**Cache局部性度量方法研究**

刘松 谢骁 李彦杰 伍卫国

西安交通大学电信学院，西安 710049

#### Research on Locality Measuring Methods for Cache

Liu Song, XieXiao, Li Yanjie and Wu Weiguo

(School of Electronic and Information Engineering, Xi’an Jiaotong University, Xi’an 710049)

**Abstract** Locality principle is one of the basic theories in computer science, which is widely used in memory management and program optimization. Although there have been a lot of approaches about locality metrics at present, they are still inadequate when deal with the cases, such as the optimization design of hierarchical memory structure and the compiler optimization of application structure due to the emergence of multithread and other new scenarios. With the development of multicore architecture and the rise of shared cache, locality metrics are more complex. In order to study the theory and its application better, it is necessary to summarize and analyze the existing related research status clearly. In this survey, we introduce recent development of locality metrics and the relevant models using reuse distance, footprint theory and probability-based quantitative approach. Firstly, we categorize the methods based on different measurement strategies and analyze the merits and demerits of different theoretical models as well as their application scopes. Then we summarize the locality problems of the present state of the field on modern computer architectures such as multicore and GPU architecture. Finally, we state our conclusion and prospect the future research direction on locality metrics.

**Key words** Locality Measuring; reuse distance; footprint theory; [performance](C:/Users/yer/AppData/Local/Yodao/DeskDict/frame/20140706224134/javascript:void(0);) [optimization](C:/Users/yer/AppData/Local/Yodao/DeskDict/frame/20140706224134/javascript:void(0);); cache;

摘要 局部性原理是计算机学科的基础理论之一，被广泛应用于内存管理和程序优化等方面。随着多核架构的普及和共享cache的出现，局部性度量变得更加复杂。虽然现存多种局部性度量方法，然而将其运用在多级共享存储系统的优化设计和编译优化时，由于多线程等新场景的出现使得很多既有方法不再适用。为了更好的对局部性理论与应用进行研究，有必要对局部性度量相关研究进行总结分析。文中首先对局部性度量的研究现状及相关模型进行了分析，主要从重用距离、footprint理论和基于概率统计的局部性度量方法三方面展开论述，依据局部性度量方法的不同分别对它们进行了归纳和总结模型的优缺点以及适用范围。其次，对目前多核架构和众核架构计算环境下局部性度量研究中cache共享和数据共享等问题进行了探讨。最后，文章对cache局部性度量进行了总结，并对未来研究方向进行了展望。

关键词局部性度量；重用距离；footprint理论；性能优化；高速缓存；

中图法分类号 TP391

在现代计算机系统结构中，缓存无处不在，高效的缓存设计依赖于局部性原理。而局部性原理作为计算机学科的基本原理之一，是计算机存储层次结构设计的主要依据，并在系统性能模型、程序优化、存储层次管理以及网络通信分析等方面有着重要的指导意义。1959年Atlas操作系统首次实现了虚拟存储[1]，1961年IBM Stretch超级计算机首次采用空间局部性原理进行指令预取，1965年CPU cache原型正式出现[2]，1966年工作集思想提出[3-7]，一系列局部性原理的应用大幅度提升了计算机的运行速度。同时由于集成电路中晶体管体积不能无限缩小，数量不能够无限增加，信号传输速度也不能无限提升，通过传统工艺手段实现计算机性能的提升已经达到了瓶颈。目前有效的解决方案之一是在现有的工艺技术基础上充分挖掘和利用已有资源来提高计算性能。根据“二八原理”，即程序运行过程中80%的计算操作仅依赖于20%的程序数据。通过有效利用局部性理论，将20%的常用数据长时间存储在cache中，能够减少该部分数据替换次数，从而节约大量的数据访存时间，提高程序执行速度。

CPU访问存储器时，无论是存取指令或存储数据，所访问的[存储单元](http://baike.baidu.com/view/1223079.htm)都趋向于聚集在一个较小的连续区域中，这一特性称为局部性原理[8]。局部性又分为时间局部性和空间局部性两种：其中时间局部性指现在被执行的指令和访问的数据在近期可能会再次访问，程序循环、[堆栈](http://baike.baidu.com/view/93201.htm)等是产生时间局部性的主要原因；而空间局部性指当前被访问数据的相邻数据很可能在近期被访问，数组的连续存放、指令顺序执行等是产生空间局部性的主要原因。一个编写良好的程序一般也具备良好的局部性。重复的引用同一变量能产生良好的时间局部性。针对步长为*k*的引用模式，步长越小空间局部性越好。对于指令执行顺序，循环有良好的时间和空间局部性，并且循环体越小、循环迭代次数越多，局部性越好。

自cache诞生以来，人们希望局部性原理不仅可以定性地指导设计编译优化系统，更希望通过定量的计算实现程序局部性的最佳利用。从工作集理论[5]到近年的cache高阶局部性理论[9]，局部性理论的研究取得了长足进步。然而，无论是基于程序代码的分析，还是基于程序运行时地址序列的分析，局部性分析仍然局限在定性的程序性能评价，而没有定量的指导程序进行优化，并且局部性分析时间较长。本文希望通过研究找到一种简单易行的方法，在线性时间内进行分析和预测，并可以直接对应用程序进行优化指导。针对现代应用日益增长的大数据处理需求，计算机操作系统管理大量的内存以满足实际需要，利用局部性理论可以使资源利用率最大化。随着多核/众核架构的普及，通信和资源竞争等因素的存在，使得共享cache的局部性分析成为计算机学科的一个难题，针对共享cache的局部性分析和优化值得深入的研究。另外，如何将局部性理论中的一些思想扩展应用在其他领域也是一个值得思考的问题。

本文第1节对一些常见的局部性度量标准进行介绍，并对各个度量标准之间的转换关系进行说明。第2节对基于重用距离的局部性分析进行归纳和总结。第3节对基于Footprint的局部性分析进行说明。第4节针对基于概率统计提出的局部性分析方法进行探讨。第5节针对多核和众核架构中计算机局部性进行分析，并对局部性分析中的大数据问题进行了探索。第6节对全文进行总结，并对局部性原理的发展进行了展望。

## 1 局部性度量标准

工作集局部性理论(Working Set Locality Theory , WSLT)中定义工作集大小为一个时间窗口内所访问的平均内存页数，并提出了一个线性时间复杂度的计算方法[3]，它奠定了局部性理论研究领域的基础，成功解决了存储层次结构所面临的巨大挑战。该方法针对程序在每个运行阶段访问的缓存块集合具有相对稳定不变的特性，作为对主存局部性进行分析的理论依据，并广泛应用于虚拟内存管理[5]，大大提高了计算机访存性能。随后针对工作集理论的扩充和改进，先后提出了多种局部性度量标准，常见的有五种，分别为：Footprint、失效时间间隔(Inter-miss Time)、容量填充时间(Volume Fill Time)和失效率(Miss Ratio)和重用距离(Reuse Distance)。

Footprint指在给定长度的时间窗口内所访问的数据量大小。对于一个可执行程序，可以得到它的访存地址序列并假设序列长度为*n*，，序列中的一个连续地址子序列称为一个时间窗口，总数为*n\**(*n+*1)/2个，每一个时间窗口中不同数据的个数即为当前窗口的Footprint。Footprint通常针对所有窗口进行计算，可以有效衡量整个程序的局部性。为了减少计算的复杂性，一般针对指定长度的时间窗口进行计算并求平均值作为局部性的评价依据。

容量填充时间指在cache容量固定时，程序访问的数据占满整个cache的时间，适合针对特定大小的cache进行分析，对机器硬件具有依赖性。容量填充时间可以通过跟踪程序执行地址直接计算得到，但是计算时间过长，实用性不高；也可以通过计算Footprint等于cache容量大小所需的时间，即Footprint的反函数来间接获得。

失效率较常见，指在特定次数的访存中，发生缓存失效次数占总访存次数的比值。失效率主要用于计算机系统结构[10,11]、内存管理和虚拟机设计[12,13]。由于存在失效开销，cache的失效率越低，计算速度越快，吞吐率越大，因此提高局部性可以有效减少失效率。失效率反映了程序执行过程的整体局部性情况，被广泛用于程序性能的评估。

失效时间间隔指在cache容量固定时，两次访存失效的时间间隔的平均值。由定义可知，失效率和失效时间间隔二者均为针对访存失效的评价指标，后者与前者相反，失效时间间隔值越大局部性越差，cache失效率越低，两者成倒数关系。

重用距离指两个连续的相同访问数据间隔内，所访问的不同数据的个数，[14,15]，也称为LRU栈距离，由Mattson等人首次提出[11]。重用距离针对重用窗口进行计算，重用窗口为footprint时间窗口的一个子集，对于给定数据量为*m*长度为*n*的地址访问序列，重用窗口个数为*n-m*，远小于时间窗口个数。重用距离反映了数据的重用情况，主要用于程序分析和优化[16]。重用距离的计算包括被访问数据本身，因此最小值为1，重用距离值越小程序局部性越好。

∞ ∞ ∞ 3 1 2 2 2

***a b c a a c a c***

(a)reuse distances of a data access trace

  (b)reuse signature (c)the miss rate curve

图1重用距离的计算

Fig.1 Calculation of reuse distance

文献[9]提出了高阶局部性理论(A Higher Order Theory of Locality, HOTL)，并揭示了各个局部性度量标准的关系，包括一系列的转换方法以及转换成功的必要条件，为程序的局部性分析提供了数学理论模型。文章指出，只有在所有重用窗口的平均Footprint等于所有时间窗口的平均Footprint的条件下，即重用窗口是时间窗口的一个特征子集时，各个度量标准才能相互推导。表1为各个度量标准的转换关系，通过从上至下求导或从下至上求积分可以得到其他变量。

失效率通常直接通过设置硬件或软件计数器计数得到，主要用于性能评价，让人们对cache失效情况有一个整体直观的了解，但它不能反映局部性的整体分布情况。失效时间间隔与失效率互为倒数关系，计算时需要记录每次失效的时间，计算过程更为复杂，并且也不能反映局部性的分布情况。而容量填充时间对机器硬件具有依赖性，可以通过Footprint的反函数得到。由此可见，目前的局部性度量方法主要采用重用距离和Footprint进行分析和计算，其中重用距离主要用来针对特定的程序进行分析和优化，Footprint则多用于共享cache的管理和优化。本文分别对这两种度量标准进行详细的介绍和分析。

表1 几种度量标准的关联性（高阶局部性理论）

Table 1 the Higher Order Relations

|  |  |  |
| --- | --- | --- |
| 局部性度量 | 函数性质 | 有用的特性 |
| 3阶：  footprint,  容量填充时间 | 凹函数/凸函数 | 线性时间,可抽样,可组合  (动态局部性) |
| 2阶:  失效率,  失效时间间隔 | 单调 | 机器无关性，如：Cache大小，映射方式  (Cache局部性) |
| 1阶：  重用距离 | 非负 | 代码单元和数据结构可分解(程序局部性) |

## 2基于重用距离的局部性分析

对于局部性理论，数据访问和重用的关系就如图论中点和线的关系一样密切，重用距离在程序局部性分析中有着重要的地位。缩短重用距离能在将来更短时间内实现当前数据的重用，减少cache的失效率。

2.1 重用距离

一个程序的数据重用距离分布情况一般采用重用距离直方图表示[15]，每个柱体代表对应重用距离长度出现的概率（不考虑重用距离无穷大的情况）。重用距离直方图直观地描绘出数据局部性的整体分布情况，很多关于重用距离的研究都是根据重用距离直方图变形得到，因此了解重用距离直方图至关重要。图1为一个访问序列实例的重用距离及其对应的重用距离直方图和cache失效率曲线图。当数据首次出现时，重用距离为无穷；当重用距离大于cache容量时出现cache访问失效，根据大于cache容量的重用距离对应的概率可以计算得到cache的失效率。重用距离直方图是时间局部性的量化表示[16]，由它计算得到的失效率曲线图说明重用距离在不同cache容量下对失效率的影响，只有保证cache容量大于重用距离或通过优化使重用距离尽量小于cache容量，程序的访存命中率才会最大。

对于一个采用LRU策略的全相联映射cache而言，当重用距离大于cache容量时就会出现数据访问失效。用*hits(C)*和*misses(C)*分别表示cache命中次数和未命中次数，*s(Δ)*表示重用时距离为*Δ*的数据量，三者关系如公式(1)、(2)和(3)所示。

 (1)

 (2)

(3)

由此可知，重用距离可以有效地评价程序的整体局部性分布。针对重用距离的局部性分析方法主要分为两类：静态局部性分析和基于训练的局部性分析。重用距离主要利用地址序列进行分析和计算，相应的地址序列处理方法也分为两类：(1)将地址序列看作列表或栈；(2)利用树等数据结构对地址序列进行优化处理。静态分析方法大多采用方法(1)，基于训练的局部性度量采用方法(2)。因为大多数程序地址序列数据量较大，重用距离的计算时间较长，且很多时候并不要求得到精确的局部性分析结果，所以可以容忍损失一定计算精度来减少计算时间，在一定的误差范围内对局部性进行计算，基于训练的局部性分析一般采用这种可控误差下的不精确测量。

失效率%

百分比%

**2.2 静态局部性分析**

计算重用距离最简单直接的一种方法是离线分析：首先收集程序的整个内存访问地址序列，再对整个地址序列进行分析。对于一个长度为*n*，数据量为*m*的访存序列，计算重用距离的时间复杂度为*O(n\*m)*[11]，因此随着*n*和*m*的增大，计算整个重用距离直方图的代价也将迅速增加[17]。对于一个运行时间较长的程序而言，由于需要记录每次的访存地址，还会耗费大量的存储空间，因此该方法对大多数程序而言不具有实用价值。另一种计算重用距离的方法是程序分析，即直接针对程序代码进行分析。但由于一般程序复杂的控制流和间接数据访问导致程序行为特征分类非常困难，因此程序分析对一般的程序也不具有通用性。

虽然静态局部性分析方法对一般程序的分析存在各种困难，但针对以线性索引的数组引用类为主的循环嵌套程序有着独特的优势，通过静态分析可以对循环中的迭代空间和数据空间进行精确的建模。静态局部性分析可以对高维数据的局部性进行分析，从而应用于编译优化，通过重排程序的执行顺序和改变程序的平衡来加快程序运行速度。由于依赖性检查的基本任务之一为分析重复访问的数据，因此可以通过依赖关系计算重用距离。文献[18]对依赖性分析进行了扩展，用其估算科学计算程序中的重用距离大小和局部性分布。文献[19]提出一种重用距离方程，在Omega库[20]的帮助下对循环多面体模型(Polyhedral Model)的重用距离实现精确计算，但存在最坏情况使得计算时间呈指数增长。文献[21]将静态局部性分析方法结合到编译技术中来计算程序的失效率。

**2.3 基于训练的局部性分析**

对于局部性分析，还可以对前期分析得到的结果建立模型，以便对其他同类型情况进行预测，即基于训练的局部性分析。相关研究通常在程序运行时搜集内存访问片段传递给分析程序，分析程序再采用不同的算法对重用距离进行分类预测。分析程序一般把地址序列构建成各种树形结构后进行数据分析，例如AVL树[22]和splay树[23]等。

文献[24]首次采用树结构将重用距离计算时间减少为*O(n\*log n)*。作为对[24]的改进，Olken使用渐近线的方法对树结构进行压缩，将计算时间复杂度减少为*O(n \*log m)*[22]，代价是牺牲了一定程度的计算精度。随后Cheetah利用splay-tree[23]对Olken的算法进行改进，并将其应用在广泛使用的模拟器SimpleScalar[25]上。文献[26]也采用了一种新的树结构，提高了算法运行速度。Ding和Zhong则采用一种逼近算法，将计算精度提高至99%，并将时间复杂度减少为*O(n\*log log m)*，该值非常接近于*O(n)*[16]。

以反馈为导向的优化度量在设计和构建优化编译方面成为一种日益重要的分析方法。基于训练的局部性分析在预测程序内存行为方面有着巨大应用潜力，不仅可以预测整个程序的局部性和失效率，也可以针对每条指令进行预测。为了预测每条指令的局部性，需要找出程序中对cache局部性影响较大的指令。Fang等人[27]对文献[16]中的工作进行了扩展，不仅利用内存地址分析重用距离进行局部性预测，而且将内存地址关联到特定的静态内存操作，从而预测每条指令的重用距离，进而找出程序中对重用距离产生主要影响的关键指令，并根据这些关键指令有效地预测局部性。此外，基于训练的局部性分析能针对一个给定的数据大小，识别出可能产生大量缓存不命中的关键内存操作。通过这些信息，编译器可以实现缓存优化，从而提高程序运行速率。然而，即便是最有效的算法，增加整个程序的内存访问分析功能也会使程序的执行速度减缓一到两个数量级[28]。文献[29]利用GPU架构的高效并行性加速了重用距离的分析过程。

**2.4 重用距离的优化应用**

重用距离不仅是衡量程序局部性的重要依据，还能对程序运行的性能进行优化。这种优化主要应用在与科学计算相关的程序上，且核心代码大都为循环嵌套。对于循环程序，在不破坏原有程序数据依赖关系的前提下，对程序进行局部性优化来提高数据的局部性，这是一种高效的编译优化方法。

循环分块是一种常用的提高科学计算程序数据局部性的技术[34,35]。文献[36]在文献[38]的基础上针对矩阵访问采用一种简单却精确的混合式循环转换[37]方法对程序进行变换，通过调整数据访问顺序有效的提高了数据局部性，然而由于大多数科学计算程序都是针对数组的循环操作，而数组的存取本身已经具备很高的局部性，所以这种方法只能提高少部分程序的局部性。文献[39]也采用循环转换的方法对程序进行优化，不同点在于作者使用基于重用和局部性的数学公式以及幺模矩阵变换进行算法设计，首次考虑cache组相联引起的数据冲突，减少循环变换搜索空间大小，使算法执行更加简单直接。文献[40]对通用计算机结构中的循环程序进行依赖性分析，实现自动向量化和并行化处理。文献[18]使用栈距离计算循环程序的cache命中率和局部性，文献[41]对Matlab脚本程序进行依赖性分析，利用静态重用距离分析方法进行局部性优化。

随着多核架构出现，仅针对共享cache进行程序行为的分析并不能充分利用程序的局部性，还需要针对私有cache进行分析。对于多级缓存结构，多级包容性设计模式存在着包容性受害者问题(Inclusion Victim)[42]，即由于私有缓存和共享缓存的存在，当多个程序同时运行时，共享缓存产生的冲突将导致私有缓存相应数据逐出，使得多个程序共同运行时cache失效率明显高于程序单独运行时cache失效率。以矩阵乘法为例（图2），图3是在私有L1 cache大小为256KB，共享L2 cache大小为2MB的多核处理器上运行矩阵乘法程序时分块大小和cache性能的关系图，其中分块因子，且。图中横轴表示第*i*层循环中*B[j][k]*的重用距离，即8\*(*B\*B+2\*B*)/1024 *KB*，纵轴表示L1 cache的失效率，两条曲线分别表示程序单独运行时以及与其他程序

*for(i=0;i<N;i=i+1)*

*for(j=0;j<N;j=j+1)*

*for(k=0;k<N;k=k+1)*

*C[i][j]=beta\*C[i][j]+alpha\*A[i][k]\*B[k][j];*

(a)Original Code

*for(jj=0;jj<N;jj=jj+Bj)*

*for(kk=0;kk<N;kk=kk+Bk)*

*for(i=0;i<N;i=i+1)*

*for(j=jj;j<min(jj+Bj,N);j=j+1)*

*for(k=kk;k<min(kk+Bk,N);k=k+1)*

*C[i][j]=beta\*C[i][j]+alpha\*A[i][k]\*B[k][j];*

(b)Tiled code

图2 矩阵乘法中的循环分块

Fig.2 Loop tiling in matrix multiplication



分块大小为B[j][k]的重用距离

失效率

图3 矩阵乘在不同分块下的cache失效率

Fig.3 The effect of cache interference on tiled matrix multiply

共同运行时L1 cache的失效率变化。

从图3可以看出失效率起初随着分块大小的增加而减小，这是因为随着分块大小的增加，L1 cache得到更充分的利用。然而，当B大于某阈值后，失效率又开始增加，一方面由于L1 cache容量限制使得没有足够的空间容纳所有的数据，另一方面则因为数据冲突产生影响。曲线最低点对应的分块大小即为最优分块大小[43]，但在程序单独运行时的最低失效率处，程序共同运行时的失效率是单独运行时的10倍左右。

程序间的影响是相互的，一方面被其他程序影响，称为程序的敏感度(sensitivity)；另一方面也影响其他程序的运行，称为亲和度(friendliness)[44,45]。文献[46]中，Bin Bao等人针对程序的亲和度进行分析，提出了防御性循环分块策略，通过循环分析和变换，评估其对共享cache产生的影响，进而选择合适分块大小，为并行程序提供较高的性能。该方法的核心是建立代价模型，该模型主要分为两部分：处理器模型和cache模型。处理器模型主要评估指令调度和寄存器不足对性能的影响，cache模型则评估缓存不命中次数对性能的影响。相对其他多核架构共享cache的研究，该方法开发了一个完全静态的程序优化框架，有效减少了包容性受害者问题对程序运行速度的影响，且不要求profiling信息，但该方法只适用于特定应用，不具有通用性。

**2.5 小结**

利用重用距离的分析结果可以对内存系统性能各个方面进行预测，例如直接测量全相联下LRU策略的cache命中率[47]、整个程序的局部性[14]、局部性阶段变化规律[48]、程序不同输入下的cache失效率[49]；也可以被用来指导各种优化，例如生成cache索引[19]、指导循环分块[50,51]、重排数据和代码以提高局部性[52]等。由于重用距离是针对程序运行地址进行分析，和cache等硬件参数无关，避免了直接代码分析时复杂的控制流和数据间接访问等问题，并且该方法不受代码编写和执行变化影响，使得计算结果更加精确。由重用距离可以得到cache失效率，反映程序访问数据重用情况的整体分布，在程序分析和局部性优化方面有着广泛的应用。

尽管众多学者对重用距离进行了大量的研究并解决了很多实际问题，但是它依然有着自身的局限性。第一，对于共享cache而言，程序间交叉存取几乎存在于整个执行阶段，重用距离却只是针对重用窗口进行计算，而重用窗口并不包括所有窗口，不能完全地反映程序间交互的情况。第二，尽管我们可以通过并行、抽样等技术进行渐近计算得到重用距离，但是对于精确计算而言，时间代价依旧很大。第三，多个程序并行运行时，重用距离会受到其他并行运行程序的影响而变长，如何量化这种变化以及如何根据这种变化来进行任务调度也值得考虑。因此我们还需要其他度量标准来解决这些问题。

## 3 基于Footprint的局部性分析

Footprint针对程序执行中所有的时间窗口进行分析，依赖于所有的cache访问，而不仅仅是cache不命中情况，能够直观地反映程序的局部性和程序的访存行为特征。相对于重用距离而言，footprint的计算更全面，同时利用自身特点解决了一些重用距离无法解决的问题，在很多方面具有不可替代的优势。

**3.1 可组合性**

Footprint最大的一个特点就是可组合性，也就是说在没有数据共享的情况下，程序共同运行时得到的footprint值可以由各个程序单独运行时的结果计算得出。设一个程序时间窗口长度为*x*的footprint的平均值表示为*prog.fp(x)*，有*k*个程序*prog1,prog2,...,prog k*共享cache，则共同运行时总的footprint值就是所有程序footprint值的总和，即：。

并行运行时的联合footprint值

程序单独运行时的footprint值

程序单独运行时失效率

程序并行运行时失效率

程序私有重用距离(PRD)

程序并发重用距离(CRD)

可组合性

度量转换

图4 footprint的组合特性(点线)以及它和其他变量的转换关系（实线）

Fig.4 Composition (dotted line) and conversion (solid lines)

Footprint可以和别的变量间相互转换，因此可以通过footprint值计算其他变量[53]。图4中是利用footprint的可组合性以及它和别的度量标准的转换关系，间接计算cache失效率和重用距离的图形关系表示，其中PRD(Private Reuse Distance)表示程序单独运行时的重用距离，称为私有重用距离，CRD(Concurrent Reuse Distance)表示多核架构中并行运行的程序交叉访存时的重用距离，称为并发重用距离[54,55]。

除了可以通过自身的可组合性简化其他度量标准的计算之外，footprint的可组合性还有很多其它优点。第一，机器无关性。由于整个分析过程是针对数据访问序列而言的，和具体的机器设备并无联系，因此可以直接计算而无需考虑程序运行的机器架构。第二，近似统计特性。程序可以在并行运行环境下计算其footprint值而不受其他程序的干扰。第三，程序独立性。程序的footprint值并不会受到其他并行运行程序的影响。第四，静态可加性。*n*个程序便会有2*n*种组合方式，footprint计算结果却不受不同排列的影响。基于这些优点，footprint理论解决了很多cache共享引起的悬而未决的问题[56]。传统的cache性能大都是通过模拟器进行评估的[57-59]，该方法评估结果精度高，但模拟时间过长。硬件检测方法虽然可以动态的加速评估时间[60]，却受限于特定机器的cache配置。另外，模拟评估和硬件检测都只能用来评估程序上下文切换对系统性能的影响[61,62]，很少涉及影响cache性能本质的分析，使得其结论很难用来提高cache性能。文献[63]利用footprint的可组合性以及它的机器无关性对分时系统下cache重载瞬态进行分析建模，解决了其他模型中参数对系统参数的依赖问题。文献[64]针对多处理器系统下cache失效率，以footprint为参数建立分析模型，并以此进行cache分区选择进而提高性能。文献[9]通过分析建立footprint和重用距离的转化关系，简化了重用距离的计算复杂度。

**3.2 Footprint计算方法**

图5举例说明数据块*a，b，c*在时间窗口长度为3时footprint值的计算过程，易知*fp*(3) = (3+2+2+3)/4 = 2.5。在动态执行过程中，不同长度的窗口和相同长度下的不同位置的窗口所访问的数据均不相同，footprint计算要求包括所有情况下窗口中数据量。由于footprint的大小和时间窗口个数有关，给定数据量为*m*长度为*n*的访问地址序列，可知其时间复杂度为*O(n\*n)*，大于重用距离复杂度*O(n\*m)*，为了降低其复杂度，可以选取一个窗口子集或者构建一个模型进行逼近计算。

***a b c c b a***

*fp=3*

*fp=2*

*fp=2*

*fp=3*

图5 时间窗口为3时的footprint值，fp(3)=2.5

Fig.5 Footprint in length-3 window

Footprint值的计算方法可以分为三种。第一种针对整个地址空间进行分析，枚举所有*O(n\*n)*个不同时间窗口的footprint值，时间复杂度为*O(n\*logm)*，其中*n*为地址序列长度，*m*为不同窗口footprint值的最大值，计算时间开销相对于原始程序有时会慢上千倍[65,66]。第二种计算footprint平均值，无需列举所有footprint值，时间复杂度为*O(n)*，但是该方法有一个缺点，平均值容易受到个别异常大的数值影响[67]。第三种是抽样分析，仅对部分时间窗口进行计算，进一步减少时间复杂度[9]。由于cache容量大小的限制，实际操作中我们无须计算时间窗口大于cache大小的情况。考虑到现代计算机多核架构中我们完全可以实现原始程序和分析程序并行运行于两个核上，因此抽样并不会特别损害到每个样本的测量精度，而时间开销却减少为原始程序执行时间的0.5%左右，因此采样计算是一个不错的测量方法，但是不恰当的采样容易陷入局部极值问题，如何选择有效的采样率仍然是一个待解决的问题。

**3.3 Footprint的优化应用**

Footprint不仅是衡量程序局部性的重要依据，同时也用来进行程序运行性能的优化。这种优化主要集中在程序热点为循环的科学计算程序上。对于循环程序，因具有较为稳定的局部性特征，允许我们通过局部采样的方式预估全体的局部性特点。

局部性感知调度算法（locality-aware scheduling algorithm）[68]，过分析任务程序的局部性特征，采取合理的方法进行任务调度，使得计算资源得到合理利用，提升系统性能。但任务的局部性度量是一个复杂的过程，利用footprint的特性，不仅简化了多核处理器任务调度预处理时间，同时也保证了调度精度。

文献[69]针对无数据共享的科学计算程序，利用footprint的可组合特性以及和其他变量的转换关系，通过抽样估算出待执行任务的footprint值，从而计算出组合运行时局部性情况。再根据任务组合局部性进行任务到核的合理预分配。从实验结果来看，相比平均执行时间和操作系统任务分配执行时间有较大的性能提升。但文中限制了执行任务数小于核数，对于任务较多的情况，并没有给出有效的调度方案。

**3.4 小结**

Footprint有着其他度量标准没有的独特的优点，解决了多个程序并行执行情况下局部性难以分析的问题，简化了很多局部性分析过程，为局部性理论的研究提供了新思路。然而，footprint在程序多线程支持方面仍然面临着一些挑战。首先，footprint的可组合性必须满足线程间无数据共享的前提条件下，对于同一个程序生成的不同线程间共享数据的情况仍然需要进一步的研究。其次，分析工具将会改变多线程间数据交叉存取顺序，进而改变程序的重用距离和footprint值，如何减少分析工具对分析结果的影响也值得探索。另外，多线程运行程序的局部性分析的时间代价远远超过单线程情况下的开销，如何在保证精度的条件下提高分析效率还需要进行更加深入的研究。因此，很有必要在现有研究的基础上对多线程程序相关的各类算法进行扩展研究。

## 4 基于概率统计的局部性分析

根据数据访问特征，局部性通常被分为时间局部性和空间局部性两大类，然而一直以来却没有行之有效的方法分别对它们进行量化，局部性相关的计算也都是通过失效率等变量来衡量。基于概率统计的局部性度量[30]却成功地实现了对时间局部性和空间局部性的数学定义和计算。

**4.1 时间局部性和空间局部性**

基于条件概率的局部性度量，首先需要对所研究的事件进行明确的定义，然后针对时间局部性中的“最近未来一段时间”和空间局部性中的“临近位置”进行明确的数学定义。用*A*表示任意地址，*N*表示最近未来一段时间的范围或者数据未来访问序列窗口的大小，*X*表示访问的数据，则*{X=A}*表示访问数据地址为*A*的事件，时间局部性*TL*和空间局部性*SL*分别可以给出基于条件概率的定义，如公式(4)和(5)所示。

 (4)

 (5)

时间局部性中的“最近未来”根据不同的情况可以有三种定义：(1)地址流中的接下来访问的*N*个地址，(2)地址流中的接下来访问的*N*个不同的地址，(3)地址流中接下来访问的*N*个不同的cache行。空间局部性中的“临近位置”*N*分为基于模和基于块两种定义。不同的定义适用于不同精度情况下的不同需求。通过贝叶斯理论，两者的计算公式可分别表示为公式(6)和公式(7)的形式。

 (6)

 (7)

图6是计算时间局部性的的一个简单实例，对于地址序列*XYYXZ*而言，假设*n=*2，“最近未来”采用第一种定义方式，则根据联合概率计算公式可知*TL*(2)*=*1/4，计算过程如图6(b)所示。若采用第二种定

N=2时的联合概率：

地址序列：



*X Y Y X Z*

(a)the access trace (b)temporal locality using Def.1



时间局部性

最近未来时间窗口大小

(c)temporal locality using different definitions

图6 时间局部性的计算

Fig.6 Calculating temporal locality

义，因为最近未来窗口需要包含两个不同的地址，则*TL*(2)=2/4。若采用第三中定义，假设*X*，*Y*，*Z*均在同一个cache行中，则时间局部性*TL*(2)=1。不同的定义将会得到不同的计算结果，图6(c)是三种不同定义下时间局部性随未来窗口大小变化得到的曲线图。

由计算公式可以很容易通过代码分析对程序局部性进行计算。该方法简单易懂，提供了一个通用的局部性分析方法，可以针对存储层次系统的每一层进行局部性分析，便于我们对访存行为的理解和对存储层次设计的优化。通过以上分析可知当空间局部性中对空间的定义为单位1时，即为时间局部性，因此可以将时间局部性视为空间局部性的一种特殊情况进行处理，统一通过空间局部性衡量程序数据局部性分布的好坏。

**4.2 方法应用**

采用概率的局部性度量方法可以通过3D网格图的建立使程序局部性分布可视化[33]，为程序代码优化和存储层次结构优化设计提供了一种新的思路。以矩阵乘法为例，“最近未来”和“临近位置”分别采用接下来*n*个地址和基于模的定义，得到不同分块大小情况下的局部性分布图如图7所示。给定大小均为256\*256的矩阵*A*和*B*，图中分别是矩阵*A*和*B*相乘时采用不分块、分块大小为32\*32和分块大小为64\*16三种情况下的局部性概率分布，其中两个水平轴分别表示未来窗口大小和临近位置大小，竖轴表示局部性大小。从图中的局部性信息可以清晰的知道不同分块下各个位置对应局部性数据变化情况，利用这些信息可以进行最优分块因子的选择和代码优化。同时也可以通过跟踪不同cache的局部性变化情况进行存储层次结构的优化设计。



(a) untiled



(b) 32\*32 tile

(c) 64\*16 tile

图7 矩阵乘在不同分块情况下的局部性

Fig.7 Locality for different matrix multiplication kernels

从图7中可以看出，该方法不但针对特定cache配置得到计算结果，而且展示了随着不同的未来窗口和邻域值的变化所产生的局部性变化规律。然而，随着程序输入数据量的增大、循环次数的增加或者程序代码的复杂程度等原因，导致地址序列长度呈指数形式增长，计算数据量过大，进而导致局部性计算时间过长。本文对Polybench3.1中的程序地址序列长度进行了实验测试，实验结果如表2所示，结果表明除了jacobi-1D和jacobi-2D的地址序列数量级为千万级之

表2 Polybench3.1中程序访存地址序列长度

Table 2 Memory trace length of ploybench3.1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | polybench | n(108) | | 2mm | 1.952 | | 3mm | 1.952 | | adi | 1.954 | | atax | 1.952 | | bicg | 1.952 | | cholesky | 1.952 | | correlation | 1.961 | | covariance | 1.956 | | doitgen | 1.952 | | durbin | 1.952 | | dynprog | 1.969 | | fdtd-2d | 1.959 | | fdtd-apml | 1.961 | | gauss-filter | 1.953 | | gemm | 1.952 | | |  |  | | --- | --- | | polybench | n(108) | | gemver | 1.955 | | gesummv | 1.952 | | gramschmidt | 1.952 | | jacobi-1D | 0.272 | | jacobi-2D | 0.981 | | lu | 1.952 | | ludcmp | 1.952 | | mvt | 1.970 | | reg-detect | 1.980 | | seidel | 1.957 | | symm | 1.952 | | syr2k | 1.952 | | syrk | 1.952 | | trisolv | 1.955 | | trmm | 1.952 | |

外，其余程序均为上亿级。如果采用三层循环结构对局部性概率进行计算，整个计算花费逻辑时间将达到1013级。鉴于这种问题，可以采用地址抽样或者在GPU单元上进行并行计算的方法来减少时间代价。

文献[31]采用抽样的方法实现了一种简化了的概率计算方法，并且可以把这种基于概率的方法运用于文献[14,32]中的局部性分析当中。文献[33]采用三维坐标图使局部性可视化，首次提出时间局部性是空间局部性的一个特例，不同的是，该方法在将要访问窗口内计算的是概率平均值，和条件概率计算方法有根本上的区别。

**4.3 小结**

相比于之前的局部性计算方法，条件概率的计算方法为局部性提供了明确的数学定义，对时间和空间局部性进行了统一的量化。如果说重用距离直方图是事件{*S：*对同一数据两次连续的访问}的概率分布函数，那么上文中介绍的时间局部性就是对同一事件的卷积函数。相对于重用距离和footprint理论，基于概率统计的度量方法更直观，也更全面。由于整个局部性计算是针对整个访存地址空间进行建模，因此不适合实时性分析。再者，循环程序中的访存地址顺序以及每个元素的重用距离和重用次数可以根据循环中所访问数组的标号以及循环中的表达式计算得到，因此对于此类地址有规律可循的程序，是否能够据此直接在*O(1)*时间内采用基于概率的方法计算得到局部性也很值得研究，并且由此可以根据局部性大小选择得到最佳分块因子。

## 5 现代计算环境下的局部性分析

局部性不仅针对单核计算环境而言，随着并行化研究的深入，计算机多核架构的发展以及众核处理器的逐渐普及，传统的程序局部性分析已经不能满足现在的需要。多核/众核架构的多样性和程序并行运行的复杂性使现代计算机环境下局部性的测量变得更加复杂，仅仅依靠访存地址序列已经无法对局部性形成全面的认识，只有通过构建新的模型才能适用于目前的应用环境。

**5.1 多核架构的局部性分析**

在多线程环境下，由于资源共享，需要新的算法对多线程局部性进行分析。文献[70]针对cache共享提出了一个基于公平性的模型，但如果单个线程的IPC(Instructions per Clock-Cycle)能够快速提升时，结果不一定为最优。文献[71]针对多线程cache争用现象建立模型。文献[72]对基于地址序列的局部性理论进行多线程环境下的扩展，分析了多线程环境下每个线程重用距离变化情况。

在多核架构下，存在多核共享cache的情况，局部性分析不再局限于核内，还应该对核间数据重用进行研究。文献[73]的研究表明充分发掘核间数据重用将平均提高程序性能21.3%左右。文中针对多核中共享缓存带来的利益和多线程应用进行研究，并对核间数据重用进行了定义。文中指出因为核间重用距离通常大于核内重用距离，因此如果只考虑核间重用而不考虑核内重用将会导致比只考虑核内重用更加糟糕的运行效果。鉴于以上原因，作者需要在核内重用和核间重用之间选择平衡策略来对程序进行优化。为了在数据块依赖关系许可的情况下使两种情况的数据重用最大化，作者提出了一种数据调度算法，即通过构建共享数据图和计算关系依赖图，给不同的调度单元分配权值，从而有效提高系统的整体性能。但调度算法的实现较为复杂，若选取更加简便的算法对性能的提高会有更大帮助。

重用距离是衡量局部性的常用技术，然而由于多线程的影响和程序的并行执行导致重用距离不再具有硬件无关性，之前的很多计算方法不再适用。文献[74]基于CMP架构，对传统的重用距离概念进行扩展，首次提出并发重用距离的概念，并采用概率模型把并发重用距离和每个线程的数据局部性联系起来，使重用距离的概念能够适用于多核架构。多核架构下的局部性度量不仅要考虑数据重用，还要考虑线程间通信。文献[17]中针对多核情况下的重用距离进行分析，采用抽样的方式建立一个快速模型，通过哈希表使每次访问的代价减为*O(*1*)*，在保证计算高精度的前提下大幅度提高了重用距离分析速度。文献[75]将Wolf等人提出的串行数据重用理论扩充到并行领域，分别提出了面向OpenMP和OpenTM应用的并行数据重用理论，针对重用在线程、事务中的关系，系统地讨论了并行应用中重用的分类、判定和求解方法。文献[76]对footprint理论进行扩展，提出共享footprint和共享率的概念以适应多线程中数据共享的情况，为以后数据共享问题相关局部性度量的研究奠定基础。重用距离作为衡量局部性的常用技术，在多核/众核情况下，如何快速和精确的计算重用距离仍然是局部性度量研究的热点。

**5.2 GPU架构的局部性分析**

尽管GPU依靠大规模线程的并发度来掩盖访存造成的长时间延迟问题，但对通用GPU应用来说，对相应存储系统的分析仍然有着重要意义。目前的GPU存储架构中，每个流多处理器(Stream Multiprocessor, SM)都有一个L1 cache，多个SM共享一个L2 cache。由于每个L1 cache被所有在该SM上的线程块block所共享，每个block又被分割为更小的线程束warp，因此可以分别在warp级、thread-block级和SM级进行程序局部性



(a) Warp level



(b) Thread-block level



(c) SM level

图8 矩阵乘在warp, thread-block, SM级下局部性

Fig.8 Locality of matrix multiplication kernel at warp, thread-block level, SM level

分析，加深对L1高速缓存的理解。L2级缓存局部性则可以依旧通过数据访问流来计算。

文献[77]对矩阵乘法分别进行warp级、thread-block级和SM级局部性分析，采用条件概率的方法[30]进行局部性计算，得到局部性3D网格图如图8所示，并用*LS(N,K)*表示空间局部性，其中*N*表示未来窗口大小，*K*表示临近位置大小。在warp级，时间局部性为*LS(N,0)*，*LS*随着*N*值的变化而变化。如果矩阵分块大小为16\*16，则对于矩阵*A*乘*B*，在warp中，由于*A*[0][0]将会被*B*[0][0~15]共16个元素重复访问，因此

时间局部性在*LS(*1,0*)*迅速增长。16次访问之后*B*矩阵中元素开始出现重用，在*LS(*16,0*)*处局部性再次得到提高。当*K*>0时，由于一直连续访问矩阵*B*中同一行元素，因此空间局部性为1。

在thread-block级，如果每个线程块被分为8个线程束的话，不同线程束的访问顺序会出现交叉，如图9所示。同warp级一样，由于矩阵*A*中元素的重用，在*LS(*1,0*)*时局部性也得到同样的提高。不同的是，*LS(*16,0*)*的局部性要高于warp级的值，原因在于多个warp同时执行时，矩阵*B*将会有更多的行被同时访问。

在SM级，如果每个SM有4个block被调度执行，则由于多个block共享L1 cache，将会引起数据访问冲突。如果cache最多只能容纳一个block的数据，多个block被调度时，将会导致颠簸，使局部性

A[0][0]\*B[0][0~15] - warp0

A[1][0]\*B[0][0~15] - warp0

A[2][0]\*B[0][0~15] - warp1

...

A[14][0]\*B[0][0~15] - warp7

A[15][0]\*B[0][0~15] - warp7

------------------------------------

A[0][1]\*B[1][0~15] - warp0

A[1][1]\*B[1][0~15] - warp0

...

A[15][1]\*B[1][0~15] - warp7

------------------------------------

...

------------------------------------

A[0][2]\*B[2][0~15] - warp0

...

A[0][15]\*B[15][0~15] - warp7

图9 矩阵在warp级下的访存顺序

Fig.9 Access trace of matrix at wrap level

下降。然而对比图8(b)和8(c)，我们可以看到block级的局部性仅部分略高于SM级的局部性，这是因为矩阵A每个分块只需要16个缓存块，它们各自对应的矩阵B分块所属的每个线程块调度只需要1个缓存块，cache资源需求过小并不会引起局部性明显变化。

文中仅对一个在GPU上运行的矩阵乘法进行分析，对于那些依赖关系和控制流等都更加复杂的程序并未提供详细的说明，对于更大规模的数据传输程序的局部性计算方法也未说明，如何对通用程序进行有效的局部性计算将是GPU架构相关局部性分析的难点。

文献[78]针对如何准确量化反映应用程序不同线程之间数据依赖关系的数据相关性问题，提出了一种基于数据重用距离的线程间数据相关性度量方法DRAQ(Data Reuse Distance based Data Affinity Quantifying)。通过剖分不同线程存储访问数据信息，分析不同线程之间的数据局部性及存储访问特征，设计线程数据重用距离计算方法，计算不同线程的数据重用距离；在数据重用距离的基础上，通过定义反映不同数据相关性的局部性模式，将不同线程归并为不同的模式类；在不同模式类的基础上通过比较不同线程间数据共享量，用相关性矩阵来度量不同线程间的数据相关性，并将线程相关性矩阵转换成相应的相关性图，来直观的反映不同应用线程之间的数据相关性。通过实验评测，DRAQ相对于默认映射方法，Cache缺失率平均降低了10.4%，达到最优映射下Cache缺失率的87%，引入的平均开销为8%。

**5.3 神威众核处理器局部性分析**

神威众核处理器采用片上融合的异构体系结构，如图10，由4个核组成，每个核组包括1个主核MPE（management processing element）和64个从核CPEs（computing processing element）。主核拥有2级缓存：第1级缓存指令数据分离，即指令和数据各自拥有32KB缓存；第2级则为指令数据共享的256KB缓存。而每个从核拥有16KB的一级指令缓存，从核数据存储的设计为一种类CELL处理器的用户控制的草稿本方式存储，即SPM（scratch pad memory），每个从核的SPM容量为64KB。这种存储方式省去了缓存实现上的控制开销，也避免了众多运算核心之间一致性带来的设计复杂性和性能降级。但不同于传统缓存结构，开发中从核的数据管理面临新的挑战，需要人为对应用在SPM上的访存进行设计与规划[79]。从核要访问数据只能通过：用户控制的SPM、直接内存访问通道（DMA）和寄存器数据通信机制实现。

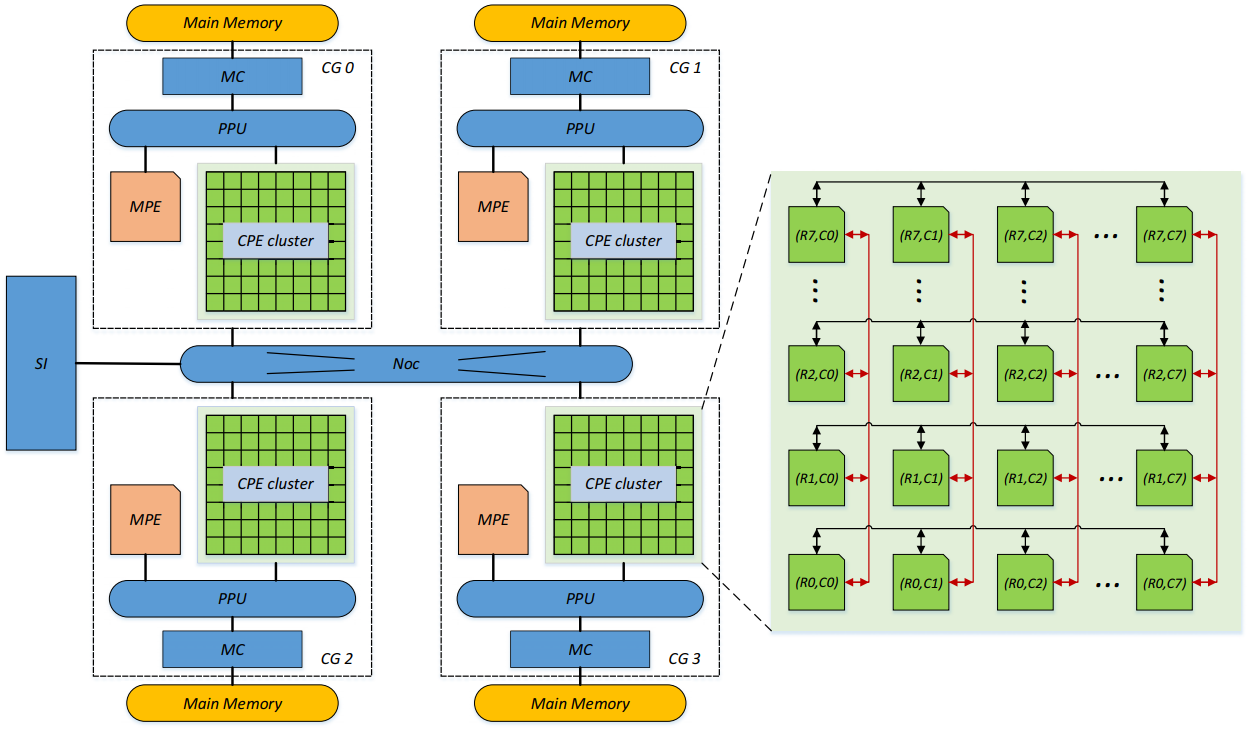


图10 神威众核处理器架构

Fig.10 The architecture of the Sunway TaihuLight multiprocessor

数据提前拷贝至访问延迟低的SPM局部存储中。而如何选择合适的数据提前进行拷贝，就需要考虑到数据的局部性，尽量选择局部性高的数据放入SPM。文献[80]提出了一种将所有在计算中涉及的数据优先传送至SPM的实现策略，这样可以提高从核访问数据的速率，从而提高计算性能。但文中并没有给出数据大于SPM情况下的解决方案，同时策略可能将同一数组拷贝到多个从核的SPM中，这样会造成存储资源的浪费。

文献[81]提出了一种针对科学和应用中广泛存在的Stencil计算在现代超级计算机上优化的方案。文章通过：计算通信重叠、局部性感知线程分块和协同数据访问三方面进行性能优化。特别是局部性感知线程分块策略，利用循环阵列和相应的双缓存方法，提高了数据重用并隐藏了内存延迟。如图11，左边矩形方块代表stencil总计算数据，而当前计算数据为虚线标识的一小块，当前数据的计算依赖于包括自身在内5块数据，如图中step1所示。在当前数据块计算完成后计算下一块时，下一块依赖的五块数据已有4块已在缓存中，只需取一块数据即可，如图中step2。同时如果配合数据预取技术，在当前块计算时，便可以预取下一块所需依赖块，大大降低了缓存的失效率。

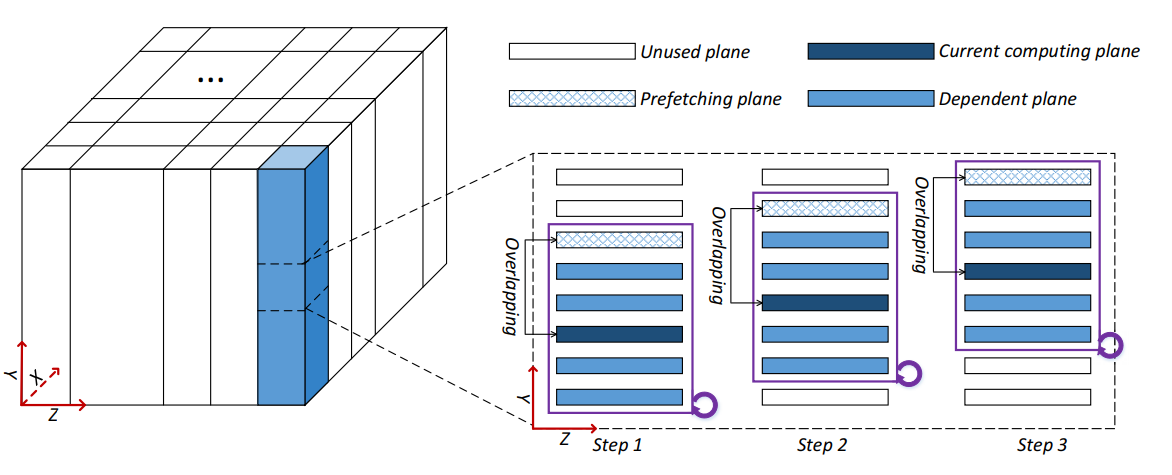


图11 局部性感知线程分块策略

Fig.11 The locality-aware thread blocking strategy

**5.4 局部性分析中的大数据问题**

随着社交媒体和云计算的兴起，从大数据方面考虑性能分析也成为一种趋势[82]。从第3小结中可知footprint计算的时间窗口个数为*n\*(n+1)/2*，即在主频为3G Hz的处理器上运行一个程序，如果运行时间为3E10个CPU周期，则会有4.5E20个不同的窗口。图12展示了时间窗口随执行时间的增加形成的变化曲线，执行时间从一秒增加到一个月时，CPU周期将会从3E9增加到2E15，不同的时间窗口个数从4.5E18增加到5.8E29。为了对此有一个更直观的认识，忽略单位的严格意义，4.8E22 cm相当于银河系的半径，4.4E28 cm相当于可观测宇宙的半径。由此可知，局部性度量中的数值计算是一个大数据处理问题。

由于局部性相关的研究已经取得了很多进展，因此可以把一些局部性理论的计算模型应用在除计算机芯片之外的场景，即任何可以被表示为离散数据流的数据集上，例如：web搜索条目、网站信息、社交通信以及在线商品销售数据等等。在线商品销售中顾客浏览和购买同一个商品的时间间隔相当于数据重用距离，通过对购买时间间隔的统计，得到相应footprint曲线或者重用距离分布图，就可以识别不同商品的销售情况，进而了解商品消费趋势。零售商根据消费趋势进行采购避免货物积压或者短缺，就如同cache中采用数据预取策略减少cache失效率、提高程序运行速度一样。多线程或者多核调度方面的局部性分析可以被用来进行交通管制，进而减少交通拥堵。在大数据的处理上，机器学习技术已经被广泛应用于以上提到的各个领域中，相信局部性度量相关技术的扩展也将会成为一次不容错失的机会，它将在社



时间窗口个数

执行时间(CPU主频为3GHz)

s min hour day

可观测宇宙半径：

4.4E28 cm

银河系半径：

4.8E22 cm

图12 不同执行时间下footprint计算问题规模的变化

Fig.12 Scale of the problem shown by the number of footprint windows in a program execution

会、多媒体和市场数据分析等看似不相关的各个领域发挥巨大的作用。

**5.5 小结**

本节针对多核架构和GPU架构上的局部性分析进行了介绍，并指出了局部性理论可能的发展前景。多核架构下的局部性分析可以分为两类：一类是不同程序并行运行，不存在数据共享，但由于最后一级cache(Last Level Cache, LLC)共享会引起cache冲突失效；另一类是程序并行执行，存在数据共享和复杂的依赖关系。前者可以通过footprint理论进行局部性分析，后者还没有合适的解决方案，需要新的概念和新的理论来支持。GPU架构下的局部性分析的相关研究并不多，一方面因为GPU架构下程序本身的运行速度就已经很高，另一方面因为分析比较复杂，要想进一步提高存储资源利用率就需要更加成熟的局部性理论支持。随着局部性理论的日益发展，局部性度量在其他领域的应用也值得关注。

## 6 结论和展望

局部性是计算机系统设计的一个理论基础，不仅是一种性能评价标准，也是一种优化方法，对算法设计、编译分析、操作系统和计算机系统结构改进等都具有指导作用。本文对不同的局部性度量模型进行了分析，包括各个度量变量的定义和度量方法，尤其是对基于重用距离相关的计算、footprint局部性理论以及基于概率统计的局部性度量方法进行了详细介绍，并分别分析了它们的优缺点以及潜在发展方向。

采用重用距离和footprint理论衡量程序局部性的方法简单有效，因此得到广泛研究。根据研究中针对的代表性问题和相应解决方案，可以把相关局部性分析的研究归纳为五个方面：程序输入，代码，数据，时间和运行环境[16]。程序输入将会影响整个程序的局部性，重用距离可以被用来针对由cache大小、程序输入[32]、程序段[83]等引起的cache失效率进行预测[84-86]。对程序代码进行局部性分析，可以把指令分为高局部性指令和低局部性指令，从而对cache替换策略进行调整[87,19]；也可以只针对高局部性代码使用代码转换、训练分析[86]和编译分析[18]等方法或者采用它们的组合方法[85]进行局部性分析。对于程序数据而言，同一数据集合访问越密集局部性越好[88]，因此数据替换策略的选择一直是局部性优化的难题[89]，数据粒度大小的划分也将直接影响到数据访问的命中次数和替换次数[90-92]。对于cache和内存访问，如果可以通过重用距离分布曲线对不久将来数据访问的时间进行预测，就可以实现数据预取，并应用于内存管理[93-95]、文件缓存[96,97]和服务系统响应时间优化[98]等各个方面。程序在配置参数不同的环境下运行，局部性也会有所变化，例如多个程序同时运行时，由于共享LLC，程序间会出现cache竞争，进而影响彼此的访存命中率。

随着计算机科技的日新月异，多核架构以及众核架构的出现使得局部性相关度量面临更大的挑战，尽管关于局部性的研究成果众多，但我们依然面临着一些困难。首先，针对访存地址进行分析的局部性模型计算代价仍旧很大，一个计算代价很大的算法很难实时有效地用来进行程序的优化和性能的提高，因此在局部性计算模型上仍然有改进的空间。其次，随着多核处理器的发展，并行应用将会越来越普及，工作负载也将会过度依赖LLC，共享cache下的访问不命中将会导致一个程序的性能受到其他并行程序的严重影响，数据共享时间和共享数据的多少将直接决定最优的LLC分配策略，这就需要在多级Cache下针对不同层Cache的局部性变化进行更加细致的分析和建模，在充分利用一级Cache的同时也提高最后一级Cache的利用率。另外，同一程序下的多线程间存在着数据共享的问题，在数据共享情况下如何衡量多线程局部性就需要提出新的概念和理论。同时，针对神威太湖之光这类非传统存储架构，如何合理利用局部性进行性能优化，是一个值得研究的方向。最后，由于现实生活中问题的相似性，如何利用Cache局部性理论解决其他领域中的类似问题也存在着很大的探索空间。针对上述问题，在现有理论的基础上，应该结合新的存储结构和程序特点，全面精准地分析多核架构上的数据访存行为，提出更加有效简洁的理论和新的算法来适应更复杂的数据访存情况，追求更加精确的程序局部性计算结果，更好的了解程序运行状态，从而进行程序优化。在扩展局部性理论的同时，把云计算和大数据等计算需求应用到局部性分析中来也是一个较好的发展思路。

**参 考 文 献**

1. Kilburn T, Edwards D B G, Lanigan M J, Sumner F H. One-level storage system [J]. IRE Transactions, 1962, EC-11(2): 223-235
2. Wilkes M V. Slave memories and dynamic storage allocation [J]. IEEE Transactions on Computers, 1965, EC-14(2): 270-271
3. Denning P J. The working set model for program behaviour [J]. Communications of ACM, 1968, 11(5): 323-333
4. Denning P J. Virtual memory [J]. ACM Computing Surveys, 1996, 28(4): 213-216
5. Denning P J. Working sets past and present [J]. IEEE Transactions on Software Engineering, 1980, 6(1): 64-84
6. Denning P J, Schwartz S C. Properties of the working set model [J]. Communications of ACM, 1972, 15(3):191-198
7. Denning P J, Slutz D R. Generalized working sets for segment reference strings [J]. Communications of ACM, 1978, 21(9): 750-759
8. Denning P J. The locality principle [J]. Communications of the ACM, 2005, 48 (7): 19-24
9. Xiang X, Ding C, Luo H, Bao B. HOTL: A higher order theory of locality [C] //Proc of the Architectural Support for Programming Languages and Operating Systems(ASPLOS). Houston, USA, 2013: 343-356
10. Hill M D, Smith A J. Evaluating associativity in CPU caches [J]. IEEE Transactions on Computers, 1989, 38(12): 1612-1630
11. Mattson R L, Gecsei J, Slutz D, Traiger I L. Evaluation techniques for storage hierarchies [J]. IBM System Journal, 1970, 9(2): 78-117
12. Yang T, Berger E D,Kaplan S F,Moss J E B. CRAMM: Virtual memory support for garbage-collected applications [C] //Proc of the 7th symposium on Operating System Design and Implementation(OSDI). Berkeley, USA, 2006: 103-116
13. Zhou P, Pandey V, Sundaresan J, Raghuraman A, Zhou Y, Kumar S. Dynamic tracking of page miss ratio curve for memory management [C] //Proc of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS). New York, USA, 2004: 177-188
14. Ding C, Zhong Y. Predicting whole-program locality with reuse distance analysis [C] //Proc of the 2003 ACM SIGPLAN Conference on Programming Language Design and Implementation ( PLDI). San Diego, USA, 2003: 245-257
15. Almasi G, Cascaval C, Padua D A. Calculating stack distances efficiently [C] //Proc of the 2002 workshop on Memory system performance. New York, USA, 2002: 37-43
16. Zhong Y, Shen X, Ding C. Program locality analysis using reuse distance [J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 2009, 31 (6): 1-39
17. Schuff D L, Kulkarni M, Pai V S. Accelerating multicore reuse distance analysis with sampling and parallelization [C] //Proc of the 19th Parallel Architectures and Compilation Techniques. Vienna, Austria, 2010: 53-64
18. Cascaval C, Padua D A. Estimating cache misses and locality using stack distances [C] //Proc of the 17th Annual International Conference on Supercomputing. New York, USA, 2003: 150-159
19. Belys K, D'Hollander E. Generating cache hints for improved program efficiency [J]. System Architecture, 2005, 51(4): 223-250
20. Kelly W, Maslov V, Pugh W, Rosser E, Shpeisman T, Wonnacott D. The Omega Library interface guide. Maryland: Department of Computer Science, University of Maryland, College Park, Technical Report, 1996
21. Xue J, Vera X. Efficient and accurate analytical modeling of whole-program data cache behavior [J]. IEEE Transactions on Computers, 2004, 53(5): 547-566
22. Olken F. Efficient methods for calculating the success function of fixed space replacement policies [R]. Berkeley: Lawrence Berkeley Laboratory, Technical Report: W-7405-ENG-48, 1981
23. Sugumar R, Abraham S. Multi-configuration simulation algorithms for the evaluation of computer architecture designs [D]. Michigan: University of Michigan, 1993
24. Bennett B, Kruskal V J. LRU stack processing [J]. IBM Journal of Research and Development, 1975, 19(4), 353-357
25. Burger D, Austin T. The simplescalar tool set, version 2.0 [R]. Wisconsin: Department of Computer Science, University of Wisconsin, Technical Report: CS-TR-97-1342, 1997
26. Almasi G, Cascaval C, Padua D A. Calculating stack distances efficiently [J]. SIGPLAN Notices, 2003, 38(2): 37-43
27. Fang C, Carr S, Önder S, Wang Z. Reuse-distance-based miss-rate prediction on a per instruction basis [C] //Proc of the 2004 workshop on Memory system performance. Washington, DC, USA, 2004
28. Sleator D D, Tarjan R E. Self adjusting binary search trees [J]. Journal of the ACM, 1985, 32(3): 652-686
29. Cui H, Yi Q, Xue J, Wang L, Yang Y, Feng X. A highly parallel reuse distance analysis algorithm on GPUs [C] //Proc of the 26th Parallel & Distributed Processing Symposium(IPDPS). Shanghai, China, 2012: 1080-1092
30. Gupta S, Xiang P, Yang Y, Zhou H. Locality principle revisited: A probability-based quantitative approach [C] //Proc of the 26th Parallel & Distributed Processing Symposium(IPDPS). Shanghai, China, 2012: 995-1009
31. Berg E, Hagersten E. Statcache: A probabilistic approach to efficient and accurate data locality analysis [J]. IEEE International Symposium on Performance Analysis of Systems and Software(ISPASS), 2004: 20-27
32. Zhong Y, Dropsho S G, Shen X, Studer A, Ding C. Miss rate prediction across program inputs and cache configurations [J]. IEEE Transactions on Computers, 2007, 56(3): 328-343
33. Grimsrud K, Archibald J. Locality as a visualization tool [J]. IEEE Transactions on Computers, 1996, 45(11): 1319-1326
34. Golub G H, Van Loan C F. Matrix computations, 4th edition. Baltimore, Maryland, USA: Johns Hopkins University Press, 1989
35. Wolfe M J. More iteration space tiling [C] //Proc of the 1989 ACM/IEEE conference on Supercomputing. New York, USA, 1989: 655-664
36. McKinley K S, Carr S, Tseng C. Improving data locality with loop transformations [J]. ACM Transactions on Programming Languages and Systems, 1996, 18(4):424-453
37. Pouchet L, Bondhugula U, Bastoul C, et al. Loop transformations: Convexity, pruning and optimization [J]. ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'11), 2011, 38: 549-562
38. Kennedy K, McKinley K S. Optimizing for parallelism and data locality [C] //Proc of the 1992 ACM International Conference on Supercomputing. Washington, DC, USA, 1992: 151-162
39. Wolf M E, Lam M. A data locality optimizing algorithm [C] //Proc of the SIGPLAN '91 Conference on Programming Language Design and Implementation. Toronto, Canada, 1991: 30-44
40. Allen R, Kennedy K. Optimizing compilers for modern architectures: a dependence-based approach. San Francisco, CA: Morgan Kaufmann Publishers, 2002
41. Chauhan A, Shei C Y. Static reuse distances for locality-based optimizations in Matlab [C] //Proc of the 24th ACM International Conference on Supercomputing(ICS). New York, USA, 2010: 295-304
42. Jaleel A, Borch E, Bhandaru M, Jr S C S, Emer J S. Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (TLA) cache management policies [C] //Proc of the 2010 43rd Annual IEEE [C] //ACM International Symposium on Microarchitecture(MICRO). Washington, DC, USA, 2010: 151-162
43. Jiang Y, Tian K, Shen X. Combining locality analysis with online proactive job co-scheduling in chip multiprocessors [C] //Proc of the [High Performance Embedded Architectures and Compilers](http://link.springer.com/book/10.1007/978-3-642-11515-8) (HiPEAC). Pisa, Italy, 2010: 201-215
44. Jiang Y, Tian K, Shen X, Zhang J,Chen J, Tripathi R. The complexity of optimal job co-scheduling on chip multi-processors and heuristics-based solutions [J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 22(7): 1192-1205
45. Tang L, Mars J, Soffa M L. Compiling for niceness: Mitigating contention for QoS in warehouse scale computers [C] //Proc of the 10th International Symposium on Code Generation and Optimization(CGO). New York, USA, 2012: 1-12
46. Bao B, Ding C. Defensive loop tiling for shared Cache [C] //Proc of the International Symposium on Code Generation and Optimization(CGO). Shenzhen, China, 2013: 23-27
47. Beyls K, D’Hollander E. Reuse distance as a metric for cache behavior [C] //Proc of the IASTED Conference on Parallel and Distributed Computing and Systems. Ghent, Belgium, 2001: 617-662
48. Shen X, Zhong Y, Ding C. Locality phase prediction [C] //Proc of the Architectural Support For Programming Languages And Operating Systems. New York, USA, 2004:165-176
49. Zhong Y, Dropsho S, Ding C. Miss rate prediction across all program inputs [C] //Proc of the 12th International Conference on Parallel Architectures and Compilation Techniques. New Orleans La, USA, 2003: 91-101
50. Xue J, Huang C. Reuse-driven tiling for improving data locality. International Journal of Parallel Programming, 1998, 26(6): 671-696
51. Xue J. Loop tiling for parallelism. Boston: Kluwer Academic Publishers, 2000
52. Marin G, Mellor-Crummey J. Pinpointing and exploiting opportunities for enhancing data reuse [C] //Proc of the IEEE International Symposium on [Performance Analysis of Systems and software(ISPASS).](http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4498398) Austin, TX, USA, 2008: 20-22
53. Chen Ding, Xiaoya Xiang, Bin Bao, Hao Luo, et al. Performance metrics and models for shared cache [J]. Journal of Computer Science and Technology, 2014, 29(4): 692-712
54. Wu M, Yeung D. Coherent profiles: Enabling efficient reuse distance analysis of multicore scaling for loop-based parallel programs [C] //Proc of the Parallel Architectures and Compilation Techniques(PACT). Galveston, TX, USA, 2011: 264-275
55. Wu M, Zhao M, Yeung D. Studying multicore processor scaling via reuse distance analysis [C] //Proc of the 40th Annual International Symposium on Computer Architecture(ISCA). New York, USA, 2013: 499-510
56. Xiang X. A higher order theory of locality and its application in multicore cache management [D]. Rochester: Computer Science Department, University of Rochester, 2014
57. Uhlig R A, Mudge T N. Trace-driven memory simulation: A survey [J]. ACM Computing Surveys, 1997, 29(2): 128-170
58. Magnusson P, Werner B. Efficient memory simulation in SimICS [C] //Proc of the 28th Annual Simulation Symposium. Phoenix, AZ, USA, 1995: 62-73
59. Rosenblum M, Herrod S A, Witchel E, Gupta A. Complete computer system simulation: The SimOS approach [J]. IEEE Parallel & Distributed Technology, 1995, 3(4): 34-43
60. Zagha M, Larson B, Turner S, Itzkowitz M. Performance analysis using the MIPS R10000 performance counters [C] //Proc of the 1996 ACM/IEEE conference on Supercomputing. Washington, DC, USA, 1996: Article No. 16
61. Kwak H, Lee B, Hurson A R, Yoon S H, Hahn W J. Effects of multithreading on cache performance [J]. IEEE Transactions on Computers, 1999: 48(2)
62. Mogul J C, Borg A. The effect of context switches on cache performance [C] //Proc of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA, 1991: 75-84
63. Thiébaut D, Stone H S. Footprints in the cache [J]. ACM Transactions on Computer Systems, 1987, 5(4): 305-329
64. Suh G E, Devadas S, Rudolph L. Analytical cache models with applications to cache partitioning [C] //Proc of the ACM International Conference on Supercomputing 25th Anniversary Volume. New York, USA, 2001: 1-12
65. Ding C, Chilimbi T. All-window profiling of concurrent executions [C] //Proc of [the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)](https://www.softconf.com/starts/ppopp08/login/). Salt Lake City, USA, 2008: 265-266
66. Xiang X, Bao B, Bai T, Ding C, Chilimbi T M. All-window profiling and composable models of cache sharing [C] //Proc of [the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)](https://www.softconf.com/starts/ppopp08/login/). New York, USA, 2011: 91-102
67. Xiang X, Bao B, Ding C, Gao Y. Linear-time modeling of program working set in shared cache [C] //Proc of Parallel Architecture and Compilation Techniques(PACT). Galveston, USA, 2011: 350-360.
68. Chen T Y, Wei H W, Wei M F, et al. LaSA: A locality-aware scheduling algorithm for Hadoop-MapReduce resource assignment[C]// International Conference on Collaboration Technologies and Systems. IEEE, 2013:342-346.
69. Liu S, Xie X, Cui Y, et al. An Efficient Locality-Aware Task Assignment Algorithm for Minimizing Shared Cache Contention[C]// International Conference on Parallel and Distributed Computing, Applications and Technologies. IEEE Computer Society, 2017:44-51.
70. Kim S, Chandra D, Solihin Y. Fair cache sharing and partitioning in a chip multiprocessor architecture [C] //Proc of Parallel Architecture and Compilation Techniques(PACT). Antibes, French, 2004: 111-122
71. Chandra D, Guo F, Kim S, Solihin Y. Predicting inter-thread cache contention on a chip multi-processor architecture [C] //Proc of HPCA. 2005: 340-351
72. Ding C, Chilimbi T. A composable model for analyzing locality of multi-threaded programs [R]. Redmond: Microsoft Research Redmond, Technical Report: MSR-TR-2009-107, 2009
73. Zhang Y, Kandemir M, Yemliha T. Studying inter-core data reuse in multicores [C] //Proc of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems. New York, USA, 2011: 25-36
74. Jiang Y, Zhang E Z, Tian K, Shen X. Is reuse distance applicable to data locality analysis on chip multiprocessors? [C] //[Rajiv Gupta](http://link.springer.com/search?facet-author=). LNCS 6011. Springer-Verlag Berlin Heidelberg, 2010: 264-282
75. 吴俊杰, 杨学军, 刘光辉,等. 面向OpenMP和OpenTM应用的并行数据重用理论[J]. 软件学报, 2010, 21(12):3011-3028.
76. Luo H, Xiang X, Ding C. Characterizing active data sharing in threaded applications using shared footprint [C] //Proc of the 11th International Workshop on Dynamic Analysis. Houston, USA, 2013:
77. Gupta S, Xiang P, Zhou H. Analyzing locality of memory references in GPU architectures [C] //Proc of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness. New York, USA, 2013: Article 12
78. 巨涛, 张兴军, 陈衡,董小社.面向众核系统的线程分组映射方法.西安交通大学学报, 2016 ,50(10), 57-63.
79. 郑方, 张昆, 邬贵明,等. 面向高性能计算的众核处理器结构级高能效技术[J]. 计算机学报, 2014, 37(10):2176-2186.
80. Wang Yichao, Lin Xinhua, Cai Linjin, Tang William, Ethier Stephane, Wang Bei, See Simon, Satoshi Matsuoka. Porting and Optimizing GTC-P on TaihuLight Supercomputer with OpenACC. Journal of Computer Research and Development, 2018, 55(4): 875-884.
81. Ao Y, Yang C, Wang X, et al. 26 PFLOPS Stencil Computations for Atmospheric Modeling on Sunway TaihuLight[C]// Parallel and Distributed Processing Symposium. IEEE, 2017.
82. Brock J, Luo H, Ding C. Locality analysis: a nonillion time window problem [J]. ACM SIGMETRICS Performance Evaluation Review, 2014, 41(4): 102-105
83. Shen X, Zhong Y, Ding C. Phase-based miss rate prediction [C] //Proc of the 17th international conference on Languages and Compilers for High Performance Computing(LCPC). Berlin, Germany, 2004: 42-55
84. Fang C, Carr S, Onder S, Wang Z. Instruction-based memory distance analysis and its application to optimization [C] //Proc of the International Conference on Parallel Architecture and Compilation Techniques. Saint Louis, USA, 2005: 27-37
85. Marin G, Mellor-crummey J. Cross architecture performance predictions for scientific applications using parameterized models [C] //Proc of the Joint International Conference on Measurement and Modeling of Computer Systems. New York, USA, 2004: 2-13
86. Rothberg E, Singh J P, Gupta A. Working sets, cache sizes, and node granularity issues for large-scale multiprocessors [C] //Proc of the 20th Annual International Symposium on Computer Architecture. New York, USA, 1993: 14-26
87. Belys K, D'Hollander E. Reuse distance-based cache hint selection [C] //Proc of the 8th International Euro-Par Conference. Paderborn, Germany, 2002: 266-275
88. Zhong Y, Orlovich M, Shen X, Ding C. Array regrouping and structure splitting using whole-program reference affinity [C] //Proc of the ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, USA, 2004: 255-266
89. Petrank E, Rawitz D. The hardness of cache conscious data placement [C] //Proc of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York, USA, 2002: 101-112
90. Berg E, Hagersten E. Fast data-locality profiling of native execution [C] //Proc of the International Conference on Measurement and Modeling of Computer Systems. Alberta, Canada, 2005: 169-180
91. Bunt R B, Murphy J M. The measurement of locality and the behaviour of programs [J]. The Computer Journal, 1984, 27(3): 238-253
92. Gu X, Christopher I, Bai T, Zhang C, Ding C. A component model of spatial locality [C] //Proc of the International Symposium on Memory Management. New York, USA, 2009: 99-108
93. Smaragdakis Y, Kaplan S, Wilson P. The EELRU adaptive replacement algorithm [J]. Performance Evaluation, 2003, 53(2): 93-123
94. Chen F, Jiang S, Zhang X. CLOCK-Pro: An effective improvement of the CLOCK replacement [C] //Proc of the Annual Conference on USENIX Annual Technical Conference. Berkeley, USA, 2005: 35-35
95. Zhou P, Pandey V, Sundaresan J, Raghuraman A, Zhou Y, Kumar S. Dynamic tracking of page miss ratio curve for memory management [C] //Proc of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA, 2004: 177-188
96. Zhou Y, Chen P M, Li K. The multi-queue replacement algorithm for second-level buffer caches [C] //Proc of the General Track: 2001 USENIX Annual Technical Conference. Berkeley, USA, 2001: 91-104
97. Jiang S, Zhang X. LIRS: An efficient low inter-reference recency set replacement to improve buffer cache performance [C] //Proc of the International Conference on Measurement and Modeling of Computer Systems. Los Angeles, USA, 2002: 31-42
98. Kelly T, Cohen I, Goldszmidt M, Keeton K. Inducing models of black-box storage arrays [J]. California: HP Laboratories Palo Alto, Technical Report: HPL-2004-108, 2004