

基于 GPU 的快速二维沃尔什变换研究

童 莹, 张 健

(南京工程学院 通信工程学院, 江苏 南京 211167)

摘 要: 提出了一种基于 GPU(Graphics Processing Unit, 图形处理器)CUDA(Compute Unified Device Architecture, 计算统一设备架构)平台的快速二维沃尔什变换(Walsh Transform)实现方法. 该方法利用 GPU 的并行结构和硬件特点, 从算法实现、存储类型、逻辑构架设置等方面提高了沃尔什变换的运算速度. 实验结果表明, 随着图像分辨率的增加, 沃尔什变换在 GPU 上运行时间远低于 CPU, GPU 比 CPU 具有更明显的加速效果.

关键词: 图形处理器; 计算统一设备体系结构; 沃尔什变换

中图分类号: TP302.7

文献标识码: A

文章编号: 1000-7180(2011)01-0046-04

Research of Fast 2-D Walsh Transformation Based on GPU

TONG Ying, ZHANG Jian

(Dept. of Comm. Eng, Nanjing Institute of Technology, Nanjing 211167, China)

Abstract: Fast 2-D Walsh Transformation algorithm is presented based on NVIDIA's GPU which support Compute Unified Device Architecture(CUDA). On the basis of the parallel architecture and hardware characteristic of GPU, the paper introduces three methods to improve the implementation performance: optimizing algorithm, texture Storage technology, and setting up logic Device Architecture. The experiment result shows that with the increasing of picture resolution, the runtime of 2-D Walsh Transformation based on GPU is far fewer than on the CPU.

Key words: graphics processing unit; compute unified device architecture; walsh transformation

1 引言

由于 GPU 硬件结构采用了更多的 ALU(Arithmetic Logic Unit, 运算逻辑部件)参与数据处理, 运算过程中 GPU 能进行大量的并行处理器, 因此, GPU 更适合那些以相同程序处理大量数据的并行计算, 即高密度数据运算.

图像变换是实现图像特征提取、图像增强、图像复原、图像压缩的基础. 常用的正交变换有傅里叶变换、余弦变换和沃尔什变换等. 这些变换都是对图像数据用相同的变换核进行矩阵运算, 符合 GPU 并行计算的特点. 但由于傅里叶变换和余弦变换的变换核由正弦、余弦函数组成, 运算速度较慢, 在实际应用中需要更高的运算速度时, 往往采用沃尔什变

换. 该变换核矩阵简单, 只有 1 和 -1, 占用存储空间少, 且容易产生, 因此得到广泛应用.

本文结合 GPU 的硬件结构和算法特点, 从算法实现、存储类型、逻辑构架设置等方面对二维沃尔什变换进行了优化, 实现了基于 GPU 的高效快速沃尔什图像变换.

2 CUDA 架构及实现平台介绍

NVIDIA 公司针对 GPU 的通用运算, 提出的 GPU 与 CUDA 相结合进行并行运算的方法, 可以通过片上(on-chip)上百个处理器同步协作, 从而快速解决复杂的运算问题. CUDA^[1-2]是把 GPU 作为并行运算设备进行程序发布和管理运算, 并且不需要将计算映射到图形应用程序接口的硬件和软件的

收稿日期: 2009-12-19; 修回日期: 2010-03-12

基金项目: 江苏省基础 Research 计划(自然科学基金)(BK2009351); 南京工程学院重点课题基金(KXJ07014); 引进人才科研启动基金(KXJ07-67)

架构.而当CUDA和图形应用程序同时访问GPU时的任务调度管理则交由操作系统的多任务机制负责.

CUDA为了便于算法的编程实现,定义了多个架构,包括内存架构、逻辑架构、物理架构.

CUDA内存架构,将内存定义为主机内存(Host Memory)和设备内存(Device Memory),而设备内存又分为片上内存和片外内存,前者包括共享内存(Shared Memory)、寄存器、局部内存(Local

Memory),后者包括全局内存(Global Memory)、常量内存(Constant Memory)和纹理内存(Texture Memory).

CUDA逻辑架构,分为网格(Grid),块(Block)和线程(Thread),它们的结构组成可以是多维的.其中,Thread是最小的逻辑运算单位,多个Thread组成Block.同样的,多个Block组成Grid. CUDA逻辑架构和设备内存的对应关系如图1所示.

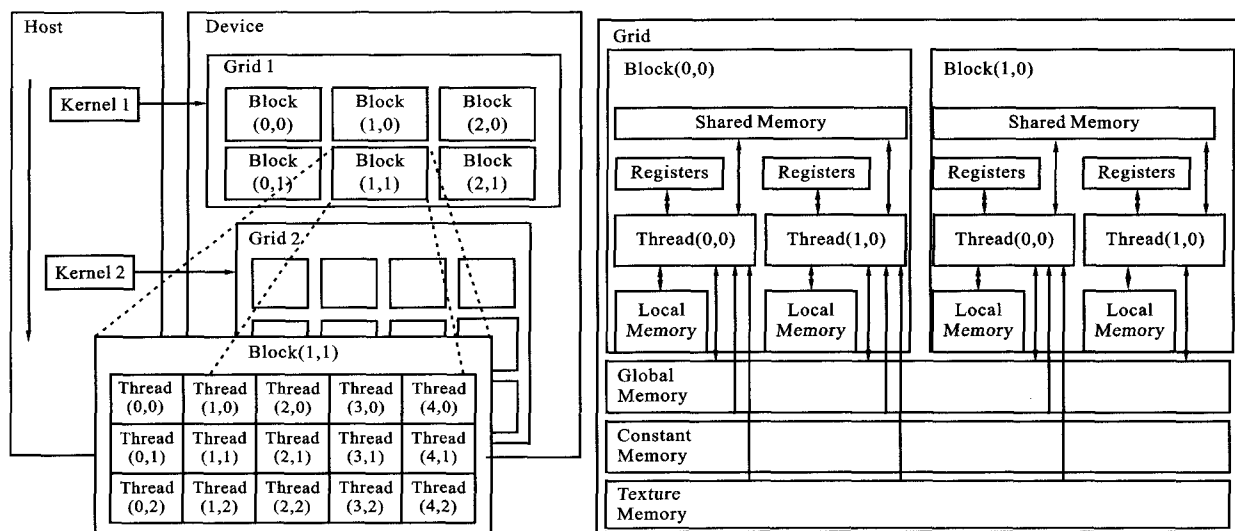


图1 CUDA逻辑架构及其与设备内存的对应关系

逻辑架构和物理架构的对应关系是,每个处理器组可以并发处理一个或多个Block,前提是这些Block可以分到足够的运算资源(如Register和供高速数据读取用的Shared Memory),因为这些运算资源是有限的.而Block中多个连续的Thread又会组合成新的逻辑执行单位——Warp(规定为32个Thread),协处理器在处理这些Warp的时候采用CUDA固有的轮询和调度机制来处理这些Thread.另外CUDA规定,逻辑上的Thread可以同步也可以实现数据互相访问和通信,因为它们共享同一块数据区域(Shared Memory);而Block之间的数据则无法同步也无法实现互相访问和通信,这是出于对不同运算平台有不同运算资源考虑的,对于运算资源较少的GPU平台,可以顺序处理完所有的Block.

3 编程方法

当用CUDA编程实现的时候,GPU可以看作并行处理很多thread(线程)的一个运算设备,也就是将程序的数据并行和密集运算部分分配给GPU

来执行^[3-4].更准确地说,如果一个程序的某个函数对于不同的数据要重复执行多次,这样就可以把这部分函数独立出来,对应到GPU的许多不同的thread来执行.也就是将这个函数编译为在GPU上执行的指令集,而这个编译的目标程序称作kernel.

多个thread组成一个thread block,它们可以共享shared memory中的数据,并可以同步执行一致的内存访问.即一个thread可以在kernel中定义一个同步点,则所有同属于当前block中的其他thread必须在此同步点进行等待,当此block中所有thread都达到这个同步点时,再执行同步点后面的操作.

每个thread都有threadID与其对应,这是其在block中的线程序号.为了便于这种基于threadID的复杂寻址,也可以将block定义成任意大小的2D或3D的矩阵,这样就可以通过2元或3元的索引来表示其threadID.例如,对于2D,大小为(Dx, Dy)的block,则索引为(x, y)的threadID为(x + y * Dx);对于3D大小为(Dx, Dy, Dz)的block,索

引为 (x, y, z) 的 thread 的 threadID 为 $(x + y * Dx + z * Dx * Dy)$.

4 基于 GPU 的快速二维 Walsh 变换实现

4.1 Walsh 变换原理

Walsh 函数是美国数学家 Walsh 于 1923 年提出的一类完备的正交函数系,其最大的特点是每个函数仅取 $+1, -1$ 两个值,比较适合用来表达和处理数字信息,因而在信号处理、图像处理、通信及计算机等众多领域得到了广泛的应用. Walsh 函数是一个 Haar 小波包,延续了小波在信号处理中的一些优势;同时与 DFT、DCT 相比, Walsh 变换的计算量很小,执行效率高. Walsh 函数常见的有三种排序方式: Paley 序、Hadamard 序(反 Paley 序)、Walsh 序. H 序因其自相似性强、分块递推、生成简单、计算简洁而得到广泛的应用^[5-10]. 因为 Walsh 函数的三种排序之间可以相互转换,因此本文重点介绍 H 序的 Walsh 变换.

设 $N = 2^n$, 则离散数据 $f(x) (x = 0, 1, 2, \dots, N-1)$ 的一维离散 Walsh 变换为

$$H(u) = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} f(x) (-1)^{\sum_{i=0}^{n-1} b_i(x) b_i(u)} \quad (u = 0, 1, 2, \dots, N-1) \quad (1)$$

式中, $b_i(x)$ 表示为 x 的二进制数的第 $i+1$ 位值(从低位往高位数). 反变换为

$$f(x) = \frac{1}{\sqrt{N}} \sum_{u=0}^{N-1} H(u) (-1)^{\sum_{i=0}^{n-1} b_i(x) b_i(u)} \quad (x = 0, 1, 2, \dots, N-1) \quad (2)$$

同理,二维离散 Walsh 正反变换为

$$H(u, v) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot (-1)^{\sum_{i=0}^{m-1} b_i(x) b_i(u) + \sum_{j=0}^{n-1} b_j(y) b_j(v)} \quad (u = 0, 1, 2, \dots, M-1, v = 0, 1, 2, \dots, N-1) \quad (3)$$

$$f(x, y) = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} H(u, v) \cdot (-1)^{\sum_{i=0}^{m-1} b_i(x) b_i(u) + \sum_{j=0}^{n-1} b_j(y) b_j(v)} \quad (x = 0, 1, 2, \dots, M-1, y = 0, 1, 2, \dots, N-1) \quad (4)$$

式中, $M = 2^m, N = 2^n$.

由式(1)~(4)可以看出,离散 Walsh 正反变换核相同,因此与 DFT、DCT 相比,可以节约存储空间,降低数据从内存到 GPU 的拷贝时间,提高运算

效率^[11-12]. 同时, H 序的 Walsh 变换核生成具有一定

规律,即 $H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix}$, 因此可以将变换核

依次展开至 $H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, 由离散值的相加减得

到最终 Walsh 变换值. 以 $N = 8$ 为例,快速一维 Walsh 变换的蝶型图如图 2 所示,图中描述 $X(0) = f(0) + f(4)$, $X(4) = f(0) - f(4)$, 依次类推得到变换值 $H(u)$. 由此我们得到快速二维 Walsh 变换实现步骤:

(1) 先将二维矩阵 $f(x, y)$ 按列顺序排成一维列向量 $g(x)$;

(2) 对一维列向量数据 $g(x)$ 进行快速一维 Walsh 变换;

(3) 最后将所得结果转换成二维矩阵形式 $H(u, v)$, 即二维矩阵 $f(x, y)$ 的 Walsh 变换.

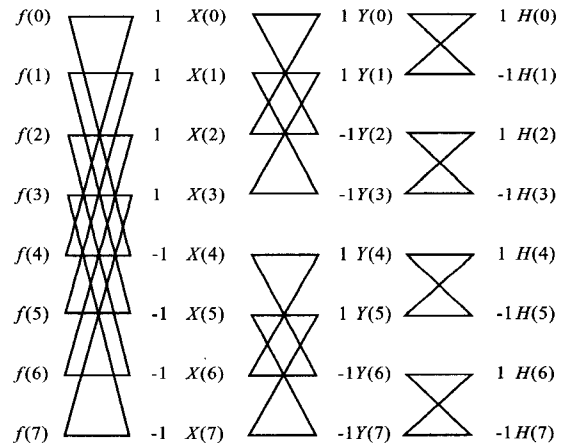


图 2 $N=8$ 快速 Walsh 变换蝶型图

4.2 二维 Walsh 变换在 GPU 上的优化实现

由式(3)可以看出,二维离散 Walsh 变换正反变换核相同,虽然相比 2DFT, 2DCT 运算效率有所提升,但当图像分辨率增加,即二维矩阵增大时,运算量增加,二维离散 Walsh 变换在 CPU 上的运行效率仍不够理想,因此,如何在 GPU 上实现快速二维 Walsh 变换成为研究重点.

首先,对于 GPU 上的二维离散 Walsh 变换实现算法,我们采用本文提出的快速二维 Walsh 实现步骤,该方法的运算量为 $O(N \log_2 N)$ (N 为数据量),相比传统运算量 $O(N^2 + N(N-1))$ 降低很多.

从快速二维 Walsh 变换实现步骤可以看出,高密度数据运算主要集中在快速一维 Walsh 变换部分,根据快速一维 Walsh 变换的原理,我们发现 Walsh 变换中的蝶形运算,级与级之间是串行的,但

每级内的蝶形运算是独立的,且运算相同,因此本文将每级蝶型运算作为一个 kernel. 考虑到图像分辨率较高时,一维列向量 $g(x)$ 数据较多,为了更好地利用 GPU 的并行性,我们将 $g(x)$ 分成多个块,每个块单独执行快速 Walsh 变换,而块与块之间是独立的. 因此定义一个 Block 执行一组数据的蝶型运算,该组运算由多个 Thread 共同完成,多个 Block 则完成整个数据的蝶型运算,即一个 Grid. 设分组数据的大小为 KernelN, $g(x)$ 数据大小为 DataN,则完成整个数据的 Walsh 变换需要的运算量为 $O(\text{DataN} * \log_2 \text{KernelN})$,由此可见,当数据在 CPU 上顺序执行时, Walsh 变换的运算时间随着 KernelN 的增大而增大,而 GPU 的并行性则大大降低顺序等待运算消耗.

另一方面,纹理存储是 GPU 定义的一种存储类型,它不同于只支持一维寻址的传统缓存,它支持二维寻址,因此为图像处理提供了优化. 纹理存储是一组高速缓存,能够保存最近访问的数据,从缓存中访问数据与访问 GPU 寄存器的速度相当. 本文需要多次访问图像中的数据进行快速 Walsh 变换,因此采用纹理类型存储图像数据可以大大提高线程对数据访问速度.

5 实验结果与分析

本文所采用的实验平台是 Intel(R)Core(TM) 2 CPU 6320, 系统内存 1GB, 显卡为 GeForce GTX260, 共有 192 个流处理器, 核心频率 576 MHz, 显卡内存 896 MB. 操作系统为 Windows XP, 整个实现基于 CUDA SDK 2.0. 我们通过调整图像分辨率, 得到 GPU 和 CPU 运行时间结果如表 1 所示(取 KernelN=128).

表 1 2-DWT 在 GPU 和 CPU 上的运算时间及其比值

Picture resolving	GPU time/ms	CPU time/ms	CPU/GPU
128 * 128	1.033 646	3.466 180	3.35
256 * 256	1.150 604	15.238 38	13.24
512 * 512	1.649 824	55.579 06	33.69
1024 * 1024	3.574 999	222.896 3	62.35
2048 * 2048	11.961 278	893.577 3	74.71
4096 * 4096	49.726 437	3 574.786 4	71.89

从表中我们可以看出,随着图像分辨率的增加, GPU 比 CPU 有明显的加速效果,当图像分辨率较小时,加速效果不明显,这是因为分辨率较低时,每个线程的运算量不充分,线程切换频繁,系统调度的

开销占用了较多执行时间,当分辨率较高时,每线程处理数据增多,系统调度占用的执行时间也较少,当图像分辨率为 2048 * 2048 时,线程利用率达到充分,加速比为 74.71.

由前面分析知道 KernelN 大小会影响 CPU 运行时间,取图像分辨率为 512 * 512,改变 KernelN 大小,得到 GPU 和 CPU 的运行时间,如表 2 所示. 从表中可看出,当 KernelN 增大时, GPU 的运行时间基本不变,而 CPU 的运行时间则显著增长. 当 KernelN 较小时, GPU 线程切换与 CPU 等待时间基本抵消,两者时间相差不大,而当 KernelN 较大时, CPU 的等待时间则明显增长, GPU 的并行优越性体现出来.

表 2 不同 KernelN 下 GPU 与 CPU 性能

KernelN size	GPU time/ms	CPU time/ms	CPU/GPU
4	1.654 80	3.551 858	2.15
8	1.655 043	5.831 583	3.52
16	1.650 858	8.705 646	5.27
32	1.652 076	15.238 38	9.22
64	1.650 536	28.578 97	17.31
128	1.649 824	55.579 06	33.69

6 结束语

本文结合 GPU 的硬件结构和 2-DWT 算法特点,从算法实现、存储类型、逻辑构架设置等方面对二维 Walsh 变换进行了优化,实现了基于 GPU 的高效快速二维 Walsh 变换,对图像变换的实时性具有推动作用. 该算法实现时采用的技术具有通用性,同样适合于 DFT、DCT 在 GPU 上的快速实现.

参考文献:

- [1] NVIDIA Corporation. Cuda Programming Guide Version 2.0[M]. NVIDIA Corporation, 2008.
- [2] Krüger J, Westermann R. Linear algebra operators for GPU implementation of numerical algorithms[J]. ACM Trans on Graphics, 2003, 22(3): 908—916.
- [3] 托马斯·布劳恩. 并行图像处理[M]. 李俊山, 译. 西安交通大学出版社, 2003.
- [4] 吴恩华, 柳有权. 基于图形处理器(GPU)的通用计算[J]. 计算机辅助设计与图形学学报, 2004, 13(5): 601—612.
- [5] 李俊山. 数字图像处理[M]. 北京: 清华大学出版社, 2007.

(下转第 53 页)

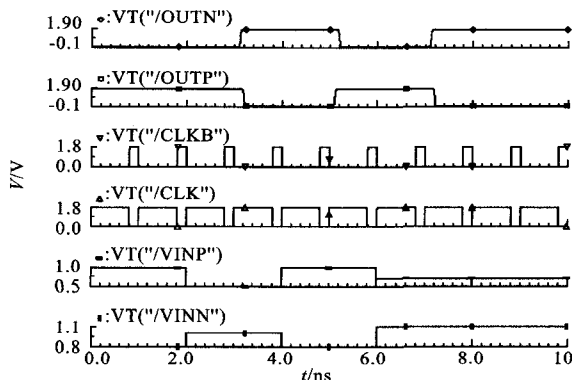


图 5 比较器的功能仿真结果

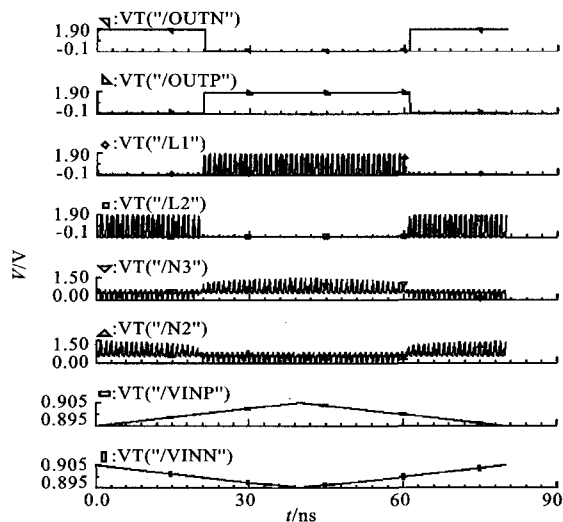


图 6 比较器的精度仿真结果

5 结束语

在分析预放大锁存比较器的基础上,文中设计了一个高速高精度的比较器.本文利用了负载电阻

与负电阻并联的方法,以提高预放大器在高带宽下的增益.同时,采用提供预设电流的方式增大再生级的驱动能力,以降低传输延迟时间.仿真结果表明,该比较器可以稳定工作在 1 GHz 的时钟频率下,最小分辨率为 0.6 mV,传输延迟时间为 320 ps.当输入信号范围为 1 V 时,能够用于 10 Bit 1GSPS 的高速 ADC.

参考文献:

- [1] 徐世六. 模拟/数字混合信号电路技术发展动态[J]. 微电子学, 2008, 38(1): 26-33.
- [2] Behzad Razavi, Bruce A Wooley. Design techniques for high-speed, high-resolution comparators[J]. IEEE Journal of Solid-State Circuits, 1992, 27(12): 1916-1926.
- [3] Allen Philip E. CMOS Analog Circuit Design[M]. 2nd ed. 北京: 电子工业出版社, 2003.
- [4] 高彬, 孟桥, 沈志远. 基于 0.18 μm CMOS 工艺的超高速比较器[J]. 微电子学, 2007, 37(4): 599-602.
- [5] 陈诚. 千兆以太网中低电压高速模数转换器研究设计[D]. 上海: 复旦大学, 2005.
- [6] Mehdi Banihashemi, A High-speed high-resolution Comparator[C]// The 47th IEEE International Midwest Symposium on Circuits and Systems. Hiroshima: IEEE, 2004, : 81-84.
- [7] Pelgrom M J M, Duinmaijer A C J, Welbers A P G. Matching Properties of MOS Transistors[J]. IEEE Journal of Solid-State Circuits, 1989, 24(5): 1433-1440.

作者简介:

郭永恒 男, (1983-), 硕士研究生. 研究方向为高速 ADC 设计.

(上接第 49 页)

- [6] 容观澳. 计算机图像处理[M]. 北京: 清华大学出版社, 2000.
- [7] 胡辉, 叶鑫华. 离散 Walsh 变换并行性分析与实现[J]. 计算机工程与应用, 2009, 45(2): 82-84.
- [8] Swick D A. Walsh function generation[J]. IEEE Trans on Information Theory. 1969, IT-15(1): 167-167.
- [9] 陈宁涛, 王能超, 施保昌. Walsh 矩阵的复制生成及其计算机图像[J]. 软件学报, 2006, 17(5): 1241-1250.
- [10] 郭卫斌, 王能超, 施保昌. M 序 Walsh 变换的快速算法设计[J]. 小型微型计算机系统, 2003, 24(2): 295-298.
- [11] 阮军, 韩定定. 基于 CUDA 的 DCT 快速变换实现方法

[J]. 微电子学与计算机, 2009, 26(8): 201-205.

- [12] 戴声奎, 刘建国, 汪国有, 等. 一种无乘法的整数 DCT 快速实现方法[J]. 微电子学与计算机, 2008, 25(5): 11-13.

作者简介:

董莹 女, (1979-), 硕士, 讲师. 研究方向为图形图像处理.

张健 男, (1977-), 博士, 讲师. 研究方向为计算机图形学、并行计算.