doi:10.3772/j.issn.1002-0470.2017.11－12.004

# 提升处理器指针追逐访存性能的指令标签辅助的数据预取机制①

刘天义②*** 肖俊华*****章隆兵*****沈海华③***

(*中国科学院计算技术研究所计算机体系结构国家重点实验室 北京 100190)

(**中国科学院大学 北京 100049)

(***龙芯中科技术有限公司 北京 100190)

**摘 要** 分析了处理器访存操作的指针追逐模式,指出了链式数据应用中的指针追逐操作的数据预取准确率低、访存延迟大的问题。为了提升处理器指针追逐访存性能,提出了指令标签辅助的数据预取(ILAMP)技术。ILAMP技术是一种指令标签提示的预取机制,其通过在指令集架构中添加新的访存指令,使该指令在处理器译码阶段产生特殊访存标签,指明该访存操作的加载内容是指针。在Cache缺失的情况下,该标签一直传递到内存控制器。当加载的指针返回内存控制器时,则提取指针、发出预取请求。实验结果表明,ILAMP技术与无ILAMP情况相比,ILAMP技术降低DRAM读请求的平均访问延迟的平均值约为15%,预取精度高于77%,访存带宽增加10%左右,硬件开销约为1kB。

**关键词** 链式数据,指针追逐,指令标签,数据预取,增强型内存控制器

## 0 引 言

20世纪80年代以来,在摩尔定律的驱动下,处理器的集成度每18个月翻一番。工艺进步带来了处理器性能的迅猛提升,而内存性能的提升很慢,因而这两者之间逐渐形成了性能的剪刀差[1],也正是这种速度上的不匹配,使访存部件成为影响处理器性能提升的关键。

对于隐藏处理器的访存延迟往往有多种技术,例如乱序执行、非阻塞Cache、同时多线程、访存操作推测执行[2]、辅助线程提前执行[3]等。除了上述技术外,还有一种重要技术是数据预取。数据预取与上述这些技术是正交关系,在这些技术的基础之上增加高效的数据预取器,有助于处理器性能的进一步提高。况且,预取技术真正缩短了访存请求的数据命中时间。

传统的顺序预取技术[4-7]和Stride预取技术[6-8]对基于规则数据集的应用效果显著。由于规则数据集操作的空间和时间局部性好,应用程序的访存规律性强,因此处理器随后访问的数据就比较容易通过猜测的方法提前取到Cache中。然而,对于基于链表、树、图等数据结构的应用,访存操作往往表现出指针追逐的模式。处理器先从内存中加载指针,然后以该指针为地址再次进行访存操作。指针追逐的访存模式降低了数据访问的空间局部性,造成了较高的Cache失效率和较长的存储器访问延迟。

为了解决该问题,本文提出了指令标签辅助的数据预取(instruction label assisted memory prefetching, ILAMP)机制,通过向指令集架构中添加指针追逐访存操作的扩展指令,使处理器对加载普通数据和加载指针进行区分,并且通过在内存侧专门设计

① 国家"核高基"科技重大专项课题(2009ZX01028-002-003,2009ZX01029-001-003,2010ZX01036-001-002,2012ZX01029-001-002-002,2014ZX01020201,2014ZX01030101),国家自然科学基金(61521092,61133004,61173001,61232009,61222204,61432016),863计划(2013AA014301)和广东省普及型高性能计算机重点实验室开放课题(SZU-GDPHPCL-2012-03)资助项目。

② 男,1991年生,硕士生;研究方向:计算机系统结构;E-mail: liutianyi@ict.ac.cn

③ 通信作者,E-mail: shenhh@ucas.ac.cn

(收稿日期:2017-05-22)

的内存控制器增加虚实地址转换的功能,完成对指针追逐操作的精准预取。

## 1 背 景

在基于指针操作的应用中,不可避免地会利用到指针追逐访存。而且指针追逐往往是这类程序的核心和最耗时的关键操作[9,10]。因为这种非规则访存模式的数据 Cache 缺失率往往高于普通访存操作[11]。本节先介绍指针追逐访存的基本概念,然后介绍传统内存控制器结构。

### 1.1 基于指针追逐的非规则访存操作

图 1 展示了链表元素求和的例子。该程序的基本数据结构和核心代码片段如图 1(a)所示。经过编译和反汇编之后的核心代码序列如图 1(b)所示。

```
struct node {
    int val;
    struct node *next;
}
```

```
While(p=p->next)
{
    sum+= p->val;
}
```

(a) 源代码

```
120000684:    70 00 2f a4    ldq     t0,112(fp)
120000688:    08 00 21 a4    ldq     t0,8(t0)
12000068c:    70 00 2f b4    stq     t0,112(fp)
120000690:    06 00 20 e4    beq     t0,1200006ac <main+0xec>
120000694:    70 00 2f a4    ldq     t0,112(fp)
120000698:    80 00 4f a0    ldl     t1,128(fp)
12000069c:    00 00 21 a0    ldl     t0,0(t0)
1200006a0:    01 00 41 40    addl    t1,t0,t0
1200006a4:    80 00 2f b0    stl     t0,128(fp)
1200006a8:    f6 ff ff c3    br      120000684<main+0xc4>
1200006ac:    80 00 2f a0    ldl     t0,128(fp)
1200006b0:    fe ff 5d 24    ldah    t1,-2(gp)
```

(b) 汇编代码

**图 1 链表求和示例**

在图 1(b)所示的汇编代码中,处理器先从内存中加载链表中头节点的指针 p。然后,以 p 中的内容为基地址,偏移 8,再从内存中加载出 p->next。接着,将 p->next 存入 p 中后,判断 p 是否为空指针。如果不为空,则重新加载 p 到 t0 寄存器,加载 sum 到 t1 寄存器中。接下来,以 t0 寄存器中的内容为地址加载变量 val 到 t0 寄存器后,进行变量 val 和变量 sum 的加法操作,并把结果放入 t0 寄存器,再存入 sum 变量中。最后,跳转到第一行继续循环执行。

在上文的链表元素求和例子中,有 2 对指针追逐操作,如图 1(b)中的虚线框所示。这种非规则访存操作,先从内存中加载指针,然后以该指针或者该指针加上偏置作为新的地址再次进行访存操作。由于进行的是指针加载操作,传统预取策略难以准确捕捉访存规律。又由于前后两条访存操作存在数据相关,在前一条发生 Cache 缺失的情况下,后一条访存操作只能被阻塞,处理器的访存操作被串行化。因此,在以指针追逐操作为核心的应用程序中,系统的整体性能受到这种访存间依赖的影响。

### 1.2 传统内存控制器结构

传统的内存控制器的结构如图 2 所示。内存控制器[12-14]主要包括前端和后端两个部分。前端提供访存的接口,独立于具体的内存类型,包括请求队列(读队列、写队列)、响应队列、内存映射部件、仲裁器。请求队列负责对访存请求进行缓存,响应队列暂存已完成的访存请求;内存映射部件将访存地址转化为动态随机存取存储器(DRAM)的实际物理结构,仲裁器负责对访存请求的顺序进行调度。内存控制器的后端与具体存储器类型相关,主要包括命令产生逻辑和时序约束的控制逻辑。

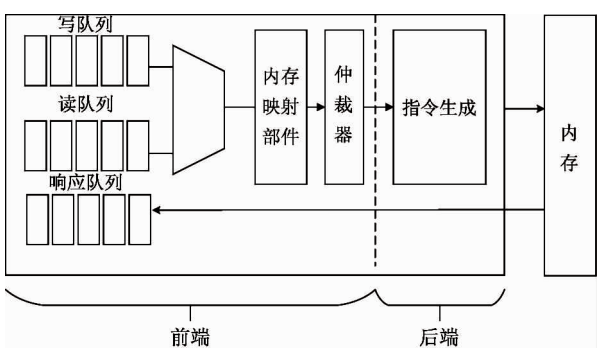


**图 2 内存控制器结构**

## 2 指令标签提示的预取系统

指令标签提示的预取系统主要包括 3 个方面的内容,分别是可产生标签的访存指令、支持虚实地址转换的增强型内存控制器和专门的预取缓冲器。

### 2.1 可产生标签的访存指令

为了在内存控制器中对指针追逐操作进行准确预取,必须对加载指针的访存请求与加载普通数据的访存请求进行区分。而现有大多数指令集架构,往往并未明确针对这点进行优化,因此导致处理器

和存储器架构不能充分利用程序的行为特点。本文提出了可产生标签的访存指令,专门用于从存储器中加载指针。指令的功能描述如图 3 所示。

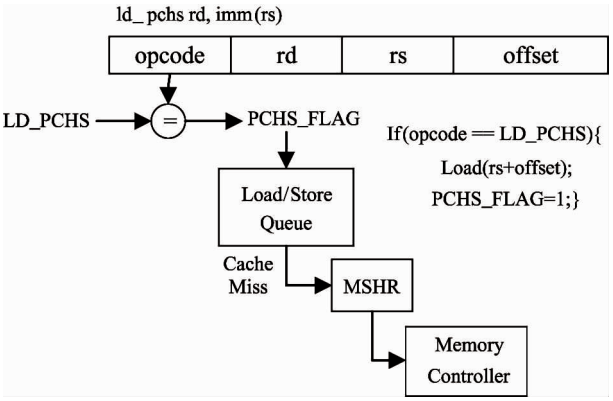


**图 3　指令标签在存储器层次中的传输**

在图 3 中,opocde 是指令码域,用于表明指令的类型,rd 域指明目的寄存器,rs 域为源寄存器,offset 域为地址的偏移量。该指令可以实现普通 load 指令的功能,即从内存地址(rs + offset)的位置处加载内容。同时,在指令译码阶段,该指令能够产生加载指针的标志 PCHS - FLAG。该标志可以跟随访存请求进入访存队列。在 Cache 缺失的情况下,进入 MSHR,在 MSHR 未分配的情况下,分配 MSHR 后,最终传递到内存控制器。对于 MSHR 已经分配的项,本文采取保守做法,不再向下传递。因此,该指令就可以在完成常规访存操作的同时,对处理器和存储器架构进行提示,表明该访存请求加载的是指针。用该访存指令替换图 1 中的汇编指令,则两个虚线框中的第一条指令均需要被替换为 ld - pchs,如图 4 所示。

```
120000684:  70 00 2f 14     ld_pchst0,112(fp)
120000688:  08 00 21 a4     ldq     t0,8(t0)
12000068c:  70 00 2f b4     stq     t0,112(fp)
120000690:  06 00 20 e4     beq     t0,1200006ac<main+0xec>
120000694:  70 00 2f 14     ld_pchst0,112(fp)
120000698:  80 00 4f a0     ldl     t1,128(fp)
12000069c:  00 00 21 a0     ldl     t0,0(t0)
1200006a0:  01 00 41 40     addl    t1,t0,t0
1200006a4:  80 00 2f b0     stl     t0,128(fp)
1200006a8:  f6 ff ff c3     br      120000684<main+0xc4>
1200006ac:  80 00 2f a0     ldl     t0,128(fp)
1200006b0:  fe ff 5d 24     ldah    t1,-2(gp)
```

**图 4　支持指针追逐访存预取的链表求和**

## 2.2　增强型内存控制器

由于 LD - PCHS 指令加载内容是指针,为了尽早利用该指针进行预取操作,本文将指针追逐的预取逻辑实现在内存控制器中。又由于指针往往是一个虚拟地址,系统若要支持内存侧预取机制,需要有虚实地址转换的装置。因此本文的方案是在内存控制器中添加了一个虚拟-物理(virtual to physical, VTP)模块,负责虚实地址转换。同时为了避免错误预取对 Cache 造成污染,预取的结果放入专门设计的预取缓冲器中,而不是直接放入 Cache。该缓冲器也实现在内存控制器中。整个增强型内存控制器的架构如图 5 所示。

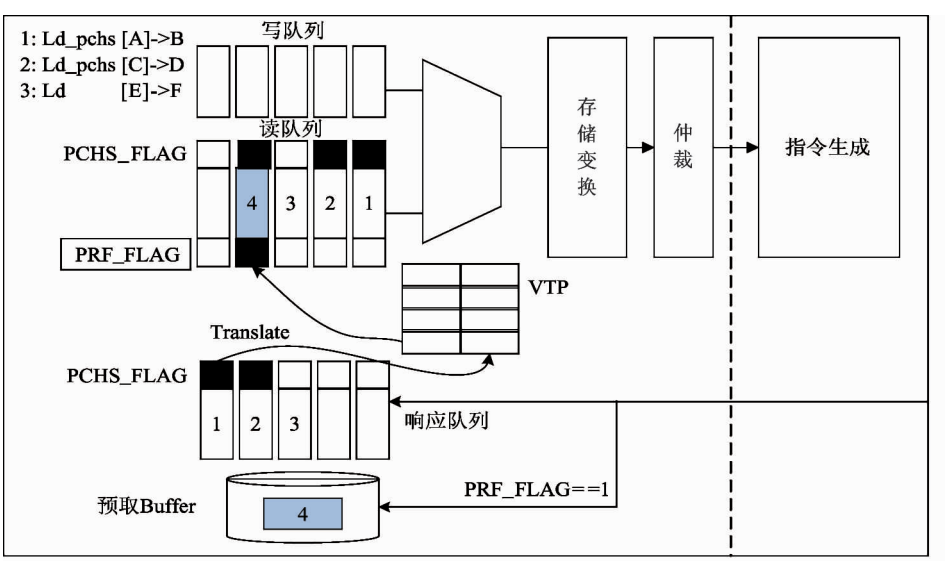


**图 5　增强型内存控制器**

除了在传统内存控制器基础上添加 VTP 模块和预取 Buffer(Prefetching-Buffer)模块之外,在读队列中的每一项都添加了两个标志,分别是 PCHS – FLAG 和 PRF – FLAG。当 PCHS – FLAG 为 1 时,表明该访存请求是指针追逐操作中的加载指针的请求。当 PRF – FLAG 为 1 时,表明该访存操作是指针追逐的预取操作。对于响应队列(Response Queue)中的访存请求,并不是立刻返回处理器核,而是检查其 PCHS – FLAG 标识是否为 1。如果为 1,则从 Cache 行大小的数据块中提取指针、经过 VTP 转换后,得到物理地址。此时,再利用该物理地址生成预取请求,放入读队列(Read Queue)中。

### 2.3 预取缓冲器

对于进入内存控制器的读请求,应先访问预取 Buffer,如果在预取 Buffer 中命中,则直接进入响应队列;如果未命中,则进入读队列。另外,对于所有的写请求,都应该写入 DRAM 的同时,对预取 Buffer 进行更新操作,以确保预取 Buffer 中保存最新数据。从而避免了“脏数据”的问题。

由于指针追逐中访存相关指令往往距离较近,并且一旦在预取 Buffer 中命中,数据立即就被搬运到 Cache 中,预取 Buffer 中的备份就可以立即丢弃。因此,预取 Buffer 的项数无需太多。

### 2.4 VTP 结构

对于 VTP 结构的实现,本文的方案是,始终保持 CPU 侧的数据地址转换查找缓冲器(transaction lookaside buffers, TLB)与内存控制器中的 VTP 同步。如果 VTP 发生缺失,则内存控制器侧的预取器放弃对该地址的预取操作。接下来的处理过程,交给处理器核执行。此时,处理器核根据返回的指针发起访存操作,同样也会产生数据 TLB 缺失。那么此时,由处理器进行异常处理,填充数据 TLB 的同时,填充 VTP 中缺失的页表。

## 3 实验方法

### 3.1 评估指标的计算方法

本文采用存储器读请求平均访问延迟、预取精度、预取块平均使用次数、预取 Buffer 的利用率、IPC 评估本方案的优势,同时采用访存带宽开销、硬件开销评估本方案的实现代价。主要评估指标的计算方法如表 1 所示。

**表 1 评估指标的计算方法**

| 评估指标 | 计算方法 |
|---|---|
| 存储器读请求平均访问延迟 | DRAM-ReadReq-Avg-Latency = DRAM 读请求访存时间总和/DRAM 读请求数量 |
| 预取精度 | Accuracy = 预取数据块中被使用过的块数/预取的总块数 |
| 访存带宽 | bandwidth = 从 DRAM 中读取的字节数/仿真时间 |
| 硬件开销 | 对增加的存储逻辑字节数进行统计,详见 4.4 节 |

### 3.2 实验配置

本实验方案采用 Gem5 模拟器进行评估,模拟器的主要配置参数如表 2 所示。实验采用的 Benchmark 为 Olden 测试集中的 9 个应用程序。测试采用的程序集和数据集特点如表 3 所示。编译器采用 alpha-linux-gcc-4.3.2 进行交叉编译,同时采用-O2 进行优化。

**表 2 系统配置**

| 功能结构 | 参数 |
|---|---|
| Core clocks | 2GHz |
| Itlb | 48 |
| Dtlb | 64 |
| Icache | Assoc 4<br>Block – size 64 |
| Dcache | Assoc 4<br>Block – size: 64 |
| L2 | Assoc 8<br>Block – size 64<br>MSHR hit latency: 20 |
| DRAM | DDR3 – 2133 × 64 |
| RoB | 192 |

**表 3　Benchmark 及输入数据集**

| 程序 | 数据集规模 | 数据结构 |
|---|---|---|
| Bh | 4k body | 八叉树 |
| Bisort | 250000 | 二叉树 |
| Em3d | 2000n-节点;100 d-节点;100times | 链表 |
| Health | Max. level = 5<br>Max. time = 500<br>Seed = 1 | 四叉树链表 |
| mst | 5120 节点 | 链表 |
| perimeter | 4k × 4k image | 四叉树 |
| power | 10000 节点 | 多路树,链表 |
| treeadd | 1024k 节点 | 二叉树 |
| TSP | 100 000 城市 | 二叉树 |

### 3.3　方案验证

以 Olden 测试集中的 Bisort 程序为例,对第 2 节提出的方案进行演示验证,以确保处理器执行了 ld - pchs 指令、译码阶段产生了 PCHS - FLAG 标签,以及在内存控制器中执行了指针提取、发预取请求以及访存操作在内存控制器的预取 Buffer 中命中的关键步骤。Bisort 程序中的核心函数为 Bisort 和 Bimerge,下面仅列出 Bimerge 函数经编译处理以及反汇编的核心代码,如图 6 所示。

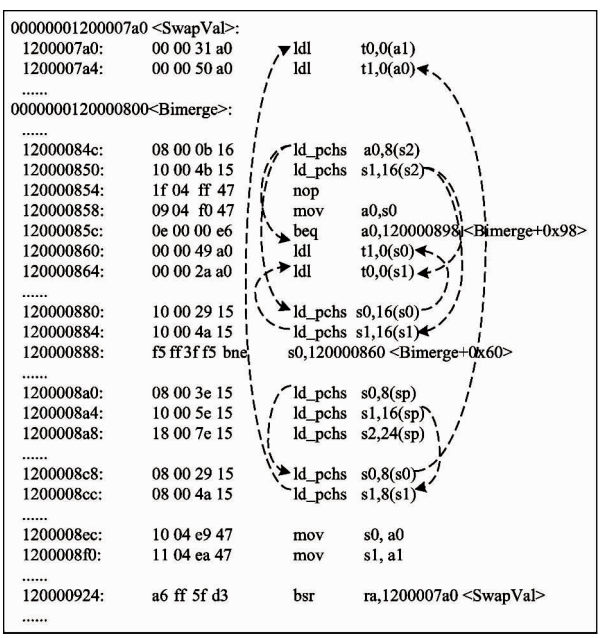


**图 6　Bimerge 函数核心代码**

图 6 中带箭头的虚线表示相匹配的指针追逐访存。箭头的起始端正是本文提出的新指令 ld - pchs。另外,运行 Bisort 程序过程中,内存控制器的总线上,收到 596003 个 ld - pchs 访存请求,因此可证明,译码阶段产生了 PCHS - FLAG 标志,该标记也确实随访存请求传递到了内存控制器中。

模拟器运行过程中,对指针提取、发起预取请求以及后续访存命中预取 Buffer 的过程如图 7 中图(a)和图(b)所示。图 7(a)打印出了在模拟器运行过程中,ld - pchs 指令在返回加载数据的同时,又进行指针提取,然后对该指针进行虚实地址转换后,放入读队列,发起预取请求的过程。图 7(b)展示了后续访存操作在内存控制器的预取 Buffer 中命中所需数据时,所打印出提示信息。

```
system.mem_ctrls: getting the pointer: 0x120096560
system.mem_ctrls: traslating the pointer: 0x120096560
system.mem_ctrls: physical address: 0x86540
system.mem_ctrls: Adding to read queue
system.mem_ctrls: Request scheduled immediately
system.mem_ctrls: prefetching 0x86540 to Buffer...
system.mem_ctrls: Done
```

(a) 指针提取及发起预取请求

```
system.mem_ctrls: accessAndRespon_X,PCHS_FLAG:0, PRF_FLAG:0
system.mem_ctrls: physical address: 0x86540
                  MC Prefetching Buffer hit!!!!!!!!!!!
system.mem_ctrls: Done
```

(b) 后续访存命中预取Buffer

**图 7　指针追逐数据预取及 Buffer 命中过程**

## 4　实验结果及分析

### 4.1　存储器访问延迟

图 8 展示了标准化后的存储器读请求的平均访问延迟。从图上可以看出,指令标签辅助的数据预取(ILAMP)技术相对于无预取和 Stride 预取的情况,读请求的平均访问延迟在各个程序上都有明显下降。在无预取的情况下实现 ILAMP 技术,存储器的读请求平均访问延迟下降的平均值为15.04%,最高为 19%。在 Stride 预取基础上实现 ILAMP 技术情况下,存储器的读请求平均访问延迟平均下降15.03%,最高为18.85%。

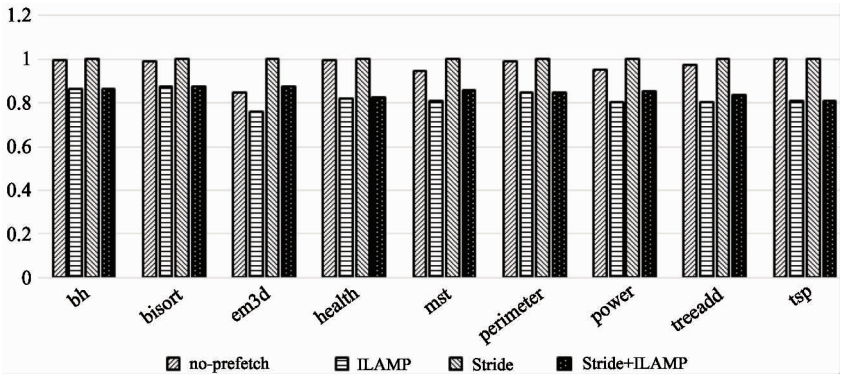


**图 8　存储器读请求平均访问延迟**

值得注意的是,图 8 并未反映 Stride 预取的效果。Stride 预取操作也参与了平均访问延迟的计算,因此才会出现 Stride 预取和无预取的情形的存储器平均访问延迟相当的情况 。同样地,ILAMP 技术和 Stride 预取结合 ILAMP 技术也存在类似情形。

实验数据表明,ILMAP 技术相对于无预取情形的 IPC 提升在 8% 左右,最高提升10.3% 。Stride 预取对于链式数据的访问也有一定的效果,尤其是对于内部数据成员表现出稳定偏移访问的情形。在本文的系统配置下,基于 Stride 预取再实现 ILAMP 预取时,IPC 的提升有5.49% 左右。

### 4.2　预取的准确性、块平均使用次数、预取 Buffer 利用率

ILAMP 技术的准确率对大多程序来说都比较高。图 9 描述了 ILAMP 技术的准确性、预取数据块的平均使用次数以及预取 Buffer 的利用率。从该图可以看出除了 em3d 程序之外,其他程序的预取准确率都在77% 以上。预取块的平均使用次数和预取 Buffer 的利用率都表现出类似的趋势。em3d 程序预取准确率较低是因为核心循环中指针追逐操作的偏移量较大,导致所需数据超过了预取 Cache 行的范围。

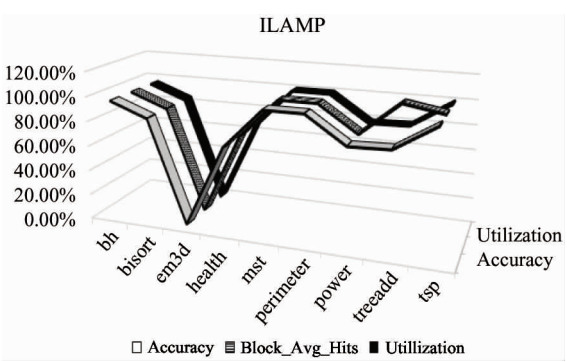


**图 9　预取 Buffer 相关参数**

### 4.3　访存带宽开销

ILAMP 技术的访存带宽开销较低。图 10 展示了 Stride 预取相对与无预取情形、ILAMP 技术相对与无预取情形,以及 Stride 预取与 ILAMP 结合的方案相对于 Stride 预取的情况下,访存带宽的增长对比。图中显示,ILAMP 相对于无预取情形的访存带宽开销平均提升10.44% ,在 Stride 预取基础上再实现 ILAMP 的访存带宽开销平均提升为 6.57% 。而 Stride 相对于无预取情形的访存带宽平均开销为 37.97% ,远高于 ILAMP 技术。

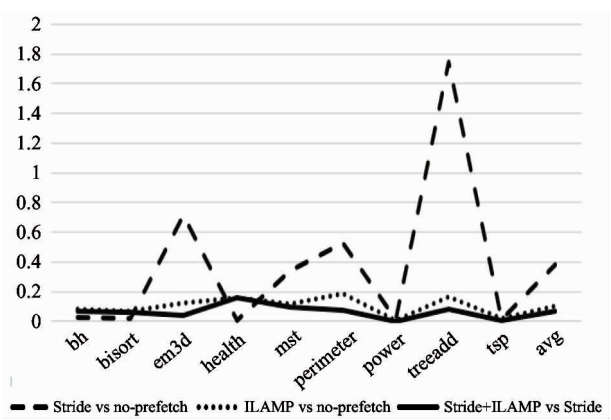


**图 10　访存带宽开销**

### 4.4　硬件开销

预取 Buffer 设计 3 项时,对于 40 位的物理地址,每个 Cache 行 64 字节的情况下,Cache 行的物理

地址为34位。3项需要205个字节。对于VTP,本方案实现64项。每项需要48位虚拟地址,40位物理地址,8位ASID,1位Valid,1位写控制位,12位掩码支持可变页面,再留10位保留位,一共120位,即15字节。64项需要960字节。再加上读队列和响应队列中的PCHS-FLAG和PRF-FLAG位。按照16项计算,需要48位。由于控制逻辑比较简单,几乎可以忽略。所以,本方案的开销约为1171字节。

## 5 相关工作

数据预取往往利用数据访问的顺序性[4-6,15,16]和空间局部性[17-22]原理,但是,在基于指针追逐的访存中该规律不够显著。因此,在处理器核、Cache和内存控制器以及内存计算方面出现了一些关于提升指针追逐访存性能的研究。本节仅介绍与本工作最相关的几个研究工作。

在处理器核方面,Roth等[23]提出依赖表的方法。在访存指令提交时,更新和查询依赖表进行预取。优点在于能够捕捉多数的访存依赖,缺点是当访存依赖的指令距离较近时,预取操作的及时性较差。Onur等[2]提出了Address-Value Delta方法,对长延迟访存指令进行值预测。该方法类似于Stride预取。同样存在预取准确性较低的问题。肖俊华等[24]则结合了Stride预取与指针预取方法,并利用全局历史表减少表中存储的冗余数据。朱会东等[25]提出在CMP系统下,利用辅助线程遍历链表,为主线程进行预取。

在Cache及内存控制器方面,Cooksey等[26]提出如果Cache中的数据与Cache行地址具有相同的高位,则判定该数据为指针。然后,对检测到的所有指针进行预取操作。该方法简单、高效,但会出现较多无用的预取。为了弥补该工作的不足,Ebrahimi等[11]提出编译器剖析的方法找到Cache行中有用的指针,仅仅对有用指针做预取。Yang等[3]在各级存储器层次都设计了预取器。该预取器可以独立执行指针依赖的访存操作序列。从而将CPU核所需要的数据提前取到处理器中。Hughes等[27]提出一种内存侧预取机制,共用处理器核的TLB。Hashemi等[28]提出当发生缺失的访存指令处于重排序缓存(ROB)的头位置时,则将依赖指令序列发送到专门设计的内存控制器,当数据返回内存控制器时,在内存控制器中完成依赖指令序列的计算,并将结果返回处理器核。

在内存计算方面,Hsieh等[9]提出了一种在3D堆叠内存的架构中的指针追逐加速器。Ahn等[29]利用3D-memory的高带宽和内存中的计算单元,进行图的大规模并行处理。Ahn等[20]又设计了监测数据局部性的监测器。当局部性较好时内存计算指令在处理器核中执行,当局部性较差时内存计算指令在3D堆叠存储器中的计算单元中完成数据的处理。

另外,漆锋滨等[30]研究了反馈式编译器优化。包云岗等[31]提出标签化冯诺伊曼架构用于提高数据中心的资源利用率同时确保应用程序的QoS。而本文则提出类似的想法,将这种标签用于数据的硬件预取。

## 6 结 论

高效地预取数据在处理器设计中非常重要。本文研究了指针追逐访存中的数据预取问题,提出了指令标签辅助的数据预取(ILAMP)技术。

ILAMP技术通过特殊的访存指令对硬件系统进行提示,当加载指针的访存操作发生Cache缺失,并返回内存控制器时,就立刻提取指针、进行地址转换,发起预取操作。根据实验结果,ILAMP技术明显降低了存储器读请求的平均访存延迟、预取的准确度高、硬件资源开销少。但同时该技术也有一定的不足之处,特别是当指针追逐操作中的后一条指令的偏移量较大时,预取准确性会受到较大影响。这也是后续工作要解决的问题。

**参考文献**

[1] Hennessy J L, Patterson D P. Computer Architecture: A Quantitative Approach[M]. 5th Edition. 北京: 机械工业出版社, 2012. 73-73

[2] Mutlu O, Kim H, Patt Y N. Address-value delta (AVD)

prediction: increasing the effectiveness of runahead execution by exploiting regular memory allocation patterns [C]. In: Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture, Barcelona, Spain, 2005. 12-12

[3] Yang C L, Lebeck A R. Push vs. pull: data movement for linked data structures [C]. In: Proceedings of the 14th International Conference on Supercomputing, Santa Fe, USA, 2000. 176-186

[4] Dahlgren F, Dubois M, Stenstrom P. Fixed and adaptive sequential prefetching in shared memory multiprocessors [C]. In: Proceedings of the International Conference on Parallel Processing, Syracuse, USA, 1993. 56-63

[5] Dahlgren F, Dubois M, Stenstrom P. Sequential hardware prefetching in shared-memory multiprocessors [J]. *IEEE Transactions on Parallel and Distributed Systems*, 1995, 6(7): 733-746

[6] Dahlgren F, Stenstrom P. Effectiveness of hardware-based stride and sequential prefetching in shared-memory multiprocessors [C]. In: Proceedings of the 1st IEEE Symposium on High Performance Computer Architecture, Raleigh, USA, 1995. 68-77

[7] Dahlgren F, Stenstrom P. Evaluation of hardware-based stride and sequential prefetching in shared-memory multiprocessors [J]. *IEEE Transactions on Parallel and Distributed Systems*, 1996, 7(4): 385-398

[8] Fu J W C, Patel J H, Janssens B L. Stride directed prefetching in scalar processors [C]. In: Proceedings of the 25th Annual International Symposium on Microarchitecture, Portland, USA, 1992. 102-110

[9] Hsieh K, Khan S, Vijayku N, et al. Accelerating pointer chasing in 3D-stacked memory: challenges, mechanisms, evaluation [C]. In: Proceedings of the 2016 IEEE 34th International Conference on Computer Design (ICCD), Scottsdale, USA, 2016. 25-32

[10] Roth A, Sohi G S. Effective jump-pointer prefetching for linked data structures [J]. *SIGARCH Computer Architecture News*, 1999, 27(2): 111-121

[11] Ebrahimi E, Mutlu O, Patt Y N. Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems [C]. In: Proceedings of the IEEE 15th International Symposium on High Performance Computer Architecture, Raleigh, USA, 2009. 7-17

[12] Mutlu O, Moscibroda T. Parallelism-aware batch scheduling: enhancing both performance and fairness of shared DRAM systems [C]. In: Proceedings of the 2008 International Symposium on Computer Architecture, Beijing, China, 2008. 63-74

[13] Mutlu O, Moscibroda T. Stall-time fair memory access scheduling for chip multiprocessors [C]. In: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, Chicago, USA, 2007. 146-160

[14] Rixner S, Dally W J, Kapasi U J, et al. Memory access scheduling [C]. In: Proceedings of the 27th International Symposium on Computer Architecture, Vancouver, Canada, 2005. 128-138

[15] Smith A J. Sequential Program Prefetching in Memory Hierarchies [J]. *Computer*, 1978, 11(12): 7-21

[16] Smith A J. Sequentiality and prefetching in database systems [J]. *ACM Transactions on Database Systems*, 1978, 3(3): 223-247

[17] Somogyi S, Wenisch T F, Ailamaki A, et al. Spatio-temporal memory streaming [C]. In: Proceedings of the 36th Annual International Symposium on Computer Architecture, Austin, USA, 2009. 69-80

[18] Kumar S, Wilkerson C. Exploiting spatial locality in data caches using spatial footprints [J]. *SIGARCH Computer Architecture News*, 1998, 26(3): 357-368

[19] Baier J L, Sager G R. Dynamic improvement of locality in virtual memory systems [J]. *IEEE Transactions on Software Engineering*, 1976, SE-2(1): 54-62

[20] Ahn J, Yoo S, Mutlu O, et al. PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture [C]. In: Proceedings of the 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, USA, 2015. 336-348

[21] Smith A J. A locality model for disk reference patterns [C]. In: Proceedings of IEEE Computer Society Conference, San Francisco, USA, 1975. 109-112

[22] Smith A J. Analysis of a locality model for disk reference patters [C]. In: Proceedings of the 2nd Conference on Information Sciences and Systems, Baltimore, USA, 1976. 593-601

[23] Roth A, Moshovos A, Sohi G S. Dependence based prefetching for linked data structures [J]. *SIGPLAN Not*, 1998, 33(11): 115-126

[24] 肖俊华,冯子军,章隆兵. 片上多处理器中基于步长和指针的预取[J]. 计算机工程, 2009(04):58-60

[25] 朱会东,黄永丽,宋宝卫. 基于 CMP 的指针数据预取方法[J]. 计算机工程, 2011(06):71-73

[26] Cooksey R, Jourdan S, Grunwald D. A stateless, content-directed data prefetching mechanism[J]. *SIGARCH Computer Architecture News*, 2002, 30(5): 279-290

[27] Hughes C J, Adve S V. Memory-side prefetching for linked data structures for processor-in-memory systems [J]. *Journal of Parallel and Distributed Computing*, 2005, 65(4): 448-463

[28] Hashemi M, Multu O, Patt Y N, et al. Accelerating dependent cache misses with an enhanced memory controller [J]. *SIGARCH Computer Architecture News*, 2016, 44(3): 444-455

[29] Ahn J, Hong S, Yoo S, et al. A scalable processing-in-memory accelerator for parallel graph processing[C]. In: Proceedings of the 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, USA, 2015. 105-117

[30] 漆锋滨,王飞,李中升. 基于反馈的链式结构数据预取. 见:全国高性能计算机学术年会, 北京: 中国计算机学会,2008. 88-91

[31] Ma J, Sui X, Sun N, et al. Supporting differentiated services in computers via programmable architecture for resourcing-on-demand (PARD)[C]. In: Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems, Istanbul, Turkey, 2015. 131-143

# ILAMP: an instruction label assistant memory side prefetching technique for improving the performance of processor pointer chasing applications

Liu Tianyi[*,**], Xiao Junhua[*,**,***], Zhang Longbing[*,**,***], Shen Haihua[*,**]

([*]Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**]University of Chinese Academy of Sciences, Beijing 100049)

([***]Loongson Technology Corporation Limited, Beijing 100190)

**Abstract**

The pointer chasing pattern of processor memory operations is analyzed, and pointer chasing operations' problems of low data prefetching accuracy and long memory access latency in linked data applications are pointed out. To improve processors' pointer chasing access performance, an instruction label assisted memory prefetching (ILAMP) technology is proposed. The technology is a prefetching mechanism under instruction label prompting. It adds a new access instruction to the instruction set architecture to make the instruction generate a special access label in the stage of decoding to demonstrate that the access operations' loading content is the pointer. The label can be sent through the memory hierarchy to the memory controller if Cache miss occurs. When the required pointer comes back to the memory controller from DRAM, the logic can make a prefetching operation immediately hide memory requests latency in the future. Experimental results show that the ILAMP technology can reduce the memory latency by 15% on average. The prefetching accuracy is above 77% for all programs. The bandwidth overhead increases only about 10% and the footprint is about 1kB.

**Key words**: linked data, pointer chasing, instruction label, data prefetching, enhanced memory controller