帮助线程预取性能的分析与优化

黄　艳[1,2]，古志民[1]

(1. 北京理工大学计算机学院　北京 海淀区　100081； 2. 郑州轻工业学院软件学院　郑州　450002)

【摘要】针对访存延迟对现代处理器性能的影响，基于片上多处理器分析与测试了访存密集型应用程序的帮助线程数据预取性能。结果表明热点区计算/访存延迟比率对帮助线程预取性能有重大影响。依据热点区计算/访存延迟比率合理安排帮助线程与主线程的访存任务比例时，能达到对帮助线程性能的优化，使帮助线程预取获得更好的性能收益。基准测试程序的测试实验结果表明当热点区计算量很小可以忽略不计时，帮助线程与主线程的访存任务比接近1时，帮助线程预取获得最好的性能收益。

关　键　词　片上多处理器；　计算/访存延迟比率；　热点区；　性能分析；　预取线程

中图分类号　TP302　　　**文献标识码**　A　　　doi:10.3969/j.issn.1001-0548.2012.01.017

Performance Analysis and Optimization of Prefetching Thread on CMPs

HUANG Yan[1,2] and GU Zhi-min[1]

(1. School of Computer Science and Technology, Beijing Institute of Technology　Haidian Beijing　100081;

2. Software Engineering College, Zhengzhou University of Light Industry　Zhengzhou　450002)

Abstract　Memory latency has become a critical bottleneck in achieving high performance on modern processors. Prefetching thread based on multiprocessor (CMP) is a well known approach to reducing memory latency and has been explored in different applications. In this paper, we analyze the performance of prefetching thread for memory intensive applications. The analysis and experimental result show that computation/access latency ratio (CALR) of hotspots has an important affect on prefethcing performance. When the memory access ratio between main thread and prefetching thread is close to (1−CALR)/(1+CALR), prefetching thread gains better performance. The thread prefetching performance of several benchmarks from Olden and SPEC2006 benchmark suite is tested, and the experimental results reflect the impact of different memory access ratio between Prefetching thread and main thread.

Key words　CMP;　computation/access latency ratio (CALR);　hotspot;　performance analysis; prefetching thread

一直以来，访存延迟始终是应用程序获得高性能的主要瓶颈。随着计算机软件行业的发展，应用程序的结构越来越复杂，访问的数据集越来越大，访存延迟对性能的影响也越来越大，对访存密集型的大型应用程序更为显著。目前，数据预取作为减少应用程序访存延迟的主要方法之一，已经取得了显著的成果。尤其是近年来，随着计算机体系结构的发展，出现了新的处理器结构——片上多处理器，通过在单一芯片上集成多个处理器核，为软件数据预取带来了新的机遇。文献[1-3]基于片上多处理器上的共享L2 CACHE，利用片上多处理器上的空闲核执行帮助线程数据预取，并取得了一定的成果。然而，因为帮助线程的预取时效性难以得到保证，数据预取的效果并不理想。一种情况是处理器发出的预取请求到达存储器的时间太晚，预取的数据不能在处理器需要之前及时到达；另一种情况是处理器发出的预取请求到达存储器的时间太早，预取的数据在处理器需要之前的较早时间到达，使CACHE中的有用数据被暂时不需要的预取数据替换。结果是数据预取不但没有发挥应有的功能，反而对系统的整体性能带来负面影响。因此，如何结合热点区特征提高数据预取的时效性至关重要，但目前有关

收稿日期：2010 – 04 – 28;　修回日期：2010 – 02 – 19

基金项目：教育部–英特尔信息技术专项科研基金(MOE-INTEL-08-10)；北京市重点学科建设项目

作者简介：黄　艳(1977 –)，女，博士，主要从事并行计算和缓存性能优化等方面的研究.

帮助线程预取的研究中还没有切实有效的办法。

为了对访存密集型应用程序进行更加有效的帮助线程数据预取，更好地提高应用程序性能，有必要分析程序访存热点区的特征信息及其对数据预取性能的影响。

1 相关工作

帮助线程数据预取是伴随着计算机系统和应用发展的新趋势而出现的。目前，帮助线程数据预取的研究工作主要集中在帮助线程的构造、帮助线程与主线程的同步和帮助线程的动态优化几个方面。

文献[4]提出了一种通过程序运行时的post-retirement队列自动构造帮助线程的硬件机制。文献[5]开发了一个post-pass的二进制改写工具，分析已有的应用程序库，从中抽取目标指令构建帮助线程，并把帮助线程附加至源库文件，形成一个新库链接到源程序中。文献[6]通过一个编译器框架在源代码级自动建造帮助线程。

帮助线程为了使其预取的数据为主线程所需，应与主线程进行同步。文献[7-8]均采用一种被称为Mailbox的同步措施，主线程通过在共享存储器中的Mailbox启动帮助线程与其同步。Mailbox除按常规的CACHE机制存放于共享L2 CACHE中，还存放于帮助线程处理器核的L1 CACHE中。

帮助线程的动态优化[7-9]是指根据程序执行时的行为动态变化特性调整帮助线程的预取策略。文献[7]将帮助线程数据预取与动态优化融合，在独立于主线程的核上执行，根据程序缓存失效的动态行为实时地优化数据预取措施。文献[8]通过扩展Trident优化器实现帮助线程预取，并通过对规则数据访问采取硬件预取措施、最小化帮助线程的控制流、动态自修复帮助线程等技术优化帮助线程。文献[9]采用实时剖析、在线分析与优化、预取休眠3个阶段的帮助线程优化措施。

以上对帮助线程的优化多是在程序执行时根据程序的缓存失效行为特征进行优化。本文基于片上多处理器分析与测试访存密集型应用程序的帮助线程数据预取性能，优化工作强调根据热点区计算/访存延迟比率进行的间隔预取调整。程序的热点区计算/访存延迟比率通常比缓存失效行为更容易获得，也更稳定。尽管目前只对有限的几个基准程序作了静态优化，下一步将开发出统一的优化框架对访存密集型程序进行统一优化，并由静态优化向动态优化过渡。

2 基于CMP的帮助线程数据预取性能测试

基于CMP(片上多处理器)的帮助线程作为目前数据预取技术研究的重点方向，其基本思想是分析热点循环结构，首先得到一个循环中访存指令之间相关性的数学描述，然后根据该数学描述分析出必定没有数据相关的访存指令，并调度它们前瞻地执行。也就是说，CMP中的帮助线程运行的程序是主线程程序的一个子集，能保证帮助线程的运行总是领先于主线程的运行，并总是先于主线程把需要的数据送达共享缓冲区内，达到数据预取的目的。典型的CMP处理器结构如图1所示。

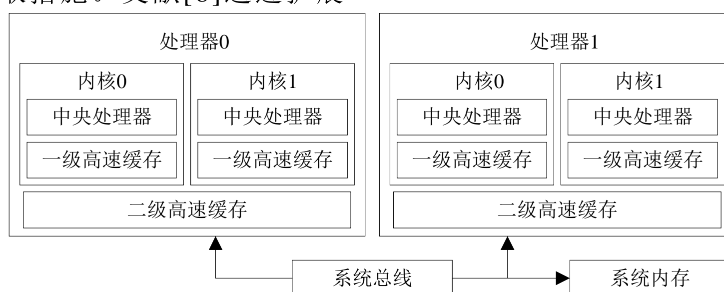


图1 共享2级缓存的双核CMP系统

应用程序的主线程与帮助线程分别运行于同一个处理器的不同处理器核，每个处理器核有一套完整独立的处理单元，相当于一个相对简单的单线程微处理器，处理器核之间(即主线程与帮助线程之间)共享系统内存和L2 CACHE。由于帮助线程只执行主线程中与频繁缺失的长延迟装载指令相关的指令代码[10]，帮助线程总是先于主线程把长延迟装载指令需要的数据提取到共享L2 CACHE中，使主线程在执行长延迟装载指令时不需要访问内存，直接在L2 CACHE中获取，隐藏了访存延迟。

但帮助线程并不保证总是能获得性能收益[11-13]，有时也会对主线程带来一些负面影响。首先，帮助线程的启动会引起额外的开销。其次，为避免帮助线程执行的指令区间离主线程执行的指令区间太

远，主线程与帮助线程间需进行同步也会引起额外的开销。如果这些开销的总和超出了主线程的性能收益，程序的实际执行性能将会下降。因此，分析程序热点区的特征，预测可能的预取性能收益，对于适当控制帮助线程的预取行为很有意义。

2.1　帮助线程性能测试

程序的热点区执行时间主要由计算延迟与访存延迟两部分组成。帮助线程通过预取主线程中热点区的访存数据，尽量减少应用程序的访存延迟，可达到提高程序性能的目的。然而，帮助线程对主线程有多大帮助，即其能使源程序中的多少L2 CACHE缺失变成L2 CACHE命中，不同的程序有不同的表现[4-6]。

例1　有一简单的指针链表结构用例程序的热点部分：

```
While(iterator){
        Temp=iterater>i_data
        While(i□□<ADDSCALE){
            ⋮
        iterator=iterator>next;
        }
```

其对应于对链表的遍历过程代码。遍历节点间插入了一段循环数值计算代码，ADDSCALE变量的值是其循环次数。改变ADDSCALE变量的值可以调整链表节点间的计算量的大小。实验中，对该程序应用帮助线程进行预取，测试当ADDSCALE变量的值从0开始以步长5递增时，L2 CACHE缺失数与运行时间的变化，以分析节点间的计算/访存特性对线程预取性能的影响。帮助线程使用POSIX的pthread线程库手工编写产生，性能参数L2 CAHE缺失数由INTEL的开源工具Vtune测试获得。实验平台采用Intel的Xeon 5110双核处理器，每个处理器核有独立的一级指令与数据缓冲区，容量均为64 KB。位于同一个处理器上的两个处理器核共享一个容量为4 MB的L2 CACHE。实验的测试结果如图3所示。

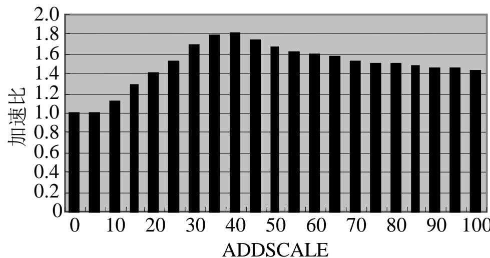


图2　线程数据预取的加速比

图2显示了帮助线程预取程序在ADDSCALE取不同值情况下的运行时间加速比。从图中可以看出，当ADDSCALE<40时，加速比呈增长趋势，在ADDSCALE=40时，加速比达到最大(1.81)。而当ADDSCALE>40时，加速比逐渐减小。随着ADDSCALE递增，相对于节点间计算量为0的源程序运行时间的增加情况如图3所示。图中，水平基准线是当ADDSCALE为0时源程序的运行时间，随着ADDSCALE值的增加，计算工作量增加，运行时间也随之增加。从图中可以看出，如果没有帮助线程预取，ADDSCALE值增加到40时，源程序运行时间增加的曲线与基准线相交，说明此时源程序的运行时间是ADDSCALE为0时的两倍。也就是说，此时的计算工作量从0增加到与访存工作量相当，帮助线程程序在该点呈现为临界状态。在该点之前(0<ADDSCALE<40)，随着计算工作量的增加，运行时间的增加不明显。在此点之后(ADDSCAL*E*>40)，随着计算工作量的增加，运行时间呈线性增加趋势。帮助线程程序与源程序运行时间之差在该临界点达到最大，进一步验证了图2的结果。图2与图3的实验结果说明，当热点区计算延迟接近于访存延迟时，帮助线程取得了更好的性能收益。

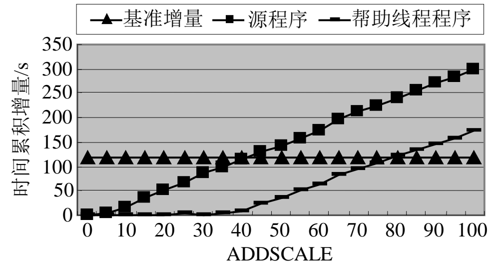


图3　计算量递增时时间增量的对比

2.2　热点区计算/访存延迟比率对帮助线程性能的影响分析

上面的测试表明，帮助线程辅助程序获取的加速比与热点区的计算延迟和访存延迟有关联。下面通过对热点区的执行时间进行分析找到该关联的特征。

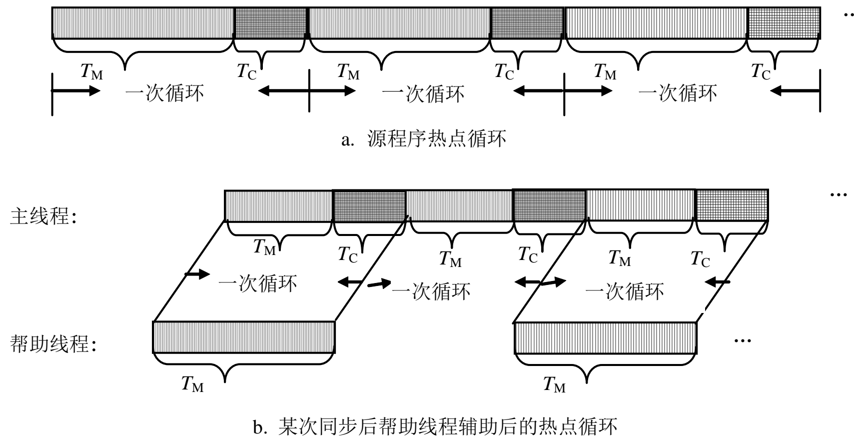
定义 1　设程序热点区执行过程中计算时间为T_C，访存时间为T_M，计算与访存共享的时间片记为T_{CM}，访存并行的时间片记为T_{MM}，程序热点区总执行时间为$T\Sigma_{C\oplus M}$，忽略线程间同步开销，则有：

$$T_{\Sigma C\oplus M}=T_C+T_M-T_{CM}-T_{MM} \tag{1}$$

定义 2　源应用程序的执行过程中，其热点区计算操作消耗的时间与访存操作消耗的时间的比值即为热点区计算/访存延迟比率(CALR)，由定义1知CALR=T_C/T_M。

源串行程序不能实现访存并行(不考虑存储器并行的情况)，即T_{MM}=0；串行程序中T_{CM}只能由预取

规则访存的硬件预取获得，假设T_{CM}可忽略不计(帮助线程主要预取对象为不规则访存，与硬件预取不冲突)。由此可得源程序热点区运行时间组成，如图4a所示。



a. 源程序热点循环

b. 某次同步后帮助线程辅助后的热点循环

图4 热点区运行时间组成

源程序热点循环的时间序列如图4a所示；应用线程数据预取时，某次同步后的执行时间序列如图4b所示。因为帮助线程提前进行了访存操作，主线程在帮助线程的访存操作完成时立即进行计算操作；在主线程计算的同时，帮助线程继续预取下一个访存数据。从图4b中可以看出，帮助线程为主线程节省的时间，也就是帮助线程的数据访问与主线程的计算操作的并行执行时间，即T_{CM}。主线程做完计算操作时，不能立即在L_2 CACHE中取得下一个数据，需发出与帮助线程预取同样的访存请求，等待预取数据到达。该情况下，主线程与帮助线程没有并行访存行为，因此$T_{\text{MM}}=0$。需要注意的是，图4中$T_{\text{C□}}<T_{\text{M}}$，以上分析也是基于$T_{\text{C□}}<T_{\text{M}}$时的分析。本文的关注对象为访存密集型应用程序，其热点区的长延迟指令是主要的性能瓶颈，如无特殊说明，热点区中均是$T_{\text{C□}}<T_{\text{M}}$，也即CALR<1。

由以上分析可知，通过帮助线程辅助应用程序可获得的加速比为：

$$\text{speedup}=(T_{\text{C}}+T_{\text{M}})/(T_{\text{C}}+T_{\text{M}}-T_{\text{CM}})$$

理想情况下，主线程的计算操作和帮助线程的访存操作无竞争地并行，$T_{\text{CM}}=T_{\text{C}}$，如图4b所示。根据定义2，可得：

$$\text{speedup}=(T_{\text{C}}+T_{\text{M}})/T_{\text{M}}=\text{CALR}+1 \quad (2)$$

因此，源程序热点区的计算延迟越大(T_{C}越大)，帮助线程分担主程序中的访存任务就越多(T_{CM}□越大)，应用程序可获得的加速比也越大。然而，如果T_{C}很小，甚至可以忽略不计时，帮助线程几乎不能为应用程序带来收益。由于同步开销，应用程序性能反而可能降低，也可解释文献[9-14]中帮助线程对有些程序失效的原因。

3 基于CALR的帮助线程间隔预取

一方面，CALR对帮助线程预取性能有重要影响；另一方面，帮助线程在试图预取主线程中的所有热点数据时，只能获得与主线程计算操作的并行，不能获得与主线程访存操作的并行。事实上，对于访存密集型热点区，如果帮助线程与主线程能共同分担访存任务，将会得到更好的性能。为此，本文提出一种新的帮助线程预取方法，即基于CALR的帮助线程间隔预取，该方法对两层以上循环的数据操作热点区有效。

定义 3 帮助线程数据预取程序中，帮助线程每间隔A个数据访问连续预取P个数据的预取方法即为间隔预取。

相对于间隔预取，称帮助线程连续预取所有的数据的方法为连续预取。本文仍然通过例1测试，当CALR一定时，帮助线程访存任务分担比率与性能获益的关系。本文选择ADDSCALE的值分别为0、4、12、20、24、40时的情况进行测试，ADDSCALE为0时，CALR为0；ADDSCALE为40时，CALR约等于1。对于不同的ADDSCALE值，通过改变A与P的值，测试预取性能变化。实验结果分别如图5和图6所示。

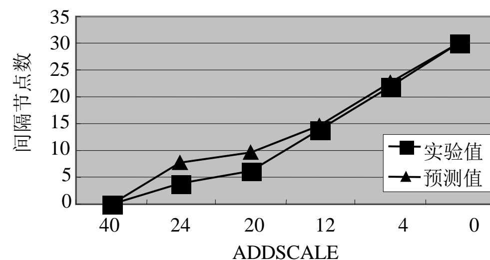


图5 $P=30$时的最佳预取间隔

当$P=30$，并且达到最大加速比时，ADDSCALE

值与间隔节点数A值的关系如图5所示。从图中可以看出，为了取得最佳性能，帮助线程跳过的节点数随着工作量的减少而增加。热点区计算量约等于访存延迟(ADDSCALE为40)时，帮助线程连续遍历数组节点，不跳过访问结点，性能最好。热点区计算量基本为0(ADDSCALE为0)时，$A=P$，性能最好。当热点区计算量位于中间状态(0<ADDSCALE<40)时，A位于0与P之间，帮助线程获取最佳性能。为了进一步说明最佳性能时A和P的关系，在10与1 000之间改变P的值做同种测试，可得到相似的实验结果。

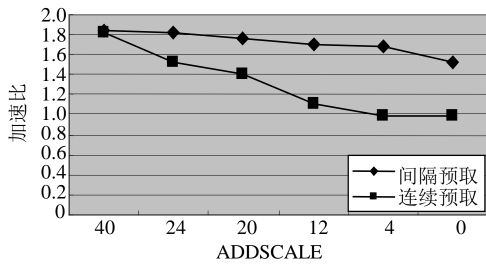


图6　加速比的对比

对图2中连续预取的最大加速比和间隔预取的最大加速比所作的比较如图6所示。从图中可以看出，热点区计算/访存延迟比值越小(ADDSCALE的值越小)，二者的区别越大，间隔预取的优势越明显。图2的实验主要是为了验证CALR对帮助线程数据预取性能的影响，因此对不同的热点区工作量采用统一的连续预取机制，帮助线程连续遍历节点数组中的所有节点。当计算工作量很小时，帮助线程总是无法领先于主线程运行，所以也就无法达到预取的目的。另外，帮助线程还引入了额外的开销，使得运行时间反而有所增加。图6也显示当计算量很小(ADDSCALE=0)时，连续预取的加速比略小于1。间隔预取根据CALR调整数据预取机制，即帮助线程每连续遍历P个节点后跳过A个节点。帮助线程与主线程进行合理的分工，可避免帮助线程盲目预取带来的负面影响。所以ADDSCALE为0时，其加速比明显大于1，如图6所示。

4　基于CALR的帮助线程性能分析

由对间隔预取的测试可以看出，CALR对最优的A和P值选择有着重要影响。同样地，通过对热点区的执行时间分析可确定CALR对A和P的影响。

帮助线程采用间隔预取，不仅可以使预取操作与主线程计算操作并行执行，还可以与主线程访存操作并行执行，如图7所示。

定理 1　理想状态(忽略线程启动、线程间同步及公共资源竞争的影响)下，对访存密集型程序采取间隔预取，当帮助线程与主线程的访存工作量比为(1+CALR)/(1−CALR)时，帮助线程可使程序热区获得最大性能加速比，其值为$2(T_M+T_C-T_{CM})/(T_M+T_C)$。

证明　根据定义1，源程序的总运行时间为$T_M+T_C-T_{CM}$。理想状态下，帮助线程与主线程完全并行，即各分担源程序时间的一半，最小执行时间为$(T_M+T_C)/2$，因此加速比最大为$2(T_M+T_C-T_{CM})/(T_M+T_C)$。设此时帮助线程访存任务量为$T_P$，主线程的访存任务量为$T_A$。由已知条件$T_P+T_A=T_M$和$T_P-T_A=T_C$得：

$$T_P=(T_M+T_C)/2$$

$$T_A=(T_M-T_C)/2$$

所以，加速比最大时，帮助线程与主线程的访存工作量比为$T_P/T_A=(T_M+T_C)/(T_M-T_C)$。又由定义中CALR=$T_C/T_M$得：

$$T_P/T_A=(T_M+T_C)/(T_M-T_C)=(1+\text{CALR})/(1-\text{CALR})$$

因此，定理 1 得证。

利用定理1可以对帮助线程预取进行优化。虽然实际情况比定理1中的假设条件复杂得多，但仍然可以用定理1指导帮助线程的预取优化，使帮助线程得到次优性能。尽管帮助线程与主线程共享低级缓存与主存储器，使得二者同时访存有困难，但以上实验说明，仍然可以通过帮助线程间隔预取，合理安排二者的访存任务，改善程序性能。

本文针对例1中的程序，选定帮助线程每连续遍历30个节点(P=30)，由定理1可得最佳预取间隔的预测值曲线，如图5所示。从图中可以看出，预测值与实际值很接近，说明可以用定理1选择合适的间隔预取，以取得较优的预取性能，如图7所示。

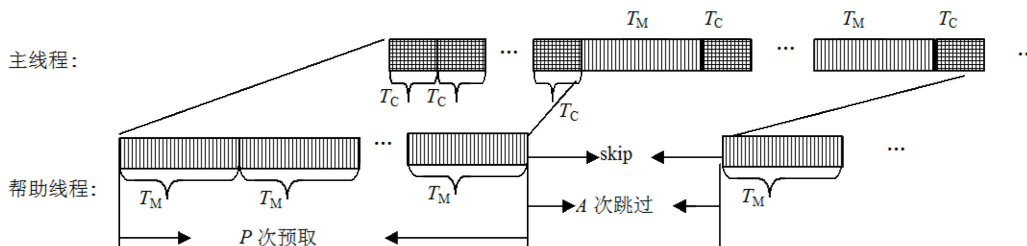


图7 间隔预取原理

5 基于CALR的帮助线程性能优化

前面的实验结果验证了间隔预取的有效性，下面基于CALR对基准测试程序集Olden中的mst、em3d程序和SPEC2006中的mcf程序进行间隔预取，以达到对帮助线程性能的优化。实验平台与之前相同，采用gcc编译器，优化选项为-O2。通过对几个源程序的分析，可以发现，几个程序中热点区的计算量都很小，基本可以忽略不计，即CALR为0。根据定理1可预测，当帮助线程与主线程的访存任务比为1(即$A/P≈1$或$A/(A+P)≈0.5$)时，间隔预取的加速比最大。

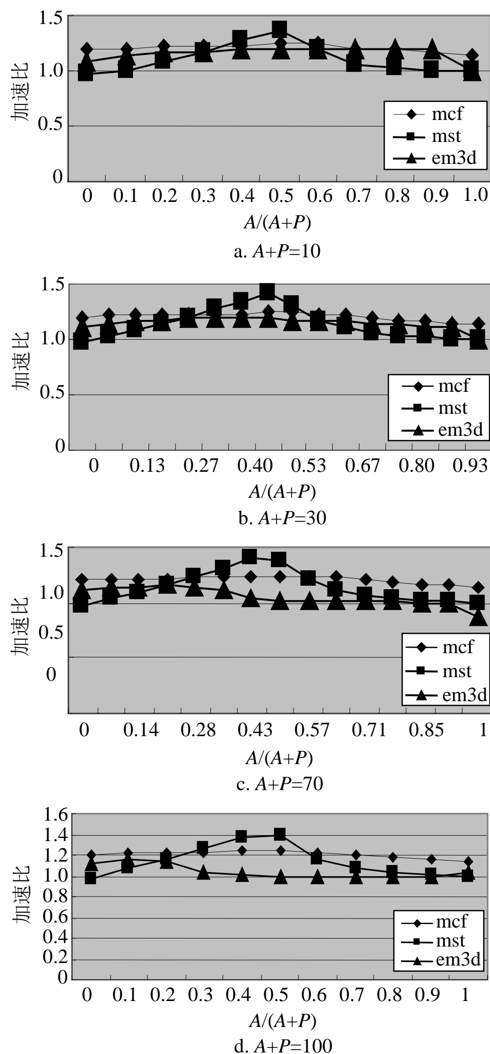


图8 帮助线程占不同访存任务比例时的加速比

为了在帮助线程性能优化的同时证明定理1的正确性，在以下的间隔预取实验中，固定$(A+P)$的值，测试加速比随A或P值变化(A/P的值也随之变化)而变化的情况。分别取$(A+P)$的值为10、30、70、100，各程序加速比的变化情况分别如图8a～图8d所示。P=0时，A/P无法在图中表示，图8中显示的是各个程序的加速比随着$A/(A+P)$值增加时的变化情况。从图中可以看到，对于不同的$(A+P)$的值，随着$A/(A+P)$的值从0～1的变化：1) mcf与mst的加速比始终是在$A/(A+P)=0.5(A/P=1)$附近达到最大；$(A+P)$的值较小时，em3d的加速比在$A/(A+P)$□$=0.5(A/P=1)$附近达到最大，随着$(A+P)$的值的增加，其最大加速比由曲线中间向左移动，与根据热点区计算/访存比率预测结果基本相符。2) mcf的加速比都是在[1.1,1.25]区间内变化；mst的加速比基本都在[0.97,1.43]区间范围内变化；em3d加速比区间从$(A+P)$=10时的[1.01,1.21]变为$(A+P)$=100时的[0.98,1.13]。以上结果表明，em3d的最大加速比及加速比区间对$(A+P)$的值都比较敏感，说明A和P值的选取除了受程序热点区计算/访存比率影响，还受其他因素影响。

帮助线程优化前(连续预取)与优化后(间隔预取)的具体数据如表1所示。从表中连续预取与间隔预取的加速比对比可以看出，基于CALR的间隔预取更有效，说明根据热点区计算/访存延迟比率预测预取间隔，能够优化帮助线程，明显改善帮助线程的数据预取性能。

表1 基准测试程序的实验结果

程序	输入参数	原执行时间/s	连续预取加速比	间隔预取加速比
mcf	参考输入集	466.2	1.14	1.25
mst	$1×10^4$个节点	17.464	0.99	1.43
em3d	$4×10^5$节点，元数128	59.184	1.11	1.21

6 结 论

本文一方面分析程序访存热点区计算/访存延迟比率特征；另一方面调整帮助线程与主线程的访存任务比率，比较不同访存任务比率时的预取性能。测试了热点区计算/访存延迟比率对帮助线程预取性能的重大影响。理论分析与实验结果均表明，如果热点区计算量很小可以忽略不计，当帮助线程与主线程的访存任务比率接近1时，帮助线程预取可获得最好的性能收益。

参 考 文 献

[1] LUK C. Tolerating memory latency through software-controlled preexecution in simultaneous multithreading processors[C]//Proc Int'l Symp on Comp Arch. . Goteborg, Sweden: Institue of Electrical and Electronic Enginecrs Society, 2001: 40-51.

[2] GRANNAES M, JAHRE M, NATVIG L. Low-cost

open-page prefetch scheduling in chip multiprocessors[C]// Proceedings of the IEEE International Conference on Computer Design. Lake Tahoe, CA: IEEE Press, 2008: 390-396.

[3] ZHANG Wei-feng, CALDER B, TULLSEN D M. A self-repairing prefetcher in an event-driven dynamic optimization framework[C]//Proceedings of the International Symposium on Code Generation and Optimization. New York: IEEE Computer Society, 2006: 50-64.

[4] COLLINS J D, TULLSEN D M , WANG H, et al. Dynamic speculative precompuation[C]//Proceedings of the 34th International Symposium on Microarchitecture. LOS Alamitos: IEEE Press, 2001: 306-317.

[5] LIAO S S W, WANG P H W, HOEHNER G, et al. Post-pass binary adaptation for software-based speculative precomputation[C]//Proceedings of the ACM SIGPLAN'02 Conference on Programming Language Design and Implementation. Berlin: ACM, 2002: 117-128.

[6] KIM D, YEUNG D. Design and evaluation of compiler algorithms for pre-execution[C]//Proceedings of the10th International Conference on Architectural Support for Programming Languages and Operating Systems. San Jose: IEEE, 2002: 159-170.

[7] LU J, DAS A, HSU W, et al. Dynamic helper threaded prefetching on the sun ultrasparc cmp processor[C]// Proceedings of the 38th International Symposium on Microarchitecture. Barcelona: Computer Society, 2005: 93-104.

[8] ZHANG Wei-feng, TULLSEN D M, CALDER B. Accelerating and adapting precomputation threads for effcient prefetching[C]//Proceedings of the 13th Symposium on High-Performance Computer Architecture. Piscataway: IEEE Press, 2007: 85-95.

[9] CHILIMBI T M, HIRZEL M. Dynamic hot data stream prefetching for general-purpose programs[C]//Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. Berlin: Association for Computing Machinary, 2002: 199-209.

[10] SONG Y, KALOGEROPULOS S, TIRUMALAI P. Design and implementation of a compiler framework for helper threading on multi-core processors[C]//Proceedings of the International Conference on Parallel Architectures and Compilation Techniques. LOS Alamitos: IEEE Computer Society, 2005: 99-109.

[11] FUKUMOTO N, MIHARA T, INOUE K, et al. Analyzing the impact of data prefetching on Chip MultiProcessors [C]//Proceedings of the Asia-Pasific Computer Systems Architecture Conference. Piscataway: IEEE Press, 2008: 1-8.

[12] ROTH A, SOHI G S. A quantitative framework for automated pre-execution thread selection[C]//Proceedings of the International Symposium on Microarchitecture. Los Alamitos: IEEE Computer Society, 2002: 430-441.

[13] AAMODT T M, CHOW P P. Optimization of data prefetch helper threads with path-expression based statistical modeling[C]//Proceedings of the 21st Annual International Conference on Supercomputing. New York: ACM, 2007: 210-221.

[14] ROGERS A, CARLISLE M C, REPPY J H, et al. Supporting dynamic data structures on distributed memory machines[J]. ACM Toplas Transactions on Programing Languages and Systems, 1995, 7(2): 233-263.

编 辑 漆 蓉