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**

[第一章 绪 论（小二黑体） 4](#_Toc443491428)

[1.1 课题的研究背景和意义 4](#_Toc443491429)

[1.2 国内外研究现状 5](#_Toc443491430)

[1.3 课题研究的内容和主要工作 6](#_Toc443491431)

[1.4 论文结构 6](#_Toc443491432)

[第二章 相关研究工作 7](#_Toc443491433)

[2.1 预取要考虑的要素（KPB） 7](#_Toc443491434)

[2.2 硬件预取 7](#_Toc443491435)

[2.3 软件预取 7](#_Toc443491436)

[2.4 软硬件结合预取 7](#_Toc443491437)

[2.5 总结 7](#_Toc443491438)

[第三章 非规则数据预取技术研究 7](#_Toc443491439)

[3.1 非规则应用程序的特点 7](#_Toc443491440)

[3.2 预取分析 7](#_Toc443491441)

[3.3 预取算法 7](#_Toc443491442)

[第四章 预取算法实现 7](#_Toc443491443)

[4.1 算法流程图 7](#_Toc443491444)

[4.2 主要算法-梯度下降 7](#_Toc443491445)

[第五章 实验验证 7](#_Toc443491446)

[5.1 实验环境 7](#_Toc443491447)

[5.2 实验benchmark 7](#_Toc443491448)

[5.2.1 benchmark介绍 8](#_Toc443491449)

[5.2.2 benchmark实验结果分析 8](#_Toc443491450)

[第六章 总结与展望 8](#_Toc443491451)

# 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，将局部用到的数据缓存提前缓存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。像非阻塞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。

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。

## 1.2 国内外研究现状

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）

## 2.2 硬件预取

## 2.3 软件预取

## 2.4 软硬件结合预取

## 2.5 总结

# 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

## 3.2 预取分析

## 3.3 预取算法

# 第四章 预取算法实现

## 4.1 算法流程图

本文的算法流程图如下

## 4.2 主要算法-梯度下降

# 第五章 实验验证

## 5.1 实验环境

## 5.2 实验benchmark

## 5.2.1 benchmark介绍

## 5.2.2 benchmark实验结果分析

# 第六章 总结与展望