面向流体系的细粒度异步访存调度 *

张宇轩, 魏廷存, 樊晓桠, 张 萌
(西北工业大学 计算机学院, 西安 710072)

摘 要: 针对异步访存调度机制,设计一种细粒度化的调度方案以提升系统性能。该机制引入信号量和自旋锁,由异构核间协作运作,以实现对流级调度的局部加速。通过在一组测试程序集以及在对应平台上进行的实验,评估了引入该机制的加速效果,并分析了影响其性能的各种因素。

关键词: 流体系; 访存调度; 异步; 细粒度
中图分类号: TP301.6 **文献标志码**: A **文章编号**: 1001-3695(2012)03-0823-03
doi:10.3969/j.issn.1001-3695.2012.03.005

Stream-oriented fine-grain asynchronous memory access schedule

ZHANG Yu-xuan, WEI Ting-cun, FAN Xiao-ya, ZHANG Meng
(*School of Computer Science, Northwestern Polytechnical University, Xi'an* 710072, *China*)

Abstract: This paper focused on the asynchronous schedule mechanism, and presented a fine-grain solution to improve further efficiency. This solution involved semaphore and spinlock, to enable the cooperation of heterogeneous cores and the promotion of local speedup. This solution was experimented on the stream prototype system with a set of testbench, to evaluate the performance of the schedule solution, and analyzed related factors.

Key words: stream architecture; memory access schedule; asynchronous; fine-grain

随着 VLSI 技术的提升和对单核多发射架构并行性的发掘逐渐到达尽头,以及芯片集成程度趋近于现有工艺的极限,体系结构的设计开始转向于多核与众核。同时,编程模型也转而适应于多核与众核结构的并行体系以及流体系。流体系结构突破传统冯· 诺依曼架构局限,以其对数据和计算的重新理解,形成了不同以往的数据组织形式和并行化加速机制。在多媒体应用、科学计算甚至通用计算领域,流体系体现出了巨大的运算优势,并在近十年来成为体系结构设计关注的热点。

流体系中,批量同构有序的数据记录以流的形式载入和处理。流计算架构由向量技术发展而来,却对数据作了重新定义。这种计算分解、数据分块、计算与访存解耦合的特点,使得程序的并行性与数据局部性可以在很大限度上得到充分发掘[1]。典型的流体系中,如 Imagine,具有高数据传输带宽和存储层次、与 SIMD 和 VLIW 相结合的运算方式[2,3]。融合片上网络技术,Tile 化的流体系架构则将部分底层细节暴露给程序开发人员,以更好地解决片上数据传输[4]。而另一种流体系中则集成大量简单快速的算术部件和存储资源,提供可靠的高传输带宽,如现今 GPGPU 架构[5]。开发性能更高的流体系架构处理器向学术界与工业界不断提出了新的挑战。

流体系中,应用程序的访存与计算过程被显式地分解,并将数据的依赖关系暴露出来。异步访存调度是流体系的核心技术之一,为异步并发进行的流数据传输与处理提供支持[1,6]。本文关注于流体系中的异步访存调度技术。

Imagine 中异步访存调度将控制权交给协处理中间件,并以线程和任务同步技术作为辅助。其中,流的分割和其调度关系预先在编译阶段确定,并在程序运行时由中间件执行调度[7]。FT64 中采用分离存储空间的异步访存调度是对原有调度方法的进一步优化,更好地提高了异构核任务的并行效率[8]。此外,其他形式的流体系尽管结构不同,但仍然将这种中间件的调度方式作为解决方案[9]。在另一种流体系结构如 GPGPU 中,则利用其线程级并行的特点,使得任务之间可以切换运行,从而以另一种形式掩盖了数据传输与计算在占用时间上的重叠[10]。

这些调度方法都依赖于编译器或程序设计中人工地对数据的划分,并在运行时依靠中间件或程序本身的调度。本文提出一种面向于流体系的细粒度异步访存调度方案,作为对流级调度的辅助。其优势是提供对更小粒度访存调度的支持,使得流级数据传输与核心级程序可以更有效地并发运行,从而提升效率。

1 流处理器原型系统

西北工业大学计算机学院研制的龙腾流处理器原型系统,面向于短向量的适应性流体系。其中,以多套单指令多数据(single instruction multiple data, SIMD)模式的短向量簇(short vector cluster, SVC)实现程序的并行化,并以层次化的片上流存储体系捕获数据局部性。本文以该原型系统作为细粒度调度优化的实验平台。系统采用主/从处理器协同机制,其中以 NIOS Ⅱ作为主处理器,流处理器原型则作为协处理器以 Aval-

收稿日期: 2011-08-30; **修回日期**: 2011-10-26 **基金项目**: 国家"863"计划资助项目(2009AA01Z110);国家自然科学基金资助项目(60773223, 61003037, 60736012);西北工业大学研究生种子基金资助项目(Z2011119)

作者简介:张宇轩(1986-),男,陕西人,硕士研究生,主要研究方向为计算机系统结构(yoursever7@mail.nwpu.edu.cn);魏廷存(1960-),男,教授,博士,主要研究方向为模拟与混合信号 VLSI 设计;樊晓桠(1962-),男,教授,博士,主要研究方向为计算机系统结构;张萌(1978-),男,讲师,博士,主要研究方向为计算机系统结构.

on 从设备接入系统,如图 1 中所示。

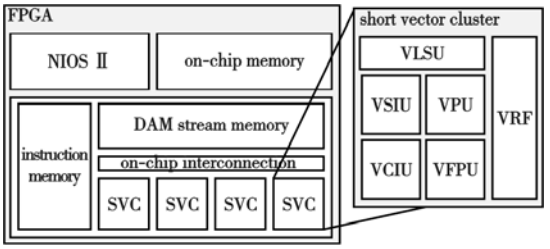

图1　龙腾流处理器原型

流处理器原型系统采用一种支持多位宽计算重构的 SVC 作为运算执行单元。每套 SVC 数据宽度为 128 bit,可根据指令类型动态重构为字节(16 路并行)、半字(8 路并行)、字(4 路并行)的运算操作。每套 SVC 分别具有各自独立的定点算术单元(VSIU)、置换操作单元(VPU)、复杂定点算术单元(VCIU)、浮点算术单元(VFPU)和访存单元(VLSU)。向量指令集支持多数算术运算和一些超越函数。

系统中,存储向量数据流的片上流存储器、标量数据存储器、指令存储器和寄存器组成了层次化的存储模型,如图 2 所示。其中流存储器由多套独立的 bank 组成,分别对应于多套 SVC。每个 bank 分别具有 128 bit 的数据接口,并以组间互连的方式支持向量的非顺序访存。流处理器原型系统中,每一套计算簇内则包含有各自独立的向量寄存器文件,作为向量指令的源和目标寄存器。

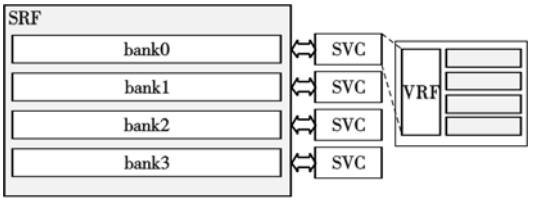

图2　流存储体系模型

流处理器原型系统上的流应用程序设计分为流级与核心级编程。流级程序组织数据并调度核心级程序的执行,核心级程序则以向量运算实现并行加速。

2　流调度策略

流体系设计的思想是计算与访存的解耦合。数据与指令以流的形式加载进入,依靠并行运算单元和片上存储层次充分发掘程序的并行性与数据局部性。

1)调度模型　传统的流调度方案中,数据流以粗粒度方式于运行时在片上流存储器上执行换入换出操作,整个过程依靠流调度中间件支持。图 3 和 4 中描述了两种简单的流数据与核程序的关系,输入流经过核程序处理后产生输出流;同样,核程序产生的输出流也可作为下一级核程序的输入流。

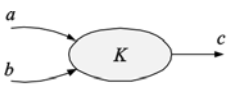

图3　流处理数据通路

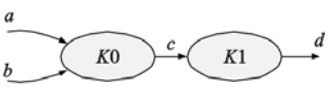

图4　涉及核程序间数据依赖的数据通路

流级的调度对分段的数据进行分析和控制,核程序需要等待数据载入后才可执行,如图 5 所示。而在多数的流应用中,核程序对流数据的访问往往可以在对流的载入过程时进行。这在规则流中最为显著,即对流数据的操作不必等待流完全载入,而在流的载入过程中便可同时处理已载入的部分,如图 6 所示的工作方式。当主处理器与作为协处理器的流处理器原型以一种高内聚的共享片上存储方式实现数据同步时,流级数据的组织与核心程序的执行,即流的生产/消费关系,便可以在更细的粒度上实现运行时间上的叠加。

2)细粒度异步访存调度机制　它对调度效率提出了更高的要求,传统软件调度方式由于速度的局限会使得调度成为负载。因此,在硬件辅助支持下,由异构核协作是一种快速可行的解决方法。

考虑到短向量 SIMD 的执行模式,顺序存储的流数据是根据存储位置批量操作的,因此,系统中将流存储器上的一个基本分块对应于连续地址上的定长度的多行,作为异步访存调度的基本单位对数据细粒度化的调度机制设计如图 7 所示。

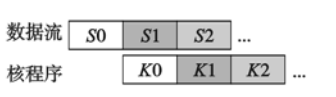

图5　流级的异步任务调度

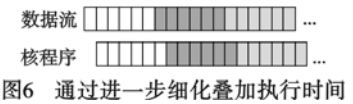

图6　通过进一步细化叠加执行时间

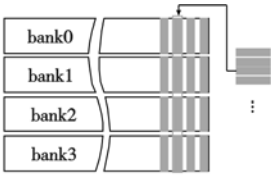

图7　对数据细粒度化的调度机制设计

为实现更细粒度的流调度操作,该机制中引入了一种标记流存储器区域状态的状态描述表。系统中,状态描述表由一组寄存器组成,其每一项对应于流存储器中的一个基本块,记录该块中数据的描述信息,并作为访问该块中记录的互斥锁。状态描述表同其他片上存储器一样,具有对主处理器可见的访存编址,又同时可被协处理器访问。在任何一个时刻,主/协处理器不能同时对该块中的数据进行读写访问。

该机制中以连续地址上的多行数据为一个数据块作为最小调度单位,一个数据块仅与其所在数据流以及访问该块内数据的核程序关联。可以仅认为是一种单一的生产/消费关系,这种关系也可用信号量的形式描述。当连续数据流被载入时,异步执行的流级数据传输与核程序分别作为生产者与消费者,设计中用信号量操作描述这种关系[11]。

数据流由主机端载入流存储器,在核程序中处理。流存储器中的数据分块作为异步处理中的临界数据,并由对应于该块的状态寄存器描述其就绪状态。在一个分块载入过程完成时,由主机端将对应于该块的状态置位,以表示数据就绪。同时,访存指令被扩展为等待数据就绪的自旋锁,以支持指令流水线阻塞和等待数据就绪。等待过程中,访存指令将使流水线阻塞,并在每一个指令周期查询其所要访问的数据块状态,直到其状态就绪。

在数据流载入的过程中,数据块的状态将随其载入就绪而置位有效;当核程序执行结束,这些输入流的数据块的状态位将被统一复位为无效。程序的执行过程中,这样的操作将同数据流与核程序的产生及消除一同被循环执行。

对于顺序写入的输出流,在核程序中由扩展指令设置其状态,由流级程序以同样的粒度分块读出。同样,这种方式也适用于图 6 中的依赖关系,不同的是在核程序间因资源限制未能实现异步执行。

3　实验与分析

基于 S2C S4 TAI Logic Module,实验在 Altera Stratix IV 上测试验证。FPGA 工作频率为 100 MHz,总集成度为 600 万等效逻辑门。测试程序集 C 程序文件的编译和运行环境借助于 Altera NIOS Ⅱ IDE,核心级程序则交由核心级指令汇编器转换

为二进制码。实验中均采用硬件计数器在 FPGA 工作频率下统计程序执行节拍数。

1)测试程序详细信息　表 1 中列出了用于性能分析的测试程序集,其中包括向量点积、矩阵乘法、FIR 滤波、中值滤波,涉及单一的算术程序、科学计算以及多媒体应用。表中列举了这些计算问题的类型、数据量和计算量。测试程序的设计均为流编程模式,核程序编写采用第 1 章中所提到的向量指令集。其中,mmul 和 mid 属于计算密集型,相对而言 dot 和 fir 属于存储密集型。

表 1　测试程序集详细信息

testbench	description	data size	Ops
dot1K	向量点积	3 × 1024	1024
dot4K	向量点积	3 × 4096	4096
mmul16	矩阵乘法	3 × 16 × 16	16^4
mmul64	矩阵乘法	3 × 64 × 64	64^4
fir	FIR 滤波	263 × 16	4096
mid	中值滤波	84 × 9 × 16	9225

2)性能分析　本文中细粒度的访存调度方式作为在流级调度基础上的局部加速,对批量数据的分段载入在流程序中完成。尽管这种访存调度的实现方法避免了因流级数据细分而产生的核函数的频繁调用代价和程序组织代价,但相比大批量完整的流数据传输所能充分发挥的批量数据传输优势,细分的数据传输却会使数据传输效率明显降低。

同时,同步代价也是影响整体效率的一个因素,过于频繁的互斥操作和信号量操作会严重损害系统的数据吞吐率。而对最小的访存调度粒度的选取需对这些决定系统性能的因素作综合的权衡。在该实验环境中,应用程序执行性能将会因访存粒度的不同而受到一定影响。当粒度过小时,并不会因此得益于调度方案,反而会因频繁的同步和数据分解代价而使得应用程序性能受到严重影响。

图 8 中显示了将访存调度应用于实验平台后的性能加速效果。在本实验所选的六个应用中,总的执行周期数均得到一定减小。其中,在对 mid 的测试中得到了最好的优化效果。图 8 中 R4、R16、R64 分别表示了在粒度选取为 4 行、16 行、64 行向量数据的情况下,运行测试程序集得到的加速效果。实验结果表明,在 R16 和 R64 中均可达到 1.1 ~ 1.3 之间稳定的加速比,两者效果接近,R16 效果略高;而在 R4 中,尽管在一些测试程序中可以得到明显高于其他的加速比,但整体性能不稳定且偏低,甚至小于 1。这一方面与实验平台有关,即在该平台下的实验对数据传输敏感;另一方面也与测试程序的算法本身以及代码设计相关。

同步模式下对测试程序结构的统计分析如图 9 所示。可以看到,受限于总线数据传输效率,流级数据传输相对核程序运行成为了系统性能的瓶颈。数据传输本身对系统的影响更为显著,因此,性能的提升来自于核程序执行时间的隐藏。

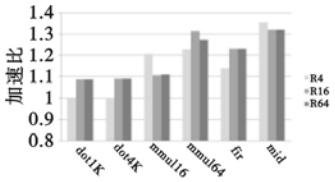

图8　加速效果分析

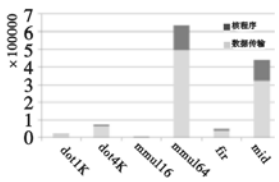

图9　测试程序结构组成

对测试程序的分析也可以看出,决定优化效果的原因首先与流级程序结构相关,即核程序与数据组织所占比例相关;其次,与程序应用目的相关,即对数据的利用方式;最后,也与核程序中访存指令所占比例相关,甚至与访存指令分布情况相关。图 10 和 11 中分别显示了访存指令在核程序中所占比例以及核程序占总执行时间的比例。此外,优化的流级程序设计会降低调度的开销,而优化的核程序设计也会相应减小流处理器等待和阻塞的代价。

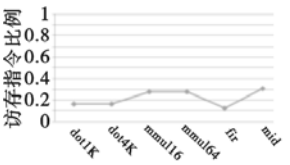

图10　访存指令在核程序中所占比例

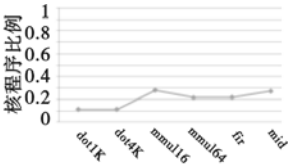

图11　核程序占总执行时间比例

4　结束语

本文介绍了一种面向流体系的异步访存调度方法的设计。这种调度机制以基本的信号量和互斥操作作为实现的基础,并通过软/硬件协同的工作实现快速的运行时细粒度异步调度。以龙腾流处理器原型系统作为对该方法的实验评估平台,对一组流应用程序的测试结果表明,该方法可以稳定高效地提升系统整体性能。此外,本文讨论了与调度方法以及加速效果相关的因素,并通过一组实验分析了这些因素对性能的影响。

参考文献:

[1] RIXNER S. Stream processor architecture[M]. Boston: Kluwer Acadamic Publishers, 2001.

[2] RIXNER S, DALLY W J, KAPASO U J, *et al*. A bandwidth-efficient architecture for media processing[C]//Proc of the 31st Annual ACM/IEEE International Symposium on Microarchitecture. New York: ACM Press, 1998: 3-13.

[3] KHAILANY B, DALLY W J, KAPASI U J, *et al*. Imagine: media processing with stream[J]. IEEE Micro, 2001, 21(2): 35-46.

[4] TAYLOR M, KIM J, MILLER J, *et al*. The raw microprocessor: a computational fabric for software circuits and general-purpose programs[J]. IEEE Micro, 2002, 22(2): 25-35.

[5] NVIDIA. Fermi compute architecture[R]. 2009.

[6] DAS A, DALLY W J. Stream scheduling: a framework to manage bulk operations in memory hierarchies[C]//Proc of the 14th International Conference on Paralled Architecture and Compilation Techniques. 2008: 337-349.

[7] LABONTE F, MATTSON P, THIES W, *et al*. The stream virtual machine[C]//Proc of the 13th International Conference on Parallel Architectures and Compilation Techniques. 2004: 267-277.

[8] WEN Mei, WU Nan, ZHANG Chun-yuan. On-chip memory system optimization design for the FT64 scientific stream accelerator[J]. IEEE Micro, 2008, 28(4): 51-70.

[9] GUMMARAJU J, COBURN J, TURNER Y, *et al*. Streamware: programming general-purpose multicore processors using streams[J]. ACM SIGARCH Computer Architecture News, 2008, 36(1): 297-307.

[10] NVIDIA. CUDA C best practices guide[R]. 2010.

[11] DIJKSTRA E W. Cooperating sequential processes[M]//The Origin of Concurrent Programming. New York: Springer-Verlag, 2002: 65-138.