# 基于异构多核的多类型 DAG 任务的响应时间分析

常爽爽 赵栩锋 刘震宇 邓庆绪
(东北大学计算机科学与工程学院 沈阳 110004)

**摘 要** 由于异构多核并行架构能够利用不同体系结构的优势来提供更高的性能,近年来受到了广泛的关注.本文是对异构多核平台上多类型 DAG(Directed Acyclic Graph) 任务的最坏响应时间进行分析.多类型 DAG 是一种任务内并行模型,其中包含不同类型的节点,每个节点必须在其指定类型的处理器内核上执行.传统的研究在分析多类型 DAG 任务的最坏响应时间时高估了节点受到的阻塞,导致得到的响应时间上界过于悲观.为此,我们首先提出了一种新的多类型 DAG 任务转化算法,该算法通过将节点拆分成单位节点,并在不破坏原有依赖关系的基础上按照单位节点分配策略在单位节点之间增加新的边,构成一个新的多类型 DAG 任务,从而减少每个节点可能并行执行的节点个数,降低被阻塞时间.在该转化算法的基础上,我们提出了一个新的最坏响应时间分析方法,用来验证支持异构并行计算的多类型 DAG 任务的可调度性.通过对随机生成的多类型 DAG 任务进行的实验表明,我们提出的最坏响应时间上界比现有方法的精确度提高 20%以上.

**关键词** 异构平台;多核嵌入式系统;实时调度;响应时间分析;多类型 DAG 任务
**中图法分类号** TP399 **DOI 号** 10.11897/SP.J.1016.2020.01052

# Response Time Analysis of Typed DAG Tasks on Heterogeneous Multi-Cores

CHANG Shuang-Shuang ZHAO Xu-Feng LIU Zhen-Yu DENG Qing-Xu
(*Department of Computer Science and Engineering, Northeastern University, Shenyang* 110004)

**Abstract** Heterogeneous multi-cores and parallel architectures have recently gained much attention owing to utilize the strength of different architectures for offering higher performance. This paper analyzes the worst-case response time of typed DAG tasks on heterogeneous multi-core platforms, i.e., a typed DAG task contains different types of vertices, and the workload of each vertex must execute on its particular type of cores. Binding code snippets of a program to a specific type of cores is a common operation in heterogeneous multi-cores scheduling and can be easily implemented in mainstream parallel programming frameworks and operating systems. In recent years, scholars domestic and overseas have made great progress in the research of heterogeneous parallel task scheduling with real-time constraints. However, there are still many problems with the existing research results. The traditional research overestimated the interference of the vertices in the DAG when analyzing the worst-case response time of typed DAG tasks, leading to the upper bound of pessimistic response time. For this reason, we propose a typed DAG transformation algorithm, which can obtain a new DAG task by adding extra edges on the basis of retaining the dependency between the original vertices, so as to reduce the number of vertices that can be executed in parallel for each vertex and reduce the possibility of each vertex being blocked. The main idea of the algorithm is to divide vertices into multiple unit-nodes and assign unit-nodes to different sets of parallel unit-nodes according to the principle of minimizing the increment of

收稿日期:2019-08-30;在线出版日期:2020-03-06.本课题得到国家重点研发计划(2018YFB1702003)、国家自然科学基金(61602104,61972076,U1908212,61871107)及兴辽英才计划(XLYC1902017)资助.**常爽爽**,博士研究生,主要研究方向为嵌入式实时系统、并行计算.E-mail:1610539@stu.neu.edu.cn.**赵栩锋**,硕士研究生,主要研究方向为物联网、嵌入式实时系统.**刘震宇**,硕士研究生,主要研究方向为物联网、嵌入式实时系统.**邓庆绪**(通信作者),博士,教授,中国计算机学会(CCF)会员,主要研究领域为物联网、嵌入式实时系统.E-mail:dengqx@mail.neu.edu.cn.

blocking time. When all unit-nodes in the DAG are allocated, the conversion of DAG tasks can be realized by connecting adjacent sets of parallel unit-nodes with edges. Based on this transformation algorithm, we propose a new worst-case response time analysis method to verify the schedulability of typed DAG tasks that support heterogeneous computing. A comparison of randomly generated DAG tasks shows that the accuracy of our new worst-case response time is more than 20% higher than the existing bounds. Based on the actual heterogeneous hardware device platform, we carried out experiments on the actual typed DAG task to evaluate the correctness of the new worst-case response time upper bound DTA proposed by us in the practical application. The experimental results show that the theoretical time obtained by our worst-case response time analysis method is always greater than the actual running time, so the new response time upper bound DTA proposed by us is safe and reliable. The research in this paper has a positive significance to improve the ability to compute the worst-case response time of the typed DAG task in heterogeneous multicores platform. Our future work will focus on extending the existing response time analysis methods to analyze the response time upper bound of multiple typed DAG tasks. At the same time, we will study a more efficient typed DAG task transformation algorithm, which can realize DAG task transformation in a shorter time, and the result is approximate to the upper bound of response time proposed by us.

**Keywords**　heterogeneous platform; multi-core embedded system; real-time scheduling; response time analysis; typed DAG tasks

## 1　引　言

为了满足日益增长的计算性能需求，并行硬件架构已成为多核嵌入式领域的主流. 并行编程模型是利用这些体系结构性能的基础. 近年来，国内外学者对具有实时性约束的并行任务调度问题的研究取得了很大的进展[1]. 一些研究人员开发了多种并行编程范例，如 MPI[2]、OpenMP① 以及并行编程语言 CilkPlus[3]. 所有这些并行编程范例目前都支持任务内并行，即任务由多个可以同时执行的并行代码部分组成. DAG 任务模型是一种很有前景的任务内并行程序模型. 它的实时调度与分析在实时高性能计算领域[4-13]得到了广泛的关注.

此外，由于异构硬件体系结构能够利用不同硬件特有的处理能力，并能提供比同构体系结构更高的性能和效率，受到人们越来越多的关注. 通常情况下，异构硬件架构由性能和功能不对称的设备组成，这些设备将低功耗通用多核处理器（称为主机）与专用的协处理器（例如 Cell/BE SPUs）或数据并行加速器（例如 FPGAs）集成在一起，如 NVIDIA Tegra X1[14]和 Xilinx UltraScale[15]. 目前的并行编程语言倾向于支持异构多核. 例如，在 OpenMP 中，proc_bind 子句可用于指定线程到某些处理内核的映射. 在 OpenCL② 中，clCreateCommandQueue 函数可用于为某些设备创建命令队列. 在本文中，我们研究在异构多核处理器上实时调度多类型 DAG 任务，任务中的每个节点被显式地绑定到特定类型的处理器内核上执行. 将程序的代码片段绑定到特定类型的内核是异构多核调度中的一种常见操作，很容易在主流并行编程框架和操作系统中实现.

文献[16-18]研究了基于异构平台下的多类型 DAG 任务的响应时间分析问题. 这些研究是在 work-conserving 算法下对多类型 DAG 任务进行调度，具体的响应时间分析方法如下. 文献[16]第一次提出了一般的多类型 DAG 任务模型的响应时间上界，但是该上界非常悲观. 文献[17]中提出了对具有约束的多类型 DAG 任务模型的响应时间分析策略. 文献[18]提出了两个新的响应时间上界，其中一个上界在精度上优于文献[16]中的算法，另一个上界通过更加详细地分析 DAG 任务图结构信息，显著提高了响应时间的分析精度. 然而文献[18]的响应时间分析方法依然非常悲观，因为在分析每条路

① Openmp application program interface, version 5.0. http://www.openmp.org 2018

② OpenCL. https://www.khronos.org/opencl/

径的最坏响应时间时,它考虑了路径上节点受到已经完成或尚未执行的同类型不同路径上节点的干扰,这显然是不必要的.文献[19]通过将每个 DAG 任务分解为一组具有人为设置的释放时间和截止日期的独立子任务,进一步分析了多类型 DAG 任务的调度问题.

本文的目的是对多类型 DAG 任务进行研究,得到一个更加精确的响应时间上界.针对先前工作中存在的问题,我们提出了一种新的多类型 DAG 任务转换策略,在保留 DAG 任务中节点原有依赖关系的基础上,通过增加新的边对原 DAG 任务进行转换,转化后得到的新的 DAG 任务中每个节点可能受到阻塞时间的计算会更加精确.基于该 DAG 转化策略,我们提出了一种新的最坏响应时间分析方法,用于验证支持异构计算的多类型 DAG 任务的可调度性.通过对随机生成的多类型 DAG 任务进行的实验表明,我们提出的新的最坏响应时间上界比现有的方法精确度提高 20%以上.

本文第 2 节论述相关工作;第 3 节介绍系统模型;第 4 节列举现有的最坏响应时间分析方法,并分析这些响应时间上界中存在的问题;第 5 节提出一种新的 DAG 转换策略;第 6 节描述一种新的多类型 DAG 任务的最坏响应时间分析方法;第 7 节将我们的方法与现有响应时间分析策略进行实验,验证我们提出的新方法的优越性以及正确性;最后,第 8 节对全文进行总结,并给出今后的研究方向.

## 2 相关工作

为了满足日益增长的处理器性能需求,并行异构硬件体系结构成为嵌入式实时领域的主流.基于 DAG 建模的应用程序调度问题是并行编程模型的基础,也是实现高性能的关键.文献[20-21]提出了经典的一般 DAG 任务(任务内部只包含一种类型的节点)的响应时间上界.在文献[20-21]的基础上,大量国内外学者针对一般 DAG 任务的响应时间上界进行了深入研究.在早期的工作中,对于一般 DAG 任务模型的调度策略分为直接调度策略和基于模型转换的调度策略.在直接 DAG 调度策略中,在不需要了解任务内部结构的情况下,直接将优先级分配给节点,包括全局调度策略[9,12,22]、联合调度策略[7,11,23-24]、划分调度策略[8]以及条件节点调度策略[22,25].基于模型转换的调度策略是对每个 DAG 任务的内部结构进行离线分析.在 DAG 任务中,任何优先级约束都是通过为每个独立的节点分配中间的释放时间和最后截止期来实现的,从而保证它们的独立执行,因此在下一个节点释放之前,前面的所有节点必须完成调度.该方法将并行 DAG 任务调度的难点分解为顺序多核调度[3,26-27].

文献[16]提出了对多类型 DAG 任务调度的最坏响应时间分析的早期研究,该文提出的响应时间上界非常悲观,而且存在非自持续性问题,即当内核数量增加时,最坏响应时间反而可能会增加.文献[18]针对该任务模型进行进一步的研究,提出了两个新的最坏响应时间上界.第一个上界在不增加时间复杂度的情况下提高了文献[16]中响应时间上界的精度,并解决了非自持续性问题.第二个上界研究了多类型 DAG 任务的内部结构信息,进一步提高了响应时间上界的精度,但计算开销比较大.然而,文献[18]中提出的响应时间上界仍然十分悲观,因为当分析每条路径的最坏响应时间时,它考虑了来自已经完成或尚未执行的不同路径上的同类型节点带来的阻塞时间,而这些阻塞不一定发生.文献[17]对具有约束的多类型 DAG 任务模型进行了响应时间分析,该 DAG 任务模型有且只有两种类型的节点(其中一种类型有多个节点,这些节点可以在多核主处理器上执行;另一种类型只有一个节点,该节点在一个单核加速处理设备上执行,称为 *Offload* 节点).文献[17]提出了一个模型转化算法,通过对多类型 DAG 任务进行转换,使与 *Offload* 节点潜在并行执行的节点集转化为在 DAG 图结构中与 *Offload* 节点并行,并在该转换算法的基础上提出了响应时间上界的分析方法.

除此之外,先前也存在大量文献研究不同工作负载在特定处理器上执行的调度问题,例如包含处理集约束[28-29]、区间处理集约束[30-31]和树层次结构处理集约束[32].国内外学者在异构多核处理器上对具有工作负载跟处理器绑定约束的独立任务的调度做了大量的研究,文献[33-34]对这项工作进行了全面的综述.

## 3 系统模型

### 3.1 平台模型

本文讨论的问题基于由 $K$ 种类型的处理器组成的异构多核平台,$\mathcal{C}=\{C_1,\cdots,C_K\}$,$C_k$($k\in[1,K]$)表示第 $k$ 种类型的处理器内核集合.我们用 $m_k$ 来表示第 $k$ 种类型的内核个数,即 $m_k=|C_k|$.当

$\mathcal{C}=\{C_1,C_2\}$且 $m_2=1$，其他参数不变时，我们的平台模型退化成文献[17]中研究的模型.

### 3.2　任务模型

本文研究的多类型 DAG 任务模型可以表示为 $G=(V,E,\Gamma,c)$，其中 $V$ 表示所有节点的集合，$E$ 表示所有边的集合. 每个节点 $v_i\in V$ 代表一段连续执行的代码. 每条边 $(v_i,v_j)\in E$ 代表节点 $v_i$ 和 $v_j$ 存在前驱后继关系. 类型函数 $\Gamma:V\mapsto\mathcal{C}$ 用来定义每个节点的类型. 我们用 $\Gamma(v_i)\in K$ 来表示节点 $v_i$ 的类型，例如 $\Gamma(v_i)=k$ 表示节点 $v_i$ 必须在类型为 $k$ 的处理器内核集合 $C_k$ 上执行. 权重函数 $c:V\times\mathbb{R}_0^+$ 用来定义每个节点的最坏响应时间，例如，类型为 $k$ 的节点 $v_i$ 在 $C_k$ 上的执行时间最多为 $c(v_i)$.

如果存在一条边 $(v_i,v_j)\in E$，那么称节点 $v_i$ 是 $v_j$ 的前驱节点，并且 $v_j$ 是 $v_i$ 的后继节点. 这表示 $v_j$ 直到 $v_i$ 完成以后才可以执行. $pre(v_i)$ 和 $suc(v_i)$ 分别用来表示节点 $v_i$ 的前驱节点集合和后继节点集合. 如果在 $G$ 中存在一条从 $v_i$ 到 $v_j$ 的路径，则称 $v_i$ 是 $v_j$ 的祖先节点，$v_j$ 是 $v_i$ 的后代节点. 我们用 $ans(v_i)$ 和 $des(v_i)$ 分别表示 $v_i$ 的祖先节点集合和后代节点集合. 没有祖先的节点称为源节点 $v_{src}$，没有后代的节点称为汇聚节点 $v_{snk}$. 为了不失一般性，我们假设 $G$ 中有且只有一个源节点和一个汇聚节点. 当 $G$ 中有多个源节点(汇聚节点)时，可以在 $G$ 中加入一个虚拟的执行时间为 0 的源节点(汇聚节点)使其与所有的源节点(汇聚节点)连接以满足我们的模型.

当一条路径 $l=\{v_1,v_2,\cdots,v_q\}$ 包含源节点和汇聚节点时，我们称这条路径为完整路径. 我们用 $L$ 来表示 $G$ 中所有完整路径的集合，$L=\{l_1,l_2,\cdots,l_n\}$. $l_{max}$ 用来表示 $G$ 中的最长路径. $len(l)$ 表示路径 $l$ 的长度，$G$ 中最长路径 $l_{max}$ 的长度可以用 $len(G)$ 表示，计算公式如下：

$$len(l)=\sum_{v\in l}c(v),$$

$$len(G)=\max_{l\in L}\{len(l)\}.$$

如图 1 所示是一个多类型 DAG 任务，它包含两种类型的节点，类型 1 的节点用浅灰色表示，类型 2 的节点用深灰色表示. 每个节点的最坏执行时间用节点旁的数字表示. 最长路径为 $l_{max}=\{v_0,v_1,v_4,v_8,v_{11},v_{12}\}$，最长路径长度为 $len(l_{max})=10$，$len(G)=len(l_{max})=10$.

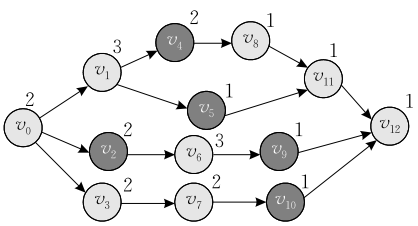



图 1　多类型 DAG 任务图 $G(m_1=2,m_2=1)$

### 3.3　调度模型

我们研究的任务模型是在异构平台 $\mathcal{C}$ 上调度多类型 DAG 任务 $G$. 当节点 $v_i$ 所有的前驱节点都执行结束时，我们说 $v_i$ 是待执行节点. 假设节点 $v_i$ 的类型为 $k$，$v_i$ 必须在处理器内核集合 $C_k$ 上执行. 在本篇论文中，我们基于 work-conserving 调度算法对多类型 DAG 任务 $G$ 进行响应时间分析，即当类型为 $k$ 的节点 $v_i$ 是待执行节点时，$C_k$ 中不允许有空闲的内核. 该调度策略的伪代码如下.

**算法 1.**　work-conserving 调度算法.

输入：DAG 任务 $G$

输出：任务 $G$ 的响应时间

```
1. WHILE G 中存在类型为 k 的待执行节点 v_i
2.     IF C_k 中存在空闲的内核 c_j
3.         在 c_j 上执行 v_i
4.     END IF
5. END WHILE
```

本文的目标是找到多类型 DAG 任务 $G$ 的最坏响应时间的安全上界，可以表示为汇聚节点 $v_{snk}$ 的完成时间和源节点 $v_{src}$ 开始执行时间的差.

## 4　最新研究方法及其悲观性分析

本节首先介绍多核响应时间分析方法的基础理论，然后对现有的异构多核的响应时间分析方法进行介绍，最后分析这些方法的悲观性. 现有的基于异构平台的多类型 DAG 任务的响应时间分析方法都是基于 work-conserving 调度算法. 该调度算法同样适用于同构多核平台下的单一类型的 DAG 任务. 本文以多类型 DAG 任务 $G$ 作为被分析任务进行阐述.

### 4.1　同构平台下 DAG 任务的响应时间分析

文献[21]提出了 $m$ 个内核的同构处理器平台下的基于 work-conserving 调度策略的单类型 DAG 任务的经典响应时间上界.

$$R^{hom}(G)\le len(G)+\frac{vol(G)-len(G)}{m}\qquad(1)$$

式(1)中不等式右侧第 2 项表示 $G$ 中最长路径上节点被阻塞的时间. 当最长路径上的节点 $v_i$ 的所有前驱节点已经完成但是 $C_k$ 中没有空闲的处理器内核时，我们称 $v_i$ 被其他节点阻塞.

### 4.2 异构平台下 DAG 任务的响应时间分析

文献[16]第一次提出了异构平台 $\mathcal{C}$ 下基于 work-conserving 调度策略的多类型 DAG 任务 $G$ 的响应时间上界，如下所示.

$$R(G)\leq len(G)+\sum_{k\in K}\frac{vol_k(G)}{m_k}-\frac{len(G)}{\max\limits_{k\in K}\{m_k\}} \quad (2)$$

$vol_k(G)$表示 $G$ 中类型为 $k$ 的节点的最坏执行时间之和，$vol_k(G)=\sum\limits_{\Gamma(v_i)=k}c(v_i)$.

文献[18]进一步改进了文献[16]的响应时间上界.

$$R(G)\leq\max_{l\in L}\{\widetilde{R}(l)\} \quad (3)$$

$\widetilde{R}(l)$表示任务 $G$ 中完全路径 $l\in L$ 的响应时间，可以通过两种方式进行计算，分别为式(4)和式(5).

$$\widetilde{R}(l)=len(l)+\sum_{k\in K}\frac{vol_k(G)}{m_k}-\sum_{v_i\in l}\frac{c(v_i)}{m_{\Gamma(v_i)}} \quad (4)$$

$m_{\Gamma(v_i)}$表示可以执行类型为 $\Gamma(v_i)$的节点的处理器内核个数. 式(3)和式(4)给出的 $R(G)$的计算方法可以在多项式时间内完成.

$$\widetilde{R}(l)=len(l)+\sum_{k\in K}\sum_{v_i\in \mathrm{IVS}(l,k)}\frac{c(v_i)}{m_k} \quad (5)$$

$\mathrm{IVS}(l,k)$表示任务 $G$ 中类型为 $k$ 但是与路径 $l$ 上类型为 $k$ 的节点不在同一条路径上的节点集合. 如式(5)所示，$\widetilde{R}(l)$包括两个部分：路径 $l$ 的长度以及路径 $l$ 上所有节点被阻塞的时间之和. 文献[18]认为该集合中的节点具有潜在的可能性会与路径 $l$ 上类型为 $k$ 的节点抢夺处理器的使用权，因此可能会阻塞路径 $l$ 上类型为 $k$ 的节点的执行. 式(3)和式(5)给出的 $R(G)$的计算方法可以在伪多项式时间内完成.

文献[17]提出了一种针对异构平台下两种类型节点的 DAG 任务(一种类型有多个节点在多核主处理器上运行，另一种类型只有一个节点在单核加速设备上运行)的最坏响应时间分析方法. 作者根据加速设备上运行的节点不会阻塞主处理器上的节点这一特点，提出了一个 DAG 转化算法使得加速设备上的节点尽可能跟主处理器上的节点并行执行，从而减少主处理器上节点的响应时间，获得更加精确的响应时间上界.

### 4.3 现有的响应时间分析方法中存在的问题

尽管现有的研究已经为异构处理器平台上的多类型 DAG 任务提供了多个有效的响应时间分析方法，但是这些方法仍存在不足之处. 例如，文献[17]关注一种特殊的多类型 DAG 任务的研究，该 DAG 任务中只有两种类型的节点，并且其中一种类型只有一个节点，因此具有很大的局限性. 文献[16]和[18]研究一般的多类型 DAG 任务模型(参见第 3 节). 文献[18]中提出的第一种响应时间计算方法(3)和(4)在精度上明显优于文献[16]中的式(2)，并解决了文献[16]中存在的非自持续性问题. 文献[18]中的第二种响应时间计算方法(3)和(5)，进一步提高了响应时间的上界的分析精度，但它仍然是非常悲观的.

正如式(5)所示，每条路径 $l$ 的响应时间由路径 $l$ 的长度和路径上节点受到节点集合 $\bigcup\limits_{k\in K}\mathrm{IVS}(l,k)$内节点阻塞的时间之和两部分组成. 然而 $\mathrm{IVS}(l,k)$在式(5)中的定义高估了可能对路径 $l$ 中的节点造成阻塞的节点数量，这将在计算路径 $l$ 的响应时间时引入过多的阻塞时间. 事实上，$\mathrm{IVS}(l,k)$中的所有节点并非都可以对路径 $l$ 上同类型节点造成阻塞. 也就是说，如果 $v_j$在路径 $l$ 上的同类型节点 $v_i$开始执行之前已经完成，或者 $v_j$在 $v_i$完成之后才开始执行，则节点 $v_j$不会阻塞 $v_i$. 此外，在 work-conserving 调度算法下，如果与 $v_i$并行执行的节点数小于类型为 $k$ 的内核数 $m_k$，则这些节点可以与 $v_i$同时执行，不会对 $v_i$造成阻塞. 以图 1 中的任务为例，因为在文献[18]中，多类型 DAG 任务由 work-conserving 算法进行调度，图 1 中 DAG 任务 $G$ 的节点可能的执行顺序如图 2 所示.

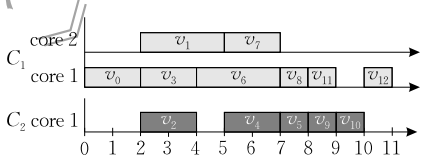



图 2 任务 $G$ 在 work-conserving 算法下可能的执行顺序

如图 1 所示，多类型 DAG 任务 $G$ 中存在路径 $l_1=\{v_0,v_1,v_4,v_7,v_8,v_{11},v_{12}\}$，根据文献[18]有 $\mathrm{IVS}(l_1,1)=\{v_3,v_6,v_7\}$以及 $\mathrm{IVS}(l_1,2)=\{v_2,v_5,v_9,v_{10}\}$. 根据路径的响应时间计算式(5)可知 $\widetilde{R}(l_1)=len(l_1)+\sum\limits_{v_i\in\mathrm{IVS}(l_1,1)}\frac{c(v_i)}{m_1}+\sum\limits_{v_j\in\mathrm{IVS}(l_1,2)}\frac{c(v_j)}{m_2}=18.5$. 由于多类型 DAG 任务 $G$ 是由 work-conserving 算法进行调度，如图 2 所示，路径 $l_1$中的节点 $v_4$开始执行时，$\mathrm{IVS}(l_1,2)$中的节点 $v_2$已经完成，因此 $v_2$不会对 $v_4$造成阻塞. 所以更准确的 $l_1$响应时间是 $R(l_1)\leq 18.5-c(v_2)/m_2=16.5$. 因为文献[18]中的第二种响应时间计算方法式(3)和式(5)是分别计算 $G$ 中每条路径的响应时间，随后从中选取最大的响应时

间的数值作为整个 DAG 任务的最坏响应时间，所以在计算每条路径的响应时间时引入不必要的阻塞时间会最终导致 DAG 任务 $G$ 的最坏响应时间过于悲观.

## 5　多类型 DAG 任务转化算法

通过上一节的分析，我们可以知道每条路径 $l$ 的响应时间计算分为两部分：一部分是路径 $l$ 的长度，另一部分是路径 $l$ 上的节点被阻塞的时间. 因此提高节点受阻塞时间的计算精度是我们工作的重点. 现有的异构多核 DAG 任务响应时间计算方法悲观的主要原因在于高估了每条路径 $l$ 上节点受到的阻塞，这些阻塞来自于 DAG 任务中其他路径上可能会与路径 $l$ 上同类型节点竞争处理器内核的节点集合. 在这一节我们提出一种多类型 DAG 任务转化算法，通过对 DAG 任务的内部结构进行分析，在不破坏节点原有依赖关系的基础上加入新的边，减少 DAG 任务中每条路径 $l$ 上同类型节点竞争处理器内核的节点个数，以减少每条路径 $l$ 上节点的被阻塞时间.

### 5.1　节点拆分

多类型 DAG 任务转化算法的第一步是将多类型 DAG 任务 $G$ 中的每个节点(除了 $v_{\text{src}}$ 和 $v_{\text{snk}}$)拆分为单位执行时间的节点(执行时间为 1，简称为单位节点). 如图 3 所示，我们使用 $v_{i,j}$ 来表示节点 $v_i$ 的第 $j$ 个单位节点. 单位节点的数量等于原节点 $v_i$ 的最坏执行时间，并且单位节点之间由表示依赖关系的边连接. 一个单位节点只能在前驱节点完成后执行.

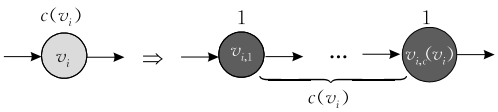



图 3　节点拆分示意图

$G$ 的每个节点都可以被分割成与 $v_{\text{src}}$ 和 $v_{\text{snk}}$ 相连的单位节点，从而构成一个新的 DAG 任务 $G'$. 图 4 为图 1 中 DAG 任务的每个节点(除了 $v_{\text{src}}$ 和 $v_{\text{snk}}$)被拆分为单位节点后的新的多类型 DAG 任务 $G'$.

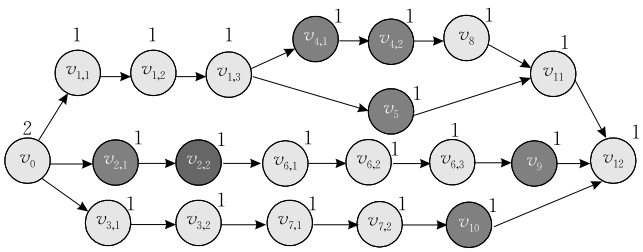



图 4　图 1 中 DAG 任务 $G$ 上节点被拆分后的 DAG 任务 $G'$

节点拆分是为了下一步确定单位节点并行连接的节点集合做准备. 由于 DAG 任务中的节点是以整数时间为单位执行的，因此我们将节点拆分为多个单位节点是合法的. 当多类型 DAG 任务中除了源节点和汇聚节点以外的所有节点的最坏响应时间具有最大公约数 $e$ 时，我们可以将 $e$ 个单位时间作为最小执行单元进行拆分，这样可以大规模减少拆分后多类型 DAG 任务 $G'$ 的规模.

我们根据节点的最坏执行时间，对节点进行拆分. 当节点的实际执行时间小于最坏执行时间时，剩余未执行的单位节点的执行时间为零，不会消耗处理器资源.

### 5.2　多类型 DAG 任务转化策略

在前面的步骤中，DAG 任务 $G$ 通过节点拆分转换为新的 DAG 任务 $G'$. 接下来，我们提出一个 DAG 任务转化策略，通过增加新的边来缩小可能会导致每个单位节点被阻塞的节点集合. 在这里我们给出一个新的定义——并行节点段.

**定义 1**(并行节点段).　假设在 $G'$ 中存在 $Q(G')$ 个并行节点段 $S_\gamma(0\leq\gamma\leq Q(G'))$，$Q(G')=l_{\max}-c(v_{\text{src}})-c(v_{\text{snk}})+2$. 所有的并行节点段之间是串行关系. 每个并行节点段 $S_\gamma$ 内包含多个并行的单位节点，$S_\gamma=\{v_{1,1},\cdots,v_{p,q}\}$.

$Q(G')$ 表示 $G'$ 中最长路径上节点(包括单位节点、源节点以及汇聚节点)的个数. 如图 5 所示，由于任务 $G'$ 中最长路径上节点个数为 9，所以有 9 个并行节点段. 通过 DAG 任务转化策略，$G'$ 最终会被转化为多个并行节点段 $S_\gamma$ 串行相连的形式，来自不同并行节点段的单位节点之间不会相互阻塞. 每个 $S_\gamma$ 内部的单位节点是并行关系，所以在响应时间计算阶段，我们只需要考虑每个并行节点段内部相同类型单位节点之间的阻塞. 因此，下一步我们需要解决的问题是如何将单位节点分配给不同的并行节点段 $S_\gamma$.

#### 5.2.1　多类型 DAG 任务转化算法

算法 2 给出了将单位节点分配到不同并行节点段的伪代码. 分配算法的主要思想是在不违反节点间依赖关系的基础上，为每一个单位节点 $v_{i,j}$ 分配尽可能大的并行节点段的可分配范围，然后从可分配的并行节点段范围内找到一个最佳的并行节点段 $S_\gamma$ 从而完成 $v_{i,j}$ 的分配. 我们通过图 5 举例描述图 4 中任务 $G'$ 的单位节点分配过程.

**算法 2.** 节点分配算法.

输入：节点划分后的 DAG 任务 $G'$

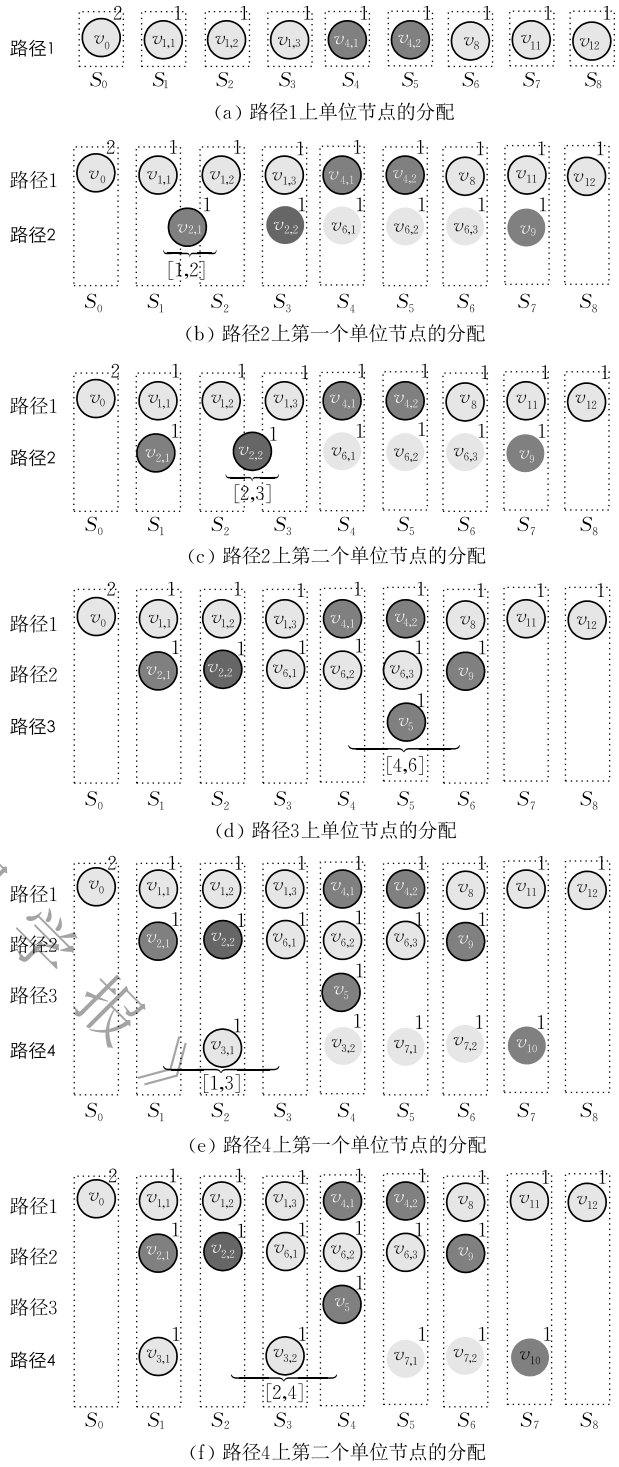
输出：$G'$中所有单位节点分配结果

1. 计算 $G'$中所有路径的长度；
2. $L_{\text{order}} \leftarrow pathLengthOrder(G')$
3. $S(v_{\text{src}})=0, S(v_{\text{snk}})=l_{\max}-c(v_{\text{src}})-c(v_{\text{snk}})+1$；
4. FOR each ($v_i \in l_{\max}/\{v_{\text{src}}, v_{\text{snk}}\}$)
5. FOR ($v_{i,j}, j \in (1, c(v_i))$)
6. $S(v_{i,j})=\lambda(v_{i,j}, l_{\max})$；
7. $S_{S(v_{i,j})}=\{v_{i,j}\}$；
8. END FOR
9. END FOR
10. $H=l_{\max}$；
11. FOR ($l \in L_{\text{order}}$ from second to the last)
12. $M=l \cap H$；
13. IF $M \cap l == l$
14. continue；
15. ELSE
16. FOR each ($v_{i,j} \in M$)
17. 计算 $\lambda(v_{i,j}, l)$和 $\lambda(v_{i,j}, M)$；
18. FOR each ($v_{p,q} \in M$ and $v_{i,j} \neq v_{p,q}$)
19. IF $(\lambda(v_{p,q}, M)-\lambda(v_{i,j}, M))==1$ 并且 $(\lambda(v_{p,q}, l)-\lambda(v_{i,j}, l))>1$
20. $DTF(v_{i,j}, v_{p,q}, l)$；
21. END IF
22. END FOR
23. END FOR
24. END IF
25. END FOR

(1) 计算 $G'$中所有路径的长度(第 1 行)，并利用函数 $pathLengthOrder(G')$按照从长到短的顺序对 $G'$中的路径进行排序，将它们存储在集合 $L_{\text{order}}$中(第 2 行).

(2) 初始化每个并行节点段 $S_\gamma$. 并行节点段的数量为 $Q(G')$. 在第 3 行中，将源节点和汇聚节点分配给第一个和最后一个并行节点段. 将最长路径上的每个单位节点通过循环依次分配给连续的并行节点段(第 4～9 行)，如图 5(a)所示，最长路径上的节点个数决定了并行节点段的数量. 位置函数 $\lambda$ 定义了 $v_{i,j}$在路径 $l$ 的位置(从 0 开始). 对于最长路径 $l_{\max}$上每一个单位节点 $v_{i,j}$，如果 $\lambda(v_{i,j}, l_{\max})=k$ 则把 $v_{i,j}$分配给并行节点段 $S_k$. 路径 $l_{\max}$中的单位节点放入已分配的单位节点集合 $H$ 中(第 10 行).

(3) 按照路径从长到短的顺序对剩余路径进行遍历，并为每条路径上单位节点分配并行节点段(第 11～25 行). 我们按照从长到短的顺序进行分配是为了保证分配的安全性和正确性，避免出现违反节点间原有的前驱后继关系，导致分配的失败. 首先，将路径 $l$ 中已分配并行节点段的单位节点顺序存储在集合 $M$ 中(第 12 行). 如果路径 $l$ 上所有单位节点已经全部分配，则进入下一条路径(第 13～14 行). 否则，分别计算已分配的单位节点在路径 $l$ 和集合 $M$




(a) 路径1上单位节点的分配
(b) 路径2上第一个单位节点的分配
(c) 路径2上第二个单位节点的分配
(d) 路径3上单位节点的分配
(e) 路径4上第一个单位节点的分配
(f) 路径4上第二个单位节点的分配

图 5 任务 $G'$单位节点分配示意图

中的位置(第 15～17 行). 如果在路径 $l$ 上,有两个来自路径 $l$ 的已分配的单位节点 $v_{i,j}$ 和 $v_{p,q}(i\neq p)$,并且它们在 $M$ 中位置是相邻的,而在 $l$ 中不是相邻的. 这意味着在 $l$ 上 $v_{i,j}$ 和 $v_{p,q}$ 之间存在未分配的单位节点(第 18～19 行),那么程序跳转到单位节点分配函数 $DTF(v_{i,j},v_{p,q},l)$(我们将在下一小节单独进行阐述),确定 $v_{i,j}$ 和 $v_{p,q}$ 中间的单位节点被分配的并行节点段(第 20 行). 如图 4 所示,$l_3=\{v_0,v_{1,1},v_{1,2},v_{1,3},v_5,v_{11},v_{12}\}$. 在图 5(d)中,路径 $l_3$ 中已分配节点 $M=H\cap l_3=\{v_0,v_{1,1},v_{1,2},v_{1,3},v_{11},v_{12}\}$. $l_3$ 存在两个已分配的单位节点 $v_{1,3}$ 和 $v_{11}$,它们在 $M$ 中相邻,但在 $l_3$ 中不相邻. 因此,$v_{1,3}$ 和 $v_{11}$ 之间的节点 $v_5$ 是未分配的,接下来跳转到单位节点分配函数 $DTF(v_{1,3},v_{11},l_3)$ 确定 $v_5$ 被分配的并行节点段(从 $S_4$,$S_5$ 和 $S_6$ 中选取).

### 5.2.2　单位节点分配函数 $DTF(u,v,l)$

算法 3 是单位节点分配函数 $DTF(u,v,l)$ 的伪代码. $u$、$v$ 和路径 $l$ 是函数 $DTF(u,v,l)$ 的输入. $u$ 和 $v$ 是路径 $l$ 上被分配了并行节点段的单位节点. 这个函数的目的是为 $u$ 和 $v$ 之间的单位节点分配并行节点段,尽可能地缩小可能会与每个单位节点并行执行的同类型节点集合. 我们用 $[A(w),B(w)]$ 表示路径 $l$ 上未分配的单位节点 $w$ 可以分配的并行节点段的范围. $A(w)$ 和 $B(w)$ 分别用来表示 $w$ 可以分配的最小和最大并行节点段. 例如,$[A(w),B(w)]=[3,5]$ 表示 $w$ 可以分配给 $S_3$,$S_4$ 和 $S_5$. 我们用 $S(w)$ 来表示 $w$ 所分配的并行节点段.

**算法 3.**　单位节点分配函数 $DTF(u,v,l)$.

输入:路径 $l$,$l$ 上的单位节点 $u$ 和 $v$

输出:$u$ 和 $v$ 之间的单位节点的分配结果

```
1.  FOR each (w∈l from λ(u,l)+1 to λ(v,l)−1)
2.    A(w)=B(pre(w))+1;
3.    B(w)=A(v)−λ(v,l)−λ(w,l);
4.    MIN=0; S(w)=A(w);
5.    FOR each (γ∈[A(w),B(w)])
6.      S_γ^*=S_γ∪{w};
7.      FOR each (S'∈{S_γ,S_γ^*})
8.        MAX(S')=0;
9.        FOR each (type k∈Δ(S'))
10.         Compute inf(S',k) by Equation (6);
11.         IF MAX(S')<inf(S',k)
12.           MAX(S')=inf(S',k);
13.         END IF
14.       END FOR
15.       E(S')=1+MAX(S');
16.     END FOR
17.     Ē=E(S_γ^*)−E(S_γ);
18.     IF (MIN>Ē) //MIN 存放最小的增量
19.       MIN=Ē; S(w)=k;
20.     END IF
21.   END FOR
22.   H=H∪{w};
23. END FOR
```

首先,因为 $u$ 和 $v$ 分配结果已知,$A(u)=B(u)=S(u)$,$A(v)=B(v)=S(v)$. 对于路径 $l$ 上 $u$ 和 $v$ 之间未分配的单位节点 $w$,我们先确定第一个未分配节点 $w$ 可分配的并行节点段的范围,分配的原则是在不违反节点间依赖关系的基础上为单位节点分配最大的可分配范围,以便后续从该范围内选择最优的并行节点段进行分配(第 2～3 行). 例如,图 5(b)中路径 $l_2$ 上的未分配节点为 $\{v_{2,1},v_{2,2},v_{6,1},v_{6,2},v_{6,3},v_9\}$,这些节点可以选择并行节点段 $S_\gamma(\gamma\in(1,7))$ 进行分配并且不能违反前驱后继关系. 我们首先为第一个未分配节点 $v_{2,1}$ 计算可分配范围,由于 $v_{2,1}$ 的未分配的后代节点有 5 个,那么 $v_{2,1}$ 可分配的并行节点段的最大范围是 $[A(v_{2,1}),B(v_{2,1})]=[1,2]$.

在得到单位节点 $w$ 的可分配范围之后,我们需要在范围内找到一个最优的并行节点段,使得 $w$ 的到来对该并行节点段的执行时间带来的增量最小($MIN$ 存放最小的增量). $S(w)$ 的初始值设为 $A(w)$(第 4 行). 对于每个并行节点段 $S_\gamma(\gamma\in[A(w),B(w)])$,我们分别计算加入 $w$ 后 $S_\gamma$ 的执行时间增量,随后将 $w$ 分配给增量最小的并行节点段(第 5～21 行). 我们使用 $S_\gamma^*$ 来表示 $w$ 分配给 $S_\gamma$ 后的新并行节点段(第 6 行). 同一并行节点段中具有相同类型的单位节点可能会相互阻塞,因此向 $S_\gamma$ 加入新的单位节点后得到 $S_\gamma^*$ 的执行时间可能增加,增量由第 7～16 行计算. 对于并行节点段 $S'\in\{S_\gamma,S_\gamma^*\}$(第 7 行),$MAX(S')$ 用于表示 $S'$ 中所有类型单位节点受到阻塞时间的最大值(第 8 行),通过第 9～14 行可以得到 $MAX(S')$ 的值. $\Delta(S')$ 表示并行节点集 $S'$ 内节点类型的集合. 对于 $\Delta(S')$ 中的每一类型 $k$,计算 $S'$ 中所有类型为 $k$ 的节点受到的阻塞时间(第 9～10 行). $inf(S',k)$ 表示类型为 $k$ 的单位节点在 $S'$ 中受阻塞的时间,由式(6)计算.

$$inf(S',k)=\left\lfloor\frac{stv(S',k)-1}{m_k}\right\rfloor \tag{6}$$

其中 $stv(S',k)$ 表示 $S'$ 中类型为 $k$ 的单位节点的数量. 如果 $(stv(S',k)-1)\%m_k=x$,$0<x<m_k$,则对于每个类型为 $k$ 的单位节点 $v_{i,j}$,它都能够在内核集

合 $C_k$ 上与其他 $x$ 个单位节点同时执行. 这是因为在 work-conserving 调度算法下,当存在待执行单位节点时,处理器不允许空闲,因此式(6)中存在向下取整符号. 文献[18]的式(5)忽略了这一点.

我们用 $E(S')$ 表示并行节点段 $S'$ 的最坏执行时间,它包含两个部分:$S'$ 的长度和 $S'$ 中所有类型单位节点受到阻塞时间的最大值(第 15 行). $\bar{E}$ 表示 $w$ 加入后,$S_\gamma$ 的最坏执行时间的增量(第 17 行). 计算 $w$ 加入前后增量最小的并行节点段(第 18～20 行). $w$ 分配给增量最小的并行节点段,并将 $w$ 加入已分配完的单位节点集合 $H$ 中(第 22 行). 例如在图 5(e)中,$v_{3,1}$ 的可分配并行节点段的范围是 $[A(v_{3,1}), B(v_{3,1})]=[1,3]$. 如果将 $v_{3,1}$ 分别分配给 $S_1$、$S_2$ 和 $S_3$,给三者带来的执行时间的增量分别是 0、0 和 1. 在 $S_1$ 中只存在一个与 $v_{3,1}$ 具有相同类型的单位节点 $v_{1,1}$,由于处理该类型节点的内核个数为 2,$v_{1,1}$ 和 $v_{3,1}$ 可以并行执行,因此 $v_{3,1}$ 的加入不会影响 $S_1$ 的执行时间,$S_2$ 同理. 而在 $S_3$ 中存在两个与 $v_{3,1}$ 具有相同类型的单位节点 $v_{1,3}$ 和 $v_{6,1}$,$v_{3,1}$ 的加入会使 $S_3$ 执行时间增加 1. 所以 $v_{3,1}$ 被分配给 $S_1$.

当前一个单位节点分配结束后,相邻的下一个单位节点依次重复上述过程,确保每个单位节点都能在其最大的可分配的范围内找到最优的分配结果. 对于同一条路径上的节点,在这条路径上的节点开始分配之前,只有第一个未分配节点的可分配范围可以确定,这条路径上的后续节点的分配范围需要在前一个节点分配完成后才能确定. 图 6 是 DAG 任务 $G'$ 中所有单位节点的分配结果.

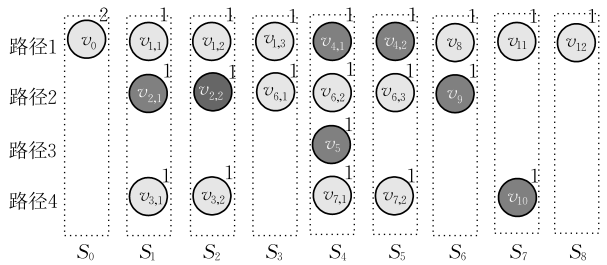
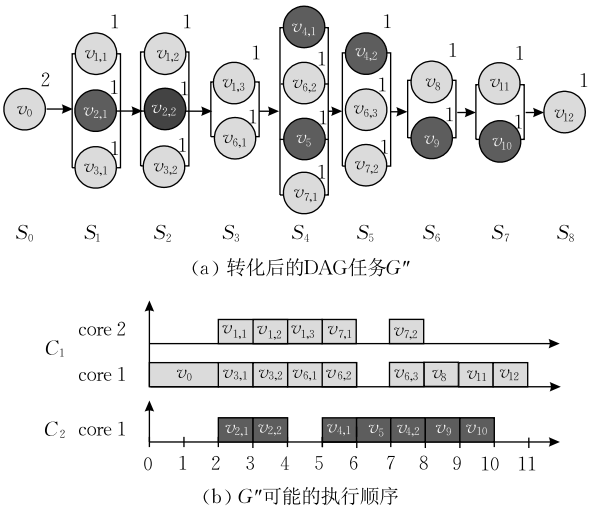



图 6 DAG 任务 $G'$ 上单位节点的分配结果

在为每个节点分配不同的并行节点段之后,我们将所有的并行代码段串行连接,因为图 4 中的多类型 DAG 任务 $G'$ 和图 1 中的多类型 DAG 任务 $G$ 是等价的,图 1 中的多类型 DAG 任务 $G$ 最终转化为如图 7(a)所示的多类型 DAG 任务 $G''$. 图 7(b)是在 work-conserving 调度算法下 $G''$ 可能的执行顺序.

本文提出的多类型 DAG 转化算法的时间复杂




(a) 转化后的DAG任务$G''$

(b) $G''$可能的执行顺序

图 7 转化后的 DAG 任务 $G''$ 及其可能的执行顺序

度为 $O(|V'|^2)$,空间复杂度为 $O(|V'|)$,$V'$ 为进行节点拆分后的 DAG 任务 $G'$ 的节点集合. 当多类型 DAG 任务中所有节点(除了源节点和汇聚节点以外)的最坏响应时间具有最大公约数 $e$ 时,在节点拆分阶段可以将 $e$ 个单位时间作为最小执行单元进行拆分,这样可以大规模减少 $V'$ 中节点的个数,从而降低多类型 DAG 转化算法的时间复杂度.

## 6 多类型 DAG 任务的响应时间分析

在本节,我们在多类型 DAG 转化算法的基础上,提出了一种新的响应时间上界 DTA 来支持异构 DAG 任务并行计算. 通过多类型 DAG 转化算法,原始 DAG 任务被转化为多个并行节点段串行相连的形式,不同并行节点段内部的单位节点不会相互阻塞,因此我们只需要考虑每个并行节点段内部所有同类型的单位节点之间相互阻塞的时间. 在本文中,我们的目标是推导出 work-conserving 调度算法下多类型 DAG 任务在异构多核平台 $\mathcal{C}$ 上的响应时间上界. 响应时间分析方法的主要思想是分别计算每个并行节点段的最坏执行时间,所有并行节点段的最坏执行时间之和就是转化后 DAG 任务 $G''$ 的响应时间上界.

由上一节可知,源节点和汇聚节点中间并行节点段的个数为 $n=len(G'')-c(v_{\text{src}})-c(v_{\text{snk}})$. 并行节点段 $S_\gamma(\gamma\in(1,n))$ 的最坏执行时间 $E(S_\gamma)$ 为

$$E(S_\gamma)=1+\max_{k\in\Delta(S_\gamma)} inf(S_\gamma,k) \tag{7}$$

其中 $\Delta(S_\gamma)$ 表示并行节点段 $S_\gamma$ 中单位节点的类型集合. $inf(S_\gamma,k)$ 表示类型为 $k$ 的单位节点被并行节

点段 $S_\gamma$ 中同类型节点阻塞的时间，由式(6)计算. 式(7)给出了并行节点段 $S_\gamma(\gamma\in(1,n))$ 的最坏执行时间为并行节点段 $S_\gamma$ 中最长路径长度(等式右侧第1项，由于 $S_\gamma$ 中所有单位节点是并行关系，所以为1)和 $S_\gamma$ 里所有类型中相同类型的单位节点的最大阻塞时间(等式右侧第2项)的和.

源节点和汇聚节点所在的并行节点段的最坏执行时间分别为 $c(v_{\text{src}})$ 和 $c(v_{\text{snk}})$，所有并行节点段的最坏执行时间之和就是整个 DAG 任务 $G''$ 的最坏响应时间上界.

**定理 1.**　转化后的 DAG 任务 $G''$ 的最坏响应时间上界为

$$R(G)\leq len(G)+\sum_{\gamma=1}^{n}\max_{k\in\Delta(S_\gamma)} inf(S_\gamma,\gamma) \tag{8}$$

其中 $n=len(G'')-c(v_{\text{src}})-c(v_{\text{snk}})$.

证明.　如果多类型 DAG 任务 $G$ 中节点的实际执行时间小于最坏执行时间，转化后的 $G''$ 的响应时间上界不会大于我们求得的上界. 我们用 $\tilde{c}(v_i)$ 表示节点 $v_i$ 的实际执行时间. 如果 $c(v_i)-\tilde{c}(v_i)>0$，使 $c(v_{i,j})=0(j\in(\tilde{c}(v_i)+1,c(v_i)))$. 当 $v_{i,j}$ 被分配给并行节点段 $S_\gamma$ 时，$S_\gamma$ 的实际执行时间 $\tilde{E}(S_\gamma)\leq E(S_\gamma)$，所以当节点实际执行时间小于最坏执行时间时，实际的响应时间小于或等于我们提出的响应时间上界.　证毕.

我们按照新的响应时间上界计算方法式(8)计算 $G$(图1)转化得到的 $G''$(图4)的响应时间上界：

$$R(G'')\leq len(G'')+\sum_{i=1}^{7}\max_{k\in\Delta(S_\gamma)} inf(S_\gamma,\gamma)=10+1=11.$$

我们基于 DAG 任务转化策略提出的响应时间分析策略，改进了节点受阻塞时间的计算方法，因此能获得更精确的响应时间上界. 本文提出的多类型 DAG 任务最坏响应时间上界的时间复杂度为 $O(|V'|)$，空间复杂度为 $O(1)$.

## 7　实验分析

在本节中，我们通过对比实验评估了我们提出的新的响应时间分析方法的性能. 由于文献[17]中研究的平台模型不同于其他已知的响应时间分析方法，所以我们将对比实验分为两个部分，分别为 7.1 小节和 7.2 小节. 在 7.3 小节，我们基于实际的异构硬件设备平台对实际 DAG 任务进行实验，用来评估我们提出的新的最坏响应时间上界 DTA 在实际应用中的正确性.

### 7.1　基于一般系统模型的对比实验

在本小节，我们通过实验对以下响应时间上界进行评估.

(1) DTA. 我们在定理 1 中给出的响应时间上界.

(2) HAN-1. 文献[18]中的第一个响应时间上界式(3)和式(4).

(3) HAN-2. 文献[18]中的第二个响应时间上界式(3)和式(5).

(4) JEF. 文献[16]中的响应时间上界式(2)，作为其他三个上界归一化的基准.

我们假设多类型 DAG 任务是周期性的，周期为 $T$，最后截止期为 $D$(因此我们不仅评估响应时间上界，还评估每个任务的可调度性). 我们首先定义了默认的参数设置，然后对不同的参数进行调优，并根据不同参数的变化来评估方法的性能. 在本文中，我们使用 Unnifast 方法[35]将多类型 DAG 任务中所有节点总的最坏执行时间分配到每个节点. 默认参数设置定义如下：

(1) 多类型 DAG 任务总的利用率 $U$ 从[1,3]中随机选取，并且 DAG 任务所有节点的最坏执行时间之和为 $vol(G)=U\times T$.

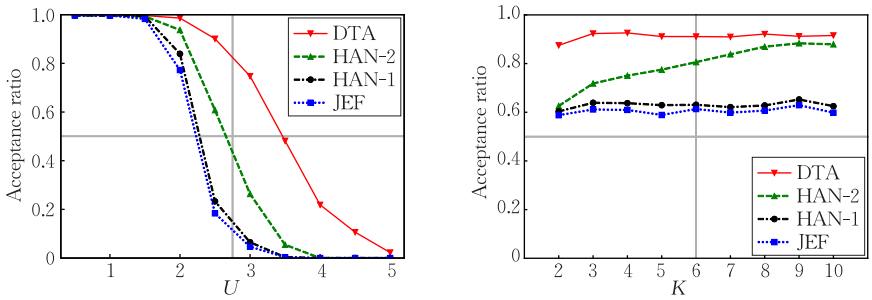
(2) 类型 $K$ 的数量在[2,6]范围内随机选择，每种类型 $K$ 的核心 $C_k$ 的数量在[2,11]中随机选择.

(3) 并行因子 $pr$ 在[0.08,0.1]范围内随机选择($pr$ 表示 DAG 任务中任意两个节点之间存在边的概率).

(4) 我们用文献[36]中的方法生成多类型 DAG 任务的图结构. 对于每个多类型 DAG 任务，总的节点的数量 $|V|$ 从[20,50]范围内随机选择. 每个节点从类型[1,$K$]中随机选取一个数值作为节点要运行的处理器类型.

图 8 显示了根据默认设置，不同响应时间上界的接受率随着参数变化的评估结果. 接受率的定义为可由指定最坏响应时间上界判定为可调度的任务数与生成的总任务数之间的比率. 可调度性的度量是指定最坏响应时间上界得到的任务的响应时间是否小于任务的截止日期. 图 8(a)显示了当任务总利用率 $U$ 从 0.5 增长到 5，指定响应时间上界接受率的变化趋势(其他参数与默认设置相同). 图 8(b)、图 8(c)、图 8(d)分别是指定响应时间上界接受率随类型 $K$、并行因子 $pr$ 和总节点个数 $|V|$ 增长而变化的示意图.

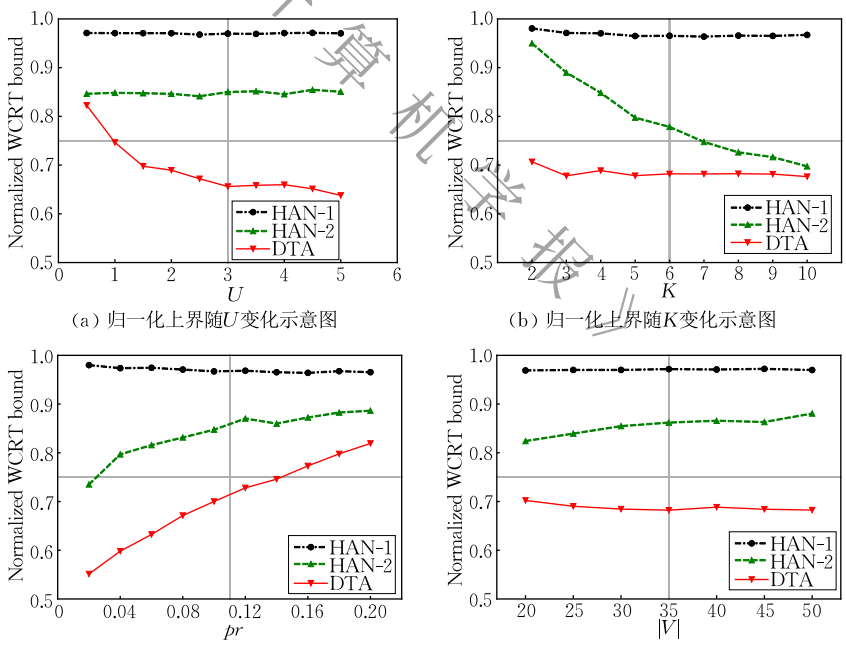
图 9 显示了归一化的响应时间上界 DTA、HAN-1




(a) 接受率随$U$变化示意图 (b) 接受率随$K$变化示意图

(c) 接受率随$pr$变化示意图 (d) 接受率随$|V|$变化示意图

图 8 不同响应时间分析方法的接受率随参数变化的对比图




(a) 归一化上界随$U$变化示意图 (b) 归一化上界随$K$变化示意图

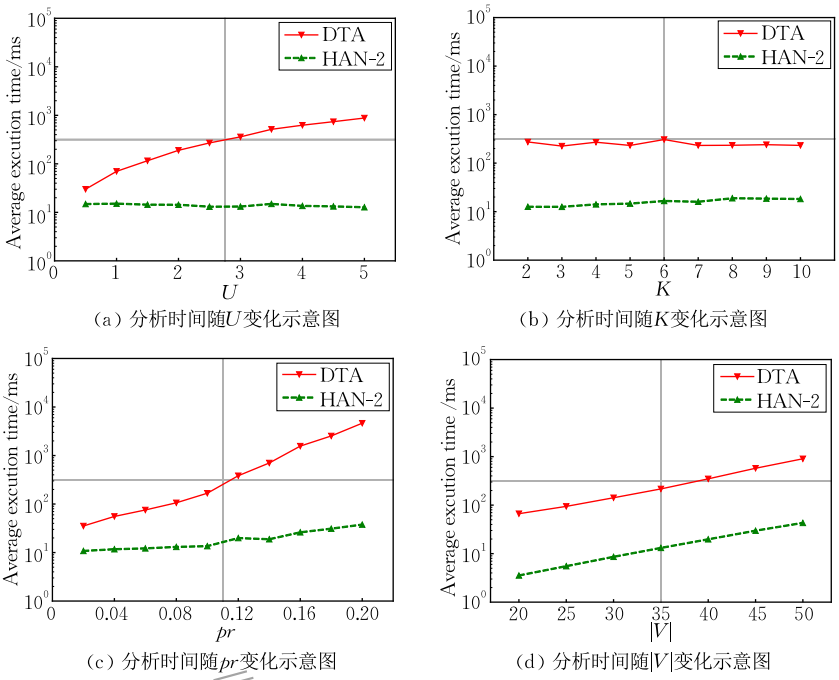
(c) 归一化上界随$pr$变化示意图 (d) 归一化上界随$|V|$变化示意图

图 9 不同响应时间上界的归一化值随参数变化的对比图

和 HAN-2 随着不同的参数增长而变化的示意图(归一化的基准为 JEF). 图 9(a)、图 9(b)、图 9(c)和图 9(d)分别是不同的归一化的响应时间上界随着利用率 $U$、类型 $K$、并行因子 $pr$ 和节点 $|V|$ 增长而变化的评估结果.

从这些实验中我们可以看到,在各种参数设置下,我们提出的新的响应时间上界 DTA 分析精度始终优于 JEF、HAN-1 和 HAN-2.

图 10 展示了根据默认设置,不同的响应时间上界的分析效率随不同参数变化的示意图. 从图中可以看出,我们提出的新的响应时间上界 DTA 要比 HAN-2 花费更长的时间,这是由于 DTA 计算过程的固有困难:在计算最坏响应时间之前,我们需要进行多类型 DAG 任务的转化. 如图 10(a)所示,利用率 $U$ 增加时,每个节点的最坏执行时间增加,因此分配单位节点的计算时间随着利用率 $U$ 增加而提




(a) 分析时间随$U$变化示意图　(b) 分析时间随$K$变化示意图

(c) 分析时间随$pr$变化示意图　(d) 分析时间随$|V|$变化示意图

图 10　不同响应时间分析方法的效率随参数变化的对比图

升. 同样地，如图 10(c)所示，当 $pr$ 增加时，DAG 的边的数量呈线性增长趋势，计算所需要的时间也随之增长. 而类型数 $K$ 的变化不会对 DAG 任务转化阶段的计算时间产生影响，图 10(b)也证实了这一点. 在实际的异构多核处理器中，DAG 任务的并行因子通常很小. 从图 10(c)可以看出，当 $pr$ 小于 0.2 时，多类型 DAG 任务转化和最坏响应时间分析可以在几秒钟内完成，这对于大多数离线设计场景是可以接受的.

综合上述实验，我们基于 DAG 转化算法提出的响应时间分析方法 DTA 跟现有的响应时间上界相比在精度上提升 20%以上.

### 7.2　基于特殊系统模型的对比实验

在本小节，我们通过对比实验对以下响应时间上界进行评估.

(1) DTA. 我们在定理 1 中给出的响应时间上界.

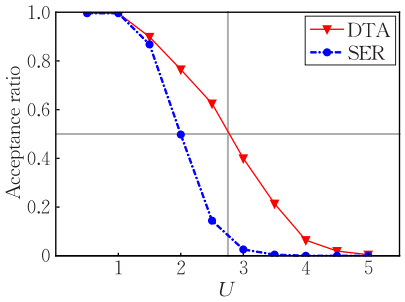
(2) SER. 文献[17]中的响应时间上界，作为归一化的基准.

因为文献[17]研究的平台模型只有两种类型的处理器，其中一种类型是拥有多个内核的主处理器，另一种类型是单核加速设备. 因此我们定义类型 $K$ 为 2，类型为 1 的内核集合 $C_1$ 中内核个数从[2,11]随机选择，类型为 1 的内核集合 $C_2$ 中内核个数为 1. 其他默认参数设置与前面定义得一致.

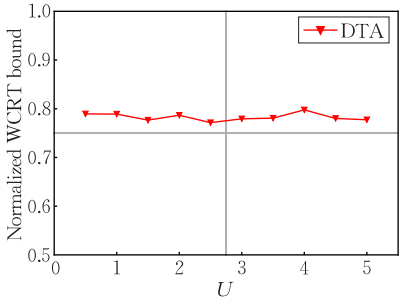
图 11 显示了我们的响应时间上界 DTA 和 SER 的分析精度的评估结果. 图 11(a)显示了指定上界的接受率随着 DAG 任务总利用率 $U$ 增加而变化的实验结果(其他参数与默认参数设置相同). 图 11(b)显示了 DTA 的规范化最坏响应时间(SER 作为基准). 图 11(c)和图 11(d)是两项指标随 $pr$ 变化的结果，图 11(e)和图 11(f)是两项指标随 $|V|$ 变化的结果. 实验表明，我们提出的新的响应时间上界 DTA 分析精度始终优于 SER.

图 12 展示了根据静态参数设置，通过某个变化的参数对我们提出的新的上界 DTA 以及 SER 的计算效率进行评估的结果. 图 12(a)中的每个点代表 DAG 任务总利用率 $U$ 变化时指定响应时间上界的平均执行时间. 实验结果表明，当 $U=1$ 时，DTA 的计算效率几乎与 SER 相同；当 $U>1$ 时，DTA 的计算效率比 SER 低. 图 12(b)和图 12(c)显示 DTA 和 SER 的平均执行时间相差不到一个数量级.

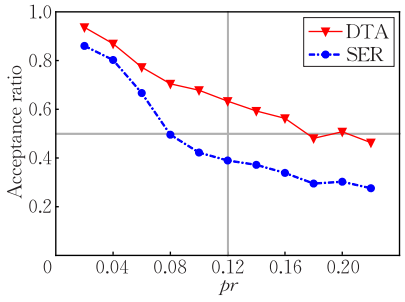
总的实验结果表明，我们提出的基于多类型 DAG 转化算法的响应时间上界 DTA 跟现有的响应时间分析方法相比在精度上有明显提升. 由于 DAG 转化算法的开销，计算时间相比有所增加，但这对于大多数离线设计场景是可以接受的，依然具有实际应用性的价值. 在实际的异构嵌入式实时系统中，由于硬件的飞速发展，处理器运算能力以数量级为单位迅速提升，我们的方法跟现有响应时间上界的计算时间差距会大幅度减少.



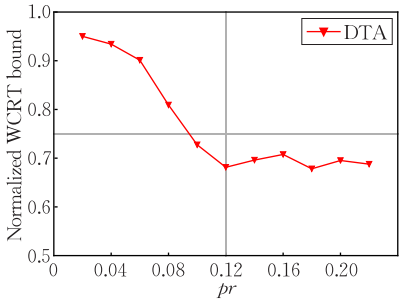

(a) 接收率随$U$变化示意图



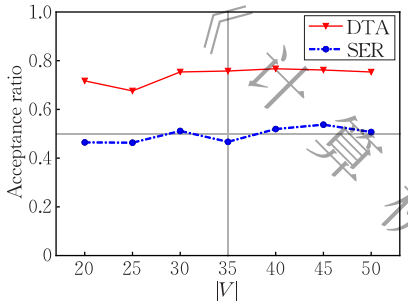

(b) 归一化上界随$U$变化示意图



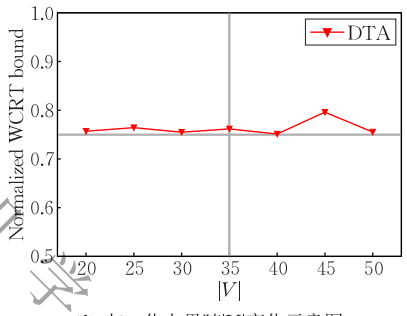

(c) 接收率随$pr$变化示意图




(d) 归一化上界随$pr$变化示意图

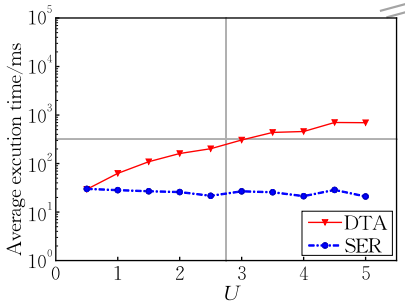



(e) 接收率随$|V|$变化示意图

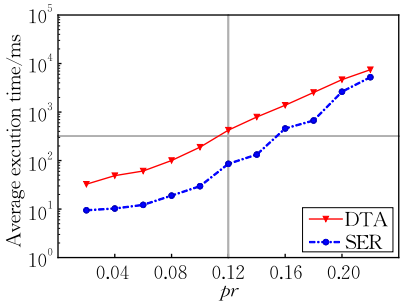



(f) 归一化上界随$|V|$变化示意图

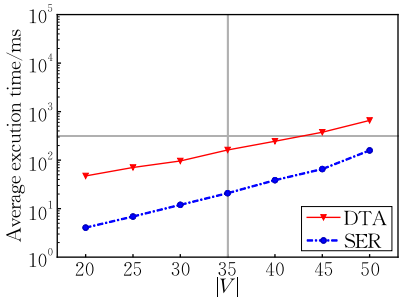
图 11 不同响应时间分析方法的精度随参数变化的对比图




(a) 分析时间随$U$增加的示意图




(b) 分析时间随$pr$增加的示意图




(c) 分析时间随$|V|$增加的变化图

图 12 不同响应时间分析方法的效率随参数变化的对比图

### 7.3　实际 DAG 任务并行类应用的结果验证

为了验证本文提出的多类型 DAG 任务响应时间分析方法的正确性，我们利用 OpenCL 架构在异构平台上进行实际 DAG 任务的验证. OpenCL 的版本为 Intel OpenCL SDK 16.0.

异构平台由以下 5 类处理器构成，分别为

(1) CPU：Intel Broadwell Xeon CPU，14 核；

(2) FPGA-1：Intel Arria 10 GX1150；

(3) FPGA-2：Intel Cyclone 10 GX；

(4) GPU-1：NVIDIA GeForce GTX 580；

(5) GPU-2：NVIDIA GeForce GTX 1080TI.

我们采用了八种 benchmark 作为多类型 DAG 任务的节点，分别为

(1) RSCD(Random Sample Consensus-Data partitioning). 随机样本一致性-数据划分是一种利用随机采样[37]从一组输入数据中估计数学模型参数的迭代方法.

(2) SC(Stream Compaction). 流压缩是一种数据操作原语[38]，它从数组中删除元素，通常用于树遍历、图像处理和数据库.

(3) PAD(Padding). 填充是一种数据操作原语，它在数组的元素之间插入空格，通常用于存储器对准调整和矩阵变换.

(4) BFS(Breadth-First Search). 广度优先搜索是一种图遍历算法. 我们使用基于队列的版本，从单个源节点开始，在每次迭代中，访问当前边界中每个节点的邻节点，并将以前未访问过的邻节点放入队列.

(5) HSTI(Image Histogram-Input Partitioning). 图像直方图-输入分区计算单色图像中像素值的直方图. 直方图统计落入不相交的 bin 的输入中的次数，被广泛用于图像处理和模式识别中.

(6) HSTO(Image Histogram-Output Partitioning). 图像直方图-输出分区的实现基于输出分区，对输出直方图 bin 进行静态数据分区.

(7) CEDD(Canny Edge Detection-Data Partitioning). Canny 边缘检测-数据划分[39]是图像处理中广泛使用的边缘检测算法.

(8) CEDT(Canny Edge Detection-Task Partitioning). 边缘检测-任务划分算法实现了对四个成像阶段的任务划分. 高斯滤波器和 Sobel 滤波器在 GPU 上执行，使它们更有规则.

由于对于同一个 benchmark，不同的输入数据集对应不同的最坏执行时间. 输入数据集规模越大，对应 benchmark 执行完所需的时间越多. 为了增加节点的种类和数量，扩大实验规模，我们参考文献[38]和[40]，为每个 benchmark 设计了 50 组数据集.

我们首先利用文献[36]中的方法生成多类型 DAG 任务的图结构. 接下来我们基于 DAG 图结构，利用 OpenCL 在异构平台上执行 benchmarks. 每个多类型 DAG 任务随机生成的默认参数设置定义如下：

(1) 节点的数量 $|V|$ 从[10,40]的范围内随机选择. 并行因子 $pr$ 在[0.02,0.1]范围内随机选择.

(2) 每个节点的类型由随机选取的 benchmark 确定，一旦确定该节点位置上的 benchmark，就可以得到运行该 benchmark 的处理器类型.

(3) 每个 benchmark 的输入数据集从 50 组已有的数据集中随机选取.

异构处理器跟所要执行的 benchmarks 的对应关系如表 1 所示.

**表 1　异构处理器和 Benchmark 的对应关系**

| 处理器类型 | Benchmark |
|---|---|
| CPU | RSCD, BFS |
| FPGA-1 | HSTI, HSTO |
| FPGA-2 | SC |
| GPU-1 | PAD |
| GPU-2 | CEDD, CEDT |

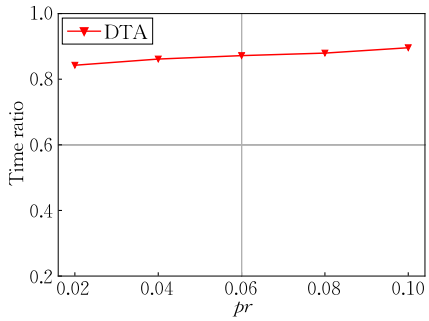
我们用 $t$ 表示实际 DAG 任务运行所耗费的时间，用 $\bar{t}$ 表示按照我们提出的最坏响应时间分析方法 DTA 得到的理论时间，两者的比值 $t/\bar{t}$ 我们用 Time ratio 来表示.

图 13 显示了根据静态参数设置，通过某个变化的参数对我们提出的新的最坏响应时间上界 DTA 在实际应用中的正确性进行评估的结果. 图 13(a)、图 13(b)分别是指 Time ratio 随 DAG 任务并行因子 $pr$ 和总节点个数 $|V|$ 增长而变化的结果.

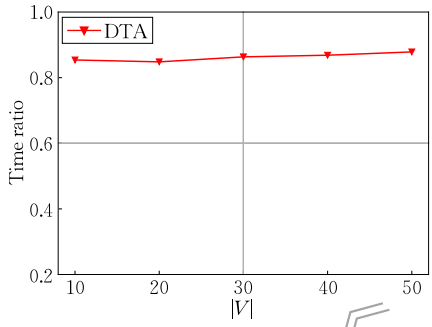
从图 13 中的数据可以看出，对于实际的多类型 DAG 任务，我们提出的最坏响应时间分析方法得到的理论时间总是大于实际的运行时间，因此我们提出的新的响应时间上界 DTA 是安全可靠的.

## 8　结　论

本文基于异构平台提出了一种新的多类型 DAG 任务转化算法和一个新的响应时间分析方法 DTA. 多类型 DAG 中的每个节点只允许在特定类型的处




(a) 时间比率随$pr$增加的示意图




(b) 时间比率随$|V|$增加的变化图

图 13 实际 DAG 任务响应时间实际值和理论值的比率随参数变化的对比图

理器内核上执行.已知现有的基于 work-conserving 调度策略的多类型 DAG 任务的响应时间分析策略是悲观的,因为这些分析策略高估了节点受到其他路径上同类型节点的阻塞时间.

为此,我们首先提出了一种新的多类型 DAG 任务转化算法,该算法通过将节点拆分成单位节点,并在不破坏原有依赖关系的基础上,按照我们提出的单位节点分配策略在单位节点之间增加新的边,构成一个新的多类型 DAG 任务,以缩小可能与每个单位节点并行执行的不同路径上同类型单位节点的范围.接下来,我们在 DAG 任务转化算法的基础上提出一种新的响应时间分析方法,该方法能够得到更加精确的单位节点受到阻塞时间,进而提高计算响应时间上界的精度.实验结果表明,我们的方法得到的响应时间上界比现有响应时间上界精确度提高 20%以上.

下一步,我们将把现有的响应时间分析方法扩展到对多个多类型 DAG 任务进行响应时间分析.另一个方向是研究更有效的多类型 DAG 任务转化算法,可以在更短的时间内实现 DAG 任务的转化,并且结果近似于我们提出的响应时间上界.

**致 谢** 感谢各位评审老师给出的宝贵意见!

## 参 考 文 献

[1] Langer T, Osinski L, Mottok J. A survey of parallel hard-real time scheduling on task models and scheduling approaches//Proceedings of the 30th International Conference on Architecture of Computing Systems. Vienna, Austria, 2017: 1-8

[2] Corbett P, Feitelson D, Fineberg S, et al. Overview of the MPI-IO parallel I/O interface//Proceedings of the IPPS'95 Workshop on Input/Output in Parallel and Distributed Systems. Boston, USA, 1995: 1-15

[3] Saifullah A, Agrawal K, Lu C, et al. Multi-core real-time scheduling for generalized parallel task models//Proceedings of the 32nd IEEE Real-Time Systems Symposium. Vienna, Austria, 2011: 404-435

[4] Fonseca J C, Nélis V, Raravi G, et al. A multi-DAG model for real-time parallel applications with conditional execution //Proceedings of the 30th Annual ACM Symposium on Applied Computing. Salamanca, Spain, 2015: 1925-1932

[5] Jiang X, Guan N, Long X, et al. Semi-federated scheduling of parallel real-time tasks on multiprocessors//Proceedings of the 38th IEEE Real-Time Systems Symposium. Paris, France, 2017: 80-91

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

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

[8] 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. Krakow, Poland, 2016: 1-10

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

[10] Bonifaci V, Marchetti-Spaccamela A, Stiller S, et al. Feasibility analysis in the sporadic DAG task model//Proceedings of the 25th Euromicro Conference on Real-Time Systems. Paris, France, 2013: 225-233

[11] Li J, Chen J J, Agrawal K, et al. Analysis of federated and global scheduling for parallel real-time tasks//Proceedings of the 26th Euromicro Conference on Real-Time Systems. Madrid, Spain, 2014: 85-96

[12] Serrano M A, Melani A, Bertogna M, et al. Response-time analysis of DAG tasks under fixed priority scheduling with limited preemptions//Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition. Dresden, Germany, 2016: 1066-1071

[13] Lee J, Chwa H S, Lee J, et al. Thread-level priority assignment in global multiprocessor scheduling for DAG tasks. Journal of Systems and Software, 2016, 113: 246-256

[14] NVIDIA Tegra K1: A new era in mobile computing. Nvidia, Corp, White Paper, 2014

[15] Leibson S, Mehta N. Xilinx UltraScale: The next-generation architecture for your next-generation architecture. Xilinx, White Paper WP435, 2013: 143

[16] Jaffe J M. Bounds on the scheduling of typed task systems. SIAM Journal on Computing, 1980, 9(3): 541-551

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

[18] Han Mei-Ling, Guan N, Sun Jing-Hao, 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

[19] Yang K, Yang M, Anderson J H. Reducing response-time bounds for DAG-based task systems on heterogeneous multicore platforms//Proceedings of the 24th International Conference on Real-Time Networks and Systems. Brest, France, 2016: 349-358

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

[21] Graham R L. Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics, 1969, 17(2): 416-429

[22] Li J, Ferry D, Ahuja S, et al. Mixed-criticality federated scheduling for parallel real-time tasks. Real-Time Systems, 2017, 53(5): 760-811

[23] Baruah S. The federated scheduling of systems of mixed-criticality sporadic DAG tasks//Proceedings of the 37th IEEE Real-Time Systems Symposium. Porto, Portugal, 2016: 227-236

[24] Chen, Jian-Jia. Federated scheduling admits no constant speedup factors for constrained-deadline DAG task systems. Real-Time Systems, 2016, 52(6): 833-838

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

[26] 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. Versaille, France, 2014: 13

[27] 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

[28] Jia Z H, Wen T T, Leung J Y T, et al. Effective heuristics for makespan minimization in parallel batch machines with non-identical capacities and job release times. Journal of Industrial & Management Optimization, 2017, 13(2): 977-993

[29] Li S. Parallel batch scheduling with inclusive processing set restrictions and nonidentical capacities to minimize makespan. European Journal of Operational Research, 2017, 260(1): 12-20

[30] Karhi S, Shabtay D. On the optimality of the TLS algorithm for solving the online-list scheduling problem with two job types on a set of multipurpose machines. Journal of Combinatorial Optimization, 2013, 26(1): 198-222

[31] Shabtay D, Karhi S. Online scheduling of two job types on a set of multipurpose machines with unit processing times. Computers & Operations Research, 2012, 39(2): 405-412

[32] Epstein L, Levin A. Scheduling with processing set restrictions: PTAS results for several variants. International Journal of Production Economics, 2011, 133(2): 586-595

[33] Leung Y T, Li C L. Scheduling with processing set restrictions: A survey. International Journal of Production Economics, 2008, 116(2): 251-262

[34] Leung Y T, Li C L. Scheduling with processing set restrictions: A literature update. International Journal of Production Economics, 2016, 100(175): 1-11

[35] Bini E, Buttazzo G C. Measuring the performance of schedulability tests. Real-Time Systems, 2005, 30(1-2): 129-154

[36] Cordeiro D, Mounié G, Pérarnau S, et al. Random graph generation for scheduling simulations//Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques. Torremolinos Malaga, Spain, 2010: 60

[37] Fischler M A, Bolles R C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 1981, 24(6): 381-395

[38] Gómez-Luna J, Hajj I E, Chang L W, et al. Chai: Collaborative heterogeneous applications for integrated-architectures//Proceedings of the 18th IEEE International Symposium on Performance Analysis of Systems and Software. Santa Rosa, USA, 2017: 43-54

[39] Canny J. A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986, PAMI-8(6): 679-698

[40] Jiang J, Wang Z, Liu X, Gómez-Luna J, et al. Boyi: A systematic framework for automatically deciding the right execution model of OpenCL applications on FPGAs//Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Seaside CA, USA, 2020: 43-52

**CHANG Shuang-Shuang**, Ph. D. candidate. Her main research interests include embedded real-time system and parallel computing.

**ZHAO Xu-Feng**, M. S. candidate. His main research interests include cyber-physical systems and embedded real-time systems.

**LIU Zhen-Yu**, M. S. candidate. His main research interests include cyber-physical systems and embedded real-time systems.

**DENG Qing-Xu**, Ph. D., professor. His main research interests include cyber-physical systems and embedded real-time systems.

**Background**

Heterogeneous multi-cores and parallel architectures have recently gained much attention owing to utilize the strength of different architectures for offering higher performance. In general, heterogeneous hardware architecture consists of equipment that is asymmetric in performance and functionality which integrates low power general purpose multi-cores (known as the host) with specialized coprocessors (e. g., Cell/BE SPUs) or data-parallel accelerators (e. g., GPUs), such as NVIDIA Tegra X1. In this paper, we consider real-time scheduling of the typed DAG task on heterogeneous multi-cores, where each vertex is explicitly bound to a specific type of cores for execution. Binding code fragments of a program to a specific type of kernel is a common operation in heterogeneous multicore scheduling, which can be easily in the mainstream parallel programming framework and operating system implementation. Traditional researches use the work-conserving scheduling strategy to schedule such a typed DAG task. Jeffrey et al proposed the first worst-case response time bound for the general typed DAG task model. However, Jeffrey's response time bound is very pessimistic. Serrano et al proposed the response time bound for a specific typed DAG task model with two typed cores that has certain limitations. Han et al developed two response time bounds in which the first bound dominated Jeffrey's bound in analysis precision and another bound significantly improved the analysis precision by exploring more detailed task graph structure information. Even Han's response time bounds are still very pessimistic. This paper aims to get a more accurate the worst-case response time upper bound for typed DAG tasks. To solve the problems in the early work, we propose a typed DAG task transformation algorithm, which can obtain a new DAG task by adding new edges on the basis of retaining the original dependencies between vertices. Based on this transformation algorithm, we propose a new worst-case response time analysis method to verify the schedulability of typed DAG tasks that support heterogeneous computing. A comparison of randomly generated typed DAG tasks shows that the accuracy of our new worst-case response time is more than 20% higher than the existing bounds. The research in this paper has a positive significance to improve the ability to compute the worst-case response time of the typed DAG task in heterogeneous multicores platform.

The work is partially supported by the National Key Research and Development Program (No. 2018YFB1702003), the National Natural Science Foundation of China (Nos. 61602104, 61972076, U1908212, 61871107), and the Talent Project of Revitalizing Liaoning (Nos. XLYC1902017).