling parallel real-time recurrent tasks on multicore platforms. IEEE Transactions on Parallel and Distributed Systems, 2017, 29(4): 915-928

[29] Voudouris P, Stenström P, Pathan R. Timing-anomaly free dynamic scheduling of task-based parallel applications//IEEE Real-Time and Embedded Technology and Applications Symposium. Pittsburgh, USA, 2017: 365-376

[30] Yalcinkaya B, Nasri M, Brandenburg B B. An exact schedulability test for non-preemptive self-suspending real-time tasks//Proceedings of the Design, Automation and Test in Europe Conference and Exhibition. Florence, Italy, 2019: 1228-1233

[31] Fonseca J, Nelissen G, Nelis V, et al. Response time analysis of sporadic DAG tasks under partitioned scheduling// Proceedings of the 11th IEEE Symposium on Industrial Embedded Systems (SIES). Krakow, Poland, 2016: 1-10

[32] Casini D, Biondi A, Nelissen G, et al. Partitioned fixed-priority scheduling of parallel tasks without preemptions// Proceedings of the IEEE Real-Time Systems Symposium. Nashville, USA, 2018: 421-433

[33] Slim B, Liliana C, Mezouak M, et al. Probabilistic schedulability analysis for real-time tasks with precedence constraints on partitioned multi-core//Proceedings of the 23rd International Symposium on Real-Time Distributed Computing. Nashville, USA, 2020: 142-143

[34] Bi R, He Q, Sun J, et al. Response time analysis for prioritized DAG task with mutually exclusive vertices//Proceedings of the IEEE Real-Time Systems Symposium (RTSS). Houston, USA, 2022: 460-473

[35] Chen N, Zhao S, Gray I, et al. Precise response time analysis for multiple DAG tasks with intra-task priority assignment// Proceedings of the IEEE 29th Real-Time and Embedded Technology and Applications Symposium. San Antonio, USA, 2023: 174-184

[36] He Q, Sun J, Guan N, et al. Real-time scheduling of conditional DAG tasks with intra-task priority assignment. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023, 42(10): 3196-3209

[37] Dick R, Rhodes D, Wolf W. TGFF: Task graphs for free// Proceedings of the 6th International Workshop on Hardware/ Software Codesign. Seattle, USA, 1998: 97-101

LI Feng, Ph. D. candidate. His research interest is real-time systems scheduling.

BI Ran, Ph. D., associate professor. Her research interests include real-time systems scheduling, scheduling optimization.

MA Ye, Ph. D., assistant professor. His research interest is real-time systems scheduling.

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

LI Xi-Sheng, M. S. candidate. His research interests include real-time systems scheduling, algorithm optimization.

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

Background

Nowadays 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 fully parallelized, such that an individual task is able to potentially utilize more than one core at the same time during its execution. Parallel tasks are commonly supported by almost all modern parallel programming languages, e. g., Cilk family, OpenMP and Intel's Thread Building Blocks. The primitives in these languages and libraries, such as parallel for, OMP task and spawn/sync, result in intra-task parallelism structures that can be well represented via Directed Acyclic Graph (DAG) task models, which have gained much attention in the past few years.

In the real-time community, researchers focus on the worst-case response time (WCRT). Graham proposes a famous WCRT bound for a DAG task G. Graham's bound has been widely used in the real-time analysis of DAG tasks. However, Graham's bound which is an upper bound of the WCRT may be pessimistic in practice. To eliminate the pessimism brought by Graham's bound, Sun et al. have proposed a method to get the exact WCRT of a DAG task which aims to formulate the response time analysis problem into an optimization problem. However, their method doesn't consider the priority of the vertices in DAG.

This paper presents a response time analysis method for DAG tasks with priorities used SMT techniques. The method solved the maximum response time among all possible schedules under a certain given scheduling algorithm. Encoding the scheduling strategies into the constrains of the optimization problem is the main challenge of this paper. Compared with the existing methods, the method in this paper can not only ensure improved

accuracy over the Graham bound, but also exhibit a substantial enhancement in computational efficiency.

This work was supported by the National Natural Science Foundation of China (No. 62472063, No. 62072085) and the Natural Science Foundation of Hebei Province (No. F2024501037), which aims to analyze the theory, models, algorithms, and applications of real-time scheduling for OpenMP task graphs.

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 the Chinese Journal of Computers, IEEE Transactions on Computers, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, ACM Transactions on Embedded Computing Systems, Real-Time Systems, Design Automation Conference, Real-Time and Embedded Technology and Applications Symposium, Embedded and Real-Time Operating Systems, EMSOFT, DATE, etc.