|  |  |
| --- | --- |
| **分 数：** |  |
| **评卷人：** |  |



**研 究 生 ( 数 据 中 心 技 术 ) 课 程 论 文 ( 报 告 )**

**题目：INSIDER——Designing In-Storage Computing System for Emerging High-Performance Drive**

|  |  |
| --- | --- |
| 学 号 | M201973008 |
| 姓 名 | 符传杰 |
| 专 业 | 计算机技术 |
| 课程指导教师 | **曾令仿、施展** |
| 院（系、所） | **武汉光电国家研究中心** |

2019年11月6日

# INSIDER：Designing In-Storage Computing System for Emerging High-Performance Drive

国光硕1905班 M201973008 符传杰

# 课程综述

## 1、背景

本论文由Zhenyuan Ruan发表于2019-USENIX会议，主要由内存智能存储计算研究中心（CRISP）资助完成。文章主要提出当前驱动器和接口性能增速不匹配带来的数据处理瓶颈，作者使用了ISC技术，设计了INSIDER系统从而充分利用SSD的性能，提升数据处理速度。

当今数据量与日俱增，大型公司如FaceBook，在2014年已经存储超过300PB的数据，且以每天600TB的数据量增长。在这样大型的数据上执行数据分析任务，需要设计一个**高效系统**来响应客户端的请求。在数据传输过程中，**驱动器的IO**是极为重要的一环。过去驱动器被认为是数据持久化过程中的一个低速设备，因为驱动器的性能远不及IO总线的性能，成为制约数据传输的主要瓶颈。但在过去的10年里，存储技术有了极大的提升，驱动器的带宽和延迟得到了极大的优化，**新兴非易失存储器（3D-NVM）更是提供了接近DRAM的性能**[1][2],与之相对的，IO总线的发展却相当缓慢，性能没有显著的提高。驱动器内外带宽增速的不匹配，使得我们无法充分利用驱动器的性能。一个有效的解决思路是应用**ISC（In-Storage Computing）技术**移动计算任务到驱动器内部**。**

ISC是一个已经提出来20多年的技术，过去没有得到发展的主要原因有两点:1) 过去在存储器内集成计算单元的成本较高；2）过去存储器内部性能不佳，集成计算单元也无法获得显著收益。与之相对的，现在高端的SSD里面都会集成一个甚至多个CPU，且性能持续提升；其次，接口和驱动器性能增速差距，使得ISC有被挖掘的潜力，这也是驱动作者使用ISC技术的关键。

## 2、研究现状

近年来ISC技术在不同领域都有应用:

[3]Sungchan Kim等人在文章中提到在数据库中应用ISP（In-storage Processing）技术，把计算密集型应用放到存储驱动中处理，对于数据库的两种简单任务，数据扫描（**scan**）和表的连接（**join**）都能起到比较明显的增速效果。

[4] Insoon Jo等人基于新兴的SSD设计了一种低能耗，高性能数据库的**YourSQL，**这是最早的基于ISC技术和NVMe接口SSD设计的数据库。将早期过滤功能和数据库密集查询相结合，低功耗实现大规模查询**，**支持卸载数据库更一般的计算任务到驱动端。

前面的应用在卸载的任务类型上有限制，不能支持通用的计算任务卸载，对于主机端开发者提出挑战。[5]提出近存储器端的处理器架构，设计了一套**辅助主机端卸载**任务到SSD驱动器的**API**，辅助开发人员动态的选择将计算任务卸载到存储驱动侧，充分利用了驱动器端的性能。

在前面的研究[5]中,提到SSD的嵌入式计算元件的性能决定ISC技术的应用效果，[6]中介绍了**lbex**,**基于FPGA**而非CPU来完成计算任务，lbex有效的降低了存储器的能耗，并设计了支持复杂查询计算任务卸载的智能存储引擎。

但是当前的ISC技术存在几个方面的制约：1）**计算能力和可扩展性的不**足；起初集成的ARM性能不足以支持常规的ISC计算任务，而ASIC确实有能力支持ISC的计算任务，但是ASIC一旦被生产出来，只能支持特定的计算任务，不具备可重新编程的能力，不具备扩展性；2）**缺少系统的关键支持**；当前的ISC技术没有过多的考虑数据的访问控制，这就存在不安全的数据访问请求的风险；其次没有涉及多应用共享环境下的资源调度机制，而如果需要充分利用驱动器的性能需要集成多核计算单元，多应用并行场景下的资源调度问题无可回避。3）**缺乏关键抽象**；ISC有自己的API，且不兼容POSIX（可移植系统接口）。这就意味着如果用户想要使用ISC驱动器，就需要对于主机端的代码进行大量的修改，这显然制约了ISC驱动器的发展。

文章作者通过INSIDER系统解决以上三个缺陷。

## 3、系统设计

### 3.1、计算元件选择：

第一个问题是驱动器的计算能力和可扩展性问题。需要计算元件有以下几个特性：1）高可编程性：支持常规的ISC任务；2）高并行：能充分利用驱动的内部带宽；3）高能效：计算单元的加入不能显著提高驱动器能耗；

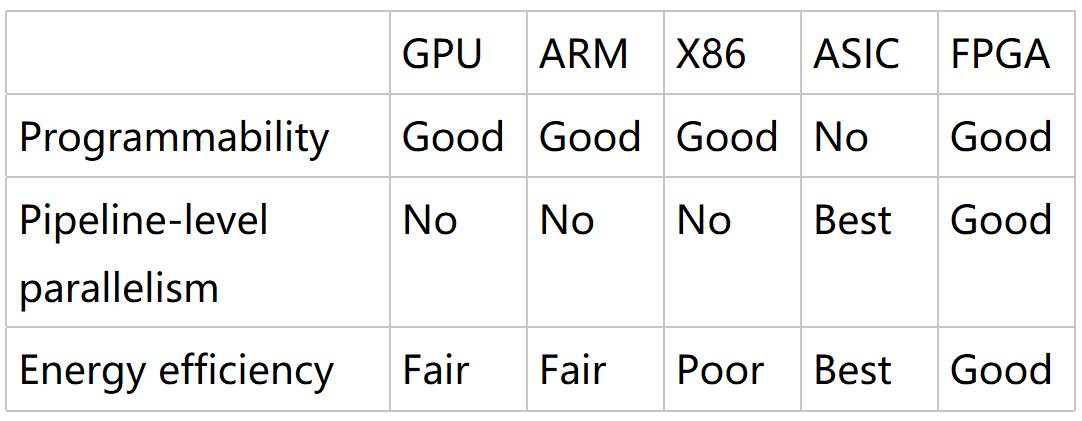


图1、计算单元的性能表现

首先，ASIC一旦制造出来只能处理特定任务，不具备可重编程性能，不能有效的支持多样的ISC任务；其次GPU，ARM，X86处理器都不能支持流水线级并行，这也意味着它们无法提供有效的算力，且它们的引入会降低驱动器的能源利用效率；最后可以看到FPGA是可重配置的，能针对目标负载形成定制架构，能通过SIMD单元实现数据级并行，通过构建深层次的硬件流水线实现流水线级并行，有仅次于ASIC的并行计算和高能效。综合三个因素的考虑，最后选用**FPGA芯片作为驱动器的计算单元[7]**。

### 3.2、系统支持

这里需要提供系统的关键支持，以提供对于数据的**访问控制和资源调度**功能。如图所示：

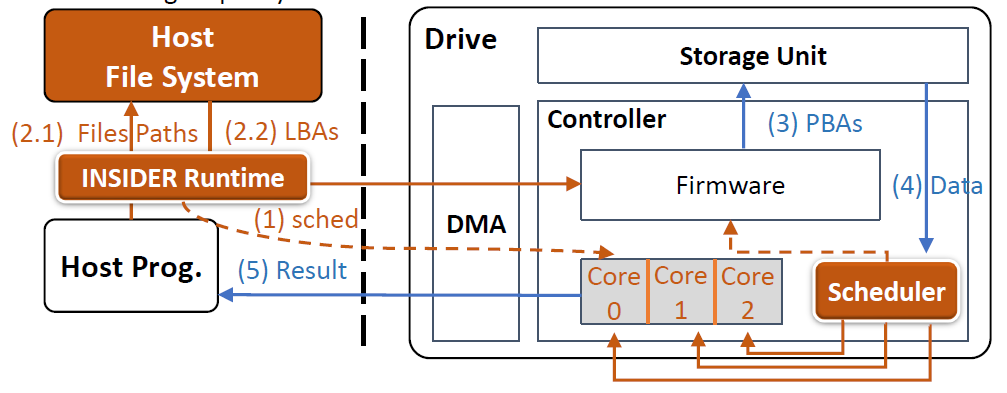


图2、系统架构图

数据的访问控制功能通过主机文件系统实现，主机文件系统接收请求，进行权限检查，通过INSIDER运行组件访问固件获取数据；同时因为FPGA处于数据传输的关键路径上，数据传送到主机端之前会被FPGA拦截并处理。

资源调度功能通过驱动器内部的调度器实现，调度器分发请求以充分利用多核FPGA的性能，同时会把当前计算元件的利用情况反馈给固件，从而动态调节请求速率。具体而言资源调度器通过在固件上扩展多路复用器和ISC单元实现（如图3所示）

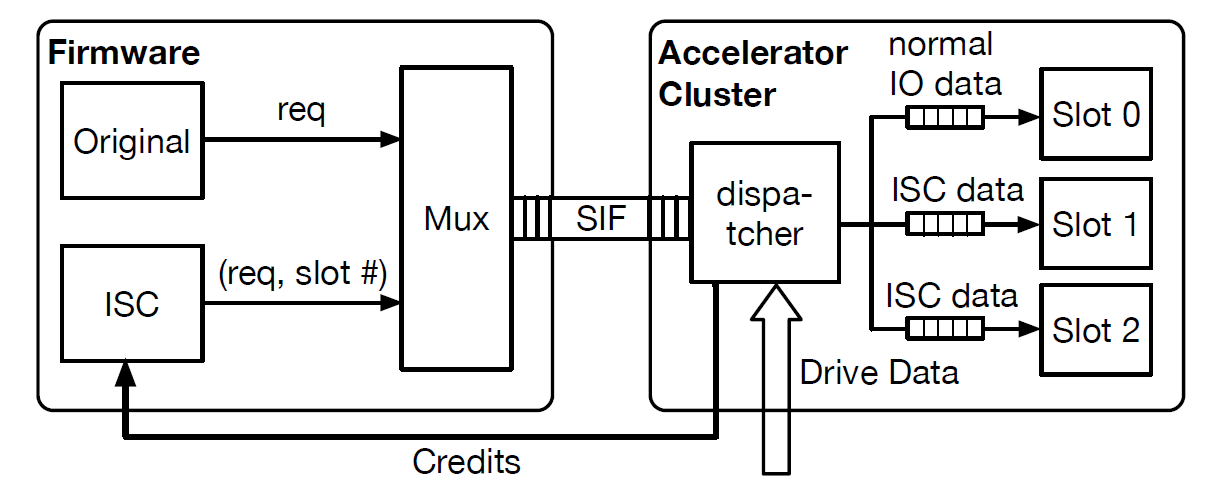


图3、资源调度

ISC接受LBA和加速器插槽索引，多路复用器接受原始固件的请求和ISC的参数，访问相应的存储单元获取数据，存储单元根据不同的Slot分发到不同的加速器单元。特别的，如图所示当加速器索引为0时意外着这是普通的IO请求不经过任意加速器。

### 3.3、系统抽象

系统抽象包括主机端抽象和驱动端抽象。

在主机端，INSIDER将ISC计算模型抽象为简单的文件操作，并提供了一系列的API。下面简述关于主机端执行虚拟文件读取的过程：

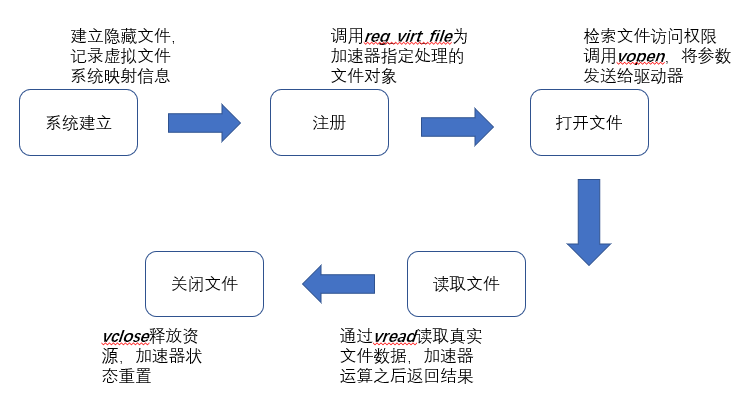


图4、虚拟文件读

1）首先，系统创建一个存储虚拟文件系统映射信息的隐藏文件；

2）通过*reg\_virt\_file*来指定加速器集群处理的文件对象，分配一个当前空闲的加速器；基于真实文件创建一个虚拟文件，写入前述存储映射信息的隐藏文件中。

3）通过*vopen*，首先找到映射的文件，向主机file系统检索文件的访问控制权限，若访问权限检查通过，则使用*send\_params*将加速器集群所需的参数发送至驱动端；

4）通过vread，首先读取文件，在发送至主机端前会被加速器拦截，并处理，然后通过DMA返回数据，

5）最后vclose，资源释放，加速器状态重置可被其他任务使用。

驱动端则只需要关注计算逻辑。加速器元件能拦截并处理数据，而数据的访问控制，数据分发都不需要驱动端开发人员管理，加速器会被动的获取数据并处理。所以驱动端开发人员只需要编写FPGA的计算逻辑。

上述抽象的结果是：主机端开发人员只需要执行简单的文件访问操作，在驱动端可以自动的执行计算逻辑，并返回计算结果；驱动端开发人员也仅需要关注数据的处理逻辑而不考虑数据分发，数据检查。

## 4、评估

为了评估上述设计是否能得到预期的结果，作者针对常见的七种应用（Grep，KNN，位图压缩等）进行了评估。

### 4.1、系统设计过程拆解

作者首先拆解了设计过程，将传统仅主机计算，使用POSIX接口，通过OpenMP利用32核CPU性能的方式作为基准，评估不同设计相对于基准方式所取得的收益。第一步，作者使用定制的IO堆栈绕过主机的IO堆栈（**Host-bypass**）；第二步，通过主机端代码重构，实现了流水线级并行，复用了计算时间和文件访问时间（**Host-bypass-Pipeline**）；最后，把计算任务卸载到驱动端（**INSIDER**）。并考虑了不同接口性能（x8和x16的PCIe）下的实验结果。如图5所示：

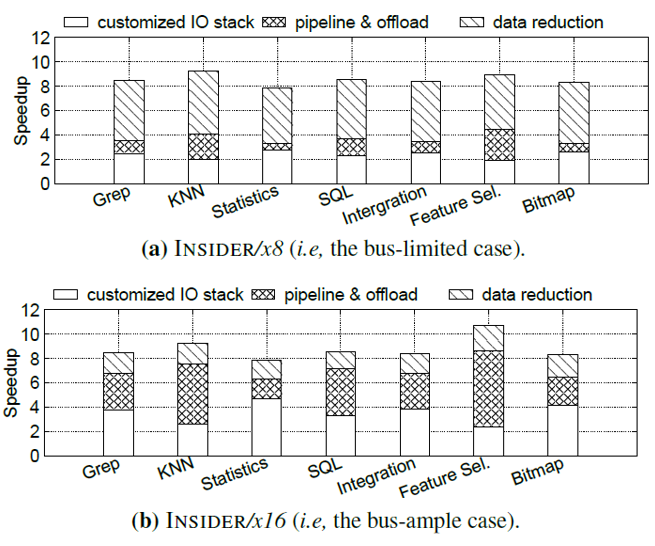


图5、INSIDER系统收益

从图5.(a)可以看出来，使用PCIe 8x时，因为此时接口性能受限，数据缩减部分因为能大大减少传输的数据量，绕开了接口性能的瓶颈，使得在各个应用下数据缩减取得的收益都是最大的；相应的当接口性能提升，使用PCIe 16x时，因为此时互连总线带宽提升，数据去重的收益相应降低，而流水线级别并行和定制IO堆栈是实现增速的主要原因。

### 4.2、瓶颈分析

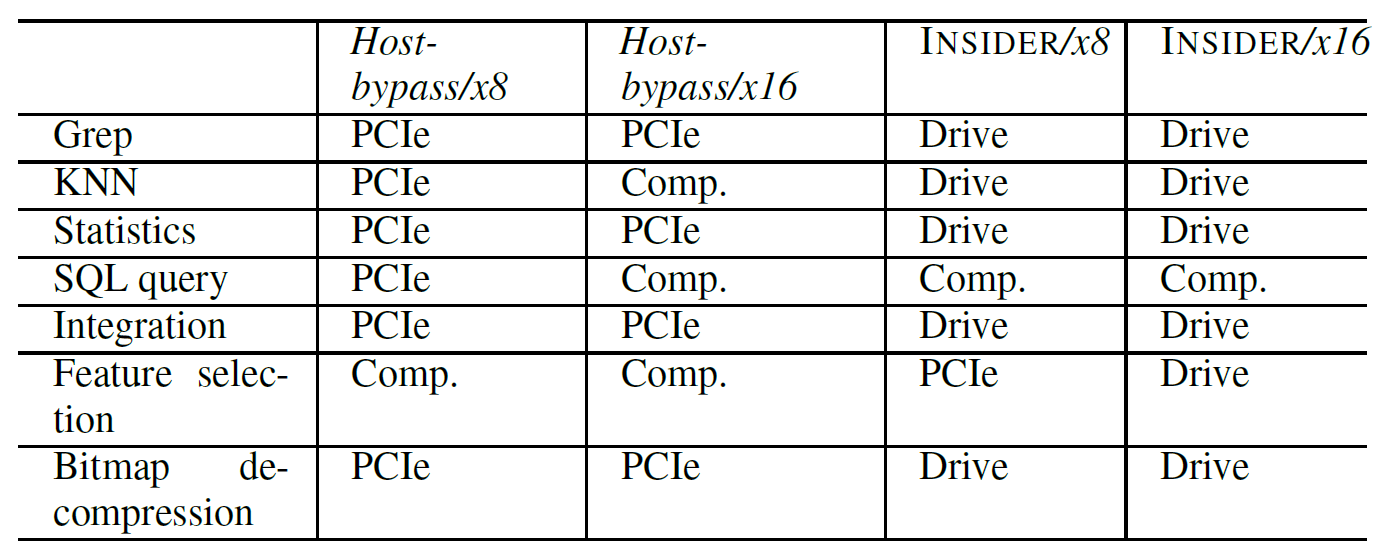


图6、系统瓶颈分析

如图6所示，7种应用在**Host-bypass**设计下的主要瓶颈是PCIe，这也符合一开始所说的驱动器性能高于接口性能。而**INSIDER**利用ISC技术把计算任务卸载到驱动端，传输数据量减少，驱动带宽被充分利用，此时系统瓶颈成为Drive。

同时此图中有几个应用的表现结果不太一致；首先，**KNN和Feature selection**都是计算密集型应用，在PCIe x16的条件下，接口性能较高，主机端的计算能力会制约整体数据处理速度，而INSIDER设计将部分计算任务卸载到驱动端，分担了一部分计算负载，使得整体数据处理速度得到提升；其次就是**Host-bypass**，我们可以看到对于SQL查询的应用，INSIDER的设计没能获得明显的收益。主要原因在于SQL的查询操作不一定设计数据滤除，数据缩减，ISC对于涉及数据缩减的任务能通过减少输出数据量实现性能提升，而不涉及数据缩减的应用会被认为是常规的IO操作，对于这些任务使用ISC技术并不会带来增益。

### 4.3、资源调度评估

作者在这里评估了前面带宽调度的收益，比较了statistics，SQL query和pass-through三种应用并行执行时带宽的利用情况。

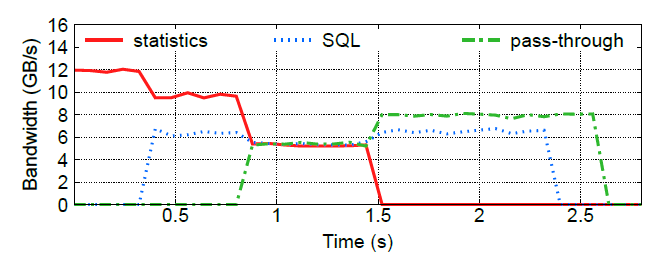


图7 带宽利用情况

首先仅有一个应用（statistics）运行时，没能充分利用上系统的带宽；当另一个应用（SQL query）并行执行时，整体带宽利用率提高，近乎完整利用上了驱动器的带宽，第三个应用（pass-through）的加入也不影响整体带宽利用率，驱动器带宽始终能保持高效的利用，前述资源调度设计能取得预期的效果。

### 4.4、开发时间分析

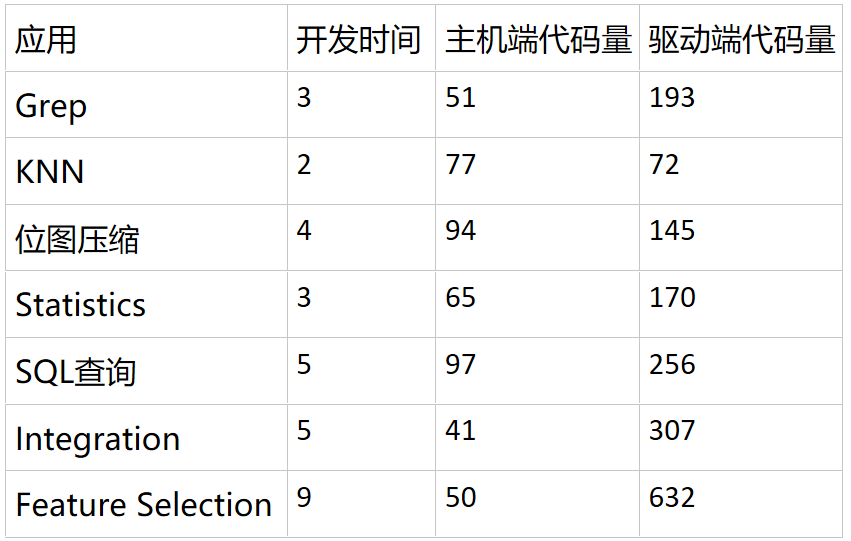


图8、开发时间分析

如图8所示，通过主机端的虚拟文件抽象，主机端代码量只有不到100行，而驱动端代码量相对较高是因为这是FPGA的开发工具不够完善，随着FPGA的开发工具链不断完善，这一部分的时间也会相应缩短。

如图9所示的传统ISC任务开发时间和代码量，仅仅执行简单的IO和权限检查操作都会耗费开发者月余的时间，代码量达到1500多行；而经过INSIDER的系统抽象，开发者开发难度降低，代码量和开发时间显著减少。



图9、传统ISC任务开发代价

### 4.5、计算元件收益

接下来作者测试了不同计算元件（ARM和FPGA）带来的收益：

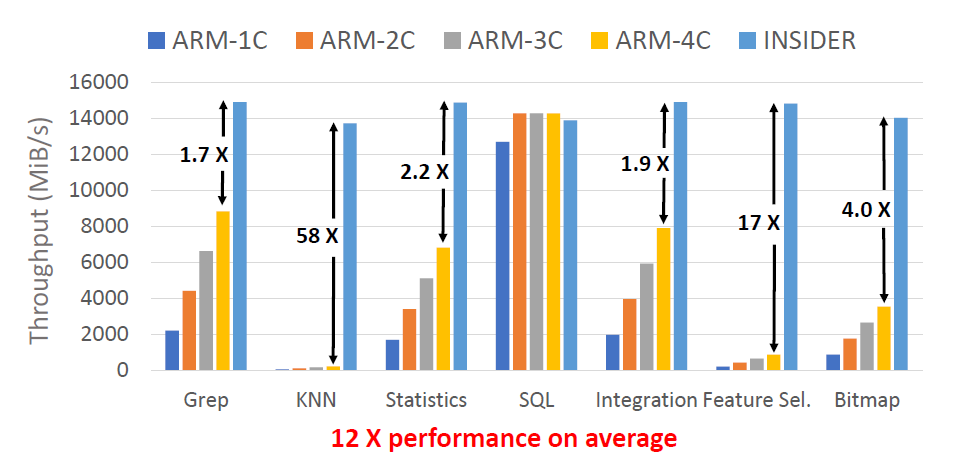


图10、计算元件收益对比

首先在大部分场景下使用FPGA都会比4核ARM的收益更大；对于KNN和Feature selection这种数据密集型应用，优势更明显。ARM没法有效处理密集计算任务，会导致整个数据处理过程卡在驱动端的计算元件处，因此FPGA相对于ARM会有极大的性能提升；而SQL query应用在前述瓶颈分析也提及过，计算量不大，ARM也能支持该应用的计算任务，在仅更换计算元件的情况下是不会带来有效的性能提升的。

## 5、总结

本文作者从存储器技术发展开始，阐述了在新兴存储器场景下，当前数据移动的瓶颈从驱动器端转移到接口侧。[8][9]中介绍了当前新兴NVM的发展趋势，据前述IO总线技术发展的趋势，驱动器内外带宽差距会随着存储技术的发展而日益扩大。为了绕开数据移动的瓶颈，作者通过ISC技术来解决接口瓶颈的问题。应用ISC技术提高数据处理速度的应用近年来日益被关注，但是ISC技术当前发展还不甚完善，存在极大发展潜力，本文作者针对ISC技术存在的几个问题，提出了INSIDER系统，针对性重构了存储系统，扩展了ISC技术的内涵，进行了相应的评估，证明了系统设计过程的收益，提高系统能效，减少开发者工作量并有效利用了驱动器带宽支持主机端的计算任务，减少了数据I/O负担。

## 6、参考文献

1. Fernando P , Kannan S , Gavrilovska A , et al. Phoenix: Memory Speed HPC I/O with NVM[C]// 2016 IEEE 23rd International Conference on High Performance Computing (HiPC). IEEE, 2017.
2. Qian C , Huang L , Xie P , et al. Efficient data management on 3D stacked memory for big data applications[C]// 2015 10th International Design & Test Symposium (IDT). IEEE, 2015.
3. In-storage processing of database scans and joins[J]. Information Sciences, 2016, 327:183-200.
4. Jo I , Bae D H , Yoon A S , et al. YourSQL: a high-performance database system leveraging in-storage computing[J]. Proceedings of the VLDB Endowment, 2016, 9(12):924-935.
5. Koo G , Matam K K , Te I , , et al. Summarizer: trading communication with computing near storage[C]// Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. ACM, 2017.
6. Woods L , Zsolt István, Alonso G . Ibex: an intelligent storage engine with support for advanced SQL offloading[J]. Proceedings of the Vldb Endowment, 2014, 7(11):963-974.
7. Ruan Z , He T , Li B , et al. ST-Accel: A High-Level Programming Platform for Streaming Applications on FPGA[C]// 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, 2018.
8. Chen, An. A review of emerging non-volatile memory (NVM) technologies and applications[J]. Solid-State Electronics, 2016:S0038110116300867.
9. Son Y , Kang H , Han H , et al. An empirical evaluation and analysis of the performance of NVM express solid state drive[J]. Cluster Computing, 2016, 19(3):1541-1553.