摘 要（小二号黑体）

对于非规则访存的应用程序, 在运行时会发生严重的 Cache 访问缺失。采用帮助线程将数据预取到离 CPU 更近的 Cache, 可以有效提高多核系统的性能。当某个应用程序的访存开销大于计算开销时, 传统的帮助线程的访存开销会高于主线程的计算开销, 从而导致帮助线程落后于主线程。 我们提出了一种改进的基于参数控制的帮助线程预取模型, 该模型采用梯度下降算法对控制参数求解最优值，从而有效地控制帮助线程与主线程的访存任务量，使帮助线程领先于主线程。实验表明， 基于参数选择的线程预取模型能获得 1.2～1.5倍的系统性能加速比。（小四宋体）  
**关键词**：**数据预取 帮助线程 多核系统 访存延迟 梯度下降**（四号宋体加粗，词组间用空格分开）

**ABSTRACT**

The applications with irregular accessing memory would causes serious Cache in the run-time. Helper thread is an effective technology to improve performance of multicore systems. Helper thread pre-fetches data from Memory to the Cache which is the closest one to CPU. If the overhead of memory access for a given application is far greater than that of computation, it would make helper thread lag behind the main thread. Hereby, we propose an improved helper thread pre-fetching model by control parameters. Furthermore, gradient descent algorithm is one of the machine learning algorithm which is adopted here to determine the optimal control parameters. The amount of the memory access tasks are controlled by the control parameters effectively, which makes helper thread be implemented ahead of main thread. The experiment results show that the speedup of the application are achieved by 1.2 times to 1.5 times.

**Key Word**: **data pre-fetch, helper thread, multi-core system, memory latency, gradient descent**

# 目 录（小二黑体）

中文摘要

**ABSTRACT**

目 录（小二黑体） 3

第一章 绪 论（小二黑体） 4

1.1 课题的研究背景和意义 4

1.2 国内外研究现状 5

1.3 课题研究的内容和主要工作 5

1.4 论文结构 6

第二章 相关研究工作 6

2.1 预取要考虑的要素（KPB） 6

2.2 硬件预取 7

2.3 软件预取 7

2.4 软硬件结合预取 7

2.5 总结 7

第三章 非规则数据预取技术研究 8

3.1 非规则应用程序的特点 8

3.2 帮助线程预取分析 8

3.3 预取算法 9

第四章 预取算法实现 9

4.1 预取参数的定义 9

4.2 预取算法-梯度下降 10

4.2.1 梯度下降算法 10

4.2.2 预取算法流程图 10

第五章 实验验证 11

5.1 实验环境 11

5.2 实验benchmark 11

5.2.1 benchmark介绍 11

5.2.2 benchmark实验结果分析 12

第六章 总结与展望 13

# 第一章 绪 论（小二黑体）

说明课题的来由及现状，本文要解决的问题，采用的方法及手段，成果及意  
义，可根据本人的课题不同加以增添。（小四宋体）

## 1.1 课题的研究背景和意义

在微处理器的发展进入多核时代之后，虽然存储器的访问速度也在逐渐提高，但是处理器的运算速度远远超过了存储器的访问速度，计算和访存的速度差距进一步拉大，存储墙[1]仍然是制约微处理器性能提升的一个重要瓶颈。

为此，人们提出了很多缓解存储墙问题的方法：一是使用更大更快的存储器,缩短访问时间。二是采用多级Cache存储结构，在处理器和主存储器之间加入多个层次的 Cache，利用程序运行时对数据的访问具有局部性的特征，将局部用到的数据提前缓存到Cache，从而提高命中率。虽然局部性优化技术可以改善程序的访存性能,却并不能改变命中时间和失效损失,本质上只是减少Cache失效次数。而且访存的局部性受多种因素影响,极易变化，对源代码或者机器平台的轻微改动就会导致局部性发生巨大变化；三是利用数据预取隐藏访存延迟，在存储器访问的同时让处理器进行有用的工作，开发时间并行性。它和局部性优化技术不同,局部性优化的目标是减少主存访问的次数,而数据预取技术则是隐藏访存延迟。诸如非阻塞Cache、软件预取、硬件预取和流缓冲器等技术,它们将访存和计算重叠起来,隐藏了访存延迟,提升了访存性能。预取是建立在利用存储器的空闲频带的基础上的,数据预取操作的目的是提前将数据从远离处理器的存储层次上移动到靠近处理器的存储层次上,甚至是寄存器中,以减少数据使用时的访存延迟等待。预取操作和处理器的计算过程并发进行,存储系统提前一定的时钟周期将数据从主存读取到Cache中。理想情况下,数据预取能够为微处理器及时提供所需的数据,从而避免处理器阻塞。

其中数据预取是缓解存储墙问题的重要手段之一。数据预取[2]技术利用访存和计算的重叠，在处理器访问数据之前预测其访存地址并提前发出访存请求，隐藏因Cache缺失而引起的访存延迟。同时预取的负面效应也是必须考虑的因素，例如Cache污染和带宽的浪费。

克服cache缺失问题已经成为微处理器设计的关键。到目前为止,人们提出了各种方案以减少或者隐藏由于cache失效带来的访问存储器的延时。从硬件设计的角度,目前普遍采用的有增加指令Cache块大小、硬件顺序预取、基于分支预测的预取以及Trace Cache。从编译器和处理器通信的角度,人们又提出由编译器控制的预取技术,在编译时加入特殊的预取指令,在指令被用到之前发出预取请求。

传统的数据预取可分为软件预取和硬件预取两种类型[2]。硬件预取[3,4]在预取引擎的控制下对程序访存的模式进行识别和预测，通过专用的硬件机制来预测可能会发生的失效，自动地进行预取,增加了硬件复杂性。软件预取[5]是指由程序员或编译器在代码中适当的位置插入预取指令，提前将数据取入 Cache，从而避免在计算的时候由于数据缺失而导致的执行暂停。

## 1.2 国内外研究现状

目前，基于帮助线程数据预取技术研究已经非常普遍。研究者从不同平台、不同角度等各方面都进行了大量的研究。帮助线程的数据预取技术，目的是利用空闲的硬件上下文执行数据预取线程，提高主线程的执行效率。帮助线程与主线程并行执行，并提前将主线程需要的数据从内存预取到 Cache 中，减少主线程的访存时间，提高系统性能。

帮助线程技术可以通过硬件与软件的方法实现[7]。硬件方法需要在处理器中增加相关的硬件，通过指令窗口动态生成帮助线程，硬件复杂度比较高。软件方法是对程序的源代码进行剖析，利用语义信息，由编译器在源代码级显式插入预取线程代码，易于实现。

Kim等人[8]利用Unravel切片工具和斯坦福大学SUIF编译框架在源代码级完成了帮助线程的自动构造。他们利用PV操作来进行主线程和帮助线程之间的同步，利用全局计数器与局部计数器控制控制帮助线程的速度，只有当两个计数器之间的差异大于一个特定的阈值PD（预取距离）时，帮助线程才继续运行。Yong Hong Song等人[9]在SUNSPARC平台上基于编译实现了帮助线程的构造方法，他通过判断帮助线程的收益来进行构造。Ou Guo dong提出的基于线程的数据预取方法[10]，在处理器上添加动态预取线程构造逻辑和预取线程控制逻辑来分析应用程序中访存行为的特点，从主线程的执行行踪中提取数据预取线程，使用空闲的现场和主线程并行执行。Yu Ji yang提出了一种线程感知的自适应的数据预取方法[11]，根据线程动态反馈信息将线程进行分类，计算个线程的访存特性和预取特性指标，从硬件层面控制线程的竞争，但是需要物理模块的支持。

以上对帮助线程预取的研究大多集中于主线程和帮助线程的构造与同步机制。但是，对于实际的应用程序，在应用程序计算开销很小的情况下，帮助线程不一定总快于主线程，从而导致帮助线程落后于主线程。此时就会频繁产生同步操作，导致系统性能下降。因此，在特定的情况下，主线程如果能承担一部分访存任务，从而使得帮助线程与主线程负载均衡，提高系统访存和计算性能。

## 1.3 课题研究的内容和主要工作

本文是面向非规则计算的帮助线程技术研究，帮助线程的访存开销分为两种情况，第一，对于计算密集型的应用程序，帮助线程承担全部的访存任务。第二，对于访存密集型的应用程序，帮助线程承担部分的访存任务。如果使帮助线程取得较好的性能，我们必须根据各个程序不同的访存开销与计算开销来调整帮助线程预取数据量的大小。帮助线程从热点程序入口处开始跳过个数据之后才开始推送个数据，从而提高帮助线程预取数据的有效性。在预取的时候，一方面要保证帮助线程能够及时地预取主线程所需要的数据，另一方面要保证帮助线程不会落后或超前于主线程太长的距离，从而替换掉主线程所需的有用数据，造成多核平台的最后一级缓存污染。

本文研究的内容如下：

1. 对帮助线程技术的理论基础进行了介绍和分析；

2. 对前期研究工作进行总结，并在此基础上进行分析和优化；

3. 本文提出一种改进的面向非规则计算的帮助线程预取算法，并在实际机器上进行验证和分析。

## 1.4 论文结构

本文的组织结构如下：

第一章，主要介绍了课题的研究背景和意义，综述国内外的研究现状并确定本课题的主要研究内容。

第二章，主要介绍了相关的研究工作，包括预取要考虑哪些因素以及现有的预取方法的优缺点。

第三章，主要介绍了非规则数据预取的特点，分析了如何使用帮助线程对非规则应用程序进行数据预取。

第四章，提出了一种面向非规则计算的帮助线程预取算法，对现存的预取算法的缺点进行分析，并进行改进，从而提出一种改进的基于梯度学习的预取算法。

第五章，对本文提出的算法进行实验验证，选取具有代表性的基准测试程序进行了实验分析和研究。

第六章，总结与展望，该章节总结了本课题的主要工作的和贡献，同时也对存在的问题进行了反思，最后对后续工作进行了展望。

# 第二章 相关研究工作

## 2.1 预取要考虑的要素（KPB）

对于非规则数据访存引起的cache缺失，通过帮助线程进行数据预取是个很好的选择。帮助线程预取是在cache失效之前提前访问存储器，将数据读取到cache, 在CPU真正需要这些数据时候能够直接从缓存得到，从而减少内存延时和阻塞的时间。帮助线程和主线程的计算过程并发进行,存储系统提前一定的时钟周期将数据从主存读取到Cache中。理想情况下,如图1所示，帮助线程能够为处理器及时提供所需的数据,从而避免访存延迟。但是如何利用帮助线程进行预取，我们要考虑以下三个因素：

1. 帮助线程什么时候进行预取；如果帮助线程预取时间控制不当，就会造成大量无用数据的预取，从而造成带宽的浪费以及cache的污染。预取时机过早或过晚，数据在没有使用前就被替换出去或者是要使用时还没有取回。
2. 帮助线程一次预取多少数据；确定好每次预取的数据量，避免带宽浪费和cache污染。
3. 帮助线程与主线程的同步频次；控制好帮助线程与主线程的同步频次，使帮助线程不能过于超前或者落后于主线程，从而避免大量的无效预取。

Compute

Load

Compute

Load

Compute

Load

源程序：

Compute

Compute

Compute

主线程： Compute：计算操作

Load：访存操作

帮助线程：

Load

Load

图1：理想情况下帮助线程预取

## 2.2 硬件预取

硬件预取[3,4]在预取引擎的控制下对程序访存的模式进行识别和预测，通过专用的硬件机制来预测可能会发生的失效，自动地进行预取,增加了硬件复杂性。

比如根据CPU 或 Cache 的数据访问历史预测未来可能访问的数据，这种方法对空间局部性较好的程序非常有效, 而对于访存行为不规则的应用，基于历史访问表的预测方法不会带来性能上的提升。

## 2.3 软件预取

软件预取[5]是指由程序员或编译器在代码中插入预取指令，提前将数据取入 Cache，从而避免在计算的时候由于数据缺失而导致的执行暂停。

## 2.4 软硬件结合预取

软硬混合预取 ,主要应用在支持多线程的处理器上, 利用硬件支持单独的线程预取数据并用软件的方法实现预取线程与计算线程的同步, 由于需要软件的支持,其移植性较差,无法保证在所有体系结构的处理器上都适用。

## 2.5 总结

基于硬件的数据预取技术已经是一项非常成熟的数据预取技术，它主要针对规则数据执行数据预取。这里的规则数据主要是指要预取的数据相对比较集中，硬件容易预测的数据，而对于不集中的数据（也就是非规则数据），硬件的预取效果就很不理想了，甚至会对缓存造成污染，影响程序的性能。

|  |  |  |
| --- | --- | --- |
| **方案** | **优点** | **缺点** |
| 硬件预取 | 没有指令开销，不需要编译和程序员的支持 | 基于历史信息，不能结合程序的特点，硬件开销大 |
| 软件预取 | 结合程序的特点，预取准确度高 | 增加额外指令开销， |
| 软件硬件结合 | 结合软件硬件预取优点 | 需要软硬件同时给予支持 |

# 第三章 非规则数据预取技术研究

## 3.1 非规则应用程序的特点

非规则计算出现在许多重要的科学计算应用程序中，由于其计算依赖关系和访存模式的非规则性，使得非规则计算程序通常表现出极少的局部性和数据重用、动态非连续存储访问和大量细粒度并行[34]。这类算法通过直接实现或依赖编译器的自动优化一般很难获得高性能，特别是在并行计算机上，而基于线程的数据预取技术解决了这个问题。

基于预取线程的数据预取技术（也称为预执行技术和预计算技术），在研究领域已

经受到了广泛的关注。相对于基于预测的预取技术，预执行技术利用空闲的硬件资源、

独立的预取线程执行长延迟的访存代码，并在主程序使用数据前将数据预取到离处理

器更近的存储层次，达到预取主线程所需数据的目的，进而减少了主线程访问内存的

次数。尤其重要的是，预取的线程先于主线程执行，并在主线程需要数据前触发缓存

缺失，从而隐藏主线程的访存延迟。

## 3.2 帮助线程预取分析

帮助线程预取技术实质上是一种Leader/Follower结构，帮助线程是剔除了原有线程计算任务的“精简版本”，它往往比原有线程运行得快，因此帮助线程可提前于主线程发出长延迟访存请求，从而达到加速程序执行速度的目的[12]。帮助线程仅仅起到预取的作用，不修改主线程的体系结构状态，因此不会引起程序的错误执行。在理想的情况下，主线程需要某个数据的时候，帮助线程恰好能将需要的数据预取到LLC。但是，如果访存开销和计算开销差别较大的时候，帮助线程并不能每次领先主线程，从而导致预取的数据不能及时到达LLC，还可能造成Cache污染。根据不同程序中访存开销和计算开销的规模，可将程序划分为以下三种类别。由于非规则访存带来大量的访存开销，使得程序的访存开销远大于计算开销，因此本文着重针对第二种模型提出参数控制的帮助线程预取方法。设访存时间为，计算时间为。

1. 计算开销与访存开销大小相当,即。此时帮助线程能够很好地发挥作用。
2. 计算开销大于访存开销，即。此时要控制好帮助线程的预取时机，防止过早预取，从而导致真正使用的时候数据已被替换出去。
3. 计算开销小于访存开销，即。此时主线程计算开销小，帮助线程访存开销大，主线程需要数据时候帮助线程还没有预取到，主线程可能要进行多次同步操作，等待数据的到达。

对于非规则数据的计算密集型应用程序，通常使用图、树或者链表等数据结构，其访存行为呈现非规则性，访存模式难以在静态编译阶段进行准确预测[6]。由于其可利用的局部性受到限制，使得传统的软硬件预取方法失效，其访存模式只能通过执行代码本身来进行预测[6]。本文采用帮助线程预取技术能够有效处理非规则访存程序。帮助线程负责访存任务，主线程负责计算任务。帮助线程提前将主线程所需的数据预取到LLC（Last-level cache）末级缓存，从而达到隐藏访存延迟的目的。我们采用梯度下降算法确定帮助线程与主线程的访存开销，通过参数控制的方法使得多线程预取程序的性能达到最优。

## 3.3 预取算法

# 第四章 预取算法实现

## 4.1 预取参数的定义

我们采用GU ZhiMing等人提出的KPB[13]（Skip-Push-Block）参数,在考虑原程序计算访存工作量的前提下，通过动态调整K、P、B三个参数值，使得帮助线程的性能达到最优。预取模型如代码1所示。

(1)K即skip，表示跳过多少个数据，即主线程负责skip个数据的访存，其他的访存任务由帮助线程来完成，此参数主要用于控制帮助线程预取的触发时机。若程序的计算开销远远大于仿存量，此时，与传统的帮助线程预取机制一样，帮助线程承担全部的访存工作。

(2)P即push，表示帮助线程给主线程推送多少个数据，即帮助线程预取的数据量。此参数用于控制帮助线程预取工作量的大小。

(3)B即block，表示帮助线程与主线程多长时间同步一次。一般情况下。此参数用于控制帮助线程与主线程的同步频次，本文采用文献[9]所述的线程间同步机制。

While（i<n）

{ //帮助线程跳过skip个数据，主线程负责skip个数据的访存与计算

For（j=0；j<skip; j++）

{

Load（data[i]）；

Computer（data[i]）；

}

//帮助线程预取prefetchSize个数据

For（k=skip；k<prefetchSize+skip; k++）

{

Prefetch(data[k]);

}

}

代码1：KPB帮助线程模型伪代码

## 4.2 预取算法-梯度下降

### 4.2.1 梯度下降算法

目前参数的选取大多都是通过枚举实验来获取，设，其中为预取率，，选取符合范围的K,P进行多次试验，最后选取实验中CPI最小的K,P参数组合即为最优的。



简单介绍梯度下降算法 。。。。。。。。

### 4.2.2 预取算法流程图

基于梯度学习的参数控制预取模型主要由两部分组成，热点分析和代价函数的构造。预取算法的基本流程如图1所示。

（1）确定程序的热点部分

我们采用Intel 的性能分析工具Vtune[14]对程序进行离线Profiling，收集CPU的时钟周期和共享Cache的缺失信息，找出引起Cache缺失的长延迟访存指令，确定要进行预取的热点循环部分。

（2）构造代价函数

通过Vtune工具，分析热点程序的访存任务量与计算任务量的大小。为确保程序性能最优，将满足的条件设为代价函数,记作。



图1 参数控制的预取模型流程图

根据预取模型，主线程负责的任务是：（I）个数据的访存与计算 (II)个数据的计算，可记做,帮助线程负责的任务为：（I）个数据的预取，可记做。理想情况下，帮助线程与主线程完全并行，此时程序性能达到最优，即



 (1)



的绝对值最小。假设给主线程分配的访存任务为，根据经验可知，随着的增加，也增加，，的关系可设为公式(2):

(2)

假设帮助线程分配的访存任务为，同样，根据经验可知，随着的增加，也增加，，关系可设为公式(3):

 (3)



我们将(2) (3)代入(1)得, 根据上述推断，可知代函数:



 (4)



(3)计算最优的，



梯度学习作为一种求解最优参数的迭代算法，广泛应用于机器学习各式model参数的求解中。梯度下降算法是一种迭代方法，先随意选取初始，然后不断的以梯度的方向修正，最终使收敛到最小。因此，本文选择梯度下降算法进行最优值的求解，通过选择不同的, 的样本值，我们可以训练出满足代价函数最小的,。



我们通过，即

 (5)



我们又有

 (6)



通过公式(5)(6)，我们可以得出近似最优解，的值。由于求解的，可能为小数，因此我们可以在近似最优解附近,选三组整数数据进行测试，通过运行程序，使用Vtune分析实验结果，得到**CPI（Clocks per Instruction）**最小的，即为最优解。



# 第五章 实验验证

## 5.1 实验环境

本文实验选取的测试程序为Olden Benchmark中用于科学计算的测试程序 EM3D (Electromagnetic wave propagation in a 3D object)、MST(Minimum spanning tree of a graph)，SPEC CPU 2006中的MCF进行帮助线程预取性能的评估。处理器是 Intel® Core™ 2 Q6600 四核处理器，该处理器共有8MB的二级高速缓存，每对核共享4MB的二级高速缓存。通过Vtune的分析，本文选取的热点模块以及输入集如表1所示，分别为EM3D中的fill\_from\_field，MST中的hashLookup，MCF中的refresh\_potential模块。

## 5.2 实验benchmark

### 5.2.1 benchmark介绍

表1 Benchmark 参数配置表

Tab.1 Benchmark parameter setting

|  |  |  |
| --- | --- | --- |
| Benchmark | 热点函数(Hot function) | 输入(Input) |
| EM3D | Fill\_from\_fields | 400000nodes , arity 128 |
| MCF | Refresh\_potential | Ref |
| MST | Hashlookup | 10000 nodes |

### 5.2.2 benchmark实验结果分析

如图2所示，EM3D，MST和MCF测试程序采用传统帮助线程（帮助线程负责全部的访存任务）、参数枚举法和本文的基于梯度学习的参数控制方法（参数学习法）相对于串行执行（不使用帮助线程的源程序）时的性能加速比，其中参数学习法获得了1.2-1.5倍的最高加速比。MST的Hashlookup模块属于访存密集型程序，使用传统的帮助线程方法与原串行程序相比加速比反而降低了4.8%；使用参数学习方法，性能提升了近50%。EM3D 的Fill\_from\_fields模块属于计算密集型的程序，使用参数学习方法与传统帮助线程方法获得的加速比相当，仅提高了4.9%。由于参数枚举法取决于经验与启发式实验，枚举粒度的大小直接影响到结果的准确性。粒度过小，需要进行大量的重复试验；粒度过大，可能错过最优值。因此，参数枚举法并不总能得到最优解。参数学习法不依赖与经验，而是通过机器学习的方法获取最优值，比参数枚举法效率更高。

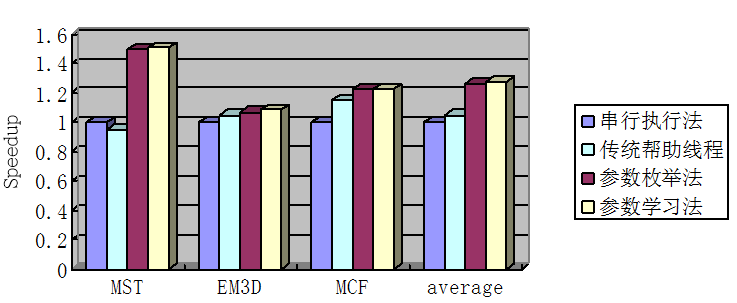


图2 性能加速比

Fig.2 Performance speedup of different methods

图3给出了测试程序在传统帮助线程、参数枚举法和参数学习法情况下的Cache缺失率的相对值。其中，MST，EM3D，MCF的热点程序的Cache缺失率相对于采用传统帮助线程的情况下分别减少了12%，10%，27%。MST，EM3D相对于参数枚举法分别减少了2.5%，1.7%。因此，通过机器学习的梯度下降算法取得K、P的最优值比参数枚举法效率更高。

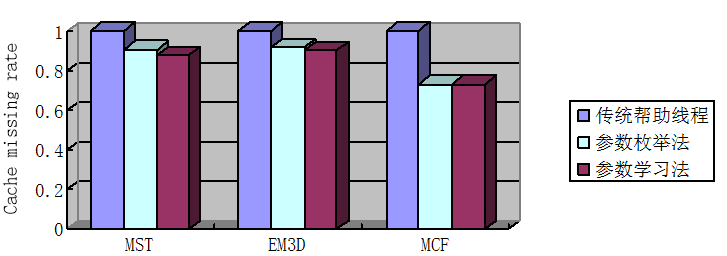


图3 Cache缺失率

Fig.3 Cache missing rate

由于活动计算核(Active Cores)数量的增加以及对资源的竞争，采用帮助线程的程序执行相比于串行程序执行，将会在一定程度上增加功耗。如果帮助线程的收益大于额外增加的功耗，则体现了帮助线程的有效性。帮助线程额外增加的功耗相对于串行程序执行功耗[15]的比例。帮助线程收益表示相对于串行程序执行时间所减少的比例。如图4所示，帮助线程的平均收益大于平均功耗。其中，MST、MCF的收益均大于功耗，因此帮助线程能有效提升MST、MCF的执行性能。因为EM3D 属于计算密集型程序，访存量很小，帮助线程反而带来了很大的同步开销，不足以弥补帮助线程带来的收益，所以，帮助线程对提高EM3D执行性能的效果有限。

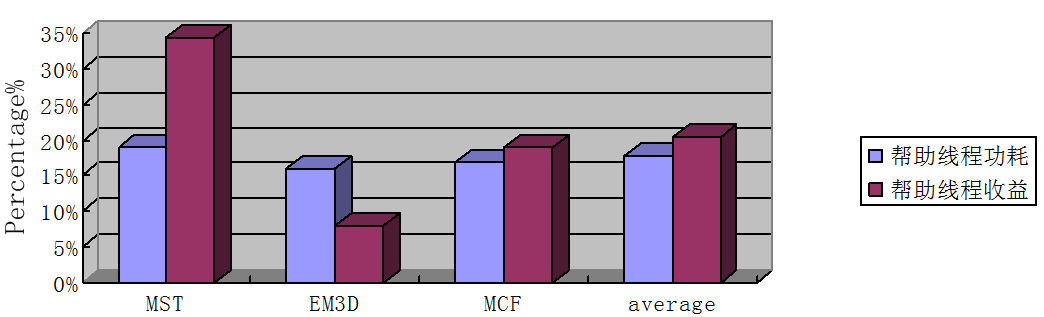


图4 帮助线程功耗和收益比

Fig.4 Comparison between the energy overhead ratio and the performance gain of the helper thread

# 第六章 总结与展望

通过分析传统的帮助线程不能有效的控制预取实时性和覆盖率缺陷，以及对访存密集型的程序预取效率低的劣势，本文提出了一种基于梯度学习的参数控制帮助线程预取模型。通过采用机器学习的梯度下降算法确定K、P的值，根据K、P的值选择性地预取部分数据，使得帮助线程与主线程的工作量相对均衡，从而使程序的执行性能达到最优。

由于帮助线程与主线程同时访存，可能引起带宽的竞争。因此，下一步的工作将考虑帮助线程对带宽的影响，具体分析程序的访存地址，适当增加预取步长，将预取相邻地址的预取指令进行合并，减少预取次数，从而可以降低带宽的竞争，提高执行性能，降低额外功耗。