基于 GPU 加速的雷暴追踪外推方法研究

王　兴[1],王　新[2],苗春生[1],王介君[1]

(1.南京信息工程大学大气科学学院,江苏 南京 210044)
(2.北京华风气象影视信息集团公司,北京 100081)

[摘要]　基于气象雷达的雷暴识别与追踪是临近预报中重要的方法之一. 为解决传统算法实时性差的问题,运用 OpenCL 构建异构计算模型对算法进行并行化改进. 通过对算法分支结构优化、OpenCL 设备内存优化,以及针对 VLIW 的优化,分步阐述算法优化的过程和原理. 这些方法不仅使得基于光流的计算速度大幅提升,还可为其他基于 OpenCL 异构计算的优化提供参考. 以 AMD 两代不同架构的 GPU 和 Intel XEON CPU 作为测试平台测试,结果表明,改进后的算法程序在硬件同等功耗的情况下,计算速度提高了 10 至 18 倍.

[关键词]　光流计算,开放运算语言,并行化,临近预报,雷达资料
[中图分类号]TP391.4　[文献标志码]A　[文章编号]1672-1292(2015)01-0035-08

Research on Thunderstorm Track and Extrapolation Based on GPU-Acceleration

Wang Xing[1], Wang Xin[2], Miao Chunsheng[1], Wang Jiejun[1]

(1.School of Atmospheric Science, Nanjing University of Information Science and Technology, Nanjing 210044, China)
(2.Huafeng Group of Meteorological Audio and Video Information, CMA, Beijing 100081, China)

Abstract: The thunderstorms recognition and tracking based on weather radar is one of the important methods in weather nowcasting. In order to solve the problem that the traditional algorithm is poor real-time, this paper discusses the heterogeneous computing model based on GPU, and presents that parallel the algorithm with OpenCL to achieve high performance. By the methods of branching structure optimization, OpenCL device memory optimization, and optimized for VLIW, this paper expounds the optimization of the algorithm step by step. These methods not only boost the speed of optical flow computation, but also provide a reference for other optimization based on OpenCL heterogeneous computing. We use AMD's two generations of different architecture of GPU and Intel XEON CPU as a test platform. The tests results show that the computing performance improve 10-18 times under the circumstance of same power consumption.

Key words: optical flow computation, OpenCL, parallel computing, nowcasting, radar data

我国是全球受自然灾害影响最严重的国家之一,每年因雷雨大风、短时强降水等灾害性天气造成的直接经济损失高达数千亿元,受灾人口多达数亿人次[1]. 基于气象雷达数据的雷暴识别追踪外推是进行强对流天气预报预警的重要手段之一. 当今,业务中应用最为广泛的雷暴外推算法主要包括交叉相关法(Cross Correlation)[2]和单体质心法(Centroid Tracking)[3]等. 光流作为计算机视觉领域中目标识别与追踪的重要方法,近年来也被研究和应用到天气预报等领域,并被实践证明,对于变化较快的强对流降水天气过程,光流法明显优于交叉相关法[4]. 但由于光流算法递归遍历数据和迭代计算的复杂性,每次计算都需花费大量时间,对于全国大区域雷达拼图的光流场计算,其计算速度往往跟不上雷达资料的更新速度,从而极大地影响了预报的实时性和业务的实用性.

OpenCL(Open Computing Language)是一套开放的标准,可广泛应用于多核 CPU、GPU 及其他处理器

收稿日期:2014-08-20.
基金项目:国家科技支撑计划课题(2012BAH05B01)、公益性行业科研计划课题(GYHY201206030)、大学生实践创新训练计划项目(201410300119).
通讯联系人:王兴,博士,工程师,研究方向:气象信息安全技术. E-mail:sweetdreamworks@qq.com

上进行通用目的的并行计算编程,使硬件充分发挥多核心的计算效能[5],进而在现有设备基础上,最大化提升业务算法程序的计算速度.

本文首先阐述基于光流场分析的雷暴识别与追踪算法,对算法进行并行化改进,然后运用 OpenCL 相关技术对 AMD 两代不同架构的 GPU 以及 Intel XEON E5 多核心 CPU 提出性能优化的方法,并通过测试数据比对,分析各个步骤优化的效果.

1　基于光流场的雷暴识别追踪算法

基于光流场的雷暴识别与追踪,其核心思想是从连续的雷达回波资料中计算光流场,得到对流系统不同方位的移动矢量特征,并由此对雷达回波的方位及强度进行外推预测,从而达到预报的目的. 光流分析最早由 Gibson 于 20 世纪 50 年代提出,经过数十年的发展,凝练出很多实用、有效的算法,其中包括 Horn 和 Schunck 提出的 HS 算法[6],Lucas 和 Kanade 提出的 LK 算法[7]等. 总体而言,依据数学理论的不同,光流法大体分为 4 类:微分法、区域匹配法、基于能量的方法和基于相位的方法. 其中,前两种因计算复杂度相当较小,得到更广泛的实践应用. 本文主要针对基于变分的 HS 算法以及金字塔分层模型[8]进行改进.

1.1　HS 光流算法

把雷达回波基本反射率数据作为一个整体加以分析,即将雷达数据的光流场计算作为一个全局优化过程,问题描述如下:

$$E(u,v)=\iint E(u,v,x,y,u_x,u_y,v_x,u_y)\,\mathrm{d}_x\mathrm{d}_y, \tag{1}$$

其中,$E(u,v)$是关于光流场函数 $u(x,y)$,$v(x,y)$和光流梯度 u_x,u_y,v_x,v_y 及坐标 x,y 的全局泛函. 根据光流法图像序列同一位置灰度值不定的假设,在相邻两个时次的雷达数据中,同一坐标的基本反射率值保持不变,则有下列公式:

$$I(x,y,t)=I(x+u,y+v,t+1). \tag{2}$$

等号两侧分别表示前后两时次(x,y)坐标的雷达基本反射率值,且这两个值假设相等. 将式(2)右侧泰勒展开,忽略二阶无穷小项,可得到:

$$I(x,y,t)=I(x,y,t)+u\frac{\partial I}{\partial x}+v\frac{\partial I}{\partial y}+\frac{\partial I}{\partial t}, \tag{3}$$

即

$$u\frac{\partial I}{\partial x}+v\frac{\partial I}{\partial y}+\frac{\partial I}{\partial t}=0. \tag{4}$$

显然,一个约束条件无法求解 u,v 两个变量,为了能够计算出光流,Horn 和 Schunck 提出了全局平滑性假设作为另一个约束条件. 该假设认为光流在整个图像上平滑变化,即变化的加速度为零. 由此,可将光流法的计算看成是求解最小化泛函 $E(u,v)$的过程,可表示为:

$$E=\left(u\frac{\partial I}{\partial x}+v\frac{\partial I}{\partial y}+\frac{\partial I}{\partial t}\right)^2+\lambda\left[\left(\frac{\partial u}{\partial x}\right)^2+\left(\frac{\partial u}{\partial y}\right)^2+\left(\frac{\partial v}{\partial x}\right)^2+\left(\frac{\partial v}{\partial y}\right)^2\right], \tag{5}$$

式(5)等号右侧的两项分别表示基本光流约束方程和全局平滑约束方程. 假设 I_x、I_y、I_t 分别表示雷达数据在 x、y 和时间 t 方向的梯度,即:

$$I_x=\frac{\partial I}{\partial x},\quad I_y=\frac{\partial I}{\partial y},\quad I_t=\frac{\partial I}{\partial t}. \tag{6}$$

定义$(\bar{u},\bar{v})$为各个点的(u,v)周围光流场的加权平均值,则式(5)又可表示为:

$$E(u,v)=(I_xu+I_yv+I_t)^2+\lambda[(u-\bar{u})^2+(v-\bar{v})^2]. \tag{7}$$

要得到 $E(u,v)$最小时 u 和 v 的取值,可令其偏导等于零,即:

$$\begin{aligned}\frac{\partial E(u,v)}{\partial u}&=2I_x(I_xu+I_yv+I_t)+2\lambda(u-\bar{u})=0,\\ \frac{\partial E(u,v)}{\partial v}&=2I_v(I_xu+I_yv+I_t)+2\lambda(v-\bar{v})=0.\end{aligned} \tag{8}$$

利用变分和递归的思想,可以得到求解 u,v 的迭代形式如下[9]:

$$u^{n+1}=\bar{u}^{n}-\frac{I_x(I_x\bar{u}^{n}+I_y\bar{v}^{n}+I_t)}{\lambda+I_x^2+I_y^2},$$
$$v^{n+1}=\bar{v}^{n}-\frac{I_x(I_x\bar{u}^{n}+I_y\bar{v}^{n}+I_t)}{\lambda+I_x^2+I_y^2},\tag{9}$$

其中,n 为迭代的次数,$\bar{u}^{n}$ 和 $\bar{v}^{n}$ 表示第 n 次迭代结果的光流场平均值,第一次计算时两者初始化为 0,此后每次迭代取与其相邻 8 个坐标的算术平均值或基于高斯模板的加权平均值. 设定好迭代终止条件或迭代次数后,利用式(9)即可求得相邻两个时次雷达基本反射率的光流场. 由此,即可推测出雷达回波的强度和位置的发展走势.

由于式(3)经泰勒公式展开舍弃了二阶以上的高阶部分,因此适用该算法的前提条件是两次雷达回波间的相对位移仅可为 1 至 2 个像素,否则,计算结果将有较大的误差. 对于变化较快的强对流天气系统,上述前提往往难以满足. 因此,还须结合采用高斯金字塔分层估计法[10]. 该方法通过高斯平滑滤波和亚像素采样技术获得不同分辨率的数据(图)层,然后从低分辨率开始,逐层进行 HS 光流计算,再将每次计算得到的光流 u,v 分量加上 1 组残差分量 d_x、d_y,迭代到更高一层分辨率的数据(图)层上,可表示为:

$$I(x,y,t)^{L}=I(x+d_x^{L-1}+u^{L},y+d_y^{L-1}+v^{L},t+1)\tag{10}$$

通过逐层计算 u^{L}、v^{L} 直到完成最高分辨率的数据(图)层的光流计算.

1.2　算法的并行化改进

本文测试数据选用东南沿海部分地区 15 部天气雷达的拼图资料,由极坐标转换为 6 000×6 400 的直角坐标形式,数据内容为 1.0 仰角的基本反射率数值.

从 1.1 节的算法描述可以看出,计算过程包括对数据的多层低分辨率采样、计算基本反射率值的导数和 u、v 分量,其中 u、v 的计算量随着迭代次数的增加而急剧上升. 结合天气情势变化的强度特征,金字塔的层数通常取 4~6 层,递归迭代次数取 50~100 次. 上述算法在不做任何改进的情况下,完成一次计算的总运算量不少于 7 680 亿次,即便 Intel 当前最新 Haswell 架构的 i7-4790k CPU,其单核计算能力约为 45 GFLOPS,完成这样一次计算至少需要 2~3 min. 而对于更大覆盖范围的雷达拼图,计算时间必然更长. 因此,如果能够利用 GPU 或 CPU 多核的特性做并行化改进,就能够将计算时间控制在数十秒以内. 因此,对算法作如下 3 方面并行化改进.

1.2.1　金字塔分层的并行

金字塔分层的算法有很多,如式(11)就是一种快速有效的计算方法:

$$I^{L}(x,y)=\frac{1}{4}I^{L-1}(2x,2y)+\frac{1}{8}[I^{L-1}(2x+1,2y)+I^{L-1}(2x-1,2y)+I^{L-1}(2x,2y+1)+I^{L-1}(2x,2y-1)]+$$
$$\frac{1}{16}[I^{L-1}(2x+1,2y+1)+I^{L-1}(2x-1,2y+1)+I^{L-1}(2x+1,2y-1)+I^{L-1}(2x-1,2y-1)].\tag{11}$$

对于每一层的计算,都是将当前层各个像素对应的反射率值经由式(11)计算. 以金字塔最底层的计算为例,程序循环次数为:

$$C_1=(6\,000-1)\times(6\,400-1)\approx 3.84\times10^{7}.$$

显然,对于如此海量的循环次数以及完全相同的算法流程,非常适合采用单指令流多数据流(SIMD)的并行计算. 实现的基本方法是将计算的代码写到 OpenCL kernel 中,由 OpenCL 平台设备自行管理和调度程序执行.

1.2.2　公有项的提取

根据式(9),每次对于 $\bar{u}^{n+1}$ 和 $\bar{v}^{n+1}$ 的求解都须计算两次 $\frac{I_x\bar{u}^{n}+I_y\bar{v}^{n}+I_t}{\lambda+I_x^2+I_y^2}$,仍以金字塔最底层为例,如此重复计算的次数为:

$$C_2=(6\,000-1)\times(6\,400-1)\times50\approx1.92\times10^{8},$$

其中,50 为同一层迭代计算的次数. 尽管此步骤只是公有项的简单提取,但减少了 10^{8} 量级的重复计算,无疑能够显著提升计算的速度.

1.2.3　亚像素插值的并行

由于每次迭代之后,前一次计算得到的 u、v 分量将参与对原始数据的重新采样,因此在每次迭代计算时,数据(图)层差分都需要通过插值得到亚像素精度. 可用的插值算法有很多,如双线性插值、高斯插值或反距离权重插值等. 基于 CUDA 或 OpenCL 平台的 GPU 因其硬件特性,在插值计算方面拥有非常强大的性能优势. 以反距离权重插值为例,基于 OpenCL 的 kenel 代码如下所示:

```
1  int id_x = get_global_id(0);
2  int id_y = get_global_id(1);
3  int width = get_global_size(0);
4  int height = get_global_size(1);
5  float sum = 0,weight = 0;
6  for (int y = -2;y <= 2;y++)
7     for (int x = -2;x <= 2;x++) {
8        float t1 = Data[(id_x+x) + (id_y+y) * width];
9        float t2 = sqrt(x * x * 1.0f+y * y)+0.1f;
10       sum += t1/ t2/ t2;
11       weight += 1/ t2/ t2;}
12 ReturnData[id_x+id_y * width] = sum/weight;
```

其中,id_x、id_y 是 kernel 工作项(Work-Item)的全局 ID,每个 ID 对应一个内核实例,kernel 程序执行时,将以 frontwave(AMD)或 Warp(Nvida)[11]为基本单位并发地执行. 由 id_x、id_y 替代最外两层 for 循环. width 和 height 为全局工作组(Global Work Size)的大小,即将 width×height 的串行计算交由 GPU 或多核 CPU 并发执行. 而同一时刻并发的线程数取决于硬件设备以及 OpenCL 程序的设计. 为简化表示,上述代码没有考虑数组边界条件的判断.

2　基于 OpenCL 的算法优化

2.1　分支结构优化

条件分支是程序开发常用的操作,现代处理器一般都有“分支预测”的优化设计,用于提前“预取”下一条可能执行的指令,使得“取指、译码、执行、保存”等步骤尽可能并行. 而当“分支预测”出错时,提前取到的指令将变得无效,不仅造成有限硬件资源的浪费,还需要再花费若干个时钟周期重新取指.

分支操作对于 GPU 性能的影响尤为突出. 以 AMD 架构的 GPU 为例,线程调度的基本单位称作 frontwave,每个 frontwave 的大小是 64[12]. 由于每个 frontwave 中的线程必须同步执行相同的指令,所以一条 if 语言将被拆成两次执行,即便只有一个线程的条件为 False,其他线程也只能空载等待. 因此,kernel 函数中,过多的条件判断语句将会严重影响程序的性能.

然而,很多情况下条件判断在业务逻辑上又是不可避免的,一种有效的解决方法是尽可以将条件判断语句写成三元表达式,如“x=x<0? 0:x;”的形式. 再如,“if(a>=b)return a;else return b;”可改写为“return a+((b-a)&-(b>a));”等非分支形式.

针对上述算法,kernel 程序中存在若干条用于判断数组越界和检查计算中间结果的分支语句,将其分别作上述形式的改写,即可降低 Processing Element(PE)部件空载的机率,能够提升约 2%的计算性能.

2.2　全局内存的访问优化

根据 AMD GPU 硬件架构,HD7970、HD6850 的内存分别由 12 个和 8 个通道(channel)组成,每个通道又分为若干个 bank. 如果同一周期有多个访存请求指向同一个通道,那么各个请求必须依次执行,此过程称作通道冲突(channel conflict)[13,14]. 对于大量并发的工作项来说,过多的通道冲突必然严重影响 PE 部件的执行效率.

建立合理的 kernel 映射关系可以有效降低通道冲突的机率. 由于 HD7970(/HD6850)的内存总线带宽为 384(/256)bits,即一次访存请求可得到 48(/32)字节的数据,如果让相邻的工作项读写相邻的内存单元,则一次访存请求最多可为 12 个工作项提供数据,AMD 将其称作合并访问(coalescing). 因此,程序设计时应尽可能使得相同工作组(Workgroup)内连续的工作项实现内存合并访问.

2.3　局部内存访问优化

由于数据在 CPU 或 GPU 中计算的速度通常比数据在内存中传输的速度快 1 至 2 个量级,因此,对于数据传输方面的优化至关重要. 根据 AMD 技术文档[13],HD7970 全局内存、二级缓存、一级缓存和局部内存的读取速率分别为 265 GB/s、710 GB/s、1 894 GB/s 和 3 789 GB/s. 如果能充分利用局部内存和缓存,减少 GPU 与全局内存的数据交互,必然能够加快程序执行的速度. 但与 channel conflict 类似,受硬件的约束,局部内存对 channel 中不同 bank 的数据可以并发访问,而相同 bank 的数据则必须串行访问[16,17]. 因此,程序设计时还须考虑到尽量减少 bank conflict 的发生. 若设计不当,本可以一个周期并发完成局部内存访问的工作项或因 bank conflict 而变为串行,致使所有参与运算的部件处于阻塞状态,进而严重影响程序执行的效率.

2.4　针对 VLIW 的优化

采用 Evergreen 架构的 HD6850,硬件指令集称作 Very Long Instruction Word(VLIW),该 GPU 包含 12 个 CU,每个 CU 由 16 个 PE 组成,每个 PE 中由 5 个算法逻辑单元(ALU)协作完成一个 VLIW 指令. AMD OpenCL 编译器会自动将 2 至 5 个没有依赖性的指令操作合成为 1 个 VLIW 指令[15]. 如对于图形图像的处理,数据通常为 RGBA 各个颜色通道的计算,VLIW 的设计能提高 ALU 的使用率,将 Frontwave 的总数降低到原来的 1/4,从而提高程序的计算速度.

借鉴 RGBA 的数据格式,将上述程序中的输入数据改用向量数据类型如 float4、uint4,充分利用每个 PE 中 4 个 ALU 的运算性能,提高 CU 的执行效率.

而与 Evergreen 架构不同,采用 GCN 架构的 HD7970,包含 32 个 CU,每个 CU 又包含 4 个 SIMD,每个 SIMD 直接由 16 个 ALU 组成. 这种架构使得每个 CU 能同时执行 4 条不同的指令. 由于取消了 VLIW 的设计,因此,这种采用向量数据类型的优化方法对于 HD7970 无法得到明显的性能改进.

3　测试结果与分析

3.1　并行优化性能分析

首先,测试传统 HS 算法程序在 CPU 上执行所需的时间,将其作为下文测试比对的基准. 本研究测试的 OpenCL 版本为 OpenCL 1.2 AMD-APP,硬件环境为:Intel XEON E5-2620 CPU,AMD GCN 架构的 HD7970 和 Evergreen 架构的 HD6850 显卡. 因 GPU 架构的差异,在性能优化的方法与效果上也将有所不同. 由于 AMD 发布的 OpenCL 还支持 Intel CPU,因此,测试时还对 Intel XEON E5-2620 CPU 的并行性能进行统计分析. 经测试,传统 HS 算法程序的执行时间为 133.61 s. 通过本节所述的基本并行化改进后,3 种硬件的测试结果如表 1 所示.

为了得到准确可信的测试结果,所有结果均为取连续 5 次同一种测试的平均值. 表 1 中的“计算用时”包括两个数据文件读取和结果输出所用的时间. 可以看出,改进后的计算速度提升了 10 倍以上.

在此基础上,使用 OpenCL 相关技术分别进行全局内存和局部内存的优化. 以光流算法中亚像素精度的计算为例,所处理的数据为二维数组结构,OpenCL NDRange 定义为 2,通过调整工作组的尺寸,得到访存性能的关系如图 1 所示.

图 1 表明工作组大小为{128,1}时 kernel 执行的时间最长,为{1,256}时执行时间最短. 两者性能相差近 32 倍. 这是因为 OpenCL kernel 中二维数组采用行优先的访问方法,当工作组大小为{1,256}时,所有的工作项访问不同的索引地址,使得合并访问的效率达到最高. 又由于硬件本身的限制,一个工作组中的工作项不能超过 256,否则无法完成计

表 1　基本的并行化改进测试结果

Table 1　Time consuming of basic parallel improvements

OpenCL 设备	最大功耗/W	计算用时/s
AMD HD7970	358	4.58
AMD HD6850	108	12.95
Intel XEON E5-2620	95	9.59

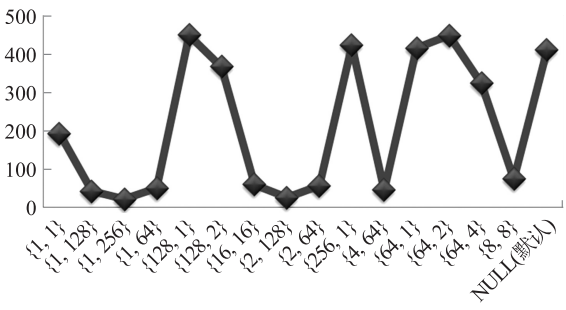

图 1　不同工作组大小的访存性能比较

Fig. 1　Relationship between work group size and memory access performance

算. 因此,选择{1,256}作为工作组大小是最佳的设置,且较工作组设为默认的 NULL,速度提升了 28 倍.

在最佳工作组{1,256}的基础上,将 kernel 计算时用于存储中间结果的各个全局变量改用 Compute Unit(CU)中的局部内存进行存储,对于迭代 100 次的 HS 计算来说,CU 与全局内存的交互次数由 100 次减少到 1 次. 由此,得到局部内存优化前后性能的对比,如表 2 所示.

表 2　局部内存访问优化的性能对比

Table 2　Performance comparison of local memory access optimization

	时间/ms	缓存命中率/%	内存占用百分比/%
优化前	11.90	62.14	85.59
优化后	9.85	71.91	98.81
提升幅度	17.23%	13.59%	13.38%

由表 2 可知,将计算过程中的中间结果保存到局部内存比每次直接与全局变量交互,kernel 执行时间减少了 17%,且缓存的命中率和内存的利用率也有 13%左右的提升.

经上述并行化改进和优化后,3 种 OpenCL 设备执行程序的时间如表 3 所示.

表 3　基于 OpenCL 的算法优化测试结果

Table 3　Performance comparison of OpenCL-based parallelized algorithm optimization

OpenCL 设备名称	最大功耗/W	计算总时长/s					优化后性能提升比例	与串行计算速度比较
		基本并行	分支结构优化	全局内存优化	局部内存优化	向量数据类型		
AMD HD7970	358	4.58	4.44	3.61	3.40	3.39	26%	10 倍
AMD HD6850	108	12.95	12.54	8.58	7.53	6.40	51%	18 倍
Intel XEON E5-2620 CPU	95	9.59	9.58	9.31	8.89	8.87	8%	15 倍

由表 3 可以看出,全局内存优化使得 AMD GPU 的计算速度提升超过 20%,在所有优化策略中效果最明显. 采用向量数据类型的优化后,HD6850 的计算速度提高了 9%,而 HD7970 的却没有任何提升. 可以看出,新一代架构的 HD7970 较上一代的 HD6850 从硬件层面作了大量改进,使用优化带来的总体性能提升不如 HD6850 显著. 另一方面,这些优化技术也能为 Intel CPU 带来 9%的性能提升,特别是对于局部内存的优化,由于区别于显存的硬件架构,因此采用了 CPU 缓存(cache)优化技术. 而 CPU 对于分支结构优化不敏感,也证明了 CPU 在逻辑处理方面的绝对优势.

总体而言,通过上述改进和优化,将原本需要 134 s 完成的计算缩短到最少 3.39 s 完成,速度提升了近 40 倍. 为更具可比性,参照表 3 的设备最大功率可以算出,设备在等功耗情况下,计算速度分别提高了 10 倍(HD7970)、18 倍(HD6850)和 15 倍(XEON E5-2620).

3.2　外推预测结果分析

测试数据使用 2014 年 3 月 12 日 06:18 和 06:24 的雷达拼图数据,雷达回波强度如图 2(a)、(b)所示. 通过上述程序计算出的光流场如图 3 所示.

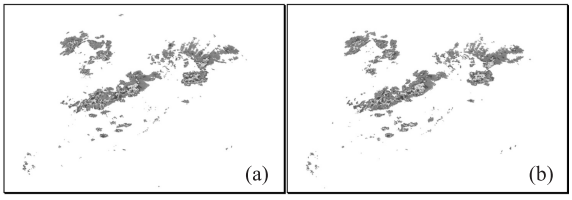

图 2　雷达回波基本反射率实况

Fig. 2　Radar echoes of weather real-time monitoring

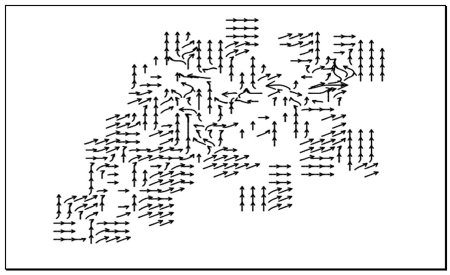

图 3　雷达回波基本反射率的光流场

Fig. 3　Optical flow of radar echoes base reflectivity

通过光流场的 u、v 分量及时间关系,推测出当天 06:30 的回波分布和强度如图 4(a),以及同一时间的实况拼图 6(b).

直观上看,图 4(a)、(b)的回波位置分布和强度都非常接近. 为了得到定量比较结果,采用误警率(FAR)、命中率(POD)和成功指标(CSI)[18]对基于光流的推测结果用实况数据进行检验,结果如表 4 所示.

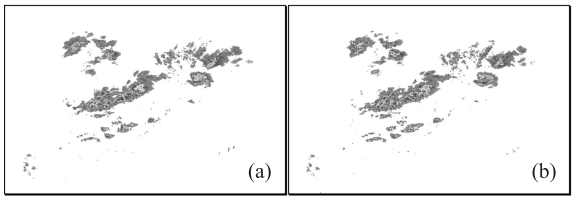

图 4　光流外推预报与实况对比

Fig. 4　Radar echoes through above extrapolating forecast

由表 4 可知,基于 HS 算法及金字塔分层模型的雷达回波追踪外推方法,其准确性能够达到强对流临近预报的基本要求,但由于 HS 算法没有考虑天气系统特别是雷暴云生、消的物理机制,因此,在外推算法及预报准确性方面仍有提升和改进的空间.

表 4　基于光流的外推预报准确性检验

Table 4　Accuracy check on optical flow-based extrapolating forecast

检验方法	检验结果
FAR	0.190
POD	0.804
CSI	0.676

4　结束语

本文利用 GPU 及多核 CPU 在并行计算方面的优势,对基于光流场的雷暴识别与追踪外推算法进行并行化改进,然后针对 OpenCL 平台及硬件特性,对算法程序进行优化. 采用的方法包括分支结构优化、设备全局内存和局部内存优化,以及针对 VLIW 的优化. 这些方法的应用不仅适用于光流计算的优化,也可为其他基于 OpenCL 异构计算的程序优化提供参考.

通过气象业务上实际数据的测试,结果表明,应用该方法外推出的雷达回波强度及位置分布,其准确性能够满足强对流临近预报的业务要求,通过一系列并行优化,不仅可以使 AMD GPU 得到 10 倍以上的性能提升,还能利用现有的多核 CPU 资源,在不增添硬件的情况下,将计算时间缩短 10 倍以上.

[参考文献](References)

[1] 陆林. 我国公共气象服务能力建设研究[D]. 云南:云南大学公共管理学院,2013.
Lu Ling. Research of the construction ability for public meteorological service in China [D]. Yun Nan: School of Public Administration, Yunnan University, 2013. (in Chinese)

[2] 陈明轩,王迎春,俞小鼎,等. 交叉相关外推算法的改进及其在对流临近预报中的应用[J]. 应用气象学报,2007,18(5):690-700.
Chen Mingxuan, Wang Yingchun, Yu Xiaoding. Improvement and application test of TREC algorithm for convective storm nowcast [J]. Journal of Applied Meteorological Science, 2007, 18(5): 690-700.

[3] Johnson J T, MacKeen P L, Witt A, et al. The storm cell identification and tracking algorithm: an enhanced WSR-88D algorithm[J]. Weather and Forecasting, 1998, 13: 263-276.

[4] 韩雷,王洪庆,林隐静. 光流法在强对流天气临近预报中的应用[J]. 北京大学学报:自然科学版,2008,44(5):751-755.
Han Lei, Wang Hongqing, Lin Yinjing. Application of optical flow method to nowcasting convective weather[J]. Acta Scientiarum Naturalium Universitatis Pekinensis, 2008, 44(5): 751-755. (in Chinese)

[5] Owens J D, Luebke D, Govindaraju N, et al. A survey of general-purpose computation on graphics hardware[J]. Computer Graphics Forum, 2007, 26(1): 80-113.

[6] Strecha C, Fransens R, Gool L V, A probabilistic approach to large displacement optical flow and occlusion detection, statistical methods in video processing[J]. Lecture Notes in Computer Science, 2004, 3 247: 1-82.

[7] Xiao J, Cheng H, Sawhney H, et al. Bilateral filtering-based optical flow estimation with occlusion detection[J]. ECCV, Lecture Notes in Computer Science, 2006, 3951: 1-224.

[8] 朱倩. 基于改进的光流场算法对运动目标的检测与跟踪技术研究[D]. 哈尔滨:哈尔滨工程大学信息与通信工程学院,2006.
Zhu Qian. Moving objects detection and tracking based on revised optical flow[D]. Harbin: College of Information and Communication Engineering, Harbin Engineering University, 2006. (in Chinese)

[9] 刘闵. 中尺度对流系统中多个云团的跟踪和预测[D]. 南京:南京理工大学模式识别与智能系统,2013.
Liu Min. Tracking and prediction of multiple-clouds in the mesoscale convective system[D]. Nanjing: Pattern Recognition and Intelligent Systems, Nanjing University of Science and Technology, 2013. (in Chinese)

[10] 江志军,易华蓉. 一种基于图像金字塔光流的特征跟踪方法[J]. 武汉大学学报:信息科学版,2007,32(8):680-683.
Jiang Zhijun, Yi Huarong. An image pyramid-based feature detection and tracking algorithm[J]. Geomatics and Information Science of Wuhan University, 2007. (in Chines)

[11] 张云泉,张先轶,龙国平,等. OpenCL 异构计算[M]. 北京:清华大学出版社,2012:1-148.
Zhang Yunquan, Zhang Xianyi, Long Guoping, et al. Heterogeneous Computing with OpenCL [M]. Beijing: Tsinghua

University Press,2012:1-148.(in Chinese)

[12] Advanced Micro Devices, Inc. AMD Accelerated Parallel Processing OpenCL Programming Guide[M]. [S.l.]. [s.n.]. November 2013.

[13] Aaftab Munshi, Benedict Gaster, Timothy G Mattson, et al. OpenCL Programming Guide[M]. Boston: Addison-Wesley Professional, 2011.

[14] 李约炯. 跨平台的多核与众核编程讲义[M]. 上海:AMD 上海研发中心,2010.
Li Yuejiong. Cross-platform and Multi-core Programming Notes[M]. Shanghai: AMD Shanghai r&d Center, 2010.(in Chinese)

[15] 佚名. AMD 的显卡架构与 OpenCL 性能之间的一点思考[EB/OL]. [2014-03-27]. http://pengx17.me/opencl/2013/09/25/amd-architect/.
Anon. Thinking about AMD Graphics between architecture and OpenCL Performance[EB/OL]. [2014-03-27]. http://pengx17. me/opencl/2013/09/25/amd-architect/.(in Chinese)

[16] 仇德元. 从 GLSL、CUDA 到 OpenCL[M]. 北京:机械工业出版社,2011.
Qiu Deyuan. From the GLSL, CUDA to OpenCL[M]. Beijing: China Machine Press, 2011.(in Chinese)

[17] Guo Yidong, Liu Bozhong, Qiu Weidong. General-purpose oriented GPU optimization techniques on AMD platforms[C]//Proceedings of 2013 IEEE International Conference on Computer Science and Automation Engineering. Guangzhou: IEEE Beijing Section, 2013: 1 284-1 288.

[18] 邵晨. 基于雷达与闪电观测的短历时强降水分类研究[D]. 南京:南京信息工程大学大气科学学院,2012.
Shao Chen. Classification of short-time heavy rain using radar and lightning observations[D]. Nanjing: School of Meteorology Science, Nanjing University of Information Science and Technology, 2012.(in Chinese)

[责任编辑:顾晓天]