计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220711
60(5): 992−1001, 2023

异构多核全局限制性可抢占并行任务可调度分析

韩美灵[1] 孙施宁[2] 邓庆绪[2]
[1](南京邮电大学现代邮政学院 南京 210023)
[2](东北大学计算机科学与工程学院 沈阳 110819)
(meilinghan@njupt.edu.cn)

Schedulability Analysis of Parallel Tasks Under Global Limited Preemption on Heterogeneous Multi-Cores

Han Meiling[1], Sun Shining[2], and Deng Qingxu[2]
[1](*School of Modern Posts*, *Nanjing University of Posts and Telecommunications*, *Nanjing* 210023)
[2](*School of Computer Science and Engineering*, *Northeastern University*, *Shenyang* 110819)

Abstract The heterogeneous multi-core architecture takes advantage of the strengths of different architectures to execute specific types of workloads and is typically more energy-efficient and faster. However, it is very difficult to migrate to large-scale heterogeneous platforms, and large-scale parallelism will lead to a high level of complexity in software scheduling. Although there have been related studies on the DAG task model, the research on the limited preemption scheduling strategy based on the DAG task model still has some shortcomings. In this paper, we present a worst-case response time (WCRT) analysis method for typed DAG tasks on heterogeneous multi-cores under global fixed priority limited preemption scheduling, in which each vertex of the DAG is allowed to execute on only one type of core. We present a method for analyzing the schedulability of multiple DAG tasks under global fixed-priority preemption scheduling based on the latest schedulability analysis of a single DAG task. First, a method to analyze the upper bound workload of higher priority tasks is proposed. Following that, a method for analyzing the upper bound workload of lower priority tasks is proposed. A pseudo-polynomial analysis method is proposed for analyzing the upper bound of WCRT of each typed DAG task based on the state-of-the-art method for single typed DAG task analysis. The results of our experiments with randomly generated workloads indicate that our proposed method is capable of analyzing a task set in a reasonable amount of time. In addition, the results of the scheduler meet all expectations.

Key words response time analysis; heterogeneous multi-cores; parallel task model; embedded real-time systems; schedulability analysis

摘 要 异构多核平台可以利用不同类别体系结构的处理器来执行特定任务,从而达到提高性能和降低功耗的目的.然而,向大规模异构平台迁移极其困难,且大规模的、必要的程序并行会导致软件调度的复杂度.虽然,基于有向无环图(directed acyclic graph, DAG)并行任务模型已有相关的研究工作,但是基于DAG任务模型的限制性可抢占的调度策略研究仍存在不足.鉴于此,主要讨论了DAG任务在异构平台上进行全局固定优先级限制性可抢占调度时的最差响应时间(worst case response time,WCRT)分析,对并行

收稿日期: 2022-08-16; **修回日期**: 2023-02-28
基金项目: 国家自然科学基金项目(62002173, 62072085);南京邮电大学引进人才自然科学研究启动基金项目(NY219167)
This work was supported by the National Natural Science Foundation of China (62002173, 62072085) and the Natural Science Research Start-up Foundation of Recruiting Talents of Nanjing University of Posts and Telecommunications (NY219167).
通信作者: 邓庆绪 (dengqx@mail.neu.edu.cn)

任务的每个结点可用的处理器资源进行了一定的限制，即只能执行在规定类型处理器上的任务. 基于最新的单分类并行任务的可调度性分析，提出了多个并行任务的可调度性分析. 进一步，提出了高优先级任务的干涉量与低优先级任务的阻塞量的计算方法；结合最新的分类并行任务的任务内干涉计算方法，最终提出了一种伪多项式的分析方法. 实验结果表明，提出的算法能够在合理的时间范围内得到任务集可调度性的分析结果，且任务集的接受率随各个参数的变化符合预期.

关键词 响应时间分析；异构多核；并行任务模型；嵌入式实时系统；可调度性分析

中图法分类号 TP391

随着嵌入式技术的发展，云计算、边缘计算等复杂计算场景对芯片的性能要求越来越高. 为了满足该应用需求，异构多核在现代计算机系统得到迅速发展. 然而，向大规模异构平台迁移非常困难. 软件层面的原因主要是大规模的、必要的程序并行导致软件调度的复杂度，即使在同构平台下这依然是大量学者致力研究的一个热点问题.

本文首次针对有向无环图（directed acyclic graph, DAG）并行任务模型在异构平台上进行全局固定优先级限制性可抢占的可调度性分析，所提到的异构平台由多个不同类型的处理器组成（每个类别的处理器个数大于等于 1）. 由于异构融合的体系结构，多种不同类型的处理器集成在同一硬件，且并行软件中某些程序片段只能在特定的处理器执行. 因此，DAG 任务的某些结点可以绑定到特定的一类处理器上执行. 而绑定处理器的操作，在一些主流的并行处理程序和操作系统都有对应的指令，例如：在 OpenMP 中可以通过 proc_bind 实现线程的绑定[1]；在 OpenCL 中，通过 clCreatCommand 可以针对特定设备创建一组命令[2]；在 CUDA 中，可以通过 cudaSetDevise 实现指令到目标设备的绑定[3]. 而对任务的如此处理可以避免任务在不同速率的处理器上迁移，减少实现难度和不必要的迁移开销. 下文中，我们称绑定处理器的 DAG 任务为分类 DAG 任务.

在经典的调度策略中，一般忽略了任务抢占和迁移的开销. 然而，可抢占调度中频繁发生的抢占所造成的开销往往比不可抢占调度高很多，需要高度重视. 尤其在异构多核体系结构中，GPU 中执行的任务片段是不可抢占的，因此一些实时系统的任务存在不可抢占域（即任务执行的某个片段不允许被抢占）. 因此，限制性可抢占的研究对于实时系统的任务执行是至关重要的.

关于分类 DAG 任务集的调度问题，难点主要存在 2 方面：1）DAG 任务内部不同结点的互相影响；2）不同 DAG 任务结点间的互相影响. 尤其是限制性可抢占导致优先级翻转，即低优先级任务会对高优先级任务结点的执行造成阻塞. 因此，分类 DAG 的某个结点的执行会受到自身其他结点的干涉、高优先级任务的干涉以及低优先级任务的阻塞. 其中，自身结点的干涉已经在早期工作[4] 中解决. 本文重点解决高优先级任务的干涉、低优先级任务的阻塞的分析问题以及分类 DAG 的最差响应时间（worst case response time,WCRT）分析，主要创新点包括 3 个层面：

1）由于分类导致 DAG 任务的关键路径（得到 WCRT 的路径）不确定[4]，从而增加高优先级任务干涉分析的难度. 结合经典的同构 DAG 理论和分类 DAG 的特点，本文提出了分类 DAG 高优先级分析方法.

2）解决了分类 DAG 限制性可抢占导致的优先级翻转问题. 本文将同构 DAG 的低优先级任务分析方法和分类 DAG 相结合，提出了分类 DAG 的低优先级任务阻塞的分析方法.

3）采用文献 [4] 的路径抽象方法并结合高、低优先级任务的分析，提出了分类 DAG 任务集的 WCRT 分析方法. 由于高、低优先级任务干涉计算的引入，本文对文献 [4] 路径抽象规则进行了创新.

1 相关工作

在嵌入式实时系统中，某些任务的执行只能在规定的地方被抢占. 考虑到完全可抢占和完全不可抢占的优缺点，限制性可抢占的研究非常重要. 近年来关于这方面的研究也较突出. 如：文献 [5] 的工作针对多核全局最早截止期优先限制性可抢占任务做出了分析；文献 [6−7] 对非并行任务在全局调度策略下进行了响应时间分析；文献 [8] 针对多核并行任务的限制性可抢占问题进行了分析，并根据抢占策略的不同提出了对应的分析方法并进行了比较；文献 [9] 针对 DAG 的限制性可抢占问题进行了研究，特别是针对低优先级任务的阻塞在效率和精确度的不同上提出了不同的分析策略. 然而，限制性可抢占的研

究工作还未考虑到资源的限制性. 随着异构体系结构的发展, 资源限制性是必须要考虑和解决的问题.

资源限制性问题的研究曾经非常热门, 也取得了一些研究成果, 基于当时的条件这些成果考虑的任务模型相对简单. 例如文献 [10−13] 考虑的问题是 2 种处理器集合分配给 2 个任务, 并且一个处理器集合是另一个集合的子集等, 相关研究总结在综述类文献 [14−15]. 可以发现, 现有研究的问题和本文提出的分类 DAG 任务的可调度性问题存在差异. 针对分类 DAG 任务的可调度性研究, 可查阅的文献较少, Han 等人在文献 [4] 中提出单个分类 DAG 的响应时间分析, 该文分析了单 DAG 分类任务可调度性, 并提出了基于路径抽象的方法进行可调度性分析. 基于此研究以及并行任务分析和响应时间分析的经典理论[16-18], Han 等人提出了基于联合策略的可调度性分析[19] 和全局固定优先级可调度性分析[20]. 文献 [21] 提出的工作和本文的任务模型接近, 该工作是基于分解策略提出的点到点(即计算路径中每个点的 WCRT)的调度分析方法, 然而该方法并未考虑资源限制的问题. 就单个分类 DAG 任务而言, 文献 [4] 进行了算法对比, 对比实验结果显示文献 [21] 算法比文献 [4] 算法明显消极.

本文的主要工作是基于文献 [4] 的理论基础, 提出基于全局固定优先级的限制性可抢占调度策略的分类 DAG 任务的可调度性分析方法. 分类导致 DAG 任务的关键路径不确定, 从而增加高、低优先级任务的分析难度. 本文将 DAG 分析的理论(基于路径的分析)和分类 DAG 任务相结合, 成功解决了该难题. 最后, 基于文献 [4] 的路径抽象算法, 更新路径抽象规则, 进行任务集的可调度性判定.

2 任务模型

2.1 任务集模型的建立

本文对由 N 个分类 DAG 任务组成的任务集 $G=\{G_1, G_2, \cdots, G_N\}$ 进行可调度性分析, 该任务集执行的系统由不同种类的核组成. S 是核类别的集合, 对于每种类型 $s \in S$ 具有 M_s($M_s \geqslant 1$)个核用于调度. 分类 DAG 任务 $G_i=(V_i, E_i, C, \gamma_i, T_i, D_i)$, 该元组中各个符号的定义为:

1) V_i 表示 G_i 所有结点的集合, 结点 $v_i^j \in V_i$, $1 \leqslant j \leqslant n_i$, n_i 为 G_i 的结点数;

2) E_i 表示 G_i 所有边的集合, 结点 $(u, v) \in E_i$, $v, u \in V_i$;

3) C 表示 G_i 每个结点最差执行时间(worst case execution time, WCET)的函数, $C\left(v_i^j\right)$ 表示结点 v_i^j 的 WCET;

4) γ_i 表示 G_i 每个结点类型的集合, γ_i^j 表示结点 v_i^j 的类型;

5) T_i 表示 G_i 相邻 2 个实例的最小释放时间间隔;

6) D_i 表示 G_i 的相对截止期, 即 G_i 在时间点 r_i 释放一个任务实例, 必须在 r_i+D_i 内完成该任务实例的执行.

如果存在边 $(u, v) \in E_i$, u 是 v 的直接前继结点, 而 v 是 u 的直接后继结点; 如果存在从 u 到 v 的路径, u 是 v 的祖先结点, 而 v 是 u 的子孙结点. 本文用 $pre\left(v_i^j\right)$, $suc\left(v_i^j\right)$, $ans\left(v_i^j\right)$ 和 $des\left(v_i^j\right)$ 分别表示结点 v_i^j 的前继结点集合、后继结点集合、祖先结点集合以及子孙结点集合. 对于每个 DAG 任务, 都假设其只有 1 个源结点和 1 个结束结点. 如果任务 G_i 具有多个源结点和多个结束结点, 可以通过给 G_i 增加虚拟源结点(结点的 WCET 为 0)和虚拟结点到 G_i 的原源结点的有向边, 使其变成单源结点. 用相同的方法使 G_i 变成单结束结点的任务. $\pi_i=\{v_i^1, v_i^2, \cdots, v_i^k\}$ 是 G_i 的一条路径, 如果路径 π_i 的 v_i^1 是源结点且 v_i^k 是结束结点, 则 π_i 是一条完整的路径. 由于任务是按照类别进行执行的, 需要对每个任务的 WCET 之和进行标记, 具体为:

1) $vol(G_i)=\sum\limits_{1 \leqslant j \leqslant n_i} C\left(v_i^j\right)$, 表示任务 G_i 所有结点的 WCET 之和;

2) $vol^s(G_i)=\sum\limits_{1 \leqslant j \leqslant n_i \wedge \gamma_i^j=s} C\left(v_i^j\right)$, 表示任务 G_i 所有类别为 s 的结点的 WCET 之和;

3) $len(\pi_i)=\sum\limits_{v_i^j \in \pi_i} C\left(v_i^j\right)$ 表示路径 π_i 上的所有结点的 WCET 之和, 即路径 π_i 的长度;

4) $len(G_i)=\max\limits_{\pi_i \in G_i} len(\pi_i)$ 表示任务 G_i 的最长路径.

例 1. 异构 DAG 并行任务模型如图 1 所示, 任务执行在具有 2 种类别的处理器平台, 白色圈表示类别 1, 灰色圈表示类别 2, 结点旁的数值表示任务的 WCET.

2.2 任务调度策略

本文中任务采用全局固定优先级限制性可抢占策略进行调度. 任务按照优先级顺序排列, 即如果 $i<j$, 则 G_i 的优先级比 G_j 高. 限制性可抢占是指任务的结点一旦开始执行, 其执行不会被高优先级任务抢占.

本文将提出一种过量估计方法分析每个任务的 WCRT 上界, 从而保证任务执行的安全性. 其中, 任务 G_i 的 WCRT 上界表示为 R_i.

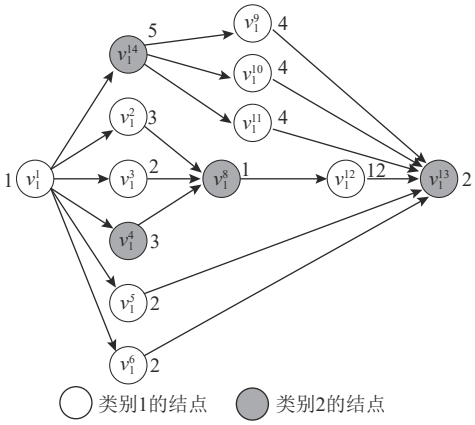


Fig. 1 Illustration of typed DAG task G_1

图 1 分类 DAG 任务G_1的示意图

3 调度算法分析

本节将针对任务集$G=\{G_1, G_2, \cdots, G_N\}$进行可调度性分析，并假设$G_k$是被分析任务，$k=1, 2, \cdots, N$.

3.1 整体分析

DAG 任务 WCRT 上界分析方法多基于关键路径. 同构平台下，最长路径即为关键路径[18]. 然而，这一结论已经不适用于分类 DAG[4]，最长路径分析方法得到的 WCRT 上界并不安全，增大了分类 DAG 的分析难度. 分类 DAG 任务进行可调度性分析需要解决 2 个问题：1）如何确定关键路径；2）已知关键路径，如何进行响应时间分析. 为了确保分析方法的安全可靠，解决问题 1）最简单的方法是枚举所有路径，后面会介绍如何解决枚举问题. 针对问题 2），本文基于路径进行解决. 首先，假设G_k的关键路径是π_k，$\pi_k=\{v_k^1, v_k^2, \cdots, v_k^k\}$.

路径π_k的执行受到 3 种不同类型的干扰：第 1 种，任务内干涉$Intra(\pi_k)$；第 2 种，其他高优先级任务的干涉；第 3 种，低优先级任务的阻塞.

由文献 [4] 的结论可知，路径π_k受到的任务内干涉定义为与π_k结点并行且实际干涉π_k上结点执行的同类结点工作量之和与对应的处理器个数的比值，即

$$Intra(\pi_k)=\sum_{s\in\gamma_k}\sum_{v_k^j\in ivs(\pi_k,s)}\frac{C\left(v_k^j\right)}{M_s}, \tag{1}$$

其中$ivs(\pi_k, s)$在文献 [4] 中定义为路径π_k实际受到的任务内干涉.

根据文献 [9] 可知，路径π_k受到的任务间干涉为路径上所有类别的高优先级任务的工作量加上低优先级任务的阻塞量之和与处理器个数总和的比值，即

$$Inter(\pi_k)=\sum_{s\in\gamma_k}\left\lfloor\frac{W^s(\pi_k)+B^s(\pi_k)}{M_s}\right\rfloor, \tag{2}$$

其中，$W^s(\pi_k)$表示高优先级任务产生的工作量之和的上界，$B^s(\pi_k)$表示低优先级任务产生的工作量之和的上界. 根据式(1)(2)，可以得到如下结论：

引理 1. 路径$\pi_k=\{v_k^1, v_k^2, \cdots, v_k^k\}$是$G_k$的一条完整路径，其 WCRT 上界

$$R(\pi_k)=len(\pi_k)+Intra(\pi_k)+Inter(\pi_k). \tag{3}$$

证明. 根据文献 [9, 18]，引理可证. 证毕.

关键问题是如何计算$Inter(\pi_k)$.$Inter(\pi_k)$包括 2 个部分，即高优先级任务的干涉和低优先级任务的阻塞，下面将对这 2 个部分产生的工作量上界进行分析.

3.2 高优先级干涉量分析

对于高优先级任务的干涉量计算，首先要确定在某个时间窗口里高优先级任务能够产生的每个类别的工作量上界. 在时间长度为x的窗口内，高优先级任务G_i产生的工作量上界表示为$W_i(x)$，产生类别为s的工作量上界表示为$W_i^s(x)$，$W_i^s(x)$计算过程详见引理 2.

引理 2. 在长度为x的时间窗口内，高优先级任务G_i能够产生的类别为s的工作量上界为

$$W_i^s(x)=\left(\frac{x-vol^s(G_i)/M_s}{T_i}+1\right)\times vol^s(G_i)+\alpha, \tag{4}$$

其中$\alpha=\min\left(vol^s(G_i), \max\left(0, M_s\times\left(\left(x-\frac{vol^s(G_i)}{M_s}\right)\bmod T_i-(T_i-R_i)\right)\right)\right)$.

证明. 引理 2 能够得到证明基于 2 个层面的假设：

1）当 carry-in 实例（该实例在窗口之前释放，在窗口内完成）和 carry-out 实例（该实例在窗口内释放，截止期在窗口外）的s类别的工作量平均分布在所有s类核上时（如图 2 所示），可以得到s类别的工作量上界.

2）假设 carry-out 实例全部带入，可以得到高优先级任务的工作量上界，该假设基于假设 1）.

首先，证明假设 1）的正确性. 该假设忽略任务的内部结构，单纯考虑长度为x的时间窗口内能够产生的工作量的最大情况. 采用反证法，如果$vol^s(G_i)$的工作量分布在M_s'个核且$M_s'<M_s$，根据式(4)，该设想显然不能增加窗口内的工作量，因此假设 1）成立.

假设 2）的证明依然采用反证法，假设 carry-out 带入的减少即窗口向左滑动$\varepsilon=\delta\times vol^s(G_i)/M_s$的长度，$0<\delta<1$. 窗口向左滑动，会导致左边 carry-in 实例长度的增加，即 carry-in 实例在窗口内增加ε的长度. 而 carry-in 和 carry-out 实例 s 类的工作量是均匀分布在

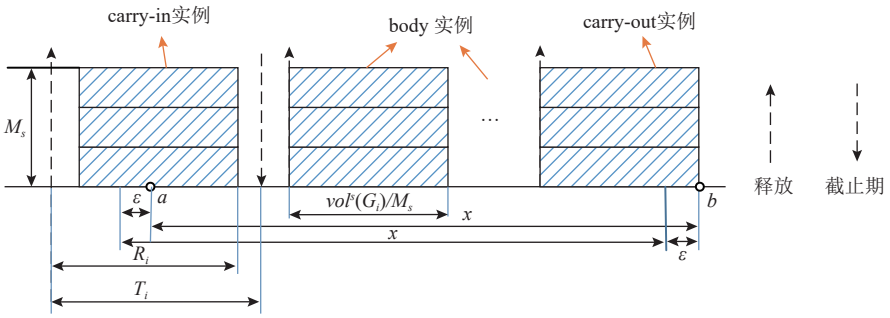


Fig. 2 worst-case scenario to maximum workload in x of G_i

图 2 G_i在 x 窗口内产生最大工作量的最坏情况

所有核上, 所以最终的工作量总和不会发生改变, 即假设 2)成立. 接下来证明式(4)计算的正确性.

图 2 中, x的时间窗口的组成包括: 长度为 $vol^s(G_i)/M_s$的 carry-out 实例、body 实例(释放和截止期都在窗口内的实例)个数的周期长度以及 carry-in 实例长度. 而 body 实例数量的上界为

$$\frac{x-vol^s(G_i)/M_s}{T_i}.$$

x 窗口还有 carry-in 的组成部分. 而 carry-in 的长度可由α计算. 根据图 2, carry-in 长度不大于$vol^s(G_i)/M_s$.

综上, 引理 2 得证. 证毕.

根据引理 2, 在长度为x的时间窗口内, 所有高优先级任务G_i产生的s类别的工作量上界为

$$W^s(x)=\sum_{i<k}W_i^s(x). \tag{5}$$

3.3 低优先级阻塞分析

限制性可抢占导致优先级翻转的问题, 即低优先级任务会阻塞高优先级任务的执行. 当路径π_k上的某个结点v_k^i可以执行时, 即所有前继结点完成执行, 而处理器被低优先级任务占用, 从而造成对该结点的阻塞. 被低优先级任务占用的处理器个数为 1~M_s(少于M_s个是因为处理器被高优先级任务或者前继结点占用), 最坏情况要结合v_k^i的情况进行讨论.

如果$i=1$即该结点是源结点, 最多有M_s个低优先级任务的结点阻塞其执行. 如果$1<i\leqslant n_i$, 分成 2 种情况进行讨论:

1) $\gamma_k^{i-1}=s$, 则最多有M_s-1个低先级任务的结点阻塞其执行. 因为只有M_s-1个低优先级任务的结点和v_k^{i-1}结点并行执行且未执行完造成对v_k^i的阻塞, 如图 3 所示.

2) $\gamma_k^{i-1}\neq s$, 则最多有M_s个低优先级任务的结点阻塞其执行. 因为前继结点的执行不占用 s 类别的核, 因此v_k^i可以执行时最多有M_s个低优先级任务的结点未完成执行, 如图 4 所示.

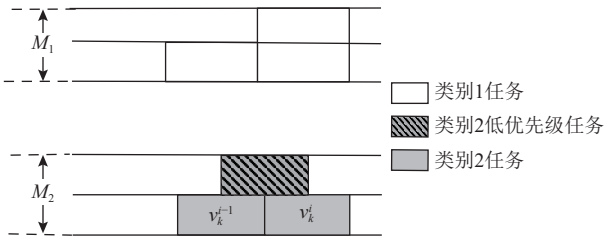


Fig. 3 Scenario for $\gamma_k^{i-1}=s$

图 3 $\gamma_k^{i-1}=s$时的情况

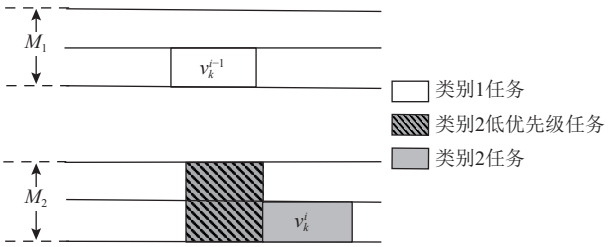


Fig. 4 Scenario for $\gamma_k^{i-1}\neq s$

图 4 $\gamma_k^{i-1}\neq s$时的情况

$\Delta_k^{M_s}$和$\Delta_k^{M_s-1}$分别表示最大的M_s个和M_s-1个不可抢占低优先级结点的工作量之和, 即

$$\Delta_k^{M_s}=\sum\max_{G_{i>k}}^{M_s}\left(\max_{1\leqslant j\leqslant n_j}^{M_s}C\left(v_i^j\right)\right), \tag{6}$$

$$\Delta_k^{M_s-1}=\sum\max_{G_{i>k}}^{M_s-1}\left(\max_{1\leqslant j\leqslant n_j}^{M_s-1}C\left(v_i^j\right)\right). \tag{7}$$

其中: $G_{i>k}$表示所有的低优先级任务; $\max\limits_{G_{i>k}}^{M_s}$和$\max\limits_{G_{i>k}}^{M_s-1}$分别表示所有的低优先级任务中的 M_s个和M_s-1个最大的不可抢占结点的工作量之和; 而$\max\limits_{1\leqslant j\leqslant n_j}^{M_s}$和$\max\limits_{1\leqslant j\leqslant n_j}^{M_s-1}$分别表示$G_i$的$M_s$个和$M_s-1$个最大的不可抢占结点的工作量之和. 式(6)(7)分别计算了所有低优先级任务中最大的M_s个和M_s-1个最大的不可抢占结点的工作量之和, 如此定义是因为最大的结点来自于不同的任务.

综上, 被分析路径上的某个结点阻塞量的计算总结见引理 3.

引理 3. 结点v_k^i是路径$\pi_k=\{v_k^1, v_k^2, \cdots, v_k^s\}$上的第 i

个结点，其类别为$\gamma_k^i=s$，则它的阻塞量上界为

$$\Theta^s\left(\pi_k, v_k^j\right)=\begin{cases}\Delta_k^{M_s}, i=1\text{或}i\neq 1\wedge\gamma_k^{i-1}\neq s.\\ \Delta_k^{M_s-1}, i\neq 1\wedge\gamma_k^{i-1}=s.\end{cases} \tag{8}$$

证明. 根据式(6)(7)，引理3得证.

证毕.

路径$\pi_k=\{v_k^1, v_k^2, \cdots, v_k^k\}$的$s$类的阻塞量之和

$$B^s(\pi_k)=\sum_{v_k^i\in\pi_k\wedge\gamma_k^i=s}\Theta^s\left(\pi_k, v_k^i\right). \tag{9}$$

4 分析过程

基于文献[4]提出的路径抽象算法，进行响应时间分析.

算法基本思路：对于结点v_k^i，每条经过它的路径均可用一个元组表示$\left\langle v_k^i, \Delta\left(\pi_k, v_k^i\right), R(\pi_k, v_k^i)\right\rangle$，其中$\Delta\left(\pi_k, v_k^i\right)$用来记录路径上计算过的并行结点(任务内干涉)，$R(\pi_k, v_k^i)$表示该路径上的结点$v_k^i$的响应时间. 经过$v_k^i$的每条路径都有一个元组与之对应，同样的元组可以被合并成为一个，不同的2个元组如果符合一定的条件可以被其中一个元组替换. 替换的基本原则是被替换掉的元组不会导致比替换它的元组更大的WCRT上界. 经过元组的合并和替换操作过滤掉不必要的元组，达到减少路径的目的. 对DAG任务中的每个结点进行相同操作，直至最终结点. 最终结点上所有元组中最大的$R(\pi_k, v_k^i)$即为任务G_k的WCRT上界.

接下来，对$\Delta\left(\pi_k, v_k^i\right)$，$R(\pi_k, v_k^i)$和替换规则进行定义.

定义1[4]. 对于G_k的一条路径$\pi_k=\{v_k^1, v_k^2, \cdots, v_k^k\}$，$\Delta\left(\pi_k, v_k^i\right)$定义为距离路径上结点$v_k^i$最近的所有类别结点的集合，即

$$\Delta\left(\pi_k, v_k^i\right)=\left\{\delta\left(\pi_k, v_k^i, s\right)|s\in\gamma_k\right\}, \tag{10}$$

其中

$$\delta\left(\pi_k, v_k^i, s\right)=\begin{cases}c(v_i), s=\gamma_k^i,\\ \perp, s\neq\gamma_k^i\wedge i=1,\\ \delta\left(\pi_k, v_k^i, s\right), s\neq\gamma_k^i\wedge 1<i\leqslant k.\end{cases}$$

定义2. 路径$\pi_k=\{v_k^1, v_k^2, \cdots, v_k^k\}$中从源结点开始到$v_k^i$的所有类型的集合，定义为

$$\gamma\left(\pi_k, v_k^i\right)=\underset{1\leqslant j\leqslant i}{\cup}\gamma_k^j. \tag{11}$$

显然，v_k^i为结束结点时$\gamma\left(\pi_k, v_k^i\right)$表示整条路径的类型集合. 本文采用$len(\pi_k, v_k^i)$表示从源结点到$v_k^i$的路径长度，即

$$len\left(\pi_k, v_k^i\right)=\sum_{1\leqslant j\leqslant i}C\left(v_k^j\right). \tag{12}$$

从源结点到v_k^i的所有s类别阻塞量之和为

$$B^s\left(\pi_k, v_k^i\right)=\sum_{1\leqslant j\leqslant i\wedge\gamma_k^j=s}\Theta^s\left(\pi_k, v_k^j\right). \tag{13}$$

定义3. 对于G_k的一条路径$\pi_k=\{v_k^1, v_k^2, \cdots, v_k^k\}$，$R(\pi_k, v_k^i)$定义为

$$R\left(\pi_k, v_k^i\right)=L\left(\pi_k, v_k^i\right)+\sum_{s\in\gamma\left(\pi_k, v_k^i\right)}\frac{W^s\left(len\left(\pi_k, v_k^i\right)\right)+B^s\left(\pi_k, v_k^i\right)}{M_s}, \tag{14}$$

其中$L\left(\pi_k, v_k^i\right)$定义为

$$L\left(\pi_k, v_k^i\right)=\begin{cases}C\left(v_k^1\right), i=1,\\ L\left(\pi_k, v_k^{i-1}\right)+C\left(v_k^i\right)+\sum\limits_{v\in\phi\left(\pi_k, v_k^i\right)}\dfrac{C(v)}{M_s}, 2\leqslant i\leqslant k.\end{cases} \tag{15}$$

$\sum\limits_{v\in\phi\left(\pi_k, v_k^i\right)}\dfrac{C(v)}{M_s}$是文献[4]中关于任务内关涉的计算，与元组的抽象息息相关. 其中，重要计算参数定义为

$$\phi\left(\pi_k, v_k^i\right)=par\left(v_k^i\right)\backslash par(\delta(\pi_k, v_k^{i-1}, \gamma\left(v_k^i\right))),$$

$$par\left(v_k^i\right)=\{v|v\in V_k\wedge v\notin ans\left(v_k^i\right)\cup v\notin des\left(v_k^i\right)\}.$$

由于其他任务干涉的增加，导致文献[4]算法中元组替换的规则不再适用，需要增加新的替换规则满足增加的其他任务干涉的分析，具体见引理4.

引理4. 当任务G_k的2个路径π_k^1和路径π_k^2汇聚在v_k^i结点时，2条路径分别由元组$\left\langle v_k^i, \Delta\left(\pi_k^1, v_k^i\right), R(\pi_k^1, v_k^i)\right\rangle$和$\left\langle v_k^i, \Delta\left(\pi_k^2, v_k^i\right), R(\pi_k^2, v_k^i)\right\rangle$表示，前者为元组1，后者为元组2. 当元组1和元组2满足条件1)~3)时，元组2可以被元组1替换.

1) $\gamma\left(\pi_k^1, v_k^i\right)=\gamma\left(\pi_k^2, v_k^i\right)$;

2) $R\left(\pi_k^1, v_k^i\right)\geqslant R\left(\pi_k^2, v_k^i\right)$;

3) 对于任意类型s，条件①②必须成立：

① $\delta\left(\pi_k^1, v_k^i, s\right)=\perp$;

② $\left(\delta\left(\pi_k^1, v_k^i, s\right)\neq\perp\right)\wedge\left(\delta\left(\pi_k^2, v_k^i, s\right)\neq\perp\right)\wedge\left(par\left(\delta\left(\pi_k^1, v_k^i, s\right)\right)\right)\cap\left(des\left(\delta\left(\pi_k^2, v_k^i, s\right)\right)\right)=\varnothing$;

证明. 替换规则2)和3)已经在文献[4]中证明. 本文只需证明规则1)的必要性. 采用反证法，如果规则1)不成立，则以下2种情况之一成立：

1) $\gamma\left(\pi_k^1, v_k^i\right)\subset\gamma\left(\pi_k^2, v_k^i\right)$. 设路径1和路径2经过结点$v_k^i$，$\gamma\left(\pi_k^1, v_k^i\right)=\{s_1, s_2\}$，$\gamma\left(\pi_k^2, v_k^i\right)=\{s_1, s_2, s_3\}$. 在$v_k^i$到结束结点的搜索过程中，2条路径都不会再增加新的类别. 路径长度在增大，而高优先级任务干涉的计算是非递减的，从而造成s_3类别的高优先级任务造成的干

涉逐渐增大. 因此路径 2 不能被替换.

2) $\gamma\left(\pi_k^1, v_k^i\right) \supset \gamma\left(\pi_k^2, v_k^i\right)$. 设路径 1 和路径 2 经过结点 v_k^i, 到达后继结点 v_k^{i+1}. $\gamma_k^{i+1} = s$ 且 $s \in \gamma\left(\pi_k^1, v_k^i\right) - \gamma\left(\pi_k^2, v_k^i\right)$, 即到达结点 v_k^{i+1} 路径 2 会增加 1 个新的类别 s, 从而导致在结点 v_k^{i+1} 路径 2 增加的高优先级任务干涉比路径 1 增加得多, 因此路径 2 不能被替换.

证毕.

最终, 本文提出的分析方法综合为算法 1, 来判定任务集的可调度性.

算法 1. 任务集可调度性分析算法.

输入: 任务集 G, 处理器池 $M = \{M_s | s \in S\}$;

输出: 是否可以被调度.

① *LimitedPreScheduling*(G,M);

② for each task G_k in G

③ $TS = \left\{\left\langle v_k^1, \left\{v_k^1\right\}, C\left(v_k^1\right)\right\rangle\right\}$;

/* 从任务 G_k 的源结点开始更新元组集合*/

④ while($\exists \langle v, \Delta, R\rangle \in TS; v \neq v_{\text{snk}}$) do

/*针对元组中的每个元组进行下面的计算*/

⑤ for each $v' \in suc(v)$ do

/* 从结点 v 搜索至结点 v'*/

⑥ 基于 $\langle v, \Delta, R\rangle$ 和式(14)计算 $\langle v', \Delta', R'\rangle$;

⑦ if ($\nexists \langle v', \Delta', R'\rangle \in TS$ or $\langle v^*, \Delta^*, R^*\rangle$ 根据代替规则可代替 $\langle v', \Delta', R'\rangle$)

⑧ $TS = TS \cup \{\langle v', \Delta', R'\rangle\}$;

⑨ end if

⑩ end for

⑪ $TS = TS \setminus \{\langle v, \Delta, R\rangle\}$;

⑫ end while

⑬ for each $\langle v, \Delta, R\rangle \in TS$ do

/* 结束结点 v_{snk} 所有元组中具有最大 R 的即为任务 G_k 的 WCRT 上界*/

⑭ $R_{\text{tmp}} = \cup\{R \in \langle v, \Delta, R\rangle\}$;

⑮ $R_k = \max\left(R_{\text{tmp}}\right)$;

⑯ end for

⑰ if $R_k > D_k$

⑱ return 不可调度的;

⑲ end if

⑳ end for

㉑ return 可调度的.

算法 1 针对任务集中的每个任务进行路径抽象, 计算其 WCRT 上界. 算法 1 的行③~⑫是针对每个任务 G_k, 从源结点开始搜索至结束结点, 针对每个结点基于其前继结点的元组进行该结点的元组计算, 计算出新的元组进行元组合并和替换操作, 直至结束结点. 结束结点所有元组中最大的 R 即为 G_k 的 WCRT 上界(行⑬~⑯). 如果 $R_k > D_k$ 说明该任务不可调度, 直接结束计算返回整个任务集不可被调度; 否则, 继续分析下一个任务. 如果最后 1 个任务被分析完, 且满足截止期的要求则整个任务集可被调度(行⑰~㉑).

基于文献 [4], 算法 1 对 DAG 任务每个结点的响应时间分析时增加了其他任务的干涉计算. 在算法 1 过程中增加了新的合并和替换规则以保证最终计算的安全性.

定理 1. 任务集 G 是否可以被调度, 可以由算法 1 来判定.

证明. 综合理论和算法 1 的伪代码分析, 定理 1 得证. 证毕.

5 实 验

本文采用仿真实验, 分别从算法的准确性和效率层面进行算法的验证. 对任务集的可调度性产生影响的参数包括: 任务集利用率、处理器个数、任务集内任务个数、结点个数、类别数以及每个任务的并行度. 首先, 对这些参数设置一组默认参数, 然后基于该默认参数进行对应参数实验变化的验证实验. 异构平台相关的实验数据基于 OpenAMP[22] 项目支持的硬件平台作为基础数据, 对类别数和每种类别核数量的范围进行适当地增大, 观察这 2 组参数对算法性能的影响. 而任务集的默认参数则结合实际情况和其他 DAG 任务集的相关参数进行设置. 默认参数设置为:

1)异构平台上核类别的数量在 [2,15] 随机生成, 每个类别核的数量 M_s 在 [2,10] 随机生成; 每个类别 s 的利用率在 $[1, M_s/3]$ 的范围内随机生成;

2)任务集中, 任务的数量在 [2,20] 的范围内随机生成;

3)用 UUnifast 方法[23] 为每个任务的 s 类别分配利用率;

4)对于每一个任务, 周期在 [100,1 000] 之内随机生成, 默认每个任务的相对截止期等于周期;

5)每次随机选取 1 000 个任务集分析平均性能.

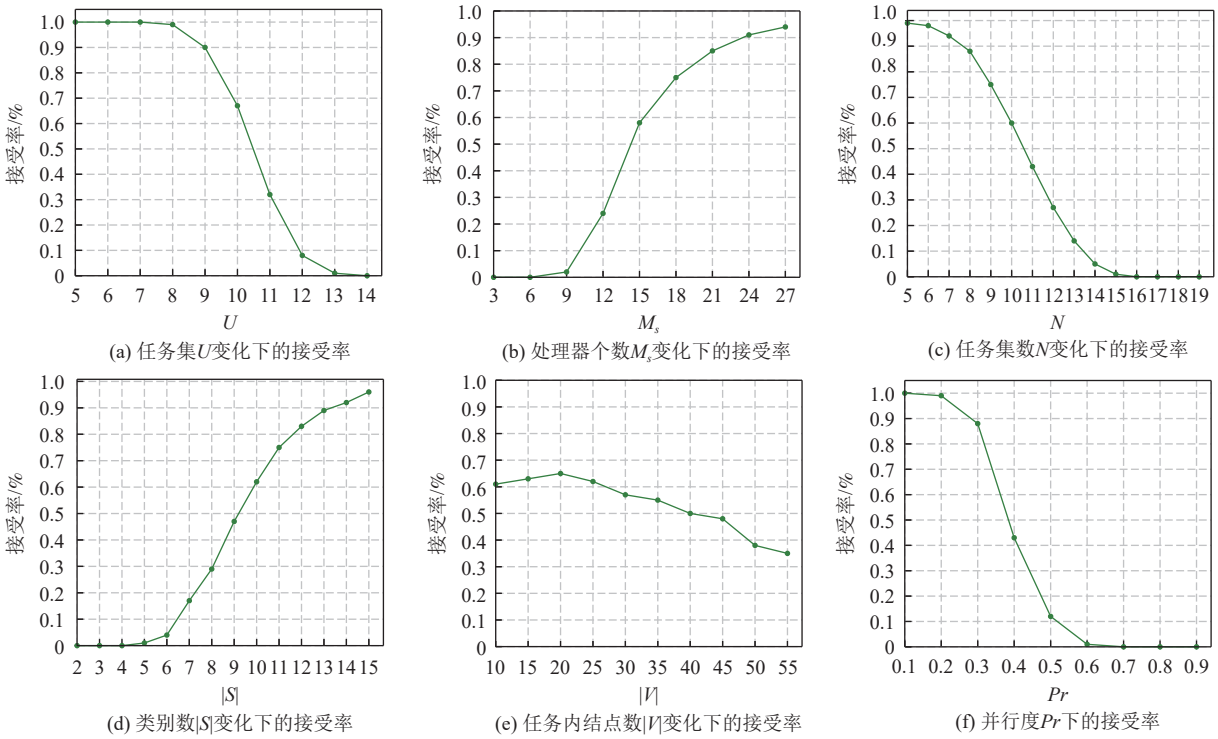
对单个任务的参数进行设置, 主要依据文献 [4], 具体参数设置为:

1)每个 DAG 任务的任务结构采用文献 [24] 提出的方法生成. 任务的结点数在 [15,30] 的范围内随

机生成；任意 2 个结点之间是否生成边，需要随机选定；假设选定为增加该边，需进一步判定，即增加该边后并行度 *Pr* 的值满足要求，则增加边，否则不增加；*Pr* 值在 [0,1] 之间随机选择，注意 *Pr* 值越大说明并行度越低，*Pr* 值为 1 则所有结点为串行.

2）采用 UUnifast 方法将已分配的利用率分配给每个结点，注意利用率是按类别分配的，结点的 WCET 值等于分配的利用率与周期的乘积.

算法的正确性采用接受率进行验证，接受率定义为可以被调度的任务集占所有被测试的任务集的比率. 图 5(a)~(f)展示的是各个参数下的接受率. 其中，图 5(a) ~ (d)是整个任务集参数对接受率的影响，图 5(e) ~ (f)是单个任务中重要参数对接受率的影响. 图 5(a)中，因为利用率增大，而每个类别的处理器个数不变，使得接受率逐渐下降；图 5(b)中，由于利用率固定，每个类别的处理器个数增加，从而导致接受率逐渐提升；图 5(c)表明，其他参数不变的情况下，任务集中任务个数越多任务越难以调度；图 5(d)表明，类别越多，相同利用率下分给每个类别的利用率就会减少，而类别的增加也会导致处理器个数增加，从而使得接受率提升；图 5(e)表明，单个任务内部结点个数对接受率的影响不显著；图 5(f)表明，单个任务内部并行度越高（*Pr* 值越低）接受率越高，这是因为并行度大则并行执行的可能性增大，从而提高接受率. 综上分析，实验数据符合各参数的性质和对实际任务可调度性的影响.



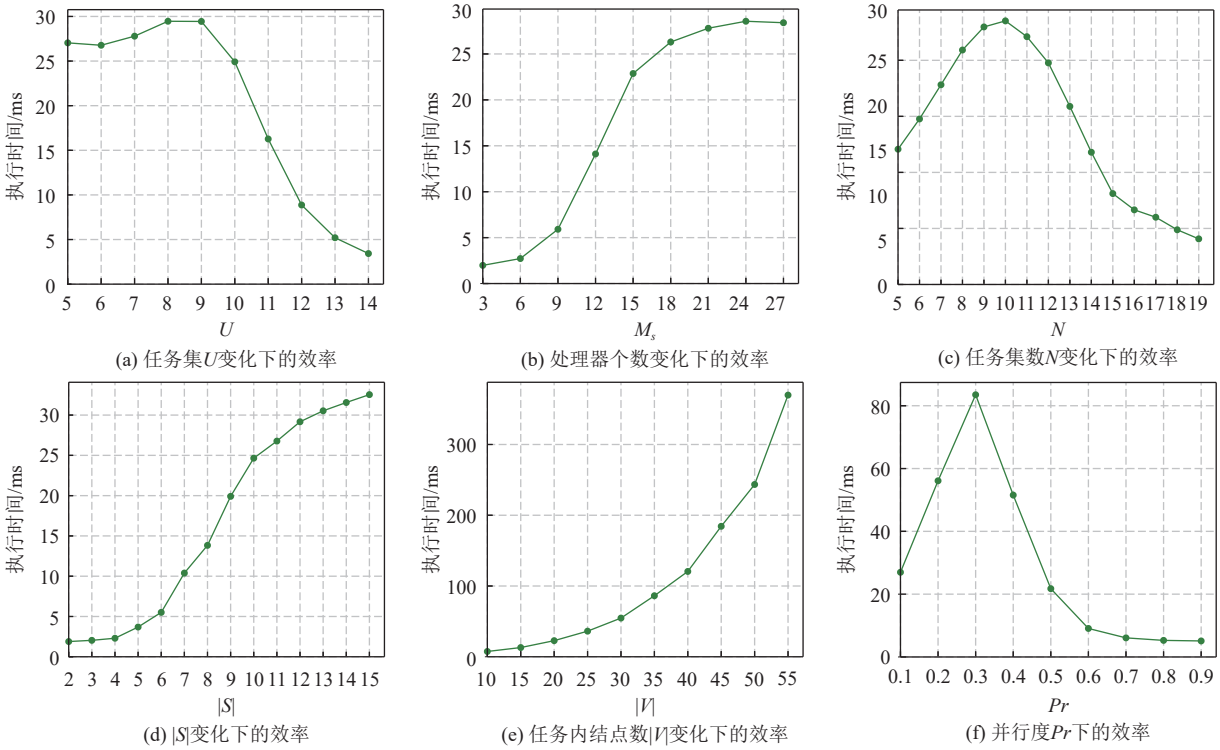
(a) 任务集U变化下的接受率　(b) 处理器个数M_s变化下的接受率　(c) 任务集数N变化下的接受率

(d) 类别数$|S|$变化下的接受率　(e) 任务内结点数$|V|$变化下的接受率　(f) 并行度Pr下的接受率

Fig. 5　Acceptance ratio of tasks under different parameters

图 5　各种参数下的任务接受率

图 6(a) ~ (f)是对应接受率实验的各个参数下的效率实验. 其中，效率趋势中的每个点是所有 1 000 个参加测试的任务集的平均分析时间. 注意，该时间随着分析设备硬件性能的不同会有所不同，其变化趋势可以表明与相关参数的关联性. 同样地，图 6(a) ~ (d)为与任务集相关的参数，而图 6(e)(f)是和单个任务相关的参数. 图 6(a)(b)(d)中时间效率的变化趋势分别和图 5(a)(b)(d)类似，这是因为接受率低，说明不可调度的任务集增多，在第 1 次出现不可调度任务时停止对任务集的分析从而出现平均分析时间下降的现象. 图 6(c)的实验结果为时间效率先升高后降低. 图 6(c)升高趋势的原因是当任务集接受率差不多时，任务集中任务个数增多，导致平均分析时间变长；图 6(c)降低趋势的原因是由于接受率降低太多. 图 6(e)中曲线变化表明，时间效率对任务内的结点个数是敏感的，当任务内的结点数增多时，时间效率呈增长趋势，且增长速度非常大. 图 6(f)中时间效率变化趋势类似于图 6(c)，即先增后降，当任务集都可调度时任务分析时间随并行度下降而升高，当并行度降低到一定程度时接受率下降过多，导致



(a) 任务集U变化下的效率 (b) 处理器个数变化下的效率 (c) 任务集数N变化下的效率

(d) $|S|$变化下的效率 (e) 任务内结点数$|V|$变化下的效率 (f) 并行度Pr下的效率

Fig. 6 Efficiency under different parameters

图 6 各种参数下的效率

任务集大量开始不可调度，从而导致平均分析时间下降.

实验结果表明，本文提出的算法具有很好的接受率且任务的平均分析时间在嵌入式实时系统离线分析时间的可接受范围内，各个参数实验数据均符合实验预期.

6 结 论

本文针对异构平台上任务执行进行资源限制即规定任务只能执行在某一类处理器上这一特殊模型，进行了基于全局固定优先级限制性可抢占调度策略的可调度性分析. 首先，提出了整体的分析思路；然后，针对高优先级任务的干涉和低优先级任务的阻塞进行了分析；最后，结合最新的异构模型提出了一个整体的分析方法. 实验结果表明，本文提出的算法性能良好，能够在有效时间内完成任务的分析，且任务的可调度性和各个参数的关系符合实验预期.

作者贡献声明：韩美灵提出了算法思路并撰写论文；孙施宁负责实现算法并设计了相关实验方案；邓庆绪提出研究思路并指导修改论文.

参 考 文 献

[1] OpenMP. OpenMP application program interface version 4.5[EB/OL]. [2022-11-20].https://www.openmp.org/

[2] OpenCL. Open standard for parallel programming of heterogeneous systems [EB/OL]. [2022-11-20] .https://www.khronos.org/opencl/

[3] CUDA. Free tools and trainings for develpers: CUDA zone[EB/OL]. [2022-11-20].https://developer.nvidia.com/cuda-zone

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

[5] Capodieci N, Cavicchioli R, Bertogna M, et al. Limited-preemption scheduling on multiprocessors[C]//Proc of the 22nd Int Conf on Real-Time Networks and Systems. New York: ACM, 2014: 225−234

[6] Thekkilakattil A, Davis R I, Dobrin R, et al. Multiprocessor fixed priority scheduling with limited preemptions[C]// Proc of the 23rd Int Conf on Real Time and Networks Systems. New York: ACM, 2015: 13−22

[7] Zhou Quan, Li Guohui, Li Jianjun, et al. Response time analysis for tasks with fixed preemption points under global scheduling[J]. ACM Transactions on Embedded Computing Systems, 2019, 18(5): 1−23

[8] Mitra A, Geoffrey N, Gerhard F. An analysis of lazy and eager limited preemption approaches under DAG-based global fixed priority scheduling[C]// Proc of the 20th IEEE Int Symp on Real-Time Distributed Computing. Piscataway, NJ: IEEE, 2017: 193−202

[9] Serrano M A, Melani A, Bertogna M, et al. Response time analysis of DAG tasks under fixed priority scheduling with limited preemptions [C]//Proc of the 19th Design, Automation & Test in Europe Conf & Exhibition. Piscataway, NJ: IEEE, 2016: 1066−1071

[10] Li Chunglun, Li Qingying. Scheduling jobs with release dates, equal processing times, and inclusive processing set restrictions[J]. Journal of the Operational Research Society, 2015, 66(3): 516−523

[11] Li Chunglun, Lee K. A note on scheduling jobs with equal processing times and inclusive processing set restrictions[J]. Journal of the Operational Research Society, 2016, 67(1): 83−86

[12] Jia Zhaohong, Li Kai, Leung J Y T. Effective heuristic for makespan minimization in parallel batch machines with non-identical capacities[J]. International Journal of Production Economics, 2015, 169: 1−10

[13] Li Shuguang. Parallel batch scheduling with inclusive processing set restrictions and non-identical capacities to minimize makespan[J]. European Journal of Operational Research, 2017, 260(1): 12−20

[14] Leung Y T, Li Chunglun. Scheduling with processingset restrictions: A survey[J]. International Journal of Production Economics, 2008, 116(2): 251−262

[15] Leung Y T, Li Chunglun. Scheduling with processing set restrictions: A literature update[J]. International Journal of Production Economics, 2016, 175(8): 1−11

[16] Baruah S, Bonifaci V, Marchetti-Spaccamela A, et al. A generalized parallel task model for recurrent real-time processes [C]// Proc of the 33rd IEEE Real-Time Systems Symp. Piscataway, NJ: IEEE, 2012: 63−72

[17] Fonseca J, Nelissen G, Nélis V. Improved response time analysis of sporadic DAG tasks for global FP scheduling [C]//Proc of the 25th Int Conf on Real-Time Networks and Systems. Piscataway, NJ: IEEE, 2017: 28−37

[18] Guan Nan, Han Meiling, Gu Chuancai, et al. Bounding carry-in interference to improve fixed priority global multiprocessor scheduling analysis[C]// Proc of the 21st IEEE Int Conf on Embedded and Real-Time Computing Systems and Applications. Piscataway, NJ: IEEE, 2015: 11−20

[19] Han Meiling, Zhang Tianyu, Lin Yuhan, et al. Federated scheduling for typed DAG tasks scheduling analysis on heterogeneous multi-cores[J]. Journal of Systems Architecture, 2021, 112: 101870

[20] Peng Xuemei, Han Meiling, Deng Qingxu. Response time analysis of typed DAG tasks for G-FP scheduling [C]// Proc of the 5th Int Symp on Dependable Software Engineering: Theories, Tools, and Applications. Berlin: Springer, 2019: 56−71

[21] Yang Kecheng, Yang Ming, Anderson J H. Reducing response-time bounds for DAG-based task systems on heterogeneous multicore platforms [C]//Proc of the 24th Int Conf on Real-Time Networks and Systems. New York: ACM, 2016: 349−358

[22] OpenAMP. Introduction to OpenAMP library[EB/OL]. [2022-11-20].https://www.openampproject.org/docs/whitepapers/Introduction_to_OpenAMPlib_v1.1a.pdf

[23] Bini E, Buttazzo G C. Measuring the performance of schedulability tests[J]. Real-Time Systems, 2005, 30(1): 129−154

[24] Cordeiro D, Mounié G, Perarnau S, et al. Random graph generation for scheduling simulations [C/OL]// Proc of the 3rd Int ICST Conf on Simulation Tools and Techniques (SIMUTools 2010). ICST, 2010[2018-03-10].https://hal.science/hal-00471255/

Han Meiling, born in 1988. PhD, lecturer. Member of CCF. Her main research interest is embedded real-time scheduling theory.

韩美灵, 1988 年生. 博士, 讲师. CCF 会员. 主要研究方向为嵌入式实时调度理论.

Sun Shining, born in 2001. PhD candidate. His main research interest is embedded real-time scheduling theory

孙施宁, 2001 年生. 博生研究生. 主要研究方向为嵌入式实时调度理论.

Deng Qingxu, born in 1970. PhD, professor, PhD supervisor. Distinguished member of CCF. His main research interests include cyber-physical systems, embedded systems, and real-time systems.

邓庆绪, 1970 年生. 博士, 教授, 博士生导师. CCF 杰出会员. 主要研究方向为物理信息系统、嵌入式系统和实时系统.