International Journal of Geo-Information

MDPI

Article

A CUDA-Based Parallel Geographically Weighted Regression for Large-Scale Geographic Data

Dongchao Wang [1], Yi Yang [1,*], Agen Qiu [2], Xiaochen Kang [2], Jiakuan Han [1] and Zhengyuan Chai [1]

[1] School of Geomatics and Marine Information, Jiangsu Ocean University, Lianyungang 222005, China; wangdongchao@jou.edu.cn (D.W.); hanjk@jou.edu.cn (J.H.); chaizhengyuan@jou.edu.cn (Z.C.)

[2] Research Center of Government GIS, Chinese Academy of Surveying and Mapping, Beijing 100039, China; qiuag@casm.ac.cn (A.Q.); kangxc@casm.ac.cn (X.K.)

* Correspondence: yangyi@jou.edu.cn

Received: 3 September 2020; Accepted: 26 October 2020; Published: 30 October 2020

Abstract: Geographically weighted regression (GWR) introduces the distance weighted kernel function to examine the non-stationarity of geographical phenomena and improve the performance of global regression. However, GWR calibration becomes critical when using a serial computing mode to process large volumes of data. To address this problem, an improved approach based on the compute unified device architecture (CUDA) parallel architecture fast-parallel-GWR (FPGWR) is proposed in this paper to efficiently handle the computational demands of performing GWR over millions of data points. FPGWR is capable of decomposing the serial process into parallel atomic modules and optimizing the memory usage. To verify the computing capability of FPGWR, we designed simulation datasets and performed corresponding testing experiments. We also compared the performance of FPGWR and other GWR software packages using open datasets. The results show that the runtime of FPGWR is negatively correlated with the CUDA core number, and the calculation efficiency of FPGWR achieves a rate of thousands or even tens of thousands times faster than the traditional GWR algorithms. FPGWR provides an effective tool for exploring spatial heterogeneity for large-scale geographic data (geodata).

Keywords: CUDA; GWR; parallel computation; large-scale geodata

1. Introduction

Large-scale geodata is currently a topic of considerable attention in many research fields, including mobile communication [1], public transportation [2], medical health [3], Earth observation [4], and climate monitoring [5]. To enhance the capability of analyzing massive geodata, geographic knowledge mining is turning to data-driven patterns [6]. Distributed system and parallel computing are two feasible technologies to solve the problem of massive geodata analysis. A tremendous amount of multisource geodata is stored in a distributed spatial index system [7], enabling people to access records efficiently. Using the advantages of the distributed system Hadoop, Aji et al. (2019) [8] proposed a scalable high-performance spatial data warehousing system (Hadoop-GIS) that can meet the needs of managing and querying massive geodata. Furthermore, based on the MapReduce parallel computing framework and the HadoopBase database (HBase) technology, the origin–destination (OD) estimation method [9] can efficiently manage massive bus travel data and directly reckon the origin and destinations of travel for bus passenger. In the parallel computing field, large-scale geodata could be parallelize into multiple data pieces utilizing the strategies of multiple instruction multiple data (MIMD) and single instruction multiple data (SIMD). MIMD handles multiple instructions simultaneously in opposition to SIMD. There are several environments to parallelize multiple tasks

based on different strategies (SIMD, MIMD), such as a message-passing interface (MPI), a multi-core CPU, and a many-core shared-memory graphics processing unit (GPU). MPI is mainly used to standardize the communication protocol of multi-program cluster, multi-core CPU relies on the computing power of CPU core, and many-core shared-memory GPU benefits from numerous stream processors (SP). Wilkinson et al. (1999) [10] introduce parallel programming techniques and how to solve problems at a greater computational speed than is possible with a single computer. Gong et al. (2013) [11] proposes a parallel approach that leverages the power of multicore systems, to cope with the computational complexity of agent-based models (ABMs), and it solves the space-time complexity of a geographic system. Tang et al. (2015) [12] and Zhang et al. (2017) [13] explored the feasibility of using GPU to carry out the massively parallel spatial computing and accelerate the spatial point pattern analysis. Sandric et al. (2019) [14] undertook parallelization for certain GIS features operations using their message-passing interface–GIS (MPI-GIS) system, which integrated the advantages of MPI input/output (I/O) and GPU on a cluster of nodes. Stojanovic et al. (2019) [15] proposed an algorithm to analyze with watershed approach, called multiple flow direction (MFD), which was designed for multicore CPU or many-core GPU. Amazing progress has been achieved in the fields of computer hardware and software, which lay a solid foundation for updating geographical research tools. However, there is still a sizable problem to be solved: how existing geographic analysis tools can be transformed to accommodate the development of big geodata mining [16]?

Spatial non-stationarity analysis is an important research field of spatial data mining. Brunsdon et al. (1996) [17] proposed the effective tool (GWR model) to explore spatial non-stationarity. GWR introduces the idea of local smoothness to calibrate the regression coefficients and detect spatial non-stationarity in the geographic space. The expansion of the location factor upgrades GWR from ordinary linear regression (OLR) model to a local regression model. The locally weighted least squares (LWLS) method is used to estimate the parameters point by point, where the weight refers to the distance kernel function of some point against each observation points. The results of parameter estimation from GWR are both clearly interpretable and statistically verifiable; therefore, GWR has become a major method for studying spatial heterogeneity. Zhang et al. (2020) [18] employed GWR to identify the driving forces of wastewater discharge between provinces in China and discovered that the macro industry policy and environmental protection measures were major reasons for its spatial changes. Wu (2020) [19] explored the influencing factors that cause spatially and temporally varying distributions of ecological footprints using GWR. Yuan et al. (2020) [20] applied GWR to reveal the spatially varying relationships in environmental variables (Pb and Al) and suggested that GWR was more effective than conventional statistical analysis tools. Hong et al. (2020) [21] researched the spatially heterogeneous relationship between price and pricing variables using multiscale geographically weighted regression (MGWR), in which it overcame the limitations of hedonic pricing model research for sharing economy accommodation. Wu et al. (2020) [22] developed a geographically and temporally neural network weighted regression (GTNNWR) model that was extended from the spatiotemporal proximity neural network (STPNN), which not only exhibited a better prediction performance but also more accurately quantified the distribution of spatiotemporal heterogeneity.

Typically, parallelization of geographic analysis tools has become a comprehensive subject across computer field and geography science. The package spgwr [23] was developed to implement GWR in the R language. Another R package (GWmodel) [24] optimized this model with a moving window weighting technique and achieved slightly better efficiency against spgwr. The Python-based implementation (mgwr [25]) of MGWR was developed for multiscale analysis that allowed varying relationships according to each coefficient. Li et al. (2019) [26] (a member of the mgwr package) upgraded its mode to distributed parallelization utilized within a high-performance computing (HPC) environment and the new package (FastGWR) achieved satisfactory results. Tran et al. (2016) [27] studied the implementation of large-scale GWR on an in-memory cluster computing framework Spark (Spark-GWR) and determined that it was a feasible solution using cluster computers to execute GWR in parallel, but great difficulty is encountered for ordinary coders in developing and testing under

the cluster environment. As a representative model of local regression, GWR incorporates all of the observations (samples) into the loop of the regression sequence. The key to geographic weighting is the calculation of distance weights for each sample, where it causes costly complexity in terms of runtime and memory. At the same time, the entire process consumes a large amount of computing time because the weight calibrator participates in multilayer loops. Under the condition of large-scale geodata, GWR needs to go through two levels of large cycle iteration, the outer iteration is responsible for point by point regression, and the inner iteration is used for matrix calculation between single sample and full samples. Therefore, limited by data structure and operating mode, GWR is less effective in addressing large-scale geodata. Concurrency methods can improve the efficiency of geographic analysis tools depending on the software optimization, but the hardware parallel environment could obtain native support and achieve the best acceleration performance. Both FastGWR and Spark-GWR could divide GWR into several parallel task sets, and the two parallel programs are designed for CPU architecture that cannot be adapted to GPU architecture. FPGWR decomposes large-scale GWR into simpler parallelizable computing units utilizing atomization algorithm and processes them with numerous parallel GPU cores.

In this paper, we develop FPGWR to reduce the computational complexity in the GWR process and enable GWR's applications in millions or even tens of millions of geodata. This technique significantly improves the efficiency in regression when utilizing the parallelization of large tasks. On the basis of the CUDA framework, atomic subtasks that are decomposed from large tasks could run on a GPU device in parallel mode. This paper contributes to the prior literature as follows. (1) FPGWR can compensate for the deficiencies of GWR in undertaking regression computation for large-scale geodata, and FPGWR with separate atomic computing units (atomization) is more efficient than GWR. (2) FPGWR is a powerful model for exploring spatial heterogeneity and incorporating high parallelism into geography analysis, which is applicable for studies in various fields, such as economic geography, social science, public health. (3) The improvement from GWR to FPGWR can provide new insights into geospatial computing from spatial and computational perspective.

2. GWR Model and Atomization Algorithm

2.1. GWR Review

Before the 1980s, OLR was frequently applied for geographical phenomena analysis. The predictive coefficients $\hat{\beta}$, calculated by the ordinary least squares (OLS) estimator method, abides by the rule of global optimal unbiased estimation. The final regression result merely reflects the average level in the study region. It is illegitimate to utilize the global regression methods in the local regression model. Therefore, Foster et al. (1986) [28] created a spatial adaptive filter (SAF) learning from varying coefficient modeling, which could describe step-jump and continuous spatial non-stationarity in the coefficients automatically. Based on the local polynomial smoothing technique, Brunsdon et al. (1996) [17] proposed the analysis tool of GWR.

2.1.1. GWR Model

The GWR model extends OLR, introducing the location factor to express the spatial variation of coefficients. In other words, we have the following:

$$y_i = \beta_0(u_i, v_i) + \sum_{m=1}^{p} \beta_m(u_i, v_i) x_{im} + \varepsilon_i i = 1, 2, \cdots, n \quad (1)$$

where y_i is the regression variable (dependent variable) at location i, (u_i, v_i) represents the coordinate (usually latitude and longitude) of the ith sample point in the study area, $\beta_m(u_i, v_i)$ denotes the kth coefficient of the ith sample point based on a function with independent variables of u_i and v_i, x_{im}

expresses the mth predictor variable (independent variable), and ε_i represents the error term, and n is the sample size. The necessary conditions for Equation (2) can be expressed as follows:

$$\varepsilon_i \sim N\left(0, \sigma^2\right) \cap Cov\left(\varepsilon_i, \varepsilon_j\right) = 0 (i \neq j) \tag{2}$$

For simplicity, Equation (1) is abbreviated as

$$y_i = \sum_{m=0}^{p} \beta_{im} x_{im} + \varepsilon_i \; i = 1, 2, \cdots, n \cap x_{i0} \equiv 1 \tag{3}$$

To prevent GWR from degenerating into a general linear regression, it is necessary that $\beta_{1m} = \beta_{2m} = \cdots = \beta_{nm}$ should not appear in the preconditions.

The variables related to GWR can be defined in the form of matrix. The independent variable matrix X can be calculated by the following form:

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \tag{4}$$

2.1.2. Spatial Weight Kernel Function

There are n terms of spatial weight w_{ij} between two sample points ($i = j$ is allowed) in the study area. In the GWR model, it is usual to denote the weight matrix W_i as a diagonal square matrix:

$$W_i = \begin{bmatrix} w_{i1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & w_{in} \end{bmatrix} \tag{5}$$

At present, there are several forms of the weight kernel function w_{ij}, and the most used are Bi-Square and Gaussian. The two functions can be expressed as Equations (6) and (7):

$$\text{Bi} - \text{Square}: \; w_{ij} = \begin{cases} \left[1 - \left(\dfrac{d_{ij}}{bw}\right)^2\right]^2 & ,d_{ij} < bw \\ 0 & ,d_{ij} \geq bw \end{cases} \tag{6}$$

$$\text{Gaussian}: \; w_{ij} = e^{-1/2\left(\frac{d_{ij}}{bw}\right)^2} \tag{7}$$

where d_{ij} represents the distance between two sample points (i and j), and bw denotes the bandwidth parameter which could be interpreted as not only the neighbors threshold but also the distance attenuation factor within the weight kernel function.

2.1.3. Model Regression

The regression coefficient estimate $\hat{\beta}_i$ at position i is defined by

$$\hat{\beta}_i = \left(X^T W_i X\right)^{-1} X^T W_i Y \tag{8}$$

The regression value $\hat{Y}_i$ of the regression point i based on $\hat{\beta}_i$ can be estimated from

$$\hat{Y}_i = X_i \left(X^T W_i X\right)^{-1} X^T W_i Y \tag{9}$$

where X_i represents the ith row vector in matrix X. The hat matrix plays a very important role in the residual analysis of the linear regression model. This study introduces the hat matrix S into GWR. The matrix S can be expressed as follows:

$$S_i = X_i\left(X^T W_i X\right)^{-1} X^T W_i \tag{10}$$

The regression result matrix $\hat{Y}_i$ can be represented with the hat matrix S:

$$\hat{Y} = SY = \begin{bmatrix} \hat{Y}_1 \\ \vdots \\ \hat{Y}_n \end{bmatrix} = \begin{bmatrix} S_1 \\ \vdots \\ S_n \end{bmatrix} Y \tag{11}$$

2.1.4. The Criteria of Optimal Bandwidth Selection

The key to discovering the optimal bandwidth bw is minimizing the AIC_c score. Loop selection and golden selection methods are available to obtain the lowest AIC_c value. Searching the optimal bandwidth bw is inseparable from the parameter estimation criterion. The criterion AIC_c [29] is introduced by Brunsdon et al. (2002) [30] to select the optimal bandwidth of the weight function. The specific formula can be expressed as

$$AIC_c = n\ln\left(\hat{\sigma}^2\right) + n\ln(2\pi) + n\left[\frac{n + tr(S)}{n - 2 - tr(S)}\right] \tag{12}$$

The residual ε can be calculated by the sample data Y and the regression result $\hat{Y}$:

$$\varepsilon = Y - \hat{Y} \tag{13}$$

The unbiased estimate of the random error variance is expressed as $\hat{\sigma}^2$:

$$\hat{\sigma}^2 = \frac{RSS}{n - 2tr(S) + tr(S^T S)} \tag{14}$$

where RSS indicates the sum of squared residuals, $tr(S)$ is the trace of the hat matrix S, and $n - 2tr(S) + tr\left(S^T S\right)$ represents the effective freedom degree of GWR. In most cases, $tr\left(S^T S\right)$ approximately equals $tr(S)$ ($tr\left(S^T S\right) \approx tr(S)$), and thereby, the above Equation (14) can be simplified as

$$\hat{\sigma}^2 = \frac{RSS}{n - tr(S)} \tag{15}$$

2.2. Atomizing the GWR Model

As mentioned above, the regression process of the GWR model involves two fixed steps: optimal bandwidth selection and model diagnosis. Most existing packages that have implemented the GWR algorithm are supported by the serial mode. Compared with the parallel mode, the serial mode carries undesirable consequences to the regression computation. The computing containers with noninfinite computational power will be overloaded with too large-scale samples. The runtime arises along with the sample size growth, following a power or even an exponential relationship [31]. In the paper, it is a feasible solution to design the Algorithm 1 (atomization) in reducing the complexity of GWR regression calculation.

Algorithm 1 Atomic Process—The Minimum Unit of Algorithms.

Atomic Process: Optimizing bandwidth searching by minimizing AIC score

1. Given test bandwidth (bw) and atomic process index (z)
2. Calculate w_{zz} $(w_{zz} \equiv 1)$ from Equation (7)
3. Loop each $a = 1, 2, \cdots, p+1,$ calculate :
4. Loop each $b = 1, 2, \cdots, p+1,$ calculate :
5. Set $B_{ab} = 0$
6. Loop each $i = 1, 2, \cdots, n,$ calculate :
7. $B_{ab} += x_{ia} \times w_{zi} \times x_{ib}$
8. End loop
9. End loop
10. End loop
11. Calculate B^{-1}
12. Set $S_z = 0, \hat{Y}_z = 0$
13. Loop each $a = 1, 2, \cdots, p+1,$ calculate :
14. Set temp_x_inv $= 0$
15. Loop each $b = 1, 2, \cdots, p+1,$ calculate :
16. temp_x_inv$+= x_{zb} \times \mathrm{B}^{-1}_{ba}$
17. End loop
18. $S_z +=$ temp_x_inv $\times x_{za} \times w_{zz}$
19. Set $temp\_x\_w = 0$
20. Loop each $i = 1, 2, \cdots, n,$ calculate :
21. $temp\_x\_w += x_{ia} \times w_{zi}$
22. End loop
23. $\hat{Y}_z +=$ temp_x_inv $\times\, temp\_x\_w$
24. End loop
25. Return $S_z, \hat{Y}_z$

2.2.1. Intermediate Matrix

In order to introduce the parallel mode legally, we design GWR atomization to decompose the matrix calculation process. The matrix elements used in the result are extracted on-demand to obtain the result value via simple algebraic calculations. It will save huge memory usage and computing resource occupy in the large matrix operation of GWR. Intermediate matrix is an important research object of GWR atomization, which exists in several common models.

OLR can be calculated by the following matrix form:

$$Y = X\beta + \varepsilon \tag{16}$$

On basis of OLS, regression coefficient $\hat{\beta}$ is estimated from

$$\hat{\beta} = \left(X^T X\right)^{-1} X^T Y \tag{17}$$

Next, regression result $\hat{Y}$ of OLR can be expressed as follows:

$$\hat{Y} = X\left(X^T X\right)^{-1} X^T Y \tag{18}$$

By comprehensively analyzing Equations (8), (9), (17), and (18), we can find the intermediate matrix M which exists in all regression models of estimating unbiased via OLS. It can be calculated by the following:

$$M = \left(X^T W_i X\right)^{-1} X^T W_i \quad or \quad M = \left(X^T X\right)^{-1} X^T \tag{19}$$

In the point-by-point regression process, the intermediate matrix M is inevitable. Matrix X^T can be defined by

$$X^T = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_{11} & x_{21} & \cdots & x_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1p} & x_{2p} & \cdots & x_{np} \end{bmatrix}. \tag{20}$$

where p is the number of independent variables. The multiplication of matrix X^T and the diagonal square matrix W_i is special. The resulting matrix A can be expressed as follows:

$$A = X^T W_i = \begin{bmatrix} 1 \times w_{i1} & 1 \times w_{i2} & \cdots & 1 \times w_{in} \\ x_{11} \times w_{i1} & x_{21} \times w_{i2} & \cdots & x_{n1} \times w_{in} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1p} \times w_{i1} & x_{2p} \times w_{i2} & \cdots & x_{np} \times w_{in} \end{bmatrix} \tag{21}$$

Similarly, matrix B can be written as

$$B = X^T W_i X = \left[\sum_{a=1}^{p+1} \left(\sum_{b=1}^{p+1} \left(\sum_{j=1}^{n} \left(x_{ja} \times w_{zj} \times x_{jb} \right) \right) \right) \right]_{(p+1)\times(p+1)} \tag{22}$$

where matrix B is a square matrix with $p+1$ dimensions. In practical applications, $p+1$ is usually less than *10*, which means that it is legal to ignore the time spent by the inverse operation for matrix B.

Comparing with matrix decomposition, the regression subprocess of GWR relies on the weight matrix W_i when calculating matrices A and B. The determination of weighting scheme W_i could be achieved: *(a)* Obtain the coordinate matrix $UV_{(n\times 2)}$ of all samples, and then transpose the matrix to matrix $UV^T_{(2\times n)}$. *(b)* Solve the distance matrix $D_{(n\times n)}$ between coordinate matrix $UV_{(n\times 2)}$ and its transposed matrix $UV^T_{(2\times n)}$. *(c)* Calculate the weight matrix $W_{(n\times n)}$ of all samples according to Equation (6) or (7), and then W_i is the diagonal matrix formed by the ith row elements of the weight matrix W. However, the process needs huge memory space and calculation time when involving the enormous sample size. In addition, each subprocess of GWR will determine W_i once, which causes high redundancy of memory and runtime. The implementation of matrix decomposition approach has been carried out to decrease memory usage and runtime occupation by means of Equations (21) and (22).

2.2.2. Implementation of the Atomization Algorithm

Unlike the large process with full-matrix multiplication, each logically independent subprocess merely participates in the regression calculation once, on the basis of the atomization algorithm. It is the prerequisite for parallelization to ensure that the subprocess is repeatable. To address the problems caused by redundant computing, two aspects (memory and time) of optimization are conducted in the study. The AIC_c scores and estimation $\hat{Y}$, generated during the bandwidth calibration process, are stored in the singleton pattern. Moreover, by means of on-demand computing, the disadvantage of a high-repetition-rate calculation is eliminated in the large process. Given a test bandwidth bw, the detailed steps of the atomization algorithm can be implemented as Algorithm 1.

3. CUDA Enabled FPGWR

FPGWR based on CUDA has the capability to process massive spatial geodata. The technique is substantially developed to increase the computing speed of GWR. Supported by a large number of SP, the GPU device can handle parallel computing as a natural carrier of HPC. Hardware performs

superior to software in terms of the multithread scheduling. Hence, it is the preferred solution to improve GWR on the basis of the CUDA framework.

3.1. Optimizing the Kernel Function of CUDA

CUDA is a general-purpose parallel computing architecture introduced by the NVIDIA Corporation [32]. In the CUDA framework, parallel tasks would be instantiated as independent controllable threads. Independence means that there are no mutually exclusive signals among all threads. Each thread could run synchronously without depending on its sibling threads. Controllability means that the specificity of the thread instances could be controlled by the same parameters. The initialization values are differently set to make the generated instances diverse from each other. Due to the identical computing processes of threads, merely one thread scheduler is needed to manage all threads.

There are two principles for designing the CUDA kernel function to maximize the usage of the GPU scheduling resource and computing cycle. We should minimize the occurrence of WARP Branch in the kernel function as much as possible. At the same time, it is recommended to choose the CUDA memory type flexibly. The specific optimization strategy is shown in Figure 1. By the method of matrix decomposition, the atomic kernel function has successfully prevented process branching. Hence the computation workload can be evened out among the threads. Each atomic task will dynamically be assigned one unique thread index (z) that is different from the others. Because the tasks execute in a completely random order, the coupling relationship between the atomic threads and SPs is disconnected. To overcome the performance bottleneck caused by frequent access to global memory, FPGWR utilizes the shared memory to store these temporary variables.

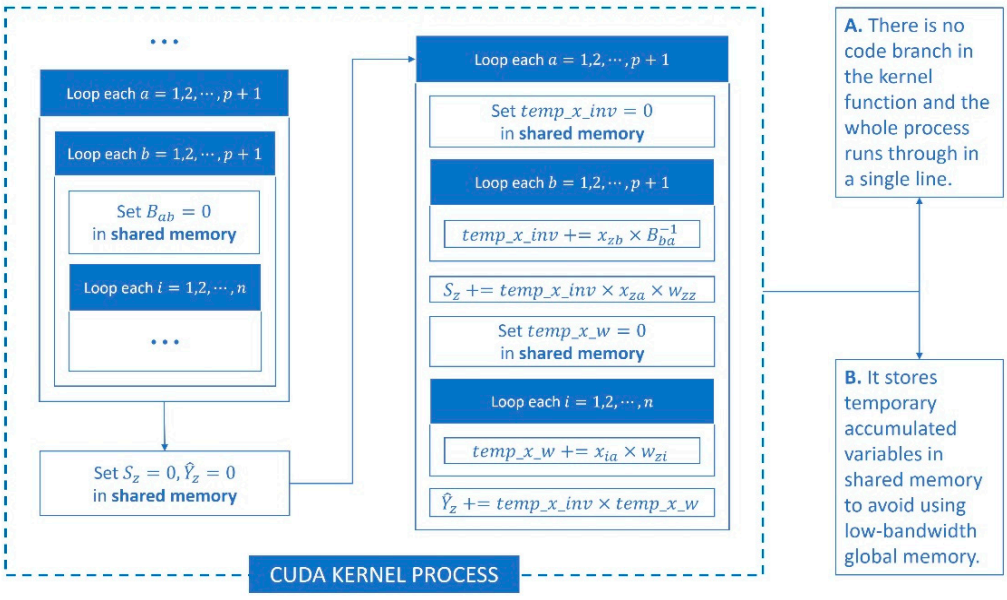
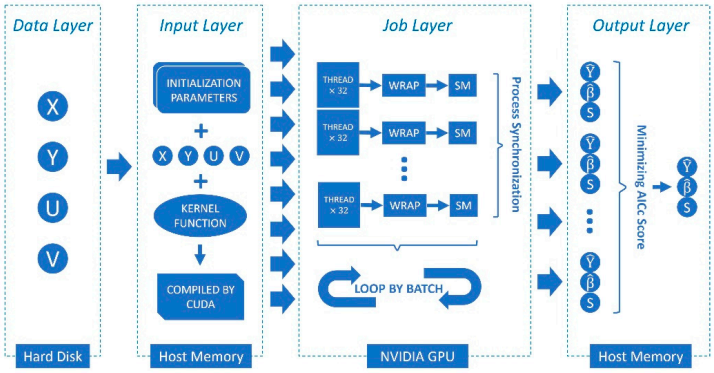


Figure 1. Optimizing the compute unified device architecture (CUDA) kernel function.

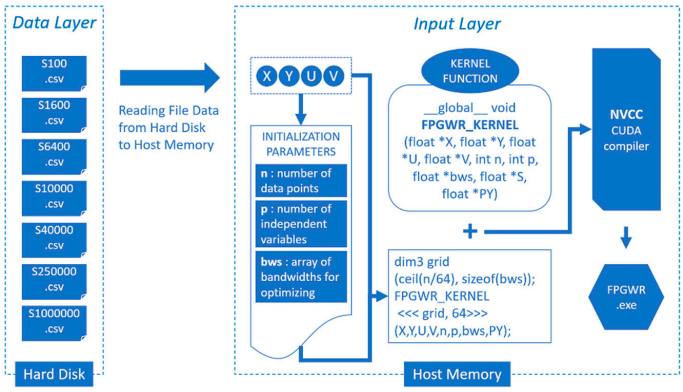
3.2. Implementing FPGWR Based on CUDA

In this study, we have implemented FPGWR in a CUDA framework by utilizing the method of atomization. FPGWR significantly shortens the total time of large-scale GWR regression and releases the memory space of massive spatial matrix data. The FPGWR implementation consists of five steps. Step 1, the program in Host device invokes the GPU device to be prepared, and at the same time, a series of initial parameters are set in the constant memory of the GPU. Step 2, the sample data are input to the global memory of the GPU. The volume of geodata is too enormous to be instantiated in either the shared memory or the local memory. It will throw an "out of memory (OOM)" error when sample data volume is too large to fit in GPU global memory. Step 3, CUDA loads the instructions

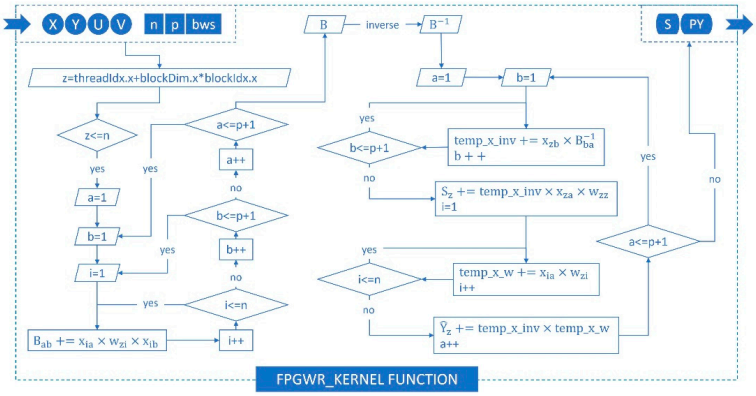
compiled from the code of the atomic kernel function, and then, the scheduler generates individual threads with the same kernel function. Step 4, all threads are assigned to Streaming Multiprocessor (SM) in the unit of WARP. To address the enormous number of threads, the GPU will activate the flow-shop scheduling mode. Step 5, CUDA feeds back the regression results from the GPU to the Host, and the GPU device resources are released immediately. The detailed workflow of the FPGWR implementation is shown in Figure 2.



(**a**) The flow overview of fast-parallel geographically weighted regression (FPGWR).



(**b**) Detailed process on data and input layers.



(**c**) core implementation process of the fpgwr_kernel function.

Figure 2. Flow diagram of FPGWR on CUDA (**a**–**c**).

As shown in Figure 2a, the FPGWR algorithm could be divided into four layers: data layer, input layer, working layer, and output layer. The data layer is dedicated to storing the files of original observations. The input layer reads the spatial observation data from hard disk into host memory.

At the same time, the part of the CUDA programming is instantiated in this layer. The initialization parameters and observation matrices are introduced together into the atomic kernel function, and then, the function will be compiled into an executable program. The working layer runs on the NVIDIA GPU. It starts massive task threads, which are managed uniformly by the multithreaded scheduler of the GPU. At the physical level, WARPs are bundled into a queue of batches, while the WARPs in the same batch are executed synchronously. The output layer is designed to collect the regression results. Based on the bandwidth indexes, these results are organized into multiple sets of regression matrices (S, $\hat{Y}$, and $\hat{\beta}$). Finally, the algorithm finds the optimal results set that corresponds to the minimum AIC_c score.

Figure 2b,c illustrate how the core part of FPGWR works at the micro level. The specific meanings of the initialization parameters (n, p and bws) and the prototype of the FPGWR_KERNEL function are described in Subfigure (b). The detailed process of FPGWR_KERNEL function is presented in Subfigure (c). The steps of the process could correspond to those of Algorithm 1. The multithread scheduling depends on the initial BLOCK and GRID settings of the kernel function in CUDA. BLOCK is set as a one-dimensional vector with a constant value (64), namely, each BLOCK contains 64 threads. GRID is set as a two-dimensional vector, in which the number of the first dimension is the sample size n divided by 64 (number of BLOCK's first dimension), and the second dimension is the size of the bandwidth array.

4. Results and Discussion

4.1. Data Source

To explore the real performance of FPGWR, three data sources—the simulation dataset, the "Zillow test dataset" [26], and the "Georgia" dataset [33]—are used for the experiment. The simulation dataset is designed to evaluate the influences caused by the sample size and the independent variables size. The "Zillow test dataset" (https://github.com/Ziqi-Li/FastGWR) is assigned to compare the acceleration performance of the different GWR packages. The "Georgia" dataset is used to validate the result accuracy of FPGWR against other schemes.

4.1.1. Simulation Dataset

The test region is displayed as a square area [34] with l length sides, where the sample points are distributed evenly. After setting the sample size of each row to c, the total number of samples could be expressed as $n = c \times c$. The distance between two adjacent samples is calculated by $\Delta l = l/(c-1)$. The lower-left corner is defined as the origin of the coordinate system. The expression for calculating the positions of the samples is given by

$$(u_i, v_i) = \left(\Delta l \times mod\left(\frac{i-1}{c}\right), \Delta l \times floor\left(\frac{i-1}{c}\right)\right) \tag{23}$$

where *mod* stands for the remainder function, and *floor* denotes the rounding function.

The sample data are generated by the GWR model below. It is predefined in Equation (24) as follows:

$$y_i = \beta_0(u_i, v_i) + \beta_1(u_i, v_i)x_{i1} + \beta_2(u_i, v_i)x_{i2} + \beta_3(u_i, v_i)x_{i3} + \beta_4(u_i, v_i)x_{i4} + \varepsilon_i \tag{24}$$

To unify the dimensions of the regression coefficients β, all of the values are limited to the interval $(0, \beta_{max})$ (β_{max} is a fixed constant). The coefficients β follow 5 functions as follows:

$$\beta_0(u_i, v_i) = \frac{2\beta_{max}}{l^2}\left(\frac{l^2}{2} - (l-u_i)^2 - (l-v_i)^2\right) \tag{25}$$

$$\beta_1(u_i, v_i) = \frac{\beta_{max}}{2}\left(\left(\sin\frac{u_i\pi}{l}\right)^2 + \left(\sin\frac{v_i\pi}{l}\right)^2\right) \tag{26}$$

$$\beta_2(u_i, v_i) = \frac{\beta_{max}}{2}\left(2 - \left(\left(\tan\left(\frac{u_i\pi}{2l} - \frac{\pi}{4}\right)\right)^2 + \left(\tan\left(\frac{v_i\pi}{2l} - \frac{\pi}{4}\right)\right)^2\right)\right) \tag{27}$$

$$\beta_3(u_i, v_i) = \beta_{max} e^{-\frac{1}{2l}\left(\left(\frac{l}{2} - u_i\right)^2 + \left(\frac{l}{2} - v_i\right)^2\right)} \tag{28}$$

$$\beta_4(u_i, v_i) = \frac{16\beta_{max}}{l^4}\left(\frac{l^2}{4} - \left(\frac{l}{2} - u_i\right)^2\right)\left(\frac{l^2}{4} - \left(\frac{l}{2} - v_i\right)^2\right) \tag{29}$$

The spatial distribution of the coefficients β is displayed in Figure 3. The five coefficients β selected by the model are closely related to the position of the sample, which demonstrates the spatial non-stationarity of the observations.

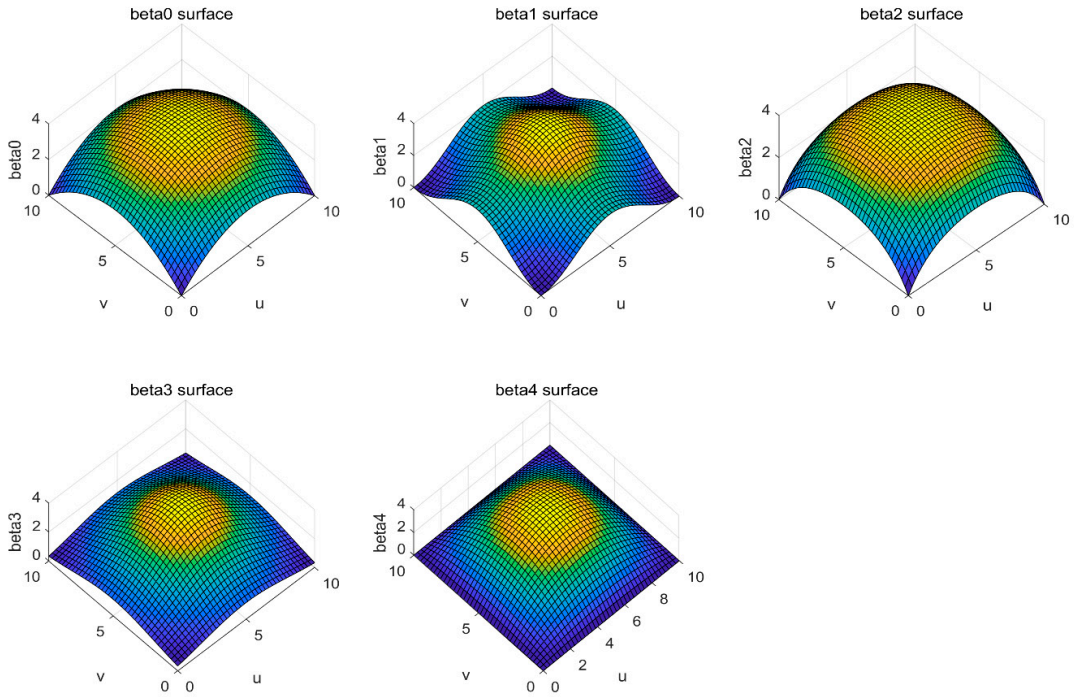


Figure 3. Coefficient (β_0, β_1, β_2, β_3 and β_4) surface.

According to Formula (27), eight sets of sample datasets are produced for testing. The datasets' construction parameters and resource link are exhibited in Table 1.

Table 1. Construction parameters of example data and access entrance of resource.

l	c	β_{max}	x_{max}	σ	Number of Data Points
10	10	4	2	0.5	100
10	40	4	2	0.5	1600
10	80	4	2	0.5	6400
10	100	4	2	0.5	10,000
10	200	4	2	0.5	40,000
10	500	4	2	0.5	250,000
10	1000	4	2	0.5	1,000,000
10	2000	4	2	0.5	4,000,000

Note: Resource URL: https://pan.baidu.com/s/1c0Ga8Ngej0SG990sdxHQ_A. Access code: 8dve.

4.1.2. Zillow Test Dataset

The "Zillow test dataset" [26] is a subset of the Zillow property dataset, which consists of the single-family housing information within the metropolitan area of Los Angeles. The mathematical expression of the dataset is expressed as Equation (30). The dataset is open source on GitHub along with the FastGWR algorithm (https://github.com/Ziqi-Li/FastGWR). This paper has downloaded eight datasets (1 k, 2 k, 5 k, 10 k, 15 k, 20 k, 50 k and 100 k) from the GitHub repository for the comparative experiment.

$$Value_i = \beta_{i0} + \beta_{i1}Area_i + \beta_{i2}Nbaths_i + \beta_{i3}Nbeds_i + \beta_{i4}Age_i + \varepsilon_i \quad (30)$$

4.1.3. Georgia Dataset

The Georgia dataset [33] contains a subset (socio-demographic characteristics) of the 1990 US census within the state of Georgia. The coordinates of the data points are set at the centroids of counties, so there are 159 records containing county population attributes in the dataset. The model could be defined as Equation (31):

$$PctBach = \beta_0 + \beta_1 Intercept + \beta_2 PctPov + \beta_3 PctRural + \beta_4 PctBlack + \varepsilon \quad (31)$$

4.2. Testing Specifications and Environment

The experiment for FPGWR is conducted on a desktop computer. The configuration of this computer is an Intel i7-9700K 3.60 GHz 8-core CPU (Intel Corporation, Santa Clara, CA, USA), 16 GB Random Access Memory (RAM) (Kingston Technology Corporation, Fountain Valley, CA, USA) and NVIDIA GeForce RTX 2080 Ti 11 GB GPU (NVIDIA Corporation, San Tomas Expressway Santa Clara, CA, USA). In addition, it has installed version 10.2.95 of the CUDA development kit, version 14.0.25431.01 Update 3 of Microsoft Visual Studio 2015 (development IDE), and the Microsoft Windows 10.0.17134 Professional Edition operating system (OS). Note that 4352 SP core units are placed in the GPU device. Relying on its ultra-high-speed task scheduling capability, the GPU can withstand the pressure of multithreaded computation tasks in parallel.

4.3. Results

4.3.1. FPGWR Performance

Due to the transformation of parallelization, the regression efficiency of FPGWR increases dramatically against GWR. The bandwidth optimization is clearly a repetitive process. To compare the acceleration performances, this study analyzes only the single subprocess with a fixed bandwidth. The runtimes of FPGWR with different sample sizes are shown in Table 2. Given four independent variables, the runtime can be controlled within 2 s when the sample size is less than 40 k. After increasing the sample size to 250 k, the runtime becomes approximately 66.6 s. As the sample size increases to the millions scale, it spends only approximately 1094.7 s. The result shows that the runtime obeys a logarithmic variation rule as the sample size changes.

The runtimes vary tremendously with different sample sizes. To enable the display of all results together, the y-axis of the logarithmic scale is plotted in Figure 4. The comprehensive analysis of Table 2 and Figure 4 reveals that both the sample size and the number of independent variables can influence the variation of runtime. The regression time has positive association with the number of independent variables. When the sample size varies, the variation in the runtime is similar with different numbers of independent variables. Given the same sample size, the results on the time exhibit a simple multiple relationship among the different numbers of independent variables. In summary, the sample size has a more pronounced impact than the number of independent variables on the regression time.

Table 2. Runtime (in seconds) for different numbers of coefficients using different numbers of data points.

Number of Data Points	Four Independent Variables	Three Independent Variables	Two Independent Variables
100	0.003	0.002	0.001
1600	0.022	0.017	0.011
6400	0.095	0.068	0.045
10,000	0.186	0.126	0.063
40,000	1.867	1.256	0.766
250,000	66.616	46.386	26.154
1,000,000	1094.654	738.636	421.922

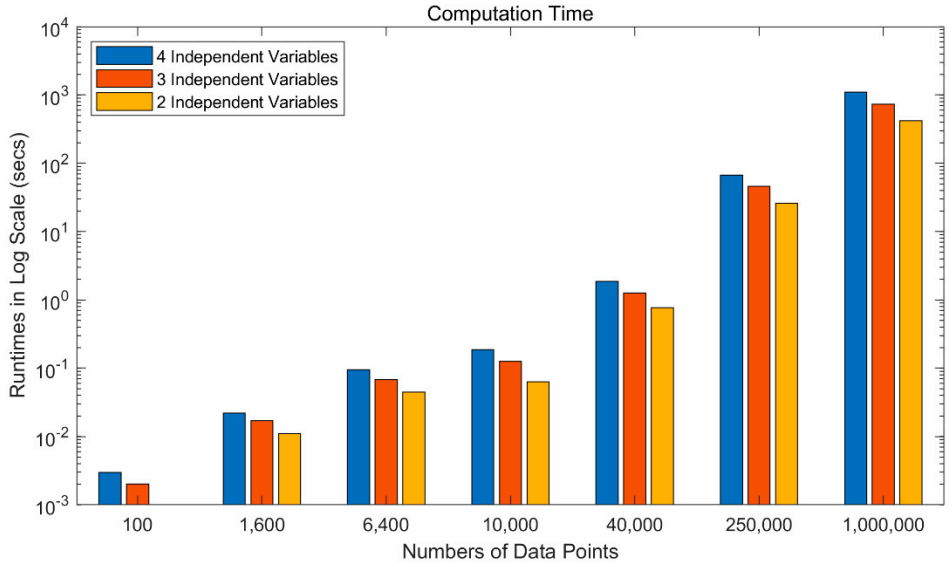


Figure 4. Runtime comparison (bar) for different numbers of coefficients using different numbers of data points.

Speed-up and efficiency are important metrics for the performance check of parallel algorithms [35]. Speed-up refers to the ratio of single processor runtime to multiprocessor runtime, and efficiency represents the average of speed-up in multiprocessors [36]. The speed-up of FPGWR relies on the GPU performance, which consists of SP number, base clock frequency, and memory bandwidth. Table 3 compares the performance configuration of different type GPUs. On basis of the 250,000 simulation samples, this study regresses the model given in equation (24) with an increasing number of GPU cores. The runtime of GTX1050 is set as the benchmark value to calculate the speed-up factor of GPUs with different SP numbers. The speed-ups growth proves that FPGWR has an outstanding parallel scalability. Figure 5 illustrates that the computation time decreases approximate-linearly as the number of GPU cores increases. It exhibits an obvious positive linear relationship between efficiency and GPU cores.

The experimental result demonstrates the outstanding capability of FPGWR to accelerate GWR, although its performance varies slightly among different orders of magnitude of observations. By setting appropriate sample sizes and independent variable numbers, the full potential of FPGWR can be achieved in various fields.

Table 3. Performance comparison for different types of graphics processing unit (GPU).

Type of NVIDIA GPU	SP Number	Base Clock Frequency (MHz)	Memory Bandwidth (GB/s)	Runtime (ms)	Speed-Up Factor
GTX 1050	640	1354	84	568,086	1
GTX 1060	1280	1506	192	391,968	1.45
GTX 1070	1920	1506	256	232,801	2.44
RTX 2070	2304	1410	448	129,665	4.38
RTX 2080	2944	1515	448	92,141	6.17
RTX 2080 Ti	4352	1350	616	65,916	8.62

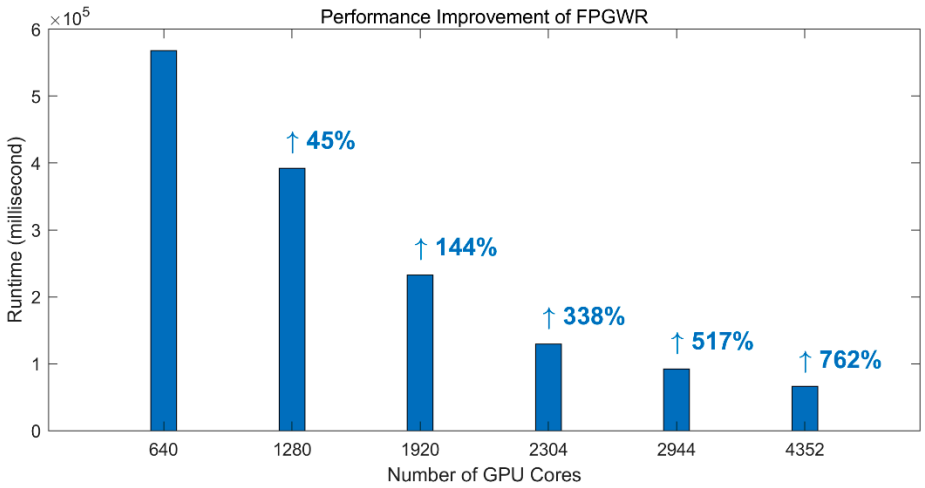


Figure 5. Performance comparison of FPGWR for an increasing number of GPU cores.

4.3.2. Comparison of FPGWR and Other GWR

Benefiting from the development of computer hardware, researchers could quickly and easily build the GPU environment for large-scale spatial study. To verify the acceleration capability, another four GWR—namely, FastGWR (Python), MGWR (Python), GWmodel (R), and spgwr (R)—are selected to compare with FPGWR. The test data utilized by the experiment is the "Zillow test dataset." GWmodel uses moving window weighting technique to decrease the computation. FastGWR implements distributed parallelism in HPC environment to improve operating efficiency. FastGWR is superior than MGWR, GWmodel and spgwr in terms of overall calculating efficiency. As a side note, although the optimal environment for FastGWR is an HPC cluster, it is more unbiased to conduct the experiments based on a single desktop environment.

The runtimes of the five packages with different sample sizes are displayed in Table 4. The runtime merely contains the single regression time with a specified bandwidth (as in Section 4.3.1). Given 1000 observations, FPGWR is 5 times faster than FastGWR, 32 times faster than MGWR, 88 times faster than GWmodel, and 865 times faster than spgwr. As the sample size increases to 10,000, FPGWR is approximately 14 times faster than FastGWR, approximately 157 times faster than MGWR, approximately 2185 times faster than GWmodel, and approximately 45,811 times faster than spgwr. Once the sample size exceeds 20,000, spgwr will fail to complete the regression task first, followed by GWmodel and MGWR. The cause is that the three schemes fail to avoid storing the high-dimensional weight matrix and other intermediate matrices.

Table 4. Runtime (in seconds) for five GWR packages using different numbers of data points.

Number of Data Points	FPGWR	FastGWR	MGWR	GWmodel	Spgwr
1000	0.01	0.05	0.32	0.88	8.65
2000	0.02	0.13	0.95	4.12	60.26
5000	0.06	0.47	5.43	53.15	1095.80
10,000	0.18	2.45	28.34	393.30	8245.93
15,000	0.27	4.42	58.97	1464.12	n/a
20,000	0.50	6.76	n/a	n/a	n/a
50,000	2.72	64.57	n/a	n/a	n/a
100,000	10.80	307.12	n/a	n/a	n/a

The runtimes of the five packages are illustrated in Figure 6. The *y*-axis is marked on a logarithmic scale to display all of the results together. Observing each package separately reveals that the runtimes of the five schemes all exhibit a logarithmic increasing trend. According to Figure 6, the performances of the five packages are enhanced generation by generation. FPGWR is the most ideal implementation among these schemes.

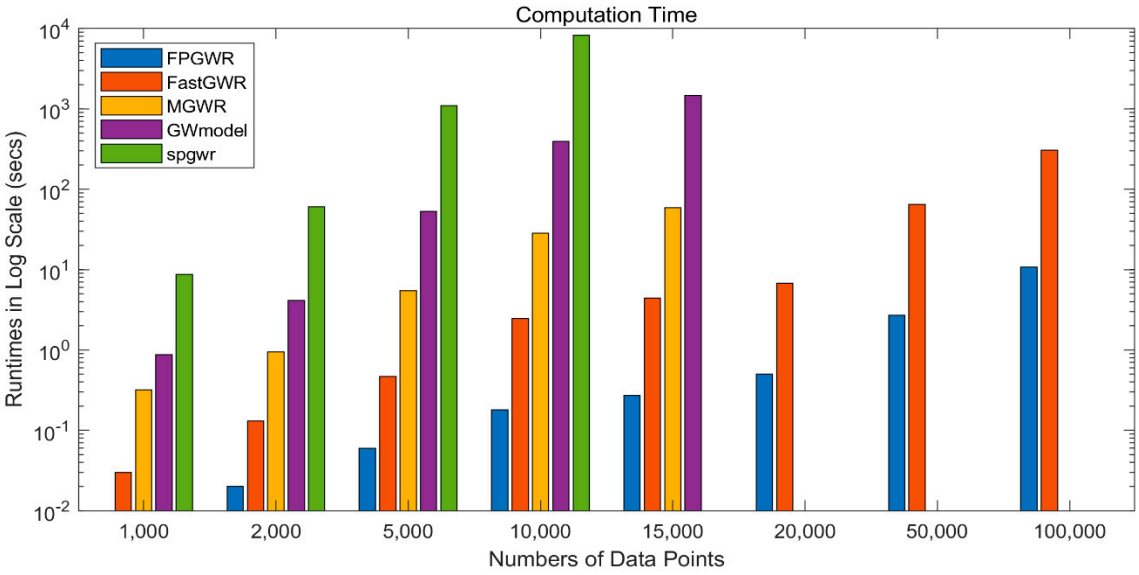


Figure 6. Runtime comparison (bar) for different packages using different numbers of data points.

Overall, FPGWR is a feasible GWR accelerator with a low development cost and simple productization process. Compared with other packages, FPGWR can greatly simplify a complicated job through decomposing the redundant full-sample regression.

4.3.3. Validation of the Result Accuracy

To validate the accuracy of FPGWR against other GWR packages, the results ($\hat{\beta}$, $Adj.R^2$ and AIC_c scores) of the five packages are compared based on the well-known "Georgia" dataset. The dependent variable *PctBach* and independent variables *Intercept*, *PctPov*, *PctRural* and *PctBlack* are chosen to calibrate the same GWR model according to Equation (31). On the basis of the adaptive Bi-square kernel function, 93 nearest neighbors are selected for the optimal bandwidth. As shown in Table 5, the Mean and Standard Deviation of the estimated coefficients $\hat{\beta}$ are displayed in the middle section, and the $Adj.R^2$ and AIC_c scores are indicated in the lower section. The FPGWR result is clearly consistent with those of the other four packages.

Table 5. Statistical results of local coefficient estimates and regression estimates for five GWR packages.

Variables	FPGWR	FastGWR	MGWR	GWmodel	Spgwr
			Mean		
			Standard Deviation		
Intercept	23.0748 4.1048	23.0748 4.1048	23.0748 4.1048	23.0748 4.1048	23.0748 4.1048
PctPov	−0.2625 0.0916	−0.2625 0.0916	−0.2625 0.0916	−0.2625 0.0916	−0.2625 0.0916
PctRural	−0.1181 0.0370	−0.1181 0.0370	−0.1181 0.0370	−0.1181 0.0370	−0.1181 0.0370
PctBlack	0.0445 0.0576	0.0445 0.0576	0.0445 0.0576	0.0445 0.0576	0.0445 0.0576
PctBach	10.9363 4.3489	10.9363 4.3489	10.9363 4.3489	10.9363 4.3489	10.9363 4.3489
			Value		
$Adj.R^2$	0.5812	0.5812	0.5812	0.5812	0.5812
AICc	896.35	896.35	896.35	896.35	896.35

The spatial distribution of the estimated coefficients $\hat{\beta}$ in the study area is illustrated in Figure 7. To simulate the spatial variation better, both the surfaces are interpolated as a continuous surface utilizing the griddata method.

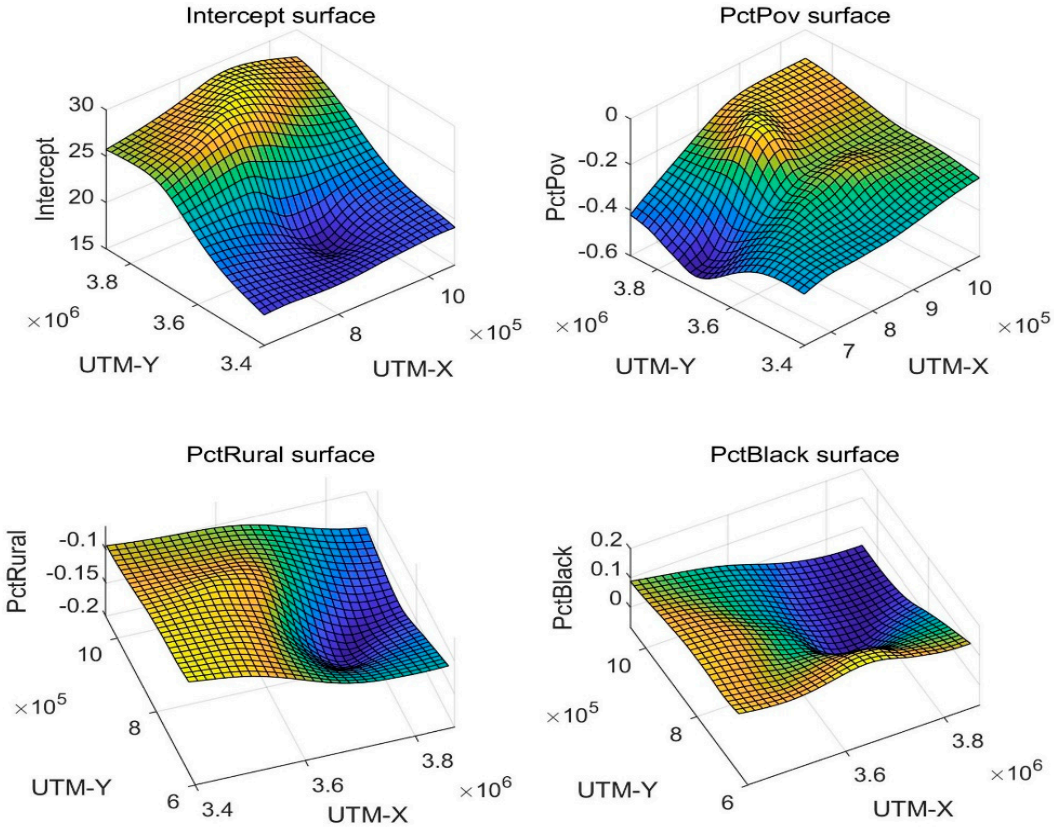


Figure 7. Surfaces of coefficient estimates.

4.4. Discussion

Multiple loops are necessary in the GWR model until FPGWR atomizes the large process. Enormous calculation redundancy would inevitably emerge in the implementation of the GWR algorithm. The algorithm structure is nested with multilevel loops, where the upper loop depends on the lower loop and the internal sequence of each loop is fixed. It is illegitimate to disturb the original iterative sequence of the subprocesses; otherwise, the accuracy of the regression results would be questioned seriously. FPGWR introduces a hybrid (parallel–serial) mode, which could enable the GPU device to not only tolerate parallel tasks of each batch but also complete all of the tasks efficiently. The subprocesses could be randomly executed without errors, and the accuracy of the results is guaranteed for the model diagnosis. FPGWR differs obviously from GWR in its memory usage and time cost.

4.4.1. Memory

The matrix storage strategy of GWR is different from FPGWR, as shown in Figure 8. The FPGWR optimizes the storage mode in utilizing the schemes, the on-demand storage and the matrix vectorization. The weight matrix W_i is stored as an $n \times n$ diagonal matrix in the GWR model. Although only the diagonal elements must be solved, a storage space of size n^2 is demanded. The memory complexity could be expressed as $O(n^2)$. In the subsequent steps, the calculations of the matrices B, B^{-1} and A all inherit the memory complexity. In comparison, the FPGWR method only stores the data as required. Its memory complexity can be reduced into $O((p+1)n)$ ($p+1 \leq 10$ in common).

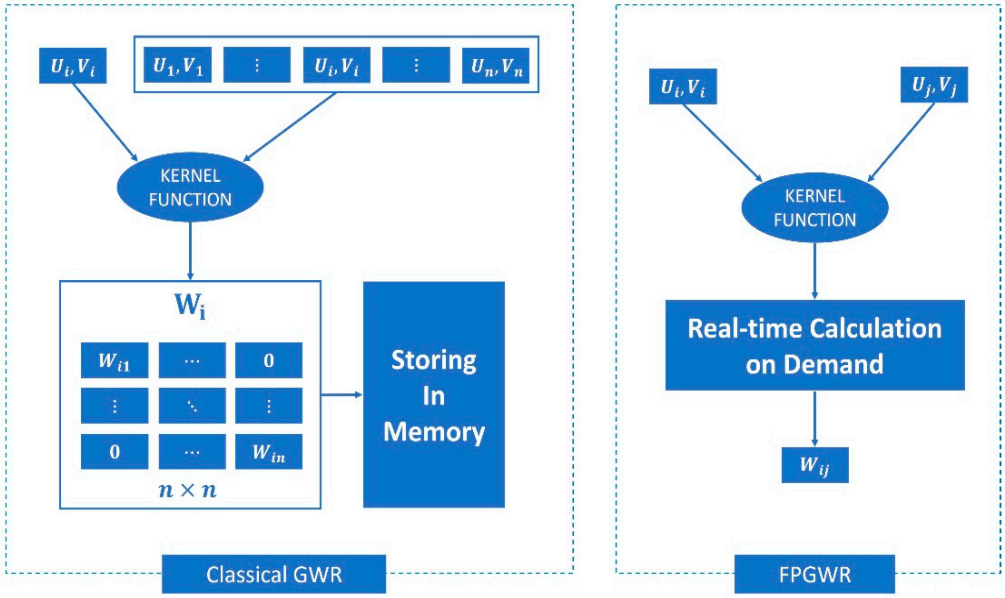


Figure 8. Different storage modes of weight matrix between classical GWR and FPGWR.

Table 6 illustrates the comparison of memory usage between FPGWR and GWR. When the sample size is less than 100,000, the memory of the GPU device is still available for usage. Once the size is increased to 10,000,000, GWR approximately requires 364 TB RAM. Any existing single GPU device could not allocate so much storage space. In contrast, FPGWR consumes only 380 *MB* RAM. To summarize, FPGWR demonstrates a tremendous advantage over GWR.

Table 6. The comparison of memory usage for FPGWR against classical GWR.

Number of Data Points	FPGWR	Classical GWR
100	3.9 KB	39 KB
1000	39 KB	3.8 MB
10,000	390 KB	380 MB
100,000	3.8 MB	38 GB
1,000,000	38 MB	3.8 TB
10,000,000	380 MB	364 TB

Note: 32-bit floats were used for all decimals and $p = 9$.

4.4.2. Time

It is necessary for the process of calculating the hat matrix S in GWR. The weight matrix W_i is a special diagonal matrix that does not increase the runtime complexity during matrix multiplication. The runtime complexity of matrices A is $O(1)$, and the runtime complexity of B is $O\left((p+1)^2 n\right)$. Because $p+1$ is far less than n, the computing time of matrix B can be ignored. Given a fixed weight bandwidth, the runtime complexity of hat matrix S_i can be expressed as $O\left((p+1)^2 n\right)$. After n operations on S_i, the runtime complexity of matrix S is defined by $O\left((p+1)^2 n^2\right)$. Different GWR schemes utilize varied ways to select the optimal bandwidth, and thereby, the study will omit a discussion about the runtime complexity of the whole process. The regression subprocess of single sample is appropriate to be used for the runtime analysis in this subsection. Instead of computing iteratively, FPGWR atomizes the regression process of each point as an independent thread. The strategy has reduced the runtime complexity of matrix S appreciably. At the same time, the runtime complexity of matrix of A becomes $O((p+1)n)$, and the runtime complexity of matrix of B becomes $O\left((p+1)^2 n\right)$. By combining matrices A and B in the form of parallel addition, the hat matrix S_i gains a runtime complexity of $O((p+1)(p+2)n)$. The runtime complexity of FPGWR is theoretically lower than GWR, but the instruction operation efficiency of host device differs significantly from the GPU device, and the methods used by the different libraries to optimize the matrix operation are inconsistent. Therefore, the actual comparison results should refer to the experimental results (in Section 4.3.2).

The achievement of GWR regression coefficients requires consuming much time for repeated iteration when handling big geodata. For example, when the sample size is 1,000,000, single point regression of GWR needs to be iterated for 1,000,000 times with a huge time occupying and memory usage. Therefore, this problem could be solved by parallelization strategy. The atomization algorithm does not store the weight matrix and other temporary matrices during each point regression iteration, but only reads and calculates the matrix elements on-demand. FPGWR shortens computation time while using much less memory space through parallelizing these atomic units on CUDA.

5. Conclusions

GWR is a local modeling technique that has been widely used in various disciplines. However, GWR has significant computational redundancy and can handle approximately 15,000 geographical observations at most. To apply the local smoothing technique on a large-scale spatial dataset, we proposed an improved algorithm FPGWR to solve these problems. FPGWR optimizes the matrix storage mode to overcome the limitation on memory space, thereby significantly reducing the memory complexity of GWR. Furthermore, it introduces a parallel computing mode, decomposing the full-sample large cycle into an atomization process, to decrease the runtime complexity substantially.

To demonstrate the practicability of FPGWR, simulation and Zillow datasets are used to conduct the experiment. The results show that the regression runtime is exponentially related to the number of observations, and thus, GWR is unable to process the regression task with large volumes of geodata. In comparison, the time taken up by FPGWR exhibits a logarithmic relationship with the number

of observations; hence, FPGWR represents a significant advance in handling the massive geodata mining task.

In summary, the dilemma that limits GWR in the data scale could be considerably alleviated by FPGWR, and thus, the application domains of GWR would be potentially expanded to a large extent. Under these circumstances, increasingly large datasets from geographical or nongeographical fields could be converted to the providers of the large-scale geographic analysis services. In the future, we will investigate a key issue: how to adapt FPGWR to non-CUDA architectures, even other non-GPU HPC devices, to enhance the versatility of the extended algorithm.

Author Contributions: Conceptualization, Dongchao Wang, and Yi Yang; Methodology, Dongchao Wang; Resources, Dongchao Wang; Software, Dongchao Wang; Validation, Dongchao Wang; Writing—original draft, Dongchao Wang; Writing—review and editing, Dongchao Wang, Yi Yang, Agen Qiu, Xiaochen Kang, Jiakuan Han, and Zhengyuan Chai. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key Research and Development Project (grant number 2019YFB2102500/2019YFB2102503), the National Natural Science Foundation of China (grant number 71903183, 41801316, 41701461), and the Basic Scientific Research Fund of CASM (grant number AR1910).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Toch, E.; Lerner, B.; Ben-Zion, E.; Ben-Gal, I. Analyzing large-scale human mobility data: A survey of machine learning methods and applications. *Knowl. Inf. Syst.* **2019**, *58*, 501–523. [CrossRef]
2. Weckström, C.; Kujala, R.; Mladenović, M.N.; Saramäki, J. Assessment of large-scale transitions in public transport networks using open timetable data: Case of Helsinki metro extension. *J. Transp. Geogr.* **2019**, *79*, 102470. [CrossRef]
3. Hicks, J.L.; Althoff, T.; Sosic, R.; Kuhar, P.; Bostjancic, B.; King, A.C.; Leskovec, J.; Delp, S.L. Best practices for analyzing large-scale health data from wearables and smartphone apps. *NPJ Digit. Med.* **2019**, *2*, 1–12. [CrossRef]
4. Tasar, O.; Tarabalka, Y.; Alliez, P. Incremental learning for semantic segmentation of large-scale remote sensing data. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2019**, *12*, 3524–3537. [CrossRef]
5. Li, Z.; Huang, Q.; Jiang, Y.; Hu, F. SOVAS: A scalable online visual analytic system for big climate data analysis. *Int. J. Geogr. Inf. Sci.* **2020**, *34*, 1188–1209. [CrossRef]
6. Miller, H.J.; Goodchild, M.F. Data-driven geography. *GeoJournal* **2015**, *80*, 449–461. [CrossRef]
7. Xia, J.; Huang, S.; Zhang, S.; Li, X.; Lyu, J.; Xiu, W.; Tu, W. DAPR-tree: A distributed spatial data indexing scheme with data access patterns to support Digital Earth initiatives. *Int. J. Digit. Earth* **2020**, 1–16. [CrossRef]
8. Aji, A.; Wang, F.; Vo, H.; Lee, R.; Liu, Q.; Zhang, X.; Saltz, J. Hadoop-GIS: A high performance spatial data warehousing system over MapReduce. In Proceedings of the VLDB Endowment International Conference on Very Large Data Bases, Trento, Italy, 26–30 August 2013; Volume 6.
9. Wu, Q.Y.; Su, K.Y.; Zou, Z.J. A mapreduce-based method for parallel calculation of bus passengers origin and destination from massive transit data. *J. Geo Inf. Sci.* **2018**, *20*, 647–655.
10. Wilkinson, B.; Allen, M. *Parallel Programming*; Prentice Hall: Upper Saddle River, NJ, USA, 1999.
11. Gong, Z.; Tang, W.; Bennett, D.A.; Thill, J.-C.F. Parallel agent-based simulation of individual-level spatial interactions within a multicore computing environment. *Int. J. Geogr. Inf. Sci.* **2013**, *27*, 1152–1170. [CrossRef]
12. Tang, W.; Feng, W.; Jia, M. Massively parallel spatial point pattern analysis: Ripley's K function accelerated using graphics processing units. *Int. J. Geogr. Inf. Sci.* **2015**, *29*, 412–439. [CrossRef]
13. Zhang, G.; Zhu, A.X.; Huang, Q. A GPU-accelerated adaptive kernel density estimation approach for efficient point pattern analysis on spatial big data. *Int. J. Geogr. Inf. Sci.* **2017**, *31*, 2068–2097. [CrossRef]
14. Sandric, I.; Ionita, C.; Chitu, Z.; Dardala, M.; Irimia, R.; Furtuna, F.T. Using CUDA to accelerate uncertainty propagation modelling for landslide susceptibility assessment. *Environ. Model. Softw.* **2019**, *115*, 176–186. [CrossRef]
15. Stojanovic, N.; Stojanovic, D. Parallelizing multiple flow accumulation algorithm using cuda and openacc. *ISPRS Int. J. Geo Inf.* **2019**, *8*, 386. [CrossRef]
16. Pei, T.; Song, C.; Guo, S.; Shu, H.; Liu, Y.; Du, Y.; Ma, T.; Zhou, C. Big geodata mining: Objective, connotations and research issues. *J. Geogr. Sci.* **2020**, *30*, 251–266. [CrossRef]

17. Brunsdon, C.; Fotheringham, A.S.; Charlton, M.E. Geographically weighted regression: A method for exploring spatial nonstationarity. *Geogr. Anal.* **1996**, *28*, 281–298. [CrossRef]
18. Zhang, P.; Yang, D.; Zhang, Y.; Li, Y.; Liu, Y.; Cen, Y.; Zhang, W.; Geng, W.; Rong, T.; Liu, Y.; et al. Re-examining the drive forces of China's industrial wastewater pollution based on GWR model at provincial level. *J. Clean. Prod.* **2020**, *262*, 121309. [CrossRef]
19. Wu, D. Spatially and Temporally Varying Relationships between Ecological Footprint and Influencing Factors in China's Provinces Using Geographically Weighted Regression (GWR). *J. Clean. Prod.* **2020**, *261*, 121089. [CrossRef]
20. Yuan, Y.; Cave, M.; Xu, H.; Zhang, C. Exploration of spatially varying relationships between Pb and Al in urban soils of London at the regional scale using geographically weighted regression (GWR). *J. Hazard. Mater.* **2020**, *393*, 122377. [CrossRef]
21. Hong, I.; Yoo, C. Analyzing Spatial Variance of Airbnb Pricing Determinants Using Multiscale GWR Approach. *Sustainability* **2020**, *12*, 4710. [CrossRef]
22. Wu, S.; Wang, Z.; Du, Z.; Huang, B.; Zhang, F.; Liu, R. Geographically and temporally neural network weighted regression for modeling spatiotemporal non-stationary relationships. *Int. J. Geogr. Inf. Sci.* **2020**, 1–27. [CrossRef]
23. Bivand, R.; Yu, D.; Nakaya, T.; Garcia-Lopez, M.A. *Package SPGWR*; R Software Package; R Foundation for Statistical Computing: Vienna, Austra, 2020.
24. Gollini, I.; Lu, B.; Charlton, M. GWmodel: An R Package for Exploring Spatial Heterogeneity Using Geographically Weighted Models. *J. Stat. Softw.* **2015**, *63*, 1–50. [CrossRef]
25. Oshan, T.M.; Li, Z.; Kang, W.; Wolf, L.J.; Fotheringham, A.S. mgwr: A Python implementation of multiscale geographically weighted regression for investigating process spatial heterogeneity and scale. *ISPRS Int. J. Geo Inf.* **2019**, *8*, 269. [CrossRef]
26. Li, Z.; Fotheringham, A.S.; Li, W.; Oshan, T. Fast Geographically Weighted Regression (FastGWR): A scalable algorithm to investigate spatial process heterogeneity in millions of observations. *Int. J. Geogr. Inf. Sci.* **2019**, *33*, 155–175. [CrossRef]
27. Tran, H.T.; Nguyen, H.T.; Tran, V.T. Large-scale geographically weighted regression on Spark. In Proceedings of the 2016 Eighth International Conference on Knowledge and Systems Engineering (KSE), Hanoi, Vietnam, 6–8 October 2016; pp. 127–132.
28. Foster, S.A.; Gorr, W.L. An adaptive filter for estimating spatially-varying parameters: Application to modeling police hours spent in response to calls for service. *Manag. Sci.* **1986**, *32*, 878–889.
29. Akaike, H. A new look at the statistical model identification. *IEEE Trans. Autom. Control* **1974**, *19*, 716–723.
30. Brunsdon, C.; Fotheringham, A.S.; Charlton, M. Geographically weighted summary statistics—A framework for localised exploratory data analysis. *Comput. Environ. Urban Syst.* **2002**, *26*, 501–524. [CrossRef]
31. Harris, R.; Singleton, A.; Grose, D.; Brundson, C.; Longley, P. Grid-enabling geographically weighted regression: A case study of participation in higher education in England. *Trans. GIS* **2010**, *14*, 43–61.
32. NVIDIA Corporation. Compute Unified Device Architecture (CUDA). Available online: https://developer.nvidia.com/cuda-toolkit (accessed on 6 October 2020).
33. Fotheringham, A.S.; Brunsdon, C.; Charlton, M. *Geographically Weighted Regression: The Analysis of Spatially Varying Relationships*; John Wiley & Sons: Hoboken, NJ, USA, 2002.
34. Zhang, H.; Mei, C. Local least absolute deviation estimation of spatially varying coefficient models: Robust geographically weighted regression approaches. *Int. J. Geogr. Inf. Sci.* **2011**, *25*, 1467–1489.
35. Eager, D.L.; Zahorjan, J.; Lazowska, E.D. Speedup versus efficiency in parallel systems. *IEEE Trans. Comput.* **1989**, *38*, 408–423.
36. Yang, L.; Sun, X.; Li, Z. An efficient framework for remote sensing parallel processing: Integrating the artificial bee colony algorithm and multiagent technology. *Remote Sens.* **2019**, *11*, 152. [CrossRef]