基于纳什均衡的AUTOSAR任务到多核ECU的映射方法

冉 正 罗 蕾 晏 华 李 允
(电子科技大学计算机科学与工程学院 成都 611731)

摘 要 随着汽车电子应用程序对处理器性能需求的不断提高,现代汽车电子系统中的电子控制单元(ECU)已升级为多核结构。多核ECU中的AUTOSAR应用程序的设计、实现和集成将面临新的挑战。其中一个重要的挑战是在映射任务到多核ECU的同时确保系统的实时性能。且在AUTOSAR静态配置过程中,实时系统的资源限制和调度分析使问题变得更加复杂。因此,文中提出了一种基于纳什均衡的AUTOSAR任务到多核ECU的映射方法。该方法将任务优先级应用于博弈过程中,对提高任务映射过程的效率具有非常重要的实用价值。最后,将所提方法应用于AUTOSAR标准的实例中。实验结果表明,所提方法在减少各个任务中可运行实体的最坏响应时间方面具有良好的表现。

关键词 AUTOSAR,纳什均衡,任务映射,多核ECU
中图法分类号 TP37 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.06.029

Nash Equilibrium Based Method for Mapping AUTOSAR Tasks to Multicore ECU

RAN Zheng LUO Lei YAN Hua LI Yun
(School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China)

Abstract With the growing requirements of automotive applications on processing power, the electronic control units (ECUs) in modern automotive system escalates to multicore architecture. The design, implementation and integration of AUTOSAR applications in multicore ECU will face new challenges. One of these challenges is mapping the tasks to multicore ECU while the real-time performance of system is ensured. In addition, the resource limitation and scheduling analysis in real-time system make the problems more complex in AUTOSAR static configuration. So this paper proposed a Nash equilibrium based method for mapping AUTOSAR tasks to multicore ECU. This method has important practical significance in improving the efficiency of task mapping process by applying the priority of tasks in game process. Finally, the proposed method was applied to the automotive electronic instance in AUTOSAR. The experimental results show that the proposed method has good performance in the worst case response time of the runnable entities in each task.

Keywords AUTOSAR, Nash equilibrium, Task mapping, Multicore ECU

1 简介

近年来,随着汽车电子技术的发展以及人们对汽车舒适性和安全性需求的不断提高,许多功能复杂且安全性要求较高的汽车电子应用程序已越来越难以适应单核ECU。为了应对汽车电子应用程序(如发动机控制程序)对ECU计算性能需求的不断提高,多核结构的ECU也越来越多地应用于汽车电子系统中。然而,虽然汽车开放系统架构(AUTomotive OpenSystem ARchitecture,AUTOSAR)标准[1]支持多核结构ECU[2],但是这也给汽车电子软硬件系统的集成带来了新的挑战。

AUTOSAR将汽车电子应用程序的设计过程分为系统级和ECU级。系统级设计以软件构件为单位设计应用程序的功能,其中软件构件包含一组可运行实体。在ECU级,这些构件内的可运行实体将被完全拆分并重新组合为若干个任务,以便操作系统调度执行。在AUTOSAR静态配置过程中,这些可运行实体将以任务为单位映射到ECU的各个核上。在映射过程中,不仅需要考虑各个ECU的负载情况,而且需要确保各个可运行实体的实时性。这使得汽车电子软件到多核ECU的集成变得非常复杂[3],且需要合适的方法以及开发相应的工具。

对于汽车电子中的映射问题,目前的方法主要为启发式算法[4],包括遗传算法[5]、变邻域搜索算法[6]、模拟退火算法[7]等。然而,这些方法主要用于降低同构系统中的数据通

到稿日期:2017-05-16 返修日期:2017-08-14 本文受国家自然科学基金(61175061/F030506)资助。
冉 正(1987-),男,博士生,主要研究方向为嵌入式系统,E-mail:ranzheng517@sina.com;罗 蕾(1967-),女,硕士,教授,主要研究方向为嵌入式系统,E-mail:lluo@uestc.edu.cn(通信作者);晏 华(1970-),女,博士,副教授,主要研究方向为嵌入式软件、计算智能;李 允(1971-),男,博士,教授,研究员,主要研究方向为普适计算、实时、嵌入式操作系统及其应用、嵌入式应用设计方法。

信量。这些方法虽然考虑了映射过程中任务的实时性,但是仅以限定 ECU 利用率的方式来保证 ECU 的可调度性,没有考虑应用程序的响应时间。虽然基于博弈论的任务分配方法[8]也考虑了任务的响应时间,但该方法主要应用于服从泊松分布的随机任务,且主要考虑任务的平均响应时间,而非嵌入式实时领域所关注的最坏响应时间。

虽然目前已有很多关于多个处理器的任务分配方法[9],但是这些方法不太适用于 AUTOSAR 系统。首先,汽车电子 ECU 可能是一个异构的多核系统,各个 ECU 核可能具有不同的硬件结构,因此对于同一段程序代码,不同的 ECU 核可能具有不同的执行效率。其次,汽车电子系统中的任务不同于普通的任务,同一任务中的可运行实体可能具有不同的运行周期,这使得任务的执行时间变得难以评估[10]。虽然任务中的各个可运行实体具有各自不同的时间属性(如周期、最坏执行时间等),但是映射又是以任务为单位,即同一任务内的可运行实体必须映射到同一 ECU 核上。目前,虽然存在一些针对汽车电子的任务到多核 ECU 的映射方法(如 ILP 方法[11]),但是这些方法主要考虑了映射过程中 ECU 的负载,而没有详细地分析各个任务或可运行实体的实时性。

因此,针对 AUTOSAR 任务的特性以及可运行实体对实时性的需求,结合博弈论中的纳什均衡[8]原理,提出了映射任务到多核 ECU 的方法,并将其应用于 AUTOSAR 标准的汽车电子应用实例中。实验结果验证了所提方法的有效性。

2　任务模型

2.1　硬件结构

本文致力于解决多核 ECU 上的任务映射问题。在此,将 ECU 定义为:$U=\{u_k \mid k=1,2,\cdots,K\}$。其中,$U$ 为 ECU,u_k 为 ECU 中的第 k 个核。

2.2　构件定义

在 AUTOSAR 架构中,汽车电子应用软件由构件(Software Component)组成[10]。通常情况下,一个构件包含一个或多个可运行实体(Runnable Entity)。构件内的可运行实体通过相互协作来实现构件的具体行为。可运行实体是一段用于实现一个简单算法或某一特定功能的程序代码。在此,构件定义为:$C=\{c_i \mid i=1,2,\cdots,n\}$,$c_i=\{r_{i,j} \mid j=1,2,\cdots,m_i\}$。其中,$C$ 为一个 ECU 中的构件集合,c_i 和 $r_{i,j}$ 分别为构件和构件中的可运行实体,m_i 为构件 c_i 中可运行实体的数量。

可运行实体的执行依赖于事件(RTE-Event)的触发[10]。AUTOSAR 大致定义了 12 种不同类型的事件[10]。这些事件主要由底层计时器触发,即每隔一个固定的时间触发一次,因此由这些事件触发的可运行实体大多呈周期性。在此,可运行实体定义为 $r_{i,j}=(P(r_{i,j}),e(u_k,r_{i,j}))$。其中,$P(r_{i,j})$ 为可运行实体的周期,$e(u_k,r_{i,j})$ 为可运行实体在 u_k 上的最坏执行时间。根据 AUTOSAR 标准,可运行实体是一段功能较为单一的代码段。一般而言,其执行时间远小于周期,即对于 $\forall u_k, e(u_k,r_{i,j}) \ll P(r_{i,j})$。

由于构件的具体行为依赖于可运行实体之间的协作,因此可运行实体之间存在着大量通信。这些通信一般分为构件内部通信和构件之间通信。位于同一构件中的可运行实体可以共享构件内部的变量,而位于不同构件之间的可运行实体则只能通过端口通信。端口主要用于发送数据或接收数据,在端口通信中发送方和接收方虽然都需要缓冲区来缓存数据,但是可运行实体的执行情况却不会因此而受到影响,因为可运行实体在执行过程中只需要向缓冲区读写数据即可。然而,构件内部的共享数据因为加锁的缘故将会对可运行实体的执行造成一定的延时。在此,定义可运行实体 r_p 与 r_q 之间的共享数据为 $sd(r_p,r_q)$。

例 1　假设两个构件 c_1 和 c_2。c_1 中有 3 个可运行实体,即 r_1,r_2 和 r_3,其中 r_1 与 r_2 共享数据 a。c_2 中也有 3 个可运行实体,即 r_4,r_5 和 r_6,其中 r_4 与 r_5 共享数据 b。这 6 个可运行实体之间的数据共享关系如图 1 所示。

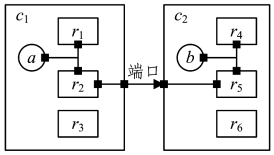

图 1　构件之间的通信关系

Fig. 1　Communication relationship between SWCs

图 1 中,不同构件之间的可运行实体通过端口通信,而同一构件内的可运行实体则通过构件内部的共享数据交互。例如,图 1 中的 a 和 b 分别为构件 c_1 和 c_2 的内部变量,r_1 和 r_2 共享数据 a,r_4 和 r_5 共享数据 b。

2.3　任务定义

任务作为 AUTOSAR 操作系统调度的基本单位,不仅为可运行实体提供了诸如上下文和堆栈空间等常用资源,而且为可运行实体的执行提供了优先级。任务的具体执行功能是由映射到任务的可运行实体确定的,因此一个或多个关联性较强的可运行实体往往被映射到一个任务中。在此,任务 $\Gamma=\{\tau_i \mid i=1,2,\cdots,N\}$,$\tau_i=\{r_{\tau_i,j} \mid j=1,2,\cdots,N_i\}$,其中,$N_i$ 为任务 τ_i 中可运行实体的数量。任务的时间属性为 $\{Pri(\tau_i),P(\tau_i),HP(\tau_i)\}$,其中,$Pri(\tau_i)$ 为任务的优先级,也是任务中可运行实体的优先级,$P(\tau_i)$ 为任务的周期,$HP(\tau_i)$ 为任务的长周期(Hyper-Period)。

在此,定义映射到任务之后的可运行实体的属性为 $\tau_i(r_j)=(P(r_j),e(u_k,r_j),I(r_j),Pri(r_j))$。其中,$P(r_j)$ 为可运行实体的周期,$e(u_k,r_j)$ 为可运行实体 r_j 在 u_k 上的最坏执行时间,$I(r_j)$ 为 r_j 在任务 τ_i 中的执行顺序,$Pri(r_j)$ 为 r_j 的优先级。任务为可运行实体赋予了优先级,即任务的优先级等于可运行实体的优先级,对于 $\forall r_{\tau_i,j}$,$Pri(r_{\tau_i,j})=Pri(\tau_i)$。

根据 AUTOSAR 规范,任务的周期为任务中所有可运行实体周期和激活偏移值的最大公约数,且不能小于可运行实体的最坏执行时间。任务的长周期为所有可运行实体周期的最小公约数。

例如,假设例 1 中的 r_1 与 r_2 被封装到任务 τ_1 中,那么这个任务中一共有 2 个可运行实体,即 r_1 和 r_2。假设其属性分别为 $\tau_1(r_1)=(20,4,1,1)$,$\tau_1(r_2)=(30,4,2,1)$,那么任务的周期为 $GCD(20,30)=10$,任务的长周期为 $LCM(20,30)=60$。任务中的可运行实体在一个长周期内的执行情况如图 2 所示。

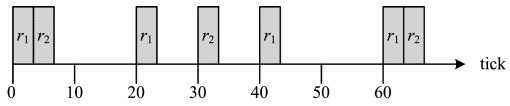

图 2 可运行实体的执行顺序

Fig. 2 Execution order of runnable entities

图 2 中，每 10 个节拍就调度一次任务，但任务中的可运行实体却并非每次任务调度都执行。例如，在任务的第 1 次执行中(区间[0,10])，可运行实体 r_1 和 r_2 都执行。此时任务具有最长的执行时间，即 $e(r_1)+e(r_2)=4+4=8<10$。

在任务的第 2 次执行中(区间[10,20])，就没有可运行实体执行，此时任务的执行时间为 0。从图 2 可以看出，在任务的每次调度中，任务的执行时间相差较大，因此本文不评估任务的执行时间，而是直接分析可运行实体的可调度性。

3 问题描述

构件到多核 ECU 的集成共分为两步。例 1 中，构件 c_1 和 c_2 描述了应用程序的功能。在 AUTOSAR 静态配置过程中，首先将构件拆分，将其内的可运行实体组合为任务，在组合过程中可运行实体的通信关系保持不变。构件的内部共享数据或成为任务内部数据，或成为任务间的共享数据。第二步是将任务映射到 ECU 的各个核上，本文主要关注第二步。以例 1 中的两个构件为例，其集成过程如图 3 所示。

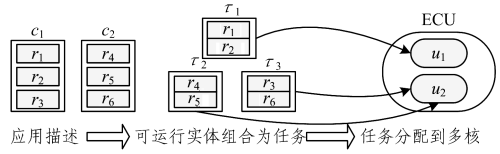

图 3 任务到多核 ECU 的映射流程

Fig. 3 Mapping process from tasks to multicore ECU

假设系统中的任务总数为 N，ECU 中核的总数为 K。本文致力于寻找一种自动映射方法，将 N 个任务映射到 K 个核上。在此，定义映射函数为 $M:\Gamma\rightarrow U$，即：

$$M(\tau_i)=u_k \tag{1}$$

其中，u_k 为 ECU 核，τ_i 为映射到 u_k 上的任务。

3.1 任务的最坏响应时间

根据 AOTUSAR 标准，部署到各个 ECU 核上的任务(由可运行实体组成)均采用固定优先级调度策略[12]。在不考虑共享数据加锁的情况下，可运行实体的最坏响应时间(Worst-case Response Time)为：

$$wcrt(r_i)=\sum_{\forall \tau_j,Pri(\tau_j)\geqslant Pri(\tau_i)}\left\lceil\frac{wcrt(r_i)}{P(r_{\tau_j,p})}\right\rceil\times e(u_k,r_{\tau_j,p}) \tag{2}$$

其中，τ_i，τ_j 位于同一 ECU 核上，即 $M(\tau_i)=M(\tau_j)=u_k$。由于 $wcrt(r_i)$在等式两边均出现，因此采用递归方式(见式(2))来求解 $wcrt(r_i)$，在此不赘述。

由于可运行实体之间存在共享数据(如图 1 中的 a 和 b)，因此 AUTOSAR 标准规定了信号量、自旋锁等同步机制来保证共享数据的一致性。然而，这将导致优先级反转。为了应对这一情况，OSEK 标准规定了操作系统将采用优先级天花板策略来处理任务之间的同步问题。对于多核操作系统，AUTOSAR 标准采用了自旋锁的方式来保证多核数据之间的一致性。设 st 为同一 ECU 核内的共享数据访问时间(即加信号量的时间)，lt 为核间的共享数据加锁时间，则可运行实体的最坏响应时间为：

$$wcrt(r_i)=\sum_{\forall \tau_j,Pri(\tau_j)\geqslant Pri(\tau_i)>Pri(\tau_m)\wedge k\neq l}\left\lceil\frac{wcrt(r_i)}{P(r_{\tau_j,p})}\right\rceil\times (e(u_k,r_{\tau_j,p})+st(u_k,sd(r_{\tau_j,p},r_{\tau_m,q}))+lt(u_l,sd(r_{\tau_j,p},r_{\tau_z,y}))) \tag{3}$$

由于任务中各个可运行实体的最坏响应时间是不同的，因此定义任务的最坏响应时间等于任务中可运行实体的最坏响应时间的最大值，即：

$$wcrt(\tau_i)=\max(wcrt(r_{\tau_i,1}),wcrt(r_{\tau_i,N_i})) \tag{4}$$

3.2 纳什均衡

N 人有限策略博弈通常描述为：

$$G=\{S_1,\cdots,S_N;p_1,\cdots,p_N\} \tag{5}$$

本文中，任务 $\tau_1-\tau_N$ 为 N 个博弈参与人，τ_i 的纯策略集为有限集，记作 $S_i=(s_{i,1},\cdots,s_{i,K})$。其中，$s_{i,k}$ 表示将 τ_i 映射到 u_k 上，每个参与人的支付函数为 u_i。在此，设定每个任务在博弈过程中均希望获得最快的最坏响应时间，支付函数为任务的最坏响应时间，即 $p_i=wcrt(\tau_i)$。

在博弈 $G=\{S_1,\cdots,S_N;p_1,\cdots,p_N\}$中，如果在由各个博弈方的各个策略组成的某个策略组合$(s^*_{1,j_1},\cdots,s^*_{N,j_N})$中，任一博弈方 c_i 的策略 s^*_{i,j_i} 都是对其余博弈方策略的组合$(s^*_{1,j_1},\cdots,s^*_{i-1,j_{i-1}},s^*_{i+1,j_{i+1}},\cdots,s^*_{N,j_N})$的最佳对策，即

$$p_i(s^*_{1,j_1},\cdots,s^*_{i-1,j_{i-1}},s^*_{i,j_i},s^*_{i+1,j_{i+1}},\cdots,s^*_{N,j_N})\geqslant p_i(s^*_{1,j_1},\cdots,s^*_{i-1,j_{i-1}},s_{i,j_i},s^*_{i+1,j_{i+1}},\cdots,s^*_{N,j_N})$$

对任意 s_{i,j_i} 都成立，则称$(s^*_{1,j_1},\cdots,s^*_{N,j_N})$为 G 的一个纳什均衡。

本文中，每个任务都致力寻找对自己最有利的 ECU 核映射，即在该 ECU 核上，任务具有最快的最坏响应时间。在经过多轮博弈之后，每个任务都不能通过单独改变自身的映射来获得更快的最坏响应时间。此时，每个任务都获得了仅对自己而言的最优映射方案，即达到纳什均衡。

4 映射方法

为了使每个任务都能获得较短的最坏响应时间，本文采用博弈论中的纳什均衡原理来求解任务对多核 ECU 的映射方案，最后得出映射函数 M。

在映射过程中，每个任务都可能映射到 K 个 ECU 核中的任何一个，因此每个任务都有 K 种映射策略。系统中一共有 N 个任务，因此系统的总映射策略有 K^N 种。如果采用常规方法进行求解，则须遍历 K^N 种映射策略，从而获得博弈中的纳什均衡点。然而，由于本文采用固定优先级调度策略，因此博弈中只存在唯一的一个纳什均衡点，且在求解纳什均衡时，无须遍历所有的映射策略。

证明过程如下：

假设任务按优先级由大到小排序，即任务编号越小，任务中可运行实体的优先级越高。

$$Pri(r_{\tau_1})>Pri(r_{\tau_2})<\cdots<Pri(r_{\tau_N})$$

由于低优先级的可运行实体在执行过程中会被高优先级的可运行实体抢占，因此低优先级的可运行实体不会对高优

先级的可运行实体的最坏响应时间造成影响。故可将求解纳什均衡的过程分为 N 步完成。

首先,假设存在一个任务集合,然后将任务 τ_1 加入到集合中。然后,将 τ_1 映射到使其具有最短最坏响应时间的 ECU 核上。最后,将 τ_2 加入集合并完成 τ_2 的映射。由于 τ_1 中可运行实体的优先级高于 τ_2 中可运行实体的优先级,因此,无论 τ_2 选择何种映射策略,均不会影响 τ_1 的最坏响应时间。同理,将 τ_{i+1} 加入任务集合中时,τ_{i+1} 的映射策略不会影响$\{\tau_1,\tau_i\}$的最坏响应时间。以此类推,直至完成 τ_N 的映射。

因此,在求解纳什均衡时,只需要将所有任务按照优先级由大到小的顺序排列,然后逐一搜索其最优策略,便可寻找到博弈中的纳什均衡点,且这个均衡点是唯一的。

证毕。

基于以上原理,按照如下步骤搜索映射过程中的纳什均衡点。

1)任务按优先级由大到小排序,初始化各个任务的最坏响应时间为无穷大;

2)任务 τ_i 从 τ_1 开始到 τ_N 依次执行步骤 3)和步骤 4);

3)为任务 τ_i 选择具有最优的 ECU 核 u_d,即任务在该 ECU 核上具有最短的最坏响应时间。

4)完成 τ_i 到 u_d 的映射,同时更新任务的最坏响应时间。

例如:假设图 3 中的 ECU 为同构系统,即例 1 中的 6 个可运行实体在 u_1 和 u_2 中的执行时间相同,其具体属性如下:$\tau_1(r_1)=(20,4,1,1)$,$\tau_1(r_2)=(30,4,2,1)$,$\tau_2(r_4)=(60,4,1,2)$,$\tau_2(r_5)=(60,4,2,2)$,$\tau_3(r_3)=(60,10,1,3)$,$\tau_3(r_6)=(120,20,2,3)$。由于 $\tau_1-\tau_3$ 的优先级关系为:$Pri(\tau_1)>Pri(\tau_2)>Pri(\tau_3)$,因此首先对任务 τ_1(即 r_1 和 r_2)进行映射操作。由于任务 τ_1 的优先级最高,且不存在与其他任务通信的情况,因此 r_1 和 r_2 的最坏响应时间等于其最坏执行时间,即 r_1 和 r_2 在 u_1 和 u_2 上的最坏响应时间为:$wcrt(r_1)=4$,$wcrt(r_2)=4$。此时,任务 τ_1 在 u_1 与 u_2 中的最坏响应时间为 $wcrt(\tau_1)=\max(4,4)=4$。由于任务 τ_1 在 u_1 和 u_2 中的最坏响应时间均为 4,因此随机选择一个 ECU 核,在图 3 中选择 u_1。

接着,计算 τ_2 在 u_1 和 u_2 中的最坏响应时间。由于 u_1 中已经存在可运行实体 r_1 和 r_2,且它们的优先级均高于 r_4 和 r_5,因此在 u_1 中 r_4 和 r_5 有可能会被 r_1 和 r_2 抢占。根据式(3),r_4 和 r_5 在 u_1 中的最坏响应时间为 $wcrt(r_4)=12$,$wcrt(r_5)=12$,因此任务 τ_2 在 u_1 中的最坏响应时间为 $wcrt(\tau_2)=\max(12,12)=12$。与 u_1 不同的是,u_2 中不存在任何可运行实体,因此 r_4 和 r_5 在 u_2 中的最坏响应时间等于其最坏执行时间,即 $wcrt(r_4)=4$,$wcrt(r_5)=4$。任务 τ_2 在 u_2 中的最坏响应时间为 $wcrt(\tau_2)=\max(4,4)=4$。由于 τ_2 在 u_2 中的最坏响应时间小于其在 u_1 中的最坏响应时间,因此将 τ_2 映射到 u_2 中。

此时,u_1 中已存在 τ_1,u_2 中也已存在 τ_2。对于 τ_3 中的 r_3 和 r_6,分别计算其在 u_1 和 u_2 中的最坏响应时间。在 u_1 中,τ_3 可能被 τ_1 抢占,根据式(3),r_3 和 r_6 在 u_1 中的最坏响应时间分别为 $wcrt(r_3)=22$,$wcrt(r_6)=36$,$wcrt(\tau_2)=\max(22,36)=36$。同理,在 u_2 中,τ_3 可能被 τ_2 抢占,根据式(3),r_3 和 r_6 在 u_2 中的最坏响应时间分别为 $wcrt(r_3)=18$,$wcrt(r_6)=28$,$wcrt(\tau_2)=\max(18,28)=28$。由于 τ_3 在 u_2 中的最坏响应时间小于其在 u_1 中的最坏响应时间,因此将 τ_3 映射到 u_2 中。

搜索纳什均衡算法(MAP_NASH)如算法 1 所示。算法 1 中使用到的数据类型和变量的定义如下:$w[N]$为 N 个任务的最坏响应时间,$M[N]$为任务的映射函数。在算法 1 中为每个 ECU 核定义一个任务集合,记录映射到该 ECU 核上的任务,记为 $U[N]$。

算法 1　MAP_NASH

输入:N,K,P,e

输出:M

```
1. 初始化 U[K]为空,M[N]为 0,w[N]为∞;
2. for τi←τ1 to τN
3.    w←w[τi];uc←M(τi);ud←0;
4.    for ui←1 to K
5.       计算 τi 在 ud 中的最坏响应时间 wcrt(τi,ud);
6.       if wcrt(τi,ud)<w
7.          w←wcrt(τi,ud);
8.          ud←ui;
9.       end
10.   end
11.   U[ud]←U[ud]∪τi;
12.   M[τi]←ud;
13.   更新 τi 的最坏响应时间 w[τi];
14. end
```

算法 1 中,为每个任务搜索最优 ECU 核时,均需要遍历所有的 ECU 核,因此算法的时间复杂度为 $O(N\times K)$。

5　实验

本文构造了两组实验,分别将基于纳什均衡的映射方法(简称 NASH 方法)应用于同构和异构的多核 ECU 系统中。在同构多核 ECU 系统中,将 NASH 方法与文献[11]中的线性规划算法(简称 ILP 方法)进行了比较。其中,ILP 方法使用 IBM 的 CPLEX 工具计算。在异构多核 ECU 系统中,NASH 方法与文献[13]中的利用率平衡算法(简称 UB 方法)进行了比较。

实验环境为 Windows7 64-bit 操作系统,Intel(R) Core(TM) i5-4460 CPU@3.2GHz 以及 8 GB 内存。NASH 方法和 UB 方法的编译环境为 MATLAB version 8.3.0.532 64-bit。ILP 方法的编译环境为 IBMCPLEX 12.6.3.0。

5.1　评估指标

本文主要从以下 3 个方面来评估算法的性能。

1)每个任务的最坏响应时间 $wcrt$。

2)每个任务在两种算法中的最坏响应时间之差 $\Delta wcrt$。在同构系统中,其为各个任务在 NASH 方法中的最坏响应时间与 ILP 方法中的最坏响应时间之差。在异构系统中,其为各个任务在 NASH 方法中的最坏响应时间与 UB 方法中的最坏响应时间之差。

3)各个 ECU 核的利用率 $utilization$。

5.2　同构系统

对于同构多核 ECU,本文抽取了 AUTOSAR 规范中关于汽车电子巡航控制部分的实例并对其进行补充完善。汽车

电子巡航控制系统[14]主要由 6 个构件组成，分别为传感器数据收集构件、目标对象选择构件、自由巡航控制构件、跟踪控制与加速仲裁构件、巡航控制与 ACC 状态机构件以及巡航控制和 ACC 车辆观测器构件。本文根据这 6 个构件的功能和通信拓扑结构构建了 39 个可运行实体、17 个任务和 23 个构件内的共享数据，如图 4 所示。

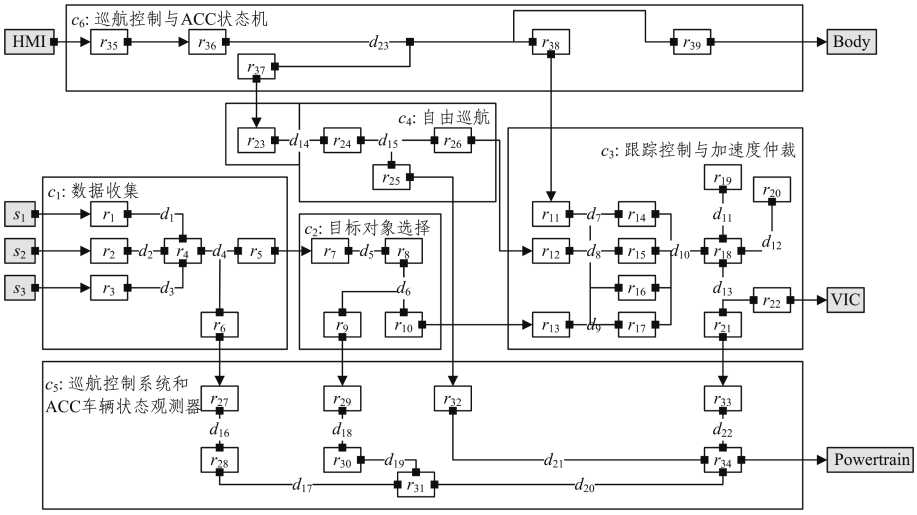

图 4 汽车电子巡航控制系统

Fig. 4 Automotive adaptive cruise control system

39 个可运行实体(r_1－r_{39})的周期与执行时间如表 1 所列。

表 1 可运行实体的周期与执行时间

Table 1 Period and execution time of runnable entities

(单位：μs)

r	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
P	20	20	20	60	30	30	30	60	30
e	1	1	1	1	1	1	1	2	1
r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}
30	120	120	120	240	240	240	240	120	40
1	1	1	1	2	2	2	2	2	1
r_{20}	r_{21}	r_{22}	r_{23}	r_{24}	r_{25}	r_{26}	r_{27}	r_{28}	r_{29}
40	120	120	60	120	120	120	30	60	30
1	1	1	1	2	1	1	1	1	1
r_{30}	r_{31}	r_{32}	r_{33}	r_{34}	r_{35}	r_{36}	r_{37}	r_{38}	r_{39}
60	120	60	120	240	30	30	60	60	120
1	2	1	1	2	1	2	1	1	1

17 个任务分别为 $\tau_1=\{r_1\}$，$\tau_2=\{r_2\}$，$\tau_3=\{r_3,r_4,r_5,r_7,r_8\}$，$\tau_4=\{r_6,r_{27},r_{28}\}$，$\tau_5=\{r_9,r_{29},r_{30},r_{31}\}$，$\tau_6=\{r_{10},r_{13},r_{14}\}$，$\tau_7=\{r_{15}\}$，$\tau_8=\{r_{16}\}$，$\tau_9=\{r_{17},r_{18}\}$，$\tau_{10}=\{r_{19}\}$，$\tau_{11}=\{r_{20}\}$，$\tau_{12}=\{r_{21},r_{33},r_{34}\}$，$\tau_{13}=\{r_{22}\}$，$\tau_{14}=\{r_{26},r_{12}\}$，$\tau_{15}=\{r_{35},r_{36}\}$，$\tau_{16}=\{r_{37},r_{38},r_{11},r_{23},r_{24},r_{25},r_{32}\}$，$\tau_{17}=\{r_{39}\}$。这 17 个任务中，任务编号越小，任务优先级越高。

1)每个任务的最坏响应时间 $wcrt$

将 NASH 方法与 ILP 方法分别应用于这 17 个任务中，并将这些任务分别映射到 4 个 ECU 核上。任务的映射情况以及每个 ECU 核上任务的最坏响应时间如图 5 所示。方法 1 的映射结果如图 5 中的第一行 ECU1－ECU4 所示，方法 2 的映射结果如图 5 中的第二行 ECU1－ECU4 所示。

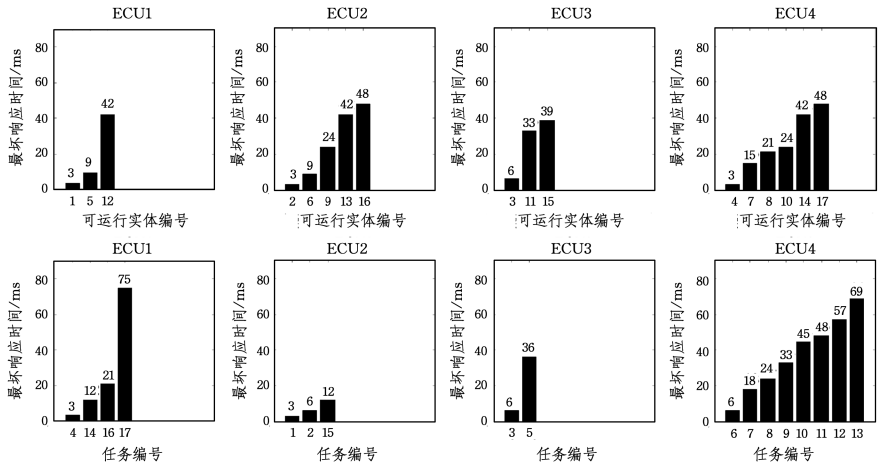

图 5 任务的最坏响应时间

Fig. 5 Worst-case response time of tasks

在图 5 中，根据 NASH 方法的映射结果，所有可运行实体的最坏响应时间的总和为 411 ms。在 ILP 方法的映射结果中，所有任务的最坏响应时间的总和为 474 ms。对于所有可运行实体和任务的总体最坏响应时间，NASH 方法的映射结果优于 ILP 方法。

对于每个 ECU 中任务的最坏响应时间，NASH 方法比 ILP 方法更加平稳。在 NASH 方法中，各个 ECU 的任务的最长最坏响应时间分别为 42 ms，48 ms，39 ms，48 ms；而在 ILP

方法中,各个 ECU 的任务的最长最坏响应时间分别为 75 ms,12 ms,36 ms,69 ms。因此,在 NASH 方法中,各个 ECU 的任务最坏响应时间分布得较为均匀,不会出现某个任务最坏响应时间过长的情况。而 ILP 方法则出现了 75 ms 和 69 ms 这样的最坏响应时间过长的情况。

2)各个任务在 NASH 方法中的最坏响应时间与 ILP 方法中的最坏响应时间之差 $\Delta wcrt$

NASH 方法与 ILP 方法在每个任务中的响应时间之差如图 6 所示。

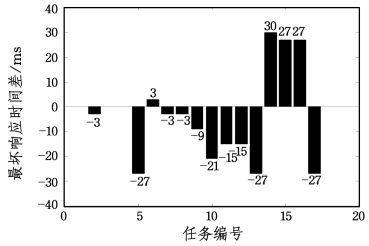

图 6　NASH 方法与 ILP 方法在每个任务中的最坏响应时间差

Fig. 6　Difference of worst-case response time between tasks from NASH and ILP

从图 6 中可以看出,NASH 方法更致力于减少高优先级任务的响应时间,如任务 τ_2,τ_5,τ_7,τ_8,τ_9,τ_{10},τ_{11},τ_{12},τ_{13},τ_{17}。而 ILP 方法则主要考虑 ECU 核的利用率,而不是任务的优先级,因此仅在任务 τ_6,τ_{14},τ_{15},τ_{16} 中取得较快的响应时间。

3)各个 ECU 核的利用率 *utilization*

图 7 给出了这两种方法中各个 ECU 核的利用率情况。

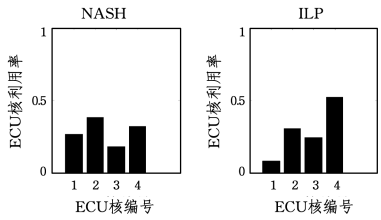

图 7　NASH 和 ILP 中各 ECU 核的利用率

Fig. 7　Utilization of ECU cores in NASH and ILP

在图 7 中,NASH 方法使得各个 ECU 核的利用率相差不大。而 ILP 方法由于在映射过程中需要考虑任务间的通信,导致各个 ECU 核的利用率相差较大。

5.3　异构系统

对于异构的多核 ECU 系统,本文以 AUTOSAR 规范中的 904 个构件[15]为原型,为每个构件生成 1 个可运行实体。其中,每个可运行实体的周期为 20～100 ms,即 $20 \leqslant P(r_{i,j}) \leqslant 100$。每个可运行实体的最坏执行时间为一个[0,1]之间的随机数(rand)乘以周期的 1/100,即 $e(u_k, r_{i,j}) = rand \times P(r_{i,j})/1000$。然后,将周期相同的可运行实体组合为一个任务,共计 80 个。最后,将这 80 个任务映射到 8 个 ECU 核上。由于 ILP 方法受限于数据规模,在异构系统中,对 NASH 方法与 UB 方法的映射结果进行比较。

1)每个任务的最坏响应时间 *wcrt*。

图 8 给出了 80 个任务分别在 NASH 方法与 UB 方法中的最坏响应时间。

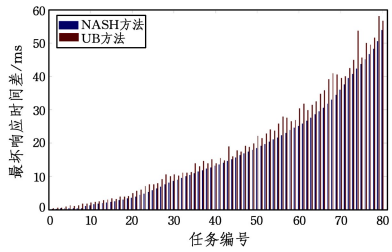

图 8　80 个任务在 NASH 方法和 UB 方法中的最坏响应时间

Fig. 8　Worst-case response time of 80 tasks with NASH and UB

从图 8 中可以看出,80 个任务在 NASH 方法中的最坏响应时间随着优先级的降低而平稳递增,且增幅明显小于其在 UB 方法中的最坏响应时间。任务在 UB 方法中的最坏响应时间随着优先级的降低也呈递增的趋势。但是,其增幅相对较大,且增幅的波动也较大。

2)各个任务在 NASH 方法中的最坏响应时间与 UB 方法中的最坏响应时间之差 $\Delta wcrt$

图 9 给出了 80 个任务在 NASH 方法与 UB 方法中的响应时间之差。

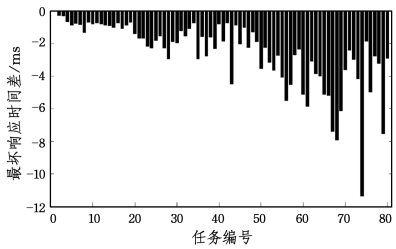

图 9　80 个任务在 NASH 方法与 UB 方法中的最坏响应时间差

Fig. 9　Difference of worst-case response time between 80 tasks with NASH and UB

从图 9 可以看出,所有任务在 NASH 方法中的最坏响应时间均比其在 UB 方法中的最坏响应时间短。

3)各个 ECU 核的利用率 *utilization*

图 10 给出了这两种方法中各个 ECU 核的利用率。

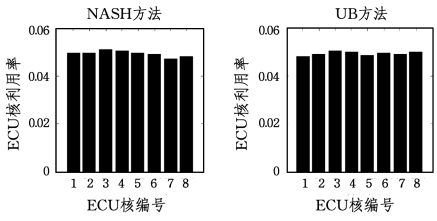

图 10　NASH 和 UB 中各 ECU 核的利用率

Fig. 10　Utilization of ECU cores in NASH and SB

从图 10 中可以看出,NASH 方法和 UB 方法均使 ECU 的各个核的利用率达到了均衡。在 ECU 负载均衡方面,这两种方法的效果相差不大。

结束语　本文根据汽车电子多核 ECU 的任务映射需求,提出任务到多核 ECU 的映射方法。该方法结合了实时系统调度策略与博弈论中的纳什均衡原理,并根据固定优先级调度策略的特点,设计并实现了求解纳什均衡的方法。最后,将所提方法应用于同构和异构的汽车电子实时系统中,并与之

(下转第 182 页)