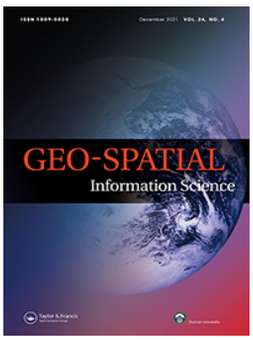

Geo-spatial Information Science

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tgsi20

High-performance solutions of geographically weighted regression in R

Binbin Lu, Yigong Hu, Daisuke Murakami, Chris Brunsdon, Alexis Comber, Martin Charlton & Paul Harris

To cite this article: Binbin Lu, Yigong Hu, Daisuke Murakami, Chris Brunsdon, Alexis Comber, Martin Charlton & Paul Harris (2022): High-performance solutions of geographically weighted regression in R, Geo-spatial Information Science, DOI: 10.1080/10095020.2022.2064244

To link to this article: https://doi.org/10.1080/10095020.2022.2064244

Published online: 20 May 2022.

GEO-SPATIAL INFORMATION SCIENCE
https://doi.org/10.1080/10095020.2022.2064244

Taylor & Francis
Taylor & Francis Group

High-performance solutions of geographically weighted regression in R

Binbin Lu [a], Yigong Hu [a], Daisuke Murakami [b], Chris Brunsdon [c], Alexis Comber [d], Martin Charlton [c] and Paul Harris [e]

[a]School of Remote Sensing and Information Engineering, Wuhan University, Wuhan, China; [b]Department of Data Science, Institute of Mathematical Statistics, Tokyo, Japan; [c]National Centre for Geocomputation, Maynooth University, Maynooth, Ireland; [d]School of Geography, University of Leeds, Leeds, UK; [e]Sustainable Agriculture Sciences North Wyke, Rothamsted Research, Okehampton, UK

ABSTRACT

As an established spatial analytical tool, Geographically Weighted Regression (GWR) has been applied across a variety of disciplines. However, its usage can be challenging for large datasets, which are increasingly prevalent in today's digital world. In this study, we propose two high-performance R solutions for GWR via Multi-core Parallel (MP) and Compute Unified Device Architecture (CUDA) techniques, respectively GWR-MP and GWR-CUDA. We compared GWR-MP and GWR-CUDA with three existing solutions available in Geographically Weighted Models (GWmodel), Multi-scale GWR (MGWR) and Fast GWR (FastGWR). Results showed that all five solutions perform differently across varying sample sizes, with no single solution a clear winner in terms of computational efficiency. Specifically, solutions given in GWmodel and MGWR provided acceptable computational costs for GWR studies with a relatively small sample size. For a large sample size, GWR-MP and FastGWR provided coherent solutions on a Personal Computer (PC) with a common multi-core configuration, GWR-MP provided more efficient computing capacity for each core or thread than FastGWR. For cases when the sample size was very large, and for these cases only, GWR-CUDA provided the most efficient solution, but should note its I/O cost with small samples. In summary, GWR-MP and GWR-CUDA provided complementary high-performance R solutions to existing ones, where for certain data-rich GWR studies, they should be preferred.

ARTICLE HISTORY
Received 14 December 2021
Accepted 6 April 2022

KEYWORDS
Non-stationarity; big data; parallel computing; Compute Unified Device Architecture (CUDA); Geographically Weighted models (GWmodel)

1. Introduction

Geographically Weighted Regression (GWR) (Brunsdon, Fotheringham, and Charlton 1996, 1998; Fotheringham, Charlton, and Brunsdon 1998; Fotheringham, Brunsdon, and Charlton 2002) is a technique specifically developed to explore spatial heterogeneities in a regression's "response to predictor variable" relationships. Unlike a fixed coefficient regression, such as an Ordinary Least Squares (OLS) regression, GWR allows regression coefficients to vary spatially; the resultant coefficient maps allow an investigation into their change (if any) across space. The GWR methodology has been extensively developed in terms of its usage and extensions (Comber et al. 2022), but where inference in GWR is not always as stable as that found with say, an OLS regression and as such, GWR adaptations exist to counter this (da Silva and Fotheringham 2016; Harris et al. 2017). GWR has been widely applied in many scientific domains, including regional economics (e.g. Jin, Xu, and Huang 2019), urban planning (e.g. Cao et al. 2019b), sociology (e.g. Yin et al. 2018), ecology (e.g. Liu et al. 2019), public health (e.g. Wang et al. 2019; Xu et al. 2021), agriculture (e.g. Harris et al. 2017), and environmental science (e.g. Cao et al. 2019a; Huang and Wang 2020).

Our increasingly digital world continues to generate huge volumes of data – many of which are spatially indexed (Lee and Kang 2015; Ivan et al. 2017). However, in order to attribute process understanding to such "Big Spatial Data" almost all spatial models require adaptation so they can be efficiently calibrated and validated within tolerable time frames. GWR is one such model that is computationally demanding and in this respect has benefitted from high-performance computing solutions (Harris et al. 2010; Murakami et al. 2020; Li et al. 2019b). Commonly, such solutions only exist for the conventional forms of GWR, where many extended GWR models are more computationally demanding still – for example, multiscale GWR (Lu et al. 2018; Li and Fotheringham 2020) which requires a complex iterative solution to its calibration. Similarly, Geographically and Temporally Weighted Regression (GTWR) (Huang, Wu, and Barry 2010; Fotheringham, Crespo, and Yao 2015) for space-time processes has a higher computational demand than that found with conventional GWR.

Unsurprisingly, there are an increasing number of (conventional) GWR applications exploring "Big Data" (e.g. Cao, Diao, and Wu 2019). Here, we conducted a bibliometric study, searching the keyword "Geographically Weighted Regression" via Web of

CONTACT Binbin Lu binbinlu@whu.edu.cn

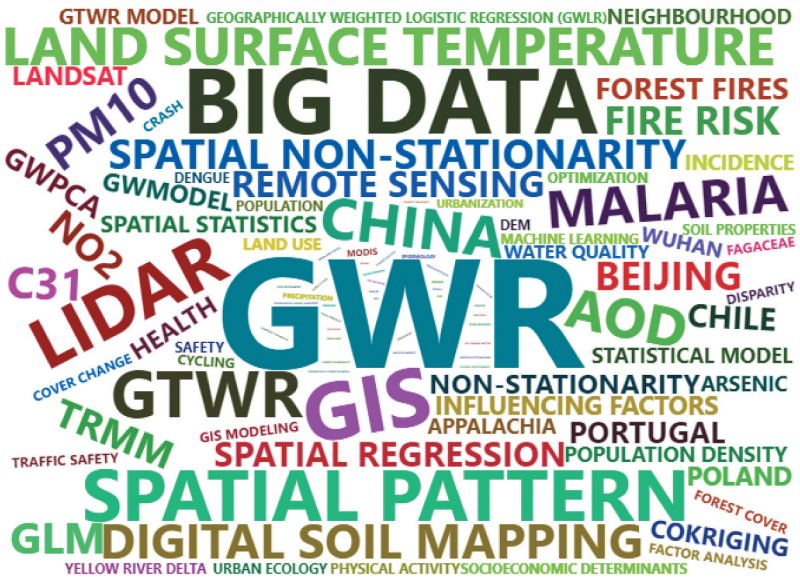

Figure 1. Word cloud of keywords from 2014 articles on GWR queried via WoS from 1999 to 2019.

Science (WoS), where in total, 2014 articles were found from 1999 to 2019, and their keywords are visualized in a word cloud form (Figure 1). Observe the frequency (size) of "Big Data", which appears second only to "GWR". Thus, the demand for high-performance solutions for GWR is clear, where its application in "Big Data" problems can be limited (Murakami et al. 2020), even with the employment of the existing solutions listed above (section 1.2).

1.1. Existing implementations of GWR

There are a number of standalone implementations with GWR enabled, such as GWR3 (Charlton, Fotheringham, and Brunsdon 2003), GWR4 (Nakaya et al. 2009), the GWR tool in ESRI ArcGIS (ESRI Corp 2011), and Multi-scale GWR (MGWR) (Li et al. 2019b). GWR is also available through scripting platforms with: the mgwr module of the PySal package in Python (Oshan et al. 2019); as part of the econometrics toolbox in MATLAB (LeSage and Pace 2009); and five R packages – spgwr (Bivand and Yu 2006), Geographically Weighted Models (GWmodel) (Lu et al. 2014b; Gollini et al. 2015), gwrr (Wheeler 2013), McSpatial (McMillen 2015) and lctools (Kalogirou 2016). The five R packages considered as a whole provide the richest suite of GWR forms (e.g. conventional, robust, heteroskedastic, multiscale, space-time and more) and therefore development here is most appropriate. However, all suffer computationally, particularly given the strict memory limit for specific operation systems (R Core Team 2020). Workarounds to exceeding computational limits exist, such as coarse-scaling the observations, or the use of aggregations via upscaling (e.g. Yang et al. 2019) – all prior to a GWR fit, but none are ideal given important sources of information, fine scale detail and variability are lost.

1.2. Existing high-performance solutions

Efforts to improve the computational efficiency of GWR exist. Firstly, through Harris et al. (2010) who implemented a grid-based (parallelization) approach to conventional GWR. More recently, Li et al. (2019a) developed a python implementation (FastGWR) that optimizes the conventional GWR algorithm together with embedding multi-core parallel computing technology. This computational scheme has also been transplanted for use with multiscale GWR (Li and Fotheringham 2020). Wang et al. (2020) proposed a high-performance solution of GWR with the Compute Unified Device Architecture (CUDA), namely Fast-Parallel-GWR (FPGWR) which was developed with Microsoft Visual Studio 2015 and CUDA development kit. Finally, a mathematical approach was taken by Murakami et al. (2020) who proposed Scalable GWR (ScaGWR) that saves on computational overheads via the pre-compressing of large matrices and vectors with polynomial kernels. For ScaGWR, the computational cost presents a linear relationship with the sample size, while a quad-quadratic order appears for the usual un-adapted GWR form. The ScaGWR routine can be found in the R package scgwr (Murakami et al. 2019) and GWmodel. ScaGWR provides approximate coefficient estimates in comparison with conventional GWR, where the results from ScaGWR might vary slightly when different parameters are specified – for example, the chosen degree or order of the polynomials (Murakami et al. 2020).

1.3. Our study's approach in R

In this sense, the computational bottleneck is still problematic for GWR (and its extensions) in the R environment, particularly for the geographically weighted functions in GWmodel. However, generic high-performance computing options have been incorporated in many packages since R release 2.14.0 (Eddelbuettel 2020), where grid computing, cloud computing, multi-core and Graphic Processing Unit (GPU) are commonly invoked. In this respect, this study investigates high-performance solutions for (conventional) GWR within the R package GWmodel, where our workflow consists of three hierarchies: 1) optimize the algorithm for a GWR calibration for accepting out-of-memory issues with "Big Data"; 2) adopt multi-thread parallel computing for a GWR calibration (GWR-MP), which enables analysis on a standard Personal Computer (PC) with a multi-core processor; 3) apply parallel computing on the GPU devices via CUDA (GWR-CUDA).

For performance evaluation, we compare the performances of the new solutions proposed (i.e. GWR-MP and GWR-CUDA) with existing solutions found in GWmodel, MGWR and FastGWR using varying sample sizes, where the latter two are outside of the R environment. We haven't included FPGWR where CUDA was also adopted as: 1) the source code or tool is not available; and 2) key aspects of FPGWR are not clear, such as distance calculation, kernel function implementation, making FPGWR difficult to fully reproduce. This study is organized as follows. Firstly, we provide a description of conventional GWR methodology and the new high-performance techniques proposed. Secondly, competing high-performance solutions to GWR are objectively compared through a designed experiment. Thirdly, we summarize and suggest future research.

2. The GWR methodology and high-performance solutions

2.1. Basics of GWR

The conventional GWR model characterizes spatially varying relationships via location-specific regressions whose coefficients are estimated by (geographically) weighted least squares. The model can be expressed as (Brunsdon, Fotheringham, and Charlton 1996; Fotheringham, Brunsdon, and Charlton 2002):

$$y_i = \beta_0(u_i, v_i) + \sum_{k=1}^{l} \beta_k(u_i, v_i) x_{ik} + \varepsilon_i \qquad (1)$$

where y_i is the dependent variable at location i on a two-dimensional space; x_{ik} is the value of the k^{th} independent variable at location i; l is the number of independent variables; $\beta_0(u_i, v_i)$ is the intercept parameter at location i; $\beta_k(u_i, v_i)$ is the local regression coefficient for the k^{th} independent variable at location i; (u_i, v_i) are the spatial coordinates of location i; and ε_i is the independent random error at location i.

In line with Tobler's first law of geography (Tobler 1970), extended to consider situations in which nearby regression relationships are more similar than distant ones, GWR consists of a series of local regressions where observations are weighted (i.e. given decreasing influence) via a distance-decay kernel function (Lu et al. 2014a). The estimator of the coefficients at location i has the following matrix expression:

$$\widehat{\boldsymbol{\beta}}(u_i, v_i) = \left(X^{\mathrm{T}} \boldsymbol{W}(u_i, v_i) \boldsymbol{X}\right)^{-1} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{W}(u_i, v_i) \boldsymbol{y} \qquad (2)$$

where $\boldsymbol{X}$ is the matrix of the independent variables with a column of 1s for the intercept; $\boldsymbol{y}$ is the dependent variable vector; $\widehat{\boldsymbol{\beta}}(u_i, v_i) = \left(\hat{\beta}_0(u_i, v_i), \hat{\beta}_1(u_i, v_i), \ldots, \hat{\beta}_m(u_i, v_i)\right)^{\mathrm{T}}$ is the vector of m +1 local regression coefficients; $W(u_i, v_i)$ is a $n \times n$ diagonal matrix denoting geographical weights of each observation for calibrating the local regression at location i, and is defined as:

$$W(u_i, v_i) = \begin{bmatrix} w_{i1} & 0 & .. & .. & 0 \\ 0 & w_{i2} & .. & .. & 0 \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & 0 & . & . & w_{in} \end{bmatrix} \qquad (3)$$

where $w_{ij} (j = 1, \cdots, n)$ is calculated via a kernel function decaying with respect to Euclidean distance, or some other distance metric (Lu et al. 2014a), between locations i and j, and n represents the number of observations. Gaussian, exponential, bi-square, box-car, tri-cube are among the many kernel functions that can be specified (Gollini et al. 2015), where an optimal kernel bandwidth is commonly found by leave-one-out cross-validation or by a corrected Akaike Information Criterion (AICc) procedure. The kernel bandwidth relays the chosen spatial scale of the regression relationships.

Diagnostics for a GWR model's fit are essential, where R-squared, adjusted R-squared and AICc are commonly reported. These can be expressed as (Fotheringham, Brunsdon, and Charlton 2002):

$$R^2 = \frac{\sum_i \left(\widehat{y}_i - \bar{y}_i\right)^2}{\sum_i \left(\widehat{y}_i - \bar{y}_i\right)^2 + \sum_i \left(y_i - \widehat{y}_i\right)^2} \qquad (4)$$

$$R^2_{\text{adjusted}} = 1 - \left(1 - R^2\right) \frac{n - 1}{n - 2\mathrm{tr}(\boldsymbol{S}) + \mathrm{tr}\left(\boldsymbol{S}^{\mathrm{T}} \boldsymbol{S}\right) - 1} \qquad (5)$$

$$\mathrm{AIC}_c = 2n\ln(\hat{\sigma}) + n\ln(2\pi) + n\left\{\frac{n+\mathrm{tr}(\boldsymbol{S})}{n-2-\mathrm{tr}(\boldsymbol{S})}\right\} \tag{6}$$

where $\widehat{y}_i$ is the fitted value at location i; $\bar{y}$ is the mean value of $\boldsymbol{y}$; $\hat{\sigma}$ is the estimated standard deviation of the error term:

$$\widehat{\sigma}^2 = \frac{\sum_i \left(\widehat{y}_i - \bar{y}_i\right)^2}{n - 2\mathrm{tr(S)} + \mathrm{tr}(S^{\mathrm{T}}S)} = \frac{y^{\mathrm{T}}(I-S)^{\mathrm{T}}(I-S)Y}{n - 2\mathrm{tr(S)} + \mathrm{tr}(S^{\mathrm{T}}S)} \tag{7}$$

where $\boldsymbol{I}$ is an $n \times n$ identity matrix and $\mathrm{tr}(\boldsymbol{S})$ and $\mathrm{tr}\left(\boldsymbol{S}^{\mathrm{T}}\boldsymbol{S}\right)$ denote the traces of the hat matrix $\boldsymbol{S}$ and $\boldsymbol{S}^{\mathrm{T}}\boldsymbol{S}$. For GWR, each row S_i of the hat matrix can be found as follows:

$$\boldsymbol{S}_i = \boldsymbol{X}_i\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}(u_i,v_i)\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}(u_i,v_i) \tag{8}$$

where $\boldsymbol{X}_i$ is its i^{th} row of the matrix $\boldsymbol{X}$ of independent variables.

Furthermore, t statistics at each individual regression point can be produced along with the coefficient estimates. For each estimated regression coefficient at location i, $\widehat{\beta}_k(u_i,v_i)$, the t statistic can be calculated by:

$$t_{k,i} = \frac{\widehat{\beta}_k(u_i,v_i)}{\mathrm{SE}\left(\widehat{\beta}_k(u_i,v_i)\right)} \tag{9}$$

where $\mathrm{SE}\left(\widehat{\beta}_k(u_i,v_i)\right)$ is the localised standard error of $\widehat{\beta}_k(u_i,v_i)$. For each location-specific calibration, the standard errors are obtained from:

$$\mathrm{SE}\left(\widehat{\beta}_{\mathbf{i}}\right) = \widehat{\sigma}\sqrt{\mathrm{diag}\left(\boldsymbol{C}_i\boldsymbol{C}_i^{\mathrm{T}}\right)} \tag{10}$$

where

$$\boldsymbol{C}_i = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}(u_i,v_i)\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}(u_i,v_i) \tag{11}$$

The given calculations are commonly reported in most GWR software tools, where the algebraic matrix operations are programmed in a straightforward manner. However, their computational cost is expensive, particularly when dataset size (number of observations n) is large. Computational burden is primarily a consequence of: 1) complex matrix operations, particularly the $n \times n$ matrices involved, like the hat matrix $\boldsymbol{S}$; and 2) a large number of matrix operations are repeated in the location-wise calibrations, kernel bandwidth optimization and when calculating the model fit diagnostics.

2.2 Reducing memory cost for GWR

A variety of GWR forms and extensions are present in GWmodel, making it the most comprehensive GWR R package (Comber et al. 2022). In early versions of GWmodel, all GWR functions were developed directly from the algebraic formulations in Section 2.1 above. This requires a number of $n \times n$ matrices to be calculated and stored, specifically for calculating diagnostic information and enabling statistical inference (Leung, Mei, and Zhang 2000). Note here, that it is almost impossible to allocate as much as 2 GB to a single vector in a 32-bit or 64-bit build of R due to predefined allocations of address space on Windows (R Core Team 2020). Allocating memory for a 16,000 × 16,000 numeric matrix in R will normally be an upper limit. This means the maximum n for any of the conventional GWR functions in R is around 16,000. However, in practice, the maximum number of observations a conventional GWR tool can handle is likely to be much smaller (i.e. $n \ll 16{,}000$).

It is therefore necessary to first relieve these memory constraints when developing high-performance solutions for GWR, and to support GWR analyses of very large datasets. In this respect, Li et al. (2019a) optimized the calculations of AICc and localised standard errors by avoiding the storage of the entire hat matrix, which reduced the memory storage size from $\mathrm{O}(n^2)$ to $\mathrm{O}(nm)$. This strategy of avoiding any $n \times n$ matrix operation or storage is effective and makes it workable when dealing with a large dataset on any basic PC. Therefore, as a potential approach for reducing memory costs, we re-formulized the algebraic operations of a GWR calibration, as follows, which are essentially the same as the optimizations proposed by Li et al. (2019a).

First observe that Equation (2) can be divided into the following two parts:

$$\begin{aligned}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}(u_i,v_i)\boldsymbol{X} &= \left(\boldsymbol{X}_1^{\mathrm{T}},\cdots,\boldsymbol{X}_n^{\mathrm{T}}\right)\begin{pmatrix} w_{i1}\cdots & 0 \\ \vdots & \ddots & 0 \\ 0 & \cdots & w_{in}\end{pmatrix}\begin{pmatrix}\boldsymbol{X}_1 \\ \vdots \\ \boldsymbol{X}_n\end{pmatrix} \\ &= \sum_{j=1}^{n} w_{ij}\boldsymbol{X}_j^{\mathrm{T}}\boldsymbol{X}_j \\ \boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}(u_i,v_i)\boldsymbol{y} &= \left(\boldsymbol{X}_1^{\mathrm{T}},\cdots,\boldsymbol{X}_n^{\mathrm{T}}\right)\begin{pmatrix} w_{i1}\cdots & 0 \\ \vdots & \ddots & 0 \\ 0 & \cdots & w_{in}\end{pmatrix}\begin{pmatrix}y_1 \\ \vdots \\ y_n\end{pmatrix} \\ &= \sum_{j=1}^{n} w_{ij}y_j\boldsymbol{X}_j^{\mathrm{T}}\end{aligned} \tag{12}$$

where $\boldsymbol{X}_j$ is the j^{th} row of the $\boldsymbol{X}$ matrix. In this sense, the point-wise estimator of GWR can be regarded as a cross manipulation between the inverse of a $(m+1)\times(m+1)$ matrix and a $(m+1)\times 1$ vector. Accordingly, the weight matrix $\boldsymbol{W}(u_i,v_i)$ can only be stored as a vector with its diagonal elements.

For diagnostics of GWR, more complicated computations are involved, particularly for the $n \times n$ matrices, including hat matrix $\boldsymbol{S}$, its square $\boldsymbol{S}^{\mathrm{T}}\boldsymbol{S}$ and the matrix $\boldsymbol{Q} = (\boldsymbol{I} - \boldsymbol{S})^{\mathrm{T}}(\boldsymbol{I} - \boldsymbol{S})$. As shown in Equations (5–7), the traces of $\boldsymbol{S}$ and $\boldsymbol{S}^{\mathrm{T}}\boldsymbol{S}$ are needed, but where they could be found in these two steps:

$$\mathrm{tr}(\boldsymbol{S}) = \sum_{i}^{n} \boldsymbol{S}_{ii} = \sum_{i}^{n} \boldsymbol{X}_i \boldsymbol{C}_i^i \tag{13}$$

$$\mathrm{tr}\left(\boldsymbol{S}^{\mathrm{T}}\boldsymbol{S}\right) = \sum_{i}^{n} \boldsymbol{S}_i \boldsymbol{S}_i^{\mathrm{T}} \tag{14}$$

where $\boldsymbol{C}_i^i$ means the i^{th} column of matrix $\boldsymbol{C}_i$. Moreover, the matrix $\boldsymbol{Q}$ can also be expressed as follows:

$$\boldsymbol{Q} = (\boldsymbol{I} - \boldsymbol{S})^{\mathrm{T}}(\boldsymbol{I} - \boldsymbol{S}) = \sum_{i=1}^{n} (\boldsymbol{e}_i - \boldsymbol{S}_i)^{\mathrm{T}}(\boldsymbol{e}_i - \boldsymbol{S}_i) \tag{15}$$

where e_i is the i^{th} row of the identity matrix $\boldsymbol{I}$. Observe the matrix $\boldsymbol{Q}$ is also required in many statistical tests for GWR (i.e. for spatial non-stationarity), such as the *F*-tests proposed by Leung, Mei, and Zhang (2000) and Fotheringham, Brunsdon, and Charlton (2002), which are similarly included in most GWR software tools (GWR3, GWR4, as well as GWmodel). In this sense, it is natural for these *F-tests* to benefit from the high-performance solutions proposed.

According to the above equations, the storage of all $n \times n$ matrices required for a GWR calibration and associated diagnostics can be avoided, by storing only vectors of length n and matrices of size $n \times (m+1)$ in the location-wise computations. Thus, the memory cost of GWR can be similarly reduced to $O(nm)$, essential for working with "Big Spatial Data" in *R*. In the current release of GWmodel, the GWR functions have already been optimized in this respect. Therefore, for this study, the next steps are an assessment of high-performance solutions embedded in paralleling computing techniques.

2.3. Parallelization solutions for GWR

The two parallelization solutions adopted were: (a) multicore Central Processing Unit (CPU) and GPU accelerator via multithreading parallel (GWR-MP) and (b) CUDA (GWR-CUDA), respectively.

As illustrated in Figure 2, the procedure of GWR-MP was carried out in the following steps:

(1) Create the coefficient matrix $\boldsymbol{\beta}_{n\times(m+1)}$ and the vectors of n dimensions ($\boldsymbol{S}_2$) for recording the diagonal elements of $\boldsymbol{S}^{\mathrm{T}}\boldsymbol{S}$ if diagnostic information is calculated[1];
(2) Create c[2] threads, and divide the n point-wise operations into them, i.e. n_t operations are conducted on the t^{th} thread, where $\sum_{t=1}^{c} n_t = n$;
(3) For each thread, create a vector of n dimensions ($\boldsymbol{S}_i$) and a vector $\boldsymbol{Q}_i$
(4) Carry out the following operations for each location i:
   a. Calculate the weight vector $w_i = (w_{i1}, \cdots, w_{in})$ from the corresponding distances of the observations from the location i;
   b. For estimating $\widehat{\boldsymbol{\beta}}_i$ calculate Equation (2) in two parts, i.e. Equation (12);

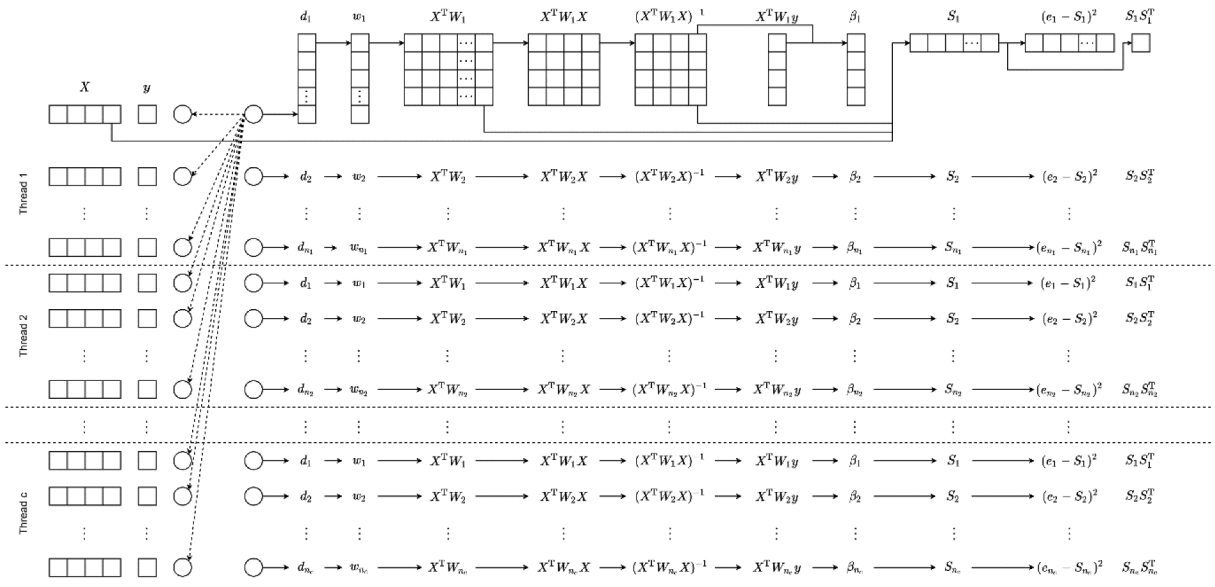

Figure 2. Parallel computing flowchart of the GWR-MP algorithm.

c. Calculate the i^{th} row of the hat matrix and assign it to $\boldsymbol{S}_i$ in memory, then renew the i^{th} element of $\boldsymbol{S}_2$ as $\boldsymbol{S}_2^i = \boldsymbol{S}_i\boldsymbol{S}_i^{\mathrm{T}}$;
d. Renew the vector $\boldsymbol{Q}_i = (e_i - \boldsymbol{S}_i)^2$.

(5) Repeat Step 4 until all the location-wise operations are finished, and the coefficient estimates $\widehat{\boldsymbol{\beta}}_{n\times m}$, $\mathrm{tr}(\boldsymbol{S})$, $\mathrm{tr}(\boldsymbol{S}^{\mathrm{T}}\boldsymbol{S})$ and $\tilde{\sigma}^2$ are ready for the final output.

By contrast, the parallelizing strategy for GWR-CUDA is designed specifically to fit GPU devices. As illustrated in Figure 3, the detailed procedure of GWR-CUDA includes the following steps:

(1) Read the data matrices or vectors (i.e. $\boldsymbol{X}$,$\boldsymbol{Y}$ and coordinates) from memory into GPU;
(2) Divide the n point-wise operations into groups, and within each group, g or fewer point-wise calibrations are conducted in parallel, where g should meet the following condition:

$$bgkn + 2bgkk + bgk \leq M - \alpha \tag{16}$$

where b is the number of bytes needed for each element in the matrix or vector (commonly set as 8), M is the GPU memory size, α is the memory reserved for intermediate calculations. In practice, the number of variables k is far less than the number of observations, n (i.e. $k \ll n$), the number of g is more dependent on the term, $bgkn$.

(3) Create arrays $\boldsymbol{\Delta}_{g\times k\times n}$, $\boldsymbol{\Omega}_{g\times k\times k}$ and a matrix $\xi_{g\times k}$, and conduct the following location-wise calibrations ($i = 1, \ldots, g$) in parallel within the current group:
a. Calculate the weight vector $w_i = (w_{i1}, \cdots, w_{in})$ from the corresponding distances of the observations from the location i;
b. Calculate $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}_i = \sum_{j=1}^{n} w_{ij}\boldsymbol{X}_j^{\mathrm{T}}$ and assign it to the i^{th} $k \times n$ component of Δ;
c. Calculate $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}(u_i, v_i)\boldsymbol{X} = \sum_{j=1}^{n} w_{ij}\boldsymbol{X}_j^{\mathrm{T}}\boldsymbol{X}_j$ and assign it to the i^{th} $k \times k$ component of $\boldsymbol{\Omega}$, and then calculate its inversion;
d. Calculate $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}(u_i, v_i)\boldsymbol{y} = \sum_{j=1}^{n} w_{ij}y_j\boldsymbol{X}_j^{\mathrm{T}}$ and assign it to the i^{th} row of ξ;
e. Calculate the location-wise coefficient estimate $\widehat{\boldsymbol{\beta}}_i$, $\boldsymbol{S}_i$, $\boldsymbol{S}_i\boldsymbol{S}_i^{\mathrm{T}}$ and $(\boldsymbol{e}_i - \boldsymbol{S}_i)^2$.

(4) Repeat Step 3 until parallel computations for all the groups are finished, and the coefficient estimates $\widehat{\boldsymbol{\beta}}_{n\times m}$, $\mathrm{tr}(\boldsymbol{S})$, $\mathrm{tr}(\boldsymbol{S}^{\mathrm{T}}\boldsymbol{S})$ and $\widehat{\sigma}^2$ will be ready for the final output.

It is important to note that both GWR-MP and GWR-CUDA procedures are designed to include the GWR model's diagnostic information calculations, with the calibration points the same as the observations and an assessment of model fit is required (which includes kernel bandwidth optimisation). Otherwise, the above procedures would be greatly simplified with only coefficient estimates returned. We implemented GWR-MP and GWR-CUDA in R, coded the parallel part via C++ and wrapped them via the Rcpp package (Eddelbuettel 2013).

3. Experimental design

3.1. General information

For this study, we compared the computational performances of GWmodel (version 2.1–4), FastGWR (updated on 12 August 2019), MGWR (version 2.1.1), GWR-MP and GWR-CUDA for

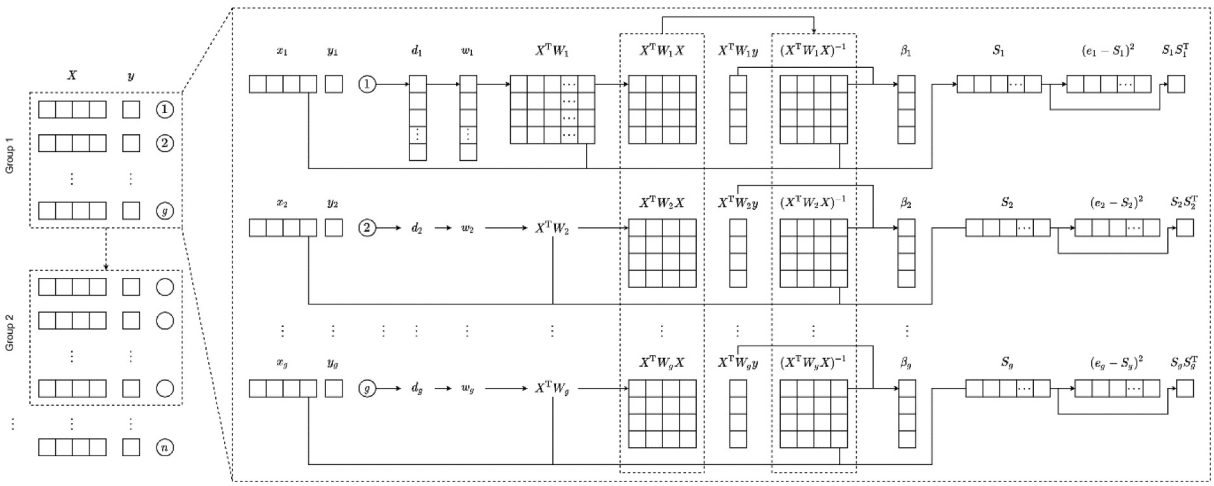

Figure 3. Parallel computing flowchart of the GWR-CUDA algorithm.

Table 1. Device parameters for running study GWR models.

	GWmodel, FastGWR, MGWR, GWR-MP, GWR-CUDA
Device model	ThinkSystem SR650
CPU	Intel(R) Xeon(R) Gold 5118 CPU @ 2.30 GHz, 24 cores
Memory	128 GB
Operating system	Ubuntu 20.04
GPU-1[a]	NVIDIA Tesla V100 PCIe, RAM 16 GB
GPU-2	NVIDIA GeForce RTX 2060 (Mobile)

[a]This version of GPU is the primary device for testing GWR-CUDA, and the results of running GWR-CUDA by default refer to using it; while the second device is only used for providing supplementary evidence.

implementations of conventional GWR only to ensure the same results. As shown in Table 1, we adopted two devices for running GWR where both devices had a GPU specification for running GWR-CUDA.

In terms of experimental data, we produced a series of simulated datasets of size n ranging from 1000 to 10,000 with increments of 1000, from 10,000 to 100,000 with increments of 10,000 and from 100,000 to 1,000,000 with increments of 100,000. For MGWR, FastGWR and GWR-MP, we specified a different number of cores ranging from 2 to 48 to run them in parallel. We didn't test them with all the combinations, but selectively adopted 2, 3, 4, 5, 6, 7, 8, 12, 24, 36 and 48 cores for typical tests. Note that the number of physical cores on the experimental device is 24, but the number of logic cores could be up to 48 through the hyper-threading technology. Notably, the current GWR routine in GWmodel is a serial program, so that the setting of multicores will not work differently for it. As shown in Figure 3, we adopted a different computing strategy for GWR-CUDA, where all samples are divided into groups, and location-wise calibrations within each group are conducted in parallel on the GPU device. Thus, the number g is the parallel computation counts for executing GWR-CUDA, and we took g = 384 (or less if samples were insufficient for the final group) according to Equation (16). For each sample size n and GWR implementation, 10 experiments were conducted independently on the two devices, respectively. Moreover, samples sizes ranging from 100,000 to 1,000,000 are adopted only for the extreme performance tests of GWR-CUDA, not for tests on the other four GWR solutions, as relatively inefficient solutions result due to unacceptably long time frames.

3.2. Performance indicators

Two indicators were used to evaluate performance – the average time cost and the "speedup" of the parallel computations. For each sample size n and GWR implementation, the average time cost was calculated as follows:

$$\bar{T}_{n,GWR_j} = \frac{\sum_{i=1}^{m} T^i_{n,GWR_j}}{m} \tag{17}$$

where T^i_{n,GWR_j} represents the time cost of running the j^{th} GWR implementation with a sample size n, m (in this case, taken as 10) is the number of individual experiments and $\bar{T}_{n,GWR_j}$ refers to the average time cost. Note that in all cases, the time costs include both the (automated) kernel bandwidth optimization (by AICc) and the GWR model calibration.

Speedup is an important indicator to evaluate the performance of parallel computations (Hill and Marty 2008). According to its original definition, we take a simple expression for its calculation, as:

$$k = \frac{T_S}{T_M} \tag{18}$$

where k is the speedup, T_S is the time cost of serial computing and T_M is the time cost of parallel computing with multi-cores. For this study, we repeated 100 independent experiments for each scenario, meaning speedup could be calculated using average time costs from m individual runs, i.e.:

$$k = \frac{\bar{T}_S}{\bar{T}_M} \tag{19}$$

where $\bar{T}_S$ and $\bar{T}_M$ are the average time costs of serial and parallel computations, respectively. We can verify that the estimation of speedup is significantly valid and reliable, by assuming the time cost for each individual test is a random variable subject to a normal distribution.

4. Results and discussion

In Figure 4, we present the averaged time costs of GWmodel, FastGWR, MGWR, GWR-MP and GWR-CUDA with a different number of cores with samples of sizes ranging from 1000 to 100,000. As the GWR implementation in GWmodel is a serial program, the time cost will not be affected by increasing cores, but averaged time costs grow exponentially as sample size increases. This indicates that the basic GWR function in the latest release of GWmodel is not working efficiently with a large dataset, even though the function

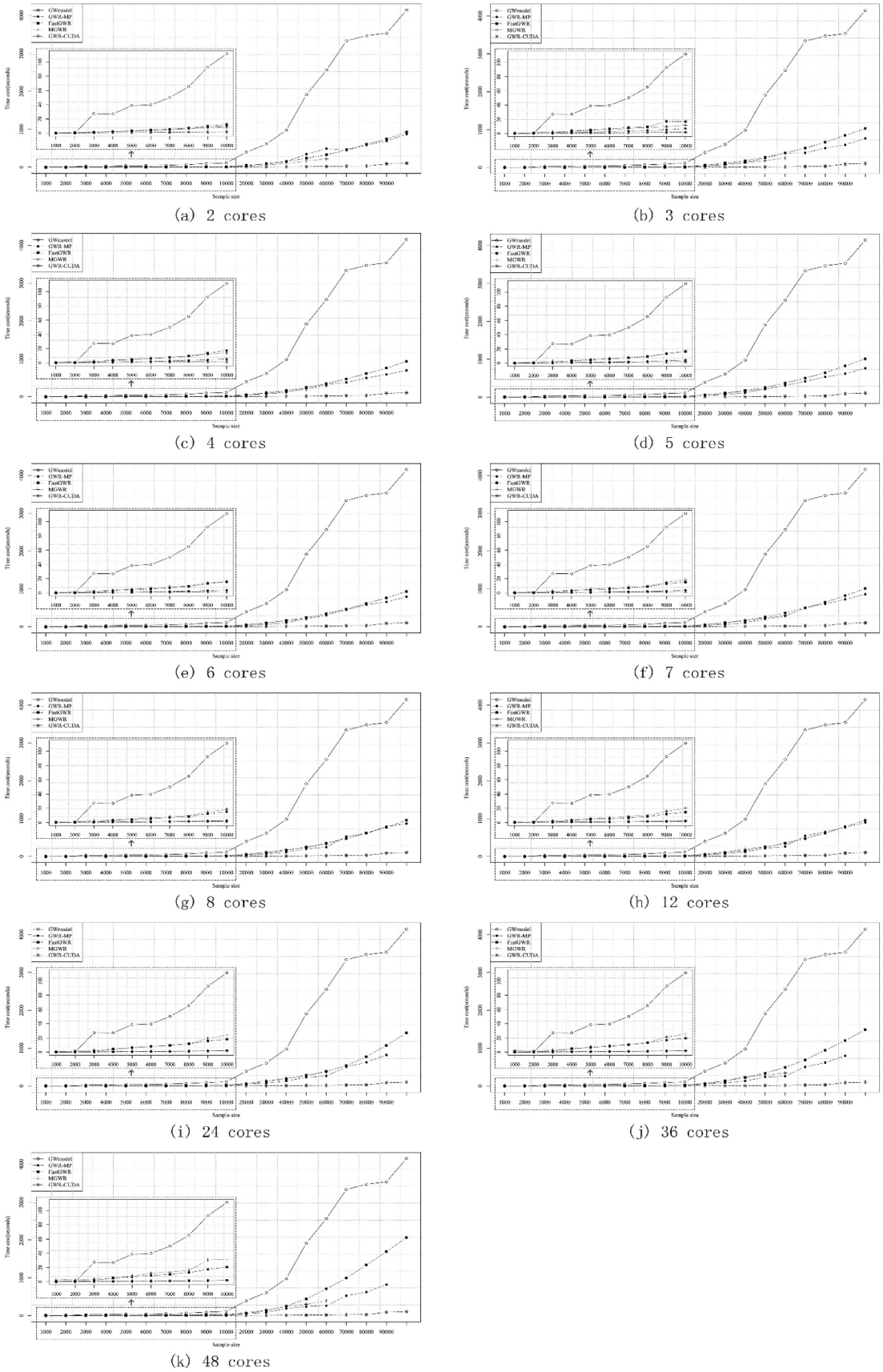

Figure 4. Average time costs of GWmodel, FastGWR, MGWR, GWR-MP and GWR-CUDA with a different number of cores.

benefits from algorithmic optimization and code implementation with C++. It can handle a relatively large dataset, say greater than 100,000, but its running time will be incredibly long, particularly when bandwidth optimization is additionally conducted. MGWR is applicable for running on multi-cores, but it failed

to calibrate the GWR models with sample sizes greater than 60,000. In the limited number of visible tests, it performed similarly to FastGWR, which is also developed via Python by the same research team (Oshan et al. 2019; Li et al. 2019a). Thus, results for MGWR can be represented by those for FastGWR and are not discussed further.

Both FastGWR and GWR-MP are naturally designed for multi-core parallelism. From Figure 4, FastGWR and GWR-MP always outperform the serial routine in GWmodel, and these advantages grow as the number of cores increase and as sample size increases. From Figure 4, GWR-MP performs similarly to FastGWR in most cases, but where GWR-MP tends to outperform FastGWR for samples greater than 60,000. Relative to GWR-MP, the time costs of FastGWR become exponentially large when the number of cores exceeds 24 for sample sizes greater than 60,000, which means that all the physical cores will be employed and the logic cores will be used via the hyper-threading technology. On this condition, the performance and stability of each physical core could worsen due to frequent switches between two logic cores on each physical core. In addition, FastGWR was developed with the Message Passing Interface (MPI), a standard and portable message-passing system for parallel programming (Dalcín et al. 2008). The MPI was originally designed for distributed memory systems, then extended to shared memory parallel computing for effectively utilizing node-level architecture (i.e. stand-alone machine with multi-cores). Its communication efficiency could be more or less affected by the memory capacity pressure, particularly when all the cores are fully occupied (Brinskiy, Lubin, and Dinan 2015). That could be the main reason of the relatively weak performance of FastGWR when the number of cores exceeds 24. From Figure 4, GWR-CUDA consistently performs the best of all across all scenarios.

For critically testing GWR-CUDA, we extend the size of samples up to one million with two different versions of GPU devices. In Figure 5, we present the average time costs of GWmodel and GWR-CUDA. The time consumption of GWmodel increases exponentially, particularly when the sample size is larger than 8000; in contrast, the time cost of GWR-CUDA grows much more slowly as sample size gets larger, but a dramatic increase occurs for sample sizes of around 800,000 or more. The two versions of GPU devices present different performances for running GWR-CUDA, dealing with samples of one million for around 3.5 h (12,766 s) on GPU-1, and around 5 h (17,828 s) on GPU-2. As one of the world's most advanced GPU, NVIDIA® Tesla® V100 (GPU-1) renders a great advantage over the GeForce RTX 2060 Mobile (GPU-2), a mobile graphics chip embedded in a laptop. Given that a laptop cannot run stably with full capacity for a long period, we only tested GWR-CUDA on GPU-2 with samples of sizes ranging from 1000 to 100,000, and

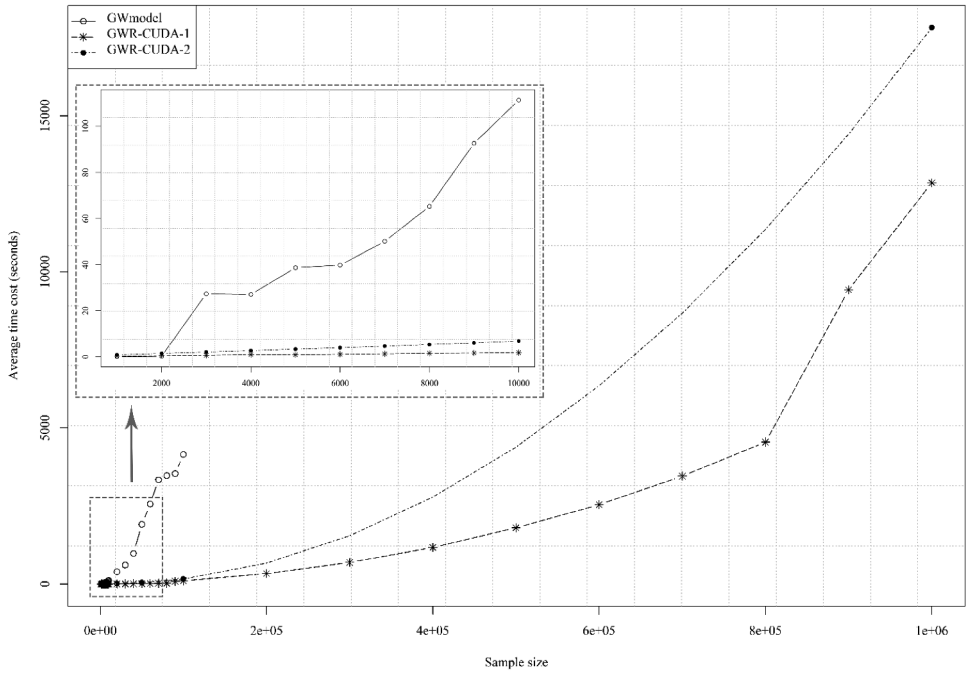

Figure 5. Average time costs of GWmodel and GWR-CUDA with different sample sizes.

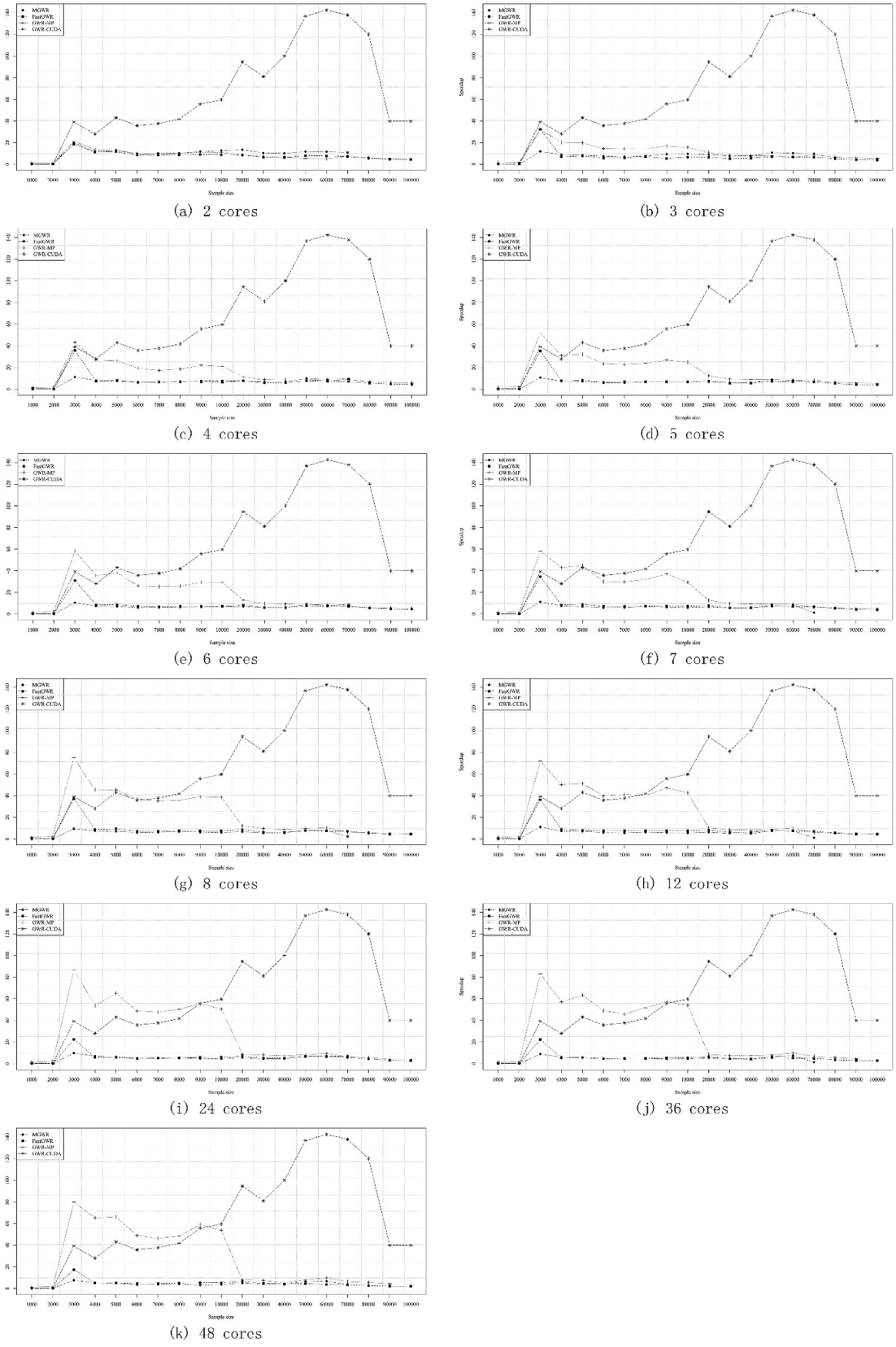

Figure 6. Speedup indicators of GWmodel, FastGWR, MGWR, GWR-MP and GWR-CUDA with a different number of cores.

1,000,000 only. Moreover, results indicate that the physical parameters of the CPU and the GPU device will affect the performances of the chosen high-performance solutions. Equipment (laptop or PC) with high-end CPU or GPU devices will provide better performances, and where High-

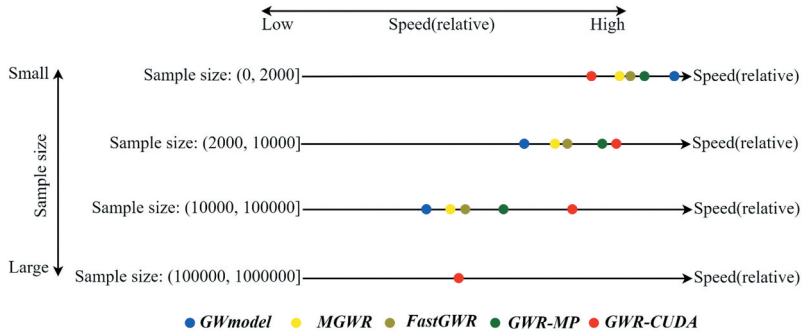

Figure 7. Overviews on the computational speeds of GWmodel, FastGWR, MGWR, GWR-MP and GWR-CUDA in terms of three different sample sizes.

Performance Computing (HPC) infrastructure should provide a better scope for potential improvement in this aspect.

The average time costs could be specific for the devices adopted, and almost impossible to be reproduced with a different device. From an objective assessment, we used the speedup indicator to evaluate how improvements benefitted from the parallel strategies implemented in FastGWR, MGWR, GWR-MP and GWR-CUDA. A larger speedup means the parallel solution for a specific GWR implementation makes a greater optimization in computational efficiency than running it serially. As shown in Figure 6, the performance of GWR-MP and FastGWR (and MGWR) improves as more cores are used for running them in parallel. Again GWR-MP demonstrates better usage of multi-core equipment for samples of sizes ranging from 3000 to 10,000, while performance does not show an improvement when the number of cores exceed 24, i.e. all the physical cores are fully occupied. For GWR-CUDA, its superiority in parallel performance is apparent when the sample size is greater than 10,000, but note that the speedup falls sharply to 40 when the sample size reaches 90,000. In Figure 5, we can see that the average time costs of GWR-CUDA also increase exponentially as the sample size becomes large, where a sharper increase occurs around a sample size of 80,000.

Note in the inset figure of Figure 5, GWR-CUDA is not always the best performer in comparison with the serial solution in GWmodel. GWR-CUDA takes more time than the serial solution when the sample size is less than 3000. To implement GWR-CUDA, all the predefined data matrices or vectors (i.e. $\boldsymbol{X}$,$\boldsymbol{Y}$ and coordinates) are transferred from memory into the GPU, and the results, including hat matrix $\boldsymbol{S}$ and coefficient estimates $\widehat{\boldsymbol{\beta}}$ are transferred from GPU back to memory – widely known as I/O issue important for GPU performance (Fujii et al. 2013). In other words, the I/O cost is predominant when the sample size is less than 3000, and the computational advantage of GWR-CUDA starts to emerge when the size is getting larger than 3000. The I/O cost could be affected by the physical parameters of GPU, CPU and protocol type, so GWR-CUDA will perform differently with different devices. Thus, the critical value (i.e. 3000 in this study) could fluctuate marginally width different computational configurations.

The results also reveal a fact that the high-performance solutions would be not be recommended for samples with relatively small sizes, say less than 5000, the most common data volume in the previous GWR applications. On the flip side, GWR applications with a relatively large data set (e.g. large than 20,000) were rarely found due to the lack of and universal access to high-performance tools. The findings in the context of rich scenarios are beneficial to both development and optimization of the high-performance solutions.

5. Summary

In this study, we have proposed two high-performance solutions for GWR via multi-core parallel and CUDA techniques: GWR-MP and GWR-CUDA, respectively. We objectively compared them with existing GWR implementations found in GWmodel, MGWR and another high-performance solution FastGWR. Results indicate that no solution was always the best in terms of computational efficiency, as summarized in Figure 7 by their relative speeds for four sample size intervals (less than 2000; greater than 2000, but less than 10,000; greater than 10,000 but less than 100,000; greater than 10,000). As (effectively) serial solutions, both GWmodel and MGWR provide adequate GWR implementations for (small) sample sizes < 10,000, as computational costs were considered acceptable.

For multi-core parallel solutions, GWR-MP provided a commensurate solution with GWR-CUDA for dealing with (large) sample sizes between 10,000 to 100,000 on a computer of common multi-core configuration, where GWR-MP demonstrated more efficient computing capacity

for each core or thread than FastGWR, whose design is more suited to non-shared memory clusters. For example, Li et al. (2019a) adopted FastGWR with a dataset of 1.28 million points on a 512-core computing cluster. However, high-performance computing clusters are usually too expensive and too few in number to be accessed by many researchers.

Conspicuously, GWR-CUDA provided a relatively cheap but highly efficient solution for analyzing a very large dataset, of which the size could be much larger than 1,000,000, the upper number in this study. The study GPU (NVIDIA GeForce RTX 2060 (Mobile)) only cost around $350, but we found we could implement a GWR model (including bandwidth optimization and model calibration) with one million data points in around 5 h. A better configuration of the GPU, like with NVIDIA Tesla V100 could reduce this time to 3.5 h, but at a cost of around $9000. Note however, GWR-CUDA should only be preferred when sample size is very large in terms of balancing cost with speed (as clearly seen in Figure 7). Note, however the Figure 7 roughly show the comparative performances of these solutions, and could more or less vary when different devices adopted.

Both GWR-MP and GWR-CUDA were implemented in R with wrappers on the C++ code, which has been incorporated into the latest release of GWmodel (say version GWmodel_2.2–8). Note that, nowadays it is straightforward to execute R from Python, and vice versa. Therefore, this is not a black-or-white type of choice to run these solutions in R or Python. Moreover, all the C++ code could be easily transferred to a standalone application, which we are currently working on. Inspired by Figure 7, an important feature of this, is to adaptively set a computational strategy according to sample size and the computing environment, and this study provides a direct support for such a strategic optimization. An ultimate solution could be an application developed under the service-oriented architecture with powerful computers or clusters, and the algorithms proposed here would provide fundamental support. Moreover, the solutions proposed here are directly applicable to extended GWR forms beyond the conventional GWR form, such as GTWR; and also directly applicable to other geographically weighted models (Lu et al. 2014b) outside of those for regression (e.g. GW PCA). Further, more pertinent issues, such as robust statistical inference in GWR with a massive data set (Griffith 2015) would also be worthy of investigation.

Notes

1. Note that the diagnostic information cannot be calculated when an individual set of regression locations are adopted.
2. Theoretically, the number of threads c could be larger than the number of cores available, but we would suggest creating no more than the number of cores for ensuring the performance of each thread.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This research was jointly supported by National Key Research and Development Program of China [grant number 2021YFB3900904] and the National Natural Science Foundation of China [grant numbers 42071368, U2033216, 41871287).

Notes on contributors

Binbin Lu is currently an Associate Professor at School of Remote Sensing and Information Engineering, Wuhan University. His research interests include geocomputation, spatial statistics, geographically weighted (GW) modeling, open-source GIS, R coding and spatio-temporal big data analysis. He is the main developer and maintainer of the R package, namely GWmodel.

Yigong Hu is currently a PhD student at University of Bristol, and got his master degree at the School of Remote Sensing and Information Engineering, Wuhan University. His research interests include spatial statistics, geoinformatics.

Daisuke Murakami is an Assistant Professor in the Department of Statistical Data Science at Institute of Statistical Mathematics, Tachikawa. His research interests include spatial and spatiotemporal statistics, Gaussian process, and regression modeling.

ChrisBrunsdon is a Professor of Geocomputation and Director of the National Centre for Geocomputation at Maynooth University, Ireland. His research interests include spatial statistics, data science and spatial analysis.

Alexis Comber is a Professor of Spatial Data Analytics at the University of Leeds and Leeds Institute for Data Analytics (LIDA). His research activities cover all areas of spatial data: remote sensing, land cover/use, demographics, public health, agriculture, bio-energy and accessibility.

Martin Charlton was an Associate Professor at the National Centre for Geocomputation at Maynooth University. He was one of the leading pioneers of quantitative geography and geocomputation whose work helped inspire the recent resurgence of spatial analysis and geographic data science.

PaulHarris is a Professor of Spatial Statistics at Rothamsted Research. His research includes methodological development with applied studies in agriculture and encompasses all scales (from the plot and field, to the continent and global).

ORCID

Binbin Lu http://orcid.org/0000-0001-7847-7560
Yigong Hu http://orcid.org/0000-0002-9553-6275

Daisuke Murakami http://orcid.org/0000-0002-9544-9542
Chris Brunsdon http://orcid.org/0000-0003-4254-1780
Alexis Comber http://orcid.org/0000-0002-3652-7846
Martin Charlton http://orcid.org/0000-0002-0622-393X
Paul Harris http://orcid.org/0000-0003-0259-4079

Data availability statement

The data that support the findings of this study are available with the identifier(s) at the link (https://figshare.com/s/13f325af1e37c3bc15fc).

References

Bivand, R., and D.L. Yu. 2006. "Spgwr: Geographically Weighted Regression." http://cran.r-project.org/web/packages/spgwr/index.html

Brinskiy, M., M. Lubin, and J. Dinan. 2015. *High Performance Parallelism Pearls:Chapter 16 - MPI-3 Shared Memory Programming Introduction*. Boston: Morgan Kaufmann.

Brunsdon, C., A.S. Fotheringham, and M.E. Charlton. 1996. "Geographically Weighted Regression: A Method for Exploring Spatial Nonstationarity." *Geographical Analysis* 28 (4): 281–98. doi: 10.1111/j.1538–4632.1996.tb00936.x.

Brunsdon, C., S. Fotheringham, and M. Charlton. 1998. "Geographically Weighted Regression-Modelling Spatial Non-Stationarity." *Journal of the Royal Statistical Society. Series D (The Statistician)* 47 (3): 431–443. doi:10.1111/1467-9884.00145.

Cao, K., M. Diao, and B. Wu. 2019. "A Big Data–Based Geographically Weighted Regression Model for Public Housing Prices: A Case Study in Singapore." *Annals of the American Association of Geographers* 109 (1): 173–186. doi:10.1080/24694452.2018.1470925.

Cao, X.S., Y.W. Liu, T. Li, and W. Liao. 2019b. "Analysis of Spatial Pattern Evolution and Influencing Factors of Regional Land Use Efficiency in China Based on ESDA-GWR." *Scientific Reports* 9 (1): 520. doi:10.1038/s41598-018-36368-2.

Cao, J.Y., F.S. Ma, J. Guo, R. Lu, and G.W. Liu. 2019a. "Assessment of Mining-related Seabed Subsidence Using GIS Spatial Regression Methods: A Case Study of the Sanshandao Gold Mine (Laizhou, Shandong Province, China)." *Environmental Earth Sciences* 78 (1): 26. doi:10.1007/s12665-018-8022-1.

Charlton, M., A.S. Fotheringham, and C. Brunsdon. 2003. *GWR 3: Software for Geographically Weighted Regression*. Maynooth, Co.kildare: National Centre for Geocomputation, National University of Ireland Maynooth.

Comber, A., C. Brunsdon, M. Charlton, G.P. Dong, R. Harris, B.B. Lu, Y.H. Lü, et al. 2022. "A Route Map for Successful Applications of Geographically Weighted Regression." *Geographical Analysis*. doi:10.1111/gean.12316.

da Silva, A.R., and A.S. Fotheringham. 2016. "The Multiple Testing Issue in Geographically Weighted Regression." *Geographical Analysis* 48 (3): 233–247. doi:10.1111/gean.12084.

Dalcín, L., R. Paz, M. Storti, and J. D'Elía. 2008. "MPI for Python: Performance Improvements and MPI-2 Extensions." *Journal of Parallel and Distributed Computing* 68 (5): 655–662. doi:10.1016/j.jpdc.2007.09.005.

Eddelbuettel, D. 2013. *Seamless R and C++ Integration with Rcpp*. New York: Springer.

Eddelbuettel, D. 2020. "Task View: High-Performance and Parallel Computing with R." *METACRAN*. Accessed 17 May. https://www.r-pkg.org/ctv/HighPerformanceComputing

ESRI Corp. 2011. *ArcGIS Desktop: Release 10*. Redlands, CA: Environmental Systems Research Institute.

Fotheringham, A.S., C. Brunsdon, and M. Charlton. 2002. *Geographically Weighted Regression: The Analysis of Spatially Varying Relationships*. Chichester: Wiley.

Fotheringham, A.S., M.E. Charlton, and C. Brunsdon. 1998. "Geographically Weighted Regression: A Natural Evolution of the Expansion Method for Spatial Data Analysis." *Environment & Planning A* 30 (11): 1905–1927. doi:10.1068/a301905.

Fotheringham, A.S., R. Crespo, and J. Yao. 2015. "Geographical and Temporal Weighted Regression (GTWR)." *Geographical Analysis* 47 (4): 431–452. doi:10.1111/gean.12071.

Fujii, Y., T. Azumi, N. Nishio, S. Kato, and M. Edahiro. 2013. "Data Transfer Matters for GPU Computing." The 2013 International Conference on Parallel and Distributed Systems, Seoul, December 15-18.

Gollini, I., B.B. Lu, M. Charlton, C. Brunsdon, and P. Harris. 2015. "GWmodel: An R Package for Exploring Spatial Heterogeneity Using Geographically Weighted Models." *Journal of Statistical Software* 63 (17): 1–50. doi:10.18637/jss.v063.i17.

Griffith, D.A. 2015. "Approximation of Gaussian Spatial Autoregressive Models for Massive Regular Square Tessellation Data." *International Journal of Geographical Information Science* 29 (12): 2143–2173. doi:10.1080/13658816.2015.1068318.

Harris, P., C. Brunsdon, B.B. Lu, T. Nakaya, and M. Charlton. 2017. "Introducing Bootstrap Methods to Investigate Coefficient Non-stationarity in Spatial Regression Models." *Spatial Statistics* 21: 241–261. doi:10.1016/j.spasta.2017.07.006.

Harris, R., A. Singleton, D. Grose, C. Brunsdon, and P. Longley. 2010. "Grid-enabling Geographically Weighted Regression: A Case Study of Participation in Higher Education in England." *Transactions in GIS* 14 (1): 43–61. doi:10.1111/j.1467-9671.2009.01181.x.

Hill, M.D., and M.R. Marty. 2008. "Amdahl's Law in the Multicore Era." *Computer* 41 (7): 33–38. doi:10.1109/mc.2008.209.

Huang, B., and J.H. Wang. 2020. "Big Spatial Data for Urban and Environmental Sustainability." *Geo-Spatial Information Science* 23 (2): 125–140. doi:10.1080/10095020.2020.1754138.

Huang, B., B. Wu, and M. Barry. 2010. "Geographically and Temporally Weighted Regression for Modeling Spatio-temporal Variation in House Prices." *International Journal of Geographical Information Science* 24 (3): 383–401. doi:10.1080/13658810802672469.

Ivan, I., A. Singleton, J. Horak, and T. Inspektor. 2017. *The Rise of Big Spatial Data: Lecture Notes in Geoinformation and Cartography*. Switzerland AG: Springer Nature.

Jin, C., J. Xu, and Z.F. Huang. 2019. "Spatiotemporal Analysis of Regional Tourism Development: A Semiparametric Geographically Weighted Regression Model Approach." *Habitat International* 87: 1–10. doi:10.1016/j.habitatint.2019.03.011.

Kalogirou, S. 2016. "Destination Choice of Athenians: An Application of Geographically Weighted Versions of Standard and Zero Inflated Poisson Spatial Interaction Models." *Geographical Analysis* 48 (2): 191–230. doi:10.1111/gean.12092.

Lee, J.G., and M. Kang. 2015. "Geospatial Big Data: Challenges and Opportunities." *Big Data Research* 2 (2): 74–81. doi:10.1016/j.bdr.2015.01.003.

LeSage, J., and R.K. Pace. 2009. *Introduction to Spatial Econometrics*. Chapman and Hall: CRC Press/Taylor & Francis Group.

Leung, Y., C.L. Mei, and W.X. Zhang. 2000. "Statistical Tests for Spatial Nonstationarity Based on the Geographically Weighted Regression Model." *Environment & Planning A* 32 (1): 9–32. doi:10.1068/a3162.

Li, Z.Q., and A.S. Fotheringham. 2020. "Computational Improvements to Multi-scale Geographically Weighted Regression." *International Journal of Geographical Information Science* 1–20. doi:10.1080/13658816.2020.1720692.

Li, Z.Q., A.S. Fotheringham, W.W. Li, and T. Oshan. 2019a. "Fast Geographically Weighted Regression (Fastgwr): A Scalable Algorithm to Investigate Spatial Process Heterogeneity in Millions of Observations." *International Journal of Geographical Information Science* 33 (1): 155–175. doi:10.1080/13658816.2018.1521523.

Li, Z.Q., T. Oshan, S. Fotheringham, W. Kang, L. Wolf, H. C. Yu, and W. Luo. 2019b. *MGWR 1.0 User Manual*. Tempe, USA: Arizona State University.

Liu, Y., N.Z. Zhao, J.K. Vanos, and G.F.U. Cao. 2019. "Revisiting the Estimations of PM2.5-attributable Mortality with Advancements in PM2.5 Mapping and Mortality Statistics." *Science of the Total Environment* 666: 499–507. doi:10.1016/j.scitotenv.2019.02.269.

Lu, B.B., M. Charlton, P. Harris, and A.S. Fotheringham. 2014a. "Geographically Weighted Regression with a non-Euclidean Distance Metric: A Case Study Using Hedonic House Price Data." *International Journal of Geographical Information Science* 28 (4): 660–681. doi:10.1080/13658816.2013.865739.

Lu, B.B., P. Harris, M. Charlton, and C. Brunsdon. 2014b. "The GWmodel R Package: Further Topics for Exploring Spatial Heterogeneity Using Geographically Weighted Models." *Geo-Spatial Information Science* 17 (2): 85–101. doi:10.1080/10095020.2014.917453.

Lu, B.B., W.B. Yang, Y. Ge, and P. Harris. 2018. "Improvements to the Calibration of a Geographically Weighted Regression with Parameter-specific Distance Metrics and Bandwidths." *Computers, Environment and Urban Systems* 71: 41–57. doi:10.1016/j.compenvurbsys.2018.03.012.

McMillen, D. 2015. "McSpatial-package." CRAN.

Murakami, D., N. Tsutsumida, T. Yoshida, T. Nakaya, and B. B. Lu. 2019. "Scgwr: Scalable Geographically Weighted Regression." CRAN.

Murakami, D., N. Tsutsumida, T. Yoshida, T. Nakaya, and B. B. Lu. 2020. "Scalable GWR: A Linear-Time Algorithm for Large-Scale Geographically Weighted Regression with Polynomial Kernels." *Annals of the American Association of Geographers* 1–22. doi:10.1080/24694452.2020.1774350.

Nakaya, T., M. Charlton, S. Fotheringham, and C. Brunsdon. 2009. *How to Use SGWRWIN (GWR4.0)*. Maynooth, Ireland: National Centre for Geocomputation.

Oshan, M.T., Z.Q. Li, W. Kang, J.L. Wolf, and S. A. Fotheringham. 2019. "MGWR: A Python Implementation of Multiscale Geographically Weighted Regression for Investigating Process Spatial Heterogeneity and Scale." *ISPRS International Journal of Geo-Information* 8 (6): 1–31. doi:10.3390/ijgi8060269.

R Core Team. 2020. "Memory Limits in R." Accessed 21 May 2021. https://stat.ethz.ch/R-manual/R-devel/library/base/html/Memory-limits.html

Tobler, W.R. 1970. "A Computer Movie Simulating Urban Growth in the Detroit Region." *Economic Geography* 46 (2): 234–240. doi:10.2307/143141.

Wang, D.C., Y. Yang, A. Qiu, X.C. Kang, J.K. Han, and Z. Y. Chai. 2020. "A CUDA-Based Parallel Geographically Weighted Regression for Large-Scale Geographic Data." *ISPRS International Journal of Geo-Information* 9: 11. doi:10.3390/ijgi9110653.

Wang, S.B., Y.L. Liu, C.Z. Zhao, and H.X. Pu. 2019. "Residential Energy Consumption and Its Linkages with Life Expectancy in Mainland China: A Geographically Weighted Regression Approach and Energy-ladder-based Perspective." *Energy* 177: 347–357. doi:10.1016/j.energy.2019.04.099.

Wheeler, D. 2013. "Gwrr: Fits Geographically Weighted Regression Models with Diagnostic Tools." R package version 0.2-1.

Xu, G., W.W. Wang, D.D. Lu, B.B. Lu, K. Qin, and L.M. Jiao. 2021. "Geographically Varying Relationships between Population Flows from Wuhan and COVID-19 Cases in Chinese Cities." *Geo-Spatial Information Science* 1–11. doi:10.1080/10095020.2021.1977093.

Yang, N.N., J.S. Li, B.B. Lu, M.H. Luo, and L.Z. Li. 2019. "Exploring the Spatial Pattern and Influencing Factors of Land Carrying Capacity in Wuhan." *Sustainability* 11 (10). doi:10.3390/su11102786.

Yin, C.H., Q.S. He, Y.F. Liu, W.Q. Chen, and Y. Gao. 2018. "Inequality of Public Health and Its Role in Spatial Accessibility to Medical Facilities in China." *Applied Geography* 92: 50–62. doi:10.1016/j.apgeog.2018.01.011.