基于 NVIDIA GPU 的机载 SAR 实时成像处理算法 CUDA 设计与实现

孟大地* 胡玉新 石 涛 孙 蕊 李晓波
(中国科学院电子学研究所 北京 100190)
(中国科学院空间信息处理与应用系统技术重点实验室 北京 100190)

摘 要：合成孔径雷达(SAR)成像处理的运算量较大，在基于中央处理器(Central Processing Unit, CPU)的工作站或服务器上一般需要耗费较长的时间，无法满足实时性要求。借助于通用并行计算架构(CUDA)编程架构，该文提出一种基于图形处理器(GPU)的 SAR 成像处理算法实现方案。该方案解决了 GPU 显存不足以容纳一景 SAR 数据时数据处理环节与内存/显存间数据传输环节的并行化问题，并能够支持多 GPU 设备的并行处理，充分利用了 GPU 设备的计算资源。在 NVIDIA K20C 和 INTEL E5645 上的测试表明，与传统基于 GPU 的 SAR 成像处理算法相比，该方案能够达到数十倍的速度提升，显著降低了处理设备的功耗，提高了处理设备的便携性，能够达到每秒约 36 兆采样点的实时处理速度。

关键词：SAR；实时成像；图形处理器(GPU)；通用并行计算架构(CUDA)
中图分类号：TN957.52 **文献标识码**：A **文章编号**：2095-283X(2013)04-0481-11
DOI: 10.3724/SP.J.1300.2013.13056

Airborne SAR Real-time Imaging Algorithm Design and Implementation with CUDA on NVIDIA GPU

Meng Da-di Hu Yu-xin Shi Tao Sun Rui Li Xiao-bo
(*Institute of Electronics*, *Chinese Academy of Sciences*, *Beijing* 100190, *China*)
(*Key Laboratory of Technology in Geospatial Information Processing and Application System*, *Chinese Academy of Sciences*, *Beijing* 100190, *China*)

Abstract: Synthetic Aperture Radar (SAR) image processing requires a considerable amount of computational resources. Traditionally, this task runs on a workstation or a server based on Central Processing Units (CPUs) and is rather time-consuming, making real-time processing of SAR data impossible. Based on Compute Unified Device Architecture (CUDA) technology, a new plan for a SAR imaging algorithm operated on an NVIDIA Graphic Processing Unit (GPU) is proposed. The new proposal makes it possible for the data processing procedure and the CPU/GPU data exchange to execute concurrently, especially when the size of SAR data exceeds the total GPU global memory size. A multi-GPU is suitably supported by the new proposal, and all computational resources are fully exploited. It has been shown by an experiment on an NVIDIA K20C and INTEL E5645 that the proposed solution accelerates SAR data processing by tens of times. Consequently, a GPU based SAR processing system that embeds the proposed solution is much more efficient and portable, thereby making it qualified to be a real-time SAR data processing system. Experiments showed that SAR data can be processed in real-time at a rate of 36 megapixels per second by a K20C when the new solution is implemented.

Key words: SAR; Real-time processing; Graphic Processing Unit (GPU); Compute Unified Device Architecture (CUDA)

1 引言

合成孔径雷达(Synthetic Aperture Radar, SAR)是一种具有全天时、全天候、获取微波散射信息丰富等特点的重要的遥感技术手段，在军事、农业、林业、海洋等领域均具有巨大的应用潜力[1]。但由于高分辨率 SAR 系统接收的原始数据量大，并且需要经过复杂的 2 维匹配滤波处理才能获得 SAR 图像，甚至有些应用领域对 SAR 图像获取具有较高的实时性要求，从而使得高速的成像处理技术成为 SAR 领域的一项关键技术。目前一般利用基于中央处理

2013-07-12 收到，2013-11-07 改回；2013-11-18 网络优先出版
国家大科学工程航空遥感系统地面数据综合处理与管理分系统项目资助课题
*通信作者： 孟大地 mengdadi@hotmail.com

器(Central Processing Unit, CPU)的个人计算机、工作站以及大型计算服务器进行后期的 SAR 成像处理，这时需要投入大量资金购置多台具有足够处理能力的计算设备，并利用 OpenMP[2]或 MPI[3]等并行编程技术进行软件实现。而 SAR 实时成像处理器一般用 FPGA 或 DSP 实现，这时均需要较为复杂的编程手段以及采购昂贵的硬件设备。

由于图形处理市场的拉动作用，图形处理器(Graphic Processing Unit, GPU)已经发展成为具有高度并行、多线程、多核心、超大带宽、具有数百个计算单元的高性能处理平台。与通用的 CPU 相比，由于 GPU 产品不具有 CPU 产品中的流程控制、缓存等功能部件而专注于进行数据运算，因此 GPU 产品更适用于作为以浮点运算为主、适于并行化的计算任务的运行平台[4]。近年英伟达(NVIDIA)公司 GPU 产品与英特尔(Intel)公司 CPU 产品的计算性能对比如图 1 所示[4]，可见各时期 GPU 产品的运算速度以及传输带宽远高于同时代的 CPU 产品。NVIDIA 于 2006 年 11 月推出了一种将 GPU 用于进行高性能计算的通用并行计算架构(Compute Unified Device Architecture, CUDA)以及相关的并行编程模型和函数库。该技术的出现大大推动了 GPU 在高性能计算领域的广泛应用[4]。借助于 CUDA 技术，在传统 CPU 上运行的需要大量浮点运算的代码可以方便地移植到 GPU 上，而代码的条件判断等逻辑控制部分仍然在 CPU 平台上运行[4]。

机载 SAR 成像处理过程一般由向量相乘、转置、快速傅里叶变换(FFT)以及插值等运算模块组成[1]，各运算模块都能以较高的效率进行并行化处理，因此有望利用 CUDA 技术在 GPU 上实现成像处理算法，并取得与 CPU 上的成像处理算法相比较大幅度的速度提升。本文拟利用 CUDA 技术将传统的机载 SAR 成像处理算法部署于 NVIDIA 的 GPU 产品上，并用于进行 SAR 实时成像处理。

当前支持 CUDA 技术的主流 GPU 产品的显存较小，一般不超过 6 GB，不足以容纳一景 SAR 数据量，且具有较大显存的高端产品价格较为昂贵。由于成像处理过程中需要多次进行转置操作，因此需要在主机内存中开辟足够空间用于存储全部处理数据，并在成像处理的各个阶段将数据进行分块，各块依次传入 GPU 设备显存，处理完成之后将处理结果传入主机内存。设备显存与主机内存通过 PCI-e 接口进行数据交互，该接口的单向传输速度约为 6 GB/s(2 代 PCI-e×16, CUDA SDK 提供 bandwidthtest 软件测试结果)，因此数据在主机内存与 GPU 显存之间的多次交互导致 GPU 的处理性能不能得到充分发挥，处理效率大大降低。

本文利用 CUDA 编程接口中的异步执行(asynchronous concurrent execution)[4]，流(stream)等技术[4]，以及对 SAR 数据的分块处理策略，提出了一种运行于 NVIDIA GPU 产品上的成像处理算法的设计方案。该方案在不降低算法处理精度的基础上，掩盖了数据在主机内存与设备显存之间的交互传输，从而能够充分利用 GPU 设备的计算资源；当主机上配置多个 GPU 设备时，该方案还支持多 GPU 设备的同时执行，提高了方案的硬件配置灵活性；该方案中的所有大规模计算任务都由 GPU 完成，CPU 只进行足以忽略不计的简单计算、参数读取以及逻辑判断，因此主机只需配置足够的内存，而无须配置昂贵的高性能 CPU，甚至单核 CPU 产品也足以胜任，降低了设备预算及设备功耗；该方案突破了 GPU 显存容量对 SAR 成像处理数据大小的限制，只需足够的主机内存，而不要求单个设备具有大容量显存，甚至不足 1 G 的显存容量也能满足处理需求，从而提高了实时成像处理算法对大数

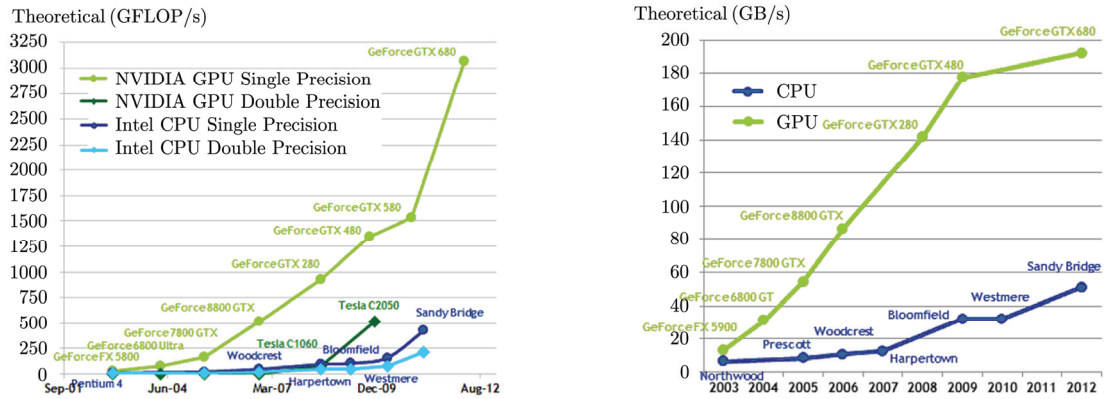


图1 近年GPU/CPU产品的性能对比

Fig. 1 Comparison of GPU/CPU products in recent years

据量 SAR 数据的处理能力，拓宽了购置 GPU 设备时的选择范围，大大降低了对 GPU 设备内存容量的要求，从而降低了设备购置的资金预算；另外，该方案不但能用于 SAR 实时成像处理，也能用于事后 SAR 数据处理，这时甚至可以将该方案部署在笔记本电脑上，大大增加了成像处理设备的便携性。

将机载 ωκ 成像处理算法在一部 NVIDIA K20C(计算能力 3.5, 2496 个 CUDA 计算核心(core)，功耗 225 W，市场价格约为 25000 元人民币，用 CUDA 5.0 编译)[5]以及一颗 INTEL Xeon E5645 (Westmere 系列，主频 2.4 GHz, 6 核心，功耗 80 W，市场价格约为 2900 元人民币，用 INTEL 编译器编译，调用 INTEL MKL 数学运算库)[6]上的对比测试表明，该方案可以达到约每秒处理 36 M 个原始数据采样点的处理速度。与基于 CPU 的处理算法相比，该方案可以达到约 15 倍的加速比，基本与图 1 的性能对比相符。对方位/距离向各为 32768 的 SAR 数据，一部 K20C 处理所需时间约为 7 s(不包括原始数据读取、数据写入、内存/显存的分配和释放)，足以保证成像处理的实时率。

本文第 2 节根据 CUDA 并行化特点对 SAR 成像处理算法的结构特征进行分析；第 3 节简要介绍了 SAR 实时成像处理算法实现所需 CUDA 编程技巧；基于以上两节的讨论，第 4 节详细讨论了基于 CUDA 的 SAR 实时成像处理算法设计思路以及最终实现方案；第 5 节利用仿真 SAR 数据验证了算法的处理速度，并与基于 CPU 的处理结果进行对比；第 6 节对本文内容进行了总结。

2　SAR 成像处理算法结构分析

目前一般采用频域 2 维匹配滤波的成像处理算法[1]进行 SAR 成像处理，而不采用运算量多出数百倍甚至千倍的时域后向反投影算法[7]，对于实时成像处理算法更是如此。频域匹配滤波算法主要包括距离多普勒(RD)算法、Chirp Scaling(CS)算法以及 ωκ 算法。3 种算法的详细介绍及优缺点对比参见文献[1]。3 种算法的处理步骤见图 2 所示流程图 [1]。图 2 中圆角矩形表示距离向处理，所处理数据按先排距离向存储；直角矩形表示方位向处理，所处理数据按先排方位向存储。

由图 2 可见，3 种频域机载 SAR 成像处理算法具有大致相同的算法结构，在研究其在 GPU 上的部署时，可以统一进行分析与设计。因此下文内容适用于 3 种算法中的任何一种。

机载 SAR 在实际数据获取过程中，载机受气流影响会偏离理想水平匀速直线运动状态，导致原始回波数据与成像处理算法失配，因此在对实际机载 SAR 数据进行处理时，还需进行运动补偿[8]。本文中采用文献[9]中介绍的具有更高精度的运动补偿算法(考虑了距离空变的距离走动补偿)，在图 2 中的步骤 A 距离压缩之后进行运动补偿处理即可。

在图 2 中，由于 SAR 原始数据以及处理中间结果数据在计算机内存中的线性存储特点，3 种算法都需要在处理过程中进行 3 次转置操作(改变 2 维数据的优先存储方向，以利于处理器对内存中数据的读取/写入效率)。SAR 原始数据最初是处于先排距离向的存储状态。3 种算法在不同的数据存储方式下(先排距离向或先排方位向)对数据执行所需操作(图 2 中表示为 A, B, C, D 4 个运算模块)，最终获得先排距离向的 SAR 图像。

由于上述 3 种算法在执行过程中都需要进行多次转置，要将其利用 CUDA 技术在 GPU 上部署，首选方案是将所有数据一次性读入 GPU 显存，并在 GPU 上用 kernel(GPU 处理任务的调度单元[4])的方式实现算法的所有运算模块。目前已有基于 CUDA 技术的 SAR 成像处理算法均采用了这种方法[10−13]，因此可处理一景数据所包含脉冲数受到 GPU 显存容量的限制。而当 GPU 显存容量(不超过 6 GB，未来预计会有更高显存容量的 GPU 产品问世，但价格总是与内存容量成正比)不能容纳一景 SAR 数据时，这时只能先将所有数据存入主机内存，在主机端利用 CPU 完成 4 次转置操作。在不同的数据存储方式下，每个运算模块由以下方式实现：根据显存容量对所需处理数据进行分块(每个分块的数据连续存储，并确保 GPU 显存能容纳每块数据，分块处理并不影响算法的处理精度)；对于每个数据块，先将其拷贝至 GPU 显存，在 GPU 中用 kernel 的方式实现模块相关处理步骤，再将结果写入主机内存的原位置；最后从主机内存中将处理结果 SAR 图像存盘。

在上述实现方式中，GPU 需要等待主机端的转置操作完成才能执行之后的运算模块，由此带来两点不便：CPU 转置操作与 GPU 运算不能同时执行，降低了算法的整体处理效率；为了提高转置效率，需要配置运算速度较快的 CPU，增加设备预算及设备功耗。

由此可见，为了充分利用 GPU 设备的运算性能，降低对 CPU 的性能要求，在将数据存储于主机内存(始终以先排距离向的存储方式)以及采用分块处理策略的前提下，需要将转置操作以及 A, B, C, D 4 个运算模块在 GPU 上执行，而 CPU 只承担流程控制、数据读取/写入以及少量简单运算。另外，还

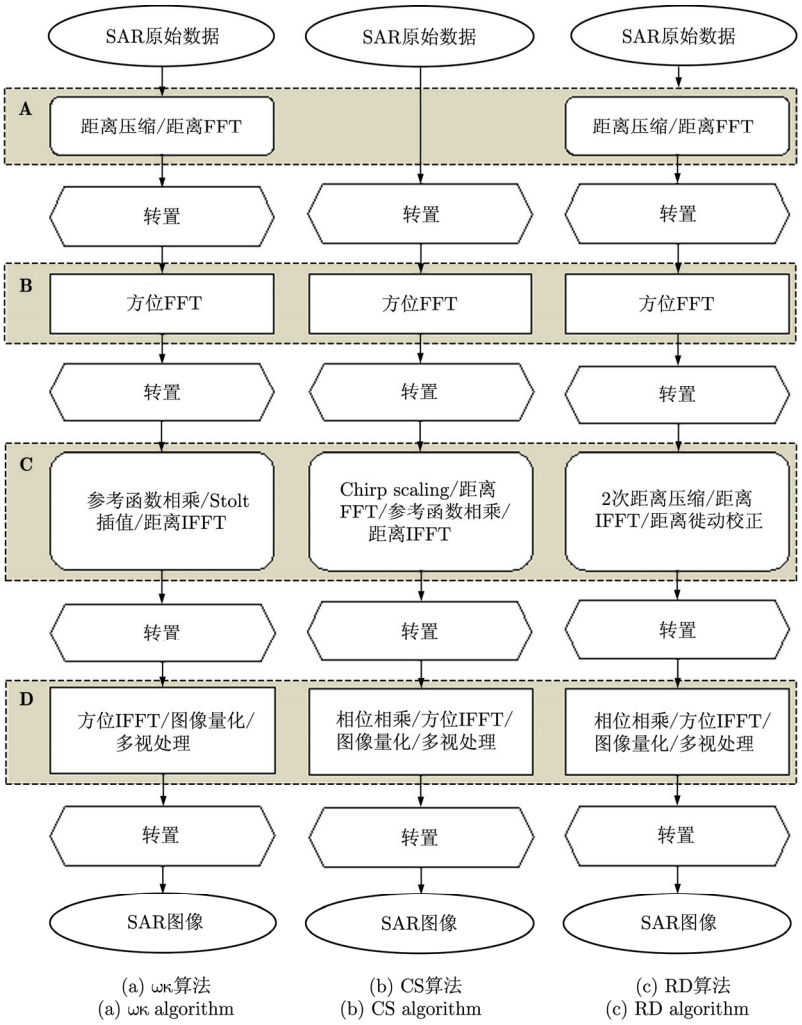


图2 常用频域SAR成像处理的算法流程图

Fig. 2 Flow charts of 3 standard SAR image formation algorithms

需要利用 CUDA 编程接口中的异步并行技术，使得主机内存与 GPU 显存之间的数据交互与 kernel 并行执行，从而减少甚至消除 kernel 执行对数据交互的等待时间。

3 相关 CUDA 编程技巧

CUDA 编程技术的基本任务划分原则是：由主机负责算法的整体架构，而将算法中的大运算量模块交由 GPU 处理。这些大运算量模块一般由连续执行的若干步骤组成，其中某些步骤可直接调用现有的 CUDA 函数库(如 FFT 可由 CUFFT 中的库函数实现)，其余无库函数可供调用的步骤需编写对应的 kernel。

根据上节对 SAR 成像处理算法的结构特征分析，对于图 2 中 A, B, C, D 4 个运算模块，均需要执行 CPU→GPU 数据复制、数据处理、GPU→CPU 数据复制 3 个步骤；对于模块 B 与模块 D，还需要在数据处理之前进行先排距离向到先排方位向的转置操作，以及在数据处理之后进行先排方位向到先排距离向的转置操作。两次转置操作均由 GPU 完成。要利用 CUDA 技术将这种实现方案在 GPU 上部署，并充分利用 GPU 设备的运算资源以及传输带宽，需要利用 CUDA 编程技术所提供的各种接口解决以下几点技术难题。

3.1 内存分段拷贝与转置处理

根据第 2 节对 SAR 成像处理算法的结构分析，需要对主机内存中的 SAR 数据分块，对每块执行“读入 GPU 显存→数据处理→写入主机内存”3 步操作。由于数据始终以先排距离向的方式存储于主机内存，因此在进行距离向操作(图 2 中以圆角表示的模块，模块 A, C)时，所需处理的数据块连续存储于主机内存的一段空间(如图 3(a)所示)，这时可直接利用 CUDA 库函数 cudaMemcpy[4]或 cudaMemcpyAsync[4]将其由主机内存复制到 GPU

显存，处理结束后再由该函数将处理结果从 GPU 显存复制到主机内存的原位置。

而在进行方位向操作(图 2 中以直角表示的模块，模块 B, D)时，所需处理的一块数据在主机内存中分散于 N_a 个连续存储数据段(如图 3(b)所示，其中 N_r 为距离向采样点数，N_a 为脉冲个数)，这时就需要调用 CUDA 编程接口所提供的支持分段复制的内存复制函数 cudaMemcpy2D[4] 或 cudaMemcpy2DAsync[4]进行模块前后的两次数据块复制。在将数据由主机内存传入 GPU 显存后，数据按照先排距离向方式存储于 GPU 显存上一块连续的存储空间，这时还需要将该数据做转置处理(数据将按先排方位向方式存储)，才能进行后续的数据处理操作。在数据处理完成之后，还需再进行一次转置处理(数据将按先排距离向存储)，才能再次调用 cudaMemcpy2D 或 cudaMemcpy2DAsync 将处理结果写入主机内存原位置。

CUDA Samples 中提供了利用共享内存技术在 GPU 上实现转置操作的示例[14]，本文转置 kernel 的设计直接采用了该方法。另外，为了让转置执行与内存显存之间的数据复制实现下节所述的异步并行化，在实现转置 kernel 时，借助于 cudaMemcpy2D 或 cudaMemcpy2DAsync，转置前后数据均位于 GPU 显存上，而未采用内存映射(mapped memory)技术[4]或统一虚拟地址空间(unified virtual address space)技术[4]所实现的从主机内存读取数据直接进行转置。

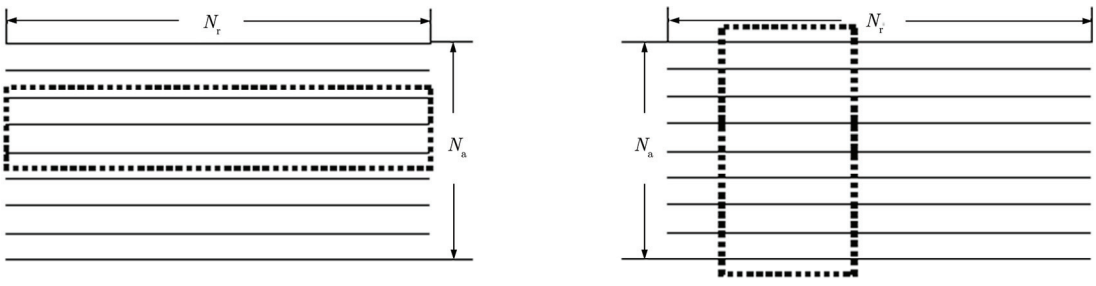
3.2 并行处理相关技术

3.2.1 异步并行技术　在计算能力不低于 1.1 的部分 GPU 设备上，CUDA 编程架构提供了主机内存与 GPU 显存之间数据传输与 kernel 执行的并行机制，称为异步并行技术。合理地利用该技术，使得数据复制与 kernel 执行同步进行，可以尽可能地保证 GPU 设备的运算核心处于忙碌状态，减少甚至消除第 2 节所述 kernel 对所需数据的等待时间。在计算能力不低于 2.0 的部分 GPU 设备上，CUDA 编程架构甚至支持“主机内存→GPU 显存数据传输、kernel 执行、GPU 显存→主机内存数据传输”3 者的并行机制[4]。

但是，这种并行机制也受到所复制数据与 kernel 执行所需数据的依赖性的限制，即在 kernel 执行所需数据复制到显存完成之前 kernel 一直处于等待状态，在 kernel 执行完成之前无法将执行结果复制到主机内存，因此该技术不能直接应用于 SAR 成像处理算法。

3.2.2 流技术　异步并行技术的实施需要借助于流(stream)技术。创建若干个流，将所需处理任务分配到各个流上。在各流的当前任务满足上节所述的并行执行条件时，由于流的可并行执行特性[4]，从而实现了当前各任务的并行执行。图 4 示意了这种异步并行方案的实现方式。其中 CPU 代表主机内存，GPU 代表 GPU 显存，各圆角矩形中的数字代表分块序号。将图 2 中的每个运算模块部署在 GPU 上时，可按照如图 4 所示策略进行任务调度(对于运算模块 B, D，在数据处理前后还包含两次转置操作)。

如图 4 所示，创建 3 个流，先将前 3 个 SAR 数据块分别交由流 0，流 1，流 2 处理，每块数据的处理都包括“CPU→GPU 数据传输、数据处理、GPU→CPU 数据传输”3 项任务(该示例中假设各处理模块的执行时间相等)。由于 3 个流的同类处理任务不能并行执行，因此相邻流之间有一个任务执行时间的延时，而 3 个流的不同类处理任务可以并行执行。再继续将后续待处理数据块依次发射(issue)在 3 个流上，即可保证所有时刻 3 个流中的处理步骤都处于并行执行状态。这时除了整个任务的首尾少量时间 3 个流未完全并行外，处理过程中绝大部分处理时间 3 个流都处于并行执行状态，从而保证了 GPU 运算核心(core)绝大部分时间处于忙碌状态，有效掩盖了数据在主机内存与 GPU 显存中的传输时间。



(a) 距离向处理时
(a) For range processing

(b) 方位向处理时
(b) For azimuth processing

图3 SAR数据分段复制时数据块的连续性示意图

Fig. 3　Data storage form considering data copy

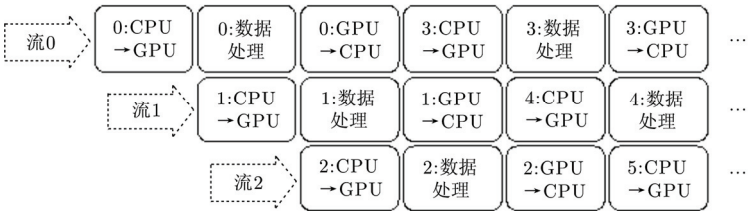


图4 对SAR数据分块并利用GPU异步并行处理流程图

Fig. 4 Flow chart of SAR data partitioning and asynchronous processing on GPU

为了降低 3 个流首尾未并行任务块占总任务的比例，需要保证 SAR 数据的分块数不能太少，而各分块尺寸太小时不能充分利用所有 CUDA 运算核心。实际中可通过多次试验选取效率较高的分块大小和分块总数组合。

3.2.3 事件同步技术 如前所述，图 4 中的数据处理环节由若干个 kernel 组成，而且 kernel 执行时间与单向数据传输时间一般情况下不完全相同，根据 GPU 对各流所发射 kernel 的调度机制，在运行过程中各流的 kernel 可能会交替执行[4]，从而导致图 4 中的某些数据处理环节(由多个 kernel 组成)开始到结束所需时间增加。为了避免此现象，确保每个数据处理环节中的所有 kernel 集中连续执行，可以利用 CUDA 中的事件(event)技术[4]。在每个数据处理环节的末尾(GPU→CPU 数据传输环节开始前)利用 cudaEventRecord 函数标记事件同步节点，并在下一个数据处理环节的起始(CPU→GPU 数据传输环节结束后)时利用 cudaEventSynchronize 函数对该事件标记进行同步。根据事件的同步特点，在 cudaEventRecord 函数调用之前的所有 CUDA 操作完成之前，cudaEventSynchronize 之后的 CUDA 操作不能进行，从而确保了每个数据处理环节中各 kernel 的集中连续执行。详见下述实验结果。

3.2.4 多 GPU 并行处理技术 当一台主机内有多部 GPU 设备时，利用流技术还可以实现所有设备之间的并行处理。在每个 GPU 设备上各部署 3 个流，并将所有 SAR 数据处理任务依次发射到各个流上，从而在每个 GPU 设备内部异步并行的同时，也实现了多 GPU 设备的并行处理。

对于第 i 个数据块，所对应的 GPU 设备号 i_{GPU} 以及流号 i_{stream} 的计算方法分别为(三者均以 0 为起始编号)：

$$i_{\mathrm{GPU}} = i\%N_{\mathrm{GPU}},\quad i_{\mathrm{stream}} = \left(\frac{i}{N_{\mathrm{GPU}}}\right)\%3 \tag{1}$$

其中 N_{GPU} 为 GPU 设备个数，%表示求余操作。

3.3 显存分配策略

由上节可知，为了充分利用 GPU 设备的异步并行技术掩盖数据传输时间，在每个 GPU 设备中需各分配 3 个流。由于各流之间除并行关系外相互独立运行，因此需要为每个流分配一个独立的存储空间用于存储从主机内存加载的数据块。另外，各个运算模块中的某些运算步骤需要用另外一块同样大小的存储空间存储运算处理结果，这些运算步骤包括转置处理，ωκ算法中的 stolt 插值处理[1]，运动补偿处理[4]等。因此，需要对每个流分配两个同样大小的存储空间。实践表明，对于机载 SAR 实时处理算法来说，两块存储空间已能够满足算法的运算需要。

由此可见，每个 GPU 设备上共需分配 6 块同样大小的存储空间。另外利用 CUFFT 库函数进行 FFT 运算时，需要为每个流根据数据尺寸调用 cufftPlan1d()创建一个 plan 配置数据。以上 9 项存储空间之和不能超过一个 GPU 设备的显存总容量。

4 设计与实现

4.1 存储空间分配与 FFT plan

在成像处理开始之前，需要先进行内存及显存分配，在所有步骤执行完毕之后，再将所分配内存及显存释放。

首先在主机内存中分配两段存储空间 H_0 与 H_1，H_0 用于存储读入的原始数据以及处理结果 SAR 图像(一般 1 个采样点占用 1 个或 2 个字节)，H_1 用于存储算法各阶段的中间处理结果(一般按单精度浮点存储，1 个采样点占用 8 个字节)。假设所需处理的一景 SAR 数据每个回波脉冲采样点数为 N_{r}，共 N_{a} 个脉冲，则 H_0 与 H_1 占用字节数分别为：

$$M_{H_0} = 2N_{\mathrm{a}}N_{\mathrm{r}},\quad M_{H_1} = 8N_{\mathrm{a}}N_{\mathrm{r}} \tag{2}$$

为了提高主机内存与 GPU 显存之间的数据传输速度，H_0 与 H_1 按照页锁定(page-locked)的方式进行空间分配[4]。

根据第 3 节所述，在每部 GPU 设备上各分配 3 个流 S_{ij}(其中 i, j 分别表示以 0 为起始的 GPU 编号以及流编号)，并为每个流在所在 GPU 设备显存上分配两块字节数分别为 M_{d}(按照上节所述进行设置)的存储空间 $D_{\mathrm{A},ij}$ 与 $D_{\mathrm{B},ij}$，用于完成图 2 中每个模块的各个计算步骤。根据显存块大小 M_{d}，可以得到用

于每个运算模块的数据分块大小。对于先排距离向的运算模块 A, C，每个数据块的方位脉冲数为：

$$B_{\mathrm{a}}=\left\lfloor\frac{M_{\mathrm{d}}}{8N_{\mathrm{r}}}\right\rfloor \tag{3}$$

其中⌊⌋表示下取整。对于先排方位向的运算模块 B, D，每个数据块的距离点数为：

$$B_{\mathrm{r}}=\left\lfloor\frac{M_{\mathrm{d}}}{8N_{\mathrm{a}}}\right\rfloor \tag{4}$$

为了方便起见，如有必要，B_{a}与 B_{r}可在以上计算结果基础上适当减小，使其分别能整除 N_{a}与 N_{r}。对于运算模块 A, C，处理数据的分块数为 $C_{\mathrm{a}}=N_{\mathrm{a}}/B_{\mathrm{a}}$；对于运算模块 B, D，处理数据的分块数为 $C_{\mathrm{r}}=N_{\mathrm{r}}/B_{\mathrm{r}}$。

另外，由于各运算模块的执行步骤中都需要用到 FFT 操作，对于每个运算模块，还需要为每个流 S_{ij}根据数据块 2 维尺寸利用 CUDA 函数 cufftPlan1d 建立 FFT 执行所需 plan，并将所创建 plan 用 cufftSetStream 函数与所属流关联[4]。由于 plan 的创建及销毁占用时间非常小，对于每个运算模块，可以在模块执行前建立并关联所需 plan，模块执行完毕后立即调用 cufftDestroy 函数销毁，或者在算法开始运行时创建所需 plan，算法结束时销毁所需 plan。

4.2 算法设计

根据以上对 CUDA 编程技术以及 SAR 成像处理算法特点的分析，在上述存储空间分配方案基础上，可按照以下方案进行机载 SAR 实时成像处理算法设计：

(1) 将所需处理 SAR 原始数据由文件读入主机内存 H_0。

(2) 将 H_0中的 SAR 原始数据沿方位向均分为 C_{a}块，各块字节数均为 $2N_{\mathrm{r}}B_{\mathrm{a}}$。各块按照图 4 所示方案依次分配给各个流 S_{ij}，各个流 S_{ij}调用 cudaMemcpyAsync 函数将所分配数据块读入显存空间 $D_{\mathrm{A},ij}$，再进行运算模块 A 的各处理步骤，最后将结果由 cudaMemcpyAsync 函数写回主机内存 H_1的对应位置，每块写回数据字节数为 $8N_{\mathrm{r}}B_{\mathrm{a}}$。此时数据在 H_1中按照先排距离向的方式存储。

(3) 将 H_1中的数据沿距离向均分为 C_{r}块，各块按照图 4 所示方案依次分配给各个流 S_{ij}，每个流将所分配数据块(非连续存储，如图 3(b)所示)先由 cudaMemcpy2DAsync 函数读入显存空间 $D_{\mathrm{B},ij}$，再由转置 kernel 将 $D_{\mathrm{B},ij}$中的数据转为先排方位向方式存储于 $D_{\mathrm{A},ij}$，然后进行运算模块 B 的各处理步骤；处理完成之后再次调用转置 kernel 将处理结果(可能在 $D_{\mathrm{A},ij}$或 $D_{\mathrm{B},ij}$中，视具体实现方式而定)转为先排距离向方式存储于该流的另一块显存空间，将结果由 cudaMemcpyAsync 函数写回主机内存 H_1的原位置。此时数据在 H_1中仍然按照先排距离向的方式存储。

(4) 按照与步骤(2)类似的方式执行运算模块 C，但不是从 H_0中读取数据，而是从 H_1中读取数据。

(5) 按照与步骤(3)类似的方式执行运算模块 D，但有以下几点区别：(a) 在对处理结果进行转置时，由于多视后每个图像的像素点按照 1 个字节或 2 个字节无符号存储，因此需要实现一个新的转置 kernel，但也可将转置 kernel 利用 C++的模板(template)技术实现，从而对中间结果的转置处理以及对多视结果的转置处理可复用一个函数代码；(b) 由于每块数据处理前后数据量变化，而且数据在主机内存中非连续存储，因此不能再将处理结果写回 H_1的原位置，但可以将处理结果写入 H_0。各块图像数据在 H_0中的组织及写回方式与 3.1 节所述及图 3(b)所示方式相同。

(6) 将 H_0中的处理结果图像写入图像文件。

在以上各步骤中，除了第(1)步数据读取以及第(6)步数据落盘(这两步受磁盘条件限制无法通过并行手段提高执行速度)以外，在步骤(2)–步骤(5)中，每步对所有数据块的处理都按照 3.2 节所述方式实现了并行执行，从而充分利用了 GPU 设备的传输带宽以及计算资源，保证在最短时间内完成所有的处理步骤。

4.3 批量任务处理策略

将上述方案用于实时成像处理时，需要对各景 SAR 数据依次进行处理。为满足实时性要求，对各景的处理之间不能有长时间等待。在上述方案中，当一景数据量较大时，对 H_0, H_1, $D_{\mathrm{A},ij}$以及 $D_{\mathrm{B},ij}$的分配及释放都需要占用较多的时间，若对每景数据都进行一次存储空间的分配及释放，将大大降低处理的实时性。

为了避免存储空间的分配和释放对处理实时性的影响，在各景数据 2 维尺寸不变的情况下(一般至少在一个条带内能够保证各景的数据 2 维尺寸不变)，可以通过图 5 所示的策略避免主机内存/GPU 显存的重复分配与释放。对于单景数据距离/方位点数未发生变化的一组处理任务，在主机内存/GPU 显存分配后对所有任务进行连续处理，处理结束后再将主机内存/GPU 显存释放。

5 实验结果

由上述设计思路可知，本文提出的基于 GPU

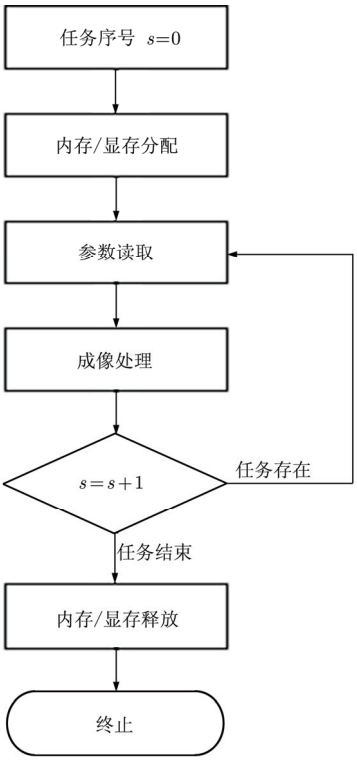


图5 批量任务处理策略图

Fig. 5 Processing strategy of a batch of tasks

的机载 SAR 实时成像处理算法并未对成像处理算法本身做任何更改，也未用单精度代替双精度进行浮点运算，因此所需运算量与基于 CPU 的实现方式相同，而且能够完全达到算法本身应有的处理精度。本文将测试同样处理算法用本文所提出方案在 GPU 上实现时相对于常规在 CPU 上实现时处理速度的提升比例，而未针对分辨率等成像处理指标进行测试。本节测试在 HP Z600 立式工作站上进行，实验所用 GPU 选用 NVIDIA K20C, CPU 选用工作站自带 INTEL E5645 @2.4 GHz。根据 GPU 处理速度提升比例的测试结果，还对价格、功耗、并行度、实时性等方面进行了分析。

5.1 加速比测试

对于某种成像处理算法，对一块机载 SAR 数据的处理时间仅取决于距离及方位向的采样点数，而与载频波段及分辨率无关。因此本节选用ωκ成像处理算法，利用一段 X 波段机载 SAR 仿真数据分别用本文方案和常规基于 CPU 的处理算法进行处理，并对处理所需时间进行对比及讨论。本节分别针对距离/方位采样点数均为 16384 与 32768 两种情况进行了仿真实验。由于实际机载 SAR 数据处理中都需要进行运动补偿处理，因此按照第 2 节所述在图 2 的运算模块 A 中加入了运动补偿的处理环节。其它相关运行参数如表 1 所示，各种处理手段消耗时间的实验结果如图 6 所示，其中蓝色表示 CPU 单线程处理，绿色表示 CPU 全速(6 线程)处理，深红色表示 GPU 处理。由于从磁盘读取原始数据和将结果图像写入磁盘两个环节对各种处理手段来说耗费时间相同，因此图 6 的时间测量结果中未计入这两个环节所花时间。

表1 仿真运行环境及配置参数

Fig. 1 Platform configuration of simulation

主机内存	12 GB
操作系统	Microsoft Windows 7 旗舰版(×64)
CPU	Intel E5645 @2.4 GHz, 6 核心(一颗)
GPU	NVIDIA K20C (一部)
GPU 设备驱动模式	TCC
GPU ECC 开启状态	未开启
每个 CUDA block 线程数	512
SAR 数据分块字节数	256 MB
sinc 插值点数	16
转置操作 tile 点数[10]	32×32

由于显存中的分块大小为 256 MB(占用显存共计 1.5 GB)，对于距离/方位点数为 16384 的 SAR 数据(中间结果数据量为 2 GB)，对所有数据的分块数目为 8；对于距离/方位点数为 32768 的 SAR 数据(中间结果数据量为 8 GB)，分块数为 32。

由图 6 可见，对于距离/方位点数为 16384 的 SAR 数据，与单线程及 6 线程 CPU 处理相比，GPU 处理的加速比分别约为 71 与 14；对于距离/方位点数为 32768 的 SAR 数据，加速比分别约为 80 与 16。通过 NVIDIA 提供的并行处理可视化分析工具 —— Visual profiler[15]观察两者各运算模块的运行情况可知，两种情况下各流的处理及数据传输速度基本相同，但由于数据分块数较大时各运算模块首尾未并行部分占模块所有运行时间比例较大，因此距离/方位点数为 32768 时的加速比较大。

从价格的角度考虑，K20C 目前市场价格约为 E5645 的将近 10 倍，可见与 E5645 相比 K20C 具有约 1.5 倍的价格优势。从功耗的角度考虑，本文方案中 CPU 无需承担任何大型运算任务，因此无需耗资配置高性能 CPU，但须配置具有高速总线以及高速 PCI-E 插槽的主板。从功耗的角度考虑，K20C 在处理时功耗约为 140 W[5]，而 E5645 的标称功耗为

80 W，可见 K20C 具有约 8 倍的功耗优势。从便携性的角度考虑，由于需要约 15 颗 E5645 才能达到一部 K20C 的处理能力，而一块主板上的 CPU 插槽有限(目前市场上主流主板支持 CPU 个数一般不超过 4 个)，因此至少需要配置 4 台主机。可见，选用 GPU 作为处理设备使得 SAR 数据处理设备的便携性大大提高。

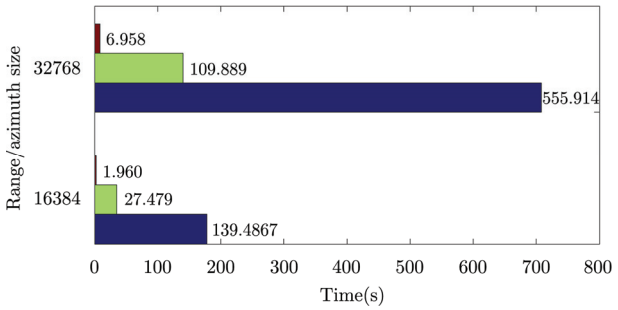
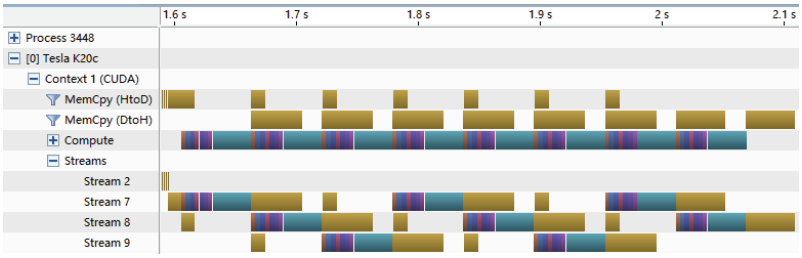


图6 CPU/GPU处理耗时对比图

Fig. 6　Comparison of time consumption between CPU/GPU processing

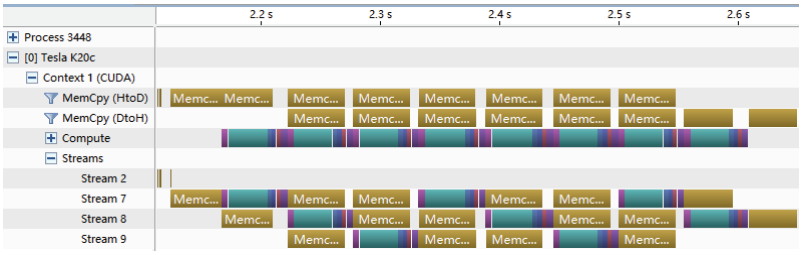
5.2 并行度分析

利用 Visual Profiler 对本文方案的并行度进行分析，观察数据在主机内存与 GPU 显存间的传输与 GPU 运算的并行程度，距离/方位点数为 16384 时的处理过程在 Visual Profiler 中如图 7 所示，其中黄色模块表示 CPU/GPU 间数据传输，其它颜色模块表示各种数据处理 kernel。由图 7 可见，对于图 2 中的运算模块 A, B, C, kernel 的运行几乎没有任何中断，即由于数据传输与 kernel 执行的并行化，各数据块的 kernel 开始执行之前，所需数据都已就绪(各流的首尾数据块除外)。对于运算模块 D，由于数据传入 GPU 时间大于数据传入 GPU 的拷贝过程以及 kernel 执行所需时间，因此前者的执行几乎没有任何中断，而后两者在前者的执行过程中已并行完成，省去了前者的等待时间(各流的首尾数据块除外)。另外，由于各流的首尾数据块未完全并行，各运算模块总体执行时间与 kernel 实际执行时间(或数据单向传输时间，取大者)相比有略微增加。



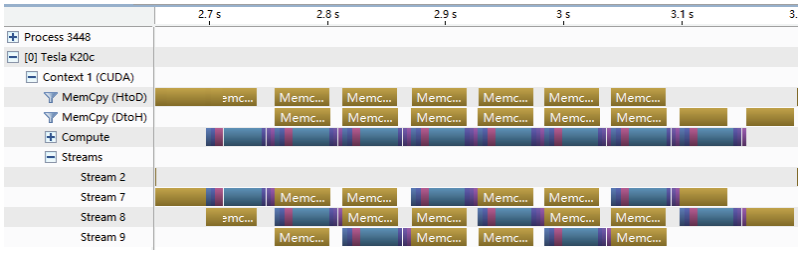
(a) 图2中运算模块A图示

(a) Module A in Fig. 2



(b) 图 2 中运算模块 B 图示

(b) Module B in Fig. 2



(c) 图2中运算模块C图示

(c) Module C in Fig. 2

(d) 图 2 中运算模块 D 图示

(d) Module D in Fig. 2

图7 Visual Profiler中显示的本文方案处理过程

Fig. 7 Processing procedure of computing modules displayed in Visual Profiler

另外，如 3.2.3 节所述，由于采用了事件同步机制，各数据分块的所有 kernel 都实现了连续集中执行，各数据分块的所有 kernel 完全实现了顺序执行，从而避免了各 GPU→CPU 数据传输环节延缓执行。

但由于事件同步机制的影响，从第 2 个数据分块开始，每次 cudaEventSynchronize 调用都需要等待上一数据分块的 cudaEventSynchronize 调用成功返回，即每个数据块的数据处理任务及其之后发射的任务都需要在上一数据块的数据处理完成之后开始执行(launch)[4]，因此导致了运算模块 A, B, C 从第 3 个数据分块开始每次的 CPU→GPU 数据传输环节的执行略有延迟。而运算模块 D 的 kernel 执行时间小于 CPU→GPU 数据传输时间，因此未出现此现象。

5.3 实时性分析

机载 SAR 成像处理算法能否满足实时性要求，取决于处理速度是否大于数据采集速度。假设距离采样点数为 N_{r}，每秒采集脉冲数为 P_{rf}，则数据采集速度为 $N_{\mathrm{r}}P_{\mathrm{rf}}$。可见处理算法的实时性与处理速度、距离采样点数以及每秒采集脉冲数有关，而与 SAR 系统波段、天线尺寸、载机速度等因素无关。基于此，本节的分析无需设计距离采样点数与每秒采集脉冲数之外的其它无关因素。

在不考虑每景 SAR 数据处理的不完全孔径截取的情况下，由图 6 可见，一部 K20C 对机载 SAR 数据的处理速度约为每秒处理 36 M 个原始数据采样点。对于距离采样点数为 64k 的 SAR 系统，则每秒处理数据量约为 576 个脉冲，因此一部 K20C 可以满足脉冲采样点数 64k、脉冲重复频率(PRF)不超过 576 的机载 SAR 系统的 2 维全分辨率实时处理速度需求。

由于机载 SAR 方位向分辨率为载机速度与多普勒带宽之比(与系统波段无关)，假设载机速度为 v，则一部 K20C 能够达到实时处理速度的机载 SAR 系统方位向分辨率约为：

$$\delta_{\mathrm{a}} > \frac{vN_{\mathrm{r}}}{36\times 1024} \tag{5}$$

在此基础上，还可以通过增加 GPU 设备个数(目前 1 部服务器或工作站主板最多可支持 4 部 GPU 设备)扩充设备的处理能力，以达到增加距离采样点数以及提高方位向分辨率等要求。

6 结论

本文对机载 SAR 成像处理算法在 CUDA 架构下的高效实现方法进行了深入研究，首先对常用机载 SAR 频域成像处理算法的结构特征以及 CUDA 架构下的实现思路进行了详细分析，并对算法的 CUDA 实现所需解决的若干技术问题及其解决方法进行了详细论述。基于以上研究工作，提出并详细阐述了一种基于 CUDA 架构的机载 SAR 实时成像处理算法实现方案。该方案针对成像处理算法的结构特征以及 CUDA 架构的并行化特点，通过对显存/内存分配策略以及数据传输/kernel 执行并行化的巧妙设计，借助于 CUDA 编程体系所提供的数据处理及传输接口，实现了处理算法在 NVIDIA GPU 设备上的高效部署。该方案实现了绝大部分的数据传输环节与 kernel 执行的并行进行，大大减少了 kernel 执行的数据等待时间。在 NVIDIA 迄今最新的通用计算高性能产品 K20C 上实验结果表明，借助于本文方案，与传统基于 CPU 的成像处理算法相比，基于 K20C 的机载 SAR 成像处理算法具有低廉的价格、极低的功耗以及高度的便携性，只需一台配备一部或多部 K20C 的主机即可具有相当于数十台普通主机的处理能力。

参 考 文 献

[1] Cumming I G and Wong F H. Digital Processing of Synthetic Aperture Radar Data: Algorithms and Implementation[M]. Norwood: Artech House, 2002.

[2] OpenMP Architecture Review Board. OpenMP application program interface[OL]. http://www.openmp.org/mp-documents/spec30.pdf, May 2008.

[3] Snir M, Otto S, Lederman S H, *et al*.. MPI: The Complete Reference[M]. US: The MIT Press, 1996.

[4] NVIDIA. CUDA Cprogramming guide[OL]. http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf, July 2013.

[5] NVIDIA. Tesla k20 GPU active accelerator: board specification [OL]. http://www.nvidia.cn/content/PDF/kepler/tesla-k20-active-bd-06499-001-v03.pdf, May 2013.

[6] Intel. Intel Xeon Processor E5645 [OL]. http://ark.intel.com/zh-cn/products/48768. 2010.

[7] Yegulalp A F. Fast backprojection algorithm for synthetic aperture radar[C]. Proceedings 1999 IEEE Radar Conference, Waltham, US, 1999: 60–65.

[8] John Jr. C K. Motion compensation for synthetic aperture radar[J]. *IEEE Transaction on Aerospace and Electronic Systems*, 1975, 11(3): 338–348.

[9] Meng Da-di, Hu Dong-hui, and Ding Chi-biao. A new approach to airborne high resolution SAR motion compensation for large trajectory deviations[J]. *Chinese Journal of Electronics*, 2012, 21(4): 764–769.

[10] Liu Bin, Wang Kai-zhi, Liu Xing-zhao, *et al*.. An efficient signal processor of synthetic aperture radar based on GPU[C]. European Conference on Synthetic Aperture Radar, Eurogress, Aachen, Germany, June 2010: 1054–1057.

[11] Ning Xia, Yeh Chun-mao, Zhou Bin, *et al*.. Multiple- GPU accelerated range-Doppler algorithm for synthetic aperture radar imaging[C]. IEEE International Radar Conference, Kansas City, MO, USA, May 2011: 698–701.

[12] Clemente C, Bisceglie M D, Santo M D, *et al*.. Processing of synthetic aperture radar data with GPGPU[C]. IEEE Workshop on Signal Processing Systems, Tampere, Finland, Oct. 2009: 309–314.

[13] 俞惊雷, 柳彬, 王开志, 等. 一种基于 GPU 的高效合成孔径雷达信号处理器[J]. 信息与电子工程, 2010, 8(4): 415–418.
Yu J L, Liu B, Wang K Z, *et al*.. A highly efficient GPU-based signal processor of Synthetic Aperture Radar[J]. *Information and Electronic Engineering*, 2010, 8(4): 415–418.

[14] NVIDIA. Optimizing matrix transpose in CUDA[OL]. http://www.cs.colostate.edu/~cs675/matrixtranspose.pdf, May 2013.

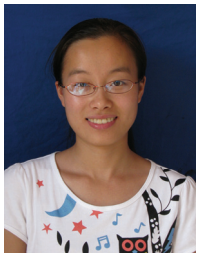
[15] NVIDIA. Profiler user's guide[OL]. http://docs.nvidia.com/cuda/pdf/CUDA_Profiler_Users_Guide.pdf, July 2013.

作 者 简 介

孟大地(1979–)，男，陕西渭南人，2006 年于中国科学院研究生院获工学博士学位，2001 年于西安交通大学获工学学士学位，2006 年起至今任职于中国科学院电子学研究所，副研究员，研究方向为机载合成孔径雷达信号处理。
E-mail: mengdadi@hotmail.com

胡玉新(1981–)，男，内蒙古赤峰人，2007 年于中国科学院研究生院获工学博士学位，2002 年于内蒙古大学获工学学士学位，2007 年起至今任职于中国科学院电子学研究所，研究员，研究方向为合成孔径雷达信号处理系统设计。
E-mail: yxhu@mail.ie.ac.cn

石　涛(1982–)，男，天津人，2007 年于哈尔滨工业大学获工学硕士学位，2007 年起至今任职于中国科学院电子学研究所，助理研究员，研究方向为机载合成孔径雷达实时成像处理系统设计。



孙　蕊(1984–)，女，哈尔滨人，2009 年毕业于吉林大学获工学硕士学位，2009 年起至 2011 年任职于中国电子科技集团公司第十一研究所，2011 年起至今任职于中国科学院电子学研究所，助理研究员，研究方向为机载合成孔径雷达实时成像处理系统设计。

李晓波(1988–)，男，山东青岛人，2011 年毕业于中国石油大学获工学学士学位，2011 年起至今就读于中国科学院大学信号与信息处理专业，博士研究生，研究方向为外辐射源信号处理及机载合成孔径雷达实时成像处理系统设计。