ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.4, April 2010, pp.863−874
doi: 10.3724/SP.J.1001.2010.03705
© by *Institute of Software*, *the Chinese Academy of Sciences*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

片上多核处理器存储一致性验证*

王朋宇[1,2+], 陈云霁[1], 沈海华[1], 陈天石[3], 张 珩[1]

[1](中国科学院 计算技术研究所 计算机系统结构重点实验室,北京 100190)
[2](中国科学院 研究生院,北京 100049)
[3](中国科学技术大学 计算机科学技术系,安徽 合肥 230027)

Memory Consistency Verification of Chip Multi-Processor

WANG Peng-Yu[1,2+], CHEN Yun-Ji[1], SHEN Hai-Hua[1], CHEN Tian-Shi[3], ZHANG Heng[1]

[1](Key Laboratory of Computer System and Architecture, Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100190, China)
[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)
[3](Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
+ Corresponding author: E-mail: wangpengyu@ict.ac.cn

Wang PY, Chen YJ, Shen HH, Chen TS, Zhang H. Memory consistency verification of chip multi-processor. *Journal of Software*, 2010,21(4):863−874. http://www.jos.org.cn/1000-9825/3705.htm

Abstract: Memory consistency verification is an important part of functional validation of CMP (chip multi-processor). Since checking an execution of a parallel program against a memory consistency model is known to be an NP-hard problem, in practice, incomplete verification methods with higher than $O(n^3)$ time complexity are used to deal with memory consistency verification. In this paper, a linear time complexity memory consistency verification tool LCHECK is introduced. In the multi-processor system which supports store atomicity, there must be a time order between two operations with disjoint execution periods: The former operation in time order must be observed by the latter operation. LCHECK localizes memory consistency verification based on time order. It infers edges of orders and checks correctness in bounded operations. LCHECK is used in the verification of an industrial CMP, Godson-3, and finds many bugs of memory subsystem of Godson-3.
Key words: memory consistency model; verification; time order; chip multi-processor; cache coherence

摘 要: 存储一致性验证是片上多核处理器功能验证的重要部分.由于验证并行程序的执行结果是否符合存储一致性模型理论上是 NP 难问题,现有的验证方法中只能采用一些时间复杂度大于 $O(n^3)$的不完全方法.发现在支持写原子性的多处理器系统中,两条执行时间不重叠的操作之间存在确定的时间序.通过引入时间序的概念,设计并实现了一种线性时间复杂度的存储一致性验证工具 LCHECK.LCHECK 利用时间序将验证局部化,使得在表示程序执行

* Supported by the National Natural Science Foundation of China under Grant Nos.60603049, 60673146, 60736012, 60721061 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2007AA01Z112, 2008AA110901, 2007AA01Z114 (国家高技术研究发展计划(863)); the National Basic Research Program of China under Grant No.2005CB321600 (国家重点基础研究发展计划(973))

Received 2008-12-10; Revised 2009-04-10; Accepted 2009-07-23

© 中国科学院软件研究所 http://www.c-s-a.org.cn

结果的有向图中,序关系边的推导和正确性检测都被限定在有限范围内.与现有其他方法相比,LCHECK 时间复杂度低,对程序长度和访存地址数没有限制,因此验证效率更高.作为国产片上多核处理器龙芯 3 号的重要验证工具,LCHECK 发现了一些存储系统的设计错误.

关键词: 存储一致性模型;验证;时间序;片上多核处理器;缓存一致性

中图法分类号: TP316 **文献标识码:** A

片上多核结构已成为当前高性能微处理器的主流技术,存储系统是片上多核处理器设计中最为复杂的部分.为了提高存储系统的性能,片上多核的存储系统设计中常常采用多级缓存、分布式内存、写缓冲等多种优化技术.这些复杂的设计,使得存储系统出现错误的概率大为增加.存储一致性是片上多核处理器存储系统的重要特性,它规定了并行程序执行结果的正确性,限定了多处理器中访存操作之间的序关系,与指令集一起构成了片上多核处理器的软硬件接口,因此存储一致性的验证是片上多核处理器验证的重要部分.

用形式化方法进行存储一致性验证只能针对一些手工构建的处理器抽象模型,难以应用到系统级的验证中[1,2].基于仿真的动态验证是存储一致性验证最主要的方法,用仿真方法对存储一致性验证最大的困难在于对程序结果正确性的判断.Gibbons 和 Korach 在理论上分析了存储一致性验证的复杂性,并把顺序一致性模型的验证定义为 VSC(verifying sequential consistency)问题,证明了该问题是 NP 难问题[3,4].此外,他们还研究了 VSC 问题的一些变种.如果有一个映射关系,对程序中每个读操作都能映射到相应的写操作,则此类问题称为 VSC-读问题,如果程序中所有写操作的序关系都已确定,则此类问题成为 VSC-写问题,如果同时知道读操作的映射关系和所有写操作之间的序关系,则此类问题称为 VSC-冲突问题.VSC-读问题和 VSC-写问题也是 NP 难问题,而 VSC-冲突问题则是 P 问题.由于用随机方法产生具有冲突的并行程序很难保证写操作之间的序关系,因此实际验证中面对的都是 NP 难的 VSC-读问题.有一些工作试图通过增加专门的调试硬件资源来简化存储一致性验证[5,6],但 Intel 和 Sun 等微处理器设计厂商在实际验证中主要采用一些通用性较高的方法.这类高通用性方法仅依赖程序可见的结果,不需要额外的硬件开销,因此虽然可能漏掉一些错误,但却适用于各种处理器和多种存储一致性模型的验证.Hangal 等人提出一种不完全的存储一致性验证工具 TSOtool[7],TSOtool 能够随机生成具有冲突的并行程序,并在并行程序运行后构建程序的执行图,通过检测执行图中是否有环来判断设计的正确性.它的复杂度为 $O(n^5)$,n 为并行程序中操作的数目.利用向量时钟,他们将算法的复杂度降低为 $O(pn^3)$,p 为处理器的个数[8].Roy 等人提出了一种与 TSOtool 类似的存储一致性验证工具,时间复杂度降低为 $O(n^4)$[9].Manovit 采用回溯的方法使得 TSOtool 能够进行完全的验证,但时间复杂度高达 $O((n/p)^p pn^3)$[10].总体来说,这些方法的缺点是算法时间复杂度较高,并且通常会对并行程序中访存操作的个数和访存地址范围有一定的限制.

本文提出一种线性时间复杂度的片上多核处理器存储一致性验证工具 LCHECK.作为一种通用性较高的方法,LCHECK 所需要的所有信息均来自程序的可见结果.LCHECK 可以验证顺序一致性[11]、处理器一致性[12]、弱一致性[13]等存储一致性模型.它利用了访存操作之间的时间序概念[14]来降低验证的复杂度.在处理器中,每条访存操作都有一段执行时间:这个时间从操作进入处理器开始,到操作提交时结束.如果访存操作 u 的提交时间早于访存操作 v 的进入时间,我们称 u 在时间序上先于 v.在支持写原子性的多处理器系统中[15],在时间序中位于前面的操作的结果一定会被后面的操作所观察到.只有执行时间有重叠部分的操作之间才存在顺序的不确定性.由于操作窗口、访存队列、写缓存等部件的大小限制和一个访存操作的执行时间相重叠的操作的数量有限,因此 LCHECK 在验证存储一致性时,序关系的推导及执行图中环的检测都可以被限定在一定的操作范围内.这使得 LCHECK 的时间复杂度仅为 $O(p^3n)$.与同类型方法(如 TSOtool 等)相比,LCHECK 有着最低的时间复杂度.此外,LCHECK 对访存操作的地址和访存操作数量也没有限制.LCHECK 已经成为国产片上多核处理器龙芯 3 号[16,17]的重要验证工具,并发现了龙芯 3 号处理器中一些存储系统的设计错误.

1 并行程序模型及正确性标准

处理器的验证中通常需要一个参考模型,参考模型能够给出与处理器实际执行结果进行比较的参考结果.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

而在共享存储的片上多核处理器系统中,多个处理器可以读写同一单元,一个处理器所存的数可能被多个处理器访问,甚至一个单元内容的变化可能在不同时刻被不同的处理器所接受.因此,简单的参考模型不能给出并行程序的正确结果.在本文中,我们利用了参考文献[18]中关于并行程序模型的定义及正确性标准:

(1) 一个程序由多个进程组成,进程是操作的有限序列.进程中只有两种操作,即取数操作 LOAD 和存数操作 STORE,每个进程运行在不同的处理器上;

(2) 每个寄存器或存储单元都有一个初始值,缺省值为 0;

(3) 程序的任一执行结果由程序中出现的所有寄存器和存储单元的最终值来决定.

在上述假设中,一个进程只包含 LOAD 和 STORE 两种操作.对于描述片上多核处理器的存储特性来说,这种假设是合理的,因为对存储一致性模型来说任何复杂的进程都表现为存数操作和取数操作的有限序列.

在给出并行程序的正确性标准之前,我们给出并行程序的 3 种序关系定义:

定义 1(程序序). 称访存操作 u_1 在程序序上先于访存操作 u_2,如果 u_1 和 u_2 在同一进程中且 u_1 在进程中出现在 u_2 的前面,标记为 $u_1 \xrightarrow{P} u_2$.

定义 2(处理器序). 称访存操作 u_1 在处理器序上先于访存操作 u_2,如果 u_1 和 u_2 在同一进程中且 u_1 在某存储一致性模型的规定下必须先于 u_2,标记为 $u_1 \xrightarrow{PO} u_2$.

定义 3(执行序). 在共享存储片上多核处理器系统中,如果两个访存操作访问的是同一单元且其中至少一个是存数操作,则称这两个访问操作是冲突的,执行序确定冲突操作之间的序关系.称写操作 w 在执行序上先于访存操作 u,如果 w 是 u 之前最后一个访问同一单元的写操作,标记为 $w \xrightarrow{E} u$;我们称写操作 w 在执行序上后于访存操作 u,如果 w 是 u 之后第 1 个访问同一单元的写操作,则标记为 $u \xrightarrow{E} w$.

不同的存储一致性模型有不同的规则定义程序的处理器序,访存操作 u_1 在程序序上先于访存操作 u_2,并不意味着 u_1 在处理器序上先于 u_2.顺序一致性模型对处理器序的规定为:任一处理器都严格按照访存操作在进程中出现的次序即程序序执行访存操作,且在当前访存操作彻底完成之前不能开始执行下一条访存操作.因此,在顺序一致性模型中,程序的程序序与处理器序是一致的.而在比顺序一致性弱的模型,如处理机一致性模型中,对处理器序的规定为:在任一取数操作 LOAD 允许被执行之前,所有在同一处理器中先于取数操作 LOAD 的其他取数操作都已完成;在任一访存操作 STORE 允许被执行之前,所有在同一处理器中先于存数操作 STORE 的访存操作都已完成.上述条件允许 STORE 操作之后的 LOAD 操作可以越过 STORE 操作执行,因此,程序序 $STORE \xrightarrow{P} LOAD$ 不意味着处理器序 $STORE \xrightarrow{PO} LOAD$.

程序的执行结果可以用有向图来表示[19],图中的节点表示访存操作,有向边表示节点间的序关系.在顺序一致性模型中,程序执行正确的充要条件是程序的程序序和执行序无环[18].如果有向图表示的程序的程序序和执行序中出现了环,则表明存储一致性设计没有满足存储一致性模型的要求,不能使程序正确执行,设计中存在错误.

图 1 的程序段 PRG_1(初始值 $R_1=R_2=a=b=0$),在顺序一致性模型中的程序序和处理器序为 $L_{11} \xrightarrow{P} L_{12}$, $L_{21} \xrightarrow{P} L_{22}$,冲突操作为$(L_{11},L_{22})$,$(L_{12},L_{21})$.如果程序的执行序为 $L_{12} \xrightarrow{E} L_{21}$,$L_{22} \xrightarrow{E} L_{11}$,则用有向图表示的程序序与执行序的关系如图 2 所示.在图 2 中存在一个环 $L_{11}\rightarrow L_{12}\rightarrow L_{21}\rightarrow L_{11}$,因此程序被错误地执行,说明设计违反了顺序一致性模型的要求,设计中有错误.对于其他较弱的一致性模型,如处理机一致性模型,因为程序的程序与处理器序不一样,对于图 1 中的程序段,即使程序的执行序为 $L_{12} \xrightarrow{E} L_{21}$,$L_{22} \xrightarrow{E} L_{11}$.在处理机一致性模型中,因为 LOAD 操作可以越过前面的 STORE 操作,程序的执行结果也是符合一致性模型的规定.表现在有向图中,则为少了 $L_{11} \xrightarrow{P} L_{12}$ 和 $L_{21} \xrightarrow{P} L_{22}$ 的两条边,有向图中不存在环,说明执行结果符合处理机一致性模型要求.因此,我们需要一种通用的序关系来判断程序执行的正确性,这种序关系就是全局序.

定义 4(全局序). 称访存操作 u_1 在全局序上先于访存操作 u_2,如果 u_1 在处理器序上先于 u_2,或者 u_1 在执行序上先于 u_2,或者 u_1 在全局序上先于某一访存操作 u,而 u 在全局序上先于 u_2,标记为 $u_1 \xrightarrow{GO} u_2$,程序的全局序是处理器序和执行序的传递闭包关系.在全局序定义下,程序执行的正确性标准为:用有向图表示的程序的全局序无环.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

程序的处理器序在程序确定后,可以由存储一致性模型静态地推导出来,且是唯一确定的.而程序的执行序则是在程序执行后,根据程序的执行结果推导出来的.程序的不同次执行可能会有不同的执行序,因此对于有向图表示的程序执行结果,只能是对程序的某一次确定执行进行检查.

Process P_1	Process P_2
L_{11}: *STORE a*,1	L_{21}: *STORE b*,1
L_{12}: *LOAD R_1,b*	L_{22}: *LOAD R_2,a*

Fig.1 Program segment PRG_1

图 1 程序段 PRG_1

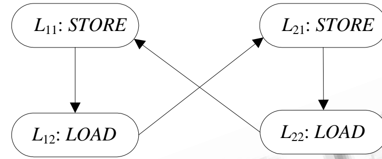


Fig.2 Program order and execution order represented by DAG of PRG_1

图 2 程序 PRG_1 程序序与执行序的有向图表示

2 时间序约束下的存储一致性验证

一般的存储一致性验证方法需要在全局范围即所有访存操作之间进行有向图边的推导和环的判断,这是导致这类方法算法复杂度高的主要原因.然而,在支持写原子性的片上多核处理器系统中,每条访存操作的执行时间都是有限的,执行时间不重叠的操作之间存在一种自然的序关系.目前,有的片上多核处理器系统中提供了显式的全局时钟.对于没有显式全局时钟的片上多核处理器系统,在文献[20]中,Lamport 证明了在分布式系统中,如果各个进程使用不同的分布时钟,则使用一定的算法可以把不同的时钟换算成一个近似的全局时钟,并且保证这个全局时钟的误差足够小.利用这个全局时钟,我们可以给出每条指令的执行时间区间,并以此判断访存操作之间的时间序关系.这种新的序关系可以极大地简化存储一致性的验证.

2.1 执行时间和时间序

一条访存操作在执行之前,先要从处理器的指令 Cache 中读出,因此访存操作都有一个进入时间.处理器内部的部件(操作窗口、访存队列等)大小都是有限的,除非发生了死锁或者活锁,访存操作都会在有限时间内完成并提交,因此访存操作存在一个提交时间.

定义 5(执行时间). 假设有一个全局时钟,访存操作 u 的进入时间 $t_e(u)$是该操作进入到处理器的内部临时部件的时间,提交时间 $t_c(u)$是该操作从处理器内部退出的时间.访存操作 u 的执行时间从进入时间开始到提交时间结束.如果访存操作 u,v 的执行时间是重叠的,则称它们处于对方的执行时间内[14].

在支持写原子性的片上多核处理器系统中,一个操作进入处理器之前不会对任何操作造成影响;一个操作被提交后,不会再有任何操作对它造成影响.因此,在访存操作之间存在一种自然的序关系,我们称其为时间序.

定义 6(时间序). 如果访存操作 u 的提交时间在访存操作 v 的进入时间之前,则称 u 在时间序上先于 v[14].

$$tc(u) < te(v) \leftrightarrow u \xrightarrow{T} v .$$

根据时间序的定义,如果访存操作 v 处于访存操作 u 的执行时间内,则有$\neg(u \xrightarrow{T} v \vee v \xrightarrow{T} u)$.时间序不要求 u 和 v 处于同一个处理器中或者访问同一个内存单元地址,只取决于访存操作的进入时间和提交时间.假设系统中有一个全局时钟,则所有操作在任意时间点上都有一个全局序关系.作为本文工作的基础,对全局序及时间序作以下假设.对于访存操作 u 和 v:

(1) 如果 u 在程序序上先于 v,则 u 的进入时间在 v 的进入时间之前:

$$(u \xrightarrow{P} v) \rightarrow (te(u) < te(v)) .$$

(2) 如果 u 在处理器序上先于 v,则 u 的提交时间在 v 的提交时间之前:

$$(u \xrightarrow{PO} v) \rightarrow (tc(u) < tc(v)) .$$

(3) 如果 u 在执行序上先于 v,则 u 的提交时间在 v 的提交时间之前:

$$(u \xrightarrow{E} v) \rightarrow (tc(u) < tc(v)) .$$

(4) 如果 u 在时间序上先于 v,则 u 的提交时间在 v 的提交时间之前:

© 中国科学院软件研究所 http://www.c-s-a.org.cn

$$(u \xrightarrow{T} v) \rightarrow (tc(u) < te(v)).$$

对不同的微体系结构来说,进入时间和提交时间的定义可能会不同有所,但上面的假设在支持写原子性的片上多核处理器系统中是成立的.基于以上假设,我们可以分析全局序和时间序之间的关系.两个访存操作之间的全局序表明,位于序关系前面的操作的执行结果可以被序关系中后面的操作看到,两个访存操作之间的时间序关系表明这两个操作之间的执行顺序.在一个正确的设计中,访存操作之间的全局序关系和时间序关系必须是一致的:在时间序中,位于前面的操作结果能被后面的操作观察到,我们称其为存储一致性的时间序约束.

定理 1(时间序约束定理). 在支持写原子性的片上多核处理器系统中,位于时间序前面的访存操作的结果能被位于时间序后面的访存操作观察到,形式化表示为[14]

$$(v \xrightarrow{T} u) \rightarrow \neg(u \xrightarrow{GO} v).$$

证明:根据时间序的定义,定理 1 等同于$(u \xrightarrow{GO} v) \rightarrow (t_e(u) < t_c(v))$.

因为$(u \xrightarrow{PO} v) \rightarrow (t_c(u) < t_c(v))$和$(u \xrightarrow{E} v) \rightarrow (t_c(u) < t_c(v))$,所以,$(u \xrightarrow{GO} v) \rightarrow (t_c(u) < t_c(v))$.由于偏序之间具有传递关系,因此,$(u \xrightarrow{GO} v) \rightarrow (t_c(u) < t_c(v))$,从而$(u \xrightarrow{GO} v) \rightarrow (t_e(u) < t_c(v))$成立. □

2.2 时间序约束下的正确性标准

时间序约束定理保证了访存操作之间的时间序和全局序是一致的.因此,我们可以将全局序和时间序结合起来形成一种新的偏序关系,称其为时间全局序,用符号 $\xrightarrow{GTO}$ 表示.程序的执行结果可以用时间全局序关系构成的有向图表示,图中的顶点为访存操作,边为操作之间的时间全局序关系.我们称用时间全局序关系表示的程序执行结果图为 GTO 图.根据时间序约束定理,程序执行结果的正确性可以用 GTO 图中是否存在环来判断.

对于访存操作 u,在 GTO 图中可能有 3 种环包含 u:

1) 环中除了 u 以外的访存操作都不在 u 的执行时间内;

2) 环中一部分访存操作处于 u 的执行时间内,另外一部分访存操作不在 u 的执行时间内;

3) 环中所有的访存操作都处于 u 的执行时间内.

引入时间序的目的是为了将访存操作之间的序关系局部化,局部化的关键是确定出上述种类 1)中的序,因为在种类 1)中,访存操作都不在 u 的执行时间内.

引理 1. 假设在 GTO 图中有一个包含访存操作 u 的环 C,如果环中除了 u 之外的所有操作在时间序上都先于 u,则环 C 必定包含一个写操作 w,且 w 在执行序上是在 u 之后的,形式化表示为[14]

$$(\forall v \in C : (v \neq u) \rightarrow (v \xrightarrow{T} u)) \rightarrow (\exists w \in C : u \xrightarrow{E} w).$$

证明:因为 u 在环 C 中,因此 u 必定有后续.假设 v 是 u 的后续,u 和 v 之间可能有 3 种序关系:$u \xrightarrow{T} v$, $u \xrightarrow{PO} v$,$u \xrightarrow{E} v$.因为环 C 中除 u 之外的其他操作在时间序上都先于 u,因此 $u \xrightarrow{T} v$ 不成立.因为环 C 中所有操作在时间序上都先于 u,所以 $t_c(v)$在 $t_c(u)$之前,由假设 2 可知,$u \xrightarrow{PO} v$ 不成立,因此必有 $u \xrightarrow{E} v$.如果 v 是读操作,u 不可能得到 v 的值,u 也因此不能在执行序上先于 v.因此,v 必定是写操作. □

基于以上定理和引理,我们给出在时间序约束下的存储一致性协议验证的正确性规则,3 个规则对应上述 3 种可能的环.

定理 2(检测规则定理). 对并行程序中的任何操作 u,用 GTO 表示的该并行程序的执行结果无环的充分必要条件是以下 3 个规则成立.w 为写操作,v 和 v'为任意访存操作,P 为并行程序[14].

规则 1. $\forall w \in P : (w \xrightarrow{T} u) \rightarrow \neg(u \xrightarrow{E} w)$.

规则 2. $\forall v, v' \in P : ((v \xrightarrow{T} u) \wedge (v' \xrightarrow{GO} v)) \rightarrow \neg(u \xrightarrow{GO} v')$.

规则 3. $\neg(\exists C : (\forall v \in C : \neg(u \xrightarrow{T} v \vee v \xrightarrow{T} u)))$.

证明:充分性:用反证法证明.假设规则 1~规则 3 都成立,且 GTO 图中有一个环 C,操作 u 是环 C 中最后一个提交的操作.根据规则 3,必定存在一个操作处于 u 的执行时间以外.从 u 开始遍历环 C,v 是遍历 C 时在时间序上先于 u 的第一操作.因为 u 是环 C 中最后一个提交的操作,所以 $u \xrightarrow{T} v$ 不成立,u,v 之间的时间序为 $v \xrightarrow{T} u$.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

如果环 C 中除了 u 之外的操作在时间序上都先于 u,则根据引理 1,必定存在写操作 w 满足 $u \xrightarrow{E} w$,这与规则相矛盾.因此,环 C 中必定存在处于 u 的执行时间内的操作.假设 v'和 v 是环 C 中的操作,且 v'是 v 的前驱,则 $a \xrightarrow{T} b$ 是环 C 中从点 u 开始第 1 个具有时间序的边,根据时间序的定义有 $t_c(u)<t_c(a)<t_e(b)<t_c(b)$.但因为 u 是环 C 中最后一个提交的操作,所以 $t_c(u)$不可能先于 $t_c(b)$.因此环 C 中没有从 u 到 v 的时间序边,从而 $u \xrightarrow{GO} v'$ 和 $v' \xrightarrow{GO} v$ 成立,但这与规则 2 相矛盾.充分性证明完毕.

必要性:如果用 GTO 图表示的并行程序执行结果无环,对于规则 1,如果任何写操作 w 都能使 $w \xrightarrow{T} u$ 成立,则 $u \xrightarrow{E} w$ 不能成立,否则将有环 $w \xrightarrow{T} u \xrightarrow{E} w$.对于规则 2,如果操作 u,v,v'使得($v \xrightarrow{T} u \wedge v' \xrightarrow{GO} v$)成立,则 $u \xrightarrow{GO} v'$不能成立,否则将有环 $v' \xrightarrow{GO} v \xrightarrow{T} u \xrightarrow{GO} v'$.对于规则 3,因为在 GTO 图中无环,因此在 u 的执行时间内也必定无环.必要性证明完毕. □

2.3 局部化检测机制

在支持写原子性的片上多核处理器系统中,访存操作从单个处理器的内部临时部件中退出时就可以全局可见.除非发生了死锁或者活锁,一个访存操作的执行时间不可能一直持续,因此,处于一个访存操作的执行时间内的操作数量是一定的.利用上述 3 条规则,可以将 GTO 图中是否存在环的检测局部化,对每条访存操作来说只需考虑有限条操作即可.

定理 3(局部化检测定理). 在时间序约束下的存储一致性验证,对任何访存操作的检测只需考虑 $O(p)$个相应的操作,p 为片上多核处理器中处理器的个数[14].

证明:根据定理 2,判断一个并行程序的执行是否正确,只需检测定理 2 中的 3 条规则是否成立.对于规则 1,w 是在 u 进入之前已经提交的操作,因此 w 的个数是常数.对于规则 2 和规则 3,需要遍历 u 的执行时间内的所有操作.假设 T 为系统中访存操作执行时间的长度,对于一个 m 发射的处理器来说,处于 u 的执行时间内的操作数不超过 $3Tmp$.因为 T 和 m 是常数,$3Tm$ 也因此是常数,假定为 C.因此要检测 u,只需考虑 Cp 个操作. □

3 LCHECK

LCHECK 是我们用上述定理验证片上多核处理器存储一致性的工具.LCHECK 的操作分为 3 步:(1) 用伪随机方法产生有冲突访问的并行程序;(2) 在片上多核处理器上运行并行程序,在运行过程中将操作时间信息及取数操作所得到的值作为运行结果记录下来;(3) 对运行结果进行分析,构建表示并行程序结果的有向图,并通过判断图中是否有环,确定设计是否符合存储一致性模型的要求.

LCHECH 可以在两种环境下运行:仿真环境和 FPGA(field programmable gate array)硬件芯片环境.当运行在仿真环境下时,可以利用仿真环境中对被验证设计的可观察性得到用于分析的结果,如取数操作取回的值、访存操作的进入时间和提交时间等.当 LCHECK 运行在 FPGA 硬件芯片环境下、需要在产生并行程序时,在程序中插入特殊指令获得取数操作所取得的值,并且需要依赖硬件实现获得访存操作的进入时间和提交时间.

为了得到访存操作之间的时间序关系,需要知道每条访存操作的进入时间和提交时间,这在仿真环境下很容易实现.但在 FPGA 硬件芯片中,很难得到每条访存操作精确的进入时间和提交时间.如果能够得到每条访存操作的进入时间上界和提交时间下界,LCHECK 就能确定出访存操作之间的时间序关系.而访存操作的进入时间上界和提交时间下界在 FPGA 硬件芯片中是可以得到的.如图 3 所示,访存操作 u 的提交时间下界在访存操作 v 的进入时间上界之前,那么 u 在时间序上先于 v:$u \xrightarrow{T} v$.对于任何操作,它的进入时间的上界可以是在程序序上先于它的操作的进入时间,而它的提交时间的下界可以为在处理器序上后于它的操作的提交时间.用访存操作的进入时间上界和提交时间下界来推导访存操作之间的时间序关系,会降低 LCHECK 结果分析的效率,因为这样操作实际上是延长了一个访存操作的执行时间,相应地要处理的处于执行时间内的操作也比原来多了.

在龙芯 3 号处理器中,有两个软件可见的寄存器:一个寄存器记录最后一条操作的程序计数器 PC(program counter)和进入时间,一个寄存器记录最后一条操作的程序计数器和提交时间.每个固定的时间,如每个 100 条指令就将这两个寄存器的值读出,可以得到一部分操作的进入时间和提交时间,从而可以获得所有操作的进入时

© 中国科学院软件研究所 http://www.c-s-a.org.cn

间和提交时间上下界.

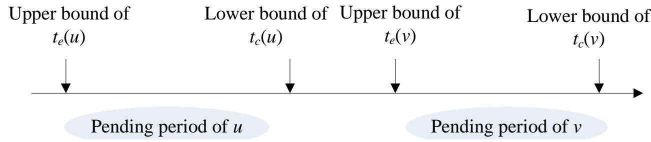


Fig.3 Upper bound and lower bound of u and v

图 3 访存操作 u,v 的进入时间/提交时间上下界

3.1 程序产生和执行

LCHECK 用伪随机的方法产生有冲突访问的并行测试程序,并行测试程序主要由存数操作 STORE 和取数操作 LOAD 组成.为了增加冲突访问的概率,程序的访问地址被限定在一个很小的范围内.对同一地址的不同存数操作 STORE,其写入的值是唯一的.存数操作值的唯一性是为了保证能够根据值判断 LOAD 操作是从哪个 STORE 操作取出的值,从而直接确定操作之间的执行序关系.如果 LCHECK 要在 FPGA 硬件环境中运行,在测试程序的每条 LOAD 操作之后,LCHECK 会插入一条专门的指令将 LOAD 操作取回的值保存到处理器的内部 RAM 中.此外,每隔 1 000 条指令,LCHECK 会插入一段指令序列来通过访问处理器的 PC 寄存器来得到指令的进入时间和提交时间的上下界.

产生的并行测试程序在被验证的多核处理器系统中运行.在运行过程中,取数操作取得的值及操作的进入时间和提交时间信息会被存到处理器内部的 RAM 中.通过一个 DMA(direct memory access)引擎,这些信息可以被完整地导出到外部调试主机上,以供存储一致性验证使用.当然,如果是在 RTL(register transfer level)仿真环境下,这些信息可以直接用$fdisplay 输出到一个记录文件中.

3.2 GTO图构造及结果检测

在用 GTO 图表示的程序执行结果图中,节点表示访存操作,并且还包含了该操作的执行时间信息,如操作的进入时间和提交时间,节点之间的时间序关系边不需要显式地表示出来,可以从进入时间和提交时间推导出来.除了节点之间的时间序边以外,时间全局序边还可以分为两类:一类是静态的,由存储一致性模型规定在程序生成后唯一确定下来;另一类是在程序执行后由程序的执行结果推导出来.

静态边推导规则

处理器边规则:根据相应的存储一致性模型规定在两个节点之间添加边.

执行边推导规则

直接边规则:如果读操作 r 读出的值是写操作 w 写入的值,且 r 和 w 访问的内存单元地址相同,则加 1 条从 w 到 r 的边.

观察边规则:如果读操作 r 读出的值是写操作 w 写入的值,且 r 和 w 访问的内存单元地址相同,写操作 w'和 r 及 w 访问同一内存单元地址,且 w'在程序序上先于 r,则加 1 条从 w'到 w 的边.

推导边规则 1:读操作 r 和写操作 w,w'访问同一内存单元地址,w'处于 w 的执行时间内,读操作 r 读出的值为写操作 w'写入的值,如果 w 在全局序或者时间序上先于 r,则加 1 条从 w 到 w'的边.

推导边规则 2:读操作 r 和写操作 w,w'访问同一内存单元地址,w'处于 r 的执行时间内,读操作 r 读出的值为写操作 w 写入的值,如果 w 在全局序或者时间序上先于 w',则加 1 条从 r 到 w'的边.

对于处理器边规则、直接边规则及观察边规则的算法,只要采用适当的数据结构,算法的时间复杂度为 $O(n)$,图 4 为相应的算法描述.在 LCHECK 中,推导边规则也是线性时间复杂度的,推导边规则只考虑每条边相关操作的执行时间内的操作,因为推导边如果与时间序一致,就不需要重复推导,如果推导边和时间序有冲突,说明设计中有错误.对于推导边规则 1 和推导边规则 2,在时间序约束下,如果操作 u 和 v 之间增加了一条边,只会对处于 u 的执行时间内的操作具有全局序关系,因此,LCHECK 只需处理 u 的执行时间内的所有写操作 w(如果 u 本身也是写操作,也需要对 u 进行处理).对于处于写操作 w 执行时间内的所有读操作 r,如果 r 在全局序或时

© 中国科学院软件研究所 http://www.c-s-a.org.cn

间序上后于 w,r 和 w 访问的内存单元地址相同,且 LCHECK 没有根据 w,r 之间的边推导新的边;如果 w'访问的内存单元地址和读操作 r 相同,且 r 读出的值为 w'写入的值,则利用推导边规则 1 推导出 w'到 w 之间的边,并调用函数 *set_inferred* 记录已根据 w,r 之间的边推导过.对于处于写操作 w 执行时间内的所有写操作 w',如果 w'在全局序或时间序上后于 w,w 和 w'访问相同的内存单元地址,且 LCHECK 没有根据 w,w'之间的边推导新的边;如果读操作 r 和 w,w'访问的内存单元地址相同,且 r 读出的值为 w 写入的值,则根据推导边规则 2 推导出 r 到 w 之间的边,并调用函数 *set_inferred* 记录已根据 w,w'之间的边推导过.因此,LCHECK 只在执行时间相互重叠的两个操作之间增加边,每个操作最多有 $O(p)$条推导边.推导边规则算法描述如图 5 所示.

```
Processor Edge Rule
(take the processor consistency for example)
For (i=n−1; i>=0; i−−) begin
  If (the operation of i is load operation) begin
          If (i!=n)
          add an edge from i to the last load operation;
          marked the load operation as the last load operation;
  end
  else begin
       add an edge from i−1 to i and add an edge from I
       to the last store operation;
       marked the store operation as the last store operation;
  end
end
```

```
Directed Edge and Observed Edge Rule
Create hash table 1 used the operand of store operation as key;
Create hash table 2 used the address of store operation as key;
For (each load operation u) begin
   find the store operation w which operand is the same to
   the operand of load operation in hash table 1, then add
   an edge from w to u;
   for (find all store operation w′ which address is equal to
   operation u in hash table 2) begin
        find the store operation w which operand is the same to
        the operand of load operation in hash table 1, then add
        an edge from w to u;
        if (w′ --P--> r)
        add an edge from w′ to w;
   end
end
```

Fig.4 Algorithm for processor edge, execution edge and observed edge

图 4 处理器边、执行边及观察边算法描述

```
int infer_edge(u,v) begin
   for (all store operation w in the pending period of u) begin
        for (all load operation r in the pending period of w) begin
           if (w --GO--> r && !inferred(w,r) && address(w)==address(r)) being
               w′ is the store operation which r get the value from;
               if (w′ --T--> w) panic();
               if (w --T--> w′) continue;
               if (there is no edge from w to w′) begin
                  add an edge from w to w′;
                  infer_edge(w,w′);
               end
               set_inferred(w,r);
           end
        end
         for (all store operation w′ in the pending period of w) begin
         if (w --GO--> w′ && !inferred(w,w′) && address(w)==address(w′))
            for (all load operation r which get value from w) begin
               if (w′ --T--> r) panic();
               if (r --T--> w′) continue;
               if (there is no edge from r to w′) being
                  add an edge from r to w′;
                  infer_edge(r,w′);
               end
            end
            set_inferred(w,w′);
        end
   end
end
```

Fig.5 Algorithm for inferring edge

图 5 推导边规则算法描述

© 中国科学院软件研究所 http://www.c-s-a.org.cn

用上述规则使 GTO 图达到稳定状态后(GTO 图中不会再添加新的边),需要检测图中是否存在环来判断设计的正确性.在用以上规则构造的 GTO 图中,根据推导边规则 1 和推导边规则 2 推导出来的边不是全局的,因此不能用传统的方法,如拓扑排序的方法,来检查 GTO 图中是否有环.如图 6 所示,LCHECK 检查 GTO 中是否有环是基于定理 2 给出的 3 条检测规则.对于访存操作 u,规则 1 的检测为:与 u 访问相同内存单元地址且 w 在时间序上先于 u 的写操作能否将值传递给 u,如果 u 为读操作,且得到访问相同内存单元地址的写操作 w'的值,如果 w'在全局序或者时间序上先于 w,则有错误.规则 2 和规则 3 的检测可以通过遍历以 u 为尾的全局序边完成.如果找到在时间序上先于 u 的操作,则违反了规则 2,如果遍历回到节点 u,则违反了规则 3.

```
W is the last entered store operation accessing the same address with u before u in time order,
w' is the store operation which u gets value from
int check_cycle() begin
    for (all operation u) begin
            if (w' --GO--> w||w' --T--> w) panic();
            travel all operation v which has edge start from u;
                If (v --T--> u||v==u) panic();
    end
end
```

Fig.6 Algorithm for checking cycle in GTO

图 6 GTO 图环检测算法

算法的时间复杂度分析:对于处理器边规则、观察边规则和直接边规则,算法的时间复杂度都为 $O(n)$,n 为并行程序中所有的访存操作数,即 GTO 中的节点数.对于推导边规则 1 和推导边规则 2,对任意操作 u,与 u 有关的边共有 $O(p)$,因此,在一个有 n 个操作的程序中,总共有不超过 $O(pn)$条推导边.算法 *infer_edge* 的时间复杂度为 $O(pn)$.在检测 GTO 图中是否存在环的算法中,每次迭代需要的时间复杂度为 $O(p^2)$,因此,总的算法时间复杂度为 $O(p^3n)$.

4 龙芯 3 号片上多核处理器验证实例

LCHECK 已经成功用在龙芯 3 号系列片上多核处理器的验证中,处理器的个数可以为 2~16 个.所验证的存储一致性模型具体为:任意存数操作在被允许执行之前,所有在同一处理器中先于该存数操作的访存操作都已完成;任意访存操作在被允许执行之前,所有在同一处理器中先于该访存操作的 LL(load link)和 SC(store conditional)都已完成;任意 LL 或 SC 在被允许执行之前,所有同一处理器中先于该操作的访存操作都已完成.

图 7 是 LCHECK 在各种配置下的运行时间分析,实验平台的配置为:AMD 速龙 64 3200+处理器,64GB 内存.图中的横坐标轴为并行程序中访存操作的个数,纵坐标为程序运行的时间.从图中可以看出,在处理器个数比较少,如 2 个和 4 个的情况下,LCHECK 的运行时间随着并行程序中访存操作个数呈线性增长;在处理器个数比较多,如 8 个和 16 个的情况下,在访存操作比较多的时候,运行时间是会比线性稍微增快一些,这是因为更多的处理器需要增加不同处理器之间冲突操作的序关系的推导和环的检测.LCHECK 对于访存操作的地址和访存操作本身的数量没有限制.这对于利用随机程序验证存储一致性来说,能够更好地发现设计中的错误,因为运行长的程序可以使片上多核处理器系统的各个部件的状态更大概率地被遍历到.

LCHECK 是在全系统中对存储一致性进行验证的,因此可以发现设计中各种违反一致性模型的错误.在实际的验证中,LCHECK 发现了多个存储一致性设计中的错误,其中有的错误是不能被其他验证工具,如 TSOtool 发现的.图 8 是我们用 LCHECK 验证龙芯 3 号处理器时发现错误的程序例子,在图 8 的程序中,内存单元的初始值都为 0,(SW_0 a,1)表示写操作 SW_0 将值 1 写入到内存单元 a,(LW_3 b=1)表示读操作 LW_3 从内存单元 b 读出的值为 1.在该图所表示的程序中,访存操作 SW_4 在时间序上先于访存操作 LW_5,其他访存操作之间没有时间序关系.图 9 是该程序的 GTO 图.

根据时间序关系定义,有如下的隐含时间序边:SW_4 到 LW_5.

根据处理器边规则,增加如下处理器序边:SW_0 到 SW_1,SW_1 到 SW_2.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

根据直接边规则,增加如下执行序边:SW_0 到 LW_5,SW_4 到 LW_3.

根据观察边规则,增加如下执行序边:SW_2 到 SW_4.

根据推导边规则 2,增加如下执行序边:LW_5 到 SW_1.

这些边结合在一起,在 SW_1,SW_2,SW_4 和 LW_5 之间形成了一个环(图中用虚线表示),违反了检测规则定理中的规则 2,这个错误可以被表 3 的环检测算法的第 6 行发现.经过分析发现,问题的原因是片上网络在特定的拥塞情况下,二级 cache 返回给处理器的数据和发送给处理器的 invalidation 请求之间发生了乱序(龙芯 3 号片上网络要求二级 cache 到处理器点到点必须有序),导致 SW_2 实际上越过了 SW_1.在龙芯 3 号处理器的验证中,多核操作系统、SPLASH2 和 SPEC2000 等大规模实际应用程序都没有发现这个隐藏得很深的错误.

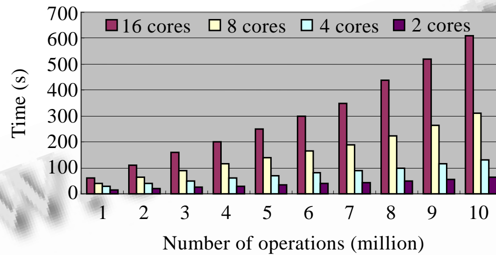


Fig.7 Analysis time of LCHECK

图 7 LCHECK 的运行时间

P_0	P_1
SW_0 a, 1	SW_4 b, 1
SW_1 a, 2	LW_5 a=1
SW_2 b, 2	
LW_3 b=1	

Fig.8 An example program which finds bug

图 8 发现错误的并行程序例子

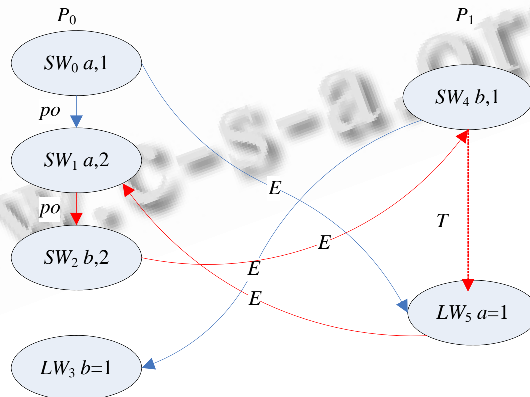


Fig.9 GTO graph of program in Fig.8

图 9 图 8 中程序的 GTO 图

5 结论和未来工作

存储一致性的验证传统上被认为是 NP 难问题,实际验证中通常采用一些多项式复杂度的不完全方法.然而对于多数片上多核处理器来说,存在一种自然的限制:处理器内的资源都是有限的,在任何时刻,都只有有限的操作处于处理器内部.基于这种限制,我们提出了操作的执行时间及操作之间一种自然存在的偏序关系:时间

© 中国科学院软件研究所 http://www.c-s-a.org.cn

序.时间序约束使得执行时间不重叠的两条访存操作之间存在确定的先后顺序.在这种约束下,操作之间的关系可以被局部化:无论边的推导还是环的检测都只需要考虑有限个操作,因此,存储一致性验证的时间复杂度被大幅度降低.

基于我们的理论工作,本文提出了一种验证片上多核处理器存储一致性协议的工具 LCHECK,LCHECK 支持多种存储一致性模型的验证,如顺序一致性模型、处理器一致性模型、弱一致性模型等,它的时间复杂度为 $O(p^3n)$.与同类型的存储一致性检验工具相比,LCHECK 有着最低的时间复杂度,能够快速检验较长的测试程序,同时不存在访存地址数量等限制,因此有效提高了验证的效率.作为访存系统的验证平台,LCHECK 在龙芯 3 号多核处理器验证过程中找到了一些隐藏很深的错误.理论和实验结果都表明了验证方法的有效性.

LCHECK 在降低验证存储一致性的复杂性方面有较大的进步,由于降低了时间复杂度,极大地增加了验证中的访存操作的数量.运行长程序更容易发现设计中的错误,但同时也给调试带来了挑战,需要提高LCHECK的可调试性,使得在调试过程中能够更容易地定位到错误的位置.

致谢 本文作者与吕毅博士在存储一致性领域进行了广泛而深入的讨论,裨益良多,在此表示衷心的感谢.

References:

[1] Chatterjee P, Sivaraj H, Gopalakrishnan G. Shared memory consistency protocol verification against weak memory models: Refinement via model-checking. In: Proc. of the 14th Int'l Conf. on Computer Aided Verification (CAV 2002). 2002. http://www.cs.utah.edu/formal_verification/papers/cav02paper.pdf

[2] Yang Y, Gopalakrishnan G, Lindstrom G, Slind K. Nemos: A framework for axiomatic and executable specifications of memory consistency models. In: Proc. of the 18th Int'l Parallel and Distributed Processing Symp. (IPDPS 2004). 2004. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1302944

[3] Gibbons P, Korach E. On testing cache-coherent shared memories. In: Proc. of the 6th ACM Symp. on Parallel Algorithms and Architectures (SPAA'94). 1994. http://delivery.acm.org/10.1145/190000/181328/p177-gibbons.pdf?key1=181328&key2=3182378621&coll=GUIDE&dl=GUIDE&CFID=82133830&CFTOKEN=76647768

[4] Gibbons P, Korach E. Testing shared memories. SIAM Journal on Computing, 1997,26(4):1208−1244.

[5] Meixner A, Sorin D. Dynamic verification of sequential consistency. In: Proc. of the 32nd Int'l Symp. on Computer Architecture (ISCA 2005). 2005. http://people.ee.duke.edu/~sorin/papers/isca05_dvsc.pdf

[6] Meixner A, Sorin D. Dynamic verification of memory consistency in cache-coherent multithreaded computer architectures. In: Proc. of the Int'l Conf. on Dependable Systems and Networks (DSN 2006). 2006. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1633497&userType=inst

[7] Hangal S, Vahia D, Manovit C, Lu J, Narayanan S. Tsotool: A program for verifying memory systems using the memory consistency model. In: Proc. of the 31st Int'l Symp. on Computer Architecture (ISCA 2004). 2004. http://ieeexplore.ieee.org/iel5/9170/29103/01310768.pdf?arnumber=1310768

[8] Manovit C, Hangal S. Efficient algorithms for verifying memory consistency. In: Proc. of the 17th ACM Symp. on Parallelism in Algorithms and Architecture (SPAA 2005). 2005. http://delivery.acm.org/10.1145/1080000/1074011/p245-manovit.pdf?key1=1074011&key2=6873378621&coll=GUIDE&dl=GUIDE&CFID=82135957&CFTOKEN=66149051

[9] Roy A, Zeisset S, Fleckenstein C, Huang J. Fast and generalized polynomial time memory consistency verification. In: Proc. of the 18th Int'l Conf. on Computer Aided Verification (CAV 2006). Berlin, Heidelberg: Springer-Verlag, 2006. 503−516.

[10] Manovit C, Hangal S. Completely verifying memory consistency of test program executions. In: Proc. of the 12th Int'l Symp. on High-Performance Computer Architecture (HPCA 2006). 2006. http://ieeexplore.ieee.org/iel5/10647/33614/01598123.pdf?arnumber=1598123

[11] Scheurich C, Dubois M. Correct memory operation of cached-based multiprocessors. In: Proc. of the 14th Int'l Symp. on Computer Architecture (ISCA'87). 1987.

[12] Goodman J. Cache consistency and sequential consistency. Technical Report, No. 61, SCI Committee, 1989.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

[13] Dubois M, Scheurich C, Briggs F. Memory access buffering in multiprocessors. In: Proc. of the 13th Int'l Symp. on Computer Architecture (ISCA'86). New York: ACM Press, 1986. 320−328.

[14] Chen YJ, Lv Y, Hu W, Chen TS, Shen HH, Wang PY, Pan H. Fast complete memory consistency verification. In: Proc. of the 15th Int'l Symp. on High-Performance Computer Architecture (HPCA'15). Raleigh, 2009. 381−392.

[15] Arvind, Maessen J. Memory model=instruction reordering+store atomicity. In: Proc. of the 33rd Int'l Symp. on Computer Architecture (ISCA2006). 2006. http://csg.csail.mit.edu/pubs/memos/Memo-493/memo-493.pdf

[16] Hu W, Gao X, Chen YJ. Micro-Architecture of Godson-3 multi-core processor. In: Proc. of the 20th Hot Chips. 2008.

[17] Hu W, Wang J, Gao X, Chen YJ, Liu Q. Godson-3: A scalable multicore—RISC processor with x86 emulation. IEEE Micro, 2009,29(2). http://filesbay.net/search/godson-3-a-scalable-multicore-risc-processor-with-x86-emulation.html

[18] Hu W. Shared Memory Architecture. Beijing: Higher Education Press, 2007 (in Chinese).

[19] Hu W. A graph model for investigating memory consistency. In: Proc. of the 8th Int'l Parallel and Distributed Processing Symp. (IPDPS'94). 1994. 516−523.

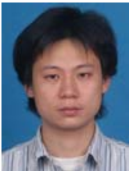
[20] Lamport L. Time, clocks, and the ordering of event in a distributed system. Communications of the ACM, 1978,21(7).

附中文参考文献:

[18] 胡伟武.共享存储系统结构.北京:高等教育出版社,2001.

王朋宇(1979－),男,河南漯河人,博士,工程师,主要研究领域为处理器设计与验证.

陈天石(1985－),男,博士生,主要研究领域为计算智能.



陈云霁(1983－),男,博士,助理研究员,主要研究领域为硬件验证,高性能计算机系统结构.

张珩(1973－),男,博士,工程师,主要研究领域为处理器设计与验证.

沈海华(1972－),女,博士,副研究员,CCF高级会员,主要研究领域为计算机体系结构,处理器设计与验证.

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn