软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(9):2723−2740 [doi: 10.13328/j.cnki.jos.005941]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向多读/写头磁畴壁存储器的优化研究*

许 瑞[2], 谷守珍[1], 沙行勉[2], 诸葛晴凤[2], 石 亮[2], 高思远[2]

[1](上海市高可信计算重点实验室(华东师范大学),上海 200062)

[2](华东师范大学 计算机科学与技术学院,上海 200062)

通讯作者: 谷守珍, E-mail: szgu@sei.ecnu.edu.cn

摘 要: 当前,大数据及人工智能技术向嵌入式系统发展,对嵌入式系统的存储访问能力提出了更高的要求.磁畴壁存储器凭借其高读写速度、高密度以及低功耗等优点,可以用于嵌入式系统,以满足数据密集型应用对访问速度、容量及能耗的需求.但是磁畴壁存储器在进行数据访问之前需要进行移动操作,这将极大影响其存储访问性能.而减少移动操作可以有效提升磁畴壁存储器的性能.面向运行数据密集型应用的多读/写头磁畴壁存储器系统,研究减少移动操作的最优指令调度与数据放置技术.首先提出了可获得最小移动次数的整数线性规划(integer linear programming,简称 ILP)模型.由于 ILP 模型不能在多项式时间内求得最优解,所以提出了多项式时间的启发式算法——生成指令调度和数据放置(generation instruction scheduling and data placement,简称 GISDP)算法.实验结果表明,ILP 模型和 GISDP 算法可以有效减少移动操作的次数.在配备 8 个读/写头的磁畴壁存储器上,GISDP 算法生成的指令调度与数据放置方案相较其他算法可以平均减少 89.7%的移动操作,并且 GISDP 算法的结果接近 ILP 模型的最优解.

关键词: 磁畴壁存储器;数据密集型应用;指令调度;数据放置;移动操作

中图法分类号: TP333

中文引用格式: 许瑞,谷守珍,沙行勉,诸葛晴凤,石亮,高思远.面向多读/写头磁畴壁存储器的优化研究.软件学报,2020,31(9):2723−2740. http://www.jos.org.cn/1000-9825/5941.htm

英文引用格式: Xu R, Gu SZ, Sha EHM, Zhuge QF, Shi L, Gao SY. Optimization of multi-port domain wall memory. Ruan Jian Xue Bao/Journal of Software, 2020,31(9):2723−2740 (in Chinese). http://www.jos.org.cn/1000-9825/5941.htm

Optimization of Multi-port Domain Wall Memory

XU Rui[2], GU Shou-Zhen[1], Edwin H-M Sha[2], ZHUGE Qing-Feng[2], SHI Liang[2], GAO Si-Yuan[2]

[1](Shanghai Key Laboratory of Trustworthy Computing (East China Normal University), Shanghai 200062, China)

[2](School of Computer Science and Technology, East China Normal University, Shanghai 200062, China)

Abstract: Nowadays, it has become a trend that embedded systems are designed for big data and artificial intelligence applications, which demand the large capacity and high access performance of memory. Domain wall memory (DWM) is a novel non-volatile memory with high access performance, high density, and low power consumption. Thus, for data-intensive applications specific embedded systems, DWM can meet the requirements of access speed, capacity, and power consumption. However, before accessing data on DWM, data in nanowires need to be shifted to align them with read/write port, which is called shift operation. Numerous shift operations take most of time and generate much quantity of heat when accessing data on DWM. It will decrease the access speed of DWM and system

* 基金项目: 国家自然科学基金(61702187, 61602180, 61972154, 61772092); 上海市扬帆计划(17YF1404400)

Foundation item: National Natural Science Foundation of China (61702187, 61602180, 61972154, 61772092); Shanghai Sailing Program (17YF1404400)

本文由“智能嵌入式系统”专题特约编辑王泉教授、吴中海教授、陈仪香教授、苗启广教授推荐.

收稿时间: 2019-07-01; 修改时间: 2019-08-18; 采用时间: 2019-11-02; jos 在线出版时间: 2020-01-13

CNKI 网络优先出版: 2020-01-14 11:27:01, http://kns.cnki.net/kcms/detail/11.2560.TP.20200114.1126.023.html

performance further. In that case, reducing shift operations of DWM can significantly improve the system performance. This study aims at data-intensive application specific embedded systems with multi-port DWM, and explores optimal instruction schedule and data placement strategy which achieve minimum shift operations. An integer linear programming (ILP) model is firstly proposed to obtain minimum number of shifts. Since ILP model cannot find the optimal solution in polynomial time, a heuristic algorithm is proposed to reduce the number of shifts on DWM—generation instruction scheduling and data placement (GISDP) algorithm. The experimental results show that ILP model and GISDP algorithm can effectively reduce shift operation. On target system with 8 read/write ports DWM, GISDP can reduce shift operations by 89.7% on average when compared with other algorithms, and the results of GISDP are close to the optimal solutions of ILP.

Key words: domain wall memory; data-intensive applications; instruction scheduling; data placement; shift operation

在大数据以及人工智能技术飞速发展的今天,大数据分析技术可以对世间万物所产生的数据进行分析,人工智能中学习算法可以对数据进行学习、分析、总结.目前,由众多嵌入式设备构建的物联网系统中,互联设备之间收集与共享的数据已经广泛使用大数据分析及人工智能技术[1].其中,大数据泛指数据规模达到 TB,甚至 PB 级别,应用均是数据密集型应用,具有高度密集的海量数据读写的特点[2].随着大数据及人工智能应用逐渐向边缘化设备发展,嵌入式设备的数据存储访问性能对系统性能的影响变得尤为重要.高速缓存和便笺式存储器经常应用于嵌入式系统中,从而提升系统的数据访问性能[3].相较于由硬件管理的高速缓存,便笺式存储器具有更高的能量利用率和存储面积[4].它采用软件管理,因此,便笺式存储器上的数据访问行为具有可预测性[5,6].在面向应用的实时嵌入式系统中,采用便笺式存储器在能耗与性能方面更具优势[7].

非易失性存储器(non-volatile memory,简称 NVM)凭借其访问速度快、低功耗、高密度以及字节寻址等优点,被广泛应用在便笺式存储器中[8],已经成为嵌入式系统中备受欢迎的存储技术候选对象[9,10].其中,磁畴壁存储器(domain wall memory,简称 DWM)是一种高密度、低功耗的新型非易失性存储器[11].它采用赛道存储技术,使用磁畴中的磁矩表示数据,利用自旋动量传递的效应,从磁性纳米线中读写数据位[12].磁畴壁存储器的密度比自旋力矩 MRAM 高 4 倍,最佳访问性能可与 SRAM 相媲美,与 DRAM 相比,减少了 92%的泄漏功率[13,14].磁畴壁存储器已经展示了可以替换目前存储器的潜能,例如:文献[15]研究了将磁畴壁存储器作为通用计算平台的片上存储器;在文献[16,17]中,将磁畴壁存储器用作图形图像处理平台的片上存储器使用;文献[18]研究在 AES 平台用磁畴壁存储器替换 SRAM 来提高加密方法的性能.除此之外,还有一种是斯格明子介质的赛道存储器,使用斯格明子表示数据.

虽然磁畴壁存储器具有高密度、低功耗、高读写速度的优势,但是由于赛道存储技术的特点,每次访问数据需要先移动纳米线中的数据,使得要访问的数据与读/写头对齐,然后才能进行数据访问.其中:读写操作均是需要 6.3ns,移动操作需要 5.87ns[5,6].但是,移动操作需要较高的驱动电流,因此,频繁的移动操作会极大地影响磁畴壁存储器的性能与能耗,甚至导致存储单元的损坏[19].特别是对于数据密集型应用,海量的数据访问需求会造成大量的移动操作.由于移动操作的次数可以由数据的放置位置以及访问数据的顺序决定,所以进行合理的数据放置以及指令调度,能够极大地提高磁畴壁存储器的存储访问性能,进而提升系统性能.

本文针对数据密集型应用,面向配备多个读/写头的磁畴壁存储器的单核处理器系统研究最优指令调度与数据放置方案来获得最少的移动操作次数,以提升磁畴壁存储器的存储访问性能,进而提升系统性能.已经有研究工作表明,在磁畴壁存储器上进行数据分配是 NP 完全问题[20],所以本文提出了能够获得最优指令调度与数据放置方案的整数线性规划(integer linear programming,简称 ILP)模型和能够获得近似最优方案的多项式时间的启发式算法.本文的主要贡献包括:

- 提出了可以在配备多个读/写头的磁畴壁存储器上生成最优的指令调度和数据放置方案的 ILP 模型,可以求得最小的移动次数;
- 提出了可以在配备多个读/写头的磁畴壁存储器上,在多项式时间内生成近似最优的指令调度和数据放置(generation instruction scheduling and data placement,简称 GISDP)方案的启发式算法,以此来减小移动次数;
- 对配备不同数量读/写头的磁畴壁存储器的存储访问性能进行了设计探索与实验.

实验结果表明:本文所提出的 ILP 模型和 GISDP 算法,分别能够生成最优和近似最优的指令调度与数据放置方案.与简单指令调度和数据放置(simple instruction scheduling and data placement,简称 SSDP)算法和 Chen 等人[20]提出在固定调度的情况下对数据进行分组放置的 S-DBC-P 算法相比,在配备 2 个读/写头的 DWM 上的实验结果表明:GISDP 相对于 SSDP,移动次数平均减少了 86.7%;GISDP 相对于 S-DBC-P,移动次数平均减少了 79.6%;ILP 模型在 12 小时内求出的局部最优解,相较于 SSDP 算法,移动次数平均减少了 75.0%;相较于 S-DBC-P 算法,移动次数平均减少了 61.8%.在配备 4 个读/写头的 DWM 上的实验结果表明:GISDP 相对于 SSDP,移动次数平均减少了 93.7%;GISDP 相对于 S-DBC-P,移动次数平均减少了 80.7%;ILP 模型在 12 小时内求出的局部最优解,相较于 SSDP 算法,移动次数平均减少了 82.6.0%;相较于 S-DBC-P 算法,移动次数平均减少了 46.3%.在配备 8 个读/写头的 DWM 上的实验结果表明: GISDP 相对于 SSDP,移动次数平均减少了 97.3%;GISDP 相对于 S-DBC-P,移动次数平均减少了 85.6%;ILP 模型相较于 SSDP 算法移动次数平均减少了 94.8%;相较于 S-DBC-P 算法移动次数平均减少了 75.6%.并且 GISDP 算法的结果接近于 ILP 模型的最优解.

本文第 1 节介绍对磁畴壁存储器性能改进的相关工作.在第 2 节中,先对本文目标系统架构进行介绍,再对本文所解决的问题进行定义.第 3 节通过一个示例阐述论文提出的相关算法.第 4 节提出了 ILP 模型与启发式算法.第 5 节展示实验结果并对实验结果进行分析.第 6 节对文章进行总结.

1 相关工作

在已有的工作中,有很多对磁畴壁存储器进行性能提升的研究.他们有从硬件层次对磁畴壁存储器进行改进,也有从软件层次对磁畴壁存储器性能进行提升.

对于硬件层次的优化,在文献[14]中,作者提出了一种电路架构协同设计方案,针对读取数据进行了优化,并提出一种新的缓存组织和管理策略.文献[21]研究了位元布局、磁头定位、纳米线利用率、移动功率、移动延时等电路设计难题,并提出了相应的解决方案.文献[22]使用斯格明子赛道存储器错位存储单元进行了非易失性存储器计算框架的研究与改进.文献[23]通过改变两个重金属衬底的相对厚度来调整电流驱动的畴壁运动.

已有一些在磁畴壁存储器及其他 NVM 上对指令进行调度以及对数据进行分配的研究工作[8,11,24,25].在文献[24]中,Hu 等人讨论了在混合 SPM 的 NVM/SRAM 上的动态数据分配算法.在文献[11]中,Chen 等人提出了在配备多个读/写头的磁畴壁存储器上数据分配的优化技术,但是他们没有考虑同时优化指令调度.文献[8]使用遗传算法对在磁畴壁存储器上的数据进行分配,比较适用于不同访问特性数据集.但是遗传算法在时间上会计算较久,所以对于实时系统不是很适用.在文献[25]中,Gu 等人提出了一种软硬件协同优化方法,以提高应用专用嵌入式系统中磁畴壁存储器的面积效率和性能.但是他们只是针对一个读/写头进行研究,磁畴壁存储器可以具有多个读/写头,并且一定的读/写头数量会对磁畴壁存储器的性能提升有所帮助.

本文针对数据密集型应用,面向配备多个读/写头的磁畴壁存储器的单核处理器系统,研究最优指令调度与数据放置方案来获得最少的移动操作次数,以提升磁畴壁存储器的存储访问性能,进而提升系统性能.

2 架构与问题定义

2.1 架 构

本文的目标系统架构是配备基于赛道存储技术的磁畴壁存储器的单核处理器系统.磁畴壁存储器用作便笺式存储器,可以由软件进行管理.磁畴壁存储器是一种非易失性存储器,它可以用比较低的功耗进行对数据的高速读取.读/写数据的原理是:利用磁畴壁中自旋极化电流与电子磁矩之间的相互作用,由磁畴中磁矩的改变来记录数据位.当要进行读写数据时,根据移动控制器产生的移动信号,电流驱动磁畴壁存储器中所存储的数据整体移动,使得要访问的数据与读/写头对齐,以便数据可以被读/写头进行读写.

在磁畴壁存储器中,数据(即磁畴中的磁矩)存储于磁性介质中.由于在该存储器中,数据的移动类似于磁带的移动,均是数据去找读写磁头进行数据访问,区别是磁畴壁存储器是数据进行移动,磁带是存储数据的介质进

行移动,因此磁性介质在本文中也可以称为磁带.每条磁带上有 n 个域(也可以称为位置),每个域可以存储 1bit 数据.对数据进行访问的端口称为读/写头.本文的磁畴壁存储架构如图 1 所示,包括一条存储多个数据的磁带、两个访问数据的读/写头(分别称为 port0 和 port1)以及一个移动控制器.假设 port0 和 port1 各自拥有访问范围:port0 的访问范围为 0~n/2–1,如图 1 中的域 0~域 3;port1 的访问范围为 n/2~n–1,如图 1 中的域 4~域 7.在该假设条件下,磁带的左边需要冗余 n/2–1 个位置供磁带内数据的移动,避免数据在移动中丢失.假设 port0 和 port1 初始时指向访问范围的最左侧位置处,即分别指在域 0 和域 4 处.移动控制器根据当前读/写头的位置和需要访问数据的地址对读/写头进行选择并生产移动信号.

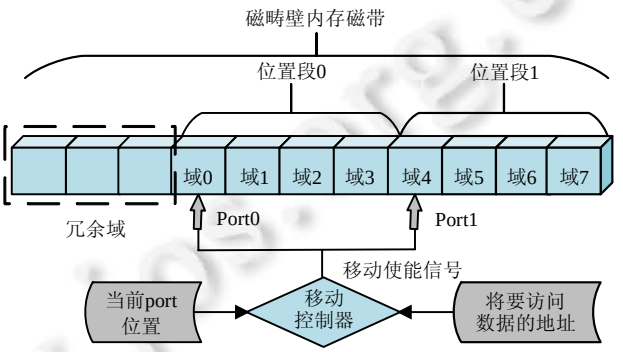


Fig.1 Architecture of domain wall memory

图 1 磁畴壁存储器结构

磁畴壁存储器中,读/写头对数据进行读写操作的速度可以与 SRAM 的读写速度相媲美,然而磁畴壁存储器进行读写操作之前必须将所访问数据移动到读/写头的位置.而磁畴壁存储器中的数据移动是所有数据沿着磁带进行整体移动的,因此移动操作需要花费较长时间并且有较高的功耗.特别是对于数据密集型应用,所需的大量移动操作将会大幅度地降低系统的整体性能.所以移动操作直接影响了磁畴壁存储器的访问性能,进而影响系统性能.本文将磁带中的数据移动一个 bit 距离称为移动一次.在图 1 所示的架构中,磁带中数据发生移动后,两个读/写头所指向的位置会同时改变.例如图 1 中,port0 从域 0 指向域 2,它的移动次数是 2,此时 port1 也从域 4 指向域 6.为了提升系统性能、降低功耗,可以通过调度程序中的加载和存储指令以及决策数据在磁带中的存储位置,即通过优化指令调度和数据放置策略,使得移动次数最小.

本文假设计算操作完成之后,数据会立即被写回到磁畴壁存储器.

2.2 问题定义

针对数据密集型应用程序,本文采用指令访问流图(instruction access flow graph,简称 IAFG)进行建模.首先对程序进行指令提取,然后生成指令访问流图,指令访问流图的定义如定义 1 所示.

定义 1(指令访问流图). $IAFG\ G=\langle V,E,D\rangle$表示一个有向图,其中:$V$ 是访存指令的集合;$E\subseteq V\times V$ 是边的集合,表示访存指令之间的依赖关系;D 是每条访存指令 V 所访问的数据,假设每个数据都为 1bit.在 $IAFG\ G$ 中,$v\in V$ 表示从磁畴壁存储器中加载数据或是往磁畴壁存储器中存储数据的指令.

在此基础上,对本文中需要解决的问题进行问题定义.

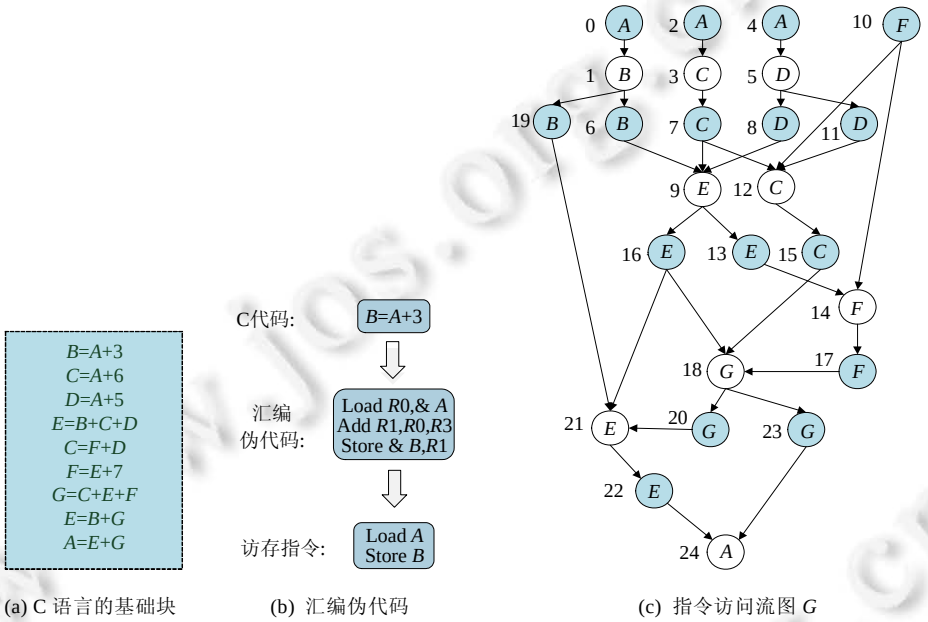
定义 2. **指令调度和数据放置问题(instruction scheduling and data placement problems (ISDPP))**. 给定一个 $IAFG\ G$ 以及目标系统架构(磁畴壁存储器容量、读/写头数量),找到一个指令调度和数据在磁畴壁存储器中的放置策略,使得在数据访问过程中磁畴壁存储器的总移动次数最少.

3 示 例

假设目标系统架构是一个包含 8 个域的基于赛道存储技术的磁畴壁存储器,读/写头数量为 2,分别是 port0

和 port1.域 0~域 3 由读/写头 port0 访问,域 4~域 7 由读/写头 port1 访问.两个读/写头的初始位置分别为域 0 和域 4.

图 2(a)是一个 C 语言代码的基础块,包含 9 个加法运算,该基础块需要访问 7 个数据:*A*,*B*,*C*,*D*,*E*,*F* 和 *G*.将 C 语言代码按照图 2(b)所示的方式,首先转化成汇编伪代码,然后提取访存指令.图 2(c)是根据访存指令之间的依赖关系生成的指令访问流图 *G*,其中,深色的圈表示加载指令(即 load 指令),白色的圈表示存储指令(即 store 指令),圈中的内容表示该访存指令所要访问的数据.简单起见,本文对每个圈进行编号.



(a) C 语言的基础块 (b) 汇编伪代码 (c) 指令访问流图 *G*

Fig.2 A motivation example

图 2 示例

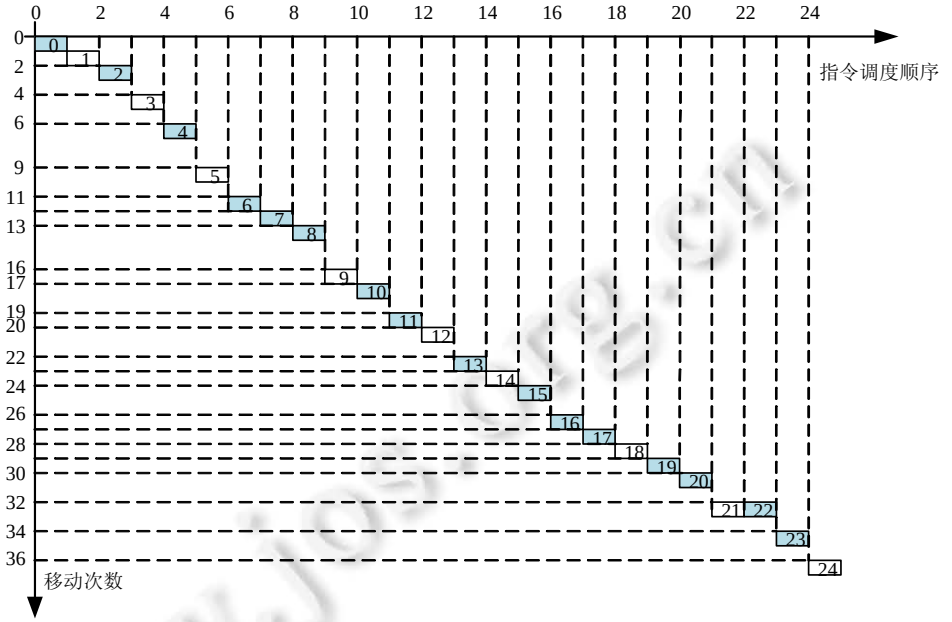
根据图 2(c),使用简单指令调度和数据放置算法可以得到一个简单的指令调度序列以及磁畴壁存储器上简单的数据放置策略,分别如图 3(a)和图 3(b)所示.其中:图 3(a)中,每个填有序号的小格子表示指令,格子中的序号对应于图 2(c)中指令的编号.横坐标表示指令调度顺序,对应指令访问数据的顺序为 *A*,*B*,*A*,*C*,*A*,*D*,*B*,*C*,*D*,*E*,*F*,*D*,*C*,*E*,*F*,*C*,*E*,*F*,*G*,*B*,*G*,*E*,*E*,*G*,*A*.纵坐标表示在移动几次之后读/写头可以与该指令所要访问的数据对齐,其中所需移动次数可以根据图 3(b)的数据放置以及指令调度顺序获得.如指令 2 和指令 3,分别需要访问数据 *A* 和数据 *C*,在访问完数据 *A* 之后要移动 2 次,读/写头才能与指令 3 所要访问的数据 *C* 对齐.而在执行指令 2 之前,已经有 2 次移动操作.所以加上此次的 2 次移动操作,在执行指令 3 之前磁带中的数据共需移动 4 次,对应于纵坐标值为 4 处.在如图 3 所示的顺序调度和数据顺序放置的方案中,一共需要 36 次移动操作,如图 3(b)所示.

Chen 等人[20]所提出的在磁畴壁存储器上进行数据放置的 S-DBC-P 算法是针对已经固定指令调度.首先,用本文所提的算法生成指令调度;然后,用 S-DBC-P 算法进行数据放置,得到的结果如图 4 所示.根据数据放置的结果及指令调度,对应访问数据的顺序为 *A*,*A*,*A*,*B*,*B*,*B*,*C*,*C*,*D*,*D*,*D*,*E*,*E*,*E*,*F*,*F*,*F*,*C*,*C*,*G*,*G*,*G*,*E*,*E*,*A*.可以得到如图 4(a)所示的指令调度顺序-移动次数图,看到采用这种指令调度与数据放置方案一共需要 10 次移动操作.

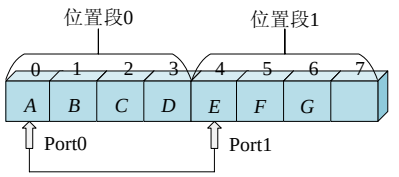
图 5 是通过本文所提出的生成指令调度和数据放置算法(GISDP)得到的结果.图 6(a)是 GISDP 生成的指令调度,对应访问数据的顺序为 *A*,*A*,*A*,*B*,*B*,*B*,*C*,*C*,*D*,*D*,*D*,*E*,*E*,*E*,*F*,*F*,*F*,*C*,*C*,*G*,*G*,*G*,*E*,*E*,*A*.图 6(b)是 GISDP 得到的数据放置.通过该算法,最终得到的总移动次数为 8 次.

通过本文所提的 ILP 模型得到的最优指令调度和数据放置方案如图 6 所示.图 6(a)是通过 ILP 得到的最优指令调度序列,对应访问数据的顺序为 *F*,*A*,*A*,*A*,*D*,*B*,*D*,*B*,*D*,*B*,*C*,*C*,*C*,*C*,*E*,*E*,*E*,*F*,*F*,*G*,*G*,*G*,*E*,*E*,*A*.图 6(b)是通过 ILP

得到的最优数据放置.通过 ILP 得到的总移动次数最少,为 7 次.



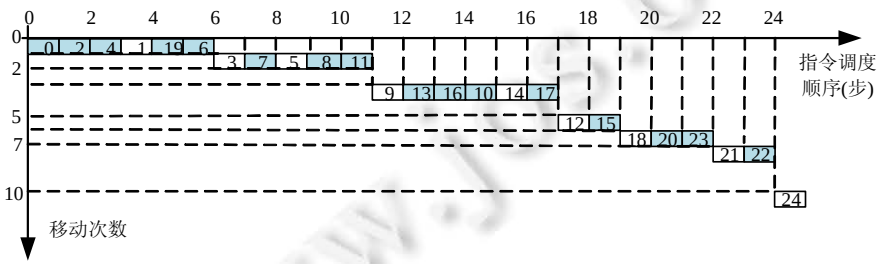
(a) 顺序指令调度



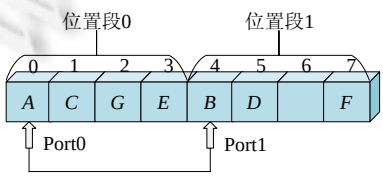
(b) 顺序数据放置

Fig.3 Solution of SSDP

图 3 SSDP 的解决方案



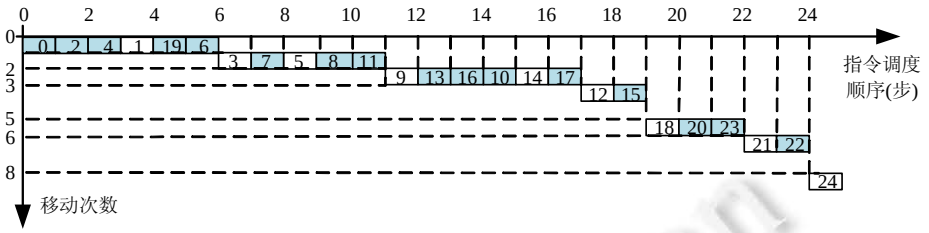
(a) S-DBC-P 得到的调度



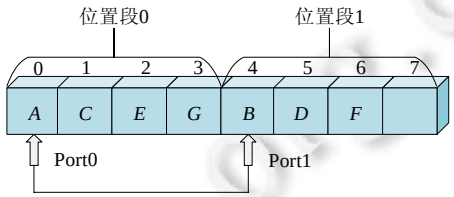
(b) S-DBC-P 的数据放置

Fig.4 Solution of S-DBC-P

图 4 S-DBC-P 的解决方案



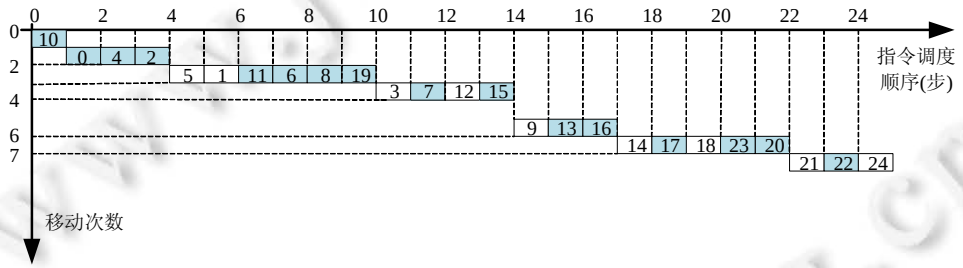
(a) GISDP 得到的调度



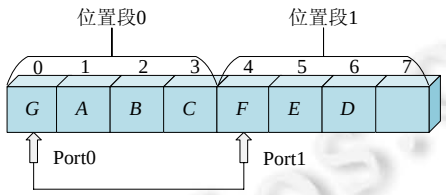
(b) GISDP 得到的数据放置

Fig.5 Solution of GISDP

图 5 GISDP 的解决方案



(a) ILP 得到的调度



(b) ILP 得到的数据放置

Fig.6 Solution of ILP

图 6 ILP 的解决方案

通过比较 4 种指令调度与数据放置方案所需的总移动次数,可以得出,本文所提算法的结果最接近 ILP 得到的最优解.在该示例中:本文所提的方案与顺序调度和数据放置方案相比,总移动次数减少了 80%;与 S-DBC-P 相比,总移动次数减少了 20%.

4 整数线性规划模型与启发式算法

本节首先给出能够得到最优指令调度与数据放置方案的整数线性规划(integer linear programming,简称 ILP)模型.由于 ILP 模型时间复杂度呈指数级,不适合输入较大的情况,因此,本节还提出了启发式算法——生成指令调度和数据放置(GISDP)算法,用以获得近似最优的指令调度与数据放置方案.

4.1 整数线性规划模型

首先给出构建 ILP 模型所需的两个定理以及符号(见表 1).

Table 1 Notations and definitions

表 1 符号及定义

符号	定义
$\|V\|$	图 G 中节点的个数
E	图 G 中的边的集合
$capacity$	磁带上域的个数
D	数据的集合
$\|D\|$	数据的个数
$e(i,j)$	从 i 到 j 的边
$X_{i,l}$	=1,当且仅当指令 i 在第 l 步执行
$Y_{k,p}$	=1,当前仅当数据 k 放置在位置 p 中
SCH_i	指令 i 在第 SCH_i 步调度执行
$access(i)$	被节点 i 访问的数据
pos_{off}	磁带上每个域的偏移地址列表
$offset(i)$	指令 i 访问数据在磁带上的偏移位置
$position(l)$	第 l 步被调度的访存指令所访问的数据的偏移位置
$shift(l)$	第 l 步到第 l+1 步的移动次数

定理 1. 假设 u 和 v 是正整数变量,x 是一个二值变量.描述“$u=v$,iff $x=1$”可以被线性建模为

$$u \geqslant v+(1-x)\times B,$$
$$u \leqslant v+(1-x)\times B,$$
$$u \geqslant 0,$$
$$u \in Z,$$

其中,B 是一个大数.

定理 2. 假设 a,b,c 是正整数变量,则描述“$c \geqslant |a-b|$”可以被线性建模为

$$c \geqslant a-b,$$
$$c \geqslant b-a,$$
$$c \in Z,$$
$$\min(c).$$

根据上述定理,可以通过对指令约束、依赖约束和数据放置约束进行建模,从而构建 ILP 模型.

(1) 指令约束

每一条访存指令在每一步都有执行和不执行两种状态,因此,定义一个二值变量 $X_{i,l}$ 表示访存指令 i 在第 l 步是否执行,其中,i 表示访存指令,l 表示第几步.对于任一条指令 i,在第 l 步执行时,$X_{i,l}$=1;否则,$X_{i,l}$=0.

目标系统架构是单核处理器,因此指令调度步骤的上限为访存指令的总数量.假设共有$|V|$条访存指令,则指令调度上限为$|V|$.

对于指令约束,分为两种.

- 第 1 种是每一步至少有一条访存指令执行,该约束可以线性表示为

$$\sum_{i=1}^{|V|} X_{i,l} \leqslant 1, l \in [1,|V|] \tag{1}$$

- 第 2 种是每条访存指令只能执行一次,将其线性表示为

$$\sum_{l=1}^{|V|} X_{i,l} = 1, i \in [1,|V|] \tag{2}$$

(2) 依赖约束

因为一些访存指令之间存在依赖,如写后读、读后写、写后写等依赖.所以进行访存指令调度时,需要满足所有依赖关系.也就是说,如果访存指令 j 依赖于访存指令 i,即 $i \rightarrow j$,那么访存指令 i 需要在访存指令 j 之前调度执行.定义变量 SCH_i,表示访存指令 i 在第 SCH_i 步调度执行.对于任意访存指令 i,SCH_i 可以被线性表示为

$$SCH_i = \sum_{l=1}^{|V|} X_{i,l} \times l, i \in [1,|V|] \tag{3}$$

如果访存指令 i 与 j 之间存在依赖关系,即在 *IAFG G* 中存在 $e(i,j) \in E$,那么该依赖关系可以表示为

$$SCH_i < SCH_j, e(i,j) \in E \tag{4}$$

表示访存指令 i 要在访存指令 j 之前调度执行,因此可以满足其依赖关系.

(3) 数据放置约束.

定义一个二值变量 $Y_{k,p}$,其中,k 表示数据,p 表示磁畴壁存储器中的域.$Y_{k,p}$ 表示数据 k 放置在域 p 处.当数据 k 放置在域 p 处时,$Y_{k,p}$=1;否则,$Y_{k,p}$=0.假设一条磁带有 *capacity* 个域,那么每个数据会有 *capacity* 种放置的方法.首先,每个数据必须放置在磁带的域中,该约束被线性表示为

$$\sum_{p=0}^{capacity-1} Y_{k,p} = 1, k \in D \tag{5}$$

不失一般性,每个域中最多只能放一个数据.当数据量小于磁带容量 *capacity* 时,有些域中不放置数据.假设有|*D*|个数据需要被访问,那么将有|*D*|个数据需要被放在磁带的域中,该约束被线性表示为

$$\sum_{k \in D} Y_{k,p} \leqslant 1, \forall p \in [0, capacity) \tag{6}$$

假设数据量不会超过磁带容量 *capacity*,该约束线性表示为

$$\sum_{k \in D} \sum_{p=0}^{capacity-1} Y_{k,p} \leqslant capacity \tag{7}$$

为了建模移动次数,需要知道指令调度、访存指令访问的数据以及数据放置位置.也就是说,需要知道每一步所调度的访存指令访问的数据放置在磁畴壁存储器的磁带上哪个域中.因为在磁畴壁存储器中有两个读/写头进行数据的读/写,分别为 port0 和 port1.并且磁带的前半部分域由 port0 进行访问,磁带的后半部分域由 port1 进行访问,因此需要建模每个位置与其对应读/写头的偏移位置.例如,磁带上有 8 个域,分别编号为域 0~域 7.port0 访问域 0~域 3,port1 访问域 4~域 7.初始状态下,port0 与域 1 对齐,port1 与域 5 对齐.对于 port0 来说,域 1 是第 1 个位置;对于 port1 来说,域 5 是第 1 个位置.如果域 1 中的数据被访问之后,立即访问域 5 中的数据,那么就不需要移动操作,此时移动次数为 0.如果使用的不是偏移位置进行计算,则是 5−1=4,会出现计算错误.因此,建模移动次数需要获得偏移位置.

定义一个列表 pos_{off} 记录每个域的偏移位置,即 $pos_{off} = \left[0,1,2,...,\frac{capacity}{2}-1,0,1,2,...,\frac{capacity}{2}-1\right]$.例如:如果磁带容量为 8,则 pos_{off}=[0,1,2,3,0,1,2,3].

对于每个数据 k 和访存指令 i,如果访存指令 i 访问数据 k,那么数据 k 和访存指令 i 直接的映射关系为

$$k = access(i) \tag{8}$$

定义 *offset*(i)表示访存指令 i 所访问数据的偏移位置,该约束可以表示为

$$offset(i) = \sum_{i=0}^{|V|} \sum_{p=0}^{capacity-1} Y_{access(i),p} \times pos_{off}[p] \tag{9}$$

如果变量 $X_{i,l}$=1 表示访存指令 i 在第 l 步调度执行,定义 *position*(l)来表示第 l 步调度执行的访存指令 i 所访问数据的偏移位置,它可以表示为

$$position(l) = offset(i), \text{iff } X_{i,l} = 1 \tag{10}$$

也就是说,当访存指令 i 在第 l 步调度执行时,就将访存指令 i 所访问数据的偏移位置赋值给 *position*(l).不幸的是,该表达式是非线性的.根据定理 1,它可以线性表示为

$$position(l) \geqslant offset(i) + (1 - X_{i,l}) \times B \tag{11}$$

$$position(l) \leqslant offset(i) + (1 - X_{i,l}) \times B \tag{12}$$

$$position(l) \geqslant 0 \tag{13}$$

$$position(l) \in Z \tag{14}$$

其中,B 是一个大数.

假设 *shift*(l)记录的是从当前步 l 到下一步 l+1 所需要移动的次数,那么从第 l 步到第 l+1 步的移动次数可以表示为

$$shift(l)=|position(l+1)-position(l)| \tag{15}$$

该公式可以根据定理 2 重写为线性公式,如下:

$$shift(l)\geqslant position(l+1)-position(l) \tag{16}$$

$$shift(l)\geqslant position(l)-position(l+1) \tag{17}$$

$$shift(l)\in Z \tag{18}$$

(4) 目标函数.

整数线性规划的目标函数是总移动次数最小,线性公式表示为

$$\min\sum_{l=0}^{|V|-1} shift(l) \tag{19}$$

4.2 生成指令调度和数据放置(GISDP)算法

本节提出了一个启发式算法——生成指令调度和数据放置(GISDP)算法,该算法可以生成一个指令调度和数据放置方案,使得总移动次数达到近似最优.由于提出的 GISDP 算法基于赛道存储技术的存储器,所以适用于本文所提到的磁畴壁存储介质的存储器,也同样适用于斯格明子介质的存储器.

GISDP 算法是一个三段式算法,其主要思想是:首先由算法 1 根据 *IAFG G* 同时生成指令调度序列和数据放置策略;然后,算法 2 在算法 1 结果的基础上对数据放置进行改进;最后,算法 3 根据算法 2 改进的结果对指令调度进行改进.

在介绍 GISDP 算法细节之前,先介绍等价类的概念.在有多个读/写头的磁畴壁存储器上,磁带上的域被 *m* 个读/写头平均分成 *m* 段,每段都有相同的偏移地址.因此在任何时刻,*m* 个读/写头所处域的偏移地址相同.例如:假设 *P* 和 *Q* 分别是读/写头 port1 和 port3 所管辖段中域的集合,那么如果 *P* 中的域 *p* 和 *Q* 中的域 *q* 有相同的偏移位置,则域 *p* 和域 *q* 是等价的,即,数据放在域 *p* 和放在域 *q* 中的效果是一样的.由于读/写头的数量为 *m*,因此等价类的大小是 *m*.

GISDP 算法第 1 部分(算法 1)的主要思想是:将需要访问但还未被放置的数据放置在读/写头当前所指向的域中,如果当前所有读/写头指向的域均已放置数据,那么进行移动操作,使得读/写头和相邻的下一个域对齐,将数据放置到距离读/写头最近的空闲域中.然后再将访问该数据且不依赖于其它未执行指令的所有指令进行调度.如算法 1 所述,输入为 *IAFG G* 和目标系统架构;输出为一个指令调度 *sch*、一个数据放置 *place*、移动次数 *shift*、一个记录产生移动操作的连续访问的数据对列表 *shift_data*、一个记录产生移动操作的移动次数的列表 *shift_count*.

算法第 1 行,统计 *G* 中入度为 0 的指令和不为 0 指令及其入度,分别存入列表 *list*0 和 *list*1 中;在算法初始阶段,指令调度 *sch* 和数据放置 *place* 为空,并且所有 port 均指向偏移位置为 0 的域处,此时的状态需要执行第 6 行~第 9 行,随机选取满足条件的指令调度,并把指令所访问数据放置在此时 port 所指向域中的任一空闲域,同时更新列表 *list*0,*list*1 以及数据 *D* 集合.另一种状态,在 port 所指向域中有数据且 *list*0 中有访问这些数据的指令时,则执行第 5 行.当以上两种状态下均未在 *list*0 中找到可调度指令,则要执行第 11 行,进行移动操作之后的状态将会是前两种状态中的一种,则重复执行第 4 行~第 9 行.在数据 *D* 集合为空的情况下,说明数据均已被放置.接下来只需要将剩余指令调度完成.调度过程中,先调度访问当前 port 所指向域中数据的指令,即第 15 行;没有满足条件的指令,则需要执行第 17 行和第 18 行,调度 *list*0 中访问距离当前域最近的域中数据的指令.直至所有指令调度完成.

生成了调度 *sch* 和数据放置 *place* 后,执行第 21 行,将统计得到总移动次数 *shift*,产生移动操作的连续访问的数据对,存入列表 *shift_data*,及数据对所对应移动次数,存入列表 *shift_count* 进行输出.

算法 1. GISDP 算法第 1 部分——由 IAFG 同时生成调度和数据放置.

输入:一个 *IAFG G*=⟨*V*,*E*,*D*⟩;

磁带的容量 *N*,*M* 个读/写头;

输出:一个指令调度 *sch*;

一个数据放置 *place*;

移动次数 *shift*;

一个记录产生移动操作的连续访问的数据对列表 *shift_data*;

一个记录产生移动操作的移动次数的列表 *shift_count*.

1. *list*0←统计入度为 0 的节点;*list*1←统计入度不为 0 节点的入度;
2. **while** *len*(*sch*)<*len*(*V*) **do**
3. **while** *D* 不为空 **do**
4. **if** *list*0 中有访问此时 port 所指向域中数据的指令 **then**
5. *sch*←满足条件的指令;更新 *list*0,*list*1;
6. **elif** *list*0 中无访问此时 port 所指向域中数据的指令且此时 port 所指向域有空闲域 **then**
7. *sch*←随机取一条满足条件的指令;更新 *list*0,*list*1;
8. 被调度指令所访问的数据 *d* 放置在此时 port 所指向的任一空闲域;
9. *D*=*D*−{*d*};
10. **else then**
11. 磁带上的数据整体左移一次,重复第 4 行~第 9 行;
12. **endif**
13. **endwhile**
14. **if** *list*0 中有访问此时 port 所指向域中数据的指令 **then**
15. *sch*←满足条件的指令;更新 *list*0,*list*1;
16. **else then**
17. *sch*←选取 *list*0 中访问数据所在域距离此时 port 所指向域最近的域中数据的指令;
18. 更新 port 的状态;重复第 14 行~第 18 行;
19. **endif**
20. **endwhile**
21. 根据 *sch* 和 *place* 统计总移动次数 *shift*,产生移动操作的连续访问的数据对 *shift_data* 及其移动次数 *shift_count*;
22. **return** *sch*,*place*,*shift*,*shift_count*,*shift_data*.

算法 2 的主要思想是:根据算法 1 生成的结果计算连续访问数据对的相关度,依据将相关度大的数据放在等价位置的原则对数据放置进行改进.如算法 2 所描述,输入为 *IAFG G*、目标系统架构以及算法 1 的输出;输出为比算法 1 总移动次数少的数据放置顺序 *place*、总的移动次数 *shift*、一个记录移动次数的列表 *shift_count*.

算法第 1 行,根据算法 1 生成的产生移动操作时连续访问的数据对的列表 *shift_data* 计算数据对连续访问的次数,并存入列表 *count* 中.第 2 行,根据列表 *count*,*shift_count* 和 *shift_data*,将连续次数 *count* 与对应数据对的总移动距离相乘得到数据对的相关度,并根据数据对的相关度的降序排列将相关度和数据对存入列表 *relate* 中.从最大相关度的数据对入手,如果数据对(*m*,*n*)中的数据 *m* 和 *n* 放置在同一个位置段,则执行第 8 行和第 9 行;否则,执行第 5 行和第 6 行.在将数据 *m* 放在等价位置处后,则原位置处的数据 *r* 被交换到数据 *m* 的原位置处.进行数据交换之后,就会导致 *r* 与原来相邻位置及等价位置处的数据的距离增加,所以需要将 *r* 当前所在域到 *m* 当前所在域之间的数据(包括 *r*,不包括 *m*)向 *r* 所在域的方向循环移一个位置.为了减少 *r* 与之前相邻域及等价域中数据的相对距离,需要对所有段中该区间的域中的数据循环移动一个位置.在进行交换位置及移动操作之后,会重新计算总的移动次数:如果移动次数减少,则会保留此次交换,更新数据放置 *place*,并重新计算相关度;如果移动次数不变或者增加,则本次交换取消,进行下一个相关度数据对的位置重置.

在交换到数据对的相关度为 1 的时候,则停止位置重置,此时的数据放置作为新的放置策略进行输出.

算法 2. GISDP 算法第 2 部分——修改数据放置.

输入:一个 *IAFG G*=⟨*V*,*E*,*D*⟩;

磁带的容量 *N*,*M* 个读/写头;

算法 1 的输出的 *sch*,*place*,*shift*,*shift_count*,*shift_data*;

输出:比算法 1 总移动次数少的数据放置顺序 *place*;

总的移动次数 *shift*;

一个记录移动次数的列表 *shift_count*.

1. *shift*←统计 *shift_data* 中每对连续访问数据的连续次数;
2. *relate*←计算 *shift_data* 中的每对连续访问数据的相关度,并根据相关度降序排序数据对;
3. **while** *relate*[*i*]>1 **do**
4. **if** 数据对(*m*,*n*)不在同一个位置段 **then**
5. 将数据 *m* 与在本位置段中和数据 *n* 偏移位置相同的数据 *r* 交换位置;
6. 对所有段中,将 *r* 所在域(及其等价域)到 *m* 所在域(及其等价域)之间的数据(包括 *r*,不包括 *m*),向 *r* 所在域的方向循环移动一个位置;
7. **else then**
8. 将其他任意一个位置段中与 *n* 偏移位置相同的数据 *r* 与 *m* 交换位置;
9. 对所有段中,将 *m* 原来放置域(及其等价域)到 *m* 当前放置域(及其等价域)之间的数据,向 *m* 原来放置域的方向循环移动一个位置;
10. 统计交换位置后的移动次数;
11. **if** 本次统计移动次数<*shift* **then**
12. *shift*=本次统计移动次数;*place*=本次放置结果;
13. 重新统计相关度,更新 *relate*;*i*=0;
14. **else then**
15. *shift* 不变;*place*=之前放置结果;*i*=*i*+1;
16. **end if**
17. **end while**
18. **return** *place*,*shift*.

在算法 2 对数据的放置进行改进之后,算法 3 根据算法 2 的数据放置和 *IAFG G* 对指令进行了重调度,以获得与新数据放置策略相适宜的指令调度.算法 3 的主要思想是:尽量将访问当前 port 所在域中数据的指令进行调度,没有可调度的指令则移动一个位置.当 port 移动到有指令可调度的域的时候,将会更新 port 的状态.最后统计该调度下的移动次数,与重调度之前的移动次数进行比较,如果移动次数减少,则保留此次的调度;否则不更新调度.

算法 3. GISDP 算法第 3 部分——改进指令调度.

输入:一个 *IAFG G*=⟨*V*,*E*,*D*⟩;

磁带的容量 *N*,*M* 个读/写头;

算法 1 的输出 *sch*,算法 2 的输出 *place*,*shift*;

输出:一个指令调度列表 *sch*;

总的移动次数 *shift*.

1. *list*0←统计入度为 0 的节点;*list*1←统计入度不为 0 的节点;
2. *i*=1;
3. **while** *len*(*sch*)<*len*(*V*) **do**
4. **if** port 此时状态下有可调度的指令 **then**

5. 调度 *list*0 中访问当前 *m* 个域中数据的指令;更新 *list*0,*list*1;
6. **else then**
7. **if** port 左移 *i* 位不会移到其他位置段 **then**
8. **if** port 此时状态下没有可调度的指令 **then**
9. port 保持原状态;
10. **if** port 右移 *i* 位不会移到其他位置段 **then**
11. **if** port 此时状态下没有可调度的指令 **then**
12. port 保持原状态;*i*=*i*+1;
13. **else then**
14. 更新 port 状态,*i*=1;调度 *list*0 中访问当前 *m* 个域中数据的指令;更新 *list*0,*list*1;
15. **end if**
16. **end if**
17. **else then**
18. 更新 port 状态 *i*=1;调度 *list*0 中访问当前 *m* 个域中数据的指令;更新 *list*0,*list*1;
19. **end if**
20. **end if**
21. **end if**
22. **end while**
23. **if** 本次统计移动次数<*shift* **then**
24. *shift*=本次统计移动次数;*sch*=本次调度结果;
25. **else then**
26. *shift* 不变;*sch*=之前调度结果;
27. **end if**
28. **return** *shift*,*sch*.

算法 1 的时间复杂度为 $O(|V|\times|D|)$,在一个磁带中,$|D|$最大为磁带容量,即数据量,$|V|$为应用中访存指令数量;算法 2 的时间复杂度为 $O\left(\frac{|V|}{2}\times(|V|+1)\right)$,其中,$\frac{|V|}{2}\times(|V|+1)$ 表示应用中连续访问数据的相关度大于 1 的数据对可能的最大个数;算法 3 的时间复杂度为 $O(|V|)$.

5 实 验

本节首先介绍实验设置,然后通过实验对比展示本文所提出的 ILP 和 GISDP 算法的效率,并对实验结果进行分析.

5.1 实验设置

本文从 MediaBench[26]基准套件中选择 8 个应用程序进行实验,这 8 个应用程序分别为 epic,ghostscript,jpeg,mesa,mpeg,pegwit,pgp 和 rasta.将本文所提出的 ILP 和 GISDP 算法与不同的算法进行比较:(1) SSDP 算法,将指令进行顺序调度并且按照数据的访问顺序进行顺序放置的一种算法; (2) S-DBC-P 算法,Chen 等人[20]提出的在固定调度的情况下,对数据进行分组放置的一个算法;(3) GISDP 算法,本文所提出的启发式算法;4) ILP 方法,本文所提出的可以得到最优解的模型.将本文提出的算法与其他算法比较之后,本文继续对启发式算法自身的三段算法进行性能提升的比较与讨论.

本文使用 SimpleScalar 模拟器[27]对基准程序进行指令提取并生产指令访问流图,同时,本文设计了磁畴壁存储器访问行为模拟器.将指令访问流图以及每条指令所访问数据输入到模拟器中,使用不同的算法可以在模拟器中生成指令调度与数据放置方案,并获得总的移动次数.在模拟器中,假设数据能够全部存储在磁畴壁存储

器中.本文分别对读/写头数量为 2,4,8 时的磁畴壁存储器进行了设计空间的探索实验.

对于 ILP 模型,使用 python3 调用 Gurobi 中的接口[28]对 ILP 模型进行编写,并运行 ILP 程序.对于某些测试程序,ILP 不能在可接受的时间范围内找到最优解,所以本文设定在 ILP 模型被执行 12 小时(43 200 秒)之后便停止执行,并将当前的结果输出.其中,最优解使用“*”标记.其他的算法均是可以在几百毫秒时间内完成的.

5.2 实验结果与分析

本节对我们的实验结果进行展示与分析.

表 2 是在配备 2 个读/写头的磁畴壁存储器上不同算法产生的实验结果.其中:“SSDP”列表示 SSDP 算法得到的移动次数;“S-DBC-P”列是 Chen 等人[20]提出的算法得到的移动次数,以及相较于 SSDP 算法移动次数减少的百分比;“GISDP”列是本文所提出的启发式算法求得的移动次数,以及分别相较于 SSDP 算法和 S-DBC-P 算法移动次数减少的百分比;“ILP”列是整数线性规划在 12 小时内求得的移动次数,以及分别相较于 SSDP 算法和 S-DBC-P 算法移动次数减少的百分比.从表 2 可以看出:本文所提的 GISDP 算法相较于 SSDP 算法,移动次数平均减少了 86.7%,相较于 S-DBC-P 算法,移动次数平均减少了 79.6%;本文所提出的 ILP 模型虽然没有在 12 小时内求出最优解,但是相较于 SSDP 算法,移动次数平均减少了 75.0%,相较于 S-DBC-P 算法,移动次数平均减少了 61.8%.

Table 2 Performance comparison for 2-port

表 2 2 个 port 的性能比较

	SSDP	S-DBC-P		GISDP			ILP		
基准测试程序	移动操作次数	移动操作次数	相对SSDP减少(%)	移动操作次数	相对SSDP减少(%)	相对S-DBC-P减少(%)	移动操作次数	相对SSDP减少(%)	相对S-DBC-P减少(%)
epic	8 394	3 492	58.4	630	92.5	81.9	1 648	80.3	52.8
ghostscript	3 308	2 116	36.0	208	93.7	90.2	642	80.6	69.6
jpeg	4 152	2 459	40.7	525	87.4	78.6	1 288	68.9	47.6
mesa	5 248	2 601	50.4	883	83.2	66.1	1 121	78.6	56.9
mpeg	8 944	4 726	47.1	1 630	81.8	65.5	2 364	73.6	49.9
pegwit	1 721	1 487	13.6	217	87.4	85.4	390	77.3	73.8
pgp	3 043	3 503	−15.1	495	83.7	85.9	973	68.0	72.2
rasta	3 253	2 117	4.1	519	84.0	83.3	887	72.7	71.5
平均减少(%)	–	–	29.4	–	86.7	79.6	–	75.0	61.8

表 3 是在配备 4 个读/写头的磁畴壁存储器上不同算法产生的实验结果.从表 3 可以看出:本文所提的 S-DBC-P 算法相较于 SSDP 算法移动次数平均减少了 93.7%,相较于 S-DBC-P 算法移动次数平均减少了 80.7%;本文所提出的 ILP 模型在 12 小时内求出的局部最优解,相较于 SSDP 算法,移动次数平均减少了 82.6%,相较于 S-DBC-P 算法,移动次数平均减少了 46.3%.

Table 3 Performance comparison for 4-port

表 3 4 个 port 的性能比较

	SSDP	S-DBC-P		GISDP			ILP		
基准测试程序	移动操作次数	移动操作次数	相对SSDP减少(%)	移动操作次数	相对SSDP减少(%)	相对S-DBC-P减少(%)	移动操作次数	相对SSDP减少(%)	相对S-DBC-P减少(%)
epic	8 359	1 912	77.1	404	95.2	78.9	1 482	82.3	22.5
ghostscript	3 286	1 124	65.8	124	96.2	88.9	466	85.8	58.5
jpeg	4 125	1 188	71.2	247	94.0	79.2	865	79.0	27.2
mesa	5 221	1 424	72.7	367	92.9	74.2	851	83.7	40.2
mpeg	8 907	2 515	71.7	679	92.4	73.0	974	89.0	61.3
pegwit	1 699	778	54.2	136	91.9	82.5	274	83.9	64.8
pgp	3 008	1 087	63.9	192	93.6	82.3	742	75.3	31.7
rasta	3 223	1 634	49.3	225	93.0	86.2	583	81.9	64.3
平均减少(%)	–	–	65.7	–	93.7	80.7	–	82.6	46.3

表 4 是在配备 8 个读/写头的磁畴壁存储器上不同算法产生的实验结果,从表 4 可以看出:本文的 S-DBC-P

算法相较于 SSDP 算法移动次数平均减少了 97.3%,相较于 S-DBC-P 算法移动次数平均减少了 85.6%;本文提出的 ILP 模型相较于 SSDP 算法移动次数平均减少了 94.8%,相较于 S-DBC-P 算法移动次数平均减少了 75.6%.

Table 4 Performance comparison for 8-port

表 4 8 个 port 的性能比较

	SSDP	S-DBC-P		GISDP			ILP		
基准测试程序	移动操作次数	移动操作次数	相对SSDP减少(%)	移动操作次数	相对SSDP减少(%)	相对S-DBC-P减少(%)	移动操作次数	相对SSDP减少(%)	相对S-DBC-P减少(%)
epic	8 342	1 019	87.8	165	98.0	83.8	434	94.8	57.4
ghostscript	3 275	618	81.1	64	98.0	89.6	204	93.8	67.0
jpeg	4 111	630	84.6	116	97.2	81.6	84*	98.0	86.7
mesa	5 207	719	86.2	136	97.4	81.1	98*	98.1	86.4
mpeg	8 888	1 383	84.4	250	97.2	81.9	156*	98.2	86.4
pegwit	1 688	455	73.0	65	96.1	85.7	50*	97.0	88.7
pgp	2 991	1 024	65.8	89	97.0	91.3	383	87.2	62.6
rasta	3 208	829	74.2	82	97.4	90.1	273	91.5	67.1
平均减少(%)	–	–	79.6	–	97.3	85.6	–	94.8	75.6

从 3 个表中可以看出:ILP 只有在配备 8 个读/写头的磁畴壁存储器上的实验结果有求出最优结果的情况,且本文的 GISDP 算法所求得结果与 ILP 最优结果相近.如表 4 中基准测试程序 pegwit 的结果,GISDP 算法的移动次数分别相较于 SSDP 算法和 S-DBC-P 算法减少了 96.1%和 85.7%,ILP 模型的移动次数分别相较于 SSDP 算法和 S-DBC-P 算法减少了 97.0%和 88.7%,所以本文的 GISDP 算法可以在多项式时间内求得近似最优解.在配备 2 个读/写头和 4 个读/写头的磁畴壁存储器上的实验结果均没有在 12 小时内找到最优解,所以这两个表中的 ILP 结果均是 12 小时得到的局部最优解.

在本文提出的 ILP 模型中,ILP 的约束条件的数量是固定的,影响 ILP 求解效率的主要因素是输入的 IAFG G 中$|V|$,$|E|$和$|D|$的大小以及读/写头的数量.当读/写头的数量为 2 时,在满足依赖关系的前提下,对指令进行调度和数据进行放置,相当于是进行全排列,并选择最优的情况.当$|E|$很小,即依赖关系很少,满足依赖的排列数接近全排列.当$|V|$和$|D|$很大时,全排列的复杂度使得 ILP 无法在多项式时间内求得最优解决策略.不过,当读/写头的数量为 8 时,一部分基准测试程序的 ILP 收敛速度有所提高(如表 4),但是所需时间仍不能与启发式算法所需的时间相比.可以看出:ILP 模型虽然可求最优解,但是不能在多项式时间内完成,不适合用在数据密集型应用的系统中.

表 5 是在配备 2 个读/写头的磁畴壁存储器上 GISDP 算法中三段算法分别得到的结果,其中,算法 2 相对于算法 1 移动次数平均减少了 9.81%,算法 3 相对于算法 2 移动次数平均减少了 9.43%.从表 5 中可以看出:部分基准测试程序在算法 2 及算法 3 中,移动次数减少了 0%,是因为不同的测试程序具有不同的数据访问特性;并且算法 2 和算法 3 均是在自身所求得的结果与前一段算法所求得的结果中选取更优的结果.因此对于部分应用,会在算法 1 或算法 2 中就求得近优解.

Table 5 Performance comparison for 3-segment of GISDP (2-port)

表 5 GISDP 的三段算法性能比较(2 个 port)

	算法 1	算法 2		算法 3	
基准测试程序	移动次数	移动次数	相对算法 1 减少(%)	移动次数	相对算法 2 减少(%)
epic	630	630	0	630	0
ghostscript	356	254	28.7	208	18.1
jpeg	911	784	13.9	525	33.0
mesa	943	883	6.4	883	0
mpeg	1 860	1 860	0	1 630	12.4
pegwit	277	243	12.3	217	10.7
pgp	522	501	4.0	495	1.2
rasta	598	519	13.2	519	0
平均减少(%)	–	–	9.81	–	9.43

实验结果显示:本文所提的算法可以找到较优的指令调度和数据放置策略,并且有效地减少了总的移动次

数.因为考虑了连续访问数据对的相关度对数据放置进行了改进,同时根据放置进行了指令重调度,所以本文所提的算法可以到达较好的性能.

6 结 论

本文针对数据密集型应用,面向配备多个读/写头的磁畴壁存储器的单核处理器系统研究最优指令调度与数据放置方案来获得最少的移动操作次数,以提升磁畴壁存储器的存储访问性能,进而提升系统性能.本文提出了可以在配备多个读/写头的磁畴壁存储器上生成最优的指令调度和数据放置方案的 ILP 模型,可以求得最小的移动次数.因为 ILP 模型的时间复杂度呈指数级,本文提出了可以在配备多个读/写头的磁畴壁存储器上,在多项式时间内生成近似最优的指令调度和数据放置方案(GISDP)的启发式算法,以此来减小移动次数.对配备不同数量读/写头的磁畴壁存储器的存储访问性能进行设计探索与实验,表明本文所提出的 ILP 模型和 GISDP 算法能够生成最优和近似最优的指令调度与数据放置方案.

由于研究问题的复杂性,本文主要考虑的是配备多个读/写头的磁畴壁存储器的单核处理器系统,但在实际用中,多核处理器系统使用更加广泛.在未来的工作中,将会研究配备多个读/写头的磁畴壁存储器的多核处理器系统中指令调度与数据放置策略,减少移动次数并提高磁畴壁存储器的存储访问性能,进而提高整个体统的性能.

References:

[1] Yang D, Ren H. The research on the technology of Internet of things and embedded system. In: Proc. of the IEEE Int'l Conf. on Software Engineering and Service Science (ICSESS). Piscataway: IEEE, 2017. 395−398. [doi: 10.1109/ICSESS.2017.8342940]

[2] Gong XQ, Jin CQ, Wang XL, Zhang R, Zhou AY. Data-Intensive science and engineering: requirements and challenges. Chinese Journal of Computers, 2012,35(8):1563−1578 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2012.01563]

[3] Dias WP, Colonese E. Performance analysis of cache and scratchpad memory in an embedded high performance processor. In: Proc. of the Int'l Conf. on Information Technology: New Generations (ITNG 2008). Piscataway: IEEE, 2008. 657−661. [doi: 10.1109/ITNG.2008.226]

[4] Chang DW, Lin IC, Chien YS, Lin CL, Su AWY, Young CP. CASA: Contention-aware scratchpad memory allocation for online hybrid on-chip memory management. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2014,33(12): 1806−1817. [doi: 10.1109/TCAD.2014.2363385]

[5] Gu SZ, Sha EHM, Zhuge QF, Chen YR, Hu JT. A time, energy, and area efficient domain wall memory based SPM for embedded systems. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2016,35(12):2008−2017. [doi: 10.1109/TCAD.2016.2547903]

[6] Gu SZ. Task scheduling and data allocation for systems with non-volatile memory [Ph.D.Thesis]. Chongqing: Chongqing University, 2016 (in Chinese with English abstract).

[7] Wang Z, Gu ZH, Yao M, Shao ZL. Endurance-Aware allocation of data variables on NVM-based scratchpad memory in real-time embedded systems. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2015,34(10):1600−1612. [doi: 10.1109/TCAD.2015.2422846]

[8] Mao HY, Zhang C, Sun GY, Shu JW. Exploring data placement in racetrack memory based scratchpad memory. In: Proc. of the IEEE Non-Volatile Memory System and Applications Symp. (NVMSA). Piscataway: IEEE, 2015. 1–5. [doi: 10.1109/NVMSA.2015.7304358]

[9] Chen XZ, Sha EHM, Jiang WW, Zhuge QF, Chen JX, Qin JJ, Zeng YS. The design of an efficient swap mechanism for hybrid DRAM-NVM systems. In: Proc. of the Int'l Conf. on Embedded Software (EMSOFT). New York: ACM, 2016. [doi: 10.1145/2968478.2968497]

[10] Chen XZ. File systems and swap mechanism for non-volatile memory systems [Ph.D.Thesis]. Chongqing: Chongqing University, 2017 (in Chinese with English abstract).

[11] Chen XZ, Sha EHM, Zhuge QF, Dai PL, Jiang WW. Optimizing data placement for reducing shift operations on domain wall memories. In: Proc. of the 2015 52nd ACM/EDAC/IEEE Design Automation Conf. (DAC). Piscataway: IEEE, 2015. 1−6. [doi: 10.1145/2744769.2744883]

[12] Parkin SSP, Hayashi M, Thomas L. Magnetic domain-wall racetrack memory. Science, 2008,320(5873):190−194. [doi: 10.1126/science.1145799]

[13] Wang YH, Yu H. An ultralow-power memory-based big-data computing platform by nonvolatile domain-wall nanowire devices. In: Proc. of the 2013 Int'l Symp. on Low Power Electronics and Design (ISLPED 2013). New York: ACM, 2013. 329−334.

[14] Venkatesan R, Kozhikkottu V, Augustine C, Raychowdhury A, Roy K, Raghunathan A. TapeCache: A high density, energy efficient cache based on domain wall memory. In: Proc. of the ACM/IEEE Int'l Symp. on Low Power Electronics & Design. New York: ACM, 2012. 185−190. [doi: 10.1145/2333660.2333707]

[15] Venkatesan R, Kozhikkottu VJ, Sharad M, Augustine C, Raychowdhury A, Roy K, Raghunatha A. Cache design with domain wall memories. IEEE Trans. on Computers, 2016,65(4):1010−1024. [doi: 10.1109/TC.2015.2506581]

[16] Mao MJ, Wen WJ, Zhang YJ, Chen YR, Li H. Exploration of GPGPU register file architecture using domain-wall-shift-write based racetrack memory. In: Proc. of the 2014 51st ACM/EDAC/IEEE Design Automation Conf. (DAC). Piscataway: IEEE, 2014. 1−6.

[17] Venkatesan R, Ramasubramanian S, Venkataramani S, Kaushik R, Raghunathan A. STAG: Spintronic-tape architecture for GPGPU cache hierarchies. In: Proc. of the 41st Annual Int'l Symp. on Computer Architecture (ISCA 2014). New York: ACM, 2014. 253−264. [doi: 10.1145/2678373.2665710]

[18] Wang YH, Yu H, Sylvester D, Kong PF. Energy efficient in-memory AES encryption based on nonvolatile domain-wall nanowire. In: Proc. of the Conf. on Design, Automation & Test in Europe (DATE 2014). New York: ACM, 2014.

[19] Zhang C, Sun GY, Zhang XY, Zhao WS. Thermal modeling and management for shift operations of racetrack memory. Journal of Computer Research and Development, 2017,54(1):154−162 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2017.20150903]

[20] Chen XZ, Sha EHM, Zhuge QF, Xue C, Jiang WW, Wang YG. Efficient data placement for improving data access performance on domain-wall memory. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2016,24(10):3094−9104. [doi: 10.1109/TVLSI.2016.2537400]

[21] Motaman S, Iyengar A, Ghosh S. Synergistic circuit and system design for energy-efficient and robust domain wall caches. In: Proc. of the 2014 Int'l Symp. on Low Power Electronics and Design (ISLPED 2014). New York: ACM, 2014. 195−200. [doi: 10.1145/2627369.2627643]

[22] Liu BC, Gu HF, Chen MS, Gu SZ, Chen WJ. An efficient processing in memory framework based on skyrmion material. Journal of Computer Research and Development, 2019,56(4):798−809 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2019.20180157]

[23] Liu Y, Liu X, Zhu JG. Tailoring the current-driven domain wall motion by varying the relative thickness of two heavy metal underlayers. IEEE Trans. on Magnetics, 2018,54(11):1−5. [doi: 10.1109/TMAG.2018.2827046]

[24] Hu JT, Zhuge QF, Xue C, Tseng WC, Sha EHM. Management and optimization for nonvolatile memory-based hybrid scratchpad memory on multicore embedded processors. ACM Trans. on Embedded Computing Systems, 2014,13(4):1−25. [doi: 10.1145/2560019]

[25] Gu SZ, Sha EHM, Zhuge QF, Chen YR, Hu JT. Area and performance co-optimization for domain wall memory in application-specific embedded systems. In: Proc. of the 52nd Annual Design Automation Conf. (DAC 2015). New York: ACM, 2015. 1−6. [doi: 10.1145/2744769.2744800]

[26] Lee C, Potkonjak M, Mangione-Smith WH. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In: Proc. of the 30th Annual ACM/IEEE Int'l Symp. on Microarchitecture. Piscataway: IEEE, 1997. 330−335. [doi: 10.1109/MICRO.1997.645830]

[27] Austin T, Larson E, Ernst D. SimpleScalar: An infrastructure for computer system modeling. Computer, 2002,35(2):59−67. [doi: 10.1109/2.982917]

[28] Optimization G. Gurobi optimizer reference manual. 2015. http://www.gurobi.com

附中文参考文献:

[2] 宫学庆,金澈清,王晓玲,张蓉,周傲英.数据密集型科学与工程:需求和挑战.计算机学报,2012,35(8):1563−1578. [doi: 10.3724/SP.J.1016.2012.01563]

[6] 谷守珍.面向非易失性存储器系统的任务调度与数据分配研究[博士学位论文].重庆:重庆大学,2016.

[10] 陈咸彰.面向非易失性内存的文件系统与页面交换机制研究[博士学位论文].重庆:重庆大学,2017.

[19] 张超,孙广宇,张学莹,赵巍胜.赛道存储器移动操作的温度模型及控制策略.计算机研究与发展,2017,54(1):154−162. [doi: 10.7544/issn1000-1239.2017.20150903]

[22] 刘必成,顾海峰,陈铭松,谷守珍,陈闻杰.一种基于斯格明子介质的高效存内计算框架.计算机研究与发展,2019,56(4):798−809. [doi: 10.7544/issn1000-1239.2019.20180157]

许瑞(1996－),女,学士,CCF 专业会员,主要研究领域为调度及优化算法,非易失性存储.

谷守珍(1987－),女,博士,讲师,CCF 专业会员,主要研究领域为非易失性存储优化研究,多核并行任务调度,高性能存储优化技术.

沙行勉(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为并行计算和系统,嵌入式软件和系统,物联网和智能计算,先进存储,信息安全,调度及优化算法,实时系统,大数据处理,云计算.

诸葛晴凤(1970－),女,博士,教授,博士生导师,CCF 专业会员,主要研究领域为并行结构,存内计算,资源分配与调度,大数据处理系统,先进存储系统,嵌入式系统.

石亮(1987－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为计算机系统,嵌入式系统,操作系统,实时系统.

高思远(1996－),女,学士,主要研究领域为调度及优化算法,非易失性存储.