

# 指令预取感知的多核系统 WCRT 和 WCEC 优化

韩丽艳<sup>1\*</sup>, 安立奎<sup>2</sup>

(1. 渤海大学 信息科学与技术学院, 辽宁 锦州 121013;

2. 渤海大学 数理学院, 辽宁 锦州 121013)

**摘要:**对嵌入式多核下的实时系统,为了保证任务的可调度性和可靠性,最坏情况下的性能是一个优先考虑的问题,同时对于能量供应有限制的多核系统,最坏情况下的能量消耗也是一个非常关键的问题。顺序指令预取可以提高实时任务的最坏情况下的性能,但对于实时系统中多个不同的子任务,多个子任务采用相同指令预取度不利于提高指令预取在最坏情况下的性能和能量效率。文中利用缓存划分技术消除实时系统中多个子任务在共享缓存上的干扰,提出了支持指令预取的 WCRT(worst-case response time)和 WCEC(worst-case energy consumption)优化模型,并设计了优化算法。该算法通过建立任务图调整实时系统中子任务的指令预取度,在最小化系统 WCRT 的基础上,减少系统的 WCEC。实验对 DEBIE 系统进行实例分析,结果表明优化算法在保证实时系统 WCRT 最小的情况下,其 WCEC 减少了 10.8%。

**关键词:**多核实时系统; WCRT; WCEC; 指令预取度; 缓存划分

中图分类号: TP314

文献标识码: A

文章编号: 1673-629X(2020)09-0082-06

doi: 10.3969/j.issn.1673-629X.2020.09.015

## Instruction-prefetching-aware WCRT and WCEC Optimization in Multicore

HAN Li-yan<sup>1\*</sup>, AN Li-kui<sup>2</sup>

(1. School of Information Science and Technology, Bohai University, Jinzhou 121013, China;

2. School of Mathematics and Physics, Bohai University, Jinzhou 121013, China)

**Abstract:** For the real-time system in embedded multicore, in order to guarantee the task schedulability and the reliability, the worst-case performance is a prior to be considered. At the same time for multicore system with limited energy supply, the worst-case energy consumption is also a critical problem. Sequent instruction prefetching can improve the worst-case performance of real-time tasks, but using the same instruction prefetching degree for multiple different subtasks is not beneficial to improve the worst-case performance efficiency and energy efficiency of instruction prefetching. In this paper, cache partitioning technology is used to eliminate interferences of multiple real-time subtasks on the shared cache, and the WCRT(worst-case response time) and WCEC(worst-case energy consumption) optimization model and algorithms with instruction prefetching are designed. The algorithm adjusts instruction prefetching degrees of real-time subtasks by establishing the task graph, and reduces the system WCEC on the basis of minimizing system WCRT. DEBIE system is analyzed in experiment to show that the optimal algorithm can reduce the cache WCEC by 10.8% under the condition that the WCRT of real-time system is kept to a minimum.

**Key words:** multicore real-time system; worst-case response time; worst-case energy consumption; instruction prefetching degree; cache partitioning

## 0 引言

对于嵌入式多核下的实时系统,最坏情况下响应时间(worst-case response time, WCRT)是一个优先考虑的问题,来保证实时任务的可靠性和可调度性<sup>[1]</sup>。同时对于一些能量供应受限的实时系统,例如卫星的传感监视系统,普适系统,最坏情况下的能量消耗

(worst-case energy consumption, WCEC)也是一个非常重要的因素<sup>[2]</sup>。

许多嵌入式多核系统支持顺序指令预取技术来提高系统的性能,文献[3]研究表明硬件预取也能够降低系统的能耗。实时系统通常具有多个并发子任务,由于系统中的子任务有事务密集型的,也有数据密

收稿日期: 2019-10-25

修回日期: 2020-03-04

基金项目: 辽宁省科技项目(20180550691)

作者简介: 韩丽艳(1979-),女,讲师,硕士,通信作者,研究方向为实时计算。

集型的,顺序指令预取的性能和能量效率就与指令预取度有着密切关系,如果这些子任务采用相同预取度,不利于提高实时系统的性能和降低系统的能耗。

目前文献[4]提出一种多核实时系统的 WCRT 分析方法,通过分析并发多个任务在共享缓存上的干扰来实现。文献[5]通过优化任务到核的映射,减少实时系统的 WCRT。文献[4-5]并没有关注指令预取对与实时系统的最坏情况下的能耗的影响。文献[6-7]研究了 Next-N 硬件指令预取技术对 WCET (worst-case execution time) 的影响,考虑的仅是单核下的单级指令缓存,并没有考虑多核实时系统的多级缓存,也没有考虑实时系统最坏情况下的能量效率。文献[8]用循环译码的指令缓存减少能量消耗,提出了一个支持指令预取的能量有效的单核嵌入式模型。文献[9]用单层指令预取缓冲区和锁缓存来优化任务的 WCET 和 WCEC。文献[10]提出一种基于基本块的指令预取方法来优化系统 WCET 的评估值。

针对具有多级缓存的嵌入式多核系统,目前还没有利用指令预取来优化其最坏情况下性能和能量消耗方面的研究工作。为此,文中提出了支持指令预取的

WCRT 和 WCEC 优化模型,设计了相应的优化算法,通过实例分析验证了优化算法的有效性。

## 1 支持指令预取的嵌入式多核模型

### 1.1 嵌入式多核结构

文中研究的支持指令预取的嵌入式多核模型有 6 个同构的处理器核,如图 1 所示,每个核独有一级 (L1) 指令缓存 I-cache 和一级数据缓存 D-cache,所有的核共享联合二级 (L2) 指令/数据缓存。共享 L2 缓存有 W 组 (set),通过组缓存划分<sup>[11]</sup>,它被分成了  $N_c + 1$  部分,即  $P = \{p_1, p_2, \dots, p_{N_c+1}\}$ ,  $\sum_{i=1}^{N_c+1} p_i = W$ 。  $p_i$  ( $i = 1, 2, \dots, N_c$ ) 列被分给了运行实时任务的核  $C_i$ ,  $p_{N_c+1}$  被余下的所有运行非实时任务的核共享。一级缓存和二级缓存通过实时总线连接。顺序指令预取器 I-prefetcher 支持 Next-N-Line 指令预取。如果处理器需要处理的当前指令  $i$  被映射到 L1 指令缓存行  $p$  在 L1 指令缓存上没有命中,那么 L1 指令缓存预取器 I-prefetcher 就会发出预取 L1 指令缓存行  $p + 1, p + 2, \dots, p + N$  的操作请求,这里  $N$  称为指令预取度。

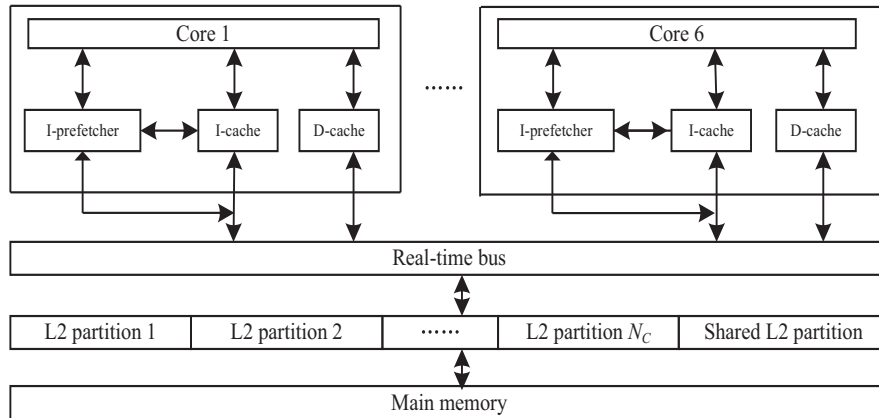


图 1 支持指令预取器的嵌入式多核架构

并发多任务的执行环境是一个基于静态优先级的非抢占系统,假设实时系统有  $N_r$  个相互依赖的实时任务。这些任务根据它们之间可并行化的关系被映射到了  $N_c$  ( $N_c < 6$ ) 处理器核,与实时系统无关的非实时任务都被映射到了剩下的核中。这里,缓存划分的目的是为了消除并发的多个实时任务在共享缓存上的干扰,确保实时任务的时间可分析性。L2 优先分给实时任务使得每个核上的实时任务 WCET 满足其截止期,剩余的 L2 缓存分给非实时任务。

### 1.2 能量模型

文中采用文献[12]中的能量消耗计算模型,对于不同的预取度  $n$  ( $n \geq 0$ ),当  $n$  为 0 时,表示没有采用预取操作,这时候关闭预取器,式(1)计算单个实时任务  $T$  支持指令预取的最坏情况下的能耗:

$$E(T, n) = E_{\text{static}}(n) + E_{\text{dynamic}}(n) \quad (1)$$

其中,

$$E_{\text{static}}(n) = \begin{cases} (C_{\text{static}} + P_{\text{static}}) * t_n & \text{if } n > 0 \\ C_{\text{static}} * t_n & \text{if } n = 0 \end{cases}$$

$$E_{\text{dynamic}}(n) = \begin{cases} E_{c\_dynamic}(n) + E_{p\_dynamic}(n) & \text{if } n > 0 \\ E_{c\_dynamic}(n) & \text{if } n = 0 \end{cases}$$

$$E_{c\_dynamic}(n) = E_c * c_n, E_{p\_dynamic}(n) = E_p * p_n$$

任务  $T$  最坏情况下总的能耗是由最坏情况下的静态能耗  $E_{\text{static}}(n)$  和动态能耗  $E_{\text{dynamic}}(n)$  组成。其中  $C_{\text{static}}$ ,  $p_{\text{static}}$  分别是处理器和预取器的静态能耗,  $t_n$  是当预取度是  $n$  时的任务最坏情况执行时间,当  $n = 0$  时,意味着关闭预取器。动态能耗  $E_{\text{dynamic}}(n)$  由处理器和预取器的动态能耗组成,  $E_c$  表示每次访问存储器所耗费在存储系统上的能耗,  $c_n$  是当预取度为  $n$  时的

最坏情况下访存次数。 $E_p$  表示每次预取操作预取器的动态能耗,  $p_n$  是最坏情况下的预取操作次数。

## 2 支持指令预取的 WCRT 和 WCEC 优化

### 2.1 实时系统 WCRT 和 WCEC 优化模型

实时系统  $RS = \{TS, D, M, WCRT, WCEC\}$ ,  $TS$  是其所有子任务的集合  $TS = \{T_1, T_2, \dots, T_{N_r}\}$ ,  $D$  是顺序指令预取度集合  $D = \{0, 1, \dots, N_p\}$ , 0 表示关闭指令预取器,  $M$  是任务  $TS$  到预取度  $D$  的映射:  $M: TS \rightarrow D$ 。

通过实时系统的 MSG (message sequence graph) 图, 得到了实时系统任务之间的依赖关系, 用  $Pred(T_i)$  来表示任务  $T_i$  的所有前驱任务的集合,  $Start(M, T_i)$  表示任务  $T_i$  在映射  $M$  下的开始时间,  $Finish(M, T_i)$  表示任务  $T_i$  在映射  $M$  下的结束时间,  $WCET(M, T_i)$  表示任务  $T_i$  在映射  $M$  下的最坏情况执行时间, 任务  $T_i$  的开始时间是  $T_i$  所有前驱任务完成时间的最大值, 如式(2), 任务  $T_i$  的结束时间是  $T_i$  开始时间与  $WCET(M, T_i)$  的和, 如式(3):

$$Start(M, T_i) = \max(Finish(M, T_j)), T_j \in Pred(T_i) \quad (2)$$

$$Finish(M, T_i) = Start(M, T_i) + WCET(M, T_i) \quad (3)$$

实时系统的支持指令预取的 WCRT 是运行在所有核上任务的结束时间的最大值, 如式(4):

$$WCRT = \max(Finish(M, T_i)), 1 \leq i \leq N_r \quad (4)$$

为了通过上面的公式计算实时系统的 WCRT, 文中根据 MSG 建立与实时系统相对应的有向无环任务图  $TG = (V, E)$ , 每个节点  $v_i \in V$  表示一个实时任务  $T_i$ , 每条有向边  $e_i \in E$  表示任务间的依赖关系。每条边上有一个权值, 它代表这条边的始点代表的任务  $T_i$  在映射  $M$  下支持指令预取的 WCET。为了计算整个系统的 WCRT, 文中需要给这个无环图的终节点再增加一个汇集节点  $T_{end}$  作为任务图的终点, 那么整个实时系统的 WCRT 是运行在所有核上的任务的结束时间的最大值, 也就是 TG 的源点 (开始任务) 到汇集节点 (终止任务) 的关键路径长度。

实时系统在最坏情况下的能量消耗 WCEC 是所有子任务在映射  $M$  下最坏能量消耗的和, 如式(5), 其中  $WCEC(M, T_i)$  是任务  $T_i$  在映射  $M$  下的 WCEC。

$$WCEC = \sum_{i=1}^{N_r} WCEC(M, T_i) \quad (5)$$

文中的优化目标是寻找映射  $M$ , 使得在实时系统的 WCRT 最小化的情况下, WCEC 最小, 即:

$$\text{Minimize } \sum_{i=1}^{N_r} WCEC(M, T_i) \quad (6)$$

当,

$$\text{Minimize } (\text{Maximize } (\text{Finish}(M, T_i))), 1 \leq i \leq N_r \quad (7)$$

### 2.2 WCRT 和 WCEC 优化算法

#### 2.2.1 WCRT 和 WCEC 优化原理

对于实时系统,  $WCET(d_j, T_i)$  表示任务  $T_i$  在预取距离  $d_j$  下的 WCET, 首先通过式(8)确定映射  $M_p$ , 对于每一个子任务  $T_i$ , 在给定的预取度范围  $D$  内,  $M_p$  确定它取得最小 WCET 的预取距离  $d_j$ :

$$M_p: M_p(T_i) = d_j, 1 \leq i \leq N_r, d_j \in D \\ \text{and for } \forall d_k \in D, WCET(d_j, T_i) \leq WCET(d_k, T_i) \quad (8)$$

在映射  $M_p$  下, 计算 TG 的关键路径  $P = \{T_{p_1}, T_{p_2}, \dots, T_{p_n}\}$ , 关键路径  $P$  的长度是整个实时系统的最小化的 WCRT。此时令  $M(T_i) = M_p(T_i)$ , 如果  $T_i \in P$ 。

对于非关键路径上的实时任务  $T_j \notin P$ , 在不改变关键路径的情况下, 最小化  $T_j$  的 WCET 并不能够减小整个系统的 WCRT, 这时候取它们的能量获益最高的预取度, 会降低整个系统的能量消耗。 $M(T_j) = d_k$ , 这里  $d_k$  使得任务  $T_j$  在不改变 TG 的关键路径情况下 WCEC 最小。

下面通过一个例子对 WCRT 和 WCEC 优化方法进行说明。假设有两个核, 实时系统具有 6 个子任务, 图 2 是其 MSG 描述, 图 3 是其相应的 TG, 其边上的权值是所有子任务支持指令预取的不同预取度下最小的 WCET。

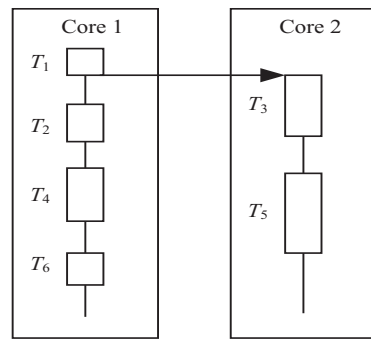


图 2 实时系统 MSG

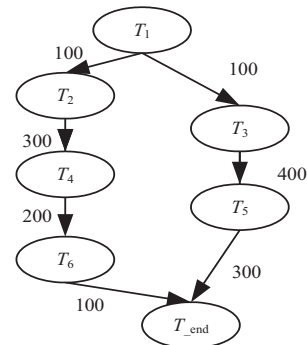


图 3 实时系统 TG

由 TG 可以计算从任务  $T_1$  到任务  $T_{end}$  的关键路径

$P: T_1 \rightarrow T_3 \rightarrow T_5 \rightarrow T_{end}$ , 路径长度是 800, 这里整个实时系统的最优的 WCRT 就是 800。对于非关键路径  $P: T_1 \rightarrow T_2 \rightarrow T_4 \rightarrow T_6 \rightarrow T_{end}$  上的实时任务  $T_2, T_4, T_6$ , 为了不影响与它们对应的关键路径  $P$ , 它们总共可以延长 100, 在这个范围内, 再次调整  $T_2, T_4, T_6$  的指令预取度, 使得能量获益最大。

### 2.2.2 WCRT 和 WCEC 优化算法实现

优化算法的难点主要有两处: 一是如何识别出非关键子路径, 计算非关键子路径的可以被延长的时间片; 二是如何把时间片分给非关键子路径上的每个任务, 取得能量收益的最大化。

对于顺序指令预取, 不同的预取度  $N$  对于系统的性能和能耗都有不同的影响。矩阵元素  $P[i][j]$  和  $E[i][j]$  分别存储任务  $T_i (1 \leq i \leq N_T)$  在预取度  $j (0 \leq j \leq N_p)$  下的最坏情况下的性能和能量消耗。 $P\_D[i] (1 \leq i \leq N_T)$  表示任务  $T_i$  在结合性能和能耗优化后, 在执行时候的应该采用的预取度。非关键子路径构建算法如下:

算法 1: 非关键子路径构建。

输入: 任务 TG,  $P[i][j]$ ,  $P\_D[i]$  (初始化为 0)。

输出: 优化后的 WCRT, 非关键子路径集合 NPS,  $P\_D[i]$  (性能优化后的预取度)。

(1) 通过  $P[i][j]$  把任务  $T_i$  取得最好的 WCET 的预取度  $k$  赋值给  $P\_D[i]$ ,  $\forall j \in [0, N_p], p[i][k] \leq p[i][j]$ , 同时给 TG 中与  $T_i$  相对应的边赋最优的 WCET 值。

(2) 根据实时系统的 TG,  $P\_D[i]$  和  $P[i][j]$  计算最优情况下支持指令预取的关键路径  $P$  和 WCRT。

(3) 对所有非关键路径上的任务, 按拓扑序列构建任务队列 TQ。

(4) 从 TQ 队头选择一个任务  $T_j$ , 以  $T_j$  直接前驱  $T_k$  (关键任务) 为头构建非关键子路径队列 NP。从  $T_j$  开始, 依次在 TQ 中寻找每个非关键任务的直接后继  $T_m$ , 并把  $T_m$  从 TQ 中删除, 加入到 NP 队尾, 直到任务  $T_m$  的直接后继为一关键任务  $T_c$ , 把  $T_c$  也加入到 NP 中, 把 NP 加入到 NPS 中。

(5) 重复过程 4, 构建非关键子路径集合  $NPS = \{NP_1, NP_2, \dots, NP_n\}$ , 直到 TQ 为空。

图 3 中的  $T_2, T_4, T_6$  是非关键路径上的任务, 按拓扑序列  $TQ = \{T_2, T_4, T_6\}$ 。第 1 个非关键任务是  $T_2$ , 它的直接前驱  $T_1$  作为非关键子路径队列  $NP_1$  的队头,  $T_2$  的直接后继是  $T_4$ , 把  $T_4$  从 TQ 中删除, 插入到  $NP_1$  队尾,  $T_4$  的直接后继是  $T_6$ , 它也被插入到  $NP_1$  队尾。  $T_6$  的后继是关键任务  $T_{end}$ , 把  $T_{end}$  也加入  $NP_1$  队尾, 此时  $NP_1$  是一条非关键子路径。这时非关键任务队列 TQ 为空, 算法结束。至此图 3 所示的 TG 中存在一条

非关键子路径  $NP_1 = \{T_1, T_2, T_4, T_6, T_{end}\}$ , 非关键路径长度是 700。

对于一条非关键子路径队列 NP, 它可以被延长的时间片是从队头到队尾之间关键路径长度与非关键路径长度的差。

在非关键子路径上的任务, 文中用穷举法在允许被延长的时间内, 调整实时任务的预取度, 优化能耗,  $P\_E[i] (1 \leq i \leq N_T)$  表示任务  $T_i$  在结合性能和能耗优化后, 在执行时应该采用的预取度。优化算法如下:

算法 2: 优化 WCEC。

输入: 实时系统任务图 TG, 非关键子路径集合  $NPS = \{NP_1, NP_2, \dots, NP_n\}, P[i][j], E[i][j], P\_D[i]$ 。

输出: 优化后的 WCEC,  $P\_E[i]$  (结合性能和能耗优化后的预取度)。

(1) 对每一个非关键子路径  $NP_i = \{T_{i1}, T_{i2}, \dots, T_{im}\}$ , 计算  $NP_i$  队列中两个关键任务  $T_{i1}$  和  $T_{im}$  之间的关键路径的长度  $t_i$ , 它是两个关键任务最早开始时间的差值。

(2) 计算最优性能下非关键子任务的能量消耗:

$$E_i = E[i_2][P\_D[i_2]] + E[i_3][P\_D[i_3]] + \dots + E[i_{n-1}][P\_D[i_{n-1}]]$$

(3) 用穷举法求出在时间不超出  $t_i$  情况下的取得最优能耗的指令预取度。

```
FOR(  $k_{i_1}$  from 0 to  $N^p$  ) DO
    FOR(  $k_{i_2}$  from 0 to  $N^p$  ) DO
        .....
        FOR(  $k_{i_{n-1}}$  from 0 to  $N^p$  ) DO
            IF(  $P[i_1][P\_D[i_1]] + P[i_2][k_{i_2}] + \dots + P[i_{n-1}][k_{i_{n-1}}] \leq t_i$  ) THEN
                IF(  $E[i_2][k_{i_2}] + E[i_3][k_{i_3}] + \dots + E[i_{n-1}][k_{i_{n-1}}] \leq E_i$  ) THEN
                     $E_i = E[i_2][k_{i_2}] + \dots + E[i_{n-1}][k_{i_{n-1}}]$ 
                     $P\_E[i_2] = k_{i_2}; \dots P\_E[i_{n-1}] = k_{i_{n-1}};$ 
                END IF
            END IF
        END FOR
    END FOR
END FOR
.....
END FOR
END FOR
```

(4) 重复过程 1 到 3, 直到 NPS 为空。

(5) 用式(9) 计算实时系统优化后的最坏情况下的能量消耗。

$$WCEC = \sum_{i=1}^{N_T} E[i][P\_E[i]] \quad (9)$$

## 3 实验评估

这部分评估文中支持指令预取的实时系统 WCRT



和 WCEC 优化算法。

### 3.1 评估环境

嵌入式多核假设有 6 个同构的处理器核,对于每一个核,有 5 阶段流水,顺序执行。L2 缓存是 8 KB,缓存行是 32 B,4 路组关联,有 64 组。L1 指令和数据缓存是 512 B,缓存行大小是 16 B,直接映射。L1 缓存的命中延迟是 1 circle,缺失延迟是 6 circles。L2 缓存的缺失延迟是 30 circles,预取度  $N$  的取值范围是从 0 到 5。

实验运行在 Intel i5-3230 机器上,4 GB 内存,运行 Ubuntu Linux 8.04 操作系统。

文中子任务的 WCET 通过支持指令预取的缓存 WCET 分析工具<sup>[13]</sup>测得。采用文献[12]中的能量消耗计算模型。芯片的工艺是 32 nm,频率是 1.0GH,总的缓存访问和缓存缺失可以通过静态分析获得。不同工艺下每次访问的动态能耗和静态能耗是 CACTIS<sup>[14]</sup>的测试结果。硬件预取器通过一些硬件表来实现<sup>[3]</sup>,它的能耗可以被精确模拟。这里假设有一个 1 024-bit 的指令预取缓冲队列,在 32 nm 工艺下,预取器消耗 0.063 mW 的静态能耗,每次读的能耗是 0.002 nJ,每次写的能耗是 0.003 nJ。

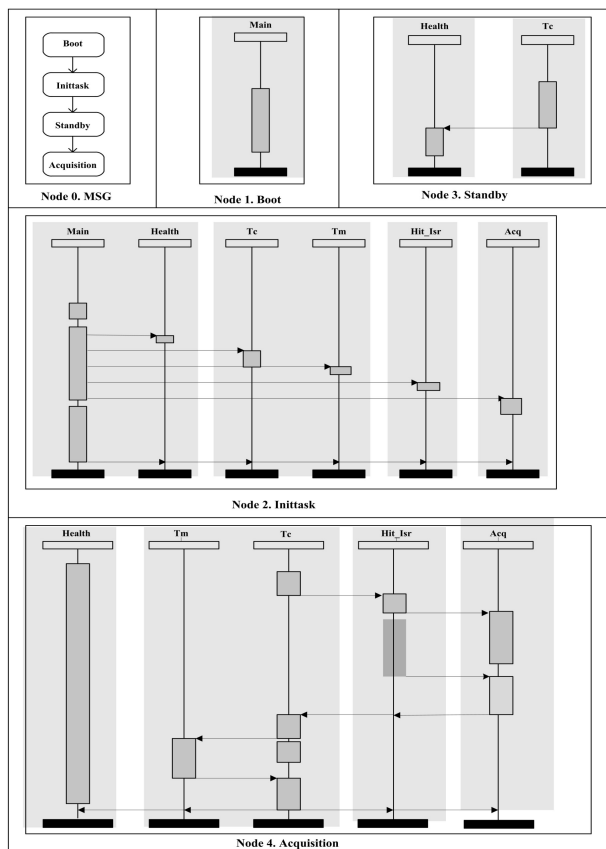


图4 DEBIE 系统的 MSG

文中使用的 benchmark 是 DEBIE 空间碎片探测器系统<sup>[15]</sup>,根据 DEBIE 的系统状态和功能,文献[16]对系统进行了并行化处理,图4 是系统的 MSG 图。当

DEBIE 的所有实时任务被映射到了核 1,2,3 到 4,分配给每个核的 L2 缓存组是由它上面运行的最大的任务所决定的,最大任务的 WCET 最小的 L2 组被分配给了该处理器核,表 1 是分配给核 1,2,3 和 4 的 L2 组。余下的组被核 5 和核 6 共享。

表1 L2 缓存划分结果

	核 1	核 2	核 3	核 4
组 (Sets)	8	8	16	8

### 3.2 实验结果

#### 3.2.1 优化算法对最坏情况下的性能的优化

为了分析文中的优化算法对于最坏情况下系统性能的影响,图 5 给出的是 DEBIE 系统在文中算法优化后的 WCRT 和不同预取度下的 WCRT 的比值,比值越小则表明性能提高越明显。

从图 5 可以看出,优化后的支持指令预取的最坏情况下的性能比没有预取的最坏情况下 ( $N$  为 0) 的性能提高了 57.7%,与预取度是 5 的比值接近于 1。这是由于整个系统的指令较多,需要进行的数据计算很少,指令预取度越大,指令预取提高最坏情况下的性能越明显,此时优化的效果反而不明显。

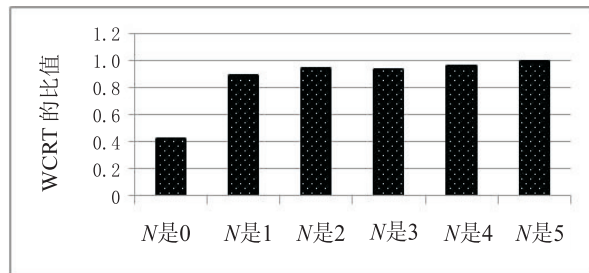


图5 不同预取度下 WCRT 的比值

#### 3.2.2 指令预取对最坏情况能耗的影响

为了分析指令预取度对于最坏情况下系统性能的影响,图 6 比较了系统在不同预取度下的支持指令预取和不支持指令预取的 WCEC,结果被 WCEC ( $N$  是 0) 归一化了。

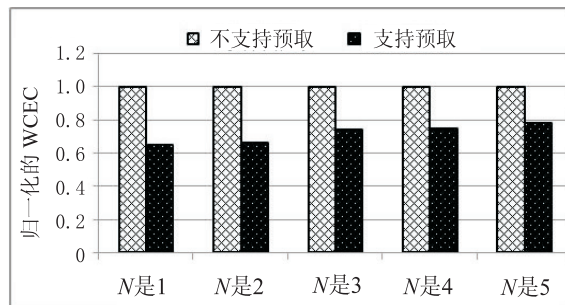


图6 归一化的 WCEC

从图 6 可以看出,不同预取度下 DEBIE 系统在最坏情况下的能量消耗都被减少了。预取节省的平均能耗为 28.3%,预取度越大,节能效果反而不好,原因是预取度越大,消耗的动态能耗比重越大,预取减少的总

的能量消耗也就越少。

### 3.2.3 最坏情况下能量的优化

为了分析文中的优化算法对于最坏情况下能量消耗的影响,图7比较了 DEBIE 系统在保证系统性能最优的情况下,文中算法优化后的 WCEC 和不同预取度下的 WCEC 比值,比值越小则表明能耗降低越明显。

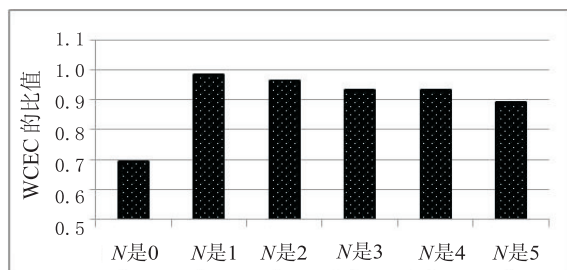


图7 不同预取度下 WCEC 的比值

从图7可以看出,优化后的 WCEC 比不支持指令预取(N是0)的最坏情况下的 WCEC 提高30.5%,但此时不是性能最优。当预取度 N 是5时,性能最优,这时优化后的最坏情况下的能量消耗减少了10.8%。能量优化的效率与程序的访存次数和动态能耗在总能耗中的比重有关系。预取度越大,访存次数也就越多,动态能耗消耗得也就越多,因而能量优化的效果越明显。

## 4 结束语

对于性能和能耗要求很高的多核实时系统,提出支持指令预取的 WCRT 和 WCEC 优化算法,通过调整子任务的预取度在保证性能最优的情况下优化最坏情况下的能耗。实验通过对 DEBIE 系统的分析,表明优化算法是有效的,系统中子任务的并行程度高,指令访存次数越多,优化的效果越明显。

今后希望能够研究数据预取对于实时系统 WCEC 的影响,并提出支持指令预取和数据预取的实时系统最坏情况下的性能和能耗优化方法。

### 参考文献:

- [1] 甘志华,张铭泉,古志民,等.基于任务映射与缓存划分的 WCRT 优化方法[J].北京理工大学学报,2018,38(3):272-278.
- [2] SIEH V, BURLACU R, HÖNIG T, et al. An end-to-end Toolchain: from automated cost modeling to static WCET and WCEC analysis[C]//IEEE international symposium on real-time distributed computing. Toronto, Canada: IEEE, 2017:158-167.
- [3] TANG J, LIU S, GU Z, et al. Prefetching in embedded mobile systems can be energy-efficient[J]. IEEE Computer Architecture Letters, 2011, 10(1): 8-11.
- [4] LIANG Yun, DING Huping, TULIKA M, et al. Timing analysis of concurrent programs running on shared cache multi-cores[J]. Real-Time System, 2012, 48(6): 638-680.
- [5] DING H, LIANG Y, MITRA T. Shared cache aware task mapping for WCRT minimization[C]//Design automation conference. Austin, Texas: IEEE, 2013.
- [6] YAN J, ZHANG W. WCET analysis of instruction caches with prefetching[J]. ACM SIGPLAN Notices, 2007, 42(7): 175-184.
- [7] DING Y, YAN J, ZHANG W. Optimizing instruction prefetching to improve worst-case performance for real-time applications[J]. Journal of Computing Science & Engineering, 2009, 3(1): 59-71.
- [8] GU J, GUO H. An energy efficient instruction prefetching scheme for embedded processors[M]//Ubiquitous computing and multimedia applications. Berlin, Heidelberg: Springer, 2010: 73-88.
- [9] GRAN R, SEGARRA J, RODRÍGUEZ C, et al. Optimizing a combined WCET-WCEC problem in instruction fetching for real-time systems[J]. Journal of Systems Architecture, 2013, 59(9): 667-678.
- [10] 王恩东,倪 璠,陈继承,等.一种面向实时系统的程序基本块指令预取技术[J].软件学报,2016,27(9):2426-2442.
- [11] YU Chenjie, PETROV P. Off-chip memory bandwidth minimization through cache partitioning for multi-core platforms[C]//47th ACM/EDAC/IEEE design automation conference. Anaheim, CA: IEEE, 2000: 132-137.
- [12] ZHANG Chuanjun, VAHID F, LYSECKY R. A self-tuning cache architecture for embedded systems[C]//Proc of design, automation and test in Europe, DATE. Paris, France: IEEE, 2004: 142-147.
- [13] 安立奎,韩丽艳.支持指令预取的多核缓存 WCET 分析方法[J].计算机工程,2018,44(10):85-94.
- [14] THOZIYOOR S, MURALIMANOHAR N, AHN J H, et al. CACTI 5.1[J]. HP Laboratories, 2008, 2(11): 111-117.
- [15] KUITUNEN J, DROLSHAGEN G, MCDONNELL J A M, et al. DEBIE - first standard in-situ debris monitoring instrument[C]//Proceedings of the third European conference on space debris. Darmstadt, Germany: European Space Agency, 2001: 185-190.
- [16] 康少华,古志民,付引霞,等.空间碎片探测软件的并行化及 WCRT 分析[J].计算机应用研究,2015,32(11):3283-3286.