doi:10.11887/j.cn.201505006 http://journal.nudt.edu.cn

CPU-GPU 协同加速 Kriging 插值的负载均衡方法*

姜春雷[1,2], 张树清[1]
(1. 中国科学院 东北地理与农业生态研究所, 吉林 长春 130102; 2. 中国科学院大学, 北京 100049)

摘 要:Kriging 插值算法被广泛应用于地学各领域,有着极其重要的现实意义,但在面对大规模输出网格及大量输入采样点时,不可避免地遇到了性能瓶颈。利用 OpenCL 和 OpenMP 在异构平台上实现了 CPU 与 GPU 协同加速普通 Kriging 插值。针对 Kriging 插值中采样点的不规则分布及 CPU 和 GPU 由于体系结构差异对其的不同适应性,提出一种基于不同设备间计算性能的差异和数据分布特点的负载均衡方法。试验结果表明,该方法能有效提高普通 Kriging 插值速度,同时还能节约存储空间和提高访存效率。

关键词:通用计算图形处理器;开放运算语言;Kriging 插值;负载均衡
中图分类号:F209 **文献标志码**:A **文章编号**:1001-2486(2015)05-035-05

A load balancing method in accelerating Kriging algorithm on CPU-GPU heterogeneous platforms

JIANG Chunlei[1,2], *ZHANG Shuqing*[1]
(1. Northeast Institute of Geography and Agroecology, Chinese Academy of Sciences, Changchun 130102, China;
2. University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Kriging interpolation algorithm is of great practical significance and is widely applied to various fields of geoscience. However, Kriging interpolation would inevitably encounter the performance bottleneck when the output grid or input samples increase. Implemented with OpenCL and OpenMP, the ordinary Kriging interpolation was accelerated on heterogeneous platforms: GPU and CPU. By considering the performance difference of CPU and GPU on the densities of samples, a new load balancing method of LBCPDD (Load Balancing based on Computation Performance and Data Distribution) was proposed, in which not only hardware performance but also data distribution characteristics were taken into account. Experiment results show that LBCPDD method can effectively enhance the speed of ordinary Kriging, save memory space and improve the efficiency of memory access.

Key words: general purpose graphics processor units; open computing language; Kriging interpolation; load balancing

Kriging 插值算法是一种考虑区域变量空间变异结构(自相关结构)特点的线性无偏最优估计插值方法[1],它在地学的很多领域都有着重要的应用。但是,当 Kriging 插值的输出网格增大及输入采样点增多时,其计算时间急剧增加[2]。当前,集成 Kriging 插值算法的主流地理资讯系统(Geographic Information System,GIS)仍以串行计算为基础框架,不能充分利用多核处理器的优势[3]。随着各种并行技术的不断成熟,地理空间栅格数据算法的并行化研究成为新的热点[4]。很多研究者试图通过不同的并行技术来加速 Kriging 插值。目前,针对 Kriging 插值的加速方法主要基于的编程模型有消息传递接口(Message Passing Interface, MPI)[5],OpenMP[6]和通用计算图形处理器(General Purpose Graphics Processing Units, GPGPU)[7-8]。基于这些不同编程模型的 Kriging 插值算法都取得了很好的加速效果,尤其是 GPGPU 编程模型的加速比更高。然而,GPGPU 加速算法仅将 CPU 作为一个流程控制端,造成了 GPU 计算时 CPU 空闲等待,显然,CPU 与 GPU 共同承担计算任务的方式是未来协同并行计算的发展方向[9]。

多设备协同加速需要解决的核心问题之一是设备间的负载均衡问题。目前关于计算设备间负载均衡的研究主要集中在根据不同设备的性能差异分配计算的静态负载均衡[10-11]以及在计算过程中根据不同计算设备进度分配计算的动态负载均衡[12]。夏飞等[13]对于异构设备(例如 CPU 与 GPU)间的负载均衡给出了根据计算能力分发计

* **收稿日期**:2015-06-24
基金项目:国家自然科学基金资助项目(41271196);中国科学院重点部署资助项目(KZZD-EW-07-02)
作者简介:姜春雷(1981—),男,吉林德惠人,博士研究生,E-mail:jiangchunlei@iga.ac.cn;
张树清(通信作者),男,研究员,博士,博士生导师,E-mail:zhangshuqing@iga.ac.cn

算任务的静态负载均衡方法;赵斯思和周成虎[14]对于单个 GPU 上多个 kernel 同时执行,提出一种基于动态规划的动态负载均衡方法,有效地加速了算法的执行。对于 Kriging 插值而言,由于受限于自然环境、经济和人力等条件的限制,采样点通常不规则,表现为分布不均匀。由于体系结构的差异,对于数据量相同但分布密度不同的数据,不同体系结构的硬件通常表现出不同的计算速度。这表明负载均衡时不仅要考虑不同设备间性能的差异,还应考虑数据的特征。

本文提出了一种新的综合考虑设备计算性能和数据特征的负载均衡方法(Load Balancing based on Computation Performance and Data Distribution,LBCPDD),来实现设备间高效合理的负载均衡,以达到更快加速 Kriging 插值的目的。

1 背景知识

1.1 普通 Kriging

普通 Kriging 插值公式可以表示为:

$$z^*(x) = \sum_{i=1}^{n} \lambda_i z(x_i) \tag{1}$$

式中,$z^*(x)$ 表示对未知点 x 的估计值,$x_i(j = 1, 2, \cdots, n)$ 是距离 x 最近的 n 个样本点,$z(x_i)$ 是样本点 x_i 的观测值,λ_i 为权重系数。为了使 $z^*(x)$ 的估计误差最小,通过两个约束条件求出 λ_i:

a. 无偏性条件。

$$\begin{aligned} &E[z(x)] = m, \\ &E[z^*(x)] = m\sum_{i=1}^{N}\lambda_i \Rightarrow \sum_{i=1}^{N}\lambda_i = 1 \end{aligned} \tag{2}$$

其中,E 表示数学期望,m 是中间变量。

b. 最优性条件(即估计值误差的方差最小)。

$$\sigma_E^2 = VAR[z^*(x) - z(x)] \tag{3}$$

依据拉格朗日乘数法原理,建立拉格朗日函数:

$F = VAR[z^*(x) - z(x)] - 2\mu(\sum_{i=1}^{n}\lambda_i - 1)$,其中 -2μ 是拉格朗日乘子,通过函数 F 求出对 n 个权重系数 λ_i 的偏导数,并令其为0,同无偏性条件建立方程组:

$$\begin{cases} \sum_{j=1}^{N}\lambda_j\gamma(x_i,x_j) + \mu = \gamma(x_i,x) \\ \sum_{i=1}^{N}\lambda_i = 1 \end{cases} \tag{4}$$

r 是变异函数,上述方程组可用矩阵表示如下:

$$\begin{Bmatrix} \gamma_{11} & \gamma_{12} & \cdots & \gamma_{1N} & 1 \\ \gamma_{21} & \gamma_{22} & \cdots & \gamma_{2N} & 1 \\ & & \cdots & & \\ \gamma_{N1} & \gamma_{N2} & \cdots & \gamma_{NN} & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{Bmatrix} \cdot \begin{Bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \\ \mu \end{Bmatrix} = \begin{Bmatrix} \gamma_{10} \\ \gamma_{20} \\ \vdots \\ \gamma_{N0} \\ 1 \end{Bmatrix}$$

解上述方程组,求得 λ_i 代入普通 Kriging 插值公式可计算出待估未知点的值。

1.2 通用计算图形处理器

GPGPU 将原本只用于图形处理的硬件用于通用计算领域,OpenCL 和统一计算设备架构(Compute Unified Device Architecture,CUDA)是实现这个理念的两种主要软件开发平台,OpenCL 由于能够在不同的硬件平台上运行,更具应用前景。目前,GPGPU 已经被广泛地应用于各领域的科学研究,其相关知识已经被相关文献大量描述,本文仅介绍在其他文献中被较少描述且被应用于本研究的相关内容。AMD 公司和 NVIDIA 公司对同一硬件概念有不同的称谓,由于本研究基于 AMD 公司的 GPU 实现,因此在以后的描述中以 AMD 公司的概念为准。在 AMD 公司的 GPU 中,一个计算单元(Compute Unit,CU)可以同时运行多个 workgroup,具体硬件运行的 workgroup 数及 workgroup 内的线程数由 GPU 的硬件环境决定,主要是内存数量,更少的内存消耗,可以让每个 CU 容纳更多的 workgroup 和每个 workgroup 容纳更多线程。workgroup 内线程被分成若干个 wavefront(等同于 NVIDIA 硬件中的 warp),它是硬件调度线程执行的最小单位,其内部的线程以严格锁同步方式执行。如果 wavefront 内线程存在不同分支,则每个线程会因为锁同步方式而执行全部分支(无计算任务线程执行空操作),导致执行时间增加。数量足够的 wavefront 可以保证在一个 wavefront 等待内存访问结果时,其他的 wavefront 可以被调入执行单元执行(即交叉访问),这种方式大大提高了 GPU 的利用效率。因此 wavefront 对程序的执行速度有着巨大的影响。OpenCL 并未对开发者提供 wavefront 抽象,但是通过对不同线程处理数据的合理安排可以提高 GPU 的运行效率,即提高 wavefront 数量以及减少单个 wavefront 内的线程分支。

2 基于计算性能与数据分布的负载均衡方法

在 Kriging 插值过程中,相邻的未知点通常有相同的邻近点。由于变异函数矩阵是基于待估未

知点的邻近点构造的,因此相邻的未知点通常具有相同的变异函数矩阵。根据这一特点,在 Kriging 插值算法中增加一个步骤用于检查相邻未知点间是否具有相同的邻近点:若存在,则只存储其中一个未知点的邻近点,并且仅对其构造变异函数矩阵和求逆矩阵,即将相同邻近矩阵合并存储及计算,其他的未知点在计算过程中使用这个计算的结果。

采样点一般是不规则(不均匀)分布的,局部 Kriging 插值在对每个未知点进行插值时,需要求出离它最近的若干个采样点,在采样点密度较小的情况下,相邻未知点具有相同邻近点的概率更大,如图 1 左上角所示,两个未知点(*)具有相同的临近采样点(·);而采样点较密集时则相反(如图 1 右下角所示):两个未知点(*)具有不同的采样点(·)。对于 GPU,较多的具有相同邻近点的线程使 wavefront 内产生更少的分支;由于合并存储和计算,采样点稀疏区域的未知点预测会消耗更少的内存和计算;内存消耗的减少让 GPU 能够容纳更多的线程并发执行以便 GPU 线程的交叉执行减少内存延迟,即不同采样点密度会影响 GPU 的性能。因而在负载均衡过程中不仅需考虑 CPU 与 GPU 自身的计算能力差异,还应考虑 CPU 与 GPU 对不同数据空间分布特征的适应性差异。在算法中,将插值区域划分为若干个网格,统计每一个网格里面的采样点个数,包含较多采样点的网格中的未知点预测交给 CPU 处理,反之,交给 GPU 处理(如图 2 所示,灰色表示密度较大,白色较小)。

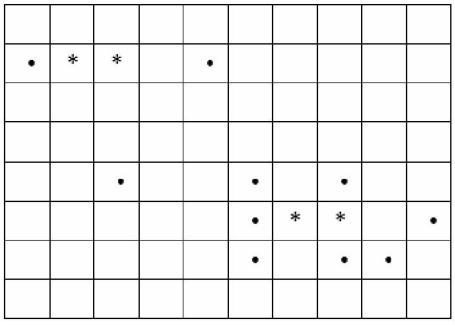

图 1　邻近点查找与采样点分布的关系

Fig. 1　Relation between searching nearest neighbor and the distribution of samples

CPU 与 GPU 之间任务的分配比例通常依据设备浮点计算能力,然而,考虑到 Kriging 插值的复杂性及数据传输的影响,浮点计算能力不能体现出 CPU 与 GPU 在计算 Kriging 插值时的性能差异,因此,依据 CPU 与 GPU 单独计算 Kriging 插值

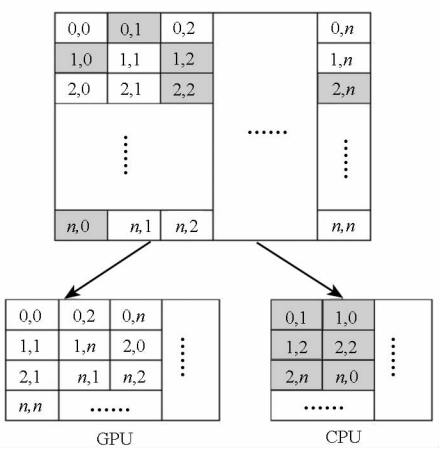

图 2　CPU 与 GPU 负载分配

Fig. 2　Load allocation between CPU and GPU

时的时间初步估计 GPU 任务分配比例:

$$Task_{\mathrm{GPU}} = 1 - \frac{Time_{\mathrm{GPU}}}{Time_{\mathrm{GPU}} + Time_{\mathrm{CPU}}} \tag{5}$$

然后根据结果向 GPU 适当倾斜任务量。上述任务分配估计方法体现了 CPU 和 GPU 对本算法的性能差异,因此仅需 CPU 或 GPU 单独运行一次就可以估算出结果,在之后的计算中即便针对不同的数据集也不需要再重新估算任务分配比例。由于 CPU 端的串行执行并不能完全利用 CPU 的计算能力,因此式(5)中 $Time_{\mathrm{CPU}}$ 指的是 OpenMP 并行方式下的执行时间。

在 LBCPDD 算法的实现中,CPU 端以 OpenMP 方式并行。为了对提出的 GPU 算法的性能进行比较,实现了同样采用局部插值的 Shi 和 Ye 描述的 Kriging 算法[8]。

3　试验与分析

采用的 GPU 为华硕 R9 280,显示核心为 AMD Tahiti XTL 次世代图形核心(Graphics Core Next,GCN)架构,拥有 3G GDDR5 显存,流处理器数量为 1792 个;CPU 为 AMD FX－8150 主频为 3.6GHz,8 核心,8 线程;操作系统为 Win7 64 位。实验数据为从美国国家海洋和大气局(National Oceanic and Atmospheric Administration, NOAA)下载的全球降雨数据,为了保证鲁棒性,根据实验所需采样点数据集的大小,随机选择相应数量的点作为采样点。所有的实验中插值的输出网格大小均为 1280×800 个像素。

试验 1:CPU 性能。为了测试算法在不同数据量下的表现,选取了 5 组数据集进行测试,数据量每次增加 1000 个采样点;OpenMP 分别选择 2 核、4 核和 8 核进行测试(在表 1 中表示为 OMP2,

OMP4 和 OMP8)。算法执行时间见表 1:随着核数的增加,运行时间持续减少;随着数据量增加,不同算法的执行时间都有所增加。图 3 给出了 OpenMP 在不同线程(核心)下的加速比,从图 3 可以看出随着线程数的增加,OpenMP 的加速比也随之增加,最高达到了 3.9 倍左右;不同线程的 OpenMP 方法加速比对数据集大小不敏感,只是数据量比较大时有些轻微波动。

表 1　串行与 OpenMP 运行时间对比

Tab. 1　Time comparison of sequential code and OpenMP　ms

数据集	6000	7000	8000	9000	10 000
串行	212 192	222 690	231 529	282 516	294 138
OMP2	120 938	124 332	132 163	174 283	183 674
OMP4	78 286	83 319	86 377	90 043	93 412
OMP8	64 238	68 079	69 591	72 696	88 873

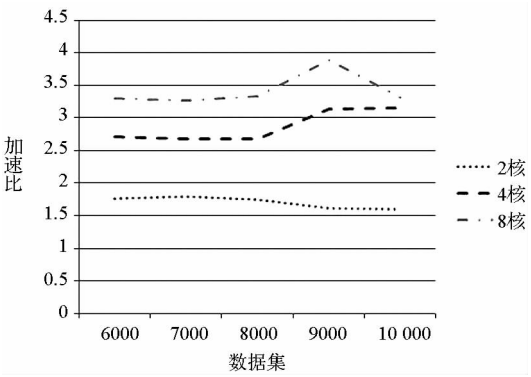

图 3　OpenMP 加速比比较

Fig. 3　Comparison of speed-up of OpenMP

试验 2:GPU 性能。表 2 给出了 GPU 并行的运行时间及加速比。GPU1 并行程序依据 Shi 和 Ye 描述的方法实现[8],GPU2 是改进后的 GPU 算法,增加了检查相邻未知点间是否具有相同的邻近点的步骤。从表 2 可以看出:GPU 算法在不同的数据集下都表现出很好的加速效果,GPU2 最高达到 21.53 倍;GPU2 由于对 Kriging 插值过程中各未知点的邻近点进行了比较,对拥有相同邻近点的未知点,只计算其中的一个未知点的变异函数矩阵和逆矩阵,因此 GPU2 的加速比明显高于 GPU1 的加速比;同 OpenMP 算法类似,GPU 算法的运行时间随数据集的大小增加而增加,但其加速比同数据集的大小无关,波动不大。

表 2　GPU 程序运行时间及加速比

Tab. 2　Time and speed-up of GPU program

数据集		6000	7000	8000	9000	10 000
GPU1	时间/ms	22 467	24 143	24 705	25 438	26 890
	加速比	9.44	9.22	9.37	11.11	10.94
GPU2	时间/ms	11 334	11 617	12 460	13 121	14 466
	加速比	18.72	19.17	18.58	21.53	20.33

试验 3:CPU-GPU 性能。表 3 给出了通过 LBCPDD 进行负载均衡后的 CPU-GPU 协同执行时间。表中,OpenMP 表示 CPU 端并行方式,采用 8 核并行,任务量占 1/8;GPU2L 表示 GPU 端使用表 2 中 GPU2 算法且通过 LBCPDD 方法负载均衡分配数据,任务量占 7/8。由于 GPU 并行与 OpenMP 并行同时执行,因此算法的执行时间取决于较长的时间(负载分配时间小于 3ms,未列入统计)。表 3 中性能提升指总时间相对 GPU 单独执行时间(表 2 中 GPU2)的性能提升((GPU2 - 总时间)/GPU2)。从表 3 中可以看出,CPU-GPU 的运算总时间比 GPU 单独运算时在不同的数据集下都有提高,最高达到了 29.95%。CPU-GPU 算法减少的时间一部分来自于 CPU 分担了部分计算,另外一部分来自于 GPU 由于处理数据的优化而节约的时间。

表 3　引入 LBCPDD 后 CPU-GPU 协同运行时间

Tab. 3　Run time of CPU-GPU cooperation with LBCPDD

数据集	6000	7000	8000	9000	10 000
OpenMP(任务比例:1/8)	7940	9001	9750	9656	10 608
GPU2L(任务比例:7/8)	4963	5687	6439	7424	9412
总时间/ms	7940	9001	9750	9656	10 608
性能提升/%	29.95	22.52	21.75	26.41	26.67

试验 4:LBCPDD 方法对性能的影响。表 4 给出了 OpenMP 及 GPU 并行在应用 LBCPDD 前后完成各自分配任务量的计算时间及性能变化。标准方法指在分配任务时根据插值点在二维平面上的坐标顺序将前 7/8 分配给 GPU,剩余部分分配给 CPU。表中 GPU2S 和 GPU2L 分别表示经过标准方法和 LBCPDD 方法进行任务分配后的 GPU 端执行,任务量为 7/8;OpenMP1 和 OpenMP2 分别表示经过标准方法和 LBCPDD 方法进行任务分配后的 CPU 端执行,任务量为 1/8;GPU2L 性能提升指 GPU2L 相对于 GPU2S 的性能提升((GPU2S-GPU2L)/GPU2S);OpenMP2 性能提升指 OpenMP2 相对 OpenMP1 的性能提升((OpenMP1-OpenMP2)/OpenMP1)。从表 4 中可

以看出,LBCPDD 方法有效减少了 GPU 的计算时间,性能最多提高 50.43%,而对 OpenMP 方法的性能影响相对较小。这主要是由于将适合 GPU 计算的数据分给了 GPU,而将其余数据分给了对数据分布并不敏感的 CPU。由于 OpenMP 运行时,各线程间以无序方式异步运行,这样做有利于 OpenMP 线程间动态负载均衡,但如果各线程间需要同步,则会极大地影响其速度,故 OpenMP 程序无法从减少冗余矩阵和数据重组中获得好处。

表 4　LBCPDD 对 CPU 和 GPU 的影响

Tab. 4　Impact of LBCPDD on GPU and CPU

数据集		6000	7000	8000	9000	10 000
GPU2S	时间/ms	10 012	10 456	11 320	12 088	13 026
GPU2L	时间/ms	4963	5687	6439	7424	9412
	性能提升/%	50.43	45.61	43.12	38.58	27.74
OpenMP1	时间/ms	8814	9328	10 280	11 122	11 154
OpenMP2	时间/ms	7940	9001	9750	9656	10 608
	性能提升/%	9.92	3.51	5.16	13.18	4.90

在 CPU 与 GPU 协同计算中,对采样点的密度进行了分析,将采样点分布密度较低的区域分给了 GPU。这样做有很多好处:更多的冗余矩阵分布在一个 workgroup 中,将有利于单个 wavefront 内的线程具有更少甚至没有分支。大量的冗余矩阵分布在一个 workgroup 内会使后续计算读取同一块内存区域,从而可以基于 GPU 内存存取机制(广播)减少延迟。CPU 与 GPU 之间解决内存延迟使用两种不同策略,CPU 用了大量的芯片空间构造片内缓存,主要通过多级缓存来减少内存访问的延迟;而 GPU 芯片大量空间被用来构造计算单元,通过大量线程的交叉访问来隐藏内存延迟,更多的冗余矩阵导致更少的内存消耗,从而可以让更多的线程调度到 CU 中执行,为线程的交叉访问提供条件。在冗余矩阵合并存储时,通过一个全局变量记录每个线程对应的矩阵编号,由于 OpenCL 不支持全局同步,因此,通过对全局变量的原子操作来完成编号的记录,原子操作具有排他性,会严重影响性能,冗余矩阵减少后,原子操作减少。一个 workgroup 内冗余矩阵增多,意味着线程计算时的内存消耗减少(大量线程空操作),这样会减少变量溢出到全局存储器中,全局存储器的存储周期约 500 个时钟周期,由于等待时间还可能翻倍,而寄存器存储周期只有 1 个时钟周期,马安国等[15]在通过预取对 GPU 程序进行优化时,性能影响最高达 38%。基于以上原因,数据重分配后 GPU 计算的时间明显减少。从表 4 可以看到,随着采样点数据的增多,采样点的密度增加,导致冗余矩阵减少,这将导致性能改善降低,表 4 中数据重分配与按序分配时 GPU 性能改善随采样点数据增加而减少就验证了这一点。

4　结论

Kriging 插值算法是一种广泛应用的算法。利用 CPU 与 GPU 协同计算对 Kriging 插值进行加速,基于 CPU 与 GPU 的计算性能差异和插值数据采样点的不均匀分布,对 CPU 与 GPU 进行负载均衡。测试结果表明,采用了 LBCPDD 后的 CPU-GPU 协同加速方法优于单纯依赖 CPU 的 OpenMP 加速方法、完全 GPU 的算法和未采用 LBCPDD 的 CPU-GPU 协同加速方法。LBCPDD 法对其他数据分布敏感的 CPU-GPU 协同加速算法在进行负载均衡时也同样具有参考价值。

参考文献(References)

[1] Krige D G. A statistical approach to some basic mine valuation problems on the witwatersrand[J]. Journal of the Chemical Metallurgical & Mining Society of South Africa, 1951, 94(3):95 - 111.

[2] De Baar J H S, Dwight R P, Bijl H. Speeding up Kriging through fast estimation of the hyperparameters in the frequency-domain [J]. Computers & Geosciences, 2013, 54: 99 - 106.

[3] 吴立新, 杨宜舟, 秦成志, 等. 面向新型硬件构架的新一代 GIS 基础并行算法研究[J]. 地理与地理信息科学, 2013, 29(4): 1 - 8.
WU Lixin, YANG Yizhou, QIN Chengzhi, et al. On basic geographic parallel algorithms of new generation GIS for new hardware architectures [J]. Geography and Geo-Information Science, 2013, 29(4): 1 - 8. (in Chinese)

[4] 程果, 陈荦, 吴秋云, 等. 一种面向复杂地理空间栅格数据处理算法并行化的任务调度方法[J]. 国防科技大学学报, 2012, 34(6): 61 - 65.
CHENG Guo, CHEN Luo, WU Qiuyun, et al. A task scheduling method for parallelization of complicated geospatial raster data processing algorithms [J]. Journal of National University of Defense Technology, 2012, 34(6): 61 - 65. (in Chinese)

[5] Hu H D, Shu H. An improved coarse-grained parallel algorithm for computational acceleration of ordinary Kriging interpolation [J]. Computers & Geosciences, 2015, 78: 44 - 52.

[6] Cheng T P, Li D D, Wang Q. On parallelizing universal Kriging interpolation based on OpenMP [C]//Proceedings of the 9th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, Hong Kong: IEEE, 2010: 36 - 39.

(下转第 148 页)

YANG Xianhuan. Research of software-defect localization based on function calling sequence mining[D]. Wuhan: Huazhong University of Science and Technology, 2013. (in Chinese)

[11] Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software [C]//Proceedings of the Network and Distributed System Security Symposium,2005.

[12] Clause J, Li W C, Orso A. Dytan: a generic dynamic taint analysis framework[C]//Proceedings of the 2007 international symposium on Software testing and analysis. ACM, 2007: 196 – 206.

[13] Kang M G, McCamant S, Poosankam P,et al. DTA + +: dynamic taint analysis with targeted control-flow propagation [C]// Proceedings of the 18th Network and Distributed System Security Symposium, 2011.

[14] Bosman E, Slowinska A, Bos H. Minemu: the world's fastest taint tracker[C]//Proceedings of Recent Advances in Intrusion Detection. Springer Berlin Heidelberg, 2011,6961: 1 – 20.

[15] Kemerlis V P, Portokalidis G, Jee K, et al. libdft: practical dynamic data flow tracking for commodity systems [C]// Proceedings of ACM SIGPLAN Notices, 2012, 47 (7): 121 – 132.

[16] DynamoRIO[EB/OL]. http://www.dynamorio.org/.

[17] Valgrind[EB/OL]. http://valgrind.org/.

[18] Intel. Pin-a dynamic binary instrumentation tool[EB/OL]. https://software.intel.com/en-us/articles/pin-a-dynamic-binaryinstrumentation-tool.

[19] Hazelwood K, Klauser A. A dynamic binary instrumentation engine for the ARM architecture [C]//Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems, 2006: 261 – 270.

[20] Common vulnerabilities and exposures[EB/OL]. https://cve.mitre.org/.

[21] Wikipedia. Open source vulnerability database[EB/OL]. http://en.wikipedia.org/wiki/Open_Source_Vulnerability_Database.

[22] Offensive Security[EB/OL]. The Exploit Database. http://www.exploit-db.com/.

(上接第39页)

[7] Cheng T P. Accelerating universal Kriging interpolation algorithm using CUDA-enabled GPU [J]. Computers & Geosciences, 2013, 54: 178 – 183. [8] Shi X, Ye F. Kriging interpolation over heterogeneous computer architectures and systems[J]. GIScience & Remote Sensing, 2013,50(2):196 – 211.

[9] 卢风顺, 宋君强, 银福康, 等. CPU/GPU 协同并行计算研究综述[J]. 计算机科学, 2011, 38(3): 5 – 9.
LU Fengshun, SONG Junqiang, YIN Fukang, et al. Survey of CPU/GPU synergetic parallel computing [J]. Computer Science, 2011,38(3): 5 – 9. (in Chinese)

[10] 方留杨, 王密, 李德仁, 等. 负载分配的 CPU/GPU 高分辨率卫星影像调制传递补偿方法[J]. 测绘学报, 2014, 43(6): 598 – 606.
FANG Liuyang, WANG Mi, LI Deren, et al. A workload-distribution based CPU/GPU MTF compensation approach for high resolution satellite images [J]. Acta Geodaetica et Cartographica Sinica, 2014, 43(6): 598 – 606. (in Chinese)

[11] Wang S, Armstrong M P. A theoretical approach to the use of cyberinfrastructure in geographical analysis [J]. International Journal of Geographical Information Science, 2009, 23(2): 169 – 193.

[12] Dong B, Li X, Wu Q M, et al. A dynamic and adaptive load balancing strategy for parallel file system with large-scale I/O servers [J]. Journal of Parallel and Distributed Computing, 2012, 72(10):1254 – 1268.

[13] 夏飞, 朱强华, 金国庆. 基于 CPU-GPU 混合计算平台的 RNA 二级结构预测算法并行化研究[J]. 国防科技大学学报, 2013, 35(6): 138 – 146.
XIA Fei, ZHU Qianghua, JIN Guoqing. Accelerating RNA secondary structure prediction applications based on CPU-GPU hybrid platforms [J]. Journal of National University of Defense Technology, 2013,35(6):138 – 146. (in Chinese)

[14] 赵斯思, 周成虎. GPU 加速的多边形叠加分析[J]. 地理科学进展, 2013, 32(1): 114 – 120.
ZHAO Sisi, ZHOU Chenghu. Accelerating polygon overlay analysis by GPU[J]. Progress in Geography, 2013, 32(1): 114 – 120. (in Chinese)

[15] 马安国, 成玉, 唐遇星, 等. GPU 异构系统中的存储层次和负载均衡策略研究[J]. 国防科技大学学报, 2009, 31(5): 38 – 43.
MA Anguo, CHENG Yu, TANG Yuxing, et al. Research on memory hierarchy and load balance strategy in heterogeneous system based on GPU [J]. Journal of National University of Defense Technology, 2009, 31(5): 38 – 43. (in Chinese)