文章编号: 1001- 2486(2008) 06- 0047- 06

基于伪临界值的 Cache 一致性协议验证方法*

屈婉霞, 郭 阳, 庞征斌, 杨晓东
(国防科技大学 计算机学院, 湖南 长沙 410073)

摘 要: 针对 Cache 一致性协议状态空间爆炸问题, 提出共享集合伪临界值(Pseudo-cutoff) 的概念, 并以采用释放一致性模型的 CC-NUMA 系统为例, 分析了共享数据的分布情况, 推导出在一定条件下共享集合伪临界值为 4 的结论, 有效优化了目录 Cache 协议状态空间, 并提出了解决小概率的宽共享事件的方法。实验数据表明, 基于伪临界值的协议模型优化, 能够有效缩小 Cache 协议状态空间, 加快验证速度, 扩大验证规模。

关键词: 形式化验证; 模型检验; 多处理机系统; Cache 一致性协议
中图分类号: TP302. 1 文献标识码: A

An Efficient Verification Method of Cache Coherence Protocol Based on Pseudo-cutoff

QU Wan-xia, GUO Yang, PANG Zheng-bin, YANG Xiao-dong
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Regarding the state space explosion problem in model checking Cache coherence protocol, the concept of pseudo-cutoff, a limit of the nodes which share the same memory block, is put forward in this paper. Based on the analysis of the inherent characteristics of parallel programs, the pseudo-cutoff value in relaxed consistency Cache coherent non-uniform memory access system under certain conditions is deduced. The state space of the directory-based Cache protocol is optimized effectively using pseudo-cutoff, and a new scheme to small probability matter of wide sharing is presented. Experimental results show that, the method of protocol model optimization based on pseudo-cutoff can effectively reduce the state space of Cache protocol, accelerate verification speed and improve the capability of verifying large-scale Cache protocol.

Key words: formal verification; model checking; multiprocessor; Cache coherence protocol

Cache 一致性协议是为实现共享数据访问一致性、提供共享存储编程接口所采用的处理机制, 不仅直接决定系统的正确性, 而且对系统的规模和性能有着重要影响, 是实现分布共享主存多处理机/多核系统的关键。系统规模的快速扩充、网络延时的不确定性和存储一致性模型的多样性等诸多因素, 使 Cache 一致性协议异常复杂, 协议的状态空间呈指数级增长, 甚至出现爆炸现象。

目前, 对 Cache 一致性协议进行验证的方法主要有三种[1]。模拟验证方法[2-3]简单有效, 但很难保证完备性。作为一种形式化验证方法, 模型检验的基本思想是通过遍历协议模型的状态空间, 自动判断协议是否满足给定性质, 当前的标准方法是在算法一级建立协议的形式化模型并在小规模配置下(典型的是 3~ 4 个结点) 对模型进行穷举式的可达性分析[4-5]或符号模型检验[6-8]。模型检验是对模拟方法的重要补充, 但是, 传统的模型检验方法只能处理有限状态系统, 不能直接应用于可扩展 Cache 协议等无限状态系统。定理证明是另一种形式化验证方法, 高阶逻辑表达能力很强, 但证明过程太复杂, 需要大量的专家指导, 另一个主要问题是定理证明方法不能完全自动化。

针对基于目录的 Cache 一致性协议的模型检验, 本文提出共享集合伪临界值(Pseudo-cutoff) 的概念, 并以采用释放一致性模型的 CC-NUMA 系统为例, 分析了共享数据的分布情况, 推导出在一定条件下共享集合伪临界值为 4 的结论, 基于此结论有效优化了 Cache 协议状态空间, 并提出了解决小概率的宽共

* 收稿日期: 2008- 08- 26
基金项目: 国家自然科学基金资助项目(60573173, 60773025); 新世纪优秀人才支持计划资助项目
作者简介: 屈婉霞(1972—), 女, 助理研究员, 博士生。

享事件的方法。不同规模的 FLASH[9] 协议和 M _ FLASH 协议实验结果表明, 基于伪临界值的协议模型优化, 能够有效缩小 Cache 协议的状态空间, 加快验证速度, 扩大验证规模。

1 Cache 一致性协议模型

本文描述的基于目录的 Cache 一致性协议, 采用写失效和写回策略, 术语和基本协议流程与 Stanford 大学的 FLASH 协议[9] 相同, 但存在以下两个不同点:

(1) 当一个请求被 Home 结点转发到拥有者结点时, 网络延迟可能导致拥有者结点的 Cache 状态已不再是独占态, 我们称这样的结点为“虚假独占结点”。本文描述的协议约定转发到虚假独占结点的请求, 由 Home 结点根据目录所处的状态产生相应的响应。

(2) 协议只支持处理器替换“脏” Cache 块的随机行为, 即协议在处理器替换共享 Cache 块时不对外流出任何请求。

2 系统规范

系统规范是用某种逻辑描述的待验证性质, 以下公式描述了设计人员最关心的协议的安全性质。

P1: $\forall p, q.(p \neq q \wedge p.CacheState = E \Rightarrow q.CacheState \neq E)$

P2: $\forall p.((p.CacheState = E \Rightarrow p.CacheData = CurrData)\ \&$

$(p.CacheState = S\&Collecting \Rightarrow p.CacheData = Prevdata)\ \&\ (p.CacheState = S\&!Collecting \Rightarrow p.CacheData = CurrData)$

P3: $!Dir.Dirty \Rightarrow Mem.Data = CurrData$

P4: $\exists p.(Dir.HeadVld \wedge Dir.Dirty \wedge Dir.HeadPtr = p \Rightarrow p.CacheState \neq E)$

第一个公式是与 Cache 状态相关的性质, 表示不存在两个结点同时独占一个数据项的情况, 是所有 Cache 协议必须验证的性质。第二个公式是与 Cache 数据相关的性质, 表示当结点独占数据项时, Cache 中的数据与最近写入的数据相同, 当结点共享数据项时, 如果正在等待失效响应, 则 Cache 中的数据与前一系统状态中的数据相同; 否则, 与当前系统状态中的数据相同。第三个公式是与主存数据相关的性质, 表示当目录不脏时, 主存中的数据与最近写入的数据相同。第四个公式表示系统中存在虚假独占结点, 即结点 Cache 的状态不为 E, 但目录状态为脏且独占结点指针指向该结点。此性质用于验证网络传输延迟造成的系统瞬态。

3 伪临界值

从协议模型的描述可以看出, 目录项中共享集合的大小与系统结点数相等, 每一项表示相应的结点是否共享该目录项对应的存储器块, 若系统包含 N 个结点和 M 个存储器块, 则每一个存储器块的共享情况有 2^N 种, 全系统存储器块的共享情况有 2^{NM} 种, 系统规模和并行应用程序的访存行为共同影响着共享集合的值域, 是导致目录 Cache 协议状态空间爆炸的重要原因之一。

从理论上讲, 同一时刻一个并行应用程序的所有任务共享同一存储器块是可能的, 但在实际应用中, 这种极端情况并不存在, 是一种不可达状态。在协议验证过程中, 我们希望能够剔除这些不可达状态。那么, 是否存在一个小于系统规模的临界值(Cutoff), 在大多数实际的并行应用程序中, 共享同一个存储器块的结点数不超过此临界值, 而超过此临界值的情况发生的概率非常小, 我们称这种临界值为伪临界值(Pseudo-cutoff)。下面以 CC-NUMA 系统为例, 给出 Cache 一致性协议共享集合伪临界值的推导过程。

设 CC-NUMA 系统 N 个处理器的编号从 1 到 N, 存储器按 Cache 块的大小划分为若干块, 对某个指定存储器块, 允许多个处理器共享访问, 但是对该存储器块的修改只能由一个处理器进行。为简化分析, 假定多任务程序具有一致的访存特性且对共享变量的访问是互斥的, 若程序访存概率为 p, 在此概率下的条件读概率为 r, 条件写概率为 $w = 1 - r$, 则多 Cache 工作状态转移如图 1 所示。

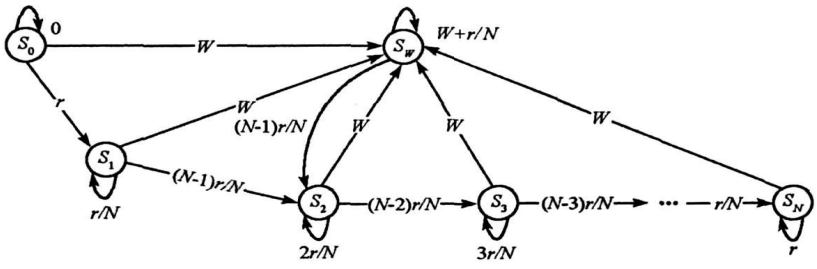


图 1 CC-NUMA 系统多 Cache 工作状态转移图

Fig. 1 Multiple Caches state transition graph in CC-NUMA system

对同一存储器块 A, 其 Cache 块有读共享和写独占两类状态。S_0 表示没有共享副本的初始状态, S_k $(1 \leqslant k \leqslant N)$ 表示有 k 个共享副本的共享状态, S_W 表示 A 被独占修改的状态。程序开始执行后, 系统状态由 S_0 开始, 然后一直在 S_k 和 S_W 之间进行转换, 而且 S_k 之间的状态变化总是依次递增的。状态 S_k 向 S_{k-1} 转移的概率为 $(N-k)r/N$ $(k=1, 2, \cdots, N-1)$。状态 S_W 向 S_2 转移的概率为 $(N-1)r/N$, 而状态 S_k 向 S_W 转移的概率均为 $w=1-r$。因此, 状态 S_k 出现的概率表示系统中存储器块 A 有 k 个共享副本的概率, S_W 则表示处理器独占修改存储器块 A 的概率。

S_0、S_k 到 S_W 的循环变化过程可看成一个有限状态马尔可夫过程, 当 $w>0$ 时转移是各态遍历的并且经过一段时间后该过程将到达平衡状态, 我们用 $\pi_0, \pi_1, \cdots, \pi_W$ $(W=N+1)$ 表示相应状态的一步转移概率, 根据多 Cache 工作状态转移图的一步转移概率矩阵可以求出所有的 π_k $(1 \leqslant k \leqslant W)$。

当系统处于共享状态 S_k $(1 \leqslant k \leqslant N)$ 时, 目录需要记录 k 个不同结点的信息。在整个马尔可夫链变化过程中, 目录需要记录结点信息的平均个数, 也即系统的平均共享副本数目为: $L=\pi_1 \times 1+\pi_2 \times 2+\cdots+\pi_N=\sum_{k=1}^{N}(\pi_k \times k)$。在 $r \leqslant 2/3$ 或 $w>1/3$ 这一合理假设条件下, 模拟结果显示 $L<5$, 且当结点数 N 较大时 L 与 N 无关。Gupta[10] 等也对 CC-NUMA 系统上应用程序的共享频度和共享集合大小进行了模拟、统计和分析, 得到了相似的结论。

因此我们得到以下结论: 在大规模 CC-NUMA 系统中, 对某个指定存储器块, 若并行应用程序的读概率 r 满足 $r \leqslant 2/3$, 则该块在 Cache 中的平均共享副本数最多为 4 个, 且当系统规模较大时平均共享副本数与系统规模无关。我们称 4 为共享集合的伪临界值。

4 基于伪临界值的协议状态空间简化

N 和 M 是影响目录 Cache 协议状态空间规模的主要因素, N 决定共享集合的长度, 而 N 和 M 共同决定一个存储器块的共享分布情况。共享分布的情况越多, 协议验证工作就越多, 在 N 和 M 增长越来越快的趋势下, 伪临界值为优化协议模型提供了有力的理论依据。

4.1 删除冗余状态

我们将共享集合的最大长度限制为 4: 当 N 小于 4 时, 共享集合的长度与 N 相等; 否则, 共享集合的长度与伪临界值相等。此方法使一个存储器块的共享情况由 2^N 种减少到 $(N^4-2N^3+11N^2+14N+24)$ 种, 协议模型状态空间与系统规模之间的指数关系转变为多项式关系, 设计一个多项式时间的验证算法成为可能。

与共享集合密切相关的另一个状态变量是失效集合, 记录被通知失效其 Cache 块的结点, 与共享集合的大小相等, 我们采用与共享集合相同的方法定义该变量, 相应的伪代码描述如下, 其中, ShrSet 表示共享集合, InvSet 表示失效集合。

```
# define Pseudo _ Cutoff 4
if (N < Pseudo _ Cutoff)
```

```
    Boolean ShrSet [N];
    Boolean InvSet [N];
else
    Boolean ShrSet [Pseudo _Cutoff];
    Boolean InvSet [Pseudo _Cutoff];
end
```

4.2 对称化简

尽管伪临界值的推导是以某一个存储器块为对象进行的,但是关于伪临界值的结论适用于系统中每一个存储器块。在不考虑存储器块编号顺序的情况下,所有存储器块在系统中的地位完全相同,一个存储器块的共享情况不影响其他存储器块的共享,即两个不同存储器块的共享情况存在对称性。

正是由于存储器块共享情况的对称性,可以把对所有 M 个存储器块的数据一致性验证化简为对一个存储器块的数据一致性验证,因此,协议模型通常只包含一个存储器块和一个 Cache 块,当其他状态变量取值不变、$N>$ Pseudo _Cutoff 时,优化后的协议模型状态数比优化前减少 $2(M-1)[2^N-(N^4-2N^3+11N^2+14N+24)/24]$,显著缩小了协议模型的状态空间。随着系统规模和共享主存容量的不断增大,减少的协议模型状态数相当可观,优化效果也更明显。

4.3 解决小概率事件

共享集合伪临界值是在并行应用程序满足合理假设条件下的一个平均值,因此,在此假设成立或者不成立时,并行应用对某个存储器块的共享副本数仍有可能超过伪临界值。从模拟和相关研究的测试数据可知,宽共享发生的概率非常小,对于这种情况,我们的解决方案是:共享集合的大小设置为伪临界值,某个存储器块的所有共享结点"分享"长度有限的共享集合,当共享副本数超过伪临界值时,协议采用类似Cache替换的策略,将最久未共享该存储器块的结点从共享集合中替换出去,同时通知被替换的结点修改 Cache 状态。

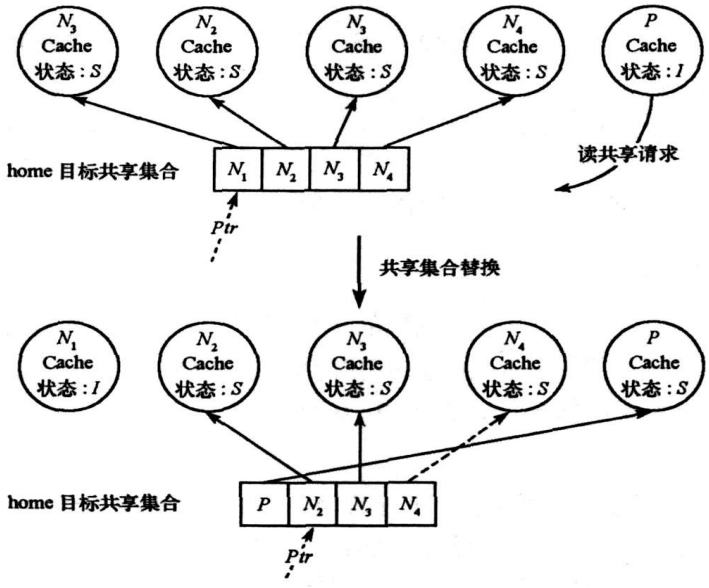


图2 验证小概率事件示意图

Fig. 2 Scheme for wide sharing verification

图2描述了验证小概率事件的过程。最初,结点 N_1、N_2、N_3 和 N_4 先后共享 home 结点中的同一个存储器块,指针变量 Ptr 指向最早共享该存储器块的结点 N_1,这4个结点的 Cache 状态均为 S 态。当第5个结点 P 因为出现读失效向 home 结点发出读共享请求时,home 结点发现共享集合已满,则首先通知 Ptr 指向的结点 N_1 失效其Cache中的共享副本,Cache 状态转换到 I 态,然后响应结点 P 的请求,并将 P

加入共享集合, 结点 P 的 Cache 状态转换到 S 态, 此时 Ptr 指向结点N_2。如果结点 N_1 再次请求该存储器块时, 由于其本地 Cache 中没有共享副本, 结点 N_1 重新向 home 发出读共享请求, 下一个将被替换出的结点将是N_2。

5 应用与实验结果

我们用 Murphi 描述语言实现了前文所述的协议模型和系统规范(命名为 M _ FLASH), 并用第 4 节的方法对其进行了优化。标准 FLASH(SV) 使用全映射的位向量表示共享集合, 优化的M _ FLASH(OV) 用伪临界值控制共享集合的长度。

我们在 cygwin 环境下利用 Murphi 验证系统[11] 对不同版本、不同规模的协议进行了验证和结果分析。实验环境参数如下: 硬件配置为主频 1.6GHz 的 Pentium 处理器、内存 1GB 的 PC 机。Murphi 编译器的参数– c 指定以压缩方式保存状态项。为帮助分析协议模型对状态空间的需求, 我们为相同规模的协议分别提供 32MB、128MB 和 512MB 的最大内存空间来保存可达状态。

表 1 列出了系统规模为 2~ 6 时的实验数据。其中, N 表示协议规模(结点数目), "验证结果" 一列下的"√"表示系统规范 P1~ P4 均正确, "×" 表示验证过程因内存不足停止, "验证时间" 以 s 为单位, 最后一列是验证停止时遍历的协议状态数目。

表 1 FLASH 协议与 M _ FLASH 协议实验结果

Tab.1 Experimental results on FLASH and M-FLASH

N	内存阈值(MB)	验证结果(SV/OV)	验证时间(s)(SV/OV)	验证停止时遍历的状态(SV/OV)
2	32	√/√	0.59/0.23	3096/678
	64	√/√	0.67/0.26	3096/678
	128	√/√	0.73/0.45	3096/678
	256	√/√	0.92/0.69	3096/678
	512	√/√	2.53/1.58	3096/678
3	32	×/√	148.73/7.67	521 000/29 077
	64	√/√	183.51/7.67	628 797/29 077
	128	√/√	182.28/7.78	628 797/29 077
	256	√/√	185.39/7.98	628 797/29 077
	512	√/√	228.78/8.48	628 797/29 077
4	32	× / ×	25.78/237.83	109 000/490 000
	64	×/√	54.61/233.73	225 000/506 021
	128	×/√	129.47/238.14	492 000/506 021
	256	×/√	311.28/231.59	107 8000/506 021
	512	×/√	829.31/269.97	244 2000/506 021
5	32	× / ×	31.84/62.25	84 000/130 000
	64	× / ×	76.83/139.22	176 000/302 000
	128	× / ×	151.22/351.61	368 000/706 000
	256	× / ×	298.73/1119.56	748 000/1 974 000
	512	×/√	659.64/7477.05	1 575 000/6 690 625
6	32	× / ×	41.94/56.84	70 000/98 000
	64	× / ×	85.25 /141.12	146 000/213 000
	128	× / ×	178.17/300.23	297 000/459 000
	256	× / ×	371.50 /708.90	602 000/997 000
	512	× / ×	801.12/1575.64	1 232 000/2 262 000

在硬件资源足够的情况下(浅灰色行指示), FLASH 和 M _ FLASH 都能够验证系统规范, 而且验证结果一致, 但是 M _ FLASH 遍历的状态数更少: N= 2 时, M _ FLASH 只遍历了 678 个状态就完成验证任务, FLASH 则在遍历 3096 个状态后才结束; 而在 N= 3 时, M _ FLASH遍历的状态数仅为 FLASH 遍历状态数的 4.6%。验证停止时遍历的状态数等于协议的状态空间大小, 可以看出, 由于采用了优化技术,

M_FLASH的状态空间远远小于FLASH的状态空间,$N=2$和$N=3$时,二者之比分别为21.9%和4.6%,这也正是M_FLASH验证速度明显快于FLASH的原因。

深灰色行中数据表示FLASH的状态空间太大,耗尽了给定内存,导致不能完成验证,而M_FLASH则能够完成验证任务。$N=3$时,FLASH完整的状态空间包括628 797个状态,32MB的内存只能存放521 000个状态,因此验证失败。$N=4$时,我们无法获得FLASH协议完整的状态空间大小(但可以肯定远远大于506 021),即使给定512MB的内存,FLASH在运行829.31s、遍历2 442 000个状态后也以失败告终。

其他无阴影行数据表示,随着协议规模继续增大,协议状态空间出现爆炸,现有的硬件资源无法保证验证任何一个协议。但是,从验证停止时遍历的状态数,我们发现,M_FLASH用同样的内存遍历了更多的状态,运行的时间更长,再次证明了我们采用的优化方法是有效的。

在能够完成验证的情况下,表中最后一列数据显示,FLASH和M_FLASH的状态空间大小随协议规模(结点数)的增加快速增长(分别为3096→628 797和678→29 077→506 021→6 690 625)。Murphi最多验证包含3个结点的FLASH协议(耗时228.78s),却能够验证包含5个结点的M_FLASH协议(耗时7477.05s)。从上述分析可知,只要有足够的内存空间,M_FLASH可以验证更大规模的协议,而协议状态空间的非线性增长方式很难保证验证包含更多结点的FLASH协议。

6 结论

不同规模的FLASH协议和M_FLASH协议验证结果表明,基于伪临界值的协议模型优化技术能够有效缩小目录Cache协议的状态空间,加快验证速度,扩大验证规模。但是,当系统规模扩大到一定程度时,局部的优化仍然不能保证完成验证任务,需要采用其他方法进一步缓解状态空间爆炸问题。

参考文献:

[1] Pong F, Dubois M. Verification Techniques for Cache Coherence Protocols[J]. ACM Computing Surveys, 1997, 29(1): 82-126.

[2] Adir A, Shurek A G. Generating Concurrent Test-programs with Collisions for Multiprocessor Verification[C]//Washington DC, USA: IEEE Computer Society, 2002.

[3] Sorin D J, Hill M D, Wood A D A. Dynamic Verification of End-to-end Multiprocessor Invariants[C]//International Conference on Dependable Systems and Networks (DSN'03), San Francisco, CA, USA, 2003.

[4] Chen X F, Gopalakrishnan G. A General Compositional Approach to Verifying Hierarchical Cache Coherence Protocols[R]. Salt Lake City: University of Utah, 2006.

[5] Chen X F, Yang Y, Gopalakrishnan G, et al. Reducing Verification Complexity of a Multicore Coherence Protocol Using Assume/Guarantee[C]// Formal Methods in Computer Aided Design (FMCAD2006), San Jose, IEEE Computer Society, 2006.

[6] Martin M M K. Formal Verification and Its Impact on the Snooping Versus Directory Protocol Debate[C]//Proceedings of the 23rd International Conference on Computer Design (ICCD'05), San Jose, CA, USA, IEEE Computer Society, 2005.

[7] Plakal M, Sorin D J, Anne E, et al. Lamport Clocks: Verifying a Directory Cache-coherence Protocol[C]//The 10th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), Puerto Vallarta, Mexico, 1998.

[8] Emerson E A, Kahlon V. Exact and Efficient Verification of Parameterized Cache Coherence Protocols[J]. Lecture Notes in Computer Science, 2003, 2860: 247-262.

[9] Mcmillan K L. Parameterized Verification of the FLASH Cache Coherence Protocol by Compositional Model Checking[C]//Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods, London, UK, 2001.

[10] Gupta A, Weber W D. Cache Invalidation Patterns in Shared-memory Multiprocessors[J]. IEEE Trans. Comput., 1992, 41(7): 794-810.

[11] Dill D L, Drexler A J, Hu A J, et al. Protocol Verification as a Hardware Design Aid[C]//Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computers and Processors, Cambridge, MA, USA, IEEE Computer Society, 1991.