

基于 GPU 的信号产生及脉冲压缩实现

金 莉, 孔文青, 宋万杰

(西安电子科技大学雷达信号处理国家重点实验室, 陕西西安 710071)

摘 要: 文中采用了一种基于 CPU+GPU 异构并行架构体系的信号处理方案。按照雷达信号处理流程, 通用处理计算机利用 CPU 串行代码完成核函数启动前数据准备和设备初始化工作, 并控制信号处理的任务调度和负载分配, 然后将数据通过 PCI-E 总线传输至显存, 利用 GPU 特有的单指令多线程方式, 并行实现线性调频信号产生以及线性调频信号频域脉冲压缩算法, 并与 CPU 进行比较。实验结果表明, 利用计算统一设备架构技术实现的线性调频信号产生以及脉冲压缩算法取得了比 CPU 更高的运算效率。

关键词: 计算统一设备架构; 图形处理机(GPU); 线性调频信号; 脉冲压缩

中图分类号: TN958.3; TN957 文献标志码: A 文章编号: 1672-2337(2017)05-0505-04

Signal Generation and Pulse Compression Realization Based on GPU

JIN Li, KONG Wenqing, SONG Wanjie

(National Key Laboratory of Radar Signal Processing, Xidian University, Xi'an 710071, China)

Abstract: A signal processing scheme based on CPU+GPU heterogeneous parallel architecture is proposed in this paper. According to radar processing flow, the general computer uses CPU to complete data preparation and device initialization before kernel function starting, and control the task scheduling and load sharing of signal processing, then, the data will be transmitted through the PCI-E bus to the memory of GPU. The specific way of SIMT of GPU is used to achieve thread parallelization, and finally finish the LFM signal generation and pulse compression algorithm. A comparison with the algorithm realized on CPU is made. The experimental results show that the algorithm achieved by CUDA has higher efficiency than CPU.

Key words: compute unified device architecture(CUDA); graphical processing unit(GPU); LFM signal; pulse compression

0 引 言

高性能计算(High Performance Computing, HPC)作为计算机科学的一个分支,随着人们对信息处理能力要求的日益提高,成为现代社会中不可或缺的一部分。高性能计算主要通过并行计算来实现,目前所采用的并行计算包括 CPU 多核并行、超级计算机、集群与分布式计算、CPU+GPU 异构并行等。其中,通过 CPU 多核并行、超级计算机、集群与分布式计算主要实现海量数据处理,但其具有硬件成本高、开发难度大等缺点。而基于 CPU+GPU 异构的软件化雷达就大大降低了成本,简化了硬件结构,与传统基于硬件的雷达相比,软件化雷达更加简单通用,显示器作为外设,整套系统

体系的硬件由一台通用计算机和雷达信息采集板构成,信号与信息处理及可视化完全由通用计算机与协处理器 GPU 来完成^[1-2]。而计算统一设备架构(Compute Unified Device Architecture, CUDA)是一种将 GPU 作为数据并行计算设备的软硬件体系,使专注于图像处理的 GPU 超级计算能力在数据处理和科学计算等通用计算领域广泛应用,也使其在软件化雷达中发挥独特优势^[3-4]。

此外, GPU 作为一种图形处理器显卡,在处理能力与存储带宽上相对 CPU 有绝对优势,对于同等规模运算处理,通过 GPU 加速的高性能计算可能从大型计算机过渡到台式机以及桌边型计算机上。同时,随着 GPU 技术的飞速发展,其并行计算能力也逐步增强,浮点运算能力甚至可以达到同代 CPU 的 10 倍以上^[5]。目前,随着成本与性能等方面的要

求逐渐提高, CPU 以摩尔定律的速度发展, 满足不了目前信息处理对精度、实时性、大数据的要求。相对而言, GPU 仍以差不多 3 倍于摩尔定律的增长速度发展, 平均每几个月其性能就提升一倍。这为未来大规模数据运算提供了新选择^[6]。

鉴于 GPU 强大的并行计算能力, 通过利用 VS 2013+CUDA 7.5 作为仿真平台, 实现雷达信号与信息处理, 并以线性调频信号的产生以及线性调频信号脉冲频域脉冲压缩为例, 与传统 CPU 运算性能进行比较, 验证 GPU 处理数据的正确性以及高效性。

1 线性调频信号及脉冲压缩原理

1.1 线性调频信号

线性调频信号是研究最早的一种脉冲压缩信号, 其产生和处理都比较容易, 技术上比较成熟, 因此得到了广泛的应用。它是在匹配滤波的基础上提出的, 突出优点是匹配滤波器对回波信号的多普勒平移不敏感^[7-8]。

假设雷达发射线性调频脉冲信号, 可表示为

$$s_1 = \text{rect}(t/T_e) \cos(2\pi f_0 t + \pi \mu t^2) \quad (1)$$

式中, T_e 为发射脉冲宽度, f_0 为中心载频, $\mu = B/T_e$ 为调频斜率, B 为调频带宽。

1.2 频域脉冲压缩基本算法

为了满足提高探测距离和距离分辨率的双重要求, 就要采取大的时宽带宽信号。脉冲压缩技术的出现有效地解决了雷达作用距离和距离分辨率的矛盾, 它既可以发射宽脉冲以提高平均功率和雷达的检测能力, 又能保持窄脉冲的距离分辨率。实现频域脉压的方法^[9]如图 1 所示。

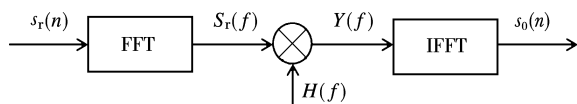


图 1 频域脉冲压缩示意图

从图 1 可以看出, 频域脉压的核心算法是快速傅里叶变换 FFT、快速傅里叶逆变换 IFFT 以及复数相乘运算。采用频域脉冲压缩方法相对时域卷积而言, 其运算量将大为减少, 而且脉冲压缩时可利用加窗函数来抑制旁瓣。只需将匹配滤波器系数预先进行频域相乘(频域加窗)或者时域相乘(时域加窗), 即

$$H(f) = \text{FFT}\{s(n) \cdot w(n)\} \quad (2)$$

式中, $w(n)$ 为窗函数, 可以根据需要选取合适的窗函数。需要注意的是, FFT/IFFT 的点数不是任意选取的。假设输入信号点数为 N , 滤波器阶数为 L , 那么经滤波后输出的信号点数应该为 $N + L - 1$, 即对于 FFT 的点数选择必须保证其大于等于 $N + L - 1$, 通常取 2 的幂对应的数值大于等于 $N + L - 1$ 。因此, 在对滤波器系数及输入信号 $s_r(n)$ 进行 FFT 之前, 要先对序列进行补零处理。

2 CUDA 编程模型

CUDA 编程模型将 CPU 及系统内存称为主机(Host), 而将 GPU 及其内存称为设备(Device)。在 GPU 上执行的函数通常称为核函数(Kernel)。CUDA C 通过某种语法方法(如 `_global_`)将一个函数标记为“设备代码”(Device Code), 这种简单的标记方法, 表示将主机代码发送到一个编译器, 而将设备代码发送到另一个编译器。而 CUDA 编译器在运行时负责实现从主机中调用设备代码。CUDA 线程并行分为线程格中线程块并行和线程块中线程并行两个层次^[10]。所有线程同时运行相同的 Kernel 并根据各自的线程 ID 接入不同的内存地址进行运算, 最终达到并行运算的目的。

CPU 完成预处理并控制信号处理的任务调度和负载分配, 按照雷达信号处理流程, 将数据通过 PCI-E 总线分块传输至显存, 利用 GPU 特有的单指令多线程(SIMT)方式实现线程并行化计算。每个脉冲重复周期所包含的采样数据处理方式相同, 充分利用 GPU 多线程、细粒度并行的优势进行处理。

简单来说, 一个 CUDA 程序的执行是由主机代码开始的, 主机代码主要完成初始化、数据输入等基本工作, 然后对设备端的显存进行分配, 将内存中数据通过 PCI-E 总线传入显存, 调用 Kernel 函数在设备端完成运算, 最后将运算结果传回主机并释放所分配内存。利用 CUDA 实现线性调频信号产生及频域脉压, 主要包含以下步骤:

第一步: 在 CPU 端进行数据初始化并为数据在设备端分配内存。

第二步: 在 GPU 上产生线性调频信号以及滤波器系数, 这里增加窗函数来抑制旁瓣。

第三步: 利用 CUDA 提供的 CUFFT 库将产生

的回波信号序列 $s_r(n)$ 和滤波器系数 $h_r(n)$ 进行快速傅里叶变换得到 $S_r(f)$ 和 $H(f)$ 。

第四步:调用复数相乘函数或者 CUDA 提供的 CUBLAS 向量运算库函数完成 $S_r(f)$ 和 $H(f)$ 的相乘运算。

第五步:利用 CUDA 提供的 CUFFT 库将运算结果进行逆傅里叶变换,即将运算结果从频域变换到时域。这里需要注意的是,时域到频域的运算结果有一个倍数关系,需要将转换结果乘以输入信号点数的倒数得到的才是最终转换的结果。

第六步:将运算结果从显存复制回主机内存,并释放相应内存和显存。

其算法流程如图 2 所示。

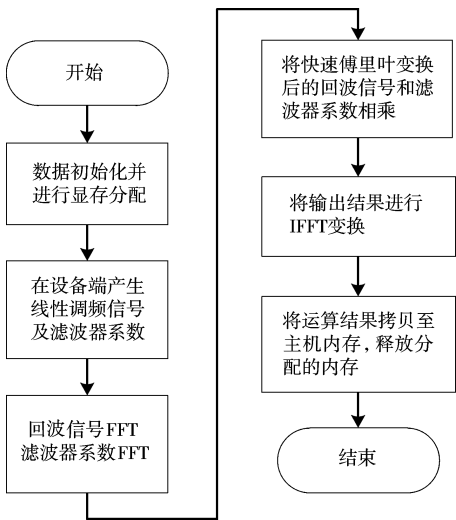


图 2 CUDA 程序实现流程

在 CUDA 程序中,主要完成的是主机与设备之间的数据传输以及内核函数的执行,所以,可以采用以下 3 种方法减少传输时间,加快运算速度,提高运算效率。

方法一:运算时采用共享存储器。共享存储器使得一个线程块中多个线程能够在计算上进行通信和协作,而且,共享内存缓冲区驻留在物理 GPU 上,因此,在访问共享内存时的延迟要远远低于访问普通缓冲区的延迟。

方法二:可以采用纹理存储器来保存一些滤波系数,使用纹理内核函数来读取纹理存储器。纹理存储器的存储方式为只读,它能够减少对内存的访问请求并提供更高效的内存带宽。

方法三:为提高数据传输带宽,可以对采样点开辟页锁定内存,从而确保了该内存始终驻留在

物理内存中,使得信号处理的内核函数可直接进行主机和设备端的通信,减少数据拷贝时间。

3 仿真结果及分析

3.1 LFM 信号产生仿真结果

仿真采用的操作系统为 Windows 7 SP1,显卡是计算能力为 2.0 的 NVIDIA Tesla C2050,支持双精度浮点运算。为了提高运算精度,数据处理均采用双精度浮点数。

在仿真中,假定线性调频信号的中心频率为 $f_0=0$ MHz,信号带宽 $B=2.5$ MHz,信号的时宽 $T=150\ \mu\text{s}$,采样频率 $f_s=15$ MHz。图 3 和图 4 分别为在 CPU 和在 GPU 上产生的线性调频信号。

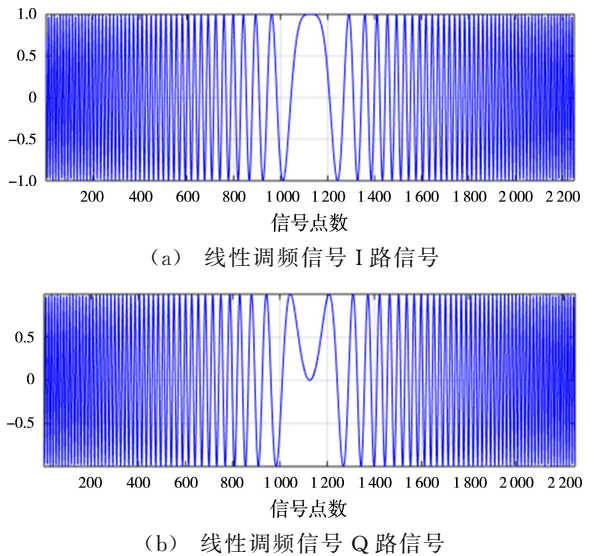


图 3 CPU 上产生的线性调频信号

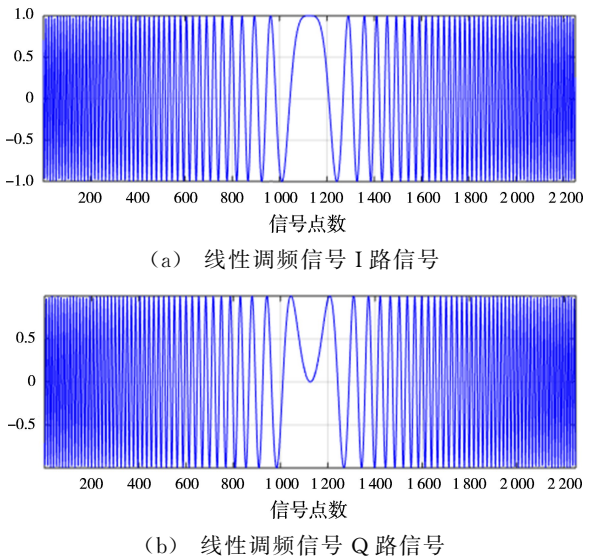


图 4 GPU 上产生的线性调频信号

3.2 频域脉压仿真结果

同样,在 GPU 和 CPU 上分别实现脉冲压缩,结果如图 5 和图 6 所示。

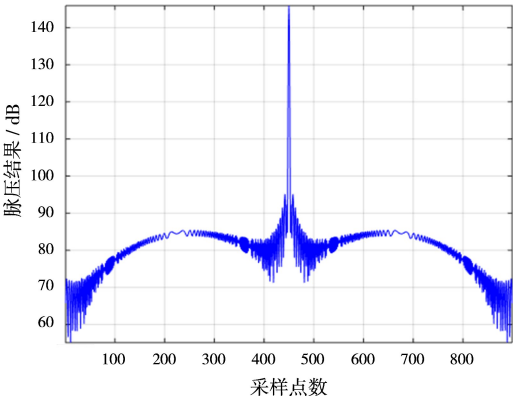


图 5 CPU 上脉冲压缩结果

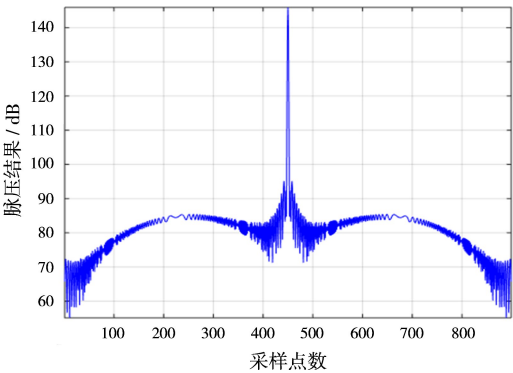


图 6 GPU 上脉冲压缩结果

从图 5 和图 6 可以看出,在 CUDA 平台上利用 GPU 产生线性调频信号以及频域脉冲压缩结果与 CPU 上处理的结果是一样的,但是通过测量,可以发现,两者的处理速度有很大差别。表 1 所示为处理相同数据时 CPU 与 GPU 所用的时间对比。

表 1 CPU 与 GPU 所用的时间对比

点数	CPU/ms	GPU/ms	时间比
1 024	2.827	0.049	57.69
2 048	4.569	0.063	72.52
4 096	11.39	0.070	162.7
8 192	15.22	0.082	185.6
16 384	29.11	0.098	297.1

从表 1 数据可以看出, GPU 比 CPU 的数据运算效率更高,而且数据量越大, GPU 的优势越明显。需要说明的是, GPU 有很大一部分时间都用

在数据传输上,所以,在 GPU 运算过程中,应当尽量减少数据传输或者采用页锁定内存等方式减少数据传输时间。

4 结束语

实验表明,通过直接在 GPU 上产生线性调频信号,不仅达到了和 CPU 上一样的效果,而且节省了将数据复制到 GPU 显存的时间,提高了运算效率。直接通过 GPU 产生滤波器系数,大大减少了数据从 CPU 传输到 GPU 显存的时间,加快了运算速度。另外,在 GPU 上实现频域脉压时, GPU 上 FFT 和 IFFT 运算均可以采用 CUFFT 库。对于短点的 FFT 运算,由于数据量小,系统直接将所有数据都拷贝到共享内存中实现。但对于长点数的 FFT,由于共享内存容量有限,因此需要将一维长点 FFT 拆分为二维短点数 FFT,每个短点数的 FFT 与复乘运算均可以在共享内存中进行,由于共享内存访问速度远高于共享全局,因此在和 CPU 运算效果相同的情况下,相比 CPU 而言,可以大大提高运算效率。总的来说,利用 CUDA 处理数据很好地满足了目前信息处理对精度、实时性、大数据的要求。

参考文献:

[1] 张舒,褚艳利,赵凯勇,等. GPU 高性能运算之 CUDA [M]. 北京:中国水利水电出版社, 2009:5-8, 14-18.

[2] 裴颂文,宁静,张俊格. CPU-GPU 异构多核系统的动态任务调度算法[J]. 计算机应用研究, 2016, 33 (11):3315-3316.

[3] 陈伯孝. 现代雷达系统分析与设计[M]. 西安:西安电子科技大学出版社, 2012:189-190.

[4] 武勇,王俊,张培川,等. CUDA 架构下外辐射源雷达杂波抑制并行算法[J]. 西安电子科技大学学报(自然科学版), 2015, 42(1):104-105.

[5] SABDERS J, KANDROT E. GPU 高性能编程 CUDA 实战[M]. 聂雪军,等译. 北京:机械工业出版社, 2011:13-18.

[6] 商哲然,谭贤四,曲智国,等. 基于 GPU 的 RFT 算法并行化[J]. 雷达科学与技术, 2016, 14(5):505-506. SHANG Zheran, TAN Xiansi, QU Zhiguo, et al. Parallel Implementation of RFT Algorithm on GPU [J]. Radar Science and Technology, 2016, 14(5): 505-506. (in Chinese)

[7] 康乃馨,何明浩,王冰切,等. 基于压缩感(下转第 518 页)