

引用格式: Yang Renzhong, Zhang Tao, Lin Botao, *et al.* Optimization and Implementation of RD Algorithm in SAR Ground Quick Look System According to GPU Platform[J]. Remote Sensing Technology and Application, 2012, 27(2): 237-242. [杨仁忠, 张萄, 林波涛, 等. GPU 平台下针对 SAR 地面快视系统的 RD 算法优化与实现[J]. 遥感技术与应用, 2012, 27(2): 237-242.]

GPU 平台下针对 SAR 地面快视系统的 RD 算法优化与实现

杨仁忠, 张 萄, 林波涛, 韦宏卫

(中国科学院对地观测与数字地球科学中心, 北京 100094)

摘要: 针对 SAR 地面系统实时关键成像技术进行研究, 采用 GPU(Graphic Processing Unit) 硬件平台 CUDA(Computer Uniformed Device Architecture) 编程模型, 对传统合成孔径雷达的 RDA(Range Doppler Algorithm) 算法核心部分进行了针对性的设计与实现, 并在 GPU 专用科学计算平台 Tesla C1060 上进行了实验。结果表明其处理速度是一台主流 4 核心 8 线程 CPU 的 20 倍以上, 并且相对 RadarSat-1 卫星可以达到 10 倍左右的实时率; 基于 GPU 的处理方式较好地实现了 SAR 实时成像系统。

关 键 词: 合成孔径雷达(Synthetic Aperture Radar); 实时成像; 并行计算; 高性能计算

中图分类号: TP 75 **文献标志码:** A **文章编号:** 1004-0323(2012)02-0237-06

1 引 言

由于合成孔径雷达是主动式发射微波的观测方式, 具有全天候、全天时的工作特性等优势, 使得这种遥感观测手段具有非常重要的应用价值。由于 SAR 成像运算量很大, 因此实时成像是个难点, 国内外一般的处理手段是采用 DSP(Digital Signal Processor)、FPGA(Field Programmable Gate Array) 专用处理器、并行集群^[1-5], 一般基本达到实时。

随着集成工艺与集成度的提高, 图形处理器(GPU) 的处理能力也在飞速发展, 目前一台主流 GPU 的浮点处理能力可以达到上百 GFLOPS 的处理能力, 以及上百 GB/s 的数据访问带宽, 都远远超出了计算机的能力^[6], 因此为大数据量的处理提供了很好的平台。最近 NVIDIA 推出了 CUDA 以及 Fermi 的编程模型, 相对于以前的编程方式比如 OpenGL, 更能有效和方便地利用 GPU 的处理能力。目前基于 GPU 的此类编程模型已经广泛应用于很多科学计算领域。

由于 GPU 产品的成熟性以及编程模型方便、简单, 因此基于 GPU 的方式对 SAR 实时成像有很好的应用前景。而目前国内外还少有这方面的工作。本文针对 SAR 传感器的地面实时系统开展研究, 目的是利用 GPU 的处理能力实现 SAR 回波的实时成像。

2 CUDA 介绍

2.1 GPU 结构及 CUDA 编程模型介绍

GPU 是采用大量核心单指令多数据 SIMD(Single Thread Multiple Data) 的并行处理方式, 为数据并行方式。适合数据并行方式的问题划分, 具有很强的数据处理能力。图 1 是一个典型的 CPU Host 端 + GPU Device 端(基于 CUDA 模型 8 个 Stream Processor(SP) 为一个流处理单元) 的结构^[6]。

Host 主机启动一次 CUDA 的并发 kernel 即让 GPU 设备 Device 并发进行一次执行。而一般一个任务完整的执行过程为 CPU 从内存拷贝数据到显存, 启动 kernel 执行并发程序, 然后拷回数据结果。

在启动 kernel 中, 线程的组织有两个层次, 第一

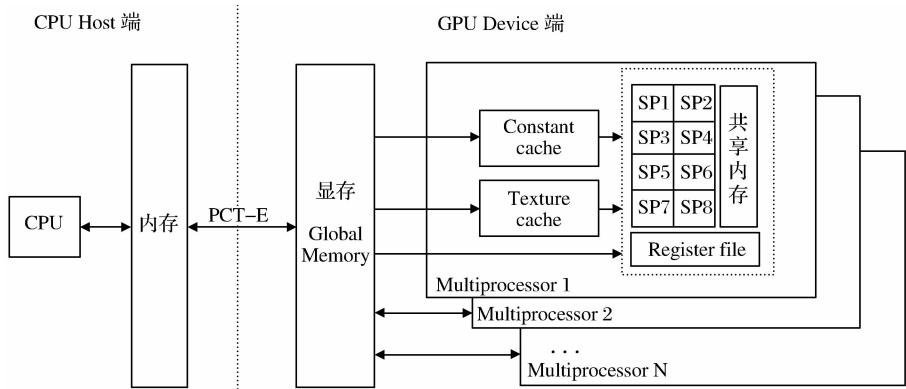


图 1 基于 CUDA 构架的典型 GPU+CPU 结构图

Fig.1 Typical CUDA based GPU+CPU platform

个是 Block 线程块,每个 Block 是一个三维的线程结构,其包含的线程量有上限,不同硬件架构有所不同,1.3 计算能力以下的每个 Block 最多可以包含 512 个线程。2.0 计算能力的可以包含 1 024 个。而再上一个层次是 Grid 网格,用于对 Block 进行组织,Grid 也是一个三维的结构,每个维度最大的规模为 65 535。

2.2 基于 CUDA 的优化设计原则

在数据运算中,一般情况下数据带宽跟不上处理器的运算速度,因此数据读写往往最为重要^[7],对存储器管理以及数据访问的优化设计最为重要。在图 1 中,按照数据带宽由大到小,依次是寄存器组、共享内存、cache、显存、PCI-E。在设计中需要考虑尽量少用带宽小的部分,多用带宽大的部分。

对显存访问时需要注意到内存融合(Memory Coalescing)。流处理器对线程的调度是以一个 Warp(32 个线程)为单位进行,而访问显存是以 16 个线程(Halfwarp)为单位进行,当访问的起始地址符合 16 线程共访问的数据量的整数倍,并且它们在同一片连续的地址空间访问数据(计算能力不同对具体要求有一定差异),则整个 16 个线程对数据的访问可以作为一个 Transaction 进行。这样能最大限度地利用显存的带宽,即内存融合^[6]。

由于 GPU 处理核心的吞吐速率一般情况下是显存的 10 倍左右,而 GPU 的处理核心非常多,一般几十到几百个(Tesla C1060 为 240 个)。为了有效利用处理核心的能力,可加大数据的处理规模,当读取数据存在延时,在并发线程足够多的情况下可运行别的线程以隐藏这个延时。因此应加大处理的数据规模。

3 RD 算法分析

传统的 RD 实现流程包括距离向 FFT、匹配滤波器相乘、距离向 IFFT、转置、方位向 FFT、RCMC

(Range Cell Migration Correction,距离徙动校正)、方位向滤波器相乘、方位向 IFFT 和转置^[8]。

这个实现流程一般是基于流水线的方式,因此只要各个步骤的实现都能满足实时要求,则系统就能满足实时的要求。而对于 GPU 方式的处理,各个步骤内部以数据并行的方式进行问题的并行划分,而各个步骤任务之间是串行进行的,因此各个步骤的处理时间之和能满足实时要求,才能使得整个数据处理系统达到实时要求。因此对于传统的 RD 结构,需要针对 GPU 平台进行重新设计,以适合总体执行时间达到最小,进而满足实时要求。

4 针对 GPU 平台的实现策略

针对 1.2 所述设计原则,对传统的结构重新进行了如下设计。

下面是按照算法流程步骤对各设计的具体阐述。

(1)去掉转置,将距离向 FFT 和方位向 FFT 合并,增大数据处理规模并避免由此带来的非内存融合的显存访问。

内存融合设计的一个必要条件是属于同一半个 Warp 的线程需要访问同一片连续空间的 Global Memory。一般情况下 RD 算法在第 3 步需要进行转置以方便进行同一个方向上的 FFT,而处理完毕需要转置以获取正常的图片。而转置运算中,如果在读入过程中同一半 Warp 的线程读入同一连续地址空间,则在输出时由于行列需要交换坐标,很难再做到对同一连续地址空间的写入操作,无法做到内存融合。而如果没有做到内存融合,则效率极为低下,耗费很多时间,使整体性能大大降低。因此去掉转置非常必要。

将方位向 FFT 与距离向 FFT 合并为一个二维 FFT,会增大数据的处理规模,使整体性能提高。

下面列出 RD 算法结构和传统算法结构的等效证明。

假设回波矩阵为 $x_0[m][n]$, m 为方位向, n 为距离向。RD 前 4 步算法可以表示为:

(a) 距离向 FFT:

$$x_{r_FFT}[m][k1] = \sum_n x_0[m][n] e^{-j\omega_{k1} n} \quad (1)$$

(b) 距离向频域滤波:

$$x_{r_filter}[m][k1] = x_{r_FFT}[m][k1] \times H_r[m][k1] \quad (2)$$

(c) 距离向 IFFT:

$$x_{r_compressed}[m][n] = \sum_k x_{r_filter}[m][k1] e^{j\omega_{k1} n} \quad (3)$$

(d) 方位向 FFT:

$$x_{a_FFT}[k2][n] = \sum_m x_{r_compressed}[m][n] e^{j\omega_{k2} m} \quad (4)$$

经过以上 4 步算法, 信号可以表示为:

$$x_{a_FFT}[k2][n] = \sum_m e^{-j\omega_{k2} m} \sum_k e^{j\omega_{k1} n} H_r[m][k1] \sum_n x_0[m][n] e^{-j\omega_{k1} n} \quad (5)$$

由于前两项积分的变量是独立的, 因此可以写成以下形式:

$$x_{a_FFT}[k][n] = \sum_k e^{j\omega_{k1} n} \sum_m e^{-j\omega_{k2} m} H_r[m][k1] \sum_n x_0[m][n] e^{-j\omega_{k1} n} \quad (6)$$

理论上讲, 雷达在同一个模式、同一个时间段内, 发射波是相同的, 由于实际实现的原因, 可能不同时刻发出的脉冲有较小差异, 但是这个差异一般很小。可以近似认为: $H_r[i][k1] = H_r[j][k1]$, i, j 是方位向不同时刻对应的脉冲号。这样每行可以进一步写为:

$$x_{a_FFT}[k2][n] = \sum_k e^{j\omega_{k1} n} H_r[i][k1] \sum_m e^{-j\omega_{k2} m} \sum_n x_0[m][n] e^{-j\omega_{k1} n} \quad (7)$$

而 $\sum_m e^{-j\omega_{k1} m} \sum_n x_0[m][n] e^{-j\omega_{k2} n}$ 即是 $x_0[m][n]$

的二维 FFT。因此可以将一般的 RD 算法的前 4 步改变为 3 步: ① 2DFFT; ② 距离向频域相乘; ③ 距离向 IFFT。

并且由于 NVIDIA 提供了高效的 FFT 库^[9] (没有列向 FFT 功能), 这部分的 FFT 可以得到高效的实现。

(2) RCMC 使用共享内存, 以减少显存访问量的设计。

每个流处理单元都有一定空间的 Shared Memory, 即共享内存, 处理器对共享内存的访问速度可以达到寄存器的访问速度, 与指令速度一致, 明显快于显存的访问速度, 因此设计时应该多注意对共享内存

的利用, 可以采用读入一片显存到共享内存, 处理完毕, 再存入显存, 这样可以做到与数据的多次交互处理只涉及与共享内存的访问, 而避免与显存的访问。

RCMC 是个插值运算, 附近几个点都会参与运算, 如果直接与显存交互, 由于显存速度相比 GPU 核心处理速度较慢会导致性能下降, 如果再加上没做到内存融合则性能会进一步下降。因此可以采用以内存融合的方式读取一段数据到共享内存, 操作完后再输出。

(3) RCMC 和方位向滤波合并, 增加数据处理的规模并减少显存访问量。

经过上一步 RCMC 使用共享内存的设计后, 中间结果存放在共享内存里面。此时可以读入对应的方位向滤波器数据, 进行相乘再输出, 这样可以加大 GPU 核心的数据处理规模。

另一方面, 对于传统的 RD 算法, RCMC 后进行方位向参考函数相乘, 这时需要将结果输出, 而在乘以方位向参考函数时, 又需要将结果读入, 这样会增加显存访问的成本。这样合并后能减小显存访问的时间成本。

(4) 采用内存融合模式的列向无转置 IFFT 的设计。

经过上面的改进后, 在最后方位向 IFFT 时需要进行列 IFFT, 而对于 FFT 库, 目前不支持列向 IFFT 操作, 因此需要自行设计。

在设计 IFFT 时应注意符合内存融合的要求, 由于数据是按行(距离向)优先的方式存储, 并且此时 IFFT 是对列(方位向)数据进行操作, 如果同一 Halfwarp 的线程共同处理一行, 这样由于数据不是存储在连续的地址空间, 不能做到内存融合。可以采用一个线程处理一行, 这样同一个 Halfwarp 的线程处理相邻的列, 而列相邻的数据是属于联系地址空间的, 这时只要保证起始地址是 256 B 的整数倍就可以做到内存融合。如图 2 所示为列向无转置 IFFT 的设计(th(i)表示第 i 个线程)。

这个 IFFT 的设计在 Tesla C1060 上进行了验证。IFFT 采用的是经典的基二 FFT 方法, 对 8192×2048 点(距离 \times 方位)复数单精度型数据进行了实验。

数据访问次数: $22; 2048$ 点需要进行 11 级蝶形运算, 读写即总共进行 22 次的访问。

共访问数据量: 2.75 GB; 每次访问数据量 $8192 \times 2048 \times 2 \times 4 \text{ B} = 128 \text{ MB}$, 22 次访问 $22 \times 128 \text{ MB} = 2.75 \text{ GB}$ 。

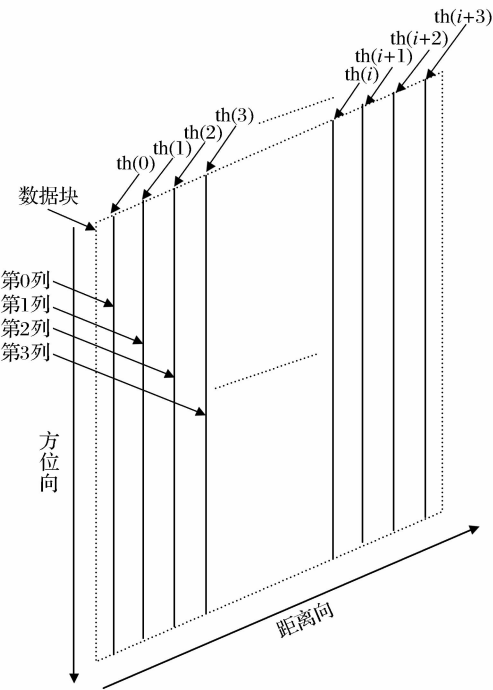


图 2 内存融合模式下的列 IFFT 的设计

Fig. 2 Column IFFT design under memory coalescing consideration

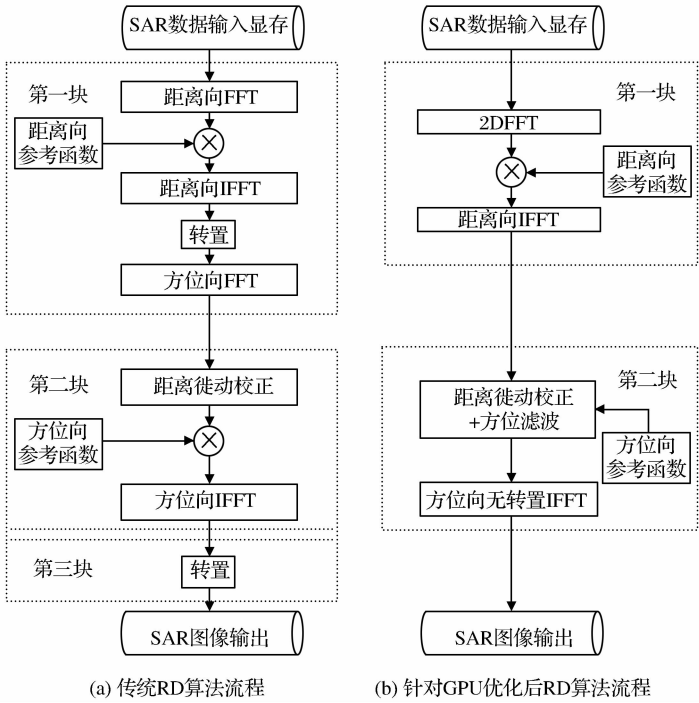


图 3 RD 算法与改进前后实现流程的对比

Fig. 3 Optimized and typical RD algorithm structure comparison

出(比如将复数型取模转换为无符号 8 位型),以减少 PCI-E 上的数据传输压力。

5 实测结果及分析

上述的改进和优化设计在 GPU 上得到了实

IFFT 实际执行时间:37 ms。
IFFT 显存速度:74 GB/s;总访问量 2.75 GB,处理运行时间 37 ms 即 74 GB/s。

显存带宽理论值:102 GB/s,实测值为 75 GB/s。
比较可以看出 IFFT 运算中显存的访问速度达到了实测值,做到了内存融合,达到了最大显存带宽。

通过上面的改进和优化,最终设计出针对 GPU 平台的 RD 核心算法结构如图 3。

图 3(a)为传统 RD 的实现流程,图 3(b)为针对 GPU 的实现流程,两个图虚线框的第一、第二块对应的是相同的功能,由于 GPU 的实现中直接采用了列向 IFFT,因此没有第三块。

此外,在实际系统设计中,由于 PCI-E 速度相对 GPU 内部资源的速度较慢,应减少内存和显存之间的数据传输,对于星载 SAR 回波数据,一般原始数据为压缩数据,比如 RadarSat-1 回波是 4 位的数据。为了减少内存显存之间的数据传输,应该在 GPU 上进行数据解压处理。而在后级对数据精度要求不高的情况下(比如快视),应将数据压缩后输

现,下面分别对优化效果以及对 RadarSat-1 卫星数据处理的实时性进行了测试。

试验测试环境:

GPU: Tesla C1060 科学计算平台;流处理器个数:30;处理核心:240;显存带宽:理论 102 GB/s;实

测 75 GB/s。

试验测试所用数据：
RadarSat-1 卫星中国西部某接收站正常接收的数据，标准 2 模式。

(1) 优化效果的评测。
采用 $8\,192\times2\,048$ 点作为一帧测评数据（每点

为复数单精度型）。
从表 1 可以看出，传统 RD 流程在第一块和第三块中占了很多时间，这两块转置花了相当大比例的时间，转置由于很难做到同一个 Halfwarp 的线程访问同一片连续地址，因此无法做到内存融合，不能有效利用显存带宽而导致运算时间很长。

表 1 优化前后性能评测数据
Table 1 Experiment data of optimized and typical performance

模块号	第一块/ms	第二块/ms			第三块/ms	总时间/ms
		RCMC	滤波	IFFT		
传统 RD 执行时间	110	5	4	7	60	186
优化后执行时间	42	5	0	37	0	85

在第二块中，可以看到优化后的 RCMC+滤波的方法有几乎一倍的效率提高，而由于优化后的无转置 IFFT 采用基二算法，会有 11 次的显存读写访问，因此时间较长。因此这里可以采用基数更大的方法减少显存访问，以提高速度，进一步提高性能。
从整体上可以看出，优化前后性能上有明显的提升，速度大约提高了一倍。

(2) 对 RadarSat-1 卫星数据处理的实时性测试。这个测试还包含了处理时间和处理速度，与计

算机的比较。
RadarSat-1(表 2 简称 R1)下行码速率:105 Mbps; 计算机平台:CPU:酷睿-i7,4 核/8 线程,3.05 GHz; 内存:3 GB。

计算机平台上 RD 算法中的 FFT 采用通用的 FFTW 库，是 PC 机上高效 FFT 的实现库^[10]。

图 4 是本测试所用数据的成像结果。从图中可以清晰地看到山脉、湖泊等显著地理目标，成像质量较好。

表 2 优化后的 RD 算法在 Tesla 执行时间以及对比
Table 2 Execution time comparison between optimized and typical RD algorithm

图像规模	$2\,048\times2\,048$	$4\,096\times2\,048$	$8\,192\times2\,048$	$8\,192\times4\,096$
PC 处理时间(1)*/ms	6 500	13 000	26 000	52 000
PC 处理时间(8)*/ms	800	1 600	3 250	6 500
Tesla 处理时间/ms	42	60	85	200
Tesla 相对 PC 加速比	19	26	38	33
Tesla 相对 R1 实时率**/ms	7(304)***	10(610)	14(1 220)	12(2 440)

* 括号内是线程数目
** A 相对 B 的实时率指 A 的处理速度除以 B 的处理速度
*** 括号内是 RadarSat-1 下行传输相应点数的数据所用的时间

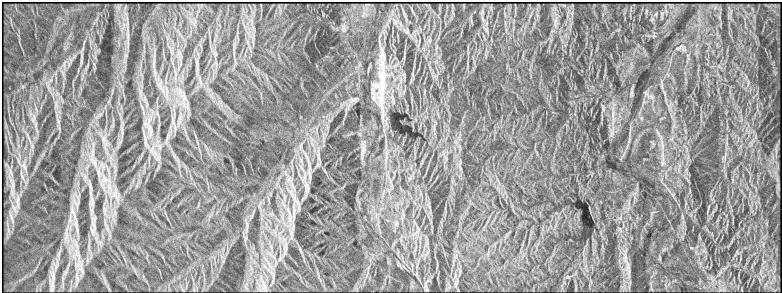


图 4 成像结果
Fig. 4 Imaging result

6 结 语

本文针对 GPU 的处理平台 CUDA 编程模型,对传统的 RD 核心结构进行了重新设计,使其在 GPU 的处理结构中对不同带宽的利用、对显存的访问,以及运算规模的增大适合 GPU 的硬件特性,最终提高了整体性能,缩短了执行时间。相比传统结构在 GPU 处理结构上的直接实现性能提高了一倍。而与一台主流 PC 相比,加速比为 20 倍以上,实时率是 RadarSat-1 的 10 倍以上。对单机系统实现实时成像提供了很大的余地。

在此基础上,可以考虑针对 GPU 平台对现有的 SAR 实时成像算法进一步改进,比如在整个核心算法的实现中,方位向无转置的列 IFFT 由于采用基二算法,导致显存数据访问量较大,可以考虑采用更大基数以及分裂基的方式实现,减少蝶形算子迭代级数,进一步缩短执行时间,以及对新的适合 GPU 平台的实时成像算法进行研究。

随着 SAR 性能的提升,数据速率的提高,必然对实时处理系统的处理能力也有更高的要求,对于性能更强的处理系统,可以考虑结合集群的方式,采用多个节点以实现多个 GPU 的并行处理能力。

参考文献(References):

[1] Chen Liang, Long Teng. Spaceborne SAR Real-time Quick-look System[J]. Transactions of Beijing Institute of Technology, 2008,

28(6):545-548. [陈亮,龙腾. 星载合成孔径雷达实时快视成像系统[J]. 北京理工大学学报, 2008, 28(6):545-548.]

[2] Tang Y S, Zhang C Y. Multi-DSPs SAR Real-time Signal Processing System based on cPCI Bus[C]//Proceedings of 2007 Asia-Pacific Conference on Synthetic Aperture Radar, Huangshan, China, 2007:661-663.

[3] Xiong Junjun, Wang Zhensong, Yao Jianping. The FPGA Design of on Board SAR Real Time Imaging Processor[J]. Chinese Journal of Electronics, 2005, 33(6):1700-1701, 2701. [熊君君,王帧松,姚建平. 星载 SAR 实时成像处理器的 FPGA 实现[J]. 电子学报, 2005, 33(6):1700-1701, 2701.]

[4] Guo Kunyi, Sheng Xinqing. Implementation of SAR Parallel Image Processing based on High Performance Beowulf Group [J]. Modern Radar, 2005, 27(1):38-40, 61. [郭琨毅,盛新庆. 高性能 Beowulf 集群的 SAR 并行成像的处理[J]. 现代雷达, 2005, 27(1):38-40, 61.]

[5] Linda M, Michel D, Bernd H, *et al.* Real-time Optical Processor Prototype for Remote SAR Applications[C]//Proceedings of 2009 SPIE, Berlin, Germany, 2009.

[6] NVIDIA. NVIDIA CUDA Programming Guide Version 3.0 [Z]. 2010:2-3.

[7] Wang J H, Hu S Q, Long T, *et al.* The Evaluation between Power PC and DSP in SAR Imaging Algorithm[C]//IET International Radar Conference, Huangshan, China, 2009.

[8] Cumming I G, Wong F H. Digital Processing of Synthetic Aperture Radar: Algorithms and Implementation[M]. Beijing: Publishing House of Electronics Industry, 2007:154-156.

[9] NVIDIA. NVIDIA CUDA CUFFT Library[Z]. 2010:2-15.

[10] Frigo M, Johnson S G. FFTW (Version 3.2.2, 12 July 2009) [Z]. Massachusetts Institute of Technology, 2009:1-70.

Optimization and Implementation of RD Algorithm in SAR Ground Quick Look System According to GPU Platform

Yang Renzhong, Zhang Tao, Lin Botao, Wei Hongwei

(Center for Earth Observation and Digital Earth, Chinese Academy of Sciences, Beijing 100094, China)

Abstract: Synthetic Aperture Radar (SAR) imaging requires huge computation load, thus it needs high-end computation platform to construct the system when comes to real time imaging process. Typical SAR real time imaging systems adopt DSP, FPGA and cluster. This paper presents a new approach that realizes SAR real time imaging processing based on Computer Unified Device Architecture (CUDA), which is a new parallel programming model introduced recently on Graphic Processing Unit (GPU). The study had analyzed the difference between data parallel mode of CUDA and task parallel mode of typical platform, including studied RD algorithm structure and optimized each step according to CUDA parallel programming model, then the corresponding RD algorithm structure is implemented. This paper also conducted the experiment results showed the speed on this platform is about 20~30 times of a common high-end PC(Intel Core-i7 950), and 10 times of the speed required for real time imaging for RadarSat-1. It proves a promising way to realize SAR real time imaging system.

Key words: Synthetic Aperture Radar; Real time imaging; Parallel computing; High performance computing