

分 数:	
评卷人:	

華 中 科 技 大 學

## 研究生数据中心技术课程论文

题 目：使用软件乱序处理技术来突破硬件限制

学 号 D201880731

姓 名 梁 益 泽

专 业 光学工程

课程指导教师 曾 令 仿

院（系、所） 武汉光电国家研究中心

2018 年 11 月 28 日

# Transcending Hardware Limits with Software Out-of-order Processing

## 使用软件乱序处理技术来突破硬件限制

**Authors:** Trevor E. Carlson , Kim-Anh Tran , Alexandra Jimborean , Konstantinos Koukos, Magnus Sjalander, and Stefanos Kaxiras

梁益泽 国光博 1801 班 光学工程 D201880731

本文讲述了使用软件乱序处理来突破硬件限制的 SWOOP 技术，本篇文献阅读报告将从背景综述，作者介绍，文章主要内容，该领域研究进展这四方面展开。

### 一. 背景综述

本文是 2017 年下半年发表在 IEEE Computer Architecture Letters 期刊上的一篇文章。IEEE Computer Architecture Letters 是一个严格的同行评审期刊，简称 LCA,用于在单处理器和多处理器计算机系统，计算机体系结构，微体系结构，工作负载表征，性能评估和模拟技术以及功耗感知计算领域发布早期，高影响力的结果。此外还涉及主题：微处理器和多处理器系统，微体系结构和 ILP 处理器，工作负载表征，性能评估和模拟技术，编译器和操作系统-硬件交互，互连体系结构，内存和缓存系统，架构级别的电源和散热问题，I/O 架构和技术，对先前发布的结果的独立验证，对不成功技术的分析，特定于域的处理架构（例如，嵌入式，图形，网络等），真实时间和高可用性架构，可重配置系统。该期刊为半年刊，17-18 年度最新影响因子为 1.521。

### 二. 作者介绍

本文的作者为 Trevor E. Carlson , Kim-Anh Tran , Alexandra Jimborean , Konstantinos Koukos, Magnus Sjalander, Stefanos Kaxiras。其中，作者 T.E. Carlson, K.-A. Tran, A. Jimborean, K. Koukos, and S. Kaxiras 来自乌普萨拉大学（Uppsala University），M. Sjalander 挂名了乌普萨拉大学和挪威科技大学（Norwegian University of Science and Technology）。

Trevor Carlson 是乌普萨拉大学信息技术学院计算机系统方向的一名博士后学者，他也是本文的第一作者，截至目前已发表 22 篇计算机方向的期刊和会议文章。Magnus Sjalander, Stefanos Kaxiras 是本文中的导师，先说 Magnus Sjalander，他是挪威科技大学(NTNU)计算机科学系副教授，同时也是乌普萨拉大学的高级

客座讲师。他是 NTNU 能源效率计算系统(E ECS)研究项目的共同负责人。Sjalander 拥有 Lulea 大学的理学硕士学位和 Chalmers 理工大学的博士学位。Sjalander 的研究集中在所有规模的高性能和节能系统的设计上。他的专业背景主要是计算机架构和电路设计，曾领导过 FPGA 和 ASIC 技术的实体项目。最近的项目越来越多地关注于硬件/软件协同设计和高效低功耗系统(编译器、体系结构和硬件实现)的高效计算。这些项目背后的驱动思想是，通过从应用程序中提取更多信息并监视系统，可以更好地控制硬件，从而在性能和功率方面更有效地使用资源。

图 1 为 Magnus Sjalander 教授的个人公开信息，Magnus Sjalander 教授在学校官网上为自己设置的三个关键词分别为：计算机结构，节能计算和存储系统。本篇文章中没有 Acknowledgements，但是在查阅 Magnus Sjalander 教授的项目经验可以看到，他目前是 NTNU 能源效率计算系统(E ECS)研究项目的共同负责人，高能源效率的计算正是本文的核心思想之一，所以本篇文章的来源应该正是这一项目。

## Magnus Sjalander




visiting senior lecturer at **Department of Information Technology, Division of Computer Systems**

**Email:** [magnus.sjalander@it.uu.se](mailto:magnus.sjalander@it.uu.se)

**Telephone:** [+4618-471 2987](tel:+4618-4712987)

**Visiting address:** Room POL 1210 ITC, Lagerhyddsv. 2, hus 1

**Postal address:** Box 337  
751 05 UPPSALA

**Keywords:**  computer architecture  energy efficient computing  memory systems


vCard 

图 1 Magnus Sjalander 教授的个人公开信息

Stefanos Kaxiras 是瑞典乌普萨拉大学的正式教授。他拥有威斯康辛大学计算机科学博士学位。1998 年，他加入了贝尔实验室(Lucent)的计算科学中心，后来

又加入了 Agere Systems。2003 年加入希腊帕特拉斯大学欧洲经委会系，2010 年成为瑞典乌普萨拉大学的正式教授。Kaxiras 的研究兴趣集中在内存系统和多处理器/多核系统领域，重点是电力效率。他与人合著了 90 多篇研究论文和 18 项美国专利，获得了两项瑞典 VR 奖助金(VR- frame 奖助金的主要 PI)，参与了 5 个欧洲重大研究项目，目前获得瑞典企业孵化机构和创新机构 VINNOVA 的资助，他是 ACM 杰出的科学家和 IEEE 成员。他在 reasearch gate 上给自己的关键词是存储，能源效率，计算机架构，相干性，多核心，缓存，多核心甚至极多核心（Multicore and Manycore）。

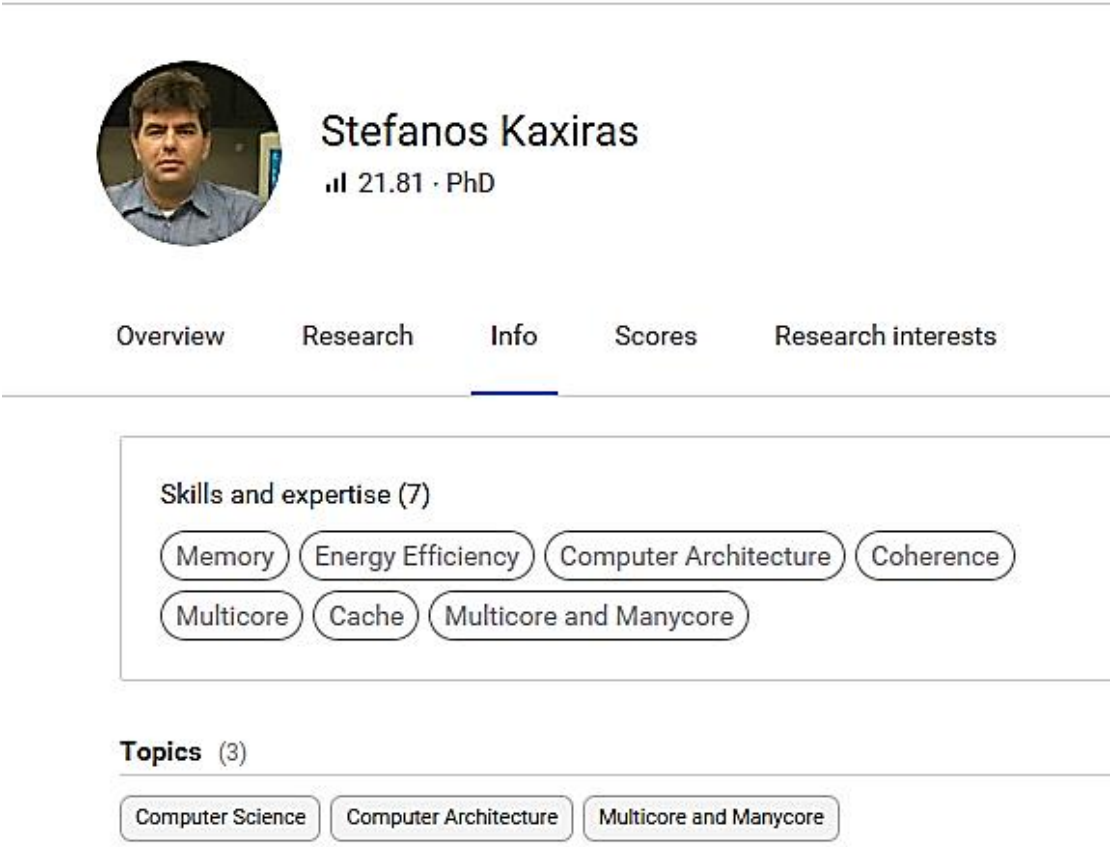


图 2 Stefanos Kaxiras 教授 research gate 主页信息

三. 文章主要内容

本文讲述了使用软件乱序处理来突破硬件限制的 SWOOP 技术，介绍了常规的 InO 和 OoO 技术，并基于目前高性能和高能源效率的需求提出了 SWOOP 技术，并从软件和硬件层面对 SWOOP 技术的实现原理进行了分析，最后通过 sniper 多核模拟器对几种技术进行了对比，下面将分别阐述文章内容。

## 1. 文章背景

新一代高性能处理器对性能和能源效率的同时提出了高的需求，传统的 in-order processors (InO) 依赖于静态指令调度程序，通过在加载和使用之间交错独立的指令来隐藏长时间的延迟，由于动态场景中仍存在高内存延迟，使得其性能大打折扣，用来提高处理器性能的 out-of-order processors (OoO) 技术虽然通过动态排序指令在隐藏内存延迟方面取得了很好的效果，但它缺乏 InO 处理器的高能源效率，综上所述，就需要一种能够兼顾性能和能源效率的处理器技术，那便是本文提出的软件乱序处理技术 (Software Out-of-order Processing)。

SWOOP 通过重新排列指令流以避免停顿 (实现了指令级并行, ILP)，通过 stall-on-use in-order core 和程序乱序执行，提高能源效率 (实现了内存级并行, MLP)，长内存延迟负载期间，核心公布额外的内存和指令级并行性，以执行有用的，非投机性的工作，SWOOP 的操作粒度很细，更喜欢加载 (load) 而不是预读取 (prefetch)。文中与 SWOOP 共同被提及的是分离的访问-执行模型 (DAE), SWOOP 也延用了这种模型中的结构，将复杂的应用程序分解为访问和执行对之后再执行。图 3 中对传统 InO, 一般的 DAE 模型和 SWOOP 技术处理机制的对比，SWOOP 具有隐藏延迟的效果，这个效果会在后文中详细说明。

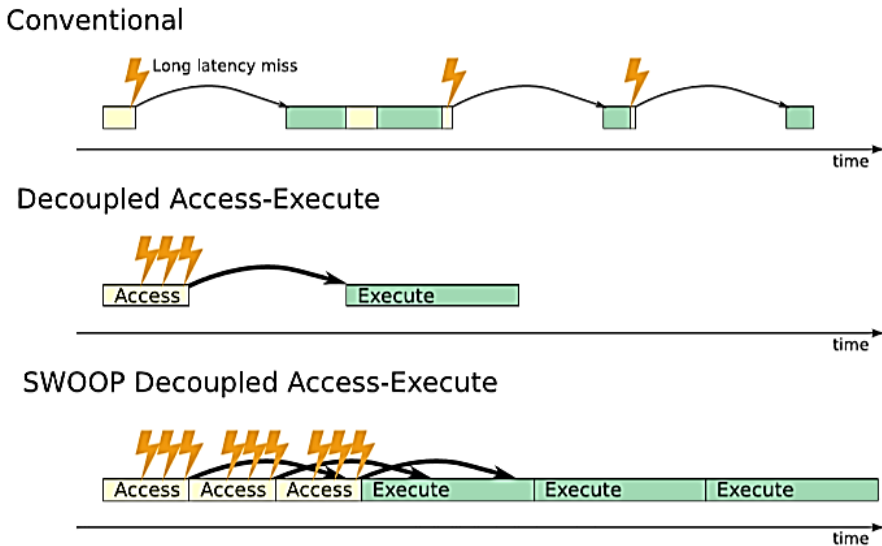


图 3 传统 InO, DAE 和 SWOOP 处理机制对比

## 2. SWOOP 软件组成

SWOOP core 是一种软硬件进行联合设计的处理器，受益于新的软件技术，能够执行传统的动态指令流，通过 SWOOP 的作用，程序中的代码被转化为重内

存绑定阶段（即访问阶段）和高度计算约束阶段。SWOOP 对应用程序的分离方法类似 SW-DAE，将应用程序分解为访问执行对，SWOOP 交错在单个线程中访问和执行代码，动态地改变这种交错。因此，SWOOP 从底层体系结构中抽象出执行顺序，并可以运行顺序或者乱序的访问和执行代码。与传统的 SW-DAE 不同的是在 SWOOP 中，没有 read-after-write 这种依赖关系的将优先访问（hoisted to access），仅为内存依赖的负载被安全预取所取代（safe-prefetches）。

SWOOP 软件技术可以被简化为三个步骤：首先 SWOOP 将关键循环标识为可修改的候选循环，实现的方式包括但不限于用户插入的程序，预先运行剖析步骤，针对 JIT 启动时的步骤。接下来通过将地址计算和负载提前到访问阶段这一手段来实现对访问和执行的分离。访问和执行之间的边界由 chkmiss 标记，chkmiss 引导执行流。Chkmiss 可以指示负载是否在上一级缓存中引发了丢失，这可能会导致内存延迟。当检测到丢失产生后，程序执行跳转到备选执行路径。

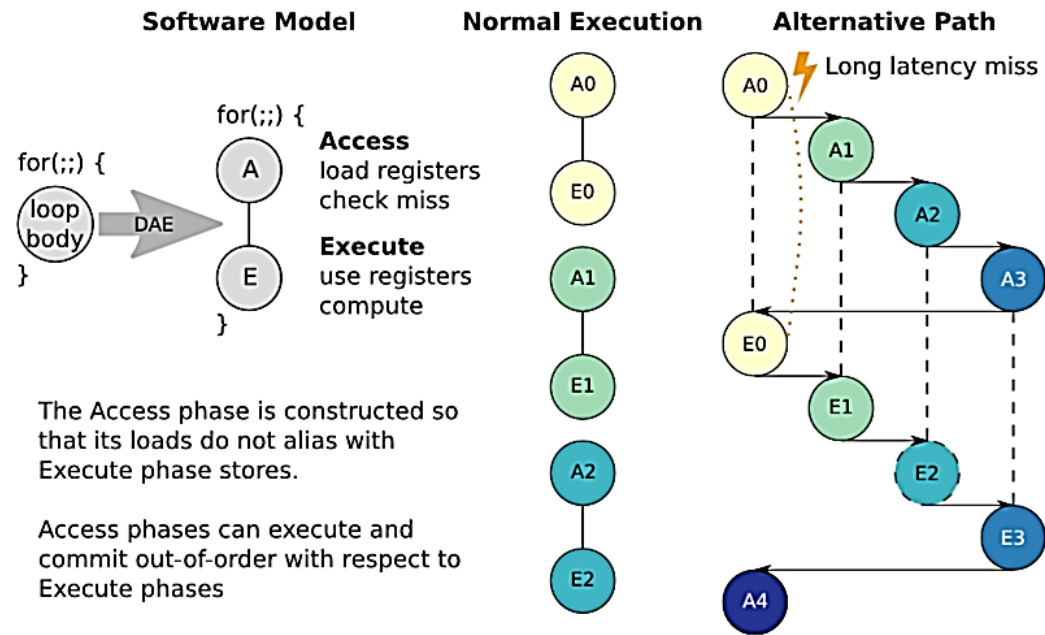


图 4 SWOOP 软件技术中的备选执行路径及常规模式

如图 4 所示为 SWOOP 软件技术中当 chkmiss 检测到缓存丢失进而跳转至的备选执行路径和常规模式的对比。从图中可以看出，当 chkmiss 发现有丢失时，转为如上图所示的备选路径这种特殊设计的路径来执行，原先访问到执行的顺序执行，例如 A0-E0,A1-E1,A2-E2 这样的执行顺序被更改为备选路径中的 A0-A1-A2-A3-E1-E2-E3-E4 来极大地减小访问和执行之间的延迟。

### 3.SWOOP 硬件结构

SWOOP 中的硬件核心, 在传统的 stall-on-use inorder 处理器上增加了三个硬件特性, 分别是 context 寄存器重新映射技术, chkmiss 机制及提前释放负载机制, 这三种硬件特性与软件方法相辅相成, 以求达成更高的性能和能源效率。

Context 寄存器重新映射技术不同于传统情况下使用寄存器文件 Access-Execute (访问-执行) 通信来维护原始优化代码的大部分性能, 它能够混合运行属于不同 SWOOP context 寄存器的访问和执行阶段, 每个 SWOOP 的 context 包括的是它的访问和执行阶段, 采用这样的技术, 减轻了额外寄存器分配的负担, 并为软件提供了额外的物理寄存器。需要注意的是, SWOOP 只在程序乱序执行时进行重新映射, 每个寄存器在访问阶段第一次被写入时只重新映射一次, 在程序执行时不执行额外的重新映射, 判定是否进行重新映射的量为 CTX, 当 CTX=0 时, 不进行重新映射。为了实现这样的 context 寄存器重新映射技术, 设置了 context 重新映射向量和 context 重新映射 FIFO。

Context 寄存器重新映射技术在执行时遵循三条规则, 其一对于新的访问阶段, CTX 自加, 将 context 重新映射向量的相应位设置为 0, 第一次写入 architectural 寄存器时, 相应的位被置位, 新的物理寄存器名称被推至 FIFO 的头部, 将来在此阶段写入相同的架构寄存器不会生成新的映射。新的物理寄存器成为物理映射的新有效架构, 并在未来的访问或执行阶段中被重新映射之前一直使用。当返回到第一个执行阶段 (E0) 时, 将使用 A0 的原始映射。这相当于重映射 FIFO 中最老的寄存器 (将被弹出的元素)。当返回到 CTX = 0 时, 有效映射从 FIFO 中的最年轻 (即头部) 变为最老 (即尾部) 物理寄存器。每个新的 Execute 阶段重新映射向量中的相应位是被置位的, 从 FIFO 中弹出物理寄存器, 尾部的新寄存器是新的有效映射。

Chkmiss 是上文中提到的一种简单而高效的检查遗漏机制, 被用于标记访问和执行的边界, 在产生延迟时 chkmiss 使程序执行转向备选路径, 用以消除上一级缓存丢失引起的内存延迟。

SWOOP 中访问阶段的乱序执行会执行大量指令, 延迟的负载将驻留在提交缓冲区的头部, 导致 SWOOP 核心在缓冲区变满后停止, 为了避免停滞, 使用提前释放负载技术。



#### 4.性能和能效测试

本文中使用了 sniper 多核模拟器对 InO, InO-Unroll, InO-Perf-L3, Runahead,OoO,OoO-Unroll 这几种不同的处理器技术进行了性能和能效上的对比,其中能效上的对比基于 28nm 的 McPAT 1.3 版本,处理器参数如图 5 所示:

TABLE 1 Microarchitecture				
Core	In-Order			OoO
	InO	Runahead	SWOOP	
u-Arch	1.5 GHz, 2-way superscalar			
ROB	-	-	-	32
RS	-	-	-	[16/32]
Phys. Reg	32/32	32/32	96/64	64/64
Br. Penalty	7	7	8	15
Exec. Units	1 int, 1 int/br., 1 mul, 1 fp, 1 ld/st			*
L1-I	32 KB, 8-way LRU			
L1-D	32 KB, 8-way LRU, 4 cycle, 8 MSHRs			
L2 cache	256 KB, 8-way LRU, 8 cycle			
L3 cache	4 MB, 16-way LRU, 30 cycle			
DRAM	7.6 GB/s, 45 ns access latency			
Prefetcher	stride-based, L2, 16 streams			
Technology	28 nm			
<i>OoO has (*) 2 int, 1 int/br., 1 mul, 2 fp, and 2 ld/st.</i>				

图 5 处理器参数

测试得到的性能效果和能源效率分别如图 6, 图 7 所示,可以得出结论,与 InO 相比,SWOOP 在性能上有很大提高,在各个小项目上的性能效果都优于 InO,相对于 InO 处理速率平均提高了 34%;与 InO 相比,SWOOP 在能效上有很大提高,能效平均提高了 23%,是这些方案中唯一一个有效提高能源效率的方案。

下面是一些可能有误的个人见解。其实从两幅图中可以看出,SWOOP 在性能方面的效果是优于 InO 的,在能耗方面更是所有被讨论的方案中最最优秀的,但明显地它的性能方面是不如 OoO,以及使用了其他两种软件技术的 OoO 的,这个问题作者文章中是回避的,也就是说,SWOOP 的优点主要还是体现在能效上面,对于性能优先级比较高的情况下,OoO 应该还是更好的选择,所以 SWOOP 客观来讲,应该是一种能源效率表现十分优秀,但性能介于 InO 和 OoO 之间的方案。



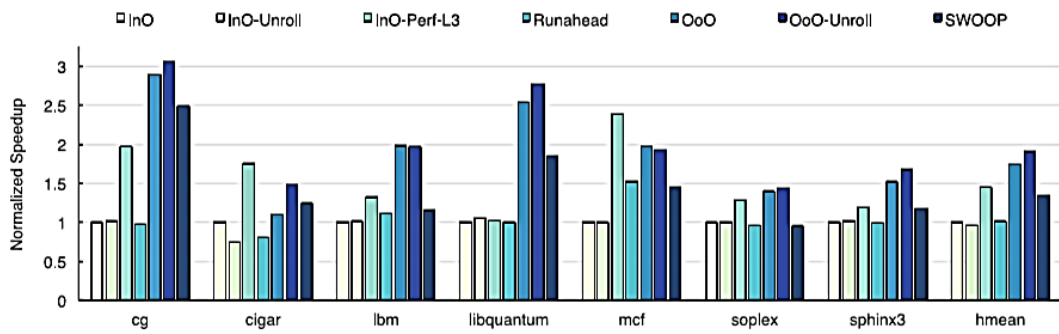


图 6 几种方案的性能比较

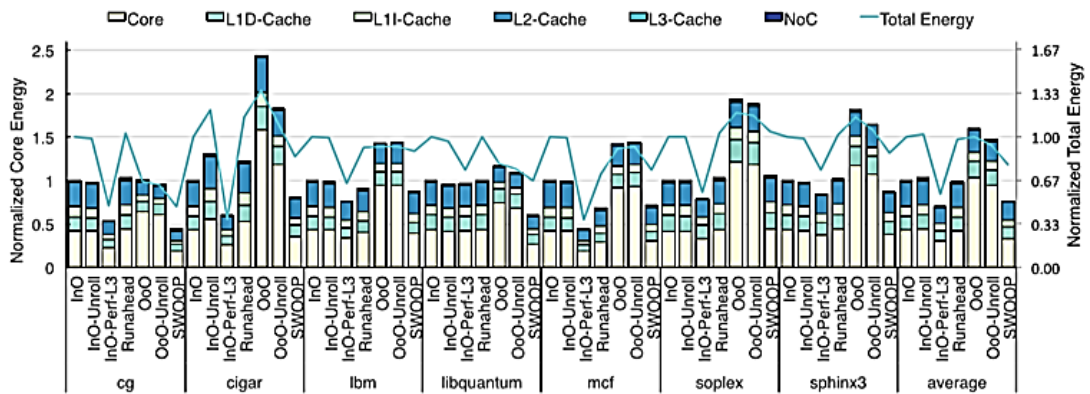


图 7 几种方案的能效比较

#### 四. 该领域研究进展

本课题的来源是高效率的计算项目，所以所处领域应该是高性能和高能效处理器研究领域。文中的 SWOOP 技术，在 SWOOP 对复杂的应用程序分解时，将其分解为访问执行对，这样的思想参考了 2014 年的自己组内的文章（参考文献 1）以及 2016 年自己组内的文章（参考文献 2），第二篇有比较详细的原理说明及代码。文中硬件部分提前释放负载技术（ECL）没有进行过多的描述，只是标注了对普林斯顿大学 Margaret Martonosi 教授 2015 年一篇文章（参考文献 4），该文章介绍了 DeSC(Decoupled Supply-Compute)，这是一种具有良好的可移植性和低复杂性的突破内存瓶颈的方法。文中最后结果比较与 in-order Runahead 进行比较，此处的 in-order Runahead 引用自 1997 年 ACM 中密歇根大学 T. Mudge 教授的文章（参考文献 6），这种技术由于需要重复执行指令和 ozer 等所以性能相比 SWOOP 有劣化。与 SWOOP 技术相比，1995 年普渡大学

T.N. Vijaykumar 教授发表（参考文献 7）的 BBE 硬件方面更为复杂繁琐。同时，作者还将 SWOOP 与 2015 年北卡罗莱纳州立大学 Eric Rotenberg 发表的控制流解耦技术（Control-Flow Decoupling (CFD)）相对比，提出了 CFD 也是类似对长延迟负载进行聚类的方法，但它能处理的负载数量要少得多。与北卡罗莱纳州立大学 T.M. Conte 教授 2005 年发表的技术（参考文献 9）相比，SWOOP 具有不需要使用线程推测的优点。

除此以外，在 Stefanos Kaxiras 教授学校官网主页上，他称他在自己所处领域的第一个突破是 2017 年发表在 ISCA 会议上的文章“Non-Speculative Load-Load Reordering in TSO”，TSO 中的非推测负载-负载重新排序技术。从文章的概要来看，通过对相关层进行重新排序并通过适当地修改典型的目录协议，成功地进行了负载重新排序，并且具有不易受到影响的性能和无负载锁定。与基本情况相比，这样的解决方案具有成本效益，并且可以提高无序保留的性能，如果一致性模式被监管，则不允许提交内存操作的基本情况。

研 究 生 签 字 \_\_\_\_\_