计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220818
61(4): 824−839, 2024

基于缓存访问模式的 C-AMAT 测量方法及其在图计算中的应用

陈炳彰[1] 刘 伟[1,2] 于萧钰[1]
[1](武汉理工大学计算机与人工智能学院 武汉 430073)
[2](交通物联网技术湖北省重点实验室(武汉理工大学) 武汉 430073)
(chenbingzhang@whut.edu.cn)

C-AMAT Measurement Method Based on Cache Access Mode and Its Application in Graph Computing

Chen Bingzhang[1], Liu Wei[1,2], and Yu Xiaoyu[1]
[1] (*School of Computer Science and Artificial Intelligence*, *Wuhan University of Technology*, *Wuhan* 430073)
[2] (*Hubei Provincial Key Laboratory of Transportation Internet of Things Technology* (*Wuhan University of Technology*), *Wuhan* 430073)

Abstract Graph application is an important branch in the field of big data. Although graph analysis has more significant performance advantages than traditional relational databases in displaying the relationship between entities, the irregular memory access pattern caused by a large number of random accesses in graph processing destroys the time and space locality of memory access, thus causing great performance pressure on the off-chip memory system. Therefore, how to correctly measure the performance of graph application in memory system is crucial for efficient graph application architecture optimization. As an extension of average memory access time (AMAT), concurrent average memory access time (C-AMAT) takes into account the locality and concurrency of memory access, and can more accurately evaluate and analyze the performance of modern processors in the storage system. However, the C-AMAT model ignores the fact that the lower-level cache layer of the processor accesses serially, which will lead to the inaccuracy of the calculation. At the same time, it is difficult to obtain the parameters required for the calculation due to the "pure miss cycle" and other reasons, which also makes it difficult for C-AMAT to be applied in practice. In order to match the computing model of C-AMAT with the memory access mode in modern computers, we propose parallel C-AMAT (PC-AMAT) and serial C-AMAT (SC-AMAT) based on C-AMAT. PC-AMAT and SC-AMAT respectively extend and characterize the computing model of C-AMAT from the parallel and serial access modes of cache. On this basis, we design and implement a "pure miss cycle" extraction algorithm to avoid the huge hardware overhead caused by direct measurement. The experimental results show that the correlation between PC-AMAT and SC-AMAT, and IPC is stronger than that of C-AMAT in single-core and multi-core mode. Finally, PC-AMAT and SC-AMAT are used to measure and analyze the memory performance of graph application, based on which the optimization strategy of graph application access is proposed.

Key words graph application; graph analysis; AMAT; C-AMAT; pure miss cycle; cache

摘 要 图应用是大数据领域的一个重要分支,尽管图分析在显示表示实体之间关系的能力相比传统的关系数据库具有更显著的性能优势,但图处理中大量的随机访问所导致的不规则访存模式破坏了访存的

收稿日期: 2022-09-23; **修回日期**: 2023-05-19
基金项目: 国家自然科学基金项目(62272356); 计算机体系结构国家重点实验室(中国科学院计算技术研究所)开放课题(CARCHB202015)
This work was supported by the National Natural Science Foundation of China(62272356)and the Open Project of the State Key Laboratory of Computer Architecture (Institute of Computing Technology, Chinese Academy of Sciences)(CARCHB202015).
通信作者: 刘伟(wliu@whut.edu.cn)

时间和空间局部性，从而对片外内存系统造成了很大的性能压力. 因此如何正确度量图应用在内存系统中的性能，对于高效的图应用体系结构优化设计至关重要. 并发式平均存储访问时间（concurrent average memory access time, C-AMAT）模型作为平均存储访问时间（average memory access time, AMAT）的扩展，同时考虑了存储器访问的局部性和并发性，能够更准确地对现代处理器下图应用在存储系统中的性能进行评估分析. 但 C-AMAT 模型忽略了处理器下级 cache 层串行访问的事实，这会导致计算的不准确性，同时由于计算所需参数纯粹缺失周期等难以获取的原因，也使得 C-AMAT 难以进行实际应用. 为了使 C-AMAT 的计算模型与现代计算机中的存储器访问模式相匹配，基于 C-AMAT 提出了 PC-AMAT（parallel C-AMAT），SC-AMAT（serial C-AMAT），其中 PC-AMAT，SC-AMAT 分别从 cache 的并行和串行访问模式对 C-AMAT 的计算模型进行了细粒度的扩展和表征，并在此基础上设计并实现了纯粹缺失周期的提取算法，避免直接测量带来的巨大硬件开销. 实验结果表明，在单核和多核模式下，PC-AMAT 和 SC-AMAT 与 IPC 之间的相关性比 C-AMAT 更强，最终利用 PC-AMAT 和 SC-AMAT 度量和分析了图应用的存储器性能并据此提出图应用访存优化策略.

关键词 图应用；图分析；平均存储访问时间；并发式平均存储访问时间；纯粹缺失周期；缓存

中图法分类号 TP391

大数据时代下，人类社会中大规模与不规律数据信息快速增长，以图结构对这些数据进行存储分析越来越普遍. 基于图结构的信息存储方式被广泛应用于人际关系、社交网络分析、社会科学等各个领域，图数据处理变得越来越重要. 而目前图应用的主要性能瓶颈就在于其数据访问层面[1]，因此对于图应用在存储器中性能的评估与分析对于相应的硬件开发和算法设计都具有重要意义.

并发式平均存储访问时间（concurrent average memory access time, C-AMAT）模型通过同时考虑存储器访问的局部性和并发性[2]，可以更准确地表征图应用的存储器性能. 但是 C-AMAT 的计算模型认为数据访问的命中时间总是固定的，存在一定的局限性，同时，又因为其测量装置硬件开销大，使得其在实际应用中是不易实现的.

为了利用 C-AMAT 准确地评估检测图应用的存储器性能，我们基于缓存的访问模式将 C-AMAT 的计算模型扩展为 PC-AMAT（parallel C-AMAT）和 SC-AMAT（serial C-AMAT），并在此基础上设计并实现了 C-AMAT 中纯粹缺失周期（pure miss cycle, PMC）的提取算法及所需各项参数的测量. 最终利用 C-AMAT 在 gem5[3] 模拟器中对各类图应用在存储器中的性能进行了实验评估与分析，提出了相应的访存优化策略.

本文的主要贡献包括 3 个方面：

1）基于缓存的数据访问模式将 C-AMAT 的计算模型扩展为 PC-AMAT 和 SC-AMAT，使 C-AMAT 的计算模型与现代计算机中不同层次的存储器访问模式相匹配，从而为更准确地测量和计算应用程序运行过程中对数据的并发式平均存储访问时间提供了理论依据；

2）设计并实现了计算 C-AMAT 所需的重要中间参数纯粹缺失周期的提取算法，避免了直接测量所需要的巨大硬件开销；

3）利用相关系数验证了 PC-AMAT 和 SC-AMAT 与 IPC 之间的相关性比 C-AMAT 更强，进一步设计了图应用的存储器性能评估实验，通过应用各种规模的图数据集对图应用的访存性能进行了系统性地分析.

1 相关工作

图处理是大数据领域中一类重要的计算方式，在机器学习、路径规划、传染病学、神经网络等领域都有广泛应用.

现实世界的图数据往往呈现出小世界性（small-world）[4] 和无尺度性（scale-free）[5]，稀疏矩阵存储格式是典型的图数据存储格式，尽管有研究表明基于稀疏矩阵的压缩稀疏行（compressed sparse row, CSR）、对角线（diagonal, DIA）[6] 等格式存储效率很高，但仍然改变不了图数据访问的不规则性，存储器性能效率低下已经成为图应用发展中最大的性能瓶颈. 目前学术界关于图数据处理性能的研究有很多. Basak 等人[1] 通过分析乱序 CPU 下图数据的访存行为发现不同数据类型加载指令之间的依赖链（load-load dependency chain, LLDC）是实现高内存级并行（memory

level parallelism，MLP)的主要瓶颈；Balaji 等人[7]在末级缓存(last level cache, LLC)更换不同的数据替换策略时发现，即使是最先进的缓存数据替换策略，图应用在 LLC 上的 MPKI(misses per kilo instructions)依然没有明显下降；汤嘉武等人[8]通过实验分析出通用高层次综合(high level synthesis, HLS)系统缺乏对不规则图算法有效支撑的问题，提出了一种面向图计算的高效 HLS 方法，实现了高效的并行流水执行；Faldu 等人[9]验证了由于图分析的高度不规则访问模式，最先进的硬件缓存管理方案在利用其重用方面依然不尽人意，为此，他们引入了专门用于 LLC 上对图分析进行缓存管理的 GRASP，GRASP 专用缓存策略利用了热顶点固有的高重用，同时保留了捕获其他缓存块中图数据重用的灵活性；Cooksey 等人[10]则从高速缓存行中获取任何看似合理的地址作为数据加载，但是，只有观察到数据结构是一个指针列表时，这些指针才将被引用，否则此类方案将显著超载，从而导致缓存污染和性能下降.

尽管学术界现有的研究工作从多个维度对图应用进行性能评测都发现了其性能瓶颈在访存层面，但采用的系统性能评估指标更多地停留在以计算为中心的层面，最为广泛使用的便是每周期完成指令数(instruction per clock, IPC)，但 IPC 是被设计用来测试 CPU 性能的，并且由于 IPC 受指令集、CPU 微架构、存储器层次结构和编译器技术等多方面的影响，无法直接用来反映存储器系统的性能. 另一方面，传统的存储器性能指标，如访存缺失率(miss rate, MR)、平均缺失代价(average miss penalty, AMP)、平均存储器访问时间(average memory access time, AMAT)，对于采用流水线缓存[11]、非阻塞缓存[12]等现代设计的存储器系统来说是不合适的. 我们在 gem5 中对内核数量进行了倍增，以此提高 LLC 上的并发性，在对 GAP[13]基准测试套件中典型的 BFS(breadth first search)，PR(PageRank)，BC(betweenness centrality)，CC(connected components)这 4 个算法在 LLC 上的 AMAT 进行了测量，其中实验环境配置及工作负载见第 4 节. 图 1 显示了当并发性作为一个因素时，AMAT 作为衡量内存性能的指标是失败的，AMAT 指标表明：随着内核数量的增加，LLC 上的总体性能会下降. 这和系统性能提升的事实相违背，显然是错误的. 这也进一步表明 AMAT 已无法准确反映现代处理器的内存性能.

2013 年，Sun 等人[2]提出了新的存储器性能度量标准 C-AMAT，C-AMAT 通过给出严格规整的数学表

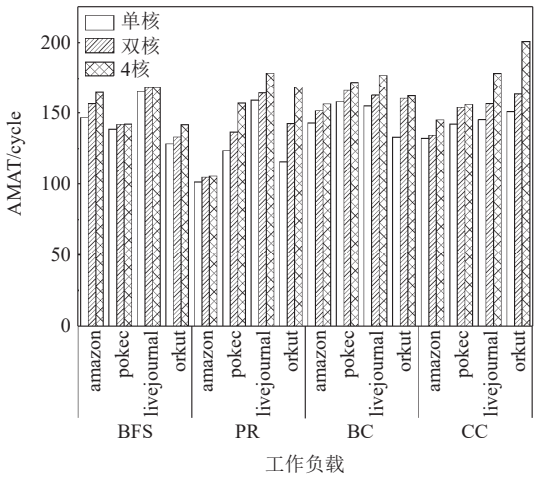

Fig. 1 AMAT of LLC at different number of cores

图 1 不同内核数量下 LLC 的 AMAT

达式和逻辑证明，将 AMAT 进行了扩展，可以更准确、全面地评估现代存储器系统.

在定量上，C-AMAT 被定义为总的存储访问周期除以总的存储访问次数[2]：

$$C\_AMAT = \frac{T_{\text{MemCycle}}}{C_{\text{MemAcc}}}, \tag{1}$$

其中 T_{MemCycle} 代表总的存储访问周期，C_{MemAcc} 则代表总的存储访问次数. 需要注意的是，T_{MemCycle} 是以重叠模式计算的，换句话说，当同一周期内存在多个内存访问时，T_{MemCycle} 仅增加 1 个周期. 此外，存储访问周期与 CPU 周期不同，至少有 1 次未完成内存访问的 CPU 周期才能算作内存访问周期.

根据相关系数的定义，C-AMAT 应该与 IPC 呈负相关[14]，即 IPC 增加时，C-AMAT 减少. 此外,当 IPC 有很大程度的增加时，C-AMAT 也应该有类似程度的变化. 然而，当我们在对 GAP 基准测试套件中图应用进行测量时发现，图应用的 IPC 并不总与 C-AMAT 呈负相关. 实验结果如图 2 所示，其中实验环境配置及工作负载见第 4 节. 为了更直观地表现 IPC 和 C-AMAT 之间的关系变化，对图算法在各个数据集中的 IPC 和 C-AMAT 的平均值进行了比较. 在图 2(a)中我们看到，随着内核数量的增加，IPC 也随之递增. 然而从图 2(b)~(d)中发现，随着内核数量的增加，在各级 cache 中的 C-AMAT 仅 BFS 算法在 L1 和 LLC 中呈现出递减的趋势，其他算法在各级 cache 中的变化则是无规律的，且变化幅度也是远小于 IPC 的.

之所以出现图 2 中的现象，是因为 C-AMAT 的计算模型总是认为数据访问过程中的命中延时是固定的，忽略了 cache 层中的数据访问模式，使用固定

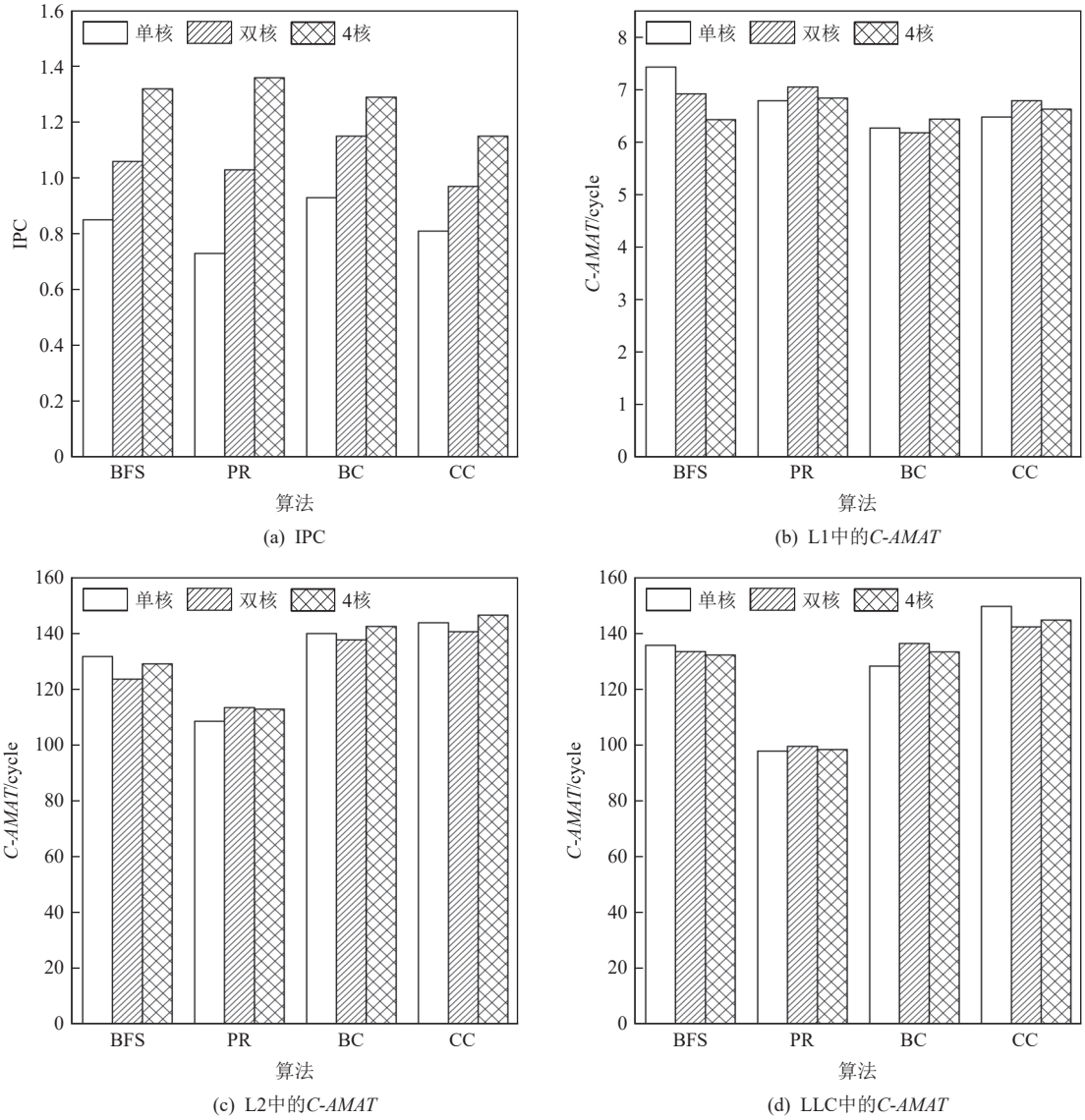

Fig. 2 IPC and *C-AMAT* of each cache level

图 2 IPC 和各级 cache 的 *C-AMAT*

的 cache 访问延时会导致测量出的总存储访问周期 T_{MemCycle} 与其理论值有所偏差.

事实上, 现代处理器多采用乱序执行来提升 CPU 的处理效率, 典型的如 Intel 在 x86 体系中的 Pentium Pro, 为了提高时钟频率并降低 cache 的缺失代价, 对于 cache 中的 Tag SRAM 和 Data SRAM 这 2 部分的访问模式, 现代处理器往往在 L1 cache 采用了并行访问的结构, 但其访问命中时间并不是简单的 Tag 与 Data 延时相加得到的. 而 L2 cache 以及下一级 cache 的数据访问则通常采用的是串行访问模式[15], 即对于 L2 cache 及下一级 cache 而言, 它们的命中延时在访问命中和访问缺失时不再是固定的, 因此, C-AMAT 的计算模型可能无法完全正确地表征存储器系统的整体性能. 其中的计算细节我们将在第 2 节重点讨论.

另一方面, 与 AMAT 相比, C-AMAT 的计算方法也更加复杂, 并且需要额外的检测逻辑和寄存器来测量 C-AMAT 所需的参数, 尽管文献 [14] 给出了每周期访问(access per cycle, APC)的测量逻辑, 但这种方法却有着较大的硬件开销和繁杂的计数逻辑, 不易实现. 因此, 如何准确获取一段完整应用程序中的纯粹缺失周期和纯粹缺失访问仍然是测量 C-AMAT 的重点和难点.

为了准确度量图应用的存储性能, 本文通过分析前人研究成果, 基于现代乱序处理器中不同层次 cache 的不同访问模式, 将 C-AMAT 的计算模型扩展为 PC-AMAT 和 SC-AMAT, 完善了 C-AMAT 的计算模型. 同时提出并实现了纯粹缺失周期和纯粹缺失访

问的提取算法，最终基于 PC-AMAT 和 SC-AMAT 对图应用的存储性能进行表征，这使得寻找到适用于图应用系统的整体最佳参数组合成为可能.

2 PC-AMAT 和 SC-AMAT 的计算模型

由于我们的工作直接基于 C-AMAT，因此在本节首先介绍了 C-AMAT 的计算模型，然后通过分析不同层次 cache 的数据访问模式，引入了基于 C-AMAT 扩展的 PC-AMAT 和 SC-AMAT 的计算模型.

2.1 C-AMAT 计算模型

式(1)给出了 C-AMAT 的原始定量公式，但并没有明显体现出 C-AMAT 的局部性和并发性，为了体现这 2 种性质，给出了 C-AMAT 的详细计算公式[2]：

$$C_AMAT = \frac{H}{C_h} + pMR \times \frac{pAMP}{C_m}, \tag{2}$$

其中 H 代表命中延时，C_h 和 C_m 分别代表命中存储请求的并发度和缺失存储请求的并发度，C_h 和 C_m 是 C-AMAT 引入的 2 个新参数. 此外，这个计算公式首次引入了“纯粹缺失”的概念，即只有在缺失访问中至少包括 1 个纯粹缺失周期，在这个周期中，整个存储系统都没有命中发生，这样的缺失才称为纯粹缺失. pMR 为纯粹缺失率，即纯粹缺失访问的次数与全部存储访问次数之比. $pAMP$ 即纯粹平均缺失代价，代表平均每个纯粹缺失访问中的纯粹缺失周期的数量[2].

然而，在 C-AMAT 中，无论当前层级存储器中的数据是否命中，它的命中时延总被认为是固定的. 事实上，现代处理器中 cache 层的数据命中时间与其访问模式息息相关，因此，我们基于 cache 访问模式对 C-AMAT 的计算模型进行了细粒度的扩展.

2.2 PC-AMAT 计算模型

现代处理器中的 cache 由 Tag 和 Data 这 2 部分组成，但在实际的实现当中，cache line 中的 Tag 和 Data 部分其实是分开放置的，称为 Tag SRAM 和 Data SRAM，如果对这 2 部分的内容同时进行访问，则称为并行访问[16]，其结构如图 3 所示；反之，如果先访问 Tag SRAM 部分，根据 Tag 的比较结果再去访问 Data SRAM 部分，这种方式则称为串行访问[16]，其结构如图 4 所示.

当对图 3 这种结构的 cache 进行访问时，在对某个 Tag 部分的地址访问的同时会对该地址对应 Data 部分的数据也进行访问，并将它们送到多路选择器中选择出指定的数据块，最终经过数据对齐便可完成数据访问. 由于现代超标量处理器使用的是流水线访问，数据块选择时间和数据对齐部分的时间是可以忽略不计的，因此，在 cache 的并行访问结构中完成一次数据访问的时间其实是由 Tag 和 Data 中访问延时较长的部分所决定的，此时如将 Tag 与 Data 访问延时之和作为 cache 命中时间是错误的，因为这将会严重高估C-AMAT 的值.因此,并行结构下基于C-AMAT 的 PC-AMAT 可以表示为式(3).

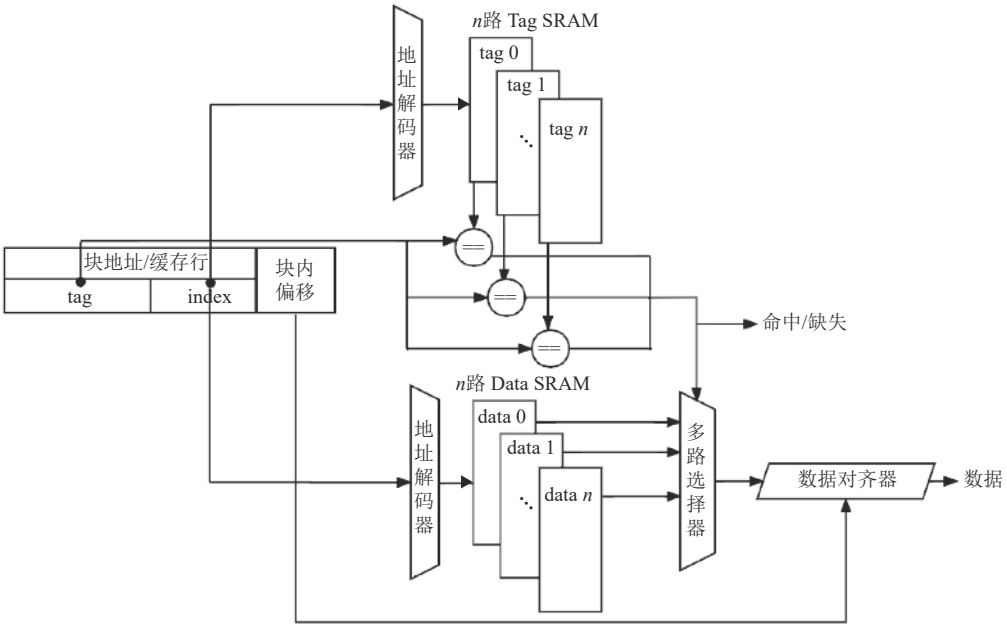

Fig. 3 Parallel access mode in cache

图 3 cache 中的并行访问模式

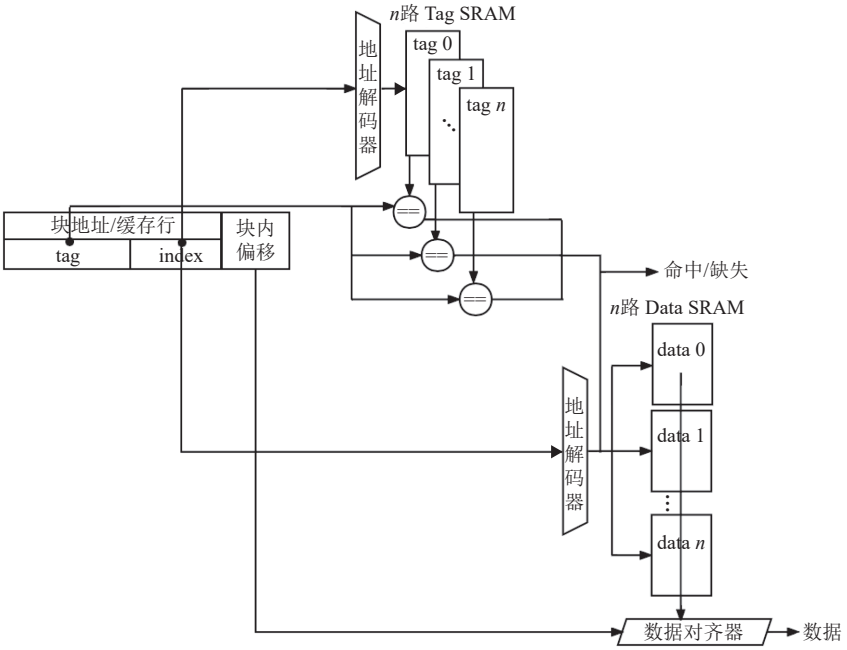

Fig. 4 Sequential access mode in cache

图 4 cache 中的串行访问模式

$$PC\_AMAT = \frac{PH}{C_{ph}} + pMR \times \frac{pAMP}{C_m}. \tag{3}$$

在 PC-AMAT 中，*PH* 代表了并发 cache 命中时间，即在并行访问结构中，cache 的命中延时 *PH* 由 Tag 延时和 Data 延时的较大者决定，因此 *PH* 可以表示为：

$$PH = \begin{cases} tag\_latency, & tag\_latency \geqslant data\_latency, \\ data\_latency, & tag\_latency < data\_latency, \end{cases} \tag{4}$$

其中 *tag_latency* 和 *data_latency* 分别表示 Tag 部分访问和 Data 部分访问所需要的命中时间，一般情况下，现代处理器中 L1 cache 在这 2 部分的延迟都在 2~5 个周期之间，同一架构处理器中的 *tag_latency* 和 *data_latency* 差距并不会太大[17]，但文献 [18] 中将命中时延统一成一个固定的参数显然是与事实不符的，这会造成 C_{ph} 和 C_m 的计算误差，最终导致计算出来的 C-AMAT 值与实际值存在较大误差. C_{ph} 和 C_m 的计算方式为：

$$C_{ph} = \frac{PH \times access}{overlap\_hitTime}, \tag{5}$$

$$C_m = \frac{pMissTime}{pMissTime\_overlap}, \tag{6}$$

其中 *access* 表示当前存储器中总的数据访问次数，*overlap_hitTime* 表示当前 cache 访问过程中的重叠命中时间，*pMissTime* 表示所有数据访问总的纯粹缺失周期，而 *pMissTime_overlap* 则代表了所有数据访问过程中总的重叠纯粹缺失周期. 我们将在第 3 节详细解读上述变量的取值.

2.3 SC-AMAT 计算模型

图 4 展示了 cache 的串行访问结构，在对数据进行访问时，首先需要对 Tag SRAM 进行访问，进行 Tag 比对之后，若 Tag 对应的 Data 部分中的数据在当前 cache 层中存在，则对其进行选择访问，此时就不再需要进行数据对齐即可对指定的数据块进行访问，这样的 1 次数据访问命中时间就是 Tag 与 Data 的访问延时之和；但若对 Tag 进行访问后在当前 cache 层中未检测到对应的数据部分，则进入下一级存储器进行访问，此时当前 cache 层中数据访问的命中时间只需将 Tag 部分延时包含在内.

因此，串行访问结构是不需要多路选择器进行数据选择的，而只需要访问数据部分指定的那个 SRAM，其他的 SRAM 由于都不需要被访问，可以将它们的使能信号置为无效，这样就可以节省很多功耗，串行访问结构也多被应用于现代处理器的 L2 cache 和 LLC. 即串行结构下的 cache 访问，其命中时间是由数据是否命中所决定的. 因此，串行结构下基于 C-AMAT 的 SC-AMAT 可以表示为：

$$SC\_AMAT = \frac{SH}{C_{sh}} + pMR \times \frac{pAMP}{C_m}. \tag{7}$$

尽管在 SC-AMAT 中，*SH* 也代表着 cache 的命中时间，但不同于式(2)中的 *PH*，*SH* 的大小是由数据是否命中决定的，因此，*SH* 可表示为：

$$SH = \begin{cases} tag\_latency + data\_latency, & \text{数据访问命中}, \\ tag\_latency, & \text{数据访问缺失}, \end{cases} \tag{8}$$

当数据访问命中时，命中时间 *SH* 是 *tag_latency* 和 *data_latency* 相加，而当数据访问缺失时，其命中时间 *SH* 则只由 *tag_latency* 决定. 现代处理器中 L2 cache 在这 2 部分的延迟都在 6~12 个周期，LLC 在这 2 部分的延迟一般在 32~64 个周期[17]，当然，精确的访问延时需要根据具体的 CPU 架构来确定. 但显而易见，此时命中延时 *SH* 不是固定值，使用式(2)中的 *H* 将会对并发式平均访问时间的计算结果造成不可忽略的误差. 相应地，串行访问结构下数据访问的命中并发度 C_{sh} 的表达式为：

$$C_{sh}=\frac{(tag\_latency+data\_latency)\times hit+tag\_latency\times miss}{overlap\_hitTime}, \tag{9}$$

其中 *hit* 代表当前存储器中数据访问的命中次数，*miss* 表示当前存储器中数据访问的缺失次数，与 C_{ph} 一样，C_{sh} 的分母 *overlap_hitTime* 依然是当前 cache 访问过程中的重叠命中时间.

第 4 节中我们将会根据 IPC 与 C-AMAT, PC-AMAT, SC-AMAT 这 3 个指标的相关系数来度量这 3 个指标的精确度，以验证扩展 C-AMAT 为 PC-AMAT 与 SC-AMAT 的必要性与合理性.

3 纯粹缺失周期提取算法

本节详细介绍如何提取 PC-AMAT 和 SC-AMAT 测量过程中所需的重要中间参数——纯粹缺失周期，并据此计算出 *pMR*, *pAMP*, C_m 等依赖 PMC 的计算参数，进而计算出 *PC-AMAT* 和 *SC-AMAT*.

根据式(2)，我们知道想要计算 C-AMAT 首先需要测量 C_h 和 C_m. 文献 [18] 提供了 C_h 和 C_m 的计算方法，如图 5 所示. 通过在原有的硬件结构上增加了 2 个探测器，分别是命中并发度探测器和缺失并发度探测器，这 2 个探测器都由检测逻辑和寄存器组成，命中并发度探测器中的检测逻辑用来监控是否有缓存 Tag 查询活动. 通过使用寄存器来统计总的命中周期和每个命中阶段的同时命中情况，并计算平均命中并发度. 因此，每个命中阶段至少需要 2 个寄存器，分别用来记录开始周期和结束周期. 缺失并发度探测器中的检测逻辑监控是否有新的请求到达 MSHR (missing status holding register)[18]，并通过寄存器来探测纯粹缺失周期的数量和每个纯粹缺失阶段的并发度. 缺失并发度探测器所需的寄存器数量计算和命中并发度探测器数量的计算方法是类似的，不同之处在于选择的寄存器的数量需要考虑同一周期内共存的最大缺失访问数，即 MSHR 表项的数量乘以 MSHR 表项中目标的数量.

基于上述分析，我们发现仅是测量 C-AMAT 的 C_h 和 C_m 便会导致很大的硬件开销，因此想要通过这种测量装置直接测量出应用程序的 C-AMAT 是比较困难的. 据此，我们在测量图应用的 C-AMAT 时并没有直接使用文献 [18] 中的这种测量装置. 同样如图 5 所示，我们并没有在程序运行的过程中去监测访存的命中并发度与缺失并发度，对于每个命中与缺失阶段，都只需要 2 个寄存器分别用来记录开始周期与结束周期，在一次访存结束后，便可立即释放当前寄存器，

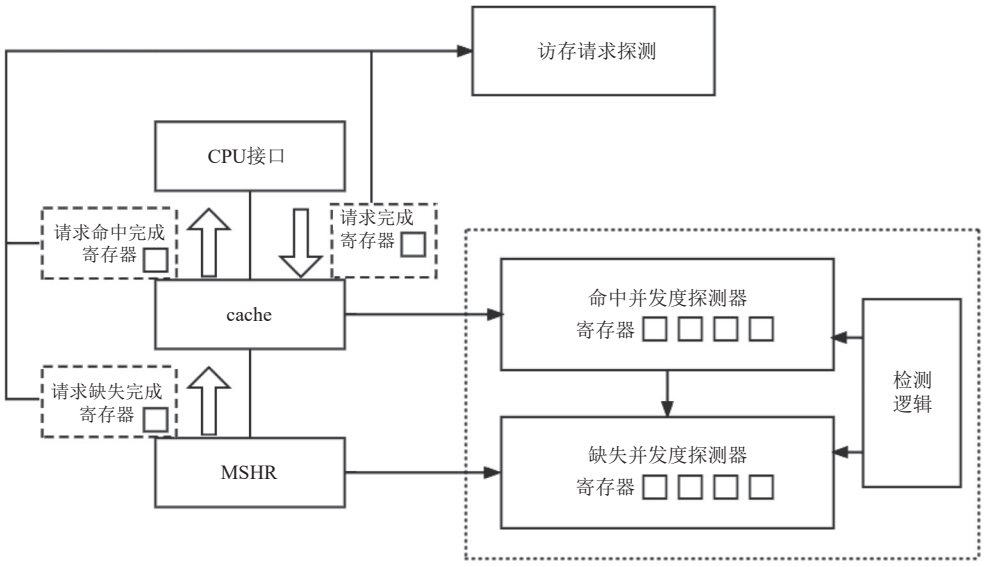

Fig. 5 C-AMAT measuring device

图 5 C-AMAT 测量装置

而无需一直保留用来计算访问并发度. 最后通过设计相应的纯粹缺失周期提取算法, 计算出 PC-AMAT 和 SC-AMAT 计算所需要的相关参数. 这种方法大大降低了 C-AMAT 测量所需的硬件开销, 同时避免了程序运行阶段探测并发度时繁杂的访存检测逻辑.

3.1 PC-AMAT 与 SC-AMAT 实例

为了更好地解释和理解纯粹缺失周期等相关概念, 我们基于 PC-AMAT 举了一个简单的例子, 如图 6 所示, 图 6 中共存在 5 个不同的存储访问. 由于 PC-AMAT 是针对并行访问结构的 cache 而言的, 我们假设当前 cache 中 Tag SRAM 部分的命中时间为 2 个周期, Data SRAM 部分的命中时间为 3 个周期, 因此, 每次数据访问都必须经过 3 个周期的命中阶段. 如果访问没有在当前 cache 中命中, 即访问缺失时, 则会产生 1 个缺失代价, 缺失代价的大小取决于最终该缺失的数据在哪一级存储层次中被找到. 图 6 中, 访问请求 1, 2, 5 是命中访问, 访问 3 和 4 是缺失访问. 访问 3 包含了 4 个周期的缺失代价, 其中有 2 个周期与访问 5 的命中周期出现了重叠, 因此, 从并发存储的角度来看, 访问 3 只存在 2 个纯粹缺失周期. 访问 4 有 1 个缺失周期, 但该周期也与访问 5 的命中周期重叠, 因此不是纯粹缺失周期. 因此, 访问 3 是纯粹缺失访问, 访问 4 不是, 所以这 5 个访问的缺失率为 0.4, 而纯粹缺失率仅为 0.2, 纯粹平均访问缺失代价 *pAMP*=1. 同时, 对于式(5)和式(9)中提到的命中重叠时间 *overlap_hitTime*, 我们将其定义为在所有访问请求中, 只要当前周期存在命中周期, 那该周期则算作 1 次重叠命中周期. 类似地, 纯粹缺失重叠时间 *pMissTime_overlap* 为在所有请求访问中, 当前周期都是纯粹缺失周期时视作 1 次纯粹缺失重叠周期.

因此, 从图 6 中我们可以观察到这 5 个访问总的命中重叠时间为 7, 总的纯粹缺失重叠时间为 1, 根据式(5)和式(6)计算可得 C_{ph}=15/7 和 C_m=2, 最后由式(3)得到 PC-AMAT 的每次访问周期个数为 1.8. 事实上, 图 6 中 5 个访问请求总共经历了 8 个时钟周期, 并且这 8 个周期都属于访问周期, 因此我们也可以直接将总重叠访问周期除以访问次数得到 PC-AMAT 的每次访问周期个数为 9/5=1.8.

而对于 SC-AMAT, 其与 PC-AMAT 最大的区别就在于当前存储器中的命中访问与缺失访问的命中时间是不一致的, 为了更好地解释 SC-AMAT, 我们同样举了一个简单的例子, 如图 7 所示, 图 7 中也存在 5 个不同的存储访问. 我们假设当前 cache 中 Tag SRAM 和 Data SRAM 的命中时间都为 3 个周期, 因此, 每次数据访问若是在当前 cache 中命中, 则其命中时间为 6 个周期. 若是访问缺失, 则只会经历 3 个周期的 Tag 访问命中时间, 接下来该缺失访问同样会根据数据最终查找到位置, 产生相应的缺失代价. 图 7 中, 由于访问请求 1, 2, 5 的命中时间都为 6 个周期, 因此都属于命中访问; 而访问 3, 4 则属于缺失访问, 它们在当前 cache 上的命中时间都仅为 3 个周期, 访问 3 包含了 8 个周期的缺失代价, 其中有 5 个周期

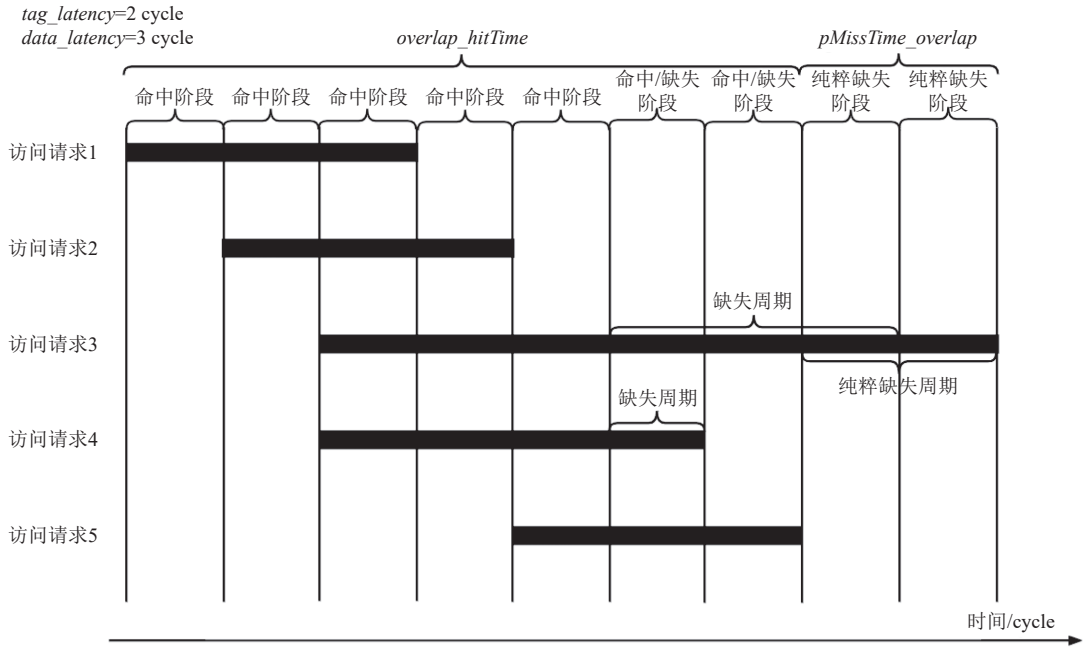

Fig. 6 An example of PC-AMAT principle

图 6 PC-AMAT 原理示例

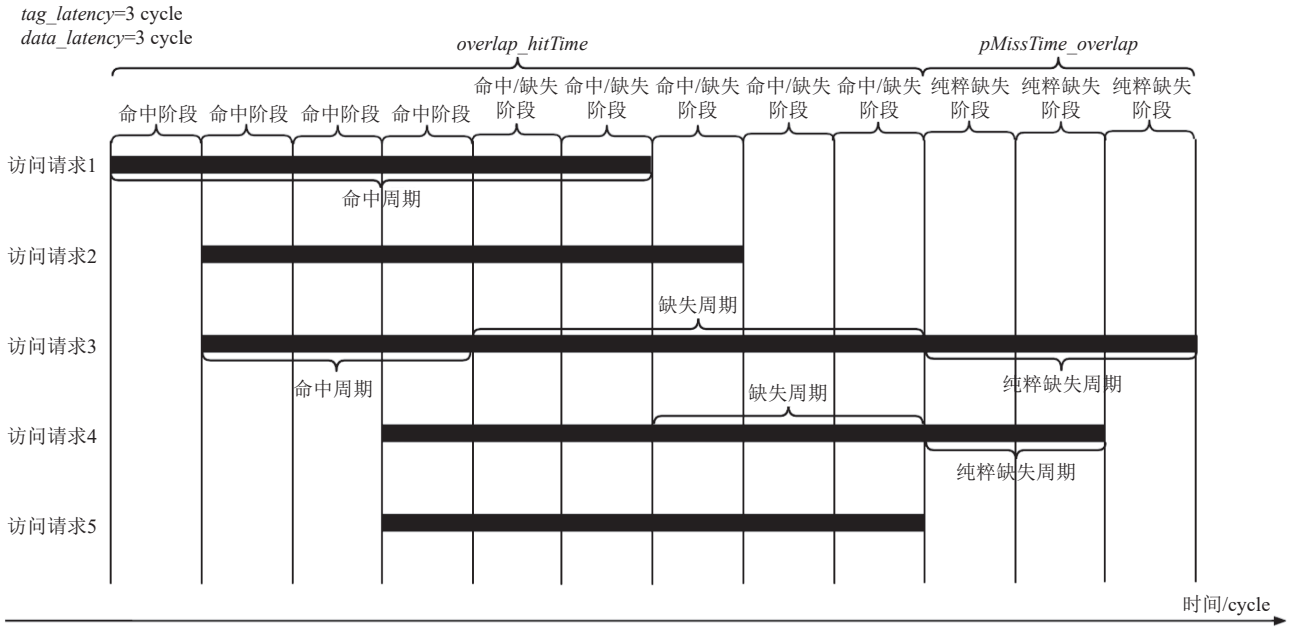

Fig. 7 An example of SC-AMAT principle

图 7 SC-AMAT 原理示例

与访问 5 的命中周期重叠，因此访问 3 存在 3 个纯粹缺失周期. 与访问 3 类似，访问 4 的缺失代价为 5 个周期，其中有 2 个周期为纯粹缺失周期. 通过以上分析，我们可以计算出图 7 示例中 5 个访问的纯粹缺失率为 0.4，纯粹平均访问缺失代价 $pAMP$=2.5，同时将命中重叠时间和纯粹缺失重叠时间分别等于 9 和 3 时代入式(9)和式(6)，可以计算出 C_{sh}=8/3 和 C_m=5/3，最后由式(7)计算得到 SC-AMAT 每次访问 2.4 个周期. 同样，由于这 5 个访问所经历的总重叠时间为 12 个周期，我们也可以得到 SC-AMAT 每次访问 2.4 个周期(12/5).

3.2 纯粹缺失周期提取算法

在实际应用程序中，并不是所有周期都属于访问周期，数据访问过程中存在着停滞周期，因此 3.1 节中的计算示例的计算方法并不适用于实际应用程序中 C-AMAT 的测量. 准确测量 PC-AMAT 和 SC-AMAT 的值的难点就在于如何准确地判断每个缺失周期是命中、失效重叠，还是纯失效. 据此，我们设计了纯粹缺失周期 PMC 提取方法，具体过程如图 8 所示.

为了获取每个缺失周期的属性值，我们首先将缺失访问进行了合并. 我们将所有缺失访问的缺失代价按照它们的开始周期进行递增排序，然后通过一个结构体 *missCycle* 记录每一个缺失周期的绝对时间 *start*、该缺失周期被缺失访问占有的次数 *count* 以及该缺失周期所在的缺失访问请求 *parent*. 我们通过

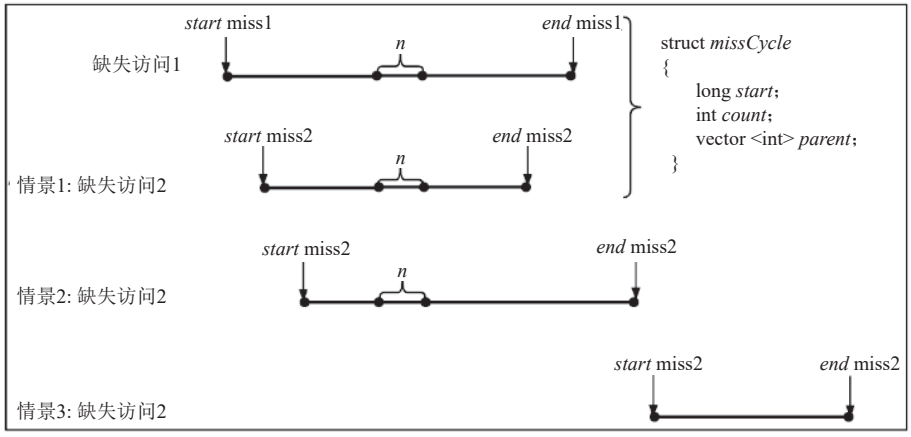

Fig. 8 An example of missing cost scenario in missing access

图 8 缺失访问中的缺失代价情景示例

分类和迭代便可获取到每个缺失周期的属性值，包括该缺失周期总共被占有的缺失访问的次数以及存在于哪些缺失访问中. 算法描述如算法 1 所示.

算法 1. 缺失周期合并算法.

输入：缺失访问数组 *missAccess*，当前缺失周期 *curr_cycle*.

```
①  for each curr_cycle in missAccess[i] do
②    curr_cycle 初始化;
③  end for
④  add first(missAccess[0]) into tempMiss array;
   /*将第 1 个缺失访问的缺失代价暂存中间
     缺失数组 tempMiss 中*/
⑤  for each missAccess>1 in missAccess array do
⑥    if missAccess[i].start≥tempMiss[i−1].start
       && missAccess[i].start<tempMiss[i−1].end
       then/*图 8 情景 1*/
⑦      if missAccess[i].end≤tempMiss[i−1].end
         then
⑧        for each curr_cycle in missAccess[i] do
⑨          location=find(curr_cycle);
⑩          curr_cycle.count++;
⑪          curr_cycle.parent.push_back(i);
             当在合并缺失访问时出现情景
             1 下的缺失访问时，只需对每个
             缺失周期的属性值进行更新*/
⑫        end for
⑬      else
⑭        for each cycle in tempMiss[i−1] do
⑮          ⑨~⑪; /*图 8 情景 2，周期更新*/
⑯        end for
⑰        for each curr_cycle from tempMiss
           [i−1].end to missAccess[i].end do
⑱          将这部分 curr_cycle 初始化;
⑲          缺失访问 2 的尾部插入到缺失访
             问 1 的尾部;
⑳        end for
㉑      end if
㉒    else
㉓      for each curr_cycle in missAccess[i] do
㉔        初始化 curr_cycle; /*图 8 情景 3*/
㉕          tempMiss.push_back(missAccess[i]);
            /*此时缺失访问 1 无法再进行合
            并操作，缺失访问 2 作为下一次
            合并操作的缺失访问 1*/
㉖        end for
㉗      end if
㉘  end for
```

对于图 8 中的缺失访问 1 和情景 1 下的缺失访问 2，第 n 个缺失周期的绝对时间即为 n，该缺失周期当前被缺失访问占有的次数为 2，其所在的缺失访问请求为缺失访问 1，2. 在经过上述处理时，缺失访问 2 相对于缺失访问 1 可能会出现图 8 所示的 3 种情况. 其中行④~㉑是当缺失访问 2 出现如图 8 中的情景 1 和情景 2 时，需要将每个 *curr_cycle* 属性进行更新，根据缺失访问 2 的结束时间是否大于缺失访问 1 来决定是否需要将其进行缺失访问合并操作，而行㉓~㉖则是当缺失访问 2 的开始时间大于缺失访问 1 时，此时这 2 次的缺失访问无法进行合并，则对缺失访问 2 的每个缺失周期进行初始化后开始新一轮的合并迭代操作. 通过对缺失访问的合并操作，我们便可以清晰地获取到每个缺失周期的信息，包括该缺失周期所处的缺失访问位置及出现次数. 同样，为了更快速地判断每个缺失周期是否是纯粹缺失周期，我们对命中阶段的周期也进行了合并操作，需要注意的是，此处命中阶段的含义不仅代表命中访问所经历的时间，而且缺失访问的命中时间也应该算作命中阶段，具体算法描述如算法 2 所示.

算法 2. 命中周期合并算法.

输入：命中阶段数组 *hitPhase*，当前命中阶段 *curr*，下一命中阶段 *next*.

```
①  while next 存在 do
②    if next.start≥curr.start && next.start ≤
       curr.end then
③      curr.end=next.end;
④      在 hitPhase 中删除 next;
⑤      next=curr, ++next;
⑥    else
⑦      curr=next, ++next;
⑧    end if
⑨  end while
```

我们在对命中阶段按开始周期以递增方式进行排序后，根据下一命中阶段的开始周期是否大于当前命中阶段的结束周期来决定是否将当前 2 个命中阶段进行合并. 通过这样的预处理，我们可以最小化每个缺失周期与命中周期的比较次数，从而快速判断该周期是否是纯粹缺失周期. 具体算法如算法 3 所示.

算法 3. 纯粹缺失周期的提取算法.

输入：经过算法 1 合并后的缺失周期数组 *missCycle*，

当前缺失周期 *curr_cycle*，经过算法 2 合并后的命中阶段数组 *hitPhase*；

输出：是否属于纯粹缺失周期.

① for each *curr_cycle* in *missCycle*[*i*] do
② if *curr_cycle* in *hitPhase* then
③ return false;
④ else
⑤ *pureMissCycle*+=*curr_cycle.count*;
⑥ if 当前周期未被记录总的重叠纯粹缺失周期 then
⑦ *pMissTime_overlap*++;
⑧ end if
⑨ return true;
⑩ end if
⑪ end for

该 PMC 的判断算法通过将经过缺失访问合并后的每个缺失访问的每个缺失周期与每个命中周期进行 *start* 属性比较，若当前缺失周期的 *start* 属性与某个命中周期的 *start* 相同，则该缺失周期不是纯粹缺失周期，反之则是.

4 实验评估

由于本文研究 C-AMAT 的目的是为了度量图应用的存储器性能，发掘图应用在存储器中的性能优化方向，因此，本节将通过 PC-AMAT 和 SC-AMAT 对大规模图数据集进行存储器性能测试和评估，并分析其内存性能表现.

4.1 实验环境

我们在 gem5 模拟器中采用了乱序超标量 CPU 模型，该模型支持单核模式下的分支预测和多线程执行. 为了更准确地测量缓存/内存架构设计下图应用的 PC-AMAT 和 SC-AMAT，我们参考了 WikiChip[17] 和丹麦技术大学[19] 最新整理的 IceLake CPU 架构及其参数，在 gem5 中增加了 3 级共享缓存的架构，并重新进行了缓存参数配置，以使缓存架构及其对应的 Tag 延时与 Data 延时更加接近真实的硬件环境. 对于每个图应用的测量点，我们都根据 4×10^8 个模拟指令收集了图应用的访问数据，PC-AMAT 和 SC-AMAT 的计算及其参数都来源于对这些访问数据的处理. 具体配置情况如表 1 所示.

Table 1 Simulator Configuration

表 1 模拟器配置

模块设置	参数配置
Processor	O3 CPU core, 4 GHz, 8-issue width
Function units	6 IntALU: 1 cycle;1 IntMul:3-cycle; 2 FPAdd: 2-cycle; 1 FPCmp: 2-cycle; 1 FPMul: 4-cycle; 1 FPDiv: 12-cycle; 1 FPCvt: 2-cycle
Private L1 caches	32 KB inst/ 32 KB data,8-way, 64 B line, 4-cycle tag latency inst/ 4-cycle data, 4-cycle data latency inst/ 4-cycle data, ICache 4 MSHR entry, DCache 4 MSHR entry
Private L2 cache	256 KB, 8-way, 64 B line, 7-cycle tag latency, 7-cycle data latency, 20 MSHR entry
Shared L3 cache	8 MB, 16-way, 64 B line, 37-cycle tag latency, 38-cycle data latency, 20 MSHR entry
DRAM latency/width	240-cycle access latency/64 bit

4.2 工作负载

本文使用 GAP 基准套件[13]，它是典型的图处理基准套件，能够有效标准化图应用的指标评估. GAP 之中的图算法均使用 C++编写，并使用了优化多线程技术，我们之所以选择 GAP 是因为它可以排除任何与框架相关的性能开销，从而真正发现图应用在存储器中的性能瓶颈. 我们选取了其中最具代表性的 BFS[20]，PR[21]，BC[22]，CC[23]4 种算法作为工作负载进行测试，具体算法介绍如表 2 所示.

Table 2 Graph Algorithm List

表 2 图算法列表

算法	描述
BFS	广度优先搜索算法，逐层遍历图
PR	图的链接分析，根据相邻顶点的秩对每个顶点进行排序
BC	测量顶点的中间性，即通过它的其他任意 2 个顶点之间的最短路径数
CC	将图分解为一组连通子图

表 2 中的图算法分别代表了社交网络中心、工程应用领域和科学中的诸多应用，由于不同的图算法计算方式是不同的，主要以遍历为中心和以计算为中心，并且它们对于图的属性考虑也各有侧重，因此为了更全面地评估各类图应用的存储器性能，我们采用的图数据集如表 3 所示.

Table 3 Experimental Datasets

表 3 实验数据集

数据集	顶点	边	大小/MB	描述
amazon[24]	0.4×10^6	3.39×10^6	32	亚马逊产品网络
pokec[24]	1.63×10^6	30.60×10^6	259	Pokec 在线社交网络
livejournal[24]	4.85×10^6	68.48×10^6	597	LiveJournal 在线社交网络
orkut[24]	3.0×10^6	117×10^6	941	社交网络

4.3 负载性能测试

由于存储器系统的性能对整体性能有很大的影

响，因此应该选择一个适当的指标来反映它们之间的关系. 我们使用相关系数(correlation coefficient)[15]来描述内存指标 PC-AMAT, SC-AMAT, C-AMAT 与 IPC 之间的变化相似度，并作为它们的评价精度. 相关系数的取值范围为−1~1. 相关系数的绝对值越高，代表内存指标和 IPC 这 2 个变量之间的关系就越密切[25]. 相关系数的数学定义为：

$$r_{XY}=\frac{\sum XY-\frac{\left(\sum X\right)\left(\sum Y\right)}{n}}{\sqrt{\left[\left(\sum X^2-\frac{\left(\sum X\right)^2}{n}\right)\left(\sum Y^2-\frac{\left(\sum Y\right)^2}{n}\right)\right]}}, \tag{10}$$

相关系数是通过积差方法进行计算的，通过以 2 个变量与各自平均值的离差为基础，将 2 个变量的离差相乘来反映 2 个变量之间的相关程度. 其中数组 X 和 Y 是 2 个变量的采样点.

我们首先分别计算了当内核数量为 1, 2, 4 时对应各级 cache 的相关系数. 对于各级 cache，这 2 种存储器性能指标的相关系数，如图 9 所示. 图 9(a)显示了 L1 cache 上 C-AMAT 与 PC-AMAT 的相关系数的对比，我们发现随着内核数量的增加，C-AMAT 的相关系数并未出现明显变化，PC-AMAT 在相关系数上则始终高于 C-AMAT，并且内核数量并未明显影响其相关系数变化，之所以出现这样的原因，是因为 L1 cache 中的数据访问命中时间是固定的，内核数量的增加对于总的数据访问周期的测量误差并没有表现出比较明显的影响，但由于 PC-AMAT 在数据访问时的命中时间参数选取更加准确，因此其相关系数相比 C-AMAT 也更高. 图 9(b)(c)分别代表了 L2 cache 和 LLC 上 C-AMAT 和 SC-AMAT 的相关系数对比，随着内核数量的增加，C-AMAT 的相关系数都有小幅度提升，我们分析这是由于内核数量的增加，导致数据访问并发度也随之增加，尽管 L2 cache 和 LLC 上的数据访问命中时间存在不一致性，但并发度的提升导致数据访问重叠周期也更多，掩盖了一部分数据访问周期计算过程中的误差，但其相关系数始终与 SC-AMAT 有一段距离.

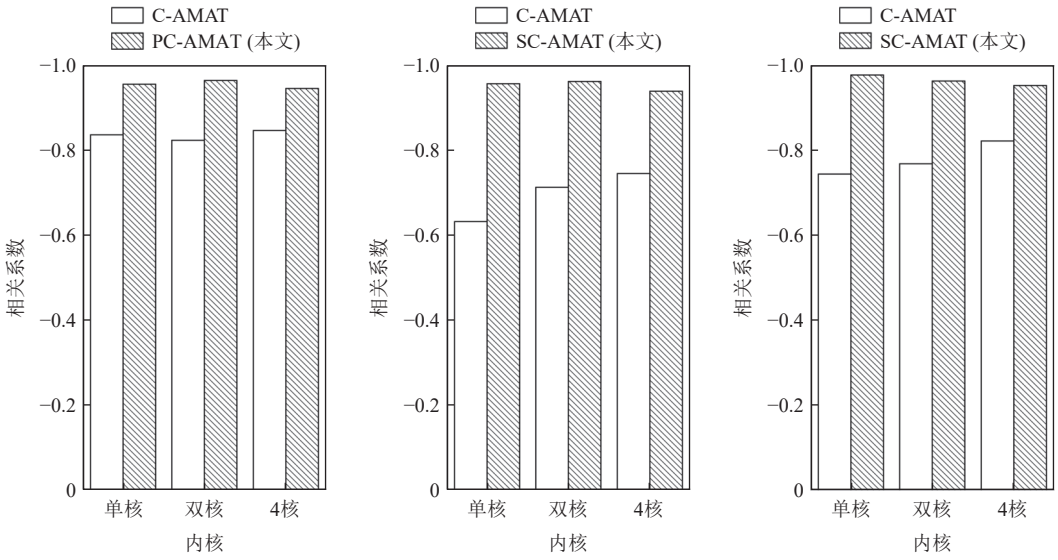

Fig. 9 The correlations coefficients of performance indicators in each cache level with different number of cores

图 9 各级 cache 在不同内核数量中性能指标的相关系数

以内核数量为 4 时各个图应用中 IPC 与 C-AMAT, PC-AMAT 和 SC-AMAT 的相关系数为例，进一步验证了将 C-AMAT 的计算模型扩展为 PC-AMAT 和 SC-AMAT 的有效性与合理性，实验结果如图 10 所示. 从图 10(a)可以看出，在几乎所有的图应用中，L1 cache 中 PC-AMAT 的相关系数都高于 C-AMAT，相比 C-AMAT, PC-AMAT 的相关系数提升了 11.7%. 从图 10(b)(c)中我们观察到，在 L2 cache 与 LLC 中，SC-AMAT 在 L2 cache 和 LLC 上的相关系数分别提升了 26.1% 和 15.8%.

由于 PC-AMAT 和 SC-AMAT 只是在计算与测量过程中对 C-AMAT 进行的扩展，因此在接下来的结果评估中将不再对 PC-AMAT 和 SC-AMAT 进行区分. 我们首先将图应用运行在单核模式下，我们计算了各级 cache 的 C-AMAT 和相关的内存性能参数，整体性能表现如图 11 所示.

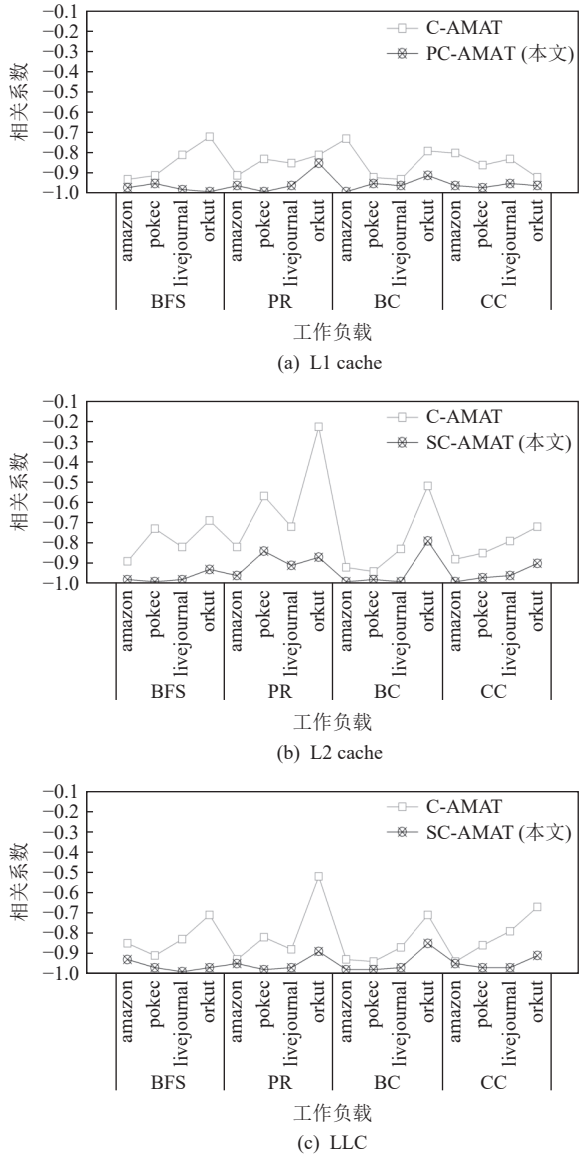

Fig. 10 Comparison of the correlation coefficients of cache performance indicators at each level

图 10 各级 cache 性能指标的相关系数对比

由图 11 可以看出，PR 算法的整体内存性能表现是优于其他算法的，同时对于 L1 data cache 来说，各类图应用程序性能表现类似，但值得注意的是，在绝大部分算法中，L2 cache 的 C-AMAT 都明显高于 LLC，这个现象直接暴露出了 L2 cache 的存在对于图数据访问而言是负收益的，而导致这种现象的本质还是图计算运行过程中不规则的细粒度访存导致 CPU 的 L2 cache 和 LLC 的命中率极低[26].

这些图应用负载的命中并发度和缺失并发度如图 12 和图 13 所示. 命中并发度与缺失并发度是影响 C-AMAT 的关键因素，可以看出，单核模式下，所有图应用在 L2 cache 的命中并发度表现都是最低的且其值都徘徊在 1 附近，这进一步表明了传统 L2 cache 的

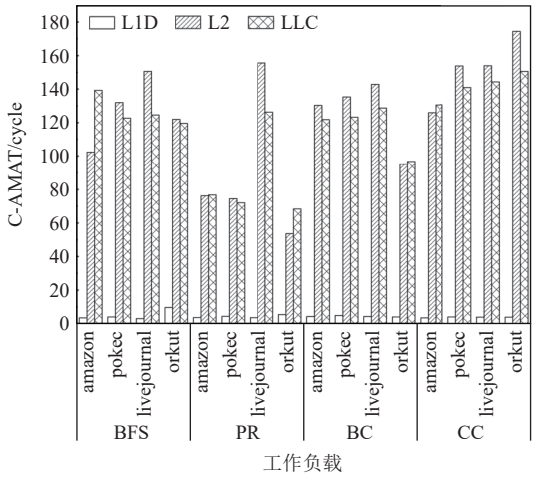

Fig. 11 C-AMAT in each level of cache

图 11 各级 cache 中的 C-AMAT

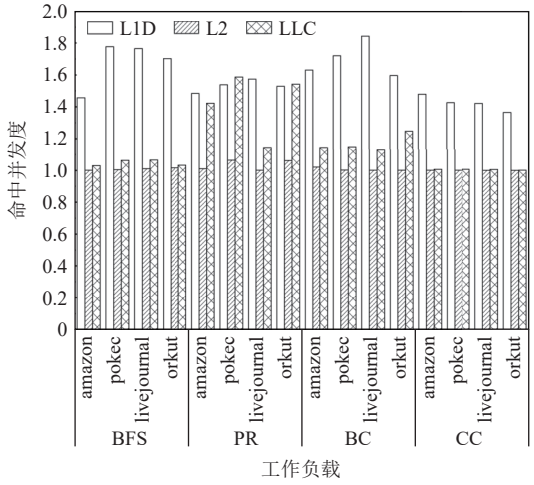

Fig. 12 Hit concurrency in each level of cache

图 12 各级 cache 中的命中并发度

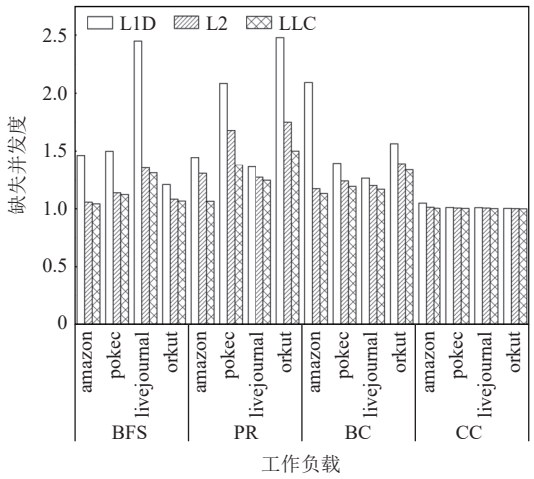

Fig. 13 Miss concurrency in each level of cache

图 13 各级 cache 中的缺失并发度

存在对于图应用并没有多大意义，甚至影响到了整体的图应用访存时间，现代处理器中的 3 级缓存存

储器层次结构的设计对于图应用而言负面影响较大. 为了提高图应用的命中并发度, 我们可以采用多端口缓存、多 bank 缓存来提高命中时间的重叠; 同时, 为了提高缺失并发度, 我们也可以通过设置合理的 MSHR 数量来允许更多的缺失访问相互重叠.

这些图应用的纯粹平均缺失代价如图 14 所示. 与命中并发度和缺失并发度类似, 各类图应用在在 L2 cache 上所表现出来的纯粹平均缺失代价都是最高的, 这也从侧面反映了 L2 cache 中图数据访问缺失率高, 而为了避免这种现象, 最简单的方式就是在 L2 cache 上进行旁路策略, 可以有效降低总体的图数据访问时间.

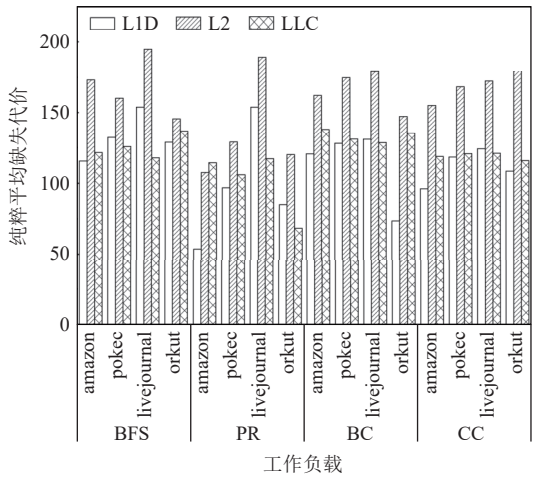

Fig. 14 Pure average miss penalty in each level of cache

图 14 各级 cache 中的纯粹平均缺失代价

多发射流水线技术是处理器微体系结构设计的一个重要改进, 它允许在同一周期内取指、解码、发射、执行和提交多个指令, 大幅度提高了指令集并行度. 由于算法数据依赖、分支错误预测的高额代价、负载数据约束, 以及一般的功率墙限制, 增加发射带宽不是一个可行的选择, 所以大多数商业处理器在不同的处理阶段对每个内核采用 4~8 大小的带宽[27]. 为了探究发射带宽对图应用的影响, 我们通过设置不同大小的发射带宽, 观察了其对图数据在各级 cache 中访存时间的影响. 由于 L1 data cache 的数据访问时间最能表现图应用的存储器性能, 因此这里我们只统计了不同发射带宽下 L1 data cache 的 C-AMAT, 其结果如图 15 所示. 可以看到, 当发射带宽设置为 4 时, 图应用的 C-AMAT 相较于其他数值的发射带宽都呈现出下降的趋势, 访存速度最快.

现代处理器中的非阻塞缓存为了在缓存缺失时依然能够提供数据, 往往采用了失效状态保存寄存器 MSHR. 对于 LLC, 当其 MSHR 表为空时, 表示没

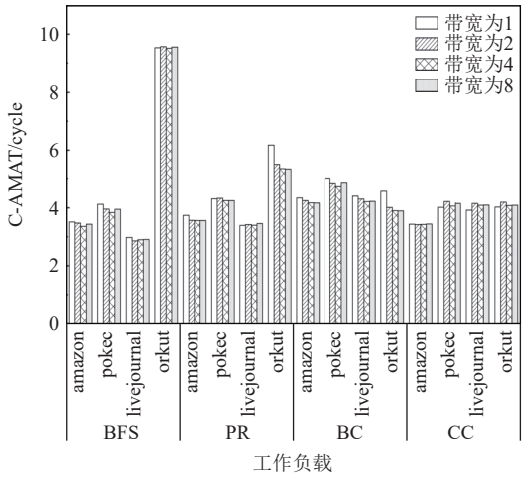

Fig. 15 C-AMAT of L1 data cache at different issue widths

图 15 不同发射带宽下 L1 data cache 的 C-AMAT

有未完成的主存访问, 而当 MSHR 表为满时, 则表示缓存无法提供更多的缓存访问, 将阻塞 CPU 对内存或下一级内存的访问. 因此, MSHR 表项的数量可以直接决定缺失并发度, 进而影响到 C-AMAT. 我们将 LLC 的 MSHR 表项的数量分别设置为 4, 8, 16, 并对图应用的 C-AMAT 进行了测量, 结果如图 16 所示. 不难发现, 当 MSHR 表项的数量设置为 8 时, 就已经对图应用的数据访问不再产生影响, 因此尽管图数据在 LLC 上的缺失率很高, 但图应用在 LLC 数据访问总占比是极低的, 尤其是单核模式下, MSHR 表项的数量的设置越大对于图应用而言意义不大, 对于图应用而言 LLC 的 MSHR 表项的数量设置在 4~8 已足够使用, 不必再浪费额外的硬件资源.

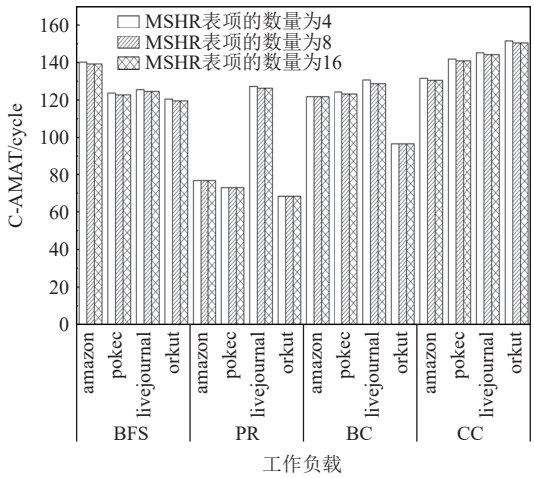

Fig. 16 C-AMAT of LLC at different sizes of MSHR

图 16 不同 MSHR 大小下 LLC 的 C-AMAT

指令集并行技术带来的图应用访存性能的提升仍存在很大的限制, 增加每个 CPU 内核数量是提高系统总体性能的最直接和最高效的手段. 我们将内

核数量进行了倍增,将图应用在 L1 data cache 的 C-AMAT 进行了统计分析,实验结果如图 17 所示. 内核数量的增加明显降低了图应用的 C-AMAT,但并不总是如此,尤其对于 PR 算法而言,在其中 3 个数据集中双核 CPU 下的 L1 data cache 的 C-AMAT 相对于单核反而出现上升,我们推测产生这种现象是由于图应用中加载指令之间的依赖链过短,从而导致了流水线的断流和低内存级并行. 但总的来说,多核 CPU 在处理图应用过程中所表现的存储器性能依然是要高于单核 CPU 的.

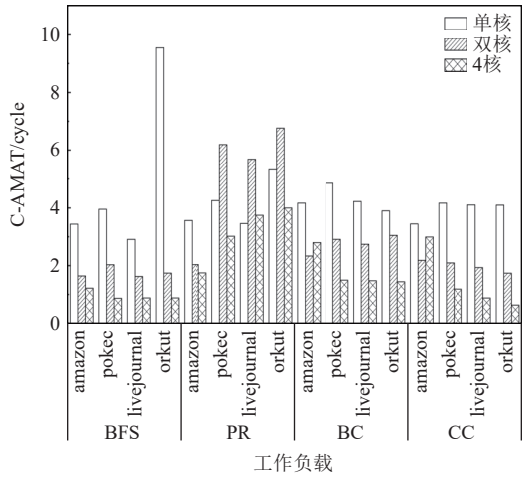

Fig. 17 C-AMAT of L1 data dcache at different number of cores

图 17 不同内核数量下 L1 data cache 的 C-AMAT

5 总 结

图计算应用拥有特有的不规则执行行为,其引发的不规则负载和不规则访问是加速图应用面临的主要挑战之一. 认识图应用高度复杂的存储系统的局部性信息和并发性信息对于加速图计算架构设计至关重要. C-AMAT 可以有效表征应用程序运行过程中高度复杂的存储系统的局部性和并发性的信息,但是其计算模型没有考虑 cache 层的访问方式,原始测量装置硬件开销巨大. 因此本文基于不同 cache 层的访问结构,对 C-AMAT 的计算模型扩展为 PC-AMAT 和 SC-AMAT,并在此基础上设计并实现了纯粹缺失周期提取算法及各参数的测量与计算,能够有效实现 C-AMAT 的测量. 实验结果表明,在单核和多核模式下,PC-AMAT 和 SC-AMAT 与 IPC 之间的相关性比 C-AMAT 与 IPC 的相关性更强,在 4 核模式下,L1 cache 中 PC-AMAT 相比 C-AMAT 的相关系数提升了 11.7%, SC-AMAT 在 L2 cache 和 LLC 上的相关系数分别提升了 26.1% 和 15.8%. 最终,我们通过更改相关的硬件配置利用 C-AMAT 评估和分析了图应用在存储器中的访存性能,并提出了相应的访存优化策略. 我们的下一步工作将根据如何提高图应用的命中并发度和缺失并发度,以及如何降低图应用的纯粹缺失率展开,拟从 cache 层面优化图应用的存储器性能.

作者贡献声明:陈炳彰提出算法思路和实验方案,并完成实验与论文撰写;刘伟指导方案设计并修改论文;于萧钰协助论文修改.

参 考 文 献

[1] Basak A, Li Shuangchen, Hu Xing, et al. Analysis and optimization of the memory hierarchy for graph processing workloads[C]// Proc of the 25th IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2019: 373−386

[2] Sun Xianhe, Wang Dawei. Concurrent average memory access time[J]. Computer, 2013, 47(5): 74−80

[3] Binkert N, Beckmann B, Black G, et al. The gem5 simulator[J]. SIGARCH Computer Architecture News, 2011, 39(2): 1−7

[4] Beutel A, Akoglu L, Faloutsos C. Graph-based user behavior modeling: From prediction to fraud detection[C]// Proc of the 21st ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2015: 2309−2310

[5] Barabási A L, Albert R. Emergence of scaling in random networks[J]. Science, 1999, 286(5439): 509−512

[6] Sun Xiangzheng, Zhang Yunquan, Wang Ting, et al. Auto- tuning of SpMV for diagonal sparse matrices[J]. Journal of Computer Research and Development, 2013, 50(3): 648−656 (in Chinese)
(孙相征, 张云泉, 王婷, 等. 对角线稀疏矩阵的 SpMV 自适应性能优化 [J]. 计算机研究与发展, 2013, 50(3): 648−656)

[7] Balaji V, Crago N, Jaleel A, et al. P-opt: Practical optimal cache replacement for graph analytics[C]// Proc of the 27th IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2021: 668−681

[8] Tang Jiawu, Zheng Long, Liao Xiaofei, et al. Effective high-level synthesis for high-performance graph processing[J]. Journal of Computer Research and Development, 2021, 58(3): 467−478 (in Chinese)
(汤嘉武, 郑龙, 廖小飞, 等. 面向高性能图计算的高效高层次综合方法 [J]. 计算机研究与发展, 2021, 58(3): 467−478)

[9] Faldu P, Diamond J, Grot B. Domain-specialized cache management for graph analytics[C]// Proc of the 26th IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2020: 234−248

[10] Cooksey R, Jourdan S, Grunwald D. A stateless, content-directed data prefetching mechanism[J]. ACM SIGPLAN Notices, 2002, 37(10):

279−290

[11] Agarwal A, Roy K, Vijaykumar T N. Exploring high bandwidth pipelined cache architecture for scaled technology[C]// Proc of the 6th Design, Automation and Test in Europe Conf and Exhibition. Piscataway, NJ: IEEE, 2003: 778−783

[12] Kroft D. Lockup-free instruction fetch/prefetch cache organization[C]// Proc of the 25th Int Symp on Computer Architecture. New York: ACM, 1998: 195−201

[13] Beamer S, Asanović K, Patterson D. The GAP benchmark suite[J]. arXiv preprint, arXiv: 1508. 03619, 2015

[14] Wang Dawei, Sun Xianhe. APC: A novel memory metric and measurement methodology for modern memory systems[J]. IEEE Transactions on Computers, 2013, 63(7): 1626−1639

[15] Soltis D, Gibson M. Itanium 2 processor microarchitecture overview[C]// Proc of the 14th Hot Chips. Los Alamitos: IEEE Computer Society, 2002: 44−54

[16] Yao Yongbin. Superscalar Processor Design[M]. Beijing: Tsinghua University Press, 2014 (in Chinese)
(姚永斌. 超标量处理器设计 [M]. 北京: 清华大学出版社, 2014)

[17] Gold X. 6130-Intel-WikiChip[EB/OL]. [2022-07-16].https://en.wikichip.org/ wiki/WikiChip

[18] Sun Xianhe. C-AMAT: Data access model in the age of big data[J]. Communications of the CCF, 2014, 10(6): 19−22 (in Chinese)
(孙贤和. C-AMAT: 大数据时代的数据存取模型 [J]. 中国计算机学会通讯, 2014, 10(6): 19−22)

[19] Fog A. The Microarchitecture of Intel, AMD and VIA CPUs : An optimization guide for assembly programmers and compiler makers[R]. Denmark: Copenhagen University College of Engineering, 2022

[20] Beamer S, Asanovic K, Patterson D. Direction-optimizing breadth-first search[C/OL]// Proc of the 12th Int Conf on High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2012 [2022-07-16]. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=646845 8

[21] Brin S, Page L. The anatomy of a large-scale hypertextual web search engine[J]. Computer Networks and ISDN Systems, 1998, 30(1-7): 107−117

[22] Madduri K, Ediger D, Jiang K, et al. A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets[C]// Proc of the 23rd IEEE Int Symp on Parallel & Distributed Processing. Piscataway, NJ: IEEE, 2009: 1−8

[23] Sutton M, Ben-Nun T, Barak A. Optimizing parallel graph connectivity computation via subgraph sampling[C]// Proc of the 32nd IEEE Int Parallel and Distributed Processing Symp (IPDPS). Piscataway, NJ: IEEE, 2018: 12−21

[24] da Rosa Righi R, Lehmann M, Gomes M M, et al. A survey on global management view: Toward combining system monitoring, resource management, and load prediction[J]. Journal of Grid Computing, 2019, 17(9): 473−502

[25] VanVoorhis C R W, Morgan B L. Understanding power and rules of thumb for determining sample sizes[J]. Tutorials in Quantitative Methods for Psychology, 2007, 3(2): 43−50

[26] Ye Nan, Hao Ziyu, Zheng Fang, et al. Adaptability of BFS algorithm and many-core processor[J]. Journal of Computer Research and Development, 2015, 52(5): 1187−1197 (in Chinese)
(叶楠, 郝子宇, 郑方, 等. BFS 算法与众核处理器的适应性研究 [J]. 计算机研究与发展, 2015, 52(5): 1187−1197)

[27] Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Approach[M]. Amsterdam: Elsevier, 2011

Chen Bingzhang, born in 1998. Master candidate. His main research interests include graph computing, memory and I/O system, and performance evaluation and optimization of storage system.

陈炳彰, 1998 年生. 硕士研究生. 主要研究方向为图计算、内存和 I/O 系统、存储系统性能评估和优化.

Liu Wei, born in 1978. PhD, associate professor. His main research interests include graph computing, green computing and memory computing, cloud computing and edge computing, and intelligent manufacturing and industrial Internet.

刘　伟, 1978 年生. 博士, 副教授. 主要研究方向为图计算、绿色计算与内存计算、云计算与边缘计算、智能制造与工业互联网.

Yu Xiaoyu, born in 1999. Master candidate. His main research interests include graph computing and memory system performance optimization.

于萧钰, 1999 年生. 硕士研究生. 主要研究方向为图计算、内存系统性能优化.