引用格式：陈雨濛, 刘松林, 何宗苗, 等. 基于边覆盖队列的异构多处理器系统调度算法 [J]. 电子科技大学学报, 2025, 54(5): 723-732.
CHEN Y M, LIU S L, HE Z M, et al. Heterogeneous multiprocessor system scheduling algorithm based on edge cover queue[J]. Journal of University of Electronic Science and Technology of China, 2025, 54(5): 723-732.

基于边覆盖队列的异构多处理器系统调度算法

陈雨濛[1]，刘松林[1]，何宗苗[1,2]，陈彦君[1]，凌 翔[1*]

（1. 电子科技大学 通信抗干扰全国重点实验室，成都 611731；2. 成都工业学院 网络与通信工程学院，成都 611730）

摘要：异构多处理器系统是具有不同计算能力和存储能力并相互连接的一组处理器。在异构多处理器系统中，优秀的任务调度算法能够缩短任务完成时间，提升系统利用率和并行度。针对异构多处理器系统，基于有向无环图的边覆盖理论提出了一种新的任务调度算法——启发式边覆盖队列调度算法（HECSA）。该算法利用改进的启发式公式，在保证拓扑正确的前提下，生成有向无环图的边覆盖队列。再利用计算复杂度低的启发式方法将边覆盖队列按顺序分配到异构多处理器上执行。常见数字信号处理任务和科学工作流任务的仿真实验结果表明，提出的 HECSA 在较低的复杂度下能够得到更好的调度结果。

关键词：异构多处理器系统；有向无环图；边覆盖队列；快速傅里叶变换；高斯消元法；科学工作流
中图分类号：TP301.6 **文献标志码：**A DOI: 10.12178/1001-0548.2024343

Heterogeneous multiprocessor system scheduling algorithm based on edge cover queue

CHEN Yumeng[1], LIU Songlin[1], HE Zongmiao[1,2], CHEN Yanjun[1], and LING Xiang[1*]

(1. National Key Laboratory of Wireless Communications, University of Electronic Science and Technology of China, Chengdu 611731, China;
2. School of Networks & Communication Engineering, Chengdu Technological University, Chengdu 611730, China)

Abstract: Heterogeneous multiprocessor system refers to a group of interconnected processors with different computing and storage capabilities. Due to the diversity of task computing requirements and differences in processor architecture, heterogeneous multiprocessor systems are widely present in various computing scenarios. In heterogeneous multiprocessor systems, excellent task scheduling algorithms can shorten task completion time and improve system parallelism and utilization. This article proposes a new task scheduling algorithm, the heuristic edge cover queue scheduling algorithm (HECSA) based on the edge cover theory of directed acyclic graphs (DAG) for heterogeneous multiprocessor systems. HECSA first utilizes an improved heuristic method to generate edge cover queue for DAG while ensuring topological correctness. Then, the heuristic method with low computational complexity is applied to sequentially allocate edge cover queue to heterogeneous multiprocessor system for execution. The simulation results of common digital signal processing tasks and scientific workflow tasks show that the HECSA can achieve better scheduling results under lower computational complexity.

Key words: heterogeneous multiprocessor system; directed acyclic graphs; edge cover queue; fast Fourier transform; Gaussian elimination; scientific workflow

随着通信集成电路的不断发展，异构计算系统广泛存在于各种通信计算场景[1]，在实际应用中通常建模为有向无环图（directed acyclic graphs, DAG）来调度[2]。高效的 DAG 任务调度方案能够提高异构计算系统性能和用户体验质量[3]。由于异构计算系统的异构性、任务之间的拓扑优先级约束以及 DAG 的 NP-H 特性，获得高效的调度十分困难[4]。现有的调度算法分为表调度算法、聚类算法、任务

收稿日期：2024−12−13
基金项目：国家重点研发计划项目子课题（22ZD201262003-1）
作者简介：陈雨濛，博士，主要从事嵌入式系统方面的研究。
*通信作者 E-mail: xiangling@uestc.edu.cn

复制算法和进化算法 4 大类[5]。表调度算法具有低的运算复杂度和时间复杂度，但是在很多场景下调度结果不够理想，容易陷入局部最优解。但是其具有极低的运算时间，因此适用于资源受限的嵌入式多处理器系统和系统架构中调度核可用资源较少的系统。进化算法是全局优化的算法，在足够的运算量下能得到优秀的调度结果，但它的时间复杂度非常高，适用于云计算等计算资源丰富的场景。聚类算法通过任务合并拆分，将大规模的任务更好地调整为处理器适配的任务，但此算法仍然依赖于表调度算法的启发式思想。任务复制算法主要是通过重复执行某些关键任务来减少子任务之间的通信量，从而降低通信资源消耗，一般是基于某种特定策略将关键父节点数据复制到该处理器上，使得多节点之间能够并行执行，但这会占用更多的处理器资源，带来更高的处理器能耗，所以任务复制类调度算法在实际运用中较少。

表调度算法分为两个阶段。首先，根据任务优先级权重对任务进行排序得到任务调度队列，再采用某种固定方式将队列中的任务依次放置到处理器上[6]。早期的调度算法大多针对同构处理系统，如 EST 和 EFT，都是最简单的完成时间贪心算法。为了适应异构环境，研究人员提出了两种经典的启发式方法：异构最早完成时间算法（heterogeneous earliest finish time, HEFT）和关键路径算法（critical path on a processor, CPOP）[2]。预测类表调度算法通过估计当下任务分配对其后续任务的影响进行调度。在著名的预测类表调度算法 PEFT（parameter-efficient fine-tuning）中构建了一个乐观成本表（optimistic cost table, OCT），列出了任务和处理器的每个组合从其子节点到出口节点的最短路径[6]。PEFT 算法利用每个任务平均 OCT 值，从高到低对任务进行拓扑排序；最后，引入插入策略并将任务分配给（EFT+OCT）最短的处理器上。lookahead 算法（LO）是一种比较特殊的表调度算法，每个任务在调度时分配到使其所有子任务完成时间最短的处理器上，在中等规模的 DAG 调度中具有良好的效果，但是也是运算复杂度最高的一类表调度算法[7]。后续改进了 PEFT 算法，如 PPTS 和 IPPTS 算法等。但是针对不同的任务类型，不同算法之间的性能差异巨大。对于快速傅里叶变换，最好的算法仍然是 HEFT 算法，而对于高斯消元法，LO 算法依旧表现出卓越的性能。

进化类算法是全局优化的算法，能够在可接受的时间内为复杂问题提供满意的解决方案。对于 DAG 调度问题（DAG-SP），各种进化算法，如遗传算法（GA）、蚁群优化（ACO）、差分进化（DE）都被尝试着在任务调度问题上使用[8]。进化算法除了直接给出任务-处理器调度方案，更进一步地，针对任务调度队列优先级的进化算法也有相关研究工作。MPEQGA 算法利用进化算法引导任务调度队列优先级的迭代来求解 DAG-SP[9]。但是，进化类算法具有很长的运算时间，其在嵌入式场景、资源受限场景下无法使用。与表调度算法相比，进化类算法需要花费数千倍的时间来获得一个调度结果。

在图论中，边覆盖是一个非常重要的理论，在通信系统和网络优化中发挥着关键作用，有许多重要的最小边覆盖和最小权重边覆盖的算法在网络拓扑优化中发挥着重要的作用。本文跳出拓扑可行点队列的传统思路，提出利用边覆盖队列完成调度。边覆盖队列让一些任务节点在启发式算法的指导下以非贪心的方式调度到某个处理器上，为一些关键节点作出更好的执行准备。边覆盖队列使得调度算法能够在低的运算复杂度和空间复杂度的前提下，探索更大的解空间，是一种启发式点贪心与启发式边贪心的混合调度策略。

1 异构系统任务调度相关理论

1.1 系统模型

DAG-SP 指 DAG 调度问题，具体为具有m个处理器的异构系统中单个应用程序的静态调度问题[10]。DAG 任务调度可分为静态和动态方法。动态调度适用于任务参数未知的情况，需要系统调度器在运行时做出调度决策，但这样会带来大量的额外开销。如用户随时向共享计算资源提交任务负载的系统，任务负载只有在运行中被提交后才知道，因此需要动态算法。上述特性决定了动态算法无法基于整体任务负载情况进行优化。相比之下，静态方法可以通过考虑整体任务负载情况来完成调度，以达到最大化的并行度和系统效率。静态调度的调度方案是在任务执行开始之前完成的，在运行时不会引入任何开销。因此，在编译时已知系统和整体任务负载情况的情况下，静态调度方法在运行时没有开销，更适合嵌入式等资源较少的系统，能降低任务完成时间并提高系统响应能力[6]。本文提出边覆盖队列调度算法的目标就是在m个处理器的异构系统中最小化任务的完成时间。

DAG-SP 中，应用程序被建模为有向无环图，$\mathrm{DAG}=(V,E)$[11]。其中每个节点$v_i \in V$表示必须在同一处理器上执行的独立应用任务。E是任务之间的一组边。每个$e_{i,j} \in E$表示任务拓扑约束。DAG 信息由矩阵$\boldsymbol{W}$和$\boldsymbol{C}$补充。$\boldsymbol{W}$是规模为$|V|\times m$的计算成本矩阵，其中$|V|$是任务数，m是系统中处理器的数量。每个矩阵表示处理器上任务的计算成本，矩阵$\boldsymbol{C}$称为通信成本矩阵，由$c_{i,j}$组成。$c_{i,j}$表示对应有向边$e_{i,j} \in E$的通信成本。当任务和被分配给同一处理器时，实际通信成本可认为是零，因为与处理器间的通信成本相比，它可忽略。在大多数模型中，处理器以完全连接的拓扑结构进行连接。此外，任何任务的执行都被认为是非抢占式的。此模型是这个调度问题的标准模型。

$\mathrm{pred}(v_i)$是任务v_i的直接拓扑前序任务集合。$\mathrm{succ}(v_i)$是任务v_i的直接拓扑后序任务集合。v_{entry}是 DAG 的入口任务。如果一个 DAG 有多个v_{entry}，则会在图中添加一个对实际计算没有影响的伪入口节点[12]。v_{exit}是一个没有直接后续节点的任务。类似地，如果一个 DAG 有多个v_{exit}，那么也会向图中添加一个伪出口节点。makespan是v_{exit}的最大完成时间，因此makespan是由 DAG 表示的应用程序的调度长度。DAG 的关键路径 (CP) 是图中从入口节点到出口节点的最长路径。因此，makespan的下限是最小关键路径长度$\mathrm{CP}_{\min}$，可以通过关键路径中每个节点的最小计算成本计算。CCR 是通信与计算成本的比率，旨在衡量通信延迟对计算性能的影响。对于应用程序，高 CCR 表示通信密集型，低 CCR 表示计算密集型。CCR 为[12]：

$$\mathrm{CCR}=\frac{\sum_{i,j} c_{i,j}/|E|}{\sum_{i} \overline{w}_i/|V|} \tag{1}$$

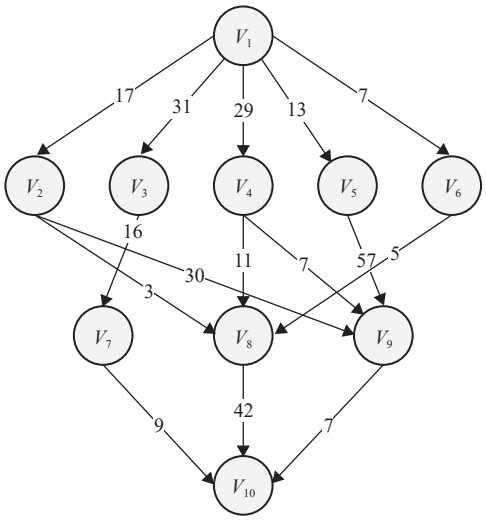
式中，$\overline{w}_i$为平均计算成本，为：

$$\overline{w}_i=\sum_{j\in m} w_{i,j}/m \tag{2}$$

处理器上计算成本的heterogenetty参数β是指处理器的异构度参数，高β值意味着处理器之间存在更高的异构度。因为每个处理器上的每个任务的计算成本$w_{i,j}$是随机设置的[6]：

$$\overline{w}_i(1-\beta/2) \leqslant w_{i,j} \leqslant \overline{w}_i(1+\beta/2) \tag{3}$$

如图 1 所示的 DAG 任务需要被分配到 3 个异构处理器系统上，每个子任务在不同处理器上的计算时间均不同[6]。调度算法分为确定调度顺序和调度方式两步。首先确定一个不冲突的调度队列，所有点的拓扑后续任务都在其之后。随后再将任务按队列顺序分配到处理器上。在调度过程中，不同的算法会给出不同的调度优先级计算方式，得到不同的点调度队列。如 HEFT 中任务节点根据从节点到 DAG 底部的最长路径长度值rank_u排序，PEFT 中按照平均 OCT 值$\mathrm{rank}_{\mathrm{OCT}}$对任务节点进行排序。



a. 10节点有向无环图模型

Task	P1	P2	P3
V_1	22	21	36
V_2	22	18	18
V_3	32	27	43
V_4	7	10	4
V_5	29	27	35
V_6	26	17	24
V_7	14	25	30
V_8	29	23	36
V_9	15	21	8
V_{10}	13	16	33

b. 异构计算成本矩阵

图 1　典型 DAG-SP 任务模型[6]

边覆盖是 DAG 图的一个边子集，该边子集能够覆盖 DAG 的点全集。边覆盖队列是指一个 DAG 图边覆盖的拓扑可行队列。一个包含了所有 DAG 边的集合一定是一个边覆盖集合，称其为全覆盖队

列。以图 1 为例，其中一个边覆盖队列为[$e_{1,2}$,$e_{1,3}$,$e_{1,4}$,$e_{1,5}$,$e_{1,6}$,$e_{3,7}$,$e_{4,8}$,$e_{5,9}$,$e_{8,10}$]。

1.2 典型算法

1.2.1 EFT 算法和 HEFT 算法

EFT（earliest finish time）算法包括任务优先级确定阶段和处理器选择阶段。算法通过计算 rank_u值来确定任务的优先级，rank_u值越大的任务优先级越高。随后，在任务分配阶段将其分配到 EFT 处理器上。在 EFT 算法基础上考虑一种插入机制来完成调度。HEFT 算法性能极为可靠，其调度长度与其他调度算法相当，但时间复杂度更低。在任务优先级确定阶段，HEFT 与 EFT 相同。但在处理器选择阶段，HEFT 算法使用插入策略，尝试在处理器上两个已调度任务之间的最早空闲时间插入任务，如果该空闲时间足够则容纳该任务；如果不行，该任务被分配给最早完成时间的处理器。

1.2.2 LO 算法

lookahead 算法（LO）基于 HEFT 算法，但在处理器选择策略上进行了改进。为当前待调度任务 v_i选择处理器时，该算法遍历所有可用处理器并计算v_i所有子任务的 EFT。调度器为任务v_i选择的处理器是所有子任务的最大 EFT 取值最小的处理器[7]。该过程依次用于所有待调度的任务。同时通过增加分析的层级数来提高算法性能。然而即使只考虑一级子任务，LO 算法也具有很高的时间复杂度。LO 算法在一些具有规则结构的 DAG 中表现良好，如中等规模 DAG 和高斯消元法。

1.2.3 PEFT 算法

文献 [6] 提出了一个基于乐观成本表（OCT）的 PEFT 算法。OCT 具有预测功能，其每个元素 $\text{OCT}(v_i,p_j)$表示假设选择处理器p_j执行任务时其子任务v_i到出口节点的最短路径的最大值。在任务优先级确定阶段，PEFT 算法计算每个任务的平均 OCT，完成优先级排序。在处理器选择阶段，算法结合 OCT 和 EFT 完成启发式调度。通过这种方式，PEFT 算法在处理器选择时进行预测。尽管当前任务不一定选择到最小 EFT 的处理器，但能选择使接下来的任务完成时间更短的处理器。PEFT 在随机任务测试中表现出了优异的性能，是一种被广泛利用的调度算法。

1.2.4 PPTS 和 IPPTS 算法

PPTS 也是一种用于异构计算的表调度算法，称为预测优先级任务调度算法[13]。此算法通过在任务优先级阶段和处理器选择阶段同时引入预测功能来最小化调度长度。现有的列表调度算法，如 PEFT 和 LO，仅在处理器选择阶段引入了预测功能。PPTS 不仅在处理器选择阶段，而且在任务优先级阶段都能体现预测思想且不会增加时间复杂度。IPPTS 算法是 PPTS 的升级版，具有更好的性能，通过更好的启发式策略实现子任务处理器分配。利用 PCM 矩阵在任务优先级阶段和处理器选择阶段同时引入预测功能来最小化调度长度的思路没变[14]。

1.3 算法评价指标

调度算法的评估指标包括完工时间（makespan）、效率 (efficiency) 和调度长度比（scheduling length ratio, SLR)。SLR是一个考虑算法下界的参数。它比makespan能更好地反映算法的改进。efficiency可以反映该算法对异构多处理器系统并行性的提高。

如式 (4) 所示，makespan是调度长度最直观的参数。调度长度越短，任务在异构多处理器系统上执行的速度就越快[6]。

$$\text{makespan} = \max \text{FT}(v_{\text{exit}}) \tag{4}$$

efficiency定义为$\text{efficiency} = \text{speedup}/m$。speedup是顺序执行时间与算法调度长度的比值。顺序执行时间的计算方法是将所有任务分配给单个处理器，使任务图的总计算成本最小，为：

$$\text{speedup} = \frac{\min\left[\sum\limits_{v_i \in V}^{m_j \in m} w(i,j)\right]}{\text{makespan}} \tag{5}$$

如果想使用一个指标来比较具有不同拓扑结构的 DAG，最常用的度量是归一化调度长度NSL，也称为归一化调度长度比SLR。SLR是调度长度 makespan 与任务不可能调度长度下界的比率。对于给定的 DAG，两者都表示归一化到下限的完工时间。SLR 定义为：

$$\text{SLR} = \frac{\text{makespan}}{\sum\limits_{p_j \in m}^{v_i \in \text{CP}_{\min}} \min[w(i,j)]} \tag{6}$$

式中，SLR 中的分母是关键路径（$\text{CP}_{\min}$）上任务的最小计算成本，没有比其更小的完成时间。因此，SLR 最低的算法是最好的算法。

2 边覆盖队列调度算法

边覆盖队列调度算法旨在不增加调度复杂度的

情况下探索更大的解空间，使得一些任务能有更高的优先级选择处理器。表调度算法中，综合参考任务运算量和其子任务来确定节点之间的优先级。一个任务越早获得调度的机会，则可以选择充足的处理器资源，如果能够放置在合适的处理器上将有可能直接降低最终调度长度，或间接影响后续子任务的执行来降低最终调度长度。边覆盖队列通过边调度的方式，让一些任务节点通过非贪心的方式调度到了某个处理器上，为一些关键节点作出更好的执行准备，提升了调度性能。这是点调度队列无法具有的特性。

2.1　边覆盖队列生成

首先根据异构计算矩阵$\boldsymbol{W}$和通信矩阵$\boldsymbol{C}$计算出该 DAG 任务的消极成本矩阵（negative cost matrix, NCM）。NCM 中的元素$\mathrm{NCM}(v_i,p_k)$代表子任务v_i分配到处理器p_k上时，后续关键路径的最坏执行时间。具体为：

$$\mathrm{NCM}(v_i,p_k)=\max_{v_j\in\mathrm{succ}(v_i)}\left[\max_{p_w\in m}\mathrm{NCM}(v_j,p_w)+\bar{c}_{i,j}\right]+w(v_i,p_k) \tag{7}$$

式中，$w(v_i,p_k)$代表子任务节点v_i在处理器p_k上的计算成本，对于出口节点v_{exit}，有$\mathrm{NCM}(v_{\mathrm{exit}},p_k)=w(v_{\mathrm{exit}},p_k)$；$\bar{c}_{i,j}$代表子任务$v_i$和$v_j$的实际通信成本，当$v_i$和$v_j$分配到同一个处理器上时，$\bar{c}_{i,N}=0$。消极成本矩阵一定程度上代表了某任务被分配到某处理器上后续关键路径的最坏执行情况，因此能够将关键路径上的节点放在拓扑更靠前的位置。

$$\mathrm{rank}_{\mathrm{NCM}}(v_i)=\frac{\sum_{k=1}^{k=m}\mathrm{NCM}(v_i,p_k)}{m} \tag{8}$$

根据$\mathrm{rank}_{\mathrm{NCM}}$从大到小完成拓扑点队列排序，生成$\mathrm{Node}_{\mathrm{NCM}}$。再将$\mathrm{Node}_{\mathrm{NCM}}$转换为边覆盖队列。从第一个节点$v_x$开始，其相邻的下一个节点$v_y$，如果$e_{x,y}\in E$，则将$e_{x,y}$放入$\mathrm{Edge}_{\mathrm{NCM}}$，如果$e_{x,y}\notin E$，则将$v_x$的任意一个父节点$v_f$之间的边$e_{f_1x}$放入$\mathrm{Edge}_{\mathrm{NCM}}$。重复上述步骤直到$\mathrm{Node}_{\mathrm{NCM}}$为空。

定理 1　对于任一拓扑可行点队列$\mathrm{Node}_{\mathrm{NCM}}$，至少有一个边覆盖队列$\mathrm{Edge}_{\mathrm{NCM}}$使得调度结果保持不变。

证明：假设任一拓扑可行点队列为$\mathrm{Node}_{\mathrm{NCM}}=[v_1,v_2,\cdots,v_i]$，根据前文提到的为 DAG 任务图添加一个虚拟入节点v_{entry}和虚拟出节点v_{exit}，拓扑可行点队列变为$\mathrm{Node}^*_{\mathrm{NOM}}=\left[v_{\mathrm{entry}},v_1,\cdots,v_i,v_{\mathrm{exit}}\right]$，由于虚拟节点没有计算量和通信量，不会对原有队列产生影响。从v_1开始，由于其前序节点$\mathrm{pred}(v_1)$已经执行过，则将边$e_{\mathrm{exit},1}$放入$\mathrm{Edge}_{\mathrm{NCM}}$。对于接下来的任意一个节点$v_x$，由于点队列为拓扑可行的，意味着其所有拓扑前序节点$\mathrm{pred}(v_x)$已经执行过，因此任选一个拓扑前序节点即v_f与v_x组成边。在这个边覆盖队列中，由于每一条边的两个节点中的前序节点已经执行，则调度时和点调度是一致的，因此这个边覆盖队列与拓扑可行点队列等价。这样生成的边覆盖队列仍然和点队列长度相当，不会引入额外的长度。

算法 1 边覆盖队列生成方式

输入：异构计算矩阵$\boldsymbol{W}$，通信矩阵$\boldsymbol{C}$

输出：边覆盖队列$\mathrm{Edge}_{\mathrm{NCM}}$

为出节点v_{exit}赋值$\mathrm{NCM}(v_{\mathrm{exit}},p_k)=w(v_{\mathrm{exit}},p_k)$

根据NCM计算公式依次计算所有节点的$\mathrm{NCM}(v_i,p_k)$

形成NCM

计算每个节点的拓扑排序值$\mathrm{rank}_{\mathrm{NCM}}(v_i)$

根据拓扑排序值$\mathrm{rank}_{\mathrm{NCM}}(v_i)$，从大到小排序生成点队列$\mathrm{Node}_{\mathrm{NCM}}$

from $\mathrm{Node}_{\mathrm{NCM}}$第一个节点$v_x$ do

　if v_x与v_y存在有向边 do

　　将$e_{x,y}$放入$\mathrm{Edge}_{\mathrm{NCM}}$

　　将v_x与v_y从$\mathrm{Node}_{\mathrm{NCM}}$删去

　else if v_x与v_y不存在有向边 do

　　将e_{f_1x}放入$\mathrm{Edge}_{\mathrm{NCM}}$

　　将v_x从$\mathrm{Node}_{\mathrm{NCM}}$删去

　end if

　重复上述步骤直到$\mathrm{Node}_{\mathrm{NCM}}$为空

Exit Loop

2.2　边覆盖队列分配

接下来依次为边覆盖队列$\mathrm{Edge}_{\mathrm{NCM}}$分配执行的处理器。如果待分配的边$e_{i,j}$中，$v_i$已经被分配过了，则在插入策略的基础上计算$v_j$在各个处理器上的最快完成时间$\mathrm{EFT}(v_i,p_k)$，将$v_j$分配到具有最小EFT的处理器上。如果$v_i$尚未分配，则在插入策略的基础上将$v_i$分配到使得$v_j$的EFT最小的处理器上，随后在插入策略的基础上继续将v_j分配到EFT最小的处理器上。重复上述步骤直到$\mathrm{Edge}_{\mathrm{NCM}}$为空。

算法 2 边覆盖队列分配方式

输入：边覆盖队列Edge_{NCM}

输出：DAG 任务调度方式

from Edge_{NCM}第一个边$e_{i,j}$ do

if v_i已被调度 do

将v_j分配到具有最小EFT的处理器上

将$e_{i,j}$从Edge_{NCM}删去

else if v_i尚未调度 do

基于插入策略计算节点v_i分配到各个处理器上时的EFT。根据v_i分配到各个处理器上的情况，基于插入策略计算v_j的最小EFT

将v_i分配到使得v_j的EFT最小的处理器上，同时将v_j分配到最小EFT的处理器上

将$e_{i,j}$从Edge_{NCM}删去

end if

重复上述步骤直到Edge_{NCM}为空

Exit Loop

定理 2 与点队列相比，边覆盖队列能够在几乎不增加运算复杂度的情况下，探索更大的解空间。

证明：边覆盖队列在调度中是点贪心与局部贪心的混合。有向边$e_{i,j}$中，如果v_i没有被调度，则将v_i调度到使得v_j在当前启发式参数最小的处理器上。有向边$e_{i,j}$中，如果v_i已经被调度，则将v_j调度到使得启发式参数最小的处理器上。在点调度队列中，v_i不可能越过当前的启发式公式，调度到为v_j更有利但使得v_i次优的处理器上。因此边覆盖队列带来了局部贪心。HECSA 中，通过算法 1 生成边覆盖队列的时间复杂度为$O(|V|^2)$，在调度边覆盖队列的算法 2 中，由于边贪心调度的边需要考虑有向边中拓扑后续节点的调度情况，因此时间复杂度为$O(\|V\|^2m^2)$。与之相比，EFT 的时间复杂度为$O(|V|m)$，HEFT 的时间复杂度为$O(|V|^2m)$，PEFT 的时间复杂度为$O(|V|^2m)$，LO 的时间复杂度为$O(|V|^4m^3)$，IPPTS 的时间复杂度为$O(|V|^2m)$。由于常见异构多处理器系统中的处理器较少 ($m\leqslant16$)，因此 m 对于时间复杂度的影响远小于$|V|$，HECSA 仍然是时间复杂度低的表调度算法。这样的调度方式的运算复杂度与点队列调度相当，远没有 LO 那么高的复杂度。

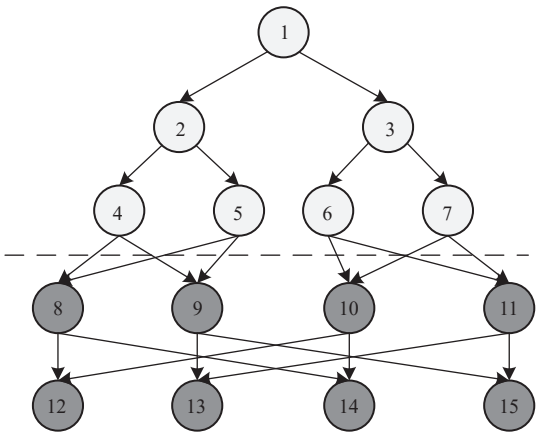
以图 1 所示的 DAG 任务中的v_1与v_4为例，对于点队列$[v_1,v_4]$，在启发式参数 EFT 的调度下，v_1将会被先分配到 $P2$ 上，随后v_4会被分配到 $P2$ 上，最终的结束时间为 31。而对于边覆盖队列$[e_{1,4}]$，由于v_1和v_4均未被分配，则v_1将会先分配到使得v_4的EFT最小的处理器上，即 $P1$，随后v_4也会分配到 $P1$，最终的结束时间为 29。边覆盖队列的思路就是尽可能将关键节点通过局部贪心的策略，最终获得更好的调度结果。

3 仿真试验结果

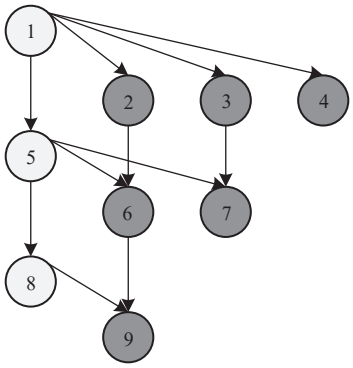
本节将 HECSA 算法与 EFT、HEFT、CPOP、LO、PEFT 和 IPPTS 算法进行仿真对比。这些算法都通过 C++编码，并在 AMD 5800x 3.8 GHz 和 32 GB RAM 的同一台计算机上运行。

3.1 快速傅里叶变换

如图 2 所示，将快速傅里叶变换（FFT）算法分为两部分：递归调用和蝶形运算。快速傅里叶变换算法的子任务数由 FFT 点数x决定，一共有$2(x-1)+1$次递归调用任务和$x\log_2 x$个蝶形运算任务。本文定义了 4 个变量生成了一系列 FFT 算法任务，其中x代表 FFT 的输入点数，随着x从 8 变为 64，FFT 任务的 DAG 规模$|V|$也由 40 变为 512，即由小型 DAG 任务变为大型 DAG 任务；CCR 代表 FFT 任务的通信计算比，随着 CCR 的变化 DAG 任务由计算密集型任务变为通信密集型任务；m代表异构多处理器系统中的处理器数量，m由 2 依次翻倍变为 16，模拟了资源匮乏和资源较为丰富的异构多处理器系统；β代表了异构多处理器系统中的异构度，设置β为 0.6 和 1，代表异构度较低和异构度较高的多处理器系统。对于每个参数组合（如$\text{DAG}_{8,1,8,0.6}$代表 $x=8$，$\text{CCR}=1$，$m=8$，$\beta=0.6$），均生成了 1 张 DAG 用于测试，共有 128 种不同 FFT 任务的 DAG 用于测试。具体的各参数设置如下。x=8, 16, 32, 64；COR=0.1, 0.5, 1, 5；m=2, 4, 8, 16；β=0.6, 1。



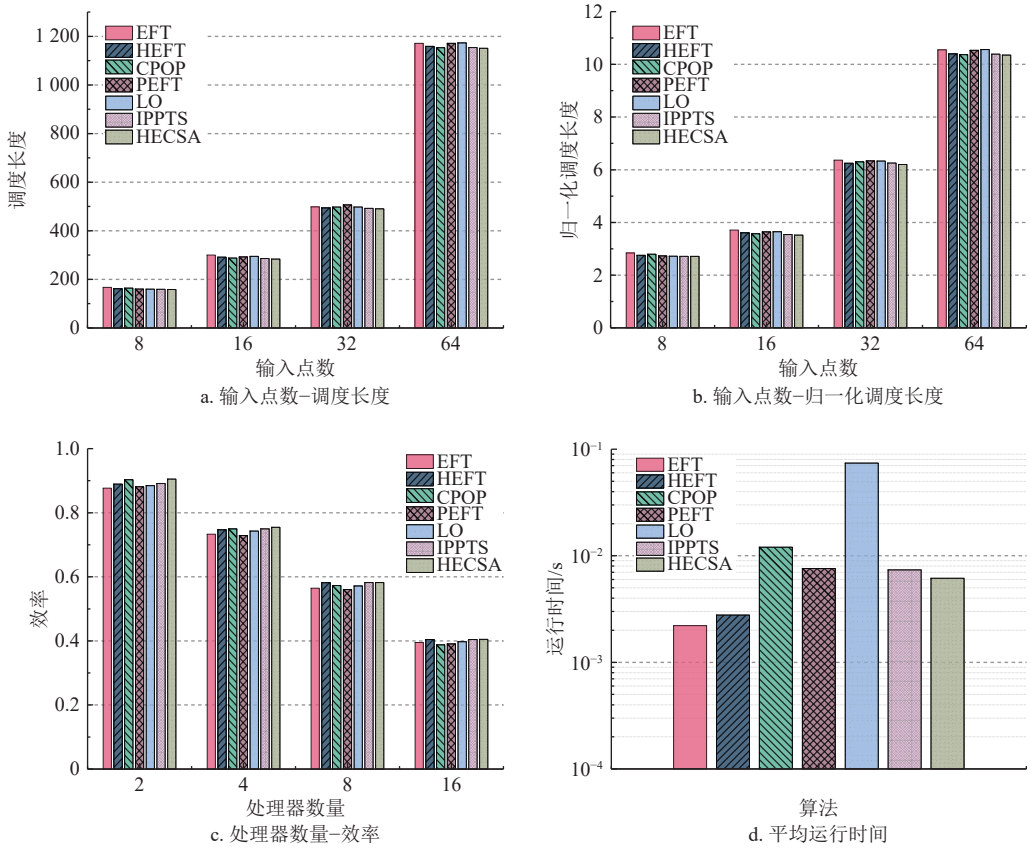
a. 快速傅里叶变换



b. 高斯消元法

图 2　典型数字信号处理器模型

本文对不同参数下的不同算法调度快速傅里叶任务表现出来的调度长度、效率平均运行时间和归一化调度长度进行了比较，如图 3 所示。可以看出，PEFT 和 LO 的结果表现最差，这与 HEFT 和 PEFT 算法中的结论类似[2, 6]。虽然 LO 和 PEFT 使用广泛，但在调度快速傅里叶任务上不尽如人意。这是因为在快速傅里叶变换中，所有任务都属于关键路径。大部分路径都是关键时，PEFT 的乐观成本矩阵即当前任务对后续任务影响的启发式方法失效，带来了坏的调度结果。这同样也适用于 LO，当前任务为子任务或多级子任务执行时间最早的启发式方法同样失效。LO 高昂的算法复杂度和运算时间并没有带来好处，拥有着数倍于其他对比算法的运行时间。HECSA 表现良好，在运行时间相当的情况下相对于 EFT 在调度长度上降低 2.53%，相对于 HEFT、CPOP、LO、PEFT 和 IPPTS 分别降低了 1.10%、0.93%、2.21%、1.97% 和 0.42%。HECSA 也是效率最高的调度算法。



a. 输入点数–调度长度　b. 输入点数–归一化调度长度

c. 处理器数量–效率　d. 平均运行时间

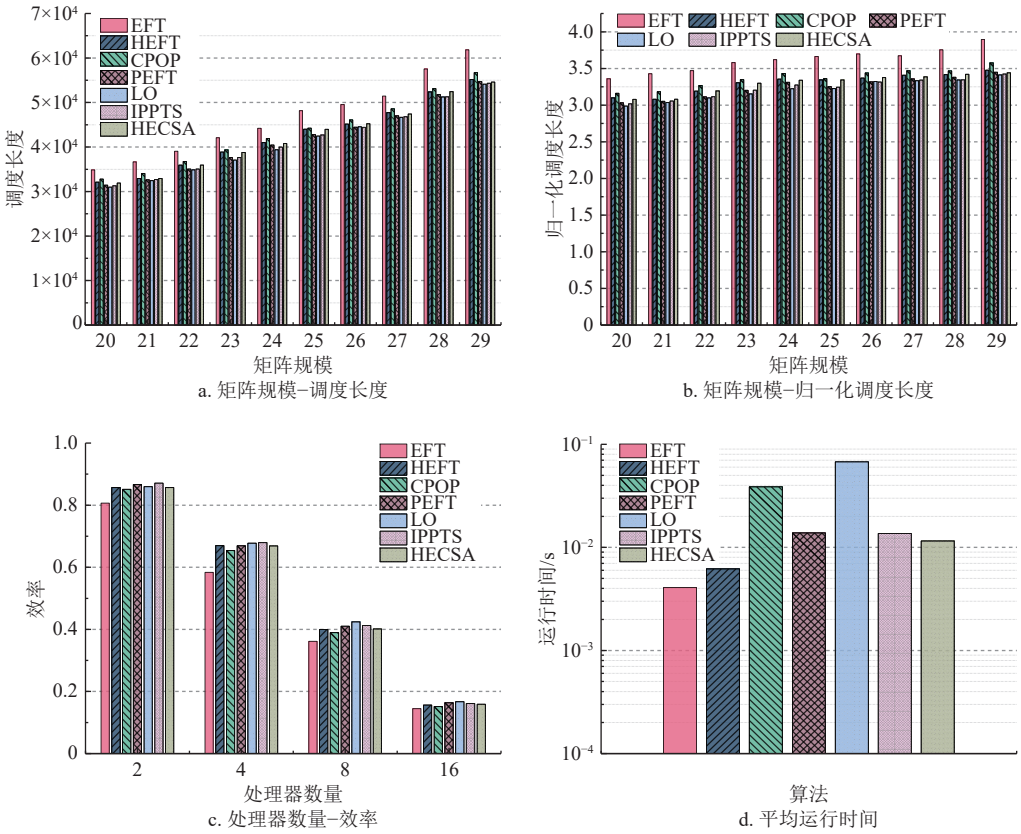
图 3　不同算法调度快速傅里叶变换任务的调度性能

3.2　高斯消元法

高斯消元法（GE）可以用来求解线性方程组、矩阵秩和逆矩阵，被广泛用于通信领域和信号处理领域。高斯消元图中的任务总数等于 $(x^2+x-2)/2$。与 FFT 任务相同，GE 同样定义了 4 个变量生成了一系列 GE 算法任务，但是在 GE 任务中 x 代表待求解的矩阵规模。x=20, 21, 22, 23, 24, 25, 26, 27, 28, 29；COR=0.1, 0.5, 1, 5；m=2, 4, 8, 16；β=0.6, 1。

同样对不同参数下的不同算法调度高斯消元法

任务表现出来的调度长度、效率平均运行时间和归一化调度长度进行比较，如图 4 所示。可以看出，LO 算法的结果表现最好，与 PEFT 研究中的结论类似[6]。PEFT 和 IPPTS 的性能也较好，在调度长度上与 LO 接近略低于 HECSA。HECSA 表现出了略差于的 LO 性能，但是相较于 LO，HECSA 运行时间节约了一半以上。相较于 PEFT 和 IPPTS 也拥有更短的运行时间。



a. 矩阵规模–调度长度　b. 矩阵规模–归一化调度长度

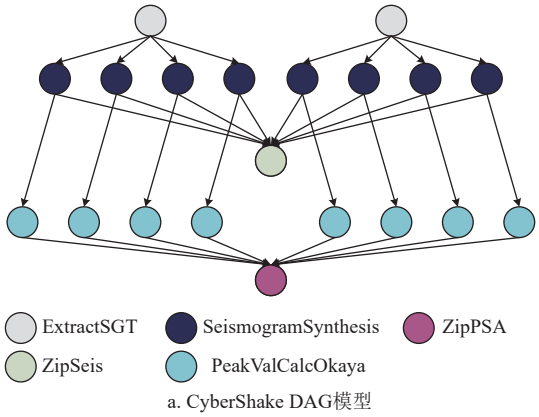
c. 处理器数量–效率　d. 平均运行时间

图 4　不同算法调度高斯消元法任务的调度性能

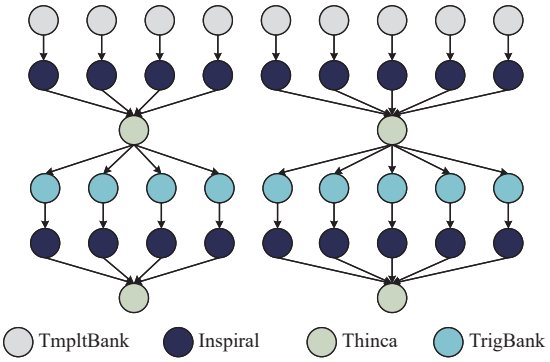
3.3　科学工作流

科学工作流也是常见的信息系统中的计算方法。CyberShake 工作流用于描述某个地区的地震危险性；而 LIGO 工作流则是激光干涉引力波探测应用任务。图 5 展示了 CyberShake 工作流和 LIGO 工作流的 DAG 模型。由于科学工作流任务结构固定，设置任务规模 $|V|=100$，$\beta=1$，CCR = 1。以不同处理器数量 $m=[2,4,8,16]$ 来获得 CyberShake 工作流和 LIGO 工作流任务调度结果。

如图 6 所示，无论是 CyberShake 工作流还是 LIGO 工作流任务，HECSA 均表现出了较好的结果，是调度长度最低的算法。对于 CyberShake 工作流任务，相对于 EFT 在调度长度上有着 4.03% 的降低，相对于 HEFT、CPOP、LO、PEFT 和 IPPTS 分别降低了 4.00%、4.28%、2.62%、3.21% 和 2.46%。对于 LIGO 工作流任务，相对于 EFT 在调度长度上有 1.66% 的降低，相对于 HEFT、CPOP、LO、PEFT 和 IPPTS 分别降低了 1.83%、2.37%、6.58%、1.99% 和 1.26%。在运行时间方面，HECSA 也表现出了接近的性能，仍然是低时间复杂度的表调度算法。

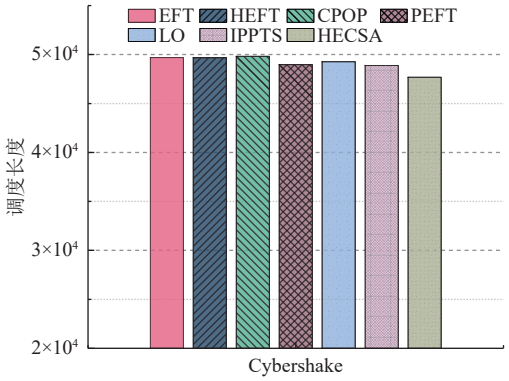


a. CyberShake DAG模型

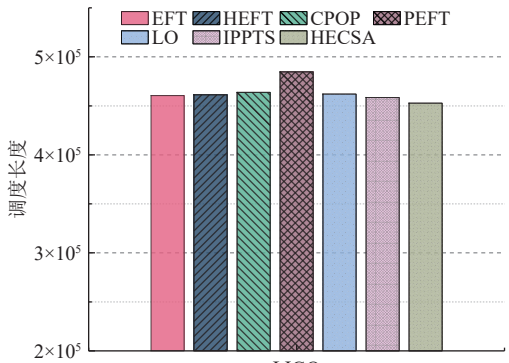


b. LIGO DAG模型

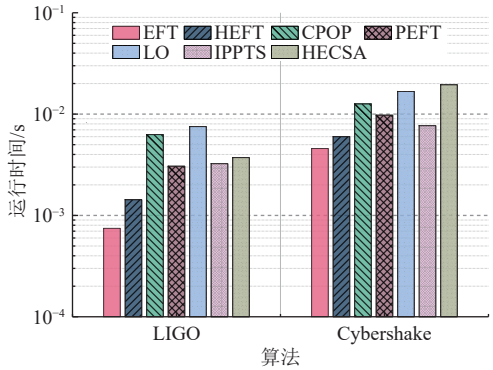
图 5　典型科学工作流任务 DAG 模型



a. CyberShake任务调度长度



b. LIGO任务调度长度



c. 平均运行时间

图 6　不同算法调度科学工作流任务的调度性能

3.4　小结

EFT 和 HEFT 的平均运行时间较短，因为其使用单一的优先级参数来进行调度队列的拓扑排序，而 PEFT、IPPTS 和 HECSA 算法利用启发式优先级参数矩阵来进行调度队列的拓扑排序；不仅如此，还引入了插入策略，因此带来了运行时间上的提升。对于 NPH 问题，通过低运算时间提升性能。

对于快速傅里叶算法，HEFT 被认为是优异的调度算法，而 PEFT 等预测类算法表现较差，这是因为快速傅里叶算法关键路径较多，PEFT 等预测类算法无法准确预测并为关键任务分配到合适的处理器。HECSA 通过消极成本矩阵（NCM）将关键路径上的任务尽可能在排在拓扑靠前的位置，并通过局部贪心的调度方式，将关键任务给予更好的局部调度，最终实现了较好的调度效果。对于高斯消元法，HECSA 通过混合贪心策略，在较快的运行时间内实现了较好的调度效果。对于科学工作流任务，HECSA 通过混合贪心策略也实现了高性能的调度。

4　结束语

本文为 DAG-SP 设计了一种新的调度算法 HECSA。高效的 DAG 任务调度算法可以提高嵌入式系统和物联网系统的性能，从而提高用户体验的质量。此外，优秀的任务调度算法可以升级具有多个内核和多个虚拟机的单个处理器中的资源平衡。通过常用数字信号处理的 DAG 任务和科学工作流任务仿真，结果证明了 HECSA 算法的优势。HECSA 相比于常用异构多处理器系统调度算法，在短的运行时间内给出了更接近最优的解决方案。

未来将从两个方面进行研究。首先是非确定性任务调度模型。在该模型中，单个任务节点所需的计算资源和完成时间不再已知或准确预测，如 D2D 网络[15]。当面对此类调度问题时，现有的任务调度算法将导致系统性能显著下降。其次，在 HECSA 的研究中注意到上述启发式边覆盖队列生成方法生成的边覆盖队列的拓扑质量不能有效发挥边覆盖队列的优势。不仅如此，边覆盖队列中边与边之间的拓扑关系对调度结果的影响也极大。因此，后续将继续研究边覆盖队列生成算法用于提升边覆盖队列拓扑质量，以进一步提高边覆盖队列调度算法性能。

参 考 文 献

[1] ZHANG L X, ZHOU L Q, SALAH A. Efficient scientific workflow scheduling for deadline-constrained parallel tasks

in cloud computing environments[J]. Information Sciences, 2020, 531: 31-46.
[2] TOPCUOGLU H, HARIRI S, WU M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260-274.
[3] YU D J, YING Y K, ZHANG L, et al. Balanced scheduling of distributed workflow tasks based on clustering[J]. Knowledge-Based Systems, 2020, 199: 105930.
[4] HU B, CAO Z C. Minimizing resource consumption cost of DAG applications with reliability requirement on heterogeneous processor systems[J]. IEEE Transactions on Industrial Informatics, 2020, 16(12): 7437-7447.
[5] WU C G, WANG L, WANG J J. A path relinking enhanced estimation of distribution algorithm for direct acyclic graph task scheduling problem[J]. Knowledge-Based Systems, 2021, 228: 107255.
[6] HAMID A, JORGE G B. List scheduling algorithm for heterogeneous systems by an optimistic cost table[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(3): 682-694.
[7] LUIZ F B, RIZOS S, EDMUNDO M. DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm[C]//2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing. Pisa: IEEE, 2010: 27-34.
[8] ELAZIZ M A, XIONG S W, JAYASENA K, et al. Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution[J]. Knowledge-Based Systems, 2019, 169: 39-52.
[9] XU Y M, LI K L, HU J T, et al. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues[J]. Information Sciences, 2014, 270: 255-287.
[10] SIH G C, LEE E A. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures[J]. IEEE Transactions on Parallel and Distributed Systems: A Publication of the IEEE Computer Society, 1993, 4(2): 175-187.
[11] NIMA E, LIANG B. Joint offloading decision and resource allocation with uncertain task computing requirement[C]//IEEE INFOCOM 2019 - IEEE Conference on Computer Communications. Paris: IEEE, 2019: 1414-1422.
[12] CHEN Y M, LIU S L, CHEN, Y J, et al. A scheduling algorithm for heterogeneous computing systems by edge cover queue[J]. Knowledge-Based Systems, 2023, 265: 110369.
[13] HAMZA D, FENG J, LU J M. Task scheduling for heterogeneous computing using a predict cost matrix[C]//ICPP Workshops '19: Workshop Proceedings of the 48th International Conference on Parallel Processing. Kyoto: ACM, 2019: 10.
[14] HAMZA D, FENG J, LU J M, et al. IPPTS: An efficient algorithm for scientific workflow scheduling in heterogeneous computing systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 32(5): 1057-1071.
[15] CHANG W, XIAO Y, LOU W, et al. Offloading decision in edge computing for continuous applications under uncertainty[J]. IEEE Transactions on Wireless Communications, 2020, 19(9): 6196-6209.

编 辑 税 红