经典论文《高速缓存的最优划分》要点归纳与启发

关键词：高速缓存　划分

宋逸斐[1]　刘虎林[2]　刘宇航[1]
[1] 中国科学院计算技术研究所
[2] 哈尔滨师范大学

引言

存储墙问题（memory wall problem）是计算机体系结构领域的一个核心问题，这一问题在上世纪90年代初被原美国工程院院长威廉·沃尔夫（William Wulf）教授和他的学生萨利·麦基（Sally McKee）[1]教授正式提出[1]，在此之前，现任澳门大学副校长倪明选教授和他的学生孙贤和教授提出了存储受限的加速比模型[2]。随着时间的推移，此问题越来越严重（尤其是近年来访存密集型应用越来越多，片上的处理器核心和线程数量越来越多，导致数据访问需求急剧增加），笔者所在的研究组长期聚焦于此问题的研究，在近10年发表了多篇相关论文[3~7]，一些关键技术已经应用到业界一线产品之中，这些论文或技术有一个共同的特点，就是都涉及到对存储访问的局部性（Locality）和并发性（Concurrency）的优化。高速缓存是解决存储墙问题的一种重要方法，相应地，层次化的高速缓存是现代计算机体系结构的一种重要特征。

划分（Partitioning）是多进程或多应用共享资源必然会面临的问题。论文《高速缓存的最优划分》（“Optimal Partitioning of Cache Memory”）[8]是计算机体系结构领域的一篇经典论文，1992年发表于*IEEE Transactions on Computers*，作者是IBM托马斯·J·沃森研究中心的哈罗德·S·斯通（Harold S. Stone）、约翰·图雷克（John Turek）和乔尔·L·沃尔夫（Joel L. Wolf），曾入选克里希纳（C. M. Krishna）教授精选的计算机体系结构性能建模经典论文集[9]（一共27篇）。**在这篇论文中，作者构建了一个在多个进程争用资源的情况下研究高速缓存最优划分的数学模型，并基于该模型，在两种不同的场景下，评估最近最少使用（Least Recently Used, LRU）替换策略的高速缓存划分效果与最优划分的差距。**

在第一种争用场景中，除了研究根据高速缓存划分达到稳定时的状态来评估替换算法的性能表现与最优划分的差距，还进一步研究了高速缓存划分在程序行为发生变化后的动态变化过程。作者发现当系统中程序的行为特征发生显著变化时，**高速缓存对应的分配稳定点也发生改变，此时系统高速缓存分配会迅速地向稳态方向移动，但是越接近稳态分配，收敛速度会越慢**。为了解决这个问题，作者提出了一种能直接计算出最优分配状态的算法，以解决收敛缓慢的问题。对于分层高速缓存，作者提出了一种算法，

[1] 萨利·麦基教授与笔者所在团队有过合作交流。

计算每一层的最优划分方案。

在第二种争用场景中，研究者发现 LRU 替换策略导致被暂停的进程的高速缓存中的数据被接下来运行的进程污染、替换，导致之前被暂停的进程重启后，这些被替换的数据必须重新载入到高速缓存中，缺失率大幅上升。为了解决这个问题，作者提出了一种优于 LRU 的高速缓存替换策略。在这个替换策略中，作者提出了一个与进程占用的高速缓存大小相关的阈值，仅在一个进程剩余的访存次数高于该阈值时，分配新的高速缓存空间给该进程（否则进程只能替换原本就被自己占用的高速缓存空间），从而**减轻了多进程在使用高速缓存时的相互污染**。

这篇近 30 年前的经典论文对于现在众核片上系统和云计算数据中心高效地利用共享资源具有重要的启发意义。通过对该问题近 30 年研究成果的前后对比，我们发现**问题几乎没变，但方法、结论有较大变化**，前后对比综合，有可能获得一些有价值的研究思路。

争用场景一：多进程同时争用高速缓存

场景描述

当多个进程同时争用高速缓存时，这些进程对高速缓存的访存流可划分为两类，一类是以获取数据为目的，一类是以获取指令为目的。因此可以将所有的访存流视为两个互相交错的访存流，一个负责获取数据，一个负责获取指令。将这两种访存流看作两个独立的“进程”同时争用高速缓存资源，从而用这两个进程的访存特征代表整个系统的访存特征。其中数据流访问频率和指令流访问频率的比值为 r，在接下来的数学推导中为了简化计算，将 r 设定为 1，否则需要在 $M_D(x)$（数据流的缺失率）前增加一个 r 作为系数。

进程特征

为了研究进程的特点，建立完整的数学模型，需要一个函数来描述对应的高速缓存性能。一个高速缓存的性能取决于命中率，命中率越大，即缺失率越小，高速缓存的性能越强。因此作者选用了进程被分配的高速缓存部分的缺失率作为这个进程的特征函数，测试不同的高速缓存分配尺寸下，指令流进程的缺失率。从实验数据可发现，函数特征比较接近指数特征，因此尝试对两个坐标轴取不同的对数来实现拟合，发现对数标度图像很接近于线性，因此可拟合成线性函数，这样得到了所有进程的特征函数——缺失率较准确的拟合结果。

稳定状态研究

计算最优划分

对一个高速缓存来说，缺失率能够代表这个高速缓存的性能，因此作者用高速缓存的总缺失率来量化高速缓存的最优性（optimality）。因此作者定义最优划分（optimal partition）就是使高速缓存总缺失率最低的分配方案。作者假设缺失率函数是连续且可导的。x 代表指令流分配高速缓存的尺寸，C 代表高速缓存总尺寸，最优划分意味着总缺失率最小，当总缺失率函数的导数等于零时可以获得此时的划分状态：$dM_I(x)/dx+dM_D(C-x)/dx=0$。

简化策略Modified-LRU

为了更好地研究 LRU 替换策略的特征，作者提出了一个简化版的 LRU 策略，称为 modified-LRU。该策略在指令流进程发生缺失时，只会从被数据流进程占用的高速缓存中选择替换目标，反之亦然。这样一来，进程占用的高速缓存尺寸增加的概率等于该进程发生缺失的概率，从而简化计算。对于 LRU 替换策略，当高速缓存发生缺失时，替换目标会从整个高速缓存中选取，因此进程占用的高速缓存尺寸增加的概率不等于缺失率，而是等于该进程访问高速缓存发生缺失并且替换策略选择的替换目标此前未被此进程占用的概率。用高速缓存的划分比例估算替换目标位于被其他应用占用的高速缓存区域的概率，这个概率将作为系数影响后续的模型推导：

$$Prob(victim\ occupied\ by\ other\ processes) = (C-x)/C \quad (1)$$

稳态特征

进程占用的高速缓存大小 x 是一个随时间变化的

随机变量，称 x 高速缓存的分配状态，因此特征化分配的状态必须要量化 x 的概率，设为函数 $S(x)$。假设这个随机过程已经运行了足够长的时间，瞬态响应已经结束，此时这个高速缓存处于统计学稳定的状态，即当前进程占用的高速缓存大小从 x 增加到 x+1 的概率等于从 x+1 减小到 x 的概率。

$$Prob[x\ increases\ to\ x{+}1] = S(x)M_I(x) \quad (2)$$

$$Prob[x{+}1\ decreases\ to\ x] = S(x{+}1)\,M_D\,(C{-}(x{+}1)) \quad (3)$$

$$\frac{S(x+1)}{S(x)} = \frac{M_I(x)}{M_D(C-(x+1))} \quad (4)$$

根据等式（4），由于 $M_I(x)$ 是单调递减函数，$M_D(C-x)$ 是单调递增函数，因此 $S(x{+}1)/S(x)$ 是单调递减的，结合 $S(x)$ 在区间 [0, C] 内不小于 $S(0)$，可知该函数是一个先增后减的单峰函数。**因此满足 $M_I(x)$ $M_D\,(C-(x{+}1))$ 的第一个点就是概率最大的峰值点，也就是统计学的稳定点**。由于 $S(x)$ $S(x{+}1)$，因此可以通过 $M_I(\tilde{x}) \approx M_D(C-(\tilde{x}+1))$ 近似求出稳定点 $\tilde{x}$。从 modified-LRU 策略到 LRU 策略，只需要多考虑一个替换目标位于哪个进程占用的高速缓存空间内。与 modified-LRU 峰值点的计算同理，可以通过 $(1-\frac{\hat{x}}{C})M_I(\hat{x}) \approx (\frac{\hat{x}+1}{C})M_D(C-(\hat{x}+1))$ 近似求得稳定点 $\hat{x}$。由于 $S(x)$ 是概率密度函数，因此 $\int_0^C S(x)dx = 1$，由此可以绘制出两个模型对应的函数图像，图中峰值点既是出现概率最高的，也是该进程占用高速缓存大小的平均值，因此这个位置最能代表该替换策略下高速缓存的分配状态。

评估LRU替换策略在这个问题中的表现

作者比较了最优替换策略（optimal policy）和两种 LRU 替换策略对高速缓存划分的效果。最优替换策略无法进行模拟测试，用前文提到的最优划分模型估算最优策略的划分效果。两种 LRU 策略选择上文推导出的稳定分配状态和作为模型的结果，并且使用踪迹驱动模拟（trace-driven simulation）测试了在两种替换策略下的高速缓存划分的稳定状态，可以看到 LRU 策略在所有的频率比值下的缺失率都很接近最优替换策略分配结果的缺失率，这说明 LRU 策略性能表现良好。

作者进一步分析了 LRU 策略表现接近最优划分的原因：（1）在这个测试例子中 LRU 的稳定值 $\tilde{x}$ 接近于总高速缓存尺寸 C 的一半，导致权重 $\frac{\tilde{x}+1}{C}$ 和 $1-\frac{\tilde{x}}{C}$ 接近相等，平衡点和最优点接近；（2）测量不同分配大小下的丢失率变化可知，在最优点附近，缺失率对高速缓存分配尺寸的变化并不敏感。并不能说明 LRU 策略在任何情况下的划分效果都接近最优划分，这一节的重点在于提出了如何使用数学模型评估替换策略对高速缓存划分的表现。

高速缓存分配状态的动态变化研究

由稳态转向瞬态

上一节研究了系统在运行足够长时间使系统内的瞬态变化结束，高速缓存分配达到稳态后，对不同的替换策略模型构建和表现评估。而这一节主要研究当程序行为特征发生变化，导致稳态被破坏时，处于瞬态响应中的高速缓存分配状态的变化过程。

通过导函数建立瞬态模型

从相对简单的 modified-LRU 替换策略着手，如上文所说，在这个替换策略下进程每发生一次访问缺失就会增加占用的高速缓存，因此增加和减少的速率可以分别用自己和其他进程的缺失率表示。当进程特征发生变化时，高速缓存分配状态会快速地靠近对应的稳定分配，但是越接近稳定分配速度就会越慢，也就是出现了收敛缓慢的问题。

优化算法

为了解决收敛缓慢的问题，作者提出了一个能够直接计算出最优划分的算法：当 N 个进程争用共享的高速缓存时，共享高速缓存的缺失率可以表示为：

$$\sum_{i=1}^{N} P_i M_i(C_i)/T$$

其中，C_i 代表进程 i 被分配到的高速缓存大小，$M_i(C_i)$ 代表这个进程的缺失率，P_i 代表这个进程的访存次数，T 是所有进程的访存次数总数。将所有的 C_i 都设定为 0，然后依次测试不同的 C_i 增加 1 使总缺失率降低的幅度，选择幅度最大的 C_i 增加 1。重复以上的操作，直到所有的 C_i 相加等于高速缓存总尺寸，即获得使整体缺失率最低的分配策略。

将算法拓展到多层高速缓存

这里以两层的高速缓存为例（从单层高速缓存

拓展到多层，主要需要考虑的是上一层高速缓存的尺寸会怎样影响下一层的缺失率）。在双层高速缓存中，所有的访问根据行为特征可被分为三类：（1）在第一层高速缓存中就获得数据的访问，访问时间为 t，发生概率为 $(1-M(c))$。（2）在第一层高速缓存中没能找到数据，在第二层高速缓存中获得了数据的访问，访问时间为 T，发生概率为 $M(c)(1-M(c, C))$。（3）在两层高速缓存中都发生了缺失，在主存中获得了数据的访问，访问时间为 τ，发生概率为 $M(c)M(c, C)$。其中，C 是第二层高速缓存的尺寸，c 是第一层高速缓存的尺寸，$M(c, C)$ 是在第一层高速缓存中发生缺失的数据再次在第二层高速缓存中发生缺失的概率。

设第一层高速缓存中与第二层高速缓存重合部分的比例为 α，访问连续发生两次缺失的概率可以看作是两个高速缓存合并为一个，并且去除重复部分之后的访问，在这个新的高速缓存中发生缺失的概率为 $M(c)M(c, C) = M(C+(1-\alpha)c)$，通常 c 要远小于 C，因此可以忽略整体缺失率中 c 的影响，$M(c, C) = M(C+(1-\alpha)c)/M(c)$ $M(C)/M(c)$。

由于两种高速缓存的访存时间不同，纯粹用总缺失率作为最优性变得不合理，因此选择使用平均访存时间作为新的最优性度量指标。因此，最优划分就是使得平均访存时间最小的高速缓存划分。根据三种不同的访存方式对应的访存时间和出现概率，可以得到平均访存时间。对于双层高速缓存来说，可以计算出在不同的分配策略下访问需要的平均时间。

争用场景二：多进程轮流访问高速缓存

场景描述

在该场景中，存在多个进程争用高速缓存资源，但与争用场景一不同的是，进程之间是串行访存高速缓存的，而不是同时运行并争用高速缓存资源。作者假设有两个进程轮流单独访问高速缓存，缺失率分别为 $M_1(x)$ 和 $M_2(x)$，每个进程每次运行时总会对高速缓存进行 Q 次访问。

轮流访问造成的问题

当进程2接替进程1时，会污染进程1的高速缓存，因此当进程 1 重新开始时，重新装载被污染的高速缓存会导致缺失率上升，尤其是当高速缓存尺寸较大，足以同时装载两个进程的工作集时。评估进程 2 替换掉原本属于进程 1 的一个高速缓存行带来的影响：当进程 2 替换掉原本属于进程 1 的一个缓存行时，如果接下来进程 2 的运行时间内还有 q 次访问操作，并且当进程 1 被重新装载之后被替换掉的缓存行会造成的额外缺失次数为 p 次，可计算出这次替换会导致缺失率降低。因此当这个降低值大于零时，这个替换行为是有益的，但是当降低值小于零时，这个替换行为反而会造成系统整体的缺失率上升。

解决方案：限制占用高速缓存的尺寸

根据上述特征，作者定义了一个阈值 $q(x)$，当且仅当进程剩余的访问次数小于阈值时，替换其他进程占用的高速缓存行能够降低整体的缺失率。因此当进程剩余运行时间内的剩余访问次数低于阈值时，则不再增加这个进程的高速缓存，只能够替换已经被自己占用的高速缓存行，这样能够避免污染其他进程的高速缓存空间。

有效性验证

为了验证上述数学模型是否可靠有效，论文使用踪迹驱动模拟测试了上述模型是否符合实际。实验中使用了多道程序（multi-programmed）的负载，实验包含超过 1800 万次访问。以缺失率函数为例，可发现除了指令流 32 KB 以后的部分，指令流的其他部分以及数据流的模拟结果都非常接近拟合出来的线性函数。考虑到在实际应用场景中，在论文写作的上世纪 90 年代，指令流很少会占用超过 32 KB 的高速缓存，因此作者忽略了 32 KB 以后的指令流的模拟结果。经测试，指令流和数据流拟合的相似系数分别为 0.994 和 0.984。论文对高速缓存分配状态的动态变化、分配状态的概率分布、高速缓存占用

增大的概率与高速缓存分配状态之间的关系也进行了踪迹驱动模拟实验。

技巧与启发

这篇论文蕴藏了一些有价值的显式建模的方法和技巧，与现在的通过黑盒子机器学习进行资源调度的方法形成了鲜明的对比：

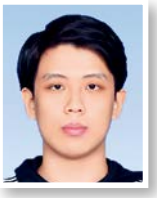
1. **可解释性模型**。作者通过观察实测数据的结果，寻找数据中体现出来的特征和规律，从而将体系结构中某一部分的行为特征拟合成具体的数学函数或其他形式的数学模型（例如对高速缓存缺失率函数的拟合）。作者以拟合出的模型为基础，通过数学推导得到新的与期望分析的过程相关的数学模型，通过观察模型，分析这个过程的特征信息，例如用缺失率函数推导出高速缓存分配状态在瞬态响应中的动态变化过程。

2. **统计学和随机过程**。由于计算机是一个不断变化的系统，因此分析计算机的状态或者行为，可以从统计学的角度着手，将进程对高速缓存的占用作为一个随机过程看待，例如文中研究进程占用的高速缓存大小 x，就是将 x 看作一个概率密度函数为 $S(x)$ 的随机过程。

3. **由简入繁**。文中分析 LRU 替换策略时，并不是直接推导 LRU 自身对应的数学模型，而是提出了一个简化的 modified-LRU 策略，在完成这个策略的数学模型后，分析它和真正 LRU 策略的差异，然后补充缺失的信息或参数。

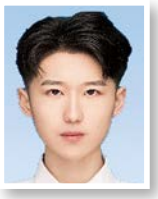
划分是多进程或多应用共享资源必然会面临的问题，如何划分、隔离至今仍是亟待解决的问题。对比现在和 30 年前，可以发现问题几乎没变，但方法、结论有较大变化，LRU 在很多时候表现不佳，作者在文章即将结束时提到的 5 个开放性问题至今仍值得研究，近 30 年来学术界先后提出了各种替换算法 [10, 11]，研究者通过前后对比综合可以获得一些有价值的研究思路和灵感。 ■

（本文责任编委：**郭得科**）



宋逸斐

中国科学院计算技术研究所研究生。主要研究方向为计算机体系结构、高性能计算。



刘虎林

哈尔滨师范大学物联网专业本科生。主要研究方向为计算机体系结构、嵌入式操作系统。

刘宇航

CCF 高级会员，*CCCF* 特邀译者，CCF 职业伦理与学术道德委员会常委。中科院计算所副研究员、硕士生导师。3 liuyuhang@ict.ac.cn

参考文献

[1] Wulf W A, McKee S A. Hitting the Memory Wall: Implications of the Obvious[J]. *ACM SIGARCH Computer Architecture News*, 1995, 23(1): 20-24.

[2] Sun X, Ni L M. Another View on Parallel Speedup[C]// *Proc. of International Conference for High Performance Computing, Networking, Storage and Analysis 1990* (SC'90). IEEE, 1990: 324-333.

[3] Liu Y, Sun X. Reevaluating Data Stall Time with the Consideration of Data Access Concurrency[J]. *Journal of Computer Science and Technology* (*JCST*), 2015, 2: 227-245.

[4] Liu Y, Sun X. C2-bound: A Capacity and Concurrency driven Analytical Model for Manycore Design[C]// *Proc. of International Conference for High Performance Computing, Networking, Storage and Analysis 2015.*

[5] Liu Y, Sun X. Evaluating the Combined Effect of Memory Capacity and Concurrency for Many-core Chip Design[J]. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* (*TOMPECS*), 2017.

[6] Liu Y, Sun X. LPM: Concurrency-driven Layered Performance Matching[C]// *Proc. of the 44th International Conference on Parallel Processing* (ICPP'15).

[7] Liu Y, Sun X, Wang Y, et al. HCDA: From Computational Thinking to a Generalized Thinking Paradigm[J]. *Communications of the ACM,* 2021, 64(5): 66-75.

[8] Stone H S, Turek J, Wolf J L. Optimal partitioning of cache memory[J]. I*EEE Transactions on Computer*s, 1992, 41(9): 1054-1068.

[9] Krishna C M. Performance Modeling for Computer Architectures. IEEE Computer Society Press. 1996.

[10]Jain A, Lin C. Back to the Future: Leveraging Belady’s Algorithm for Improved Cache Replacement[C]// *Annual IEEE/ACM International Symposium on Computer Architecture* (ISCA). 2016:78-89.

[11]Jin H, Chen D, Liu H, et al. Miss Penalty Aware Cache Replacement for Hybrid Memory Systems[J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.