

引用格式: 骆亮. 多核平台两级抢占式固定优先级 DAG 递归调度[J]. 微电子学与计算机, 2020, 37(4): 70-75. [LUO L. Two level preemptive global fixed priority DAG recursive scheduling for multi-core platforms[J]. Microelectronics & Computer, 2020, 37(4): 70-75.]

多核平台两级抢占式固定优先级 DAG 递归调度

骆 亮

(广西大学 计算机与电子信息学院, 广西 南宁 530004)

摘 要: 为提高应用程序调度实时性, 提高程序调度应用价值, 采用有向无环图(DAG)建立多核平台上的并行和递归任务的集合, 获得具有优先级约束(即有向边)的子任务(即节点), 并保证在某个指定的截止日期之前完成其所有子任务的执行. 每个任务都可能生成无限数量的实例, 其中连续实例的发布被一些最小到达时间隔开. 同时, DAG 任务的每个子任务被分配一个固定优先级, 提出一种两级抢占式全局固定优先级调度(GFP)策略: 任务级调度器首先确定最高优先级就绪任务, 然后, 子任务级调度器选择其最高优先级子任务来执行. 然后, 为所提两级 GFP 调度器推导出一个可调度性测试, 如果满足该测试则保证所有任务在 GFP 下满足其最后期限. 最后, 通过实验分析, 验证了所提算法在接受率指标上的性能优势.

关键词: 多核平台; 两级; 抢占式; 固定优先级; 有向无环图; 递归调度

中图分类号: TP391.9

文献标识码: A

文章编号: 1000-7180(2020)04-0070-06

Two level preemptive global fixed priority DAG recursive scheduling for multi-core platforms

LUO Liang

(College of Computer and Electronic Information, Guangxi University, Nanning, 530004, China)

Abstract: For real-time application scheduling, directed acyclic graph (DAG) is used to set up a set of parallel and recursive tasks on multi-core platforms to obtain sub-tasks (i. e. nodes) with priority constraints (i. e. directed edges) and to ensure that all the sub-tasks are executed before a specified deadline. Each task may generate an unlimited number of instances, in which the publication of successive instances is separated by some minimum arrival time. At the same time, each sub-task of the DAG task is assigned a fixed priority. A two-level preemptive global fixed priority scheduling (GFP) strategy is proposed. The task-level scheduler first determines the highest priority ready task, and then the sub-task-level scheduler selects its highest priority sub-task to execute. The sub task level scheduler then selects its highest priority sub task to perform. Then, a schedulability test is derived for the proposed two-level GFP scheduler, which guarantees that all tasks meet their deadlines under GFP. Finally, through experimental analysis, the performance advantage of the proposed algorithm on acceptance rate index is verified.

Key words: multi-core platform; two level; preemptive; fixed priority; directed acyclic graph; recursive scheduling

1 引言

随着计算数据量的大幅度增加, 使用多核处理

器的计算能力来满足更高级功能的不断增长的需求成为人们关注的焦点^[1-2]. 此类系统的主要挑战之一是最大限度地利用并行多核体系结构, 顺序编程是

在多核平台和多核平台上实现实时任务的主要模式. 顺序范例不允许任务内并行; 每个任务必须顺序执行, 这限制其处理能力提升^[3].

早期关于并行任务模型的许多工作针对各种调度算法提出了资源增加界限和可调度性测试. 调度算法的资源扩充界限表明, 如果存在对多个相同单位速度核上的任务集进行调度, 则保证算法成功地调度多个核上的相同任务集进行调度^[4-5]. 然而, 资源增加界限不能用作可调度性测试, 因为它是基于假设的最优调度器导出的. 考虑到每个组任务模型, 关于多核上并行任务的实时调度可分为三类^[6]: fork-join 模型; 同步并行任务模型; DAG 任务模型. 例如, 文献[7]中提出了多项式和伪多项式可调度性测试, 用于调度分治(即递归)DAG 任务, 使用基于全局动态优先级的最早截止期限优先(GEDF)调度. 文献[8~9]对于多个离散 DAG 任务的 GEDF 调度, 从理论分析角度导出了资源增加界限. 文献[10]提出多个递归 DAG 任务的响应时间计算结果. 以上算法是一种全局性的调度. 对于一些不频繁(即, 最坏)的情况, DAG 的完成时间可能非常大, 且可能错过任务最后期限.

对此, 本文提出了一种考虑子任务级的 GFP 调度的递归 DAG 任务的可调度性分析. 基于每个 DAG 任务的结构(即拓扑结构), 提出了一种为 DAG 任务的子任务分配固定优先级的简单方法.

2 系统模型描述

考虑了在具有 m 核的多核处理器上, 集合 $\Gamma = \{G_1, G_2, \dots, G_n\}$ 中的 n 次递归 DAG 任务的抢占式 GFP 调度. 每个核的归一化速度是 1. 每个 DAG 任务 G_k 的特征可使用四个参数 $G_k = (T_k, D_k, V_k, E_k)$ 进行定义, 其中: (1) T_k 是任务 G_k 的连续作业的最小到达时间(也称为周期); (2) D_k 是相对截止时间, 即 $D_k \leq T_k$; (3) $V_k = \{v_{k,1}, v_{k,2}, \dots, v_{k,n_k}\}$ 是集合的 n_k 个并行子任务; (4) $E_k \subset (V_k \times V_k)$ 是一组有向弧(或边). DAG 任务 G_k 的一个例子在图 1 中示出, 其中 $T_k = 100$ 和 $D_k = 52$ ^[11-12].

假设 G_k 正好有一个源点(表示为 v_k^{src})和汇点(表示为 v_k^{sink}). 如果存在多个源(sink)并行子任务, 则将 WCET 等于零的虚拟并行子任务作为新的源(sink)并行子任务用圆弧添加到(从)所有实际源(sink)并行子任务. 定义任务 G_k 的 DAG 的一个路径, 该并行子任务源于并行子任务 $(v_{k,a}, \dots, v_{k,b})$, 使得 $(v_{k,j}, v_{k,j+1}) \in E_k, a \leq j < b$. 路径的长度是路径

中并行子任务的 WCET 的总和. G_k 的最长路径由 L_k 表示, L_k 是 G_k 中最长路径的长度. 例如, 对应图 1 中路径 $(v_{k,1}, v_{k,3}, v_{k,4}, v_{k,6})$ 的 $L_k = 46$.

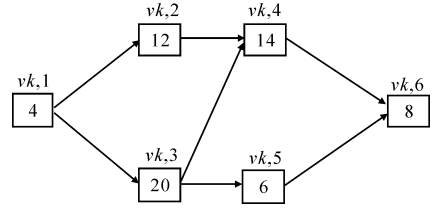


图 1 具有六个并行子任务的 DAG 任务

采用 W_k 表示任务 G_k 的总功, 它等于 V_k 的所有并行子任务(即 $W_k = \sum_{j=1}^{n_k} C_{k,j}$) 的 WCET 之和. 当 Ancst_k^j 中的每个并行子任务完成执行(即, 释放所有依赖项)时, 并行子任务 $v_{k,j} \in V_k$ 就准备好执行. 在 G_k 表示中, L_k, W_k 和集合 Ancst_k^j 的值可以在多项式时间内计算^[13~14].

(1) 任务级优先级分配. n 个任务的固定优先级 $\{G_1, G_2, \dots, G_n\}$ 是基于截止期单调(DM) 优先级排序级排序分配. 由 hpt_k 表示具有比任务 G_k 更高的固定优先级的任务集.

(2) 并行子任务级优先级分配. 基于 G_k 并行子任务的拓扑顺序, 对 G_k 并行子任务的固定优先级进行分配. 线性时间内可以计算 DAG 大小 G_k 的的拓扑顺序. 如果并且仅当 $v_{k,j} \in V_k$ 以 G_k 的拓扑顺序出现在并行子任务 $v_{k,g}$ 之前, 则并行子任务 $v_{k,j}$ 被分配比并行子任务 $v_{k,g} \in V_k$ 更高的固定优先级. 图 1 中 DAG 的 G_k 的拓扑排序如下($u > w$ 意味着 u 具有比 w 更高的优先级):

$$v_{k,1} > v_{k,3} > v_{k,2} > v_{k,5} > v_{k,4} > v_{k,6} \quad (1)$$

给定特定任务 G_k 的并行子任务 $v_{k,j} \in V_k, V_k$ 中具有比 $v_{k,j}$ 更高的固定优先级的并行子任务集由 hpt_k^j 表示.

(3) 两级 GFP 调度器. DAG 的并行子任务基于两级调度器执行. 任务级调度器确定最高优先级就绪任务 G_{np} , 而并行子任务级调度器选择 G_{np} 的最高优先级并行子任务来执行. 当所有内核都忙碌时, 允许新发布的相对较高优先级的并行子任务抢占较低优先级的并行子任务的执行.

3 多核平台并行任务作业时间响应

3.1 并行子任务时间响应

如果 $R_k \leq D_k$, 则保证 G_k 的所有工作都达到了最后期限. 为计算 R_k , 计算任务 G_k 的每个并行子任

务 $v_{k,j} \in V_k$ 的响应时间 R_k^i . 响应时间 R_k 等于接收器并行子任务 v_k^{sink} 的响应时间:

$$R_k = \max_{v_{k,j} \in V_k} \{R_k^j\} = R_k^{\text{sink}} \quad (2)$$

在计算(低优先级)多核平台并行任务 G_k 的响应时间之前, 计算 hpt_k 中的高优先级任务的响应时间. $v_{k,j}$ 的响应时间仅在计算 $Ancst_k^i$ 中所有其祖先 3 个并行子任务的响应时间之后才计算. 为确定并行子任务 $v_{k,j}$ 的响应时间 R_k^i , 遵循以下四个步骤:

步骤 1:(预备时间) 首先确定并行子任务 $v_{k,j}$ 准备好执行的最新时间(即, rdy_k^i 的值), 相对于 G_k 的任何(任意)作业的源并行子任务 $v_{k,sc}$ 的释放时间. 通过考虑并行子任务 $v_{k,j}$ 必须具有不小于 t 的响应时间, 在其长度 t 的问题窗口 $[rdy_k^i, rdy_k^i + t]$ 内执行 $v_{k,j}$ 的可调度性分析, 以计算 R_k^i . $v_{k,j}$ 准备执行的最新时间是其所有祖先的最大响应时间:

$$rdy_k^i = \max_{v_{k,a} \in Ancst_k^i} \{R_k^a\} \quad (3)$$

步骤 2:(任务内冲突发现工作量) 在问题窗口 $[rdy_k^i, rdy_k^i + t]$ 上, 确定并行子任务 $v_{k,j}$ 上的集合 hpt_k^i 中较高优先级子任务最大冲突工作负载.

由于 $v_{k,j}$ 的所有祖先必须在 rdy_k^i (根据等式 (3)) 之前完成, 所以 $v_{k,j}$ 的任何祖先都不能干扰问题窗口 $[rdy_k^i, rdy_k^i + t]$ 内的 $v_{k,j}$ 的执行. 来自集合 hpt_k^i 的可能干扰并行子任务 $v_{k,j}$ 的执行的高优先级并行子任务在集合 S_k^i 中:

$$S_k^i = hpt_k^i - Ancst_k^i \quad (4)$$

如果 $R_k^h \leq rdy_k^i$, 然后, 高优先级并行子任务 $v_{k,h}$ 在时间 rdy_k^i 之后完成其执行, 且最多贡献 $(R_k^h - rdy_k^i)$ 作为问题窗口内的任务内冲突工作负载. 否则, 存在 $R_k^h \leq rdy_k^i$, 这意味着并行子任务 $v_{k,h}$ 已经按时间 rdy_k^i 完成其任务执行, 并且不能对任务内冲突工作负载做出贡献. 由于集合 $S_k^i = hpt_k^i - Ancst_k^i$ 中的所有高优先级并行子任务, 由 W_k^{intra} 表示的任务内冲突工作负载的总和:

$$W_k^{\text{intra}} = \sum_{v_{k,h} \in S_k^i} \min\{C_{k,h}, \max\{0, R_k^h - rdy_k^i\}\} \quad (5)$$

步骤 3:(任务间冲突发现工作量) 在问题窗口 $[rdy_k^i, rdy_k^i + t]$ 中, 在并行子任务 $v_{k,j}$ 上确定集合 hpt_k 中较高优先级并行子任务最大冲突工作负载.

在并行子任务 $v_{k,j}$ 的问题窗口 $[rdy_k^i, rdy_k^i + t]$ 中执行任务 $G_i \in hpt_k$ 的不同作业被分成三类: 进位作业、本体作业、执行作业. 进位作业是任务 G_i 的第一个作业, 它在 $[rdy_k^i, rdy_k^i + t]$ 中执行, 使得它的释

放时间在 rdy_k^i 之前, 并且它的截止日期在 $[rdy_k^i, rdy_k^i + t]$ 中.

步骤 4:(冲突查找和响应时间) 基于任务内和任务间冲突工作负载, 确定并行子任务 $v_{k,j}$ 在其问题窗口中遭受的最大冲突. 最后, 基于冲突确定响应时间 R_k^i .

3.2 最坏情况任务间冲突释放模式

最坏情况释放模式指的是高优先级任务 $G_i \in hpt_k$ 的作业的释放和执行对其问题窗口内的低优先级任务造成最大干扰的场景. 考虑下面的最坏情况下的释放模式, 如图 2 所示:

模式 1: 执行作业在时间 $(rdy_k^i + t - W_i/m)$ 时开始执行, 并在 $(rdy_k^i + t)$ 时间前完成工作量 W_i ;

模式 2: 所有早先的工作都尽可能晚地到达(即严格地周期性地);

模式 3: 进入作业 G_i 在时间 $(rdy_k^i + t_{\text{cin}})$ 完成其执行, 使得 $(rdy_k^i + t_{\text{cin}} - R_i)$ 是某些 $t_{\text{cin}} > 0$ 的 G_i 的进入作业的释放时间, 其中 R_i 是任务 G_i 的响应时间.

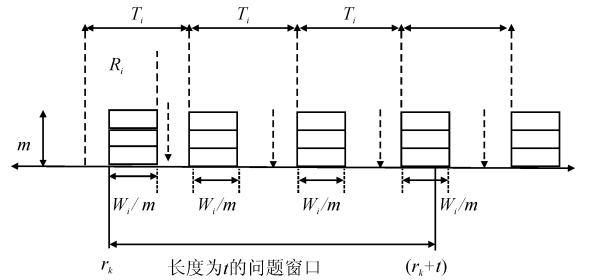


图 2 最坏情况下的释放模式

根据图 2, 进位和本土作业在区间 $[rdy_k^i, rdy_k^i + t - W_i/m]$ 中执行. 给出了区间 $[rdy_k^i, rdy_k^i + t - W_i/m]$ 的长度 $X_i(t)$ 如下:

$$X_i(t) = \max\{0, rdy_k^i + t - W_i/m\} \quad (6)$$

其中, rdy_k^i 可利用等式 (3) 进行计算. 接下来确定 t_{cin} 值, 它是在问题窗口中执行 τ_i 的进入作业间隔长度. 根据图 3, 进位任务在时间 $(rdy_k^i + t_{\text{cin}})$ 上完成其执行. 执行进位作业的问题窗口内的间隔长度(即, t_{cin} 值) 计算如下:

$$t_{\text{cin}} = t - (T_i - R_i) - \lfloor \frac{X_i(t)}{T_i} \rfloor \cdot T_i - \frac{W_i}{m} \quad (7)$$

为避免方程 (7) 中 t_{cin} 的长度出现负值, 将 t_{cin} 的值下界设置为零, t_{cin} 给出如下:

$$t_{\text{cin}} = \max\left\{0, t - (T_i - R_i) - \lfloor \frac{X_i(t)}{T_i} \rfloor \cdot T_i - \frac{W_i}{m}\right\} \quad (8)$$

如果 $t_{\text{cin}} = 0$, 则不存在进位作业. 在问题窗口内执行的唯一 G_i 作业是本体作业和执行作业. 利用 $CR_i(t_{\text{cin}})$ 表示 $[rdy_k^i, rdy_k^i + t_{\text{cin}}]$ 中任务 G_i 的进位作业的最大任务间冲突工作量. $G_i \in hpt_k$ 任务间冲突工作量为: $(CR_i(t_{\text{cin}}) + \lfloor X_i(t)/T_i \rfloor \cdot W_i + W_i)$. 在 $v_{k,j}$ 的问题窗口内 hpt_k 中的所有任务之间的任务间冲突工作量为:

$$W_{k,t}^{\text{inter}} = \sum_{G_i \in hpt_k} \left(CR_i(t_{\text{cin}}) + \lfloor \frac{X_i(t)}{T_i} \rfloor \cdot W_i + W_i \right) \quad (9)$$

通过考虑任务 $G_i \in hpt_k$ 的进位作业的子任务在 $[rdy_k^i, rdy_k^i + t_{\text{cin}}]$ 间隔内所做的工作来确定 $CR_i(t_{\text{cin}})$ 的值. R_i^g 是子任务 $v_{i,g} \in V_i$ 的响应时间, 在计算 $v_{k,j}$ 的响应时间之前已得到计算. 基于步骤2相似分析, G_i 的进位作业高优先级子任务 $v_{i,g}$ 最多可执行 $v_{k,j}$ 问题窗口内 $\min\{C_{i,g}, \max\{0, R_i^g - rdy_k^i\}\}$ 时间单位. A_i 表示任务 G_i 的进位作业集合 V_i 中所有高优先级任务的总冲突工作负载上限:

$$A_i = \sum_{v_{i,g} \in V_i} \min\{C_{i,g}, \max\{0, R_i^g - rdy_k^i\}\} \quad (10)$$

然而, 在 $[rdy_k^i, rdy_k^i + t_{\text{cin}}]$ 区间中 G_i 的进位作业任务间冲突工作量不能大于 $(m \cdot t_{\text{cin}})$, 因为最多可以在 m 个核上 t_{cin} 长度间隔内完成 m 个 t_{cin} 工作量. 因此, $CR_i(t_{\text{cin}})$ 值可计算为:

$$CR_i(t_{\text{cin}}) = \min\{m \cdot t_{\text{cin}}, A_i\} \quad (11)$$

在子任务 $v_{k,j}$ 的长度 t 的问题窗口中, 由 $I_{k,j}(t)$ 表示的总冲突工作负载是任务内和任务间冲突工作负载的总和:

$$I_{k,j}(t) = W_{k,t}^{\text{intra}} + W_{k,t}^{\text{inter}} \quad (12)$$

式中, $W_{k,t}^{\text{intra}}$ 和 $W_{k,t}^{\text{inter}}$ 可分别采用公式(5)和公式(9)所示. 子任务 $v_{i,j}$ 在其准备好执行之后执行, 在其问题窗口 $[rdy_k^i, rdy_k^i + t)$ 内的最多 $I_{k,j}(t)/m$ 时间单元被干扰. 使用下面递归给出子任务 $v_{k,j}$ 的响应时间:

$$t^{h+1} \leftarrow rdy_k^j + I_{k,j}(t^h)/m + C_{k,j} \quad (13)$$

在等式(13)中右侧的第一项是子任务 $v_{k,j}$ 准备执行的最近时间, 第二项表示在大小 t^h 的问题窗口中对子任务 $v_{k,j}$ 的干扰, 最后第三项 $C_{k,j}$ 是子任务 $v_{k,j}$ 的 WCET.

3.3 理论分析

现在在定理1中证明, 使用等式(13)来计算每个任务的响应时间的递归过程的正确性.

定理1: 如果对于 $k = 1, 2, \dots, n$, 存在 $R_k \leq D_k$, 则可得集合 $\Gamma = \{G_1, G_2, \dots, G_n\}$ 中的所有任务满足

其截止期限.

证明: 如果 $R_k \leq D_k$, 然后对每个子任务 $v_{k,j} \in V_k$ 根据等式(1)可得 $R_k^j \leq D_k$. 该定理的正确性可通过证明等式(13)正确地计算 $v_{k,j} \in V_k$ 的任务 G_k 的每个子任务上界获得.

基于公式(12), 可计算 $I_{k,j}(t) = W_{k,t}^{\text{intra}} + W_{k,t}^{\text{inter}}$, 则可将公式(13)改写为:

$$t^{h+1} \leftarrow rdy_k^j + \frac{W_{k,t}^{\text{intra}}}{m} + C_{k,j} + \frac{W_{k,t}^{\text{inter}}}{m} \quad (14)$$

式中, $W_{k,t}^{\text{intra}}$ 和 $W_{k,t}^{\text{inter}}$ 可利用前述公式进行计算. 根据图2分析结果, 高优先级任务 $G_i \in hpt_k$ 的作业的释放模式使任务间干扰工作负载 $W_{k,t}^{\text{inter}}$ 最大化. 在大小 t 的问题窗口中任务间干扰 $W_{k,t}^{\text{inter}}/m$ 可实现最大化.

任务内干扰工作负载 $W_{k,t}^{\text{intra}}$ 是通过考虑在时间 rdy_k^j 之后有资格执行的 hpt_k 中的每个较高优先级子任务完成的最大工作量来计算的, 其中 rdy_k^j 是节点 $v_{k,j}$ 准备执行的最新时间. 在完成 $v_{k,j}$ 的执行中的最大延迟包含: (1) 就绪时间 $(rdy_k^j - \Delta)$; (2) 任务内干扰 $\Delta + W_{k,t}^{\text{intra}}/m$; (3) 自身执行时间 $C_{k,j}$; (4) 任务间干扰 $W_{k,t}^{\text{inter}}/m$. 则响应时间模型为:

$$t^{h+1} \leftarrow (rdy_k^j - \Delta) + \Delta + \frac{W_{k,t}^{\text{intra}}}{m} + C_{k,j} + \frac{W_{k,t}^{\text{inter}}}{m} \quad (15)$$

这相当于等式(14)中的递归, 这又等价于等式(13). 因此, 式(12)中导出的递推通过考虑子任务 $v_{k,j}$ 在长度为 t^h 的问题窗口中遭受的最大干扰, 可正确地计算子任务 $v_{k,j}$ 的响应时间.

4 实验分析

4.1 实验设置

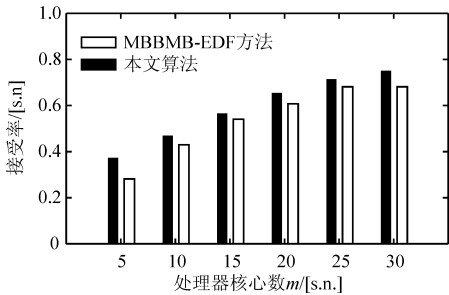
使用文献[15]中给出的 DAG 任务集生成算法, 实验的最大递归深度设置为2. 通过扩展非终端顶点来生成终端顶点和并行子图的概率分别是 p_{term} 和 $(1 - p_{\text{term}})$. 一旦子任务和 DAG 任务 G_k 的边缘被生成, 每个 DAG 任务 G_k 的参数生成如下: (1) 任务 G_k 的子任务 $v_{k,i} \in V_k$ 在范围 $[1, 100]$ 中被均匀地选择; (2) 计算最长路径 L_k 的长度和工作量 W_k ; (3) 在 $[L_k, W_k/\beta]$ 上均匀选取周期参数 T_k , 其中 $\beta \leq 1$ 用于控制任务 G_k 的最小利用率. G_k 的利用率 u_k 为 $[\beta, W_k/L_k]$.

实验中, 设置 $p_{\text{term}} = 0.5$, $n_{\text{par}} = 5$, $p_{\text{add}} = 0.1$ 和 $\beta = 0.1$. 对于任务集的目标总利用率 U , 重复向任务集添加新的 DAG 任务, 直到任务集的总利用率等于 U .

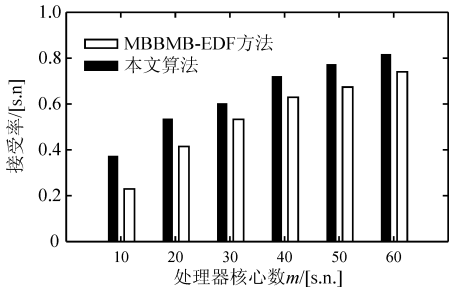
为了生成总利用率恰好等于 U 的任务集,调整最后一个任务的周期,使得任务集的总利用率恰好等于 U .在每个利用级别 $U = [0.25, 0.5, \dots, m]$,总共生成500个任务集,其中 m 是核数.在特定利用级别 $U = [0.25, 0.5, \dots, m]$ 生成的500个任务集中具有总利用率 U .对于特定的可调度性测试和 m, U 值,考虑接受率度量指标,其表示500个DAG任务集中的一部分,这些任务集通过以下测试保证可调度性,并计算了 U 和 m 的各种值的两个测试的接受率.

4.2 处理器核数影响实验

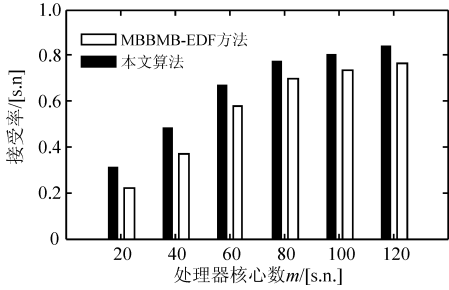
实验中改变内核数量,同时保持每个任务集的总利用率 U 固定.图8给出使用三种不同的总利用率: $U=2, U=4$ 和 $U=8$,验证所提方法和MBBMB-EDF测试方法对于 m 的不同值的接受率.当核数变大时,图3a~图3c中三个图中的接受率也增加.因为相对多的内核更有可能满足两个测试的任务集的所有最后期限.



(a) $U=2$



(b) $U=4$



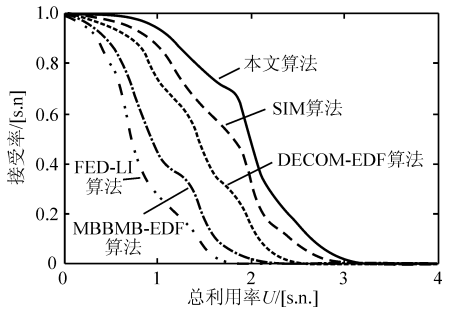
(c) $U=8$

图3 处理器核数影响实验

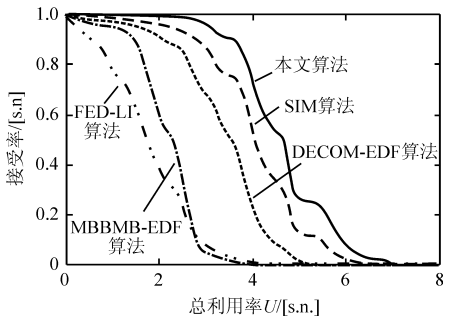
然而,对于许多资源受限的系统,其中大小、重量和功率是有限的,并且成本很高,负担不起更多数量的核心.在接受率指标方面,本文算法相对于选取的MBBMB-EDF测试方法具有更高的接受率.例如在图8c中,对于 $U=2$ 情形,本文算法当 $m \geq 8$ 时即可获得接受率为80%,而MBBMB-EDF测试方法需要 $m \geq 20$ 时方可获得接受率为80%.该结果显示了所提可调度性测试在减少硬件资源(尤其是核心数量)需求方面的有效性,可有效降低开发大规模生产系统的成本.

4.3 算法对比

本节选取三个可调度性测试方法作为对比算法,测试算法之间的接受率对比情况:(1)文献[12]中所提出MBBMB-EDF测试方法,其考虑了DAG任务的全局EDF调度;(2)文献[13]中所提出DECOM-EDF测试方法,通过向每个子任务分配人工发布时间和人工截止日期来分解每个DAG任务,其中使用全局EDF策略调度这些子任务;(3)文献[14]中所提出FED-LI测试方法,考虑联合调度,其中具有高利用率的DAG任务被分配专用数量的处理器;(4)文献[15]中所提出SIM测试方法,SIM的接受率是随机任务集合的接受率的上限,随机任务集合使用所提出的调度器实际上是可调度的.实验结果见图4所示.



(a) $m=4$



(b) $m=8$

图4 算法对比

根据图4结果可知,本文所提测试方法在接受率指标上,要显著的优于选取的对比算法.其次是文献[15]提出的SIM测试方法,本文方法要略优于SIM测试方法.而选取的MBMB-EDF等三种测试方法的接受率均要差于本文所提测试方法.上述实验结果验证了本文所提测试方法在多核平台上的性能优势.

5 结束语

本文提出了一种考虑子任务级的GFP调度的递归DAG任务的可调度性分析.基于每个DAG任务的结构(即拓扑结构),提出了一种为DAG任务的子任务分配固定优先级的简单方法.本文考虑一个特定的子任务级GFP调度器来调度子任务,以便有效地利用处理核.通过展开DAG任务的内部结构(即拓扑),提出一种简单但非常有效的启发式算法,将固定优先级分配给每个DAG任务的子任务.提出一种新的响应时间为基础的可调度性测试GFP调度的递归DAG任务.实验证明了所提检验方法的有效性.

参考文献:

- [1] UMARANI S G, GEETHA R. Task scheduling using Ant Colony Optimization in multicore architectures: a survey [J]. *Soft Computing*, 2018, 22 (15): 5179-5196.
- [2] ARASH D, MAHMOUD N, SAEID A. CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud[J]. *The Journal of Supercomputing*, 2017, 73(2): 756-781.
- [3] WANG Y, LI K L, LI K Q. Partition Scheduling on Heterogeneous Multicore Processors for Multi-dimensional Loops Applications[J]. *International Journal of Parallel Programming*, 2017, 45(4): 827-852.
- [4] 于俊清, 张维维, 陈文斌, 等. 面向多核集群的数据流程层次流水线并行优化方法[J]. *计算机学报*, 2014, 37(10): 2071-2083.
- [5] RISAT M P. Real-time scheduling algorithm for safety-critical systems on faulty multicore environments [J]. *Real-Time Systems*, 2017, 53(1): 45-81.
- [6] 亢良伊, 王建飞, 刘杰, 等. 可扩展机器学习的并行与分布式优化算法综述[J]. *软件学报*, 2018, 29(1): 109-130.
- [7] ANURADHA P, HEMALATHA R, NARSIMHA G. Energy efficient scheduling algorithm for the multicore heterogeneous embedded architectures[J]. *Design Automation for Embedded Systems*, 2018, 22 (2): 1-12.
- [8] 刘松, 伍卫国, 赵博, 等. 面向局部性和并行优化的循环分块技术[J]. *计算机研究与发展*, 2015, 52(5): 1160-1176.
- [9] ZHOU Z, WILLIAM P, SHUVRA S B, et al. Scheduling of Parallelized Synchronous Dataflow Actors for Multicore Signal Processing[J]. *Journal of Signal Processing Systems*, 2016, 83(3): 309-328.
- [10] 刘德儿, 熊证, 沈敬伟, 等. 完全拓扑关系结构 D-TIN 多核并行构建[J]. *计算机辅助设计与图形学学报*, 2016, 28(6): 950-959.
- [11] GEORGE F Z, WILLIAM PLISHKER, SHUVRA S BHATTACHARYYA, et al. Implementation, Scheduling, and Adaptation of Partial Expansion Graphs on Multicore Platforms[J]. *Journal of Signal Processing Systems*, 2017, 87(1): 107-125.
- [12] SANDRA C, FRANCISCO D I, RAFAEL M, et al. Architecture-aware configuration and scheduling of matrix multiplication on asymmetric multicore processors [J]. *Cluster Computing*, 2016, 19 (3): 1037-1051.
- [13] RAFAEL R S, ENRIQUE S Q. Architecture-aware optimization of an HEVC decoder on asymmetric multicore processors[J]. *Journal of Real-Time Image Processing*, 2017, 13(1): 25-38.
- [14] NAMHOON K, BRYAN C W, MICAIAH C, et al. Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning[J]. *Real-Time Systems*, 2017, 53(5): 709-759.
- [15] RYAN J, IPPOKRATIS P, RADU S, et al. Scalability of write-ahead logging on multicore and multisoocket hardware[J]. *The VLDB Journal*, 2012, 21 (2): 239-263.

作者简介:

骆亮男, (1984), 硕士, 高级工程师. 研究方向为大数据挖掘、信息系统设计开发、计算机网络及系统集成等.
E-mail: luuooliang@163.com