doi:10.12068/j.issn.1005-3026.2019.03.003

基于G-EDF的DAG并行任务多核响应时间分析

韩美灵,邓庆绪,张天宇,林宇晗
(东北大学 计算机科学与工程学院,辽宁 沈阳 110169)

摘 要:由于未考虑DAG(directed acyclic graph)任务的自身结构,基于G-EDF(global earliest deadline first)的DAG并行任务模型的可调度性分析存在很大的悲观性,因此本文针对DAG任务集在多处理器系统中采用G-EDF调度策略下的响应时间分析进行了研究.首先针对carry-in任务实例执行的情况提出更加精确的carry-in工作量估算方法.基于该carry-in工作量估算方法提出一种基于完成时间的问题窗口工作量估算方法.最后,结合上述两个改进策略提出了基于G-EDF的DAG任务响应时间分析方法.仿真实验表明,所提出的方法较目前已知的调度策略方法可调度性至少提高15%,最高可达25%.

关 键 词:嵌入式实时系统;多核处理器;并行任务模型;全局调度;响应时间分析
中图分类号:TP 316.2 **文献标志码**:A **文章编号**:1005-3026(2019)03-0315-06

Response Time Analysis of Multiprocessor Systems for DAG Parallel Tasks Based on G-EDF

HAN Mei-ling, DENG Qing-xu, ZHANG Tian-yu, LIN Yu-han
(School of Computer Science & Engineering, Northeastern University, Shenyang 110169, China. Corresponding author: DENG Qing-xu, E-mail: hanmeiling@stumail.neu.edu.cn)

Abstract: Since the self-structure of the DAG(directed acyclic graph) task is not considered, the schedulability analysis of the DAG parallel task model based on G-EDF(global earliest deadline first) is very pessimistic. The response time analysis of the DAG task set under the G-EDF scheduling strategy in multiprocessor systems was studied in this paper. First, a more accurate carry-in workload estimation method was proposed for the execution of the carry-in task instance. Then a method for estimating the problem window workload of completion time was put forward based on the carry-in workload estimation method. Based on the two proposed methods, this paper proposed a response time analyzing method to derive a response time upper bound of each task. The experiments show that the proposed method outperforms the state-of-the-art method by at least 15% and at most 25%.

Key words: embedded real-time systems; multiprocessors; parallel tasks model; global scheduling; response time analysis

随着半导体技术的发展,CMOS晶体管尺寸不断变小,在一个模组中可以集成更多的核.例如,因特尔公司的Intel Xeon Phi协处理器可以多达60个核左右[1].芯片在硬件设计上朝着越来越多核体系结构发展,然而目前的软件和调度策略并不能充分地利用多核的性能.例如:一个应用程序的执行时间是100个时间单位,无论系统中的核个数是多少,应用程序都必须在100个时间单位后完成执行.假如该应用程序可以同时在100个处理器上执行,那么仅需1个时间单位便可完成.

多核实时调度领域中常用的并行模型主要有:Fork-Join模型[2]、同步并行模型[3-6]以及DAG(directed acyclic graph)模型[7-12].由于DAG并行模型对节点并行约束较少,因此针对DAG并行模型的研究呈上升趋势.由于任务内的

收稿日期:2018-01-22
基金项目:国家自然科学基金资助项目(61472072,61528202);辽宁重大装备制造协同创新中心资助项目.
作者简介:韩美灵(1988-),女,山东聊城人,东北大学博士研究生;邓庆绪(1970-),男,河南南阳人,东北大学教授,博士生导师.

并行关系导致顺序执行任务模型的调度策略并不适用于并行任务模型.

本文针对 DAG 模型分析 DAG 任务集在多核处理器上的响应时间. 响应时间分析是实时嵌入式系统中判定任务可调度性的重要手段. 目前,并行任务的可调度性分析已初步取得了一些成果,但针对 G - EDF(global earliest deadline first)的响应时间分析的工作却相对欠缺. 文献[6]针对同步并行模型,提出一种基于干涉量的响应时间分析方法. 然而该方法和同步并行模型有很大的相关性,并不适用于 DAG 并行模型. 文献[10]提出基于 DAG 模型的响应时间分析方法,然而该方法却忽略了 DAG 模型的特有结构. 文献[12]针对 DAG 并行模型基于 G - EDF 的可调度性分析提出一种基于松弛时间的 G - EDF 调度策略,该方法为了获得分析效率而盲目过量估计特定窗口的任务执行情况,导致分析结果过于悲观.

本文针对以上问题,首先提出一种更加准确的 carry - in 工作量的估算方法. 其次,根据被分析任务实际完成的时间计算问题窗口的最大工作量. 最后,结合两种新提出的工作量估算策略分析被分析任务的最差响应时间. 本文通过仿真实验,验证了本文提出的分析方法的性能优于文献[12]中提出的算法. 下文中将文献[12]中提出的算法简称为 DAG - G - EDF,把新提出的基于改善 carry - in 工作量的 G - EDF 响应时间分析方法简称为 IMP - G - EDF - RTA.

1 问题模型

本文讨论的问题是在 M 个同构单位速率处理器组成的处理器平台上分析具有 n 个 DAG 任务组成的任务集,采用 G - EDF 调度策略进行调度时的可调度性. 任务集表示为 $\tau=\{\tau_1,\cdots,\tau_n\}$. 1 个 DAG 任务 τ_i 表示为 (G_i,D_i,T_i). 其中,G_i 表示有向无环图,D_i 表示任务 τ_i 的相对截止期,T_i 表示任务 τ_i 的两个连续实例的最小释放时间间隔也称为周期. $G_i=(V_i,E_i)$,V_i 为 G_i 所有节点的集合,E_i 为 G_i 所有边的集合. 每个节点 $\theta_{i,u}\in V_i$ 的最差执行时间(WCET: worst case execution time)表示为 $e_{i,u}$. E_i 中的 1 条有向边 $(\theta_{i,u},\theta_{i,v})$ 表示节点 $\theta_{i,v}$ 必须在节点 $\theta_{i,u}$ 完成后才能开始执行. 节点 $\theta_{i,u}$ 进入就绪状态当且仅当其所有直接前继完成了执行.

DAG 任务 τ_i 的 1 条路径定义为 G_i 的 1 个节点序列 $\theta_{i,1},\cdots,\theta_{i,f}$,该节点序列中连续的 2 个节点之间存在 1 条有向边,即 $(\theta_{i,j},\theta_{i,j+1})\in E_i$,$1\leqslant j<f$. 路径的长度定义为路径上所有节点的 WCET 之和. G_i 的关键路径定义为最长路径,表示为 LC_i. 任务 τ_i 的 WCET 表示为 C_i:

$$C_i=\sum_{v\in V_i}e(v).$$

DAG 任务 τ_i 会释放无限多的实例,每个实例的响应时间定义为从其释放到其完成之间的时间间隔. 任务 τ_i 的响应时间定义为所有实例的最差响应时间(WCRT: worst case response time): $R_i=\max(f_i-r_i)$. 显然枚举所有实例是无法实现的,因此常用方法是估算一个任务的 WCRT. 本文只考虑限制截止期任务的情况,即: $D_i\leqslant T_i$. 显而易见,LC_i 严格不大于 D_i 时任务 τ_i 才能被调度,但 $C_i\leqslant D_i$ 不是必须的. 任务 τ_i 的利用率表示为 U_i,且定义 $U_i=C_i/T_i$. 而任务集的利用率用 U 表示,且定义 $U=\sum_{\tau_i\in\tau}U_i$.

任务的优先级按照 EDF 进行分配,即每个已经释放且未完成的任务实例的最小绝对截止期优先调度的策略进行调度. 本文假设处理器时钟是单位时间的,即所有的任务相关参数都是正整数.

2 最新研究方法及其悲观性分析

本节首先介绍常用的多核处理器分析方法的基础理论[10],然后对文献[12]提出的 G - EDF 分析方法进行介绍,最后分析该方法的悲观性. 本文以任务 τ_k 作为被分析任务进行阐述.

2.1 问题窗口和工作量

定义 1(问题窗口)设被分析任务 τ_k 在 r_k 处释放在 f_k 处完成执行,则窗口 $[r_k,f_k]$ 称为任务 τ_k 的问题窗口.

任务集中的任何一个除了 τ_k 以外的任务在问题窗口内产生的工作量可以由两种类型的实例组成: carry - in 任务实例(该类实例在问题窗口前释放,但截止期在问题窗口内)和 body 任务实例(释放和截止期都在问题窗口内的实例). 每个任务 τ_i 只有一个 carry - in 任务实例称之为 J_i^{CI},而 body 任务实例有多个.

引理 1(最坏执行情况)当 $d_i=d_k$ 时,任务 τ_i 在问题窗口 $[r_k,f_k]$ 内的工作量最大.

如图 1 所示,该引理的正确性是显而易见的. 任务 τ_i 的 body 实例在窗口 $[r_k,f_k]$ 内的个数最多为

$$N_i^{\mathrm{b}} = \left\lfloor \frac{L_k - D_i}{T_i} \right\rfloor + 1 \ ,$$

$$L_k = f_k - r_k .$$

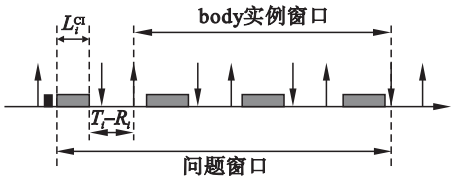

图 1　任务 τ_i 的最坏执行情况

Fig. 1　Worst case execution of task τ_i

图 1 仅仅为了示意任务的释放和执行，因此用 1 个矩形表示 DAG 任务的执行是合理的. 任务 τ_i 在长度为 L_k 的问题窗口内的工作量组成为 N_i^{b} 个 body 实例和 1 个 carry－in 实例. 因此，carry－in 执行的窗口长度为最差，即

$$L_i^{\mathrm{CI}} = (L_k - D_i) - (N_i^{\mathrm{b}} - 1) \times T_i .$$

文献[4]证明了当 carry－in 的实例执行模式为处理器个数是无限多的模式执行时 carry－in 的工作量最大，下文按此结论进行分析.

2.2　基本分析方法及其悲观性

为了忽略 DAG 任务的结构特性，文献[10]把 DAG 任务按照所有 WCET 平均分配到所有处理器上执行的模式进行分析，如图 2 所示.

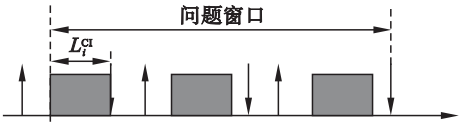

图 2　基本分析方法示例

Fig. 2　Illustration of baseline method

在该模式下任务的可调度性判定条件为

$$R_k = \mathrm{LC}_k + \frac{1}{M}\left(C_k - \mathrm{LC}_k + \sum_{\tau_i \in \tau \setminus \tau_k} I_{i,k}^{\mathrm{hp}}\right) ,$$

$$I_{i,k}^{\mathrm{hp}} = N_i^{\mathrm{b}} \times C_i + \min\left(\max(0, (R_k - D_i)) \bmod T_i, \frac{C_i}{M}\right) \times M .$$

该方法忽略 DAG 执行结构会在问题窗口内代入更多的干涉量. 为了改善这一悲观性，文献[12]提出了目前最新的研究方法.

2.3　DAG－G－EDF 调度方法及悲观性

文献[12]的基本思路是：在计算carry－in的工作量时，考虑每个任务的松弛时间，从而减少 carry－in 实例执行窗口的长度. 该方法下的 carry－in长度最差为

$$L_i^{\mathrm{CI}} = \max(0, (D_k - D_i) - (N_i^{\mathrm{b}} - 1) \times T_i - S_i) .$$

式中 S_i 表示任务 τ_i 的最差松弛时间. 然而该方法依然存在悲观性：首先，通过对窗口和任务的执行观察发现，在可调度的前提下每个任务的完成不晚于其截止期，则问题窗口长度不大于 D_k，因此按照 D_k 分析的松弛时间太过悲观；其次，carry－in 任务实例执行在最坏情况下，carry－in 工作量的估算可以更加精确. 针对这两点的悲观性，下文提出 IMP－G－EDF－RTA 分析方法.

3　IMP－G－EDF－RTA 分析

为了解决上文中的悲观性，首先提出 carry－in 实例工作量的估算方法. carry－in 实例执行窗口定义为从问题窗口开始时刻到该实例完成执行的时间窗口. 然后提出问题窗口内每个任务的最大工作量的计算. 最终提出被分析任务的上界.

表 1　算法 1

Table 1　Algorithm 1

Algorithm 1 $W_i^{\mathrm{out}}(L_i^{\mathrm{out}})$

$W(L_i^{\mathrm{out}}) \leftarrow \sum_{\tau_j \in \tau \setminus \tau_i} W(\tau_j, L_i^{\mathrm{out}})$

$W(\tau_j, L_i^{\mathrm{out}}) \leftarrow \left\lceil \frac{L_i^{\mathrm{out}}}{T_j} \right\rceil \times C_j$

$\mathrm{Left}_{\mathrm{out}} \leftarrow \max\left(0, L_i^{\mathrm{out}} - \left\lfloor \frac{W(L_i^{\mathrm{out}})}{M} \right\rfloor\right)$

if $\mathrm{Left}_{\mathrm{out}} > 0$ **then**
　for each thread $\Theta_{i,u} \in V_i$ **do**
　　$s[u] \leftarrow 0, f[u] \leftarrow s[u] + e(u)$
　end for
　$G_i^{\mathrm{T}} \leftarrow (V_i, E_i^{\mathrm{T}})$
　topolohically sorted the thread in G_i^{T}
　for each thread $\Theta_{i,u} \in V_i$ **do**
　　for each thread $\Theta_{i,v} \in$ Children $[U]$ **do**
　　$s[v] \leftarrow f[u], f[v] \leftarrow s[v] + e(u)$
　　end for
　end for
　for each thread $\Theta_{i,u} \in V_i$ **do**
　　if $f[u] \leqslant \mathrm{Left}_{\mathrm{out}}$ **then**
　　$W_i^{\mathrm{out}} \leftarrow W_i^{\mathrm{out}} + e(u)$
　　else
　　if $s[u] \leqslant L_i^{\mathrm{out}}$ and $f[u] > \mathrm{Left}_{\mathrm{out}}$ **then**
　　$W_i^{\mathrm{out}} \leftarrow W_i^{\mathrm{out}} + \max(0, \mathrm{Left}_{\mathrm{out}} - s[u])$
　　end if
　　end if
　end for
else
　$W_i^{\mathrm{out}} \leftarrow 0$
end if
return W_i^{out}

3.1　Carry－in 工作量估计技术

假设 τ_i 的 carry－in 实例窗口长度为 L_i^{CI}，则从该实例释放到问题窗口开始的长度至少为

$L_i^{\text{out}} = \max(0, T_i - L_i^{\text{CI}} - (T_i - R_i))$,如图 3 所示. 在该窗口内 carry – in 实例的执行受到高优先级任务实例的干涉,每个任务的干涉量最多是

$$W(L_i^{\text{out}}) = \sum_{\tau_j \in \tau \setminus \tau_i} W(\tau_j, L_i^{\text{out}}),$$

$$W(\tau_j, L_i^{\text{out}}) = \left\lfloor \frac{L_i^{\text{out}}}{T_j} \right\rfloor \times C_j.$$

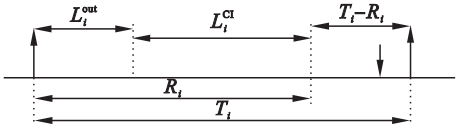

图 3 Carry – in 实例窗口的组成

Fig. 3 Constitution of carry-in window

任务 τ_i 的 carry – in 任务实例在时间窗口 L_i^{out} 内执行的工作量至少为 $W_i^{\text{out}}(L_i^{\text{out}})$,$W_i^{\text{out}}(L_i^{\text{out}})$ 可由算法 1 计算. 算法 1 的基本思路是:在窗口长度为 L_i^{out} 内的 carry – in 实例的工作量根据窗口提供的计算能力减去高优先级任务实例在最坏情况下执行的工作量,则剩余的可用于 carry – in 实例执行,再结合 carry – in 实例节点的结构特性进行估算. 在算法 1 中 $s[u]$ 和 $f[u]$ 分别表示节点 u 的开始执行的时间和完成执行的时间.

引理 2 任务 τ_i 的 carry – in 任务实例,在时间窗口 L_i^{CI} 内执行的工作量最多为 $W_i^{\text{CI}}(L_i^{\text{CI}})$,

$W_i^{\text{CI}}(L_i^{\text{CI}}) = C_i - W_i^{\text{out}}(\max(0, T_i - L_i^{\text{CI}} - (T_i - R_i)))$.

证明 一个任务实例的最大工作量是 C_i,已知在 carry – in 实例窗口开始前至少已经完成了 $W_i^{\text{out}}(\max(0, T_i - L_i^{\text{CI}} - (T_i - R_i)))$,剩下的工作量需要在 carry – in 执行窗口内完成,最多是 $C_i - W_i^{\text{out}}(\max(0, T_i - L_i^{\text{CI}} - (T_i - R_i)))$.

该方法的优越性主要体现在在 L_i^{CI} 长度的时间窗口里不一定都在执行 carry – in 实例. 而该窗口内 carry – in 最晚可以执行到的时刻可以根据该实例承受的最大干涉去估计,从而减少 carry – in 实例的干涉.

3.2 工作量的估算

本节提出一种更加精确的 carry – in 实例执行窗口长度计算方法. 首先,假设任务 τ_k 从释放到完成执行的长度为 X_k. 显而易见,$X_k \leqslant D_k$. 在 τ_k 问题窗口内,任务 τ_i 的 body 实例的计算发生了变化,由于任务 τ_k 在 X_k 内完成了执行,则只有在 X_k 内的工作量才能干涉任务 τ_k 的执行. 任务 τ_i 的 body 任务实例最多为

$$P_i^{\text{b}}(X_k, L_k) = \left\lfloor \frac{\max(0, P_i(X_k, L_k))}{T_i} \right\rfloor,$$

$$P_i(X_k, L_k) = \min(X_k - \text{LC}_i, L_k - D_i).$$

$P_i(X_k, L_k) < 0$ 表示任务 τ_i 没有 body 实例,则 carry – in 长度为 $L_i^{\text{CI}}(X_k, L_k)$,如图 4 所示.

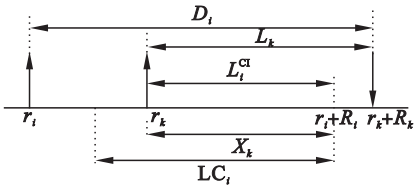

图 4 $P_i(X_k, L_k) < 0$ 情况下,carry – in 实例执行窗口的长度

Fig. 4 Length of carry-in window when $P_i(X_k, L_k) < 0$

而当 $P_i(X_k, L_k) \geqslant 0$ 需要在 $P_i(X_k, L_k)$ 长度上减去 body 实例的周期长度再减去松弛时间,剩余的是 carry – in 的长度 $L_i^{\text{CI}}(X_k, L_k)$,如图 5 所示.

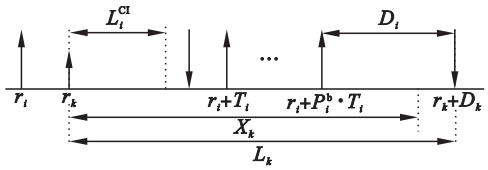

图 5 $P_i(X_k, L_k) \geqslant 0$ 情况下,carry – in 实例执行窗口的长度

Fig. 5 Length of carry-in window when $P_i(X_k, L_k) \geqslant 0$

因此,

$$L_i^{\text{CI}}(X_k, L_k) = \begin{cases} \max(0, \min(X_k, \min(\text{LC}_i, \beta))), \\ \text{if } P_i(X_k, L_k) < 0; \\ \max(0, \min(\text{LC}_i, \max(0, \alpha))), \\ \text{if } P_i(X_k, L_k) \geqslant 0. \end{cases}$$

其中:

$$\beta = L_k - (D_i - R_i);$$

$$\alpha = P_i(X_k, L_k) \bmod T_i - (T_i - R_i).$$

综上,每个任务 τ_i 在被分析任务 τ_k 长度为 L_k 的问题窗口内的工作量为

$$W_{i,k}(X_k, L_k) = C_i \times P_i^{\text{b}}(X_k, L_k) + W_i^{\text{CI}}(L_i^{\text{CI}}(X_k, L_k)).$$

被分析任务 τ_k 除了关键路径上的节点,其他节点的执行会干涉关键路径上节点的执行,总的干涉量之和为

$$W_{k,k}(X_k, L_k) = C_k - \text{LC}_k.$$

该分析方法从任务 τ_k 可能的完成时刻进行分析,避免由于窗口过大造成的干涉量的悲观估计,因此本文的分析结果优于文献[12]的分析方法.

3.3 IMP – G – EDF – RTA 方法

本文分析了任何一个任务 τ_i 在已知被分析任务的问题窗口长度内的工作量. 根据问题窗口的定义 $L_k \leqslant D_k$, 最悲观情况下 $L_k = D_k$. 显然 X_k

的取值满足 $L_k \leqslant X_k \leqslant D_k$.

已知 X_k 和 L_k 的情况下，所有可以干涉任务 τ_k 执行的工作量总和定义为 $\Omega_k(X_k, D_k)$：

$$\Omega_k(X_k, D_k) = \sum_{\tau_i \in \tau} W_{i,k}(X_k, L_k).$$

定理 1　R_k 是式(1)从 $X = \mathrm{LC}_k$ 开始迭代的最小不动解，

$$X = \mathrm{LC}_k + \left\lfloor \frac{\Omega_k(X_k, D_k)}{M} \right\rfloor, \tag{1}$$

则 R_k 是任务 τ_k 的 WCRT 上界.

证明　根据上文的分析，该定理显然正确.

4　实验分析

采用仿真实验，验证当参数不同时本文所提算法的性能. 基于 Win 7 系统下的 python 3.6.5 进行仿真实验.

4.1　任务集的随机生成

本文采用文献[13]提出的 DAG 任务集生成方法随机生成任务集. 每个任务 τ_i 各个参数按照以下方式生成：1) 周期 T_i 采用均匀分布取值，范围为[100,1 000]，截止期 $D_i = T_i$；2) 任务 τ_i 的节点个数也采用均匀分布从[30,40]内取值；3) 1 个 DAG 最多有 $N_i(N_i - 1)/2$ 条边，为了表示随机性，每个 DAG 任务边的数量用参数 $\mathrm{Pr} \times (N_i \times (N_i - 1))/2$ 表示，当 $\mathrm{Pr} = 1$ 时表示 DAG 图是全连接的，当 $\mathrm{Pr} = 0$ 时表示图中没有边；4) 对于 τ_i 每个节点的 WCET 从 $[1, T_i/N_i]$ 范围内随机生成.

基本分析方法表示为 Base－line；最新算法表示为 DAG－G－EDF；本文提出的算法称为 IMP－G－EDF. 通过接受率和提升率来对比三个算法.

接受率(Acc. Ratio)：算法 A 的接受率定义为可以被算法 A 调度的任务集数量和全部被调度的任务集数量的比值.

提升率(Imp. Ratio)：算法 A 和 Base－line 比较的提升率定义为该算法比 Base－line 算法增加的可调度性的平均比率.

4.2　仿真结果

实验 1 对比三个算法的可调度性随利用率变化的趋势. 基本参数设计：图 6 中曲线的每一个点表示在该利用率下算法接受率，而图 7 中的曲线每个点表示该利用率下对应算法的提升率. 利用率本文采用了规范化利用率，即 $U_\mathrm{m} = U/M$. 每个任务的并行度也可用参数 Pr 表示，DAG 任务中边越多，表示节点需要满足的制约关系越多，并行度越低. 因此仿真实验时取 $\mathrm{Pr} \in (0.1, 1]$. 该实验包括两个结果，对应不同的处理器个数的结果.

仿真实验取 $M = 8$，实验步骤如下：

1) 对利用率进行分段，从 0 到 1 取 15 个利用率段，每一个利用率段表示为 $(U_{\min}, U_{\max}]$.

2) 针对每个利用率段，首先生成一个具有两个任务的任务集；该任务集中的利用率范围为 $(U_{\min}, U_{\max}]$；其他参数采用 4.1 节中任务各个参数的生成方法生成.

3) 如果任务集的利用率 $U > U_{\max}$，抛弃该任务集回到 2).

4) 先判定该任务集的可调度性，然后根据调度后的结果进行操作，如果新算法不能调度该任务集则回到 2)，否则增加一个任务回到 3).

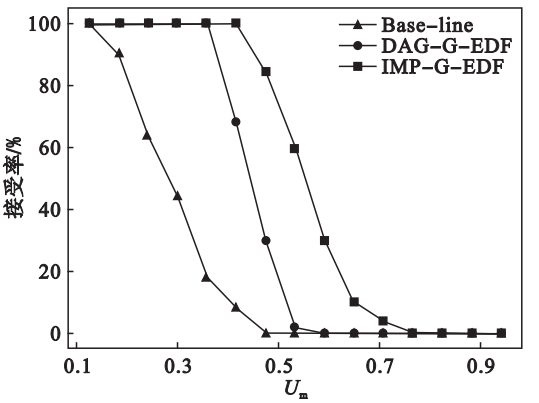

图 6　三个算法的接受率比较

Fig. 6　Acceptance ratio for three algorithms

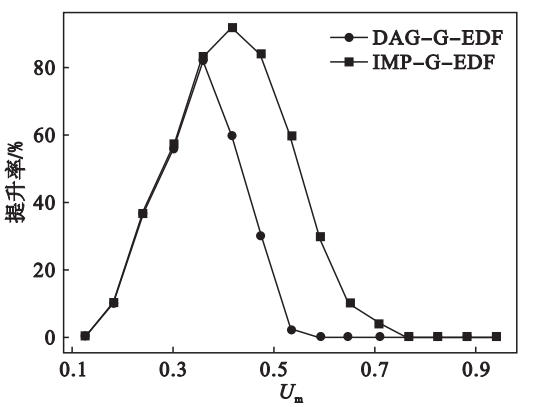

图 7　算法的提升率

Fig. 7　Improvement ratio of algorithms

图 6 的实验结果表明本文提出的算法在接受率层面较 DAG－G－EDF 算法有所提升，且在 U_m 取值为 0.5 时接受率提高了 25%. 图 7 的实验结果表明在 WCRT 上的改进最高可达 30%.

为了观察相同并行度不同处理器个数下的任务可调度性变化的趋势，固定 $\mathrm{Pr} = 0.5$，对处理器个数为[2,16]上任务集利用率为[3.9,4.1]上任务集的可调度性的变化规律进行了实验. 实验中

每个采样点取样 5 000 个任务集,任务的各项参数根据 4.1 节的介绍获取. 实验结果如图 8 所示,在处理器个数比较多的情况下两个算法的可调度性趋于 1,因为任务集的利用率远小于处理器个数. 在处理器个数相对较少时,本文算法的接受率明显高于 DAG – G – EDF 算法和 Base – line 算法,本文所提算法的接受率最多高于 DAG – G – EDF 30% .

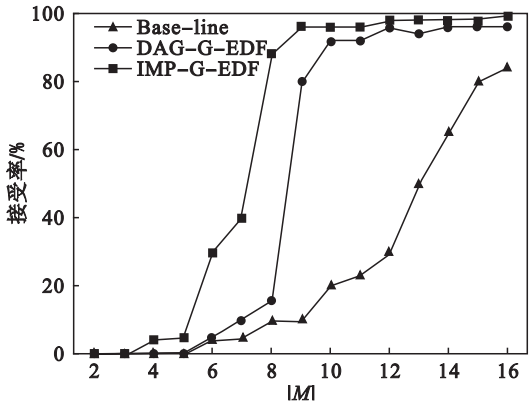

图 8 处理器参数 M 不同时的接受率实验

Fig. 8 The test of acceptance ratio for the varying M

5 结 论

1) 本文分析了 DAG 并行任务模型 G – EDF 可调度性分析的基本现状及悲观性.

2) 结合造成 DAG 并行任务模型 G – EDF 可调度性分析悲观的原因,提出一种新的基于窗口长度的 carry – in 实例工作量估算方法.

3) 结合新的 carry – in 估算方法,提出一种更加精确的问题窗口工作量估算方法.

4) 为了验证本文算法的性能,进行了随机仿真实验,实验结果表明本文提出的算法最高可提升目前最好算法的 25% 的性能.

参考文献:

[1] Intel. Intel Xeon Phi product family [DB/OL]. (2013 – 07 – 03) [2017 – 06 – 20]. http://www. intel. com/content/www/us/en/processors/xeon/xeon-phi-detail. html,2018.

[2] Lakshmanan K, Kato S, Rajkumar R. Scheduling parallel real-time tasks on multi-core processors[C]// IEEE Real-Time Systems Symposium. San Diego: IEEE Computer Society, 2010:259 – 268.

[3] Saifullah A, Agrawal K, Lu C, et al. Multi-core real-time scheduling for generalized parallel task models[C]// Real-Time Systems Symposium. San Juan: IEEE, 2012:217 – 226.

[4] Chwa H S, Lee J, Phan K M, et al. Global EDF schedulability analysis for synchronous parallel tasks on multicore platforms [C]// Euromicro Conference on Real-Time Systems. Paris: IEEE Computer Society, 2013:25 – 34.

[5] Cláudio M, Bertogna M, Pinho L M. Response-time analysis of synchronous parallel tasks in multiprocessor systems[C]// International Conference on Real-Time Networks and Systems. Versailles: ACM, 2014:3 – 12.

[6] Nelissen G, Berten V, Milojevic D. Techniques optimizing the number of processors to schedule multi-threaded tasks [C]// Euromicro Conference on Real-Time Systems. Pisa: IEEE Computer Society, 2012:321 – 330.

[7] Axer P, Quinton S, Neukirchner M, et al. Response-time analysis of parallel fork-join workloads with real-time constraints [C]// Euromicro Conference on Real-Time Systems. Paris: IEEE Computer Society, 2013:215 – 224.

[8] Li J, Agrawal K, Lu C, et al. Analysis of global EDF for parallel tasks [C]// Euromicro Conference on Real-Time Systems. Paris: IEEE Computer Society, 2013:3 – 13.

[9] Bonifaci V, Marchetti-Spaccamela A, Stiller S, et al. Feasibility analysis in the sporadic DAG task model[C]// Real-Time Systems. Vancouver: IEEE, 2013:225 – 233.

[10] Baruah S, Bonifaci V, Marchettispaccamela A, et al. A for recurrent real-time processes [C]// Real-Time Systems Symposium. San Juan: IEEE Computer Society, 2012: 63 – 72.

[11] Li J, Chen J J, Agrawal K, et al. Analysis of federated and global scheduling for parallel real-time tasks [C]// Euromicro Conference on Real-Time Systems. Madrid: IEEE Computer Society, 2014:85 – 96.

[12] Chwa H S, Lee J, Lee J, et al. Global EDF schedulability analysis for parallel tasks on multi-core platforms[J]. *IEEE Transactions on Parallel & Distributed Systems*, 2017, 28(5):1331 – 1345.

[13] Cordeiro D, Mounié G, Perarnau S, et al. Random graph generation for scheduling simulations[C]//Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques. Torremolinos: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010:60 – 71.