多核实时线程间干扰分析及WCET估值

陈芳园[1],张冬松[1,2],王志英[1]
(1.国防科学技术大学计算机学院,湖南长沙 410073; 2.国防科学技术大学并行与分布处理国家重点实验室,湖南长沙 410073)

摘 要: 在共享Cache的多核处理器中,线程在共享Cache中的指令可能被其他并行线程的指令替换,从而导致了线程间在共享Cache上的干扰.多核结构下WCET估值需要考虑并行线程间在共享Cache上的干扰.针对当前典型的共享Cache和共享总线的多核结构,本文提出了一种迭代的WCET估值分析方法.考虑共享总线对共享Cache访问的时序影响,基于该时序分析线程间在共享Cache上的干扰,得到较精确的WCET估值.理论分析证明了该方法的有效性,实验结果表明本文的分析方法较之当前的两种方法分别可以提高21%和14%的精确度.

关键词: 多核体系结构; 共享Cache; 共享总线; 干扰; WCET
中图分类号: TP316 **文献标识码:** A **文章编号:** 0372-2112 (2012) 07-1372-07
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.07.015

Inter-Thread Interference Analysis for Real-Time WCET Estimations of Multi-Core Architectures

CHEN Fang-yuan[1], ZHANG Dong-song[1,2], WANG Zhi-ying[1]
(1. *College of Computer, National University of Defense Technology, Changsha, Hunan* 410073, *China*;
2. *National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha, Hunan* 410073, *China*)

Abstract: In a shared-cache multi-core architecture one thread may interfere with a second thread if the second one tries to access the shared cache simultaneously. Consequently, this causes the eviction of the second thread instructions. To track this challenge, designers need to consider runtime inter-thread interference while analyzing WCET of a real-time application on multi-core architectures. This paper proposes an iterative approach for WCET estimation which considers the circular dependence between shared bus and the runtime inter-thread interference in shared cache. Our approach analyzes inter-thread interference in shared cache based on access timings, which combines static analysis and dynamic timing estimation. The iterative method presented can improve the tightness of WCET estimation by refining estimations of inter-thread interference. Our experiments demonstrate that the proposed approach can reasonably estimate inter-thread interference in shared caches and improve the tightness of WCET estimation by an average of 21% and 14%, compared with the estimations in literatures.

Key words: multi-core architecture; shared cache; shared bus; interference; WCET

1 引言

随着半导体工艺的进步,片上多核处理器已经逐渐取代单核处理器,成为市场的主流.而高端实时应用对高性能需求的增长使得多核处理器也开始应用于实时系统以获得更高的性能吞吐量比[1].实时系统要求事先获知任务的最坏情况下执行时间(Worst Case Execution Time, WCET)[2]以进行可调度性分析.相对于单核处理器,多核处理器由于共享资源的存在很难获得任务的WCET估值,例如,共享Cache、片上互连等.并行任务对共享资源的竞争访问使得任务在访问共享资源时会受到干扰.一个任务的执行时间可能会由于并行任务而改变,从而使得任务的WCET不再容易确定.因此,在多核体系结构下,共享资源的模拟和分析极大地影响着任务WCET估值的有效性和精确性.

当前多核结构通常采用共享Cache来提高处理器性能以及Cache利用率[3,4],处理器核通过共享片上互连访问共享Cache.共享Cache会导致并行线程间干扰[5,6],运行在一个核上的任务可能会破坏运行在另一个核上的并行任务在共享Cache中的一些数据.同时,

收稿日期:2011-08-01;修回日期:2012-02-21
基金项目:国家教育部博士点基金(No.20104307110005);国防科学技术大学优秀研究生创新资助(No.B100601);湖南省研究生科研创新项目资助(No.CX2010B026)

多个处理器核通过片上互连访问共享 Cache 时的竞争使用也使得线程的执行可能受到干扰而延迟.为了获得多核实时系统中线程的有效 WCET 估值,必须考虑并行任务间在共享 Cache 和互连上的干扰.

多核中线程在共享资源上的竞争干扰已经吸引了越来越多的研究者注意.文献[7]针对共享 Cache 和共享总线的多核结构,对可能访问共享总线的存储访问总是考虑最大的等待时间.这种方法消除了总线对 WCET 分析的影响,但在任务实际执行过程中,Cache 未命中对互连的访问请求会产生不同的延迟.文献[7]这种考虑最大互连竞争延迟的方法过于保守,WCET 估值精度较低.文献[8]根据总线的调度确定 Cache 未命中情形下不同的访问延迟,但是该研究并未考虑共享 Cache 的多核结构,未对共享 Cache 和总线访问的时序交互影响进行分析.文献[9]采用迭代方法进行 WCET 分析,提出只有执行周期重叠的任务之间才会产生干扰,通过这一限制来提高 WCET 估值精度.

综上,当前研究主要存在两个方面的问题:(1)当前研究均从程序逻辑和 Cache 结构分析干扰,都基于一个简单假设:无论何种执行情况,并行线程中映射到同一 Cache 行的指令总是会产生干扰.这种假设过于保守,使得 WCET 估值精度较低;(2)当前研究对于互连也只是考虑了互连访问对线程执行周期的影响,未分析互连与共享 Cache 干扰之间的相互影响.

本文在前期工作中[10]针对共享 Cache 的多核结构分析了取指执行时序对线程间在共享 Cache 上干扰的影响.但是在考虑互连的多核结构中,共享 Cache 干扰不再仅仅只与取指执行时序相关,也与片上互连密切相关.本文针对当前典型的共享 Cache 和共享总线的多核结构,提出了一种迭代的 WCET 估值分析方法,考虑共享总线对共享 Cache 访问的时序影响,更合理地判断线程间在共享 Cache 上的干扰,从而得到较精确的 WCET 估值.

2　系统模型

为了使研究更具有针对性和实用性,本文考虑当前常见的共享 Cache 的存储层次,处理器核之间通过共享总线的互连结构进行通信.本文考虑具有 m 个处理器核的同构多核处理器,$\{core_1, core_2, \cdots, core_m\}$.每个处理器核具有独立的一级 Cache L1,处理器核之间共享二级 Cache L2.在后续分析中若无特殊说明私有 Cache L1 和共享 Cache L2 均指指令 Cache.本文只针对指令 Cache 展开分析.假设数据 Cache 访问不会对指令 Cache 访问产生影响.

本文所采用的片上互连结构为共享总线.为了保证可预测性,需要设定仲裁机制保证任务在执行过程中具有可预测的访问延迟.目前常用的仲裁机制中时分多址 TDMA 是一个确定性的仲裁逻辑,有固定的总线请求时间槽.这种确定性的时间槽提供了较好的可预测性,逐渐被实时系统所采用.因此本文针对基于 TDMA 的共享总线展开分析.

共享总线和共享 Cache 干扰均与指令执行时间密切相关:(1)对于多核共享总线,在进行总线仲裁时需要知道处理器核访问总线的时间,才能确定请求的处理时间以及延迟;(2)对于多核中的共享 Cache,核间共享 Cache 的干扰主要取决于下列几个因素:线程访问共享 Cache 的指令地址、指令映射的 Cache 行以及指令何时执行.其中"指令何时执行"因素对应着指令访问共享 Cache 的时间.共享总线和共享 Cache 均与指令执行密切相关.根据指令的执行顺序以及指令间在时间上的先后顺序,本文将这些时间信息称为时序.基于时序分析共享总线访问和并行线程间在共享 Cache 上的干扰.表 1 列出了系统模型中的一些重要参数.

表 1　共享总线和共享 Cache 访问参数

Parameters	Description
$l1\_access\_overhead$	the overhead of L1 access
$bus\_req\_overhead$	the overhead of bus request
c	The time length enough to deal a bus request

线程在执行过程中访问私有 Cache L1 取指,本文将这个访问时序称为取指执行时序(Instruction Fetch Timing,IFT).设指令 I 取指的开始执行时序为:$t^{start}_{IF(I)}$.若 I 在 L1 中未命中(L1 *miss*),处理器核通过总线访问共享 Cache L2,总线的访问时序(Bus Access Timing,BAT)为:

$$t^{start}_{BAT(I)} = t^{start}_{IF(I)} + l1\_access\_overhead \tag{1}$$

本文在 3.1 节中根据仲裁和访问时序计算总线请求被处理的等待延迟 *wait*.此时可以得到总线请求开始处理的时序(Bus Request-Dealt Timing,BRDT):

$$t^{start}_{BRDT(I)} = t^{start}_{BAT(I)} + wait \tag{2}$$

处理器核开始访问共享 Cache 的时序(Shared Cache Access Timing,SCAT)为:

$$t^{start}_{SCAT(I)} = t^{start}_{BRDT(I)} + bus\_req\_overhead \tag{3}$$

本文采用静态方法进行共享总线和共享 Cache 分析,而指令的执行时序只有在动态执行时才可以确定,静态分析无法获得指令确切的执行时序.但是通过静态分析可以获得指令执行的上界和下界时序:最早执行时序和最晚执行时序.以 SCAT 为例,本文通过静态分析可以获得最早和最晚 SCAT,即 $earliest(t^{start}_{SCAT(I)})$ 和 $latest(t^{start}_{SCAT(I)})$.由最早执行时序和最晚执行时序确定的时间段称为时序范畴,例如,[$earliest(t^{start}_{SCAT(I)})$, *latest*

($t^{start}_{SCAT(I)}$)].因此,本文基于 BAT、BRDT 和 SCAT 的时序范畴进行共享总线和共享 Cache 分析.

3 基于共享总线和共享 Cache 的多核 WCET 分析

针对共享总线和共享 Cache 的相互关联和相互依赖性,本文提出了一种迭代的 WCET 分析方法,不仅考虑总线对线程执行周期的影响,而且进一步考虑总线对共享 Cache 访问时序的影响,从而更细粒度的分析线程间在共享 Cache 上的干扰.分析流程如图 1 所示.

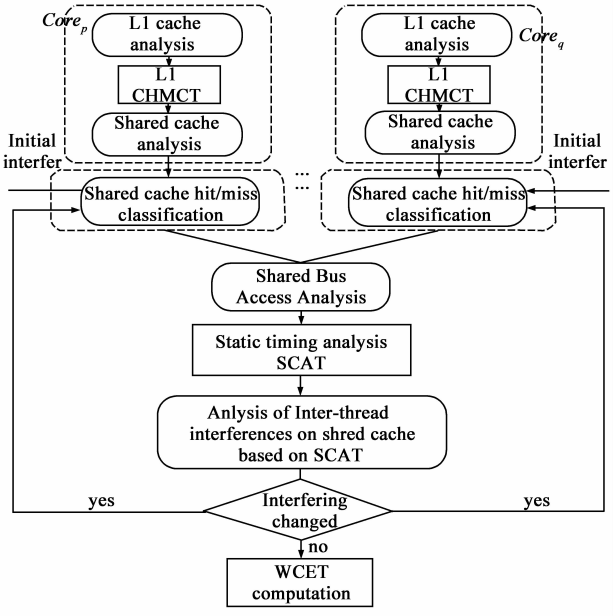

图1 基于共享总线和共享Cache的多核WCET分析

首先对每个处理器核进行私有 Cache L1 分析,获得线程在私有 Cache L1 中的 hit/miss 分类.本文针对 L1 *miss* 和 L1 *nc* 这类可能产生共享 Cache 访问的指令进行后续的共享总线和共享 Cache 分析.在进入迭代分析过程之前,本文先对线程进行了共享 Cache 独立分析,获得线程在不考线程间干扰时的共享 Cache hit/miss 分类.基于这个 *hit/miss* 分类进行后续迭代分析,考虑共享总线和共享 Cache 之间的相互影响.

迭代的第一步是根据当前的干扰状态和共享 Cache hit/miss 分类来确定线程在考虑并行线程干扰时的 hit/miss 分类.本文将线程间可能的干扰状态分为两种:(1)非干扰状态"*non-in*",在此干扰状态下所有的指令保持原有的分类不变;(2)干扰状态"*in*",本文设置在此干扰状态下 *hit* 变为 *nc*,其他的分类保持不变.在第一次迭代时,本文对干扰状态进行了保守的、安全的初始化假设:所有的指令均受到并行线程的干扰,即干扰状态均设置为"*in*".

基于当前的共享 Cache hit/miss 分类进行共享总线分析,获得总线访问延迟(详见 3.1 节),根据式(2)和式(3),在考虑总线影响下分析共享 Cache 访问时序 SCAT 范畴.然后根据非干扰状态的充分不必要条件,基于当前的共享 Cache 访问时序 SCAT 范畴确定非干扰状态,从而更合理地估计线程间在共享 Cache 上的干扰(详见 3.2 节).

若当前迭代得到的干扰状态较之前一次迭代发生改变,则再次进行迭代分析以获得更精确的干扰状态.若干扰状态未发生改变,则结束迭代,基于最终干扰状态进行 WCET 估值分析(详见 3.3 节).这种迭代的分析方法可以更好的、更合理地分析线程间的干扰状态,获得更精确的 WCET 估值.本文通过定理 3 证明提出的迭代分析方法具有收敛性.

3.1 共享总线分析

处理器核在私有 Cache L1 访问未命中时(L1 *miss*)通过共享总线访问共享 Cache.共享总线访问由于发生时刻不同其等待延迟也不尽相同.本文从最早和最晚取指执行时序入手,分析共享总线等待延迟以获得最早和最晚共享 Cache 访问时序,从而进行共享 Cache 干扰分析.

本文针对 TDMA 总线展开分析.在 TDMA 仲裁机制下,总线被分成多个时间槽,以轮转形式分配给每个处理器核.处理器核在其分配的时间槽内才可以访问总线.TDMA 总线调度由下列递归关系表示[9]:$CS_k^{(i+1)} = CS_k^{(i)} + B$, $CS_k^{(0)} = A_k$.其中 $CS_k^{(i)}$ 是总线调度在第 i 次被分配给处理器核 $core_k$ 的开始时间,$B = m \times S_l$, m 是处理器核数,S_l 是分配给每个处理器核的时间槽大小,A_k 是 $core_k$ 上第一个时间槽的开始时间.

L1 *miss* 指令 I 向总线发出访问共享 Cache 的请求.通过流水线分析可以获得处理器核执行取指操作的最早和最晚执行时序,即 $earliest(t^{start}_{IF(I)})$ 和 $latest(t^{start}_{IF(I)})$(详见本文的前期工作[10]).根据式(1)得到总线请求访问时序范畴为:

$$earliest(t^{start}_{BAT(I)}) = earliest(t^{start}_{IF(I)}) + l1\_access\_overhead$$

$$latest(t^{start}_{BAT(I)}) = latest(t^{start}_{IF(I)}) + l1\_access\_overhead$$

来自处理器核 $core_k$ 的总线请求只有在其分配的时间槽内才可以被处理,本文根据总线请求访问时序计算总线请求处理时序,即根据总线的请求时序获得总线真正执行请求处理的时序 BRDT.对于 $t^{start}_{BAT(I)} \in [earliest(t^{start}_{BAT(I)}), latest(t^{start}_{BAT(I)})]$,设总线的访问请求等待时间上界为常量 c(一般设置为总线请求的最大时间开销),存在三种情况:

case1 若 $0 \leqslant t^{start}_{BAT(I)} \bmod B < A_k$,则该请求发生在 $core_k$ 的第 i 个总线槽$[CS_k^i, CS_k^i + S_l]$之前,必须等待该

处理器对应的时间槽到达才可以处理,等待延迟为: $wait = CS_k^i - t_{BAT(I)}^{start}$.

case2 若 $A_k \leqslant (t_{BAT(I)}^{start} \bmod B) \leqslant A_k + S_l - c$,则该请求在当前第 i 个总线槽内,且当前总线槽的剩余时间可以处理一个请求,故该请求可以立即被处理,即 $t_{BRDT(I)}^{start} = t_{BAT(I)}^{start}$,等待延迟为: $wait = 0$.

case3 若 $(A_k + S_l - c) < (t_{BAT(I)}^{start} \bmod B) \leqslant (B-1)$,则该请求必须要等到第 $i+1$ 个总线槽才开始被处理,等待延迟为: $wait = CS_k^{i+1} - t_{BAT(I)}^{start}$.

对于 $earliest(t_{BAT(I)}^{start})$ 和 $latest(t_{BAT(I)}^{start})$,可以得到相应的 BRDT,本文将其标记为 $BRDT(earliest(t_{BAT(I)}^{start}))$ 和 $BRDT(latest(t_{BAT(I)}^{start}))$:

$$BRDT(earliest(t_{BAT(I)}^{start})) = earliest(t_{BAT(I)}^{start}) + wait \quad (4)$$

$$BRDT(latest(t_{BAT(I)}^{start})) = latest(t_{BAT(I)}^{start}) + wait \quad (5)$$

引理 1 和定理 1 证明由式(4)和式(5)计算得到的 BRDT 即为最早和最晚 BRDT.

引理 1 BRDT 保持 BAT 的 $\leqslant$ 不变

证明 BRDT 保持 BAT 的 $\leqslant$ 不变,即对于指令 I(L1 *miss*)的总线访问请求,若 $t_{BAT(I)}^{start} \leqslant t_{BAT(I)}^{start\ *}$,则 $t_{BRDT(I)}^{start} \leqslant t_{BRDT(I)}^{start\ *}$.

本文基于上述三种可能发生的情况给予讨论.假设 $t_{BAT(I)}^{start\ *}$ 位于总线的第 i 个时间段内, $core_k$ 当前分配的总线槽为 CS_k^i.

(1)假设在 $t_{BAT(I)}^{start\ *}$ 时刻发生的请求在总线中的处理属于 case1,即 $0 \leqslant t_{BAT(I)}^{start\ *} \bmod B < A_k$,此时 BRDT 为 $t_{BRDT(I)}^{start\ *} = \lfloor t_{BAT(I)}^{start\ *}/B \rfloor \times B + A_k = CS_k^i$.

(*a*)若 $t_{BAT(I)}^{start}$ 和 $t_{BAT(I)}^{start\ *}$ 在同一个时间段内,则 $\lfloor t_{BAT(I)}^{start}/B \rfloor = \lfloor t_{BAT(I)}^{start\ *}/B \rfloor$ 且 $(t_{BAT(I)}^{start} \bmod B) \leqslant (t_{BAT(I)}^{start\ *} \bmod B) < A_k$.故此时 $t_{BAT(I)}^{start}$ 满足 case1,即 $t_{BRDT(I)}^{start} = CS_k^i$.此时 $t_{BRDT(I)}^{start} \leqslant t_{BRDT(I)}^{start\ *}$;

(*b*)若 $t_{BAT(I)}^{start}$ 和 $t_{BAT(I)}^{start\ *}$ 不在同一个时间段内,假设 $t_{BAT(I)}^{start}$ 位于第 $j(j \leqslant i-1)$ 个时间段内,则 $\lfloor t_{BAT(I)}^{start}/B \rfloor = \lfloor t_{BAT(I)}^{start\ *}/B \rfloor - (i-j)$.此时 $t_{BAT(I)}^{start}$ 可能出现总线处理的所有三种情况,无论何种情况,请求 r_1 在总线的执行时刻 BRDT 存在: $t_{BRDT(I)}^{start} \leqslant CS_k^i$,而 $t_{BRDT(I)}^{start\ *} = CS_k^i$,故 $t_{BRDT(I)}^{start} \leqslant t_{BRDT(I)}^{start\ *}$;

(2)假设在 $t_{BAT(I)}^{start\ *}$ 时刻发生的请求在总线中的处理属于 case2,即 $A_k \leqslant (t_{BAT(I)}^{start\ *} \bmod B) \leqslant A_k + S_l - c$.此时其 BRDT 为: $t_{BRDT(I)}^{start\ *} = t_{BAT(I)}^{start\ *}$.证明同(1).

(3)假设在 $t_{BAT(I)}^{start\ *}$ 时刻发生的请求在总线中的处理属于 case3,即 $A_k + S_l - c < (t_{BAT(I)}^{start\ *} \bmod B) \leqslant (B-1)$.此时 $t_{BRDT(I)}^{start\ *} = (\lfloor t_{BAT(I)}^{start\ *}/B \rfloor + 1) \times B + A_k = CS_k^{i+1}$.同(1).

综上,命题得证. □

定理 1 根据最早(最晚)BAT 计算得到的 BRDT 为最早(最晚)BRDT.

证明 对于指令 I,其可能的任意时刻总线请求 $t_{BAT(I)}^{start}$ 满足: $earliest(t_{BAT(I)}^{start}) \leqslant t_{BAT(I)}^{start} \leqslant latest(t_{BAT(I)}^{start})$.

(1)若 $t_{BAT(I)}^{start} \leqslant latest(t_{BAT(I)}^{start})$:由引理 1 可知, $t_{BAT(I)}^{start}$ 时刻的请求在总线上的处理一定晚于总线对 $latest(t_{BAT(I)}^{start})$ 时刻发生的请求处理,即 $BRDT(t_{BAT(I)}^{start}) \leqslant BRDT(latest(t_{BAT(I)}^{start}))$.由 $t_{BAT(I)}^{start}$ 的任意性可知,式(5)中根据 $latest(t_{BAT(I)}^{start})$ 以及相应的总线访问延迟获得的总线请求处理时序 $BRDT(latest(t_{BAT(I)}^{start}))$ 为指令 I 的总线请求最晚处理时序,即 $BRDT(latest(t_{BAT(I)}^{start})) = latest(t_{BRDT(I)}^{start})$.

(2)若 $earliest(t_{BAT(I)}^{start}) \leqslant t_{BAT(I)}^{start}$:证明同(1).

由(1)、(2)可知,对于指令 I 任意时刻 $t_{BAT(r)}^{start}$ 的请求, $earliest(t_{BAT(I)}^{start}) \leqslant t_{BAT(r)}^{start} \leqslant latest(t_{BAT(I)}^{start})$,总线对该请求的处理一定满足 $BRDT(earliest(t_{BAT(I)}^{start})) \leqslant BRDT(t_{BRDT(r)}^{start}) \leqslant BRDT(latest(t_{BAT(I)}^{start}))$.即证. □

因此,式(4)、(5)重写为:

$$earliest(t_{BRDT(I)}^{start}) = earliest(t_{BAT(I)}^{start}) + wait \quad (6)$$

$$latest(t_{BRDT(I)}^{start}) = latest(t_{BAT(I)}^{start}) + wait \quad (7)$$

3.2 共享 Cache 分析

基于式(6)和(7)可以得到最早和最晚 BRDT,此时总线开始进行仲裁以及地址译码(所需时间为 $bus\_req\_overhead$),然后向共享 Cache 发出访问请求.根据式(3)可得共享 Cache 最早和最晚访问时序 SCAT,即

$$earliest(t_{SCAT(I)}^{start}) = earliest(t_{BRDT(I)}^{start}) + bus\_req\_overhead$$

$$latest(t_{SCAT(I)}^{start}) = latest(t_{BRDT(I)}^{start}) + bus\_req\_overhead$$

本文基于得到的 SCAT 进行共享 Cache 分析.在前期工作中,我们针对共享 Cache 的多核结构提出了基于 IFT 的共享 Cache 干扰分析方法[10],提出了非干扰状态充分不必要条件,基于该条件排除线程间在共享 Cache 中的非干扰状态.本文提出了基于 SCAT 的共享 Cache 干扰分析方法.

定理 2 对于任意一条指令 aj(L1 *miss* 且 L2 *hit*),设 aj 在共享 Cache 中的 *hit* 由指令 ai 对共享 Cache 的访问局部性原理造成.则并行线程中映射到同一共享 Cache 行的指令 bm(L1 *miss*)对 aj 的非干扰状态存在一个充分不必要条件:

$$earliest(t^{start}_{SCAT(bm)}) > latest(t^{start}_{SCAT(aj)}) \parallel latest(t^{start}_{SCAT(bm)}) < earliest(t^{start}_{SCAT(ai)}) \quad (8)$$

证明 因篇幅所限,证明过程此处不再赘述. □

3.3 迭代

本文提出的迭代分析方法通过迭代循环,采用 3.2 节的干扰分析方法,以获得一个最终的、合理的、紧凑的干扰状态.在迭代之前,由于未知任何执行信息,首先对指令的干扰状态进行初始化设置,以获得一个初始的、保守的、最大可能的执行时序范畴:对所有的指令考虑其可能的、最大执行时序,干扰状态均设置为 *in*(受到干扰).

迭代过程中,基于当前的干扰状态分析指令的最早和最晚 SCAT(见算法 1).首先根据干扰状态设置指令取指操作延迟(见第 1 至 8 行):对于 L1 *miss* & L2 *hit* 指令,若其干扰状态为 *in*,则说明该指令在共享 Cache 中受到干扰,设置取指操作延迟最小为 $l1\_miss\_lat+1$,最大为 $l1\_miss\_lat+l2\_miss\_lat+1$.然后,基于文献[13,14]所提出的最早开始执行时间分析算法 *EarliestTime* 和最晚开始执行时间分析算法 *LatestTime*,可以获得未考虑总线延迟的执行取指执行时序.基于该取指执行时序,进行共享总线分析(见第 10 至 19 行).最后根据更新后的延迟设置,重新分析共享 Cache 访问时序(见第 20 行).

算法 1: Estimate_SCAT

```
1: for any instruction inst do
2:   if inst is L1 miss & L2 hit then
3:     if(interfering_status == in)
4:       lat.lo = l1_miss_lat + 1;
5:       lat.hi = l1_miss_lat + l2_miss_lat + 1;
6:     else
7:       lat.lo = lat.hi = l1_miss_lat + 1;
8: end for
9: EarliestTime(); LatestTime(); /* 文献[13]和[14] */
10: for any instruction inst do
11:   if inst is L1 miss then
12:     earliest(t^start_BAT(I)) += l1_judge_overhead;
13:     根据 earliest(t^start_BAT(I))计算总线等待延迟 wait;
14:     lat.lo += wait + bus_req_overhead + bus_ack_overhead;
15:     latest(t^start_BAT(I)) += l1_judge_overhead;
16:     根据 latest(t^start_BAT(I))计算总线等待延迟 wait;
17:     lat.hi += wait + bus_req_overhead + bus_ack_overhead;
18:   end
19: end for
20: Update_SCAT( )[10];
```

下面通过引理 2 和定理 3 证明本文的迭代分析方法具有收敛性.

引理 2 SCAT 保持 IFT 的≤不变

证明 由引理 1 可知 BRDT 保持 BAT 的≤性不变.而由式(3)可知 $t^{start}_{SCAT(I)} = t^{start}_{BRDT(I)} + bus\_req\_overhead$,其中 $bus\_req\_overhead$ 为常数,故 SCAT 保持 BAT 的≤性不变.而由式(1) $t^{start}_{BAT(I)} = t^{start}_{IF(I)} + l1\_judge\_overhead$ 可知,BAT 与 IFT 成线性关系,故 SCAT 保持 IFT 的≤性不变. □

定理 3 在迭代分析过程中,"非干扰(*non-in*)"状态单调递增,"干扰(*in*)"状态单调递减.

证明 基于引理 2 采用归纳法证明.由于篇幅所限此处不再赘述. □

4 实验

4.1 实验环境与参数设置

本文针对多核结构中 WCET 分析的实验基于 Chronos[15]扩展得到.Chronos 是一个开源的单核 WCET 分析工具.本文增加了共享二级 Cache 和共享总线分析,将其融入到单核 WCET 计算的流水线分析和路径分析中,实现多核结构中的 WCET 计算.

本文针对双核处理器展开分析,处理器核为双发射,5 段流水.存储结构如表 2 所示.每个处理器核拥有私有一级指令 Cache 和数据 Cache,共享二级 Cache L2.处理器核通过总线访问共享 Cache.总线采用 TDMA 仲裁机制,每个处理器核的总线访问槽大小设置为 220 个处理器时钟.本文将私有 Cache L1 的 Cache 行缓存大小设置为一条指令.需要说明的是,由本文前期工作可知,虽然设置更大的私有 Cache 行缓存会使得精度提高变小,但是本文方法相比现有的 Extended ILP 方法,总是获得更大的精度提高.L1-DCache 配置较大,假设数据 Cache 不会对指令 Cache 产生影响.本文从 SNU 实时测试集[16]中选取典型的测试程序进行测试.

表 2 存储结构配置

	sets	line size	associativity	latency
L1-ICache	32	8	1	10
L1-DCache	perfect			
L2-ICache	64	32	1	90

4.2 实验策略

为了更好地验证指令取指执行时序对干扰的影响,本文选取了控制流较复杂、条件转移指令较多的 *bs* 程序作为并行程序.实验将本文的方法与下列两种方法进行比较:文献[9](标记为 WCET-Address),文献[17](标记为 Extended ILP).本文的方法则标记为 WCET-Timing.

4.3 实验结果

由于共享总线可预测性差且分析复杂,一些文献简单的不考虑共享总线的影响或者将共享总线访问延

迟设置为一个常量.本文首先在不考虑总线影响的前提下分析多核结构中线程间在共享 Cache 上的干扰.此时将总线访问延迟设置为一个常量并将该延迟直接计算到取指操作的延迟中,即在考虑地址映射的分析方法 WCET-Address (Address of Interference)和考虑时序干扰的分析方法 WCET-Timing (Timing of Interference)中将总线访问延迟设置为 0.表 3 在排除总线影响的前提下对两种分析方法进行了比较.

表 3　不考虑总线影响的 WCET 估值

bench	WCET		Ratio
	WCET-Address	WCET-Timing	
insertsort	5381	4571	0.84
bs	3083	2273	0.73
matmul	2838	2208	0.77
fibcall	2280	1830	0.80
qurt	59558	40499	0.68
Average			0.76

WCET-Address 简单的将所有指令均根据地址映射分析将其设为“干扰(*in*)”状态,这种设置未考虑共享 Cache 访问行为对干扰的影响.与之相比,本文提出的分析方法可以根据共享 Cache 访问时序更好地分析线程间的干扰,从而获得更精确的 WCET 估值.由表 4 可知,在不考虑总线影响的前提下,本文的方法所获得的 WCET 估值可以提高 24%的精确度.

在表 3 的基础上,表 4 和表 5 考虑了共享总线对共享 Cache 干扰的影响.表 4 针对 L2 *hit* 给出了线程间在共享 Cache 上的干扰.由表 4 可以看出,在考虑共享总线的影响时,本文的干扰分析方法仍然可以有效地排除非干扰状态.与简单考虑地址映射方法将所有的 L2 *hit* 设置为“干扰”状态相比,本文的方法可以将 39%的 L2 *hit* 界定为“非干扰”状态,从而计算出更精确的 WCET 估值.

表 4　考虑总线影响前提下非干扰状态的排除

bench	L1 *miss*	L2 *hit*	L2 *nc*(inter-thread interference)		Ratio
			WCET-Address	WCET-Timing	
insertsort	45	31	31	22	0.71
bs	31	16	16	15	0.94
matmul	26	15	15	0	0.00
fibcall	20	10	10	7	0.70
qurt	554	254	254	168	0.66
Average					0.61

表 5 考虑共享总线影响,给出了 WCET 估值和分析时间比较.由表 5 可知,针对共享总线和共享 Cche 的多核结构,较之简单的地址映射,本文提出的分析方法平均可以获得 31%的精确度提高.这与表 3 一起验证了本文的分析方法的有效性.

此外,表 5 对分析时间进行了比较.较之简单的地址映射分析,本文方法的分析时间较大,是地址映射分析方法的 1.12 倍.即本文方法用 12%的时间耗费获得了 31%的精度提高.实验证明了本文方法的分析效率是可取的.

表 5　考虑总线影响前提下 WCET 估值和分析时间比较

bench	WCET-Address		WCET-Timing		Ratio	
	WCET	time cost(s)	WCET	time cost(s)	WCET	time
insertsort	9859	63	8630	56	0.87	1.13
bs	5137	49	5036	48	0.97	1.03
matmul	4863	44	2686	37	0.55	1.18
fibcall	3799	41	3280	39	0.86	1.06
qurt	89337	149	64197	124	0.72	1.20
Average					0.79	1.12

为了进一步验证本文提出的分析方法,本文将文献[17]提出的 Extended ILP 方法进行了扩展,使其可以处理共享总线影响,然后将该方法与本文的方法进行了比较,如表 6 所示.

表 6　与 Extended ILP[17]方法的比较

bench	Extended ILP	WCET-Timing	Ratio
insertsort	9696	8630	0.89
bs	5534	5036	0.91
matmul	3123	2686	0.86
fibcall	4152	3280	0.79
qurt	77346	64197	0.83
Average			0.856

文献[17]探讨了由于程序逻辑先后顺序导致的干扰问题,排除了不可能同时存在的干扰情况.本文提出的时序分析方法可以解决这种逻辑先后引起的干扰问题,表 6 给出了实验验证.文献[17]的方法虽然考虑了程序中由于逻辑结构导致的不可能同时存在的干扰,但是这种逻辑结构并未完全排除由于时序原因导致的不可能存在的干扰.与之相比,本文方法平均可以提高 14%精确度.

5　结束语

本文针对共享总线和共享 Cache 的多核结构提出了一种迭代的分析方法,考虑了共享总线和线程间在共享 Cache 上干扰之间的相互影响.迭代方法通过每次迭代过程中对干扰的不断精确以提高 WCET 估值精度.

参考文献

[1] J Calandrino, J Anderson, D Baumberger. A hybrid real-time scheduling approach for large-scale multi-core platforms[A]. Proceedings of the 19th Euromicro Conference on Real-Time Systems[C]. USA: IEEE Press, 2007. 247 – 258.

[2] P Puschner, C Koza. Calculating the maximum execution time of real-time program[J]. Real-time Systems, 1989, 1(2): 159 – 176.

[3] C Liu, A Sivasubramaniam, M T Kandemir. Organizing the last line of defense before hitting the memory wall for CMP[A]. Proceedings of HPCA[C]. USA: IEEE Press, 2004. 176 – 185.

[4] 谢子超,陆俊林,佟冬等.一种面向超标量处理器的高能效指令缓存路选择技术[J].电子学报,2011,39(11):2473 – 2479.
XIE Zi-chao, LU Jun-lin, TONG Dong, et al. An energy-efficient combining way selective technique for the instruction cache in superscalar microprocessors[J]. Acta Electronica Sinica, 2011, 39(11): 2473 – 2479. (in Chinese)

[5] Chen Shimin, Gibbons Phillip B, Kozuch Michael. Scheduling threads for constructive cache sharing on CMPs[A]. Proceedings of Annual ACM Symposium on Parallel Algorithms and Architectures[C]. New York: ACM, 2007. 105 – 115.

[6] 谭国真,杨际祥,王凡等.多核集群任务分配问题复杂性分析[J].电子学报,2012,40(2):241 – 246.
TAN Guo-zhen, YANG Ji-xiang, WANG Fan, et al. Complexity analysis of task assignment problem on multi-core clusters[J]. Acta Electronica Sinica, 2012, 40(2): 241 – 246. (in Chinese)

[7] M Paolieri et al. Hardware support for WCET analysis of hard real-time multicore systems[A]. Proceedings of the 36th Annual International Symposium on Computer Architecture[C]. NY, USA: ACM Press, 2009. 66 – 75.

[8] A Andrei, P Eles, Z Peng, J Rosen. Predictable implementation of real-time applications on multiprocessor systems-on-chip [A]. Proceedings of the 21st International Conference on VLSI Design[C]. USA: IEEE Press, 2008. 103 110.

[9] S Chattopadhyay, A Roychoudhury, T Mitra. Modeling shared cache and bus in multi-cores for timing analysis[A]. Proceedings of SCOPES[C]. NY, USA: ACM Press, 2010. 57 – 67.

[10] Fangyuan Chen, Dongsong Zhang, Zhiying Wang. Static analysis of run-time inter-thread interferences in shared cache multi-core architectures based on instruction fetching timing[A]. Proceedings of CSAE[C]. USA: IEEE Press, 2011. 208 – 212.

[11] Lundqvist T, Stenstrom P. Timing anomalies in dynamically scheduled microprocessors[A]. Proceedings of IEEE Real-Time Systems Symposium[C]. USA: IEEE Press, 1999. 12 – 21.

[12] Langenbach M, Thesing S, Heckmann R. Pipeline modeling for timing analysis[A]. Proceedings of Static Analysis Symposium[C]. USA: ACM Press, 2002. 294 – 309.

[13] Li Xianfeng. Microarchitecture Modeling for Timing Analysis of Embedded Software[D]. National University of Singapore: Computer science, 2005.

[14] Li X, Roychoudhury A, Mitra T. Modeling out-of-order processors for software timing analysis[A]. Proceedings of IEEE Real-Time Systems Symposium[C]. USA: IEEE Press, 2004. 92 – 103.

[15] X Li, Y Liang, T Mitra, A Roychoudhury. Chronos: A timing analyzer for embedded software[OL]. http://www.comp.nus.edu.sg/rpembed/chronos. 2007.

[16] Homepage of SNU real-time benchmark suite. http://archi.snu.ac.kr/realtime/benchmark/. 2007.

[17] Jun Yan, Wei Zhang. Accurately estimating worst-case execution time for multi-core processors with shared direct-mapped instruction caches[A]. Proceedings of Real-Time Computing Systems and Applications[C]. USA: IEEE Press, 2009. 455 – 463.

作者简介

陈芳园 女,1982 年生,湖北钟祥人.博士生.主要研究领域为实时系统和计算机体系结构.
E-mail: fychen@nudt.edu.cn

张冬松 男,1980 年生,河南信阳人.博士生.CCF 学生会员.主要研究领域为实时系统和低功耗嵌入式系统.

王志英 男,1956 年生,湖南长沙人.教授,博士生导师.主要研究领域为计算机体系结构、高性能微处理器设计和嵌入式系统.