一种面向节能的星载实时任务动态调度算法研究

李延祺[1],任 海[2],白 亮[2],邱 源[2],张凤源[2],牛建伟[1],李辉勇[1]
(1.北京航空航天大学 计算机学院,北京 100191; 2. 上海航天电子技术研究所,上海 200082)

摘 要:星载嵌入式操作系统性能和任务能耗之间的平衡非常重要,采用合理的处理器和电压分配策略是一个重要手段。提出一系列基于计算概率的建模方法,用来解决星载实时嵌入式系统中对于具有数据依赖的非周期性任务的处理器和电压分配相关问题,并且能够确保所有的任务都能在时间约束下执行。首先设计一个处理器调度算法,将任务的有向无环图(DAG)映射到一组可变电压处理器上,然后使用动态编程算法为每个任务分配合适的电压。通过带有重启的局部搜索策略从候选解集中获取最优解,以解决局部最优陷阱问题。通过实验说明,所提出的算法与其他算法相比,在所有时间约束下具有更好的能效。

关键词:概率调度; 实时嵌入式系统; 节能; 实时任务调度; 星载系统
中图分类号:TP 316.2;TP 309 **文献标志码**:A **DOI**:10.19328/j.cnki.1006-1630.2019.03.012

An Energy-Saving Oriented Dynamic Scheduling Algorithm for Space-Borne Real-Time Tasks

LI Yanqi[1], REN Hai[2], BAI Liang[2], QIU Yuan[2], ZHANG Fengyuan[2], NIU Jianwei[1], LI Huiyong[1]
(1. Department of Computer Science and Engineering, Beihang University, Beijing 100191, China;
2. Shanghai Aerospace Electronic Technology Institute, Shanghai 200082, China)

Abstract: The trade-off between system performance and energy efficiency (service time) is critical for the battery-based aerospace real-time embedded system. By adopting a probabilistic approach, this paper proposes a model and a set of algorithms to address the processor and voltage assignment with probability (PVAP) problem of data-dependent aperiodic tasks in real-time embedded systems, ensuring that all the tasks can be done under the time constraint with a guaranteed probability. A task directed acyclic graph (DAG) is adopted to model the PVAP problem. This paper first proposes a task-scheduling algorithm to map the task DAG onto a set of voltage-variable processors, and then uses the dynamic programming algorithm to assign a proper voltage to each task. Finally, to escape from local optima, a local search with restarts searches the optimal solution from candidate solutions by updating the objective function, until the task ends or the task deadline is reached. The test results show that compared with other algorithms, this algorithm has higher energy efficiency under all the time constraints.

Keywords: probabilistic scheduling; real-time embedded system; energy saving; real-time task scheduling; space-borne system

0 引言

由于星载嵌入式系统在独立的太空环境中工作,所以航天设备需要的电量资源必须能够自给自足。为了延长设备的工作时间,需要在操作系统层面针对各任务进行合理的调度,以统筹各任务的能耗。对于功能强大的嵌入式处理器来说,合理的分配资源以降低整体能耗是十分重要的。因此,星载实时操作系统的性能和能耗之间的平衡问题是使用

收稿日期:2018-09-13;**修回日期**:2018-11-20
基金项目:上海航天科技创新基金项目(SAST2016097,SAST2017107)
作者简介:李延祺(1995—),男,硕士研究生,主要研究方向为嵌入式系统。
通信作者:李辉勇(1983—),男,博士,实验师,主要研究方向为嵌入式系统。

电池的嵌入式系统所必须面对的重要课题。

为了提高嵌入式系统对电池资源的利用率，满足任务执行的时间约束以及保证系统的实时性，目前国内外大多采用实时电压和频率缩放(DVS)技术。DVS 技术可以让嵌入式设备动态适配 CPU 的当前电压来达到理想中的低能耗状，目前多数嵌入式和分布式系统都采用 DVS 技术以降低系统能耗。DVS 技术的研究主要从具有严格时间约束的实时独立任务方面进行，以达到显著降低任务整体能耗的目的。如文献[1]采用一种应用在单核处理器的嵌入式系统中的伪多项式动态规划算法，以求解具有离散电压值的周期性实时任务的最优解。文献[2]通过重复利用时间以允许其余任务可以在低于标准电压的情况下运行。

为了能够在满足时间约束的前提下提高整个系统的能效，可以从基于概率的嵌入式系统任务调度方向进行研究。文献[3]首先考虑具有不确定性的执行时间的任务，然后引入概率轮转调度算法来缩短具有无环拓扑架构任务图的执行时间。文献[4]采用调度算法将任务动态地重新映射到适当的处理器中，并设定处理器内部执行任务的顺序，将多余的时间分配给未执行的任务。文献[5]提出一种不限制嵌入式系统处理器数量的动态规划算法来解决异构嵌入式系统的任务分配问题。

本文采用一种局部重启搜索算法(LSR)以找到嵌入式系统的最佳调度策略。首先，设计一个有效的处理器分配算法来生成处理器分配方案；然后使用电压分配算法，以一定概率下完成任务的思路来最小化整体的能耗；最后，为了避免局部最优解，使用带有重新启动的本地搜索再次对资源进行优化并得到全局最优解。本文采用的 LSR 算法和传统调度算法能够为软实时嵌入式系统提供一些满足时序和置信度要求的更低成本解决方案，以动态调度的思想来解决专属虚拟接入点(PVAP)问题。

1 问题建模

由于条件指令和时变外部环境(如波动的网络带宽、不同的用户输入和 DVS)的变化，星载实时嵌入式操作系统的同一任务会在不同环境下产生不同的执行时间。即，一个任务在不同的环境下可能会有不同的完成概率。

令 i 为任务编号 ID，$r(i)$ 是任务 i 的执行时间，$ct(i)$ 是关于任务 i 的本地处理器计算时间。$c_{i,r(i)}$ 和 $p_{i,r(i)}$ 分别是任务 i 在 $r(i)$ 时间内完成时的能耗和完成任务的概率。$c_{ict(i)}$，$p_{ict(i)}$ 分别是任务 i 在 $ct(i)$ 时间内完成的能耗和概率。pe_m^n (m 是处理器的 ID，n 是电压 ID)表示处理器 pe_m 运行在 vol_n (电压 ID 为 n 时的电压值)电压下。

定义 B_i 为一个三元组链表 $(c_{i,r(i)}, p_{i,r(i)}, pe_m^n)$，即 B_i 是一个存放任务 i 信息的链表。三元组链表 $B_i=(c_{i,r(i)}, p_{i,r(i)}, pe_m^n)$ 表示：当任务 i 在电压 vol_n 选择处理器 pe_m^n 时，任务 i 将会在 $r(i)$ 时间内完成，能耗为 $c_{i,r(i)}$，置信度为 $p_{i,r(i)}$；$B=\{B_1, \cdots, B_i, \cdots, B_x\}$，$x$ 代表的任务数量，B 是 B_i 的集合。

定义 $T_{pe_m^n}(v)$ 为每个节点 $v \in V$ 在处理器上 pe_m^n 以电压 vol_n 运行的时间。首先获得不同电压时的处理器功率[6]，然后通过建立表格，统计分析得到每个任务执行时间的概率密度函数[6]。任务执行的能耗为处理器功率和执行时间相乘的结果，建立 B_i 链表。

以下符号用于问题的数学表述中：DAG $G=\langle V, E\rangle$ 用于表示所有任务及其数据的依赖关系，$V=\{v_1, \cdots, v_i, \cdots, v_x\}$ 是任务节点集合，x 是任务数量。$E=\{e_{11}, \cdots, e_{ij}, \cdots, e_{mn}\}$ 是边的集合，当 e_{ij} 为 1 时，表示当前 v_i 和 v_j 节点之间存在数据依赖，反之为 0 无依赖。

关于嵌入式处理器的定义如下：$PE=\{pe_1, \cdots, pe_i, \cdots, pe_y\}$ 是处理器单元的集合，y 是处理器数量。pe_i 表示第 i 个处理器。$Vol=\{Vol_1, \cdots, Vol_i, \cdots, Vol_M\}$，$Vol_i=\{Vol_{i,l}, \cdots, Vol_{i,j}, \cdots, Vol_{i,L}\}$ 是 pe_i 电压的集合，L 是处理器的可用电压值。

接下来讨论如何来计算 DAG 的执行完成的概率。DAG 的执行概率是任务图中各节点执行概率的总集合。假设执行概率 $b(c_{i,r(i)}, p_{i,r(i)}, pe_m^n) \in B_i$ 是任务 v_i 选择的，s_i 是任务 v_i 的起始时间，则全部执行时间 $T_A(G)$、能耗 $C_A(G)$、置信度 $P_A(G)$，在给定 DAG G 和处理器且电压假定为 A 情况下可以进行如下计算，即

$$T_A(G)=\max\{s_i+r(i), s_i+ct(i)\},\quad \forall v_i \in V \tag{1}$$

$$C_A(G)=\min\left(\sum_{i=1}^{N} c_{i,r(i)}, \sum_{i=1}^{N} c_{i,ct(i)}\right) \tag{2}$$

$$P_A(G)=\min\left(\prod_{i=1}^{N} p_{i,r(i)}, \prod_{i=1}^{N} p_{i,ct(i)}\right) \tag{3}$$

由于任务 v_i 取决于前置的数据输出，当且仅当只有

前置任务完成后才能运行。

建模之后,给出 PVAP 问题的定义:给定一个有限的处理器集合 PE,一个电压电平集合 Vol,一个执行概率集合 B,有向无环图 DAG $G=\langle V,E\rangle$,一个时间约束 T 和一个置信度 P,问题是要为每个任务 v_i 确定一个合适的处理器和电压电平,其提供了在时间约束 T 下的最小能量消耗和具有保证置信概率 P 的优先约束。

2　PVAP 问题的解决方案

提出了一种面向星载嵌入式系统,具有数据依赖和 CPU 资源约束的非周期性任务图的可变电压概率调度的解决方案。解决方案分为三个阶段:第一阶段,使用有效的调度算法以获得初始处理器的分配方案;第二阶段,基于已得到的分配策略,提出电压分配算法,以保证任务能在一定的置信度和时间约束下执行完成,同时对能耗较高的任务进行优化分配,并将能耗最小化;第三阶段,使用 LSR 算法来摆脱局部最优解。LSR 算法通过更新目标函数从候选解决方案中搜索最优解,直到任务结束或者超出时间约束,这些都将在下文中对其分别讨论。

2.1　初始处理器分配策略

由于嵌入式系统中处理器的数量有限,研究时需要为每个任务分配合适的处理器,以有效使用处理器。为了避免局部最优解,本文使用不同的调度算法,如基于尽可能晚(ALAP)调度算法[7]、基于尽可能快(ASAP)调度算法和整数线性规划(ILP)调度算法[8]来生成原始的处理器分配策略。

处理器调度算法(ASAP/ALAP)的实现过程如下:首先使用拓扑排序获得目标任务列表具有优先级限制的 DAG 拓扑序列;然后计算每个任务的最差执行时间,在最早/最近状态下,使用 ASAP/ALAP算法根据任务的优先级进行调度,通过为同一处理器上的连续任务间插入模拟边界来保证任务的执行顺序;最后使用 ASAP/ALAP 算法将任务图中的每个任务分配给适当的处理器,就可以生成用于解决 PVAP 问题的新 DAG。

ILP 算法的实现如下:首先计算每个任务的执行时间;然后采用整数线性规划(ILP)调度算法[8]完成 ILP 的 PVAP 问题的数学表达式;最后使用 LINGO 9.0 软件的非线性规划解算器来获得处理器分配策略。

2.2　电压分配方案

该节讨论如何使用动态编程(DP)来解决 PVAP 中电压分配问题。

假设 G_i 是以节点 v_i 为根的图 G 的子图,包含节点 v_i 可以到达的所有节点。G'_i 是以节点 v_i 为根的子图,包含除节点 v_i 外可以到达的所有节点。$T_A(G_i)$,$C_A(G_i)$ 是 G_i 指定 A 下的执行时间和全部能耗,$P_A(G_i)$ 是 G_i 在 A 的假设下以时间 $T_A(G_i)$ 完成的概率。S_i 是关于 $(c_{i,j},p_{i,j})$($c_{i,j}$ 和 $p_{i,j}$ 分别代表能耗、概率)的链表,表示 G_i 的 PVAP 问题的解决方案,按照概率的升序排序。和 S_i 类似,S'_i 也是 G'_i 的 PVAP 问题的解决空间,S'_i 是 $s'_{i,j}$ $(c_{i,j},p_{i,j})$($c_{i,j}$ 和 $p_{i,j}$ 分别代表能耗、概率)的链表。这样,PVAP 问题就变成根据给定图 G 求解 S_i 的问题。例如:为了得到如图 1(a)所示任务树中根节点的解决方案 S_0,必须先得到其 2 个子图 G_1 和 G_2,因此,就要将 2 个子图的解决方案结合,如下:

1)令 $K=\{S_1,S_2\}$,式中,S_1 和 S_2 分别是图 G_1 和 G_2 的解空间;

2) $S'_0=\mathrm{CombineSubSolution}(K)$;

3)$S_0=S'_0\odot B_0$;

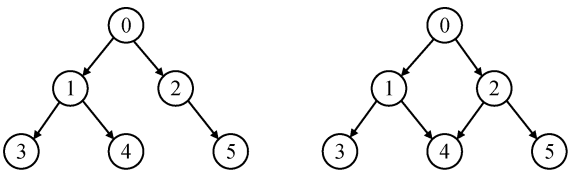
4) 删除 S_0 中的冗余解和不可行解。

在以上步骤中,通常采用组合子集解(CSS)算法结合多个子图的最优规划来估计,首先计算 S'_i 的子节点 $s'_{i,j}(c_{i,j},p_{i,j})$,其中 $p'_{i,j}=p_{i,j_1}\cdot p_{i_k,j_2}$,$c'_{i,j}=c_{i,j_1}\cdot c_{i_k,j_2}$,得出 $S'=S'\cup\{s'_{i,j}\}$;然后删除其中的不可行解和冗余解,便可得到所求的解决方案。

在步骤 3 中,通过引入运算符"⊙"来处理从 S'_i 到 S_i 的转换。S'_i 和 B_i 之间的"⊙"运算符定义如下:给定 S'_i 和 B_i,在两个对象之间使用"⊙"后,对于每个 $s'_{i,j_1}=(c_{i,j_1},p_{i,j_1})\in S'_i$ 和 $b(c_{i,r(i)},p_{i,r(i)},pe^n_m)\in B_i$,都能得到 $s_{i,j}=(c_{i,j},p_{i,j})$,其中 $p_{i,j}=p_{i,j_1}\cdot p_{i,j_2}$,$c_{i,j}=c_{i,j_1}\cdot c_{i,j_2}$,且 $j=j_1+j_2$。步骤 4 是删除 S_0 中的冗余和不可行的解决方案。

根据上述思想,按照自下向上的顺序计算出图 1(a)任务树中各个节点 i 的解决方案。当算法到达根节点 0 时,需计算出 S_0 所需要的全部结果,并重复使用子图的解来计算。然后从最小的子图开始,逐步解决大规模的问题。在此计算过程中应保存已有子图的解决方案,通过重复使用子图来提高问题的解决速度。整个图的 PVAP 问题可以分解为多个子图的 PVAP 问题。通过去除 S_i 中的冗余和不可行解,可获得满足时间和概率约束的候选解,

并将具有最小能耗的解视为整棵树的最优解。共同节点问题如图 1 所示。

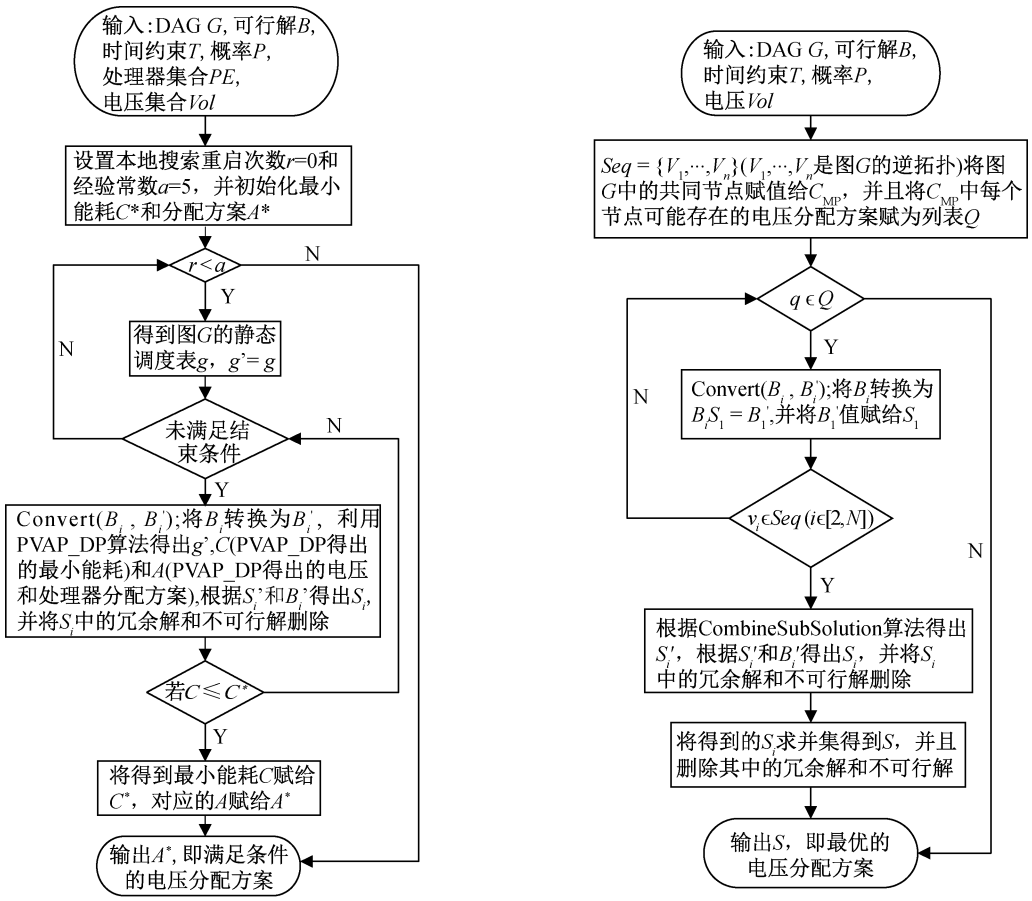


(a) 具有最优子结构的PVAP任务树　(b) 具有共同节点的任务树

图 1　共同节点问题

Fig.1　Common node problem

然而对于具有任意数量边和节点的 DAG，由于其中可能存在公共节点（出现在 2 个或更多子图中的节点），所以在电压选择过程中会产生冲突。例如在图 1(b)中，v_4 节点属于 2 个子图 G_1 和 G_2，可能存在 V_4 在 S_1 中为 v_4 优先选择 vol_0，而在 S_2 中选择。因此即使 G_1 和 G_2 的解决方案（S_1 和 S_2）都是最佳的，但是由于电压的选择会发生冲突，所以 S_0 不能获得最佳的分配方案。下面列举所有节点中可能存在的组合方案来解决此问题，在所有可能的组合中，PVAP_DP 算法能够选择最佳的电压分配方案。算法流程如图 2 所示。



(a) PVAP_DP算法流程图　(b) PVAP_LSR算法流程图

图 2　算法流程图

Fig.2　Algorithm flowchart

采用 PVAP_DP 算法来解决电压分配问题，如图 2(a)所示。首先要解决子问题，得到逆拓扑排序。之后为了解决公共节点的问题，需要列举多个节点中可能存在的电压分配方案。由于从逆拓扑顺序的第 1 个节点（假设为 v_1）没有子节点，所以可行解 S_1 就是概率执行列表 B_1。

为了得到 S_i，第一步首先需要通过 G_i，G_{i_1}，…，G_{i_w} 的解来得到 S'_i，然后对于每个 $S'_{i,j} \in S'_i$ 和 $b = B_i$，对 $s'_{i,j}$ 和 b 进行⊙操作从而得到 S_i。当内层的循环结束时，得到所有节点的最优电压分配 S_N。最后合并每个循环中产生的最佳电压分配方案 S_N。当外层循环结束时，在当前的处理器选择

下获得所有节点的最优电压分配方案。当外循环结束后,从集合 S 中去除冗余解和不可行解,得到最优解。

2.3 LSR 局部重启搜索策略

采用局部重启搜索算法(LSR)在所有候选解中寻找最优解。图 2(b)中给出了 LSR 算法流程图。首先使用有效调度算法(如 ALAP)生成初始处理器调度(DAG);然后基于新生成的 DAG,在该步骤中采用带有局部搜索的 PVAP_DP 算法来最小化能耗;最后从初始调度开始,在内部循环中,采用局部搜索算法在候选解的解空间中寻找更优的处理器和电压分配。为了避免局部最优,在外循环中应用 LSR。使用不同的处理器调度策略(本例中使用 PVAP_DP 算法)来生成初始处理器分配方案,以便对不同的初始调度表采用 LSR。当达到重启次数时终止循环。重启次数是一个经验常数,取决于不同的应用场景。

3 实验

依据参考文献[9]进行实验设计,并且利用实验结果对 PVAP_LSR 算法进行性能评估[10]。ASAP 和 ALAP 调度算法广泛应用于各种处理器和电压分配算法中,本文实验中使用的基于 ASAP/ALAP 的调度算法是 ASAP/ALAP 和 PVAP_DP 的结合,而 PVAP_LSR 算法是 LSR 和 PVAP_DP 算法的结合,通过这种方式能够提升 LSR 算法的性能。本文还对 PVAP_LSR 算法和其他搜索算法,如禁忌搜索(TS)和模拟退火算法(SA)进行比较。

3.1 实验基准

针对表 1 中 2 个基准进行了对比实验,使用随机化任务图生成器[7]得到任务图,TGFF-1 具有很长的关键路径,TGFF-2 具有相对较短的关键路径。

表 1　实验任务图的基准描述

Tab.1　Benchmark description of experimental task diagram

基准	任务数	边界	宽度	共同节点
TGFF-1	43	47	5	0
TGFF-2	35	41	5	4

首先针对估计不同频率下的 CPU 功率进行分析。本文采用 PowerPC 处理器和 ARM 处理器。处理器的电压以及各自的功率与频率见表 2。

表 2　ARM 和 PowerPC 处理器的电压和功耗

Tab.2　Voltage and power consumption of ARM and PowerPC processor

处理器(单位)	电压/V	频率/MHz	功率/MHz
ARM	1.0	800	5.0
	1.1	1 000	9.8
	1.2	1 200	17.0
PowerPC	0.9	900	9.0
	1.0	1 200	14.6
	1.1	1 500	18.0

在表 3～5、图 3～4 中,能源消耗的单位是能源单位(EU),时间约束或者任务执行时间的单位是时间单位(TU),列“TC/PEs”表示“时间约束/处理器数量”。PVAP_LSR 算法的平均提升效果显示在每个表的最后一行。实验中可以灵活的选择要部署的处理器数量(2 个或者 3 个)。前者意味着所有任务都在 2 个处理器(PowerPC 和 ARM 处理器)上执行,后者意味着所有任务在 2 个 PowerPC 处理器和 1 个 ARM 处理器上执行。

3.2 与 ILP 算法的比较

将本文中使用的算法和 ILP 进行了比较。对使用 TGFF-1 标准的实验结果分别见表 3,4。在表 3,4 中,“ILP1”“ILP2”和“PVAP_LSR”三列分别表示不含 DVS 的 ILP、具有 DVS 的 ILP 和 PVAP_LSR 算法获得的结果。“%I1”和“%I2”两列分别代表本文中使用的算法相对于不具有 DVS 的 ILP 算法和具有 DVS 的 ILP 算法的改进。在本文中,所有表中带有“×”的条目表示相应的分配未能生成有效的调度。

实验结果表明:PVAP_LSR 算法在保证置信概率的同时,能够提高能效。表 3 中 PVAP_LSR 算法与 ILP2 算法相比,在置信概率分别为 0.8,0.9 和 1.0 时,平均(最大)能耗降低分别为 18.4%(23.7%),13.3%(17.4%)和 9.8%(14.1%);与 ILP1 算法相比,分别获得 37.2%(41.4%),33.8%(38.5%)和 31.2%(36.0%)的平均(最大)能耗降低,置信概率分别为 0.8,0.9 和 1.0。表 4 中 PVAP_LSR 算法与 ILP2 算法相比,平均(最大)能

表 3 有 ILP 的 DVS、无 ILP 的 DVS 和 PVAP_LSR 的能耗比较

Tab.3 Energy consumption of DVS with ILP, DVS without ILP and PVAP_LSR

TC/Pes	ILP1	ILP2	PVAP_LSR								
			0.8			0.9			1.0		
	能耗	能耗	能耗	%I1	%I2	能耗	%I1	%I2	能耗	%I1	%I2
750/2	21 520	17 934	15 140	25.1	15.6	16 119	25.1	10.1	16 107	25.2	10.2
800/2	20 440	15 509	12 702	37.9	18.1	13 767	32.6	11.2	13 767	32.6	11.2
850/2	20 108	14 107	12 109	39.8	14.8	12 518	37.7	11.9	13 412	33.8	6.3
900/2	19 103	13 560	11 745	38.5	13.4	12 103	36.6	10.7	13 097	32.5	4.9
600/3	22 987	17 112	13 796	40.0	19.4	14 138	38.5	17.4	14 701	36.0	14.1
700/3	20 671	15 861	12 105	41.4	23.7	13 502	34.7	14.9	13 665	33.9	13.8
800/3	18 972	14 703	11 607	38.8	21.1	12 570	33.7	14.5	13 148	30.7	10.6
900/3	17 209	13 896	10 937	36.4	21.3	11 764	31.6	15.3	12 907	25.0	7.1
平均提升			—	37.2	18.4	—	33.8	13.3	—	31.2	9.8

表 4 有 ILP 的 DVS、无 ILP 的 DVS 和 PVAP_LSR 的能耗比较

Tab.4 Energy consumption of DVS with ILP, DVS without ILP and PVAP_LSR

TC/Pes	ILP1	ILP2	PVAP_LSR								
			0.8			0.9			1.0		
	能耗	能耗	能耗	%I1	%I2	能耗	%I1	%I2	能耗	%I1	%I2
320/2	8 410	8 002	6 611	21.4	17.4	7 112	15.4	11.1	7 903	6.0	1.2
360/2	8 170	7 159	5 907	27.7	17.5	6 501	20.4	9.2	7 213	11.7	−0.8
400/2	8 155	6 638	5 268	35.4	20.6	5 829	28.5	12.2	6 377	21.8	3.9
440/2	8 048	5 990	4 746	41.0	20.8	5 047	37.3	15.7	5 773	28.3	3.6
320/3	8 277	7 831	6 616	20.1	15.5	7 317	11.6	6.6	7 715	6.8	1.5
360/3	7 912	7 212	6 093	23.0	15.5	6 618	16.4	8.2	7 303	7.7	−1.3
400/3	7 635	6 901	5 701	25.3	17.4	6 001	21.4	13.0	6 589	13.7	4.5
440/3	7 323	6 008	5 138	29.8	14.5	5 389	26.4	10.3	6 147	16.1	−2.3
平均提升			—	28.0	17.4	—	22.2	10.8	—	14.0	1.3

耗降低分别为 17.4%(20.8%),10.8%(15.7%)和 1.3%(4.5%),置信概率分别为 0.8,0.9 和 1.0;与 ILP1 算法相比,平均(最大)能耗降低分别为 28.0%(41.0%),22.2%(37.3%)和 31.2%(36.0%),置信概率分别为 0.8,0.9 和 1.0。

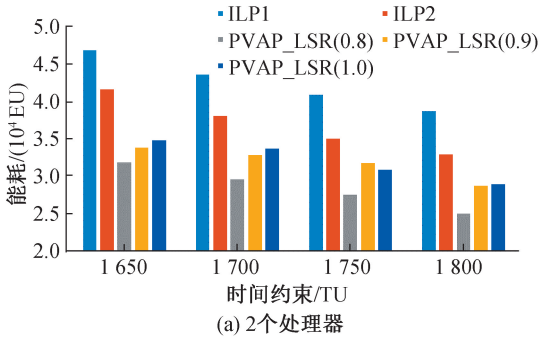
对于 TGFF-1 基准,图 3 显示算法相对于 ILP1 和 ILP2 的改进。与 ILP1 和 ILP2 相比,PVAP_LSR 算法在所有时间约束下能够实现更好的能效,横坐标为人为设定的时间约束,纵坐标为调度算法执行完的任务总能耗。ILP 算法在一定的时间范围内可能无法找到近似最优解。

3.3 与基于 ALAP 和 ASAP 的算法比较

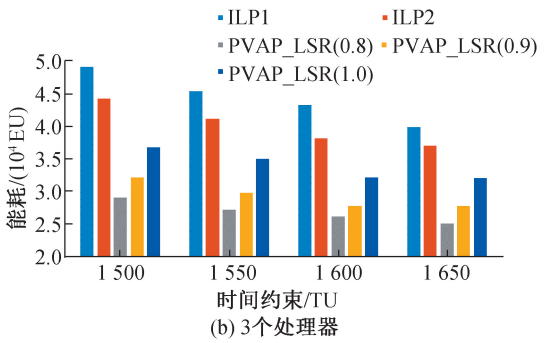
将本文使用的算法与其他 2 种算法(ALAP,ASAP)进行比较。ASAP 的使用方法为:对于处理器分配,使用基于 ASAP 的列表调度;对于电压分配,使用 PVAP_DP 算法。ALAP 的使用方法为:对于处理器分配,使用基于 ALAP 的列表调度;对于电压分配,使用 PVAP_DP 算法。

基于 TGFF-2 标准的实验结果见表 5。表中“A1”“A2”和“PVAP_LSR”分别表示由 ASAP,ALAP 和 PVAP_LSR 算法的结果,“%A1”和“%A2”分别表示相对于 ASAP 和 ALAP 算法的改进效果。

图 4 是 PVAP_LSR 相对于 TGFF-1 基准的 ASAP 和 ALAP 调度的改进。在对于 TGFF-1 基准,PVAP_LSR 在功耗方面可以获得最佳的可变电压调度。与基于 ALAP 的调度相比,PVAP_LSR 相对于基于 ALAP 的调度分别实现 15.6%(23.0%),13.9%(19.2%)和 11.5%(17.4%)的平均(最大)能耗降低,置信概率分别为 0.8,0.9 和 1.0)。PVAP_LSR 算法与基于 ASAP 算法相比,PVAP_LSR 分别获得 15.2%(24.3%),13.4%(20.6%)和 11.0%(18.8%)的平均(最大)功率优化,置信概率分别为 0.8,0.9 和 1.0。而且 PVAP_LSR 算法在所有时间约束下都具有更好的能效。



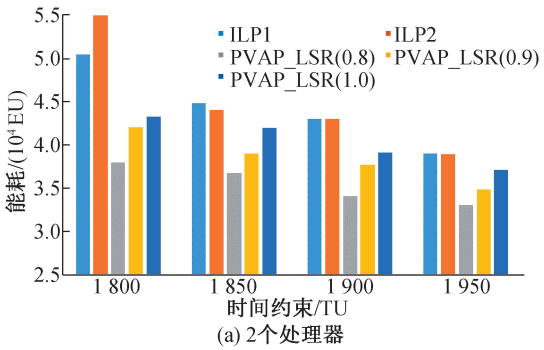
(a) 2个处理器



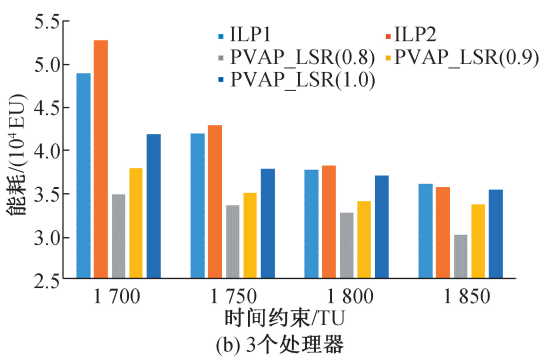
(b) 3个处理器

图 3　ILP1, ILP2 和 PVAP_LSR 基于 TGEF-1 基准的能耗

Fig.3　Benchmark energy consumption of ILP1, ILP2 and PVAP_LSR based on TGEF-1



(a) 2个处理器



(b) 3个处理器

图 4　ALAP, ASAP 和 PVAP_LSR 基于 TGEF-1 基准能耗

Fig.4　Benchmark energy consumption of ALAP, ASAP and PVAP_LSR based on TGEF-1

表 5　ASAP, ALAP 和 PVAP_LSR 基于 TGFF-2 基准的能耗对比

Tab.5　Benchmark energy consumption of ASAP, ALAP and PVAP_LSR based on TGFF-2

TC/(Pes)	ASAP	ALAP	PVAP_LSR								
			0.8			0.9			1.0		
	能耗	能耗	能耗	%A1	%A2	能耗	%A1	%A2	能耗	%A1	%A2
1100/2	×	×	×	×	×	×	×	×	27 380	×	×
1150/2	26 440	25 934	22 330	15.5	13.9	23 074	12.7	11.0	23 141	12.5	10.8
1200/2	25 079	24 870	21 019	16.2	15.5	21 197	15.5	14.8	22 516	10.2	9.5
1250/2	23 158	23 715	19 674	15.0	17.0	20 209	12.7	14.8	20 951	9.5	11.7
1000/3	24 901	25 327	19 173	23.0	24.3	20 117	19.2	20.6	20 574	17.4	18.8
1050/3	23 887	23 597	22 013	7.8	6.7	21 774	8.85	7.7	21 989	7.9	6.8
1100/3	23 712	23 106	19 994	15.7	13.5	20 512	13.5	11.2	21 174	10.7	8.4
1150/3	23 107	22 847	19 375	16.2	15.2	19 763	14.5	13.5	20 338	12.0	11.0
平均提升			—	15.6	15.2	—	13.9	13.4	—	11.5	11.0

4 结束语

现有的研究对实时嵌入式系统的执行时间和能耗等不确定性因素的关注度不够。针对资源有限的星载实时嵌入式系统,本文采用概率方法,提出一套处理器和电压分配算法来解决具有优先级约束的非周期任务的可变电压调度问题。实验结果表明:与目前流行的方法相比,该方法能够提高星载嵌入式操作系统的能效,为用户实现能效提供更多选择,同时在保证置信度的前提下满足定时约束。本方法对于软实时应用程序较为有效。

参考文献

[1] ZHONG X, XU C Z. System-wide energy minimization for real-time tasks: lower bound and approximation[J]. ACM Transactions on Embedded Computing Systems, 2008, 7(3): 1-24.

[2] ZHU D, MELHEM R, CHILDERS B. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems[J]. IEEE Transation on Parallel and Distributed Systems, 2002, 14(7): 84-89.

[3] TONGSIMA S, SHA E H M, CHANTRAPORNCHAI C, et al. Probabilistic loop scheduling for applications with uncertain execution time[J]. IEEE Transactions on Computers, 2000, 49(1): 65-80.

[4] KANG J, RANKA S. Energy-efficient dynamic scheduling on parallel machines[C]//International Conference on High Performance Computing. Springer-Verlag, 2008.

[5] QIU M K, SHA E H M. Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems[J]. ACM Transactions on Design Automation of Electronic Systems, 2009, 14(2): 1-30.

[6] AHMED A, WOLF W, HELLESTRAND G. Statistical characterization of execution time through simulation [C]//International Workshop on Intelligent Solutions in Embedded Systems. IEEE, 2008: 1-13.

[7] GAJSKI D D, ABDI S, GERSTLAUER A, et al. Embedded system design: modeling, synthesis and verification[M]. New York: Springer, 2009: 101-120.

[8] CONG J, GURURAJ K. Energy efficient multiprocessor task scheduling under input-dependent variation[J]. Design, Automation & Test in Europe Conference & Exhibition, 2009: 411-416.

[9] 李立,朱野,赵灵峰,等. 卫星能源约束检查模型改进及仿真[J].上海航天, 2018, 35(5): 116-122.

[10] 张甜甜,陈龙,袁卫文,等. Ka 波段数字信道化体制宽带通信卫星链路预算[J].上海航天, 2017, 34(6): 50-57.

(本文编辑:姚麒伟)