# Fast Geographically Weighted Regression (FastGWR): a scalable algorithm to investigate spatial process heterogeneity in millions of observations

Ziqi Li, A. Stewart Fotheringham, Wenwen Li & Taylor Oshan

INTERNATIONAL JOURNAL OF GEOGRAPHICAL INFORMATION SCIENCE
2019, VOL. 33, NO. 1, 155–175
https://doi.org/10.1080/13658816.2018.1521523

# Fast Geographically Weighted Regression (FastGWR): a scalable algorithm to investigate spatial process heterogeneity in millions of observations

Ziqi Li [a], A. Stewart Fotheringham[a], Wenwen Li [a] and Taylor Oshan[b]

[a]Spatial Analysis Research Center, School of Geographical Sciences and Urban Planning, Arizona State University, Tempe, AZ, USA; [b]Center for Geospatial Information Science, Department of Geographical Sciences, University of Maryland, College Park, MD, USA

**ABSTRACT**
Geographically Weighted Regression (GWR) is a widely used tool for exploring spatial heterogeneity of processes over geographic space. GWR computes location-specific parameter estimates, which makes its calibration process computationally intensive. The maximum number of data points that can be handled by current open-source GWR software is approximately 15,000 observations on a standard desktop. In the era of big data, this places a severe limitation on the use of GWR. To overcome this limitation, we propose a highly scalable, open-source *FastGWR* implementation based on Python and the Message Passing Interface (MPI) that scales to the order of millions of observations. *FastGWR* optimizes memory usage along with parallelization to boost performance significantly. To illustrate the performance of *FastGWR*, a hedonic house price model is calibrated on approximately 1.3 million single-family residential properties from a Zillow dataset for the city of Los Angeles, which is the first effort to apply GWR to a dataset of this size. The results show that *FastGWR* scales linearly as the number of cores within the High-Performance Computing (HPC) environment increases. It also outperforms currently available open-sourced GWR software packages with drastic speed reductions – up to thousands of times faster – on a standard desktop.

**ARTICLE HISTORY**
Received 15 May 2018
Accepted 5 September 2018

**KEYWORDS**
Geographically Weighted Regression; GWR; parallel computing; statistical software; spatial analysis

## 1 Introduction

Geographically Weighted Regression (GWR) is a widely used tool for exploring potential spatial heterogeneity in processes over geographic space (Brunsdon *et al.* 1996, Fotheringham *et al.* 1996, 2002). It has been widely used in many areas such as climate science (Brown *et al.*, 2012), geology (Atkinson *et al.* 2003), criminology (Cahill and Mulligan 2007), transportation analysis (Cardozo *et al.* 2012), and house price modeling (Fotheringham *et al.* 2015). While a traditional global model assumes the processes generating the observed data are the same everywhere so that a single parameter is estimated for each covariate in the model, GWR allows this assumption to be relaxed by calibrating a model at each location to obtain location-specific parameter estimates for each process. This is achieved by 'borrowing' data using a distance-weighting scheme such that nearby

**CONTACT** Ziqi Li lziqi@asu.edu

data are weighted more heavily than data observed at locations farther away. In this way, local parameter estimates, associated local standard errors, and t-values can be obtained for each location, and these can be mapped to explore potential spatial variation in the processes that generated the data.

However, since GWR entails an individual regression for each calibration location, the calibration algorithm is computationally intensive. Many studies have explicitly reported the computational limitations of GWR (Osborne and Suárez-Seoane 2002, Griffith 2008), although these are typically much less than those of alternative frameworks for local modeling such as eigenvector filtering and Bayesian spatially varying coefficients models (Wolf *et al*. 2018). Harris *et al*. (2010) stated that GWR scales exponentially as the number of calibration locations increases, and that a GWR model with 100,000 observations may take two or more weeks to calibrate. Yu (2007) had to use a sub-sample of 68,906 records to calibrate a GWR model because it was computationally prohibitive to use all the observations. However, using a subset of the data has the obvious problems of potentially compromising the quality of the results and introducing sample bias. Feuillet *et al*. (2018) explored the use of GWR on a dataset with 40,480 observations to investigate spatial variation among walking-environment relationships but were limited from directly utilizing the entire dataset because of the computational burden. As a result, they split the study area into smaller geographic units, calibrated a separate GWR model for each geographic unit in parallel, and then subsequently pooled the results. However, this solution may not be equivalent to a full GWR model because the subsetted analyses will be subject to boundary and zoning effects that limit the ability to capture the true spatial variation in relationships. The challenges highlighted above demonstrate that traditional GWR tools begin to fail when the number of observations becomes excessively large (i.e. tens of thousands).

A few studies have made efforts to address the computational challenges in GWR from various perspectives. Harris *et al*. (2010) developed a Grid-enabled GWR within the open-source *spgwr* software package in the R environment. They leveraged the parallelizable nature of GWR by implementing an algorithm that divides the set of location-specific model calibrations across a set of nodes on a grid computing system. Despite this advance, a dataset containing 31,378 observations still required 1–2 h for model calibration. The modesty of this improvement despite utilizing 100 nodes on the grid is probably the result of the algorithm not parallelizing the calculation of the model diagnostics, which can require significant computational costs. Pozdnoukhov and Kaiser (2011) implemented a scalable local regression algorithm using a MapReduce parallelization framework to analyze streaming geo-referenced data. The algorithm is able to handle a large amount of streaming data for a relatively small number of locations, but it is severely limited in handling a large number of locations. Zhang (2010) proposed a theoretical parallel framework for running GWR on a Graphical Processing Unit (GPU) using a spatial indexing perspective, but no operational implementation was provided. Finally, Tran *et al*. (2016) implemented distributed GWR by leveraging Spark framework; however, this requires a cluster computing environment, and it is not easily accessible to regular users.

Despite these efforts, therefore, there is still a lack of an operational and accessible GWR algorithm that is able to analyze a large number of observations within a reasonable timeframe. As data are collected at increasingly finer spatial resolutions, datasets with hundreds of thousands of observations are becoming commonplace. Therefore, novel implementations of the GWR algorithm are needed to remove computational bottlenecks and to enable applications in very large datasets.

## 2 The basics of GWR

### 2.1 *Model formulation*

Geographically Weighted Regression (GWR) calibrates a separate regression model at each location through a data-borrowing scheme that distance-weights observations from the regression point. It is formulated as

$$y_i = \sum_{j=1}^{k} \boldsymbol{\beta}_{ij} x_{ij} + \varepsilon_i \tag{1}$$

where $y_i$ is the response variable at location *i*, $i \in \{1, 2, \ldots, n\}$, $x_{ij}$ is the jth predictor variable, $j \in \{1, 2, \ldots, \mathrm{k}\}$, $\boldsymbol{\beta}_{ij}$ is the jth parameter estimate, and $\varepsilon_i$ is the error term. GWR calibration in matrix form is given by

$$\hat{\boldsymbol{\beta}}_{\boldsymbol{i}} = \left(\boldsymbol{X}^T \boldsymbol{W}_{\boldsymbol{i}} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{W}_{\boldsymbol{i}} \boldsymbol{y} \tag{2}$$

where $\hat{\boldsymbol{\beta}}_i$ is a *k* by *1* row vector of parameter estimates at location *i*. ***X*** is an *n* by *k* matrix of predictor variables, ***y*** is an *n* by *1* matrix of the response variable, and $\mathbf{W_i}$ is an *n* by *n* diagonal spatial weighting matrix specific to location *i*, which is calculated based on a specified kernel function and bandwidth. The most widely used kernel functions are bisquare and Gaussian although other functions, such as a tri-cube kernel have been used (McMillen and McDonald, 2004). The equations for the bisquare and Gaussian kernel functions are given by:

$$\text{Bisquare } w_{ij} = \begin{array}{ll} \left[1 - \left(\dfrac{d_{ij}}{bw}\right)^2\right]^2 & \textit{if } d_{ij} < bw \\ 0 & \textit{otherwise} \end{array}$$
$$\text{Gaussian } \; w_{ij} = e^{-1/2\left(\frac{d_{ij}}{bw}\right)^2} \tag{3}$$

where $w_{ij}$ is the weight between locations *i* and *j*, $d_{ij}$ is the distance between *i* and *j*, and *bw* is the bandwidth parameter, which can be either the number of nearest neighbors (which generates a spatially adaptive bandwidth) or the distance of the neighborhood radius (which generates a spatially fixed bandwidth) (Fotheringham *et al*. 2002). The bandwidth is a crucial parameter in controlling the degree of smoothness in GWR. If the bandwidth is very large, the GWR results will be similar to those obtained from global OLS with little spatial variation in the parameter estimates. If the bandwidth is very small, estimated parameters will vary greatly over space but may have very large standard errors because of being estimated using very few data points. Finding the optimal bandwidth is therefore based on a bias-variance tradeoff. This is usually done by examining the corrected Akaike Information Criterion (AICc) score, which is given by

$$AIC_c = 2n\ln(\hat{\sigma}) + n\ln(2\pi) + n\left(\frac{n + tr(\boldsymbol{S})}{n - 2 - tr(\boldsymbol{S})}\right) \tag{4}$$

where $n$ is the number of observations, $\hat{\sigma}$ is the estimated standard error, and *tr(**S**)* is the trace of the hat matrix, which serves as the effective number of parameters (*ENP*) of the model and describes the model complexity. Other optimization criteria can also be used to generate the bandwidth such as AIC, Bayesian Information Criterion (BIC), or cross-validation (CV). A golden section search heuristic is usually used to efficiently search for the optimal bandwidth associated with lowest AICc score.

## 2.2 *The hat matrix in GWR*

The hat matrix *S*, also referred to as the projection matrix or influence matrix, is an $n$ by $n$ matrix that projects the response values onto the fitted values such that

$$\hat{\boldsymbol{y}} = \boldsymbol{S}\boldsymbol{y} \tag{5}$$

and is important in GWR because it is needed to compute the effective number of parameters (ENP), model diagnostics such as the AICc, and the inferential statistics such as local standard errors and local t-values. Each row $i$ of the hat matrix $\boldsymbol{S}$ is given by:

$$\boldsymbol{s_i} = \boldsymbol{x_i}\left(\boldsymbol{X}^T\boldsymbol{W_i}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{W_i} \tag{6}$$

Consequently, the full hat matrix, ***S,*** can be expressed as

$$\boldsymbol{S} = \begin{pmatrix} \boldsymbol{s}_1 \\ \cdots \\ \boldsymbol{s}_n \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}_1\left(\boldsymbol{X}^T\boldsymbol{W}_1\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{W}_1 \\ \cdots \\ \boldsymbol{x_n}\left(\boldsymbol{X}^T\boldsymbol{W_n}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{W_n} \end{pmatrix}_{n\times n} \tag{7}$$

The *ENP* of the model is then given by

$$ENP = tr(\boldsymbol{S}) \tag{8}$$

## 2.3 *Standard errors of the local parameter estimates*

The parameter estimates from calibrating a GWR model expressed in equation (2) may also be expressed as a function of the response variable $\boldsymbol{y}$ such that

$$\hat{\boldsymbol{\beta}}_i = \boldsymbol{C_i}\,y \tag{9}$$

where

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

In addition, let $\hat{\sigma}^2$ be the estimated variance defined by Yu *et al.* (2018) as

$$\hat{\sigma}^2 = \frac{\sum(y_i - \hat{y}_i)^2}{n - tr(\boldsymbol{S})} \tag{11}$$

Then the variance of the local parameter estimates at location $i$ is given by

$$Var\left(\hat{\boldsymbol{\beta}}_i\right) = diag\left(\boldsymbol{C_i}\boldsymbol{C_i}^T\right)\hat{\sigma}^2 \tag{12}$$

and the standard errors are obtained from:

$$SE\left(\hat{\boldsymbol{\beta}}_i\right) = sqrt\left[var\left(\hat{\boldsymbol{\beta}}_i\right)\right] \tag{13}$$

so that the local standard errors for all the parameter estimates are

$$SE\left(\hat{\boldsymbol{\beta}}\right) = \left[SE\left(\hat{\boldsymbol{\beta}}_1\right), SE\left(\hat{\boldsymbol{\beta}}_2\right) \ldots SE\left(\hat{\boldsymbol{\beta}}_{\boldsymbol{n}}\right)\right]^T_{n\times k} \tag{14}$$

and the local t-values can be computed as

$$t = \left(\frac{\hat{\boldsymbol{\beta}}}{SE\left(\hat{\boldsymbol{\beta}}\right)}\right)_{n\times k} \tag{15}$$

## 3. Computational issues in GWR calibration

### 3.1 *Time*

As indicated above, much of the GWR calibration routine involves computing matrix operations. For matrix multiplication, the time complexity for an $n$ by $m$ matrix multiplied by an $m$ by $k$ matrix is O($nmk$), where the big O is an asymptotic notation for describing the upper bound of an algorithm's efficiency. In calculating the local parameters $\hat{\boldsymbol{\beta}}_i$ in equation (2), the two most time-consuming computations are calculating $\boldsymbol{X}^T\boldsymbol{X}_i\boldsymbol{X}$ that takes O($k^2n$) and calculating its inverse $\left(\boldsymbol{X}^T\boldsymbol{W}_i\boldsymbol{X}\right)^{-1}$ that takes O($k^3$) where $n$ is the number of observations and $k$ is the number of predictor variables. Given that $k$ is normally far smaller than $n$, the time complexity for calculating $\hat{\boldsymbol{\beta}}_i$ is O($k^2n$), and this will be repeated at each location for a total of $n$ times. So the total time complexity in GWR for calibrating parameter surfaces is O($k^2n^2$) if the bandwidth is given. If golden section search is used to find the optimal bandwidth, which has a time complexity close to *log(n)*, bandwidth selection and model calibration will have a total time complexity of O($k^2n^2$log$n$). Harris *et al.* (2010) and Feuillet *et al.* (2018) both mention that GWR scales exponentially as the number of observations grows. However, given a detailed explanation of the time complexity of GWR as O($k^2n^2$log$n$), it scales in a quasi-quadratic fashion that is more efficient than an algorithm that scales exponentially.

### 3.2 *Memory*

Despite the time complexity that arises from repeated calculations for each calibration location, the primary bottleneck in a GWR calibration concerns the memory complexity incurred by storing the hat matrix and other model results. Suppose $n$ = 100,000 and each number is represented by a 32-bit float, the $n$ by $n$ hat matrix alone needs 38GB of memory, which currently exceeds or is very near to the limit of most desktop computers. Current open-source GWR software such as the Python-based MGWR module in Python Spatial Analysis Library (*PySAL*), or the R-based *GWmodel* (Gollini *et al.* 2013) and *spgwr* (Bivand *et al.* 2017) packages, store the entire $n$ by $n$ hat matrix in the memory. Harris *et al.* (2010) mention this memory requirement to store the hat matrix but do not report a solution.

## 4 Increasing computational efficiency in GWR calibration

Generally, GWR calibration takes place in two steps. The first step (section 4.1) involves finding the optimal bandwidth by minimizing a model diagnostic (e.g. AICc). The second step (section 4.2) involves the computation of local parameter estimates, standard errors and t-values, as well as some model diagnostics based on the optimal bandwidth. We now demonstrate how *FastGWR* is able to undertake both these steps without storing the entire hat matrix, significantly reducing the memory complexity from O($n^2$) to O($nk$) and making the calibration procedure accessible and efficient for very large datasets.

### 4.1 *Optimizing the AICc calculation for optimal bandwidth selection*

Consider the AICc formulation in equation (4), which can be expressed equivalently as

$$AIC_c = n\ln\left(\frac{RSS}{n - tr(\boldsymbol{S})}\right) + n\ln(2\pi) + n\left(\frac{n + tr(\boldsymbol{S})}{n - 2 - tr(\boldsymbol{S})}\right) \tag{16}$$

where RSS is the residual sum of squares and $tr(S)$ is the effective number of parameters. Note that *RSS* is calculated as

$$RSS = \sum_{i=1}^{n} \varepsilon_i{}^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{17}$$

where $\varepsilon_i$ is the residual at location $i$, $y_i$ and $\hat{y}_i$ are the observed and fitted values of the dependent variable at location $i$, and that

$$\hat{y}_i = \boldsymbol{x_i}\hat{\boldsymbol{\beta}}_i \tag{18}$$

where $\boldsymbol{x_i}$ is the *ith* row of the predictor matrix $\boldsymbol{X}$. It is also possible to substitute $\hat{\boldsymbol{\beta}}_i$ with the right-hand side of equation (9) to obtain

$$\hat{y}_i = \boldsymbol{x_i C_i y} \tag{19}$$

where

$$\mathbf{C_i} = \left(\boldsymbol{X}^T\boldsymbol{W_i X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{W_i} \tag{20}$$

Since the location-specific distance weight, $\boldsymbol{W_i}$, used to compute $\mathbf{C_i}$ is a sparse matrix with only diagonal non-zero elements, it can also be represented as a dense $1 \times n$ row vector such that

$$\boldsymbol{W_i} = \begin{pmatrix} w_{i1} & & & \\ & w_{i2} & & \\ & & \ddots & \\ & & & w_{in} \end{pmatrix}_{n\times n} \Rightarrow \boldsymbol{w_i} = \begin{pmatrix} w_{i1} & \ldots & w_{in} \end{pmatrix}_{1\times n} \tag{21}$$

Row vector $\boldsymbol{w_i}$ may have a different degree of sparsity depending on the bandwidth estimate and spatial kernel employed. $\boldsymbol{w_i}$ can be highly sparse if there is a small bandwidth and a bisquare kernel is used, since distant observations will be weighted to zero. In contrast, $\mathbf{w_i}$ can be dense if a large bandwidth is used, especially for nontruncated functions such as a Gaussian kernel. The computational costs may vary according to the sparsity of $\boldsymbol{w_i}$.

Let

$$\boldsymbol{P_i} = \mathbf{X}^{\mathbf{T}}\boldsymbol{W_i} = \begin{pmatrix} \boldsymbol{x}_1{}^{\boldsymbol{T}} \circ \boldsymbol{w_i} \\ \cdots \\ \boldsymbol{x_k}{}^{\boldsymbol{T}} \circ \boldsymbol{w_i} \end{pmatrix}_{k\times n} \tag{22}$$

where $\mathbf{x_j}^{\boldsymbol{T}}$ is a row vector of the jth column of the predictor matrix $\boldsymbol{X}$ and $\boldsymbol{x_j}^{\boldsymbol{T}} \circ \boldsymbol{w_i}$ is an element-wise multiplication of two row vectors of dimension $1 \times n$. Then $\boldsymbol{C_i}$ can be replaced by $\mathbf{C_i^*}$ in subsequent calculations where

$$\boldsymbol{C_i}^* \; = (\boldsymbol{P_i X})^{-1}\boldsymbol{P_i} \tag{23}$$

and the local squared residual $\varepsilon_i{}^2$ can be calculated as

$$\varepsilon_i{}^2 = (y_i - \hat{y}_i)^2 = (y_i - \boldsymbol{x_i C_i}^*\boldsymbol{y})^2 \tag{24}$$

The next step is to compute the effective number of parameters for GWR denoted in equation (8). Updating equation (7) to use $\mathbf{C_i^*}$ (equation 23), the hat matrix can also be equivalently expressed as

$$\boldsymbol{S} = \begin{pmatrix} \boldsymbol{x_1}\ \boldsymbol{C_1}^* \\ \cdots \\ \boldsymbol{x_n}\ \boldsymbol{C_n}^* \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & & \\ \cdots & & \ddots & \\ & r_{n1} & & r_{nn} \end{pmatrix}_{n\times n} \tag{25}$$

Let $r_{ii}$ be the *ith* element of the diagonal that can be computed directly as

$$r_{ii} = \boldsymbol{x_i c_i}^* \tag{27}$$

where $\mathbf{c_i^*}$ is the *ith* column of $\mathbf{C_i^*}$ in equation (23). Then the *ENP* can be computed much more efficiently as the sum the diagonal elements of the hat matrix such that

$$ENP = \sum_{i=1}^{n} r_{ii} \tag{26}$$

rather than as in equation (8). This is because, for each row of $\boldsymbol{S}$, we only calculate the element on the diagonal and ignore the off-diagonal elements since they are not used in the trace operator. In this way, we do not need to compute and store the entire hat matrix $\boldsymbol{S}$ to extract its diagonal elements. Moreover, in bandwidth selection, the goal is to find the bandwidth that minimizes the AICc. Therefore, further computational savings can be obtained by avoiding the calculation or storage of other model diagnostics (e.g. local standard errors) that are unnecessary for this step of the calibration routine.

In sum, given a predefined bandwidth, the steps to efficiently calculate the AICc are summarized in *Algorithm 1*.

*Algorithm 1*: Optimizing AICc calculation in bandwidth searching

```
1: Given a bandwidth bw
2: For location 1...n, at each location i, calculate:
3:     Local spatial matrix $w_i$ from equation (3) and (21)
4:     Local residual squared $\varepsilon_i{}^2$ from equation (24)
5:     Local diagonal element of hat matrix $r_{ii}$ from equation (27).
6: End for
7: Calculate RSS from equation (17)
8: Calculate ENP from equation (26)
9: Calculate AICc from equation (16)
```

As discussed above, parallelization is a straight-forward and natural way to reduce the computation time of a GWR model calibration (Harris *et al.* 2010). However, to more effectively reap the benefits of parallelization, it is important to also consider how the model diagnostics are computed. Since the additive method to compute *ENP* presented here lends itself to parallelization, as does the computation of the *RSS*, it is possible to much more efficiently parallelize the entire GWR calibration. Compared with the conventional approach that involves storing the entire *n* by *n* hat matrix ***S***, the memory overhead in this algorithm is reduced since the largest stored matrix is the *n* by *k* design matrix of predictors, which only results in a modest memory complexity of O(*kn*).

## 4.2 Optimizing the local standard error calculation

Once the optimal bandwidth is found (e.g. by golden-section search), the local standard errors are computed as

$$SE\left(\hat{\boldsymbol{\beta}}_i\right) = sqrt\left[diag\left(\mathbf{C}_i{}^{*}\ \mathbf{C}_i^{*\mathbf{T}}\right)\hat{\sigma}^2\right] = sqrt\left[diag\left(\mathbf{C}_i{}^{*}\ \mathbf{C}_i^{*\mathbf{T}}\right)\right]\hat{\sigma} \tag{28}$$

where $\mathbf{C}_\mathbf{i}^{*}$ is from equation (23). Let

$$\boldsymbol{h}_\mathbf{i} = sqrt\left[diag\left(\mathbf{C}_i{}^{*}\mathbf{C}_i{}^{*T}\right)\right] \tag{29}$$

so that

$$SE\left(\hat{\boldsymbol{\beta}}_i\right) = \boldsymbol{h}_\mathbf{i}\hat{\sigma} \tag{30}$$

where

$$\hat{\sigma} = sqrt\left[\frac{\sum(y_i - \hat{y}_i)^2}{n - tr(\boldsymbol{S})}\right] = sqrt\left[\frac{RSS}{n - tr(\boldsymbol{S})}\right] \tag{31}$$

And the *RSS* and $tr(\mathbf{S})$ can be computed from equations (16) and (25), respectively.

The details of the algorithm to efficiently compute the standard errors are summarized in *Algorithm 2*.

*Algorithm 2*: Optimizing the local standard error calculation

```
1: Given the optimal bandwidth $bw$
2: For location 1 . . . $n$, at each location $i$, calculate:
3:      Local parameter estimates $\hat{\boldsymbol{\beta}}_i$ from equation (9)
4:      Local residual squared $\varepsilon_i{}^2$ from equation (24)
5:      Local diagonal element of hat matrix $r_{ii}$ from equation (27)
6:      $\boldsymbol{h}_i$ from equation (29)
7: End for
8: Calculate $RSS$ from equation (17)
9: Calculate $ENP$ from equation (26)
10: Calculate $\hat{\sigma}^2$ from equation (31)
11: Calculate $SE\left(\hat{\boldsymbol{\beta}}\right)$ from equation (30)
```

As with the AICc calculation, the local standard error calculation can also be easily parallelized since the regression at each calibration location is independent of all the others. The base memory complexity of the above algorithm is O(*kn*), while the base time complexity is O($k^2n^2$). If golden section search is used for bandwidth selection, then the time complexity of the AICc calculation is increased by O(log$n$), which results in a total time complexity of O($k^2n^2$log$n$).

## 5 *FastGWR*: a parallel implementation using Python and MPI

The computational enhancements described above are implemented in a fully parallelizable GWR algorithm called *FastGWR*. The implementation is carried out using Python, which is a high-level, open-source programming language. For the parallelization of the algorithm, we use Message Passing Interface (MPI), which is a common protocol for facilitating communication across cores and nodes of parallel computing architectures (Gropp *et al*. 1996) and has been widely adopted for scientific computing in many research fields (Schmidt *et al*. 2002, Neese 2012, Wu *et al*. 2013). The primary reason for choosing MPI over other alternatives, such as MapReduce on Hadoop, is that MPI-based programs can be executed on both computers with a single CPU and HPC clusters. There are many MPI implementations available, and for convenience we used OpenMPI[1] that is an open-source implementation (Gabriel *et al*. 2004). We also used a Python wrapper for MPI called *mpi4py*[2] (Dalcin *et al*., 2008) to directly integrate MPI into the *FastGWR* algorithm. A summary of the *FastGWR* parallel implementation is described in *Algorithm 3*.

*Algorithm 3*: MPI-based parallel *FastGWR* implementation

```
1: The root processor $P_0$ reads in data from a disk. Assuming there are $m$ processors available, $P_0$ broadcasts the data
   to processor $P_{1...m}$, so each processor has a copy of the data.
2: The local regressions are divided evenly across the $m$ processors, and each processor calculates $1/m$ proportion of
   them.
3: In the golden section search for the current $bw_j$:
4:     $P_0$ broadcasts $bw_j$ to $P_{1...m}$
5:     For each processor $p$ in $P_{0...m}$:
6:         Calculate $1/m$ portion of $RSS$, $\sum_{1/m} \varepsilon_i^2$
7:         Calculate $1/m$ portion of $ENP$, $\sum_{1/m} r_{ii}$
8:     End for
9:     Root $P_0$ gathers $\sum_{1/m} \varepsilon_i^2$ and $\sum_{1/m} r_{ii}$ from $P_{1...m}$
10:    Calculate AICc for $bw_j$
11: Once the optimal bandwidth is found, run another iteration, for each processor $P_{1...m}$:
12:    Calculate $1/m$ portion of parameter estimates, $\left[\hat{\beta}\right]_{1/m}$
13:    Calculate $1/m$ portion of $RSS$, $\sum_{1/m} \varepsilon_i^2$
14:    Calculate $1/m$ portion of $ENP$, $\sum_{1/m} r_{ii}$
15:    Calculate $1/m$ portion of $[h]_{1/m}$
16:    End for
17: Root $P_0$ gathers $\sum_{1/m} \varepsilon_i^2$, $\sum_{1/m} r_{ii}$, $\left[\hat{\beta}\right]_{1/m}$, and $[h]_{1/m}$ calculated from $P_{1...m}$
18: Calculate $SE\left(\hat{\beta}\right)$
19: Output $\hat{\beta}$, $SE\left(\hat{\beta}\right)$ to disk
```

## 6 *FastGWR* as an open-source program

A script to run the *FastGWR* software routine and documentation of the development of the source code are available via GitHub.[3] Once the software is obtained, an example call to the *FastGWR* program is as follows:

***mpiexec – np 32 python fastgwr-mpi.py – data input.csv – out gwr.csv – a -bw 1000***

where ***mpiexec*** is the command to execute an MPI-based program; argument – ***np*** 32 indicates the number of processors to allocate; – ***data*** *input.csv* is the name of the input data table containing coordinates and associated dependent and independent variables; ***-out*** gwr.csv is the file containing the GWR outputs that include an ID for each calibration location, predicted values, residuals, local parameter estimates, and local standard errors; ***-a*** (***-f***) indicates the use of an adaptive (fixed) bandwidth using a bisquare (Gaussian); and – ***bw*** 1000 indicates a user-defined bandwidth for calibrating the GWR model. If the argument – ***bw*** is not given, the program will default to searching for an optimal bandwidth using a golden section search heuristic and an AICc selection criterion to calibrate the GWR model. Since *FastGWR* and *MGWR (PySAL)* are both written in Python, it is possible to integrate the advantages of *FastGWR* into the *MGWR (PySAL)* implementation that can be accessed via either a user-friendly command-line API or a graphical user interface. However, this is left for future work.

## 7. An empirical example

### 7.1 *Data and model*

To illustrate the performance of *FastGWR*, we use a Zillow property dataset containing over a million geo-referenced properties in the city of Los Angeles. Zillow is an online real estate company with property data across the entire United States. In 2017, Zillow held a data science competition aimed at helping them improve their property value estimation algorithm, *Zestimate*. As a result, the data were published on Kaggle.com[4] and are freely available to download. From this database, we select a subset of data consisting of single-family housing having one or more bedrooms and one or more bathrooms within the metropolitan area of Los Angeles, which results in approximately 1.28 million geo-referenced properties. Figure 1(a) shows the spatial distribution of the selected properties. Solely for the purposes of visualization, we aggregate the properties to a 1km by 1km grid covering the city of Los Angeles and display the number of properties within each grid cell with darker shading representing larger numbers of properties. Based on the attributes of the data, we constructed a simple hedonic house price model as:

$$Value_i = \boldsymbol{\beta}_{i0} + \boldsymbol{\beta}_{i1}Area_i + \boldsymbol{\beta}_{i2}NBaths_i + \boldsymbol{\beta}_{i3}NBeds_i + \boldsymbol{\beta}_{4i}Age_i + \varepsilon_i \tag{32}$$

where $Value_i$ is the assessed value of the property at location $i$ expressed in thousands of US dollars provided by Zillow; $Area_i$ is the total finished living area; $NBeds_i$ is the number of bedrooms; $NBaths_i$ is the number of bathrooms; and $Age_i$ is the age of the property in years as of 2017. Before conducting a GWR analysis, we checked that the variance inflation factor (VIF) of each covariate in the global model is less than 4, to ensure that strong multicollinearity is not present.

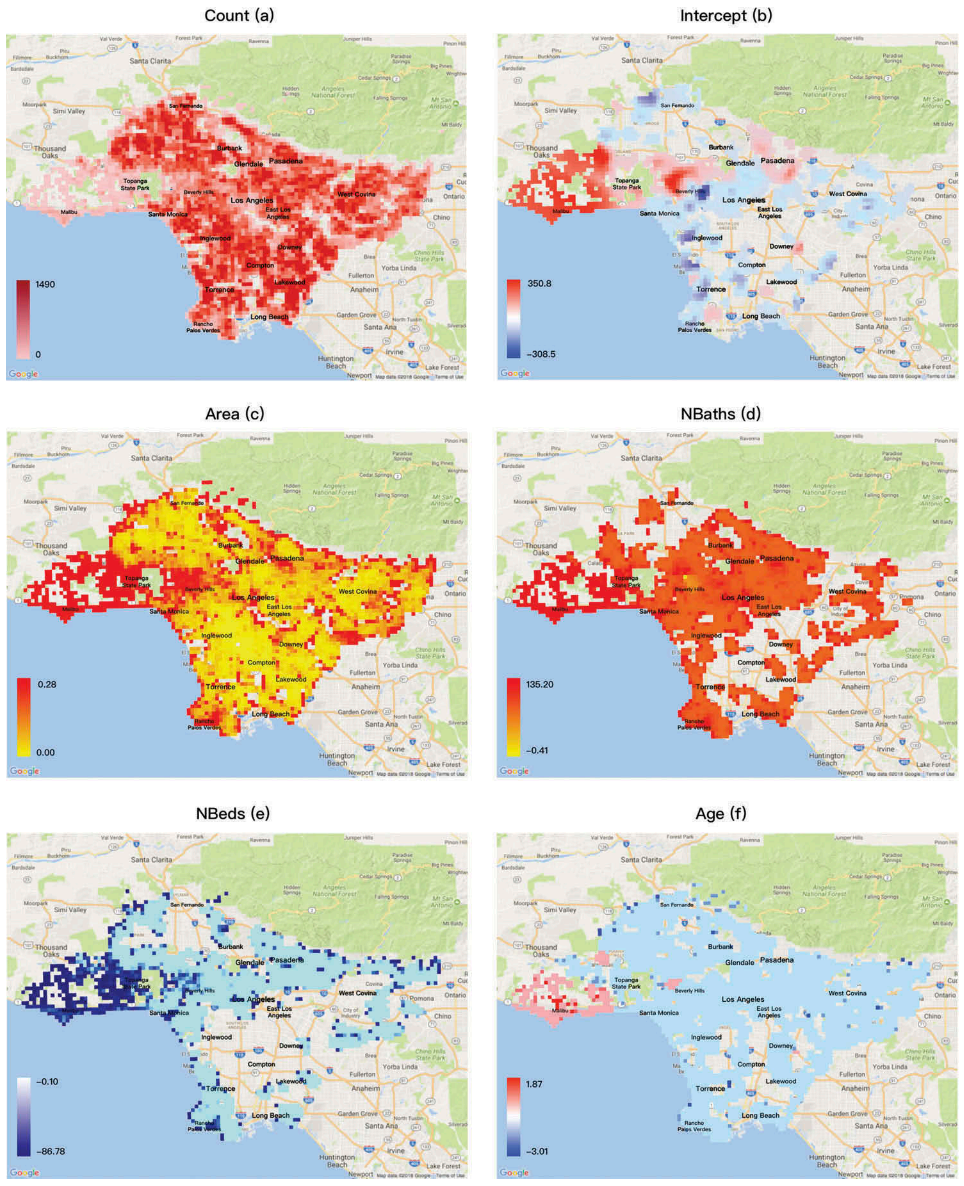

**Figure 1.** (a) Locations of properties in the city of Los Angeles, California. The color represents the number of properties within each 1km by 1km grid cell. (b)–(e) Maps of significant parameter estimates for the predictor variables *Intercept*, *Area*, *NBeds*, *Nbaths*, and *Age*. Parameter estimates are aggregated into 1km by 1km grids with values displayed indicating the average of the local parameter estimates falling into each grid-cell.

### 7.2 Testing environment and specifications

The calibration of the model in equation (32) using the *FastGWR* algorithm was undertaken on the Agave HPC Cluster managed by Arizona State University Research Computing. It has 268 compute nodes, each with 28 Intel Broadwell CPU cores with 128 GB of Random Access Memory (RAM). The computers are connected using Inter

Omni-Path, which is a high-throughput, low-latency network. Python and any additional dependencies were installed using the Anaconda3 version 4.4.0 installer, and version 3.0.0 of OpenMPI was utilized. The single-CPU desktop used for comparing the performance of other packages is equipped with an Intel i7–4790 3.60Ghz 4-core CPU and 16GB of RAM. Note that Python was installed on this machine using the Anaconda3 version 4.3.34 installer, rather than version 4.4.0, though the version of MPI used is the same as above. The differences in versions between the Python installers are minimal and are not problematic for the results we subsequently present since we only compare performances across a single machine and not between machines.

## 7.3 *Results*

We first describe the results of calibrating both an OLS model and a GWR model using the specification from equation (32) on the 1.28 million observations, and then we compare performance times using *FastGWR* versus other existing algorithms. Only *FastGWR* allows the calibration of a GWR model on 1.28 million observations so the comparison against other GWR algorithms is based on various subsets of the data.

### 7.3.1 *OLS and GWR calibration for 1.28 million house price observations*

Table 1 summarizes the parameter estimates and associated t-values for the predictor variables generated from an OLS and a GWR model, respectively. Unsurprisingly with 1.28 million observations, all the parameter estimates in the OLS model are highly significant based on their t-values. The variables *Area* and *NBaths* both have positive associations with *Value*, indicating larger properties with more bathrooms tend to have higher prices, *ceteris paribus*. In contrast, the variables *NBeds* and *Age* have negative associations with *Value*, indicating that newer properties and those with fewer bedrooms given the size of the property (i.e. properties with larger living areas) tend to be higher priced.

In GWR, the location-specific parameter estimates (and associated t-values) are highly varied across locations within the city of Los Angeles. The optimal bandwidth in GWR is close to 1% of the total observations. Figure 2 shows maps of the local parameter estimates for *Area, NBeds, NBaths, and Age*. Due to the challenges of visualizing 1.2 million location-specific parameter estimates, we aggregate them into 1km by 1km grid cells. Consequently, the values displayed in Figure 2 are the average significant parameter estimates within each grid cell and insignificant parameter estimates are not

**Table 1.** Comparison of parameter estimates between OLS and GWR.

| | OLS | | GWR | |
|---|---|---|---|---|
| | Beta | t | Beta range | t range |
| Intercept | −43.35 | −54.23 | −312.62 ~ 373.22 | −39.48 ~ 32.82 |
| Area | 0.17 | 666.94 | 0.01 ~ 0.37 | 2.71 ~ 240.99 |
| NBaths | 21.72 | 87.20 | −45.96 ~ 136.65 | −25.63 ~ 63.85 |
| NBeds | −34.56 | −184.31 | −107.94 ~ 9.75 | −56.82 ~ 4.06 |
| Age | −0.47 | −63.69 | −6.44 ~ 2.03 | −58.20 ~ 20.55 |
| N | 1,276,889 | | 1,276,889 | |
| ENP | 5 | | 1357.1 | |
| Adj. $R^2$ | 0.562 | | 0.686 | |
| AICc | $1.63 \times 10^7$ | | $1.59 \times 10^7$ | |

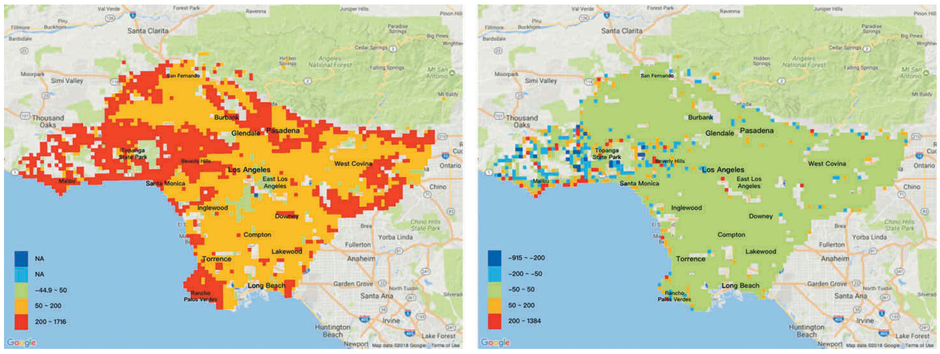

**Figure 2.** Maps of Residuals from OLS (left) and GWR (right).

included in the visualizations. Note that hypothesis testing was carried out using the correction of da Silva and Fotheringham (2016) to account for multiple testing (adjusted critical t-value = 3.74). The parameter estimates for *Area* can be considered as a proxy for property value in dollars per square foot, adjusted for the effects of the remaining covariates. As can be seen in Figure 2(c), GWR is able to capture a spatially varying 'unit price' effect with high unit price hotspots in Beverly Hills, Malibu, Pasadena and Rancho Palos Verdes and price cold spots in South Los Angeles, San Fernando, and West Covina. In Figure 2(d), the number of bathrooms generally has a significant positive effect on house value, except for the south and east part of LA where the number of bathrooms is not significant. In Figure 2(e), the parameters for the number of bedrooms ate not significant in most of the eastern half of the city, though where they are significant, they indicate a negative effect on house values. For the Malibu region, in particular, an increase in bedrooms will decrease house value, indicating that people may prefer a larger living area, *ceteris paribus*. In Figure 2(f), it can be seen that older properties are associated with lower property values across the city of LA except for the Malibu region where older properties are more expensive, *ceteris paribus*. One caution in interpreting the parameter maps presented here is that there seems to be a local multicollinearity or concurvity issue between *NBeds* and *NBaths*, despite the VIFs from the global model all being less than 4. These two parameter estimate surfaces display inverse trends such that one is high where the other is low and vice versa and several potential lines of investigation could be pursued to examine this further: determine if a variable should be removed based on local measures of multicollinearity (Wheeler 2007); apply regularized versions of GWR that can reduce the variance in some parameter estimates but can increase bias in others (Wheeler 2009); or decrease the chance of bandwidth misspecification by adopting Multi-scale GWR (Fotheringham *et al*. 2017). However, since the main purpose of this empirical study is to demonstrate the computational power of *FastGWR*, we do not pursue this issue here.

In terms of model diagnostics, although GWR uses more effective parameters than OLS (1357.1 vs. 5), the AICc for GWR is smaller than that from OLS ($1.59 \times 10^7$ vs. $1.63 \times 10^7$) and the adjusted $R^2$ for GWR is larger than that or OLS (0.562 vs. 0.686), both indicating GWR achieves a better model fit. Maps of the residuals from OLS (left) and GWR (right) are shown using the same color scheme in Figure 2. The OLS residuals are much larger than the GWR residuals, and they are spatially clustered, providing further evidence that GWR is a more appropriate specification than OLS for this house price model.

### 7.3.2 The scalability of *FastGWR*

This section examines the sensitivity of the performance of *FastGWR* to the number of cores utilized within an HPC environment. Based on the 1.28 million house price observations, we calibrated the same GWR model given in equation (32) with the optimal bandwidth while varying the number of cores. As can be seen in Figure 3 (left), as the number of cores increases from 32 to 512, the computation time decreases linearly and is approximately halved with each doubling of the number of available cores. To further examine this scaling relationship, Figure 3 (right) compares the empirical speed-ups of *FastGWR* against the theoretical speed-ups for a linear relationship. Here speed-ups are based on the reduction of computation time for 64, 128, 256, and 512 cores compared to 32 cores. The orange bars are the theoretical levels, while the blue bars depict the measured speed-up factor using *FastGWR*. The obvious similarity in the empirical and theoretical speed-up factors indicates that the *FastGWR* algorithm has little communication overhead across an increasing number of cores and nodes.

### 7.3.3 A comparison of computation time for *FastGWR* and other open-source GWR software on a single desktop

HPC can be costly and is not always available for every research project. However, the *FastGWR* algorithm can also offer advantages when deployed on a single desktop that has multiple cores, a scenario that is now commonplace. Consequently, we can compare the performance of *FastGWR* to other open-source GWR software. Specifically, we compare computation time across four GWR software packages,[5] *FastGWR, MGWR (PySAL)*,[6] *GWmodel*,[7] and *spgwr*[8], using differently sized subsets of the 1.28 million house price observations described above. To do this, we randomly select subsets of the Zillow data and examine the performance of the four GWR packages in calibrating the house price model specified in equation (32). The computation times[9] shown in Table 2 and Figure 4 are based on house price model calibrations using a predefined bandwidth on the single-CPU desktop specified in section 7.2. Computational times for optimal bandwidth searching are not compared because searching is a repetitive approach that evaluates a set of calibrations using different predefined bandwidths and the processes are the same throughout the implementations compared here. Note that due to the fact that *spgwr* took such a large amount of time, it was necessary to use a log scale for the runtime displayed on the y-axis. The result shows that for 10,000 observations, *FastGWR* is approximately 12 times faster than *MGWR (PySAL)*, approximately 160 times faster than *GWmodel*, and approximately 3400 times faster than *spgwr*. Furthermore, when the data are increased to 15,000 observations, *FastGWR* is approximately 14 times faster than *MGWR (PySAL)*, approximately 348 times faster than *GWmodel*, and *spgwr* was not be able to yield results. Once the number of observations is increased to 20,000, *FastGWR* is the only software that is able to successfully calibrate a GWR model, and this is due to the memory bottleneck caused by storing the full hat matrix in the other software routines.

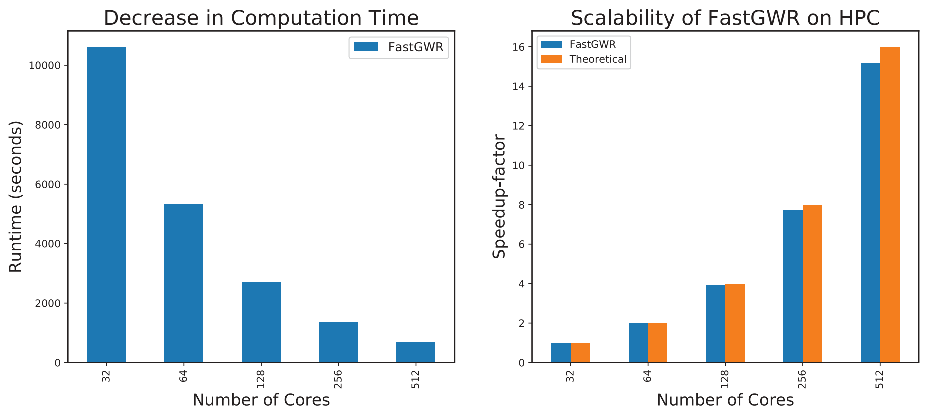

**Figure 3.** Scalability of *FastGWR* for an increasing number of cores.

Figure 5 further compares the scalability of the four GWR implementations as the number of observations grows. Because of the large amount of variation in computation time across the four implementations, it is useful to visualize differences using several subplots. Each subplot in Figure 5, for example, depicts runtime on the vertical axis and the number of observations on the horizontal axis but each has a different range of run times on the vertical axis. Figure 5(a) compares all four software packages and clearly demonstrates the limitation of *spgwr* in terms of producing acceptable runtimes, and so this package is not included in the comparison shown in Figure 5(b). Here it is now more clearly seen that *GWmodel* produces much slower runtimes compared to the other two packages. Finally, in Figure 5(c), a comparison is made solely between *FastGWR* and *MGWR (PySAL)*. This final plot clearly demonstrates that *FastGWR* is the most efficient of all the software implementations.

### 7.3.4 Validation of *FastGWR* parameter estimates

To provide assurance that *FastGWR* results are in alignment with those from other GWR implementations, we compare parameter estimates computed from all four open-source GWR packages based on the well-known ‘Georgia’ dataset often used for examples in *MGWR(PySAL), GWmodel*, and *spgwr*. The same GWR model was calibrated using *PctBach* as the dependent variable, and *PctPov, PctRural*, and *PctBlack* as the independent variables, respectively, for each package. The optimal bandwidth estimated by all four packages is 93 nearest neighbors when using a bisquare kernel and golden-section search. Means and standard deviations of local parameter estimates computed based on that optimal bandwidth are summarized in Table 3. As shown in Table 3, the results from *FastGWR* are the same as results from the other three packages confirming that *FastGWR* is a reliable implementation of GWR.

### 7.3.5 *Memory usage reduction*

The improvements that allow *FastGWR* to be applied to very large datasets are not only concerned with speed-ups in computational time but also result in the reduction of

**Table 2.** Runtime (in seconds) for four open-source GWR packages using different numbers of data points on a single desktop computer.

| Number of Data Points | FastGWR | MGWR (PySAL) | GWmodel | spgwr |
|---|---|---|---|---|
| 500 | 0.05 | 0.33 | 0.38 | 1.75 |
| 1,000 | 0.09 | 0.54 | 1.46 | 14.41 |
| 2,000 | 0.21 | 1.58 | 6.93 | 100.43 |
| 5,000 | 0.78 | 9.05 | 88.59 | 1826.34 |
| 10,000 | 4.09 | 47.24 | 655.51 | 13,743.22 |
| 15,000 | 7.37 | 98.29 | 2440.20 | n/a |
| 20,000 | 11.27 | n/a | n/a | n/a |
| 50,000 | 107.61 | n/a | n/a | n/a |
| 100,000 | 511.86 | n/a | n/a | n/a |
| 200,000 | 2444.32 | n/a | n/a | n/a |

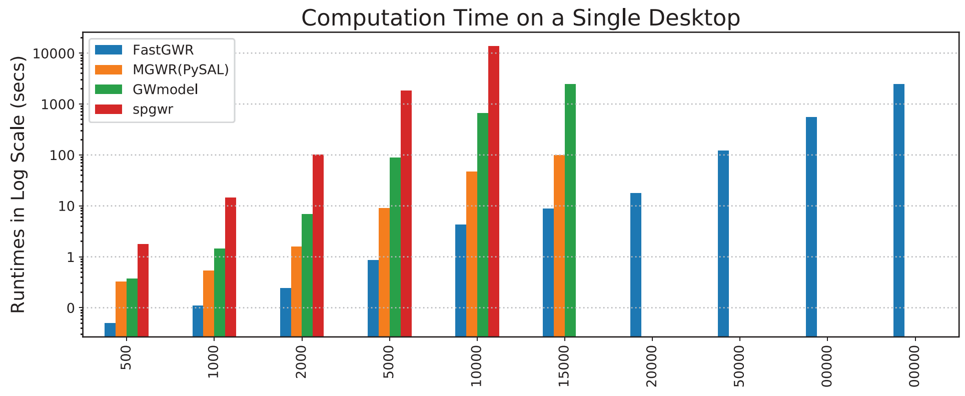

**Figure 4.** Runtime comparison for four open-source packages using different numbers of data points on a single desktop.

memory usage. By avoiding storing the *n* by *n* hat matrix, the memory complexity is reduced from O$(n^2)$ down to O$(nk)$, where O$(nk)$ is the dimension of the predictor matrix $\boldsymbol{X}$ ($k$ = 5 for the house price model used above). Though it is hard to report the exact amount of memory used for each GWR software, we compare the largest memory allocation in *FastGWR* to that of the other three implementations in Table 4. Note that 32-bit floats are used in all matrices. For 1,000 and 10,000 observations, *FastGWR* with O $(nk)$ memory complexity only needs approximately 19KB and 0.19MB of memory, respectively, while other implementations with O$(n^2)$ memory complexity need approximately 3.8MB and 380MB of memory, respectively. For this magnitude of data, an algorithm with O$(n^2)$ memory complexity is not problematic. However, for 100,000 observations, an O$(n^2)$ algorithm needs approximately 38GB, and for 1,000,000 observations, it needs approximately 3.8 TB of memory, which in both cases is prohibitive for many modern desktops. In comparison, *FastGWR* with O$(nk)$ memory complexity only requires approximately 1.9MB and 19MB of memory for 100,000 and 1,000,000 observations, respectively.

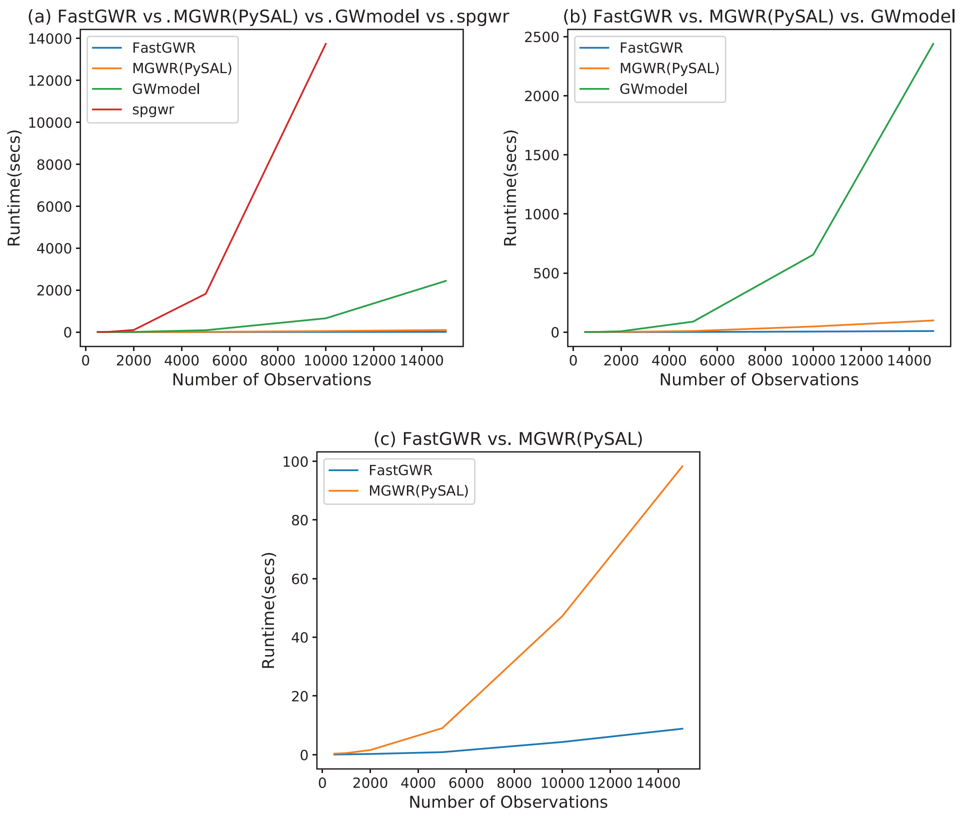

**Figure 5.** Comparison of scalability as data size grows.

## 8 Conclusions

Geographically Weighted Regression is a local modeling technique that is widely applied across a variety of disciplines. However, the location-specific calibration process makes GWR computationally intensive, and this has restricted its application to only small and medium-sized datasets. For example, current open-source GWR software can only handle datasets up to around 15,000 geo-referenced observations because the memory demands at that point become too large. To allow GWR to be applied to larger datasets, we proposed a *FastGWR* implementation, which has several advantages over current software. First, *FastGWR* optimizes the linear algebra within the GWR calibration routine to overcome memory restrictions. As a result, memory requirements are reduced from O($n^2$) to O($nk$), where $n$ is the number of observations, and $k$ is the number of covariates. Given that $k$ is far smaller than $n$ in most GWR applications, this approach saves a significant amount of memory. For example, if $n$ = 100,000 and $k$ = 10, an $n$ by $k$ matrix only needs 3.8MB of memory compared to 38GB for an $n$ by $n$ matrix. Second, *FastGWR* introduces parallel model diagnostic calculation procedures that significantly reduce the computation time required for GWR calibration by factors up to 1000x faster than existing implementations. In addition, since both of these advantages can naturally be leveraged in a

**Table 3.** Means and standard deviations (in parenthesis) of GWR local parameter estimates for four open-source GWR packages.

| Independent Variables | FastGWR | MGWR (PySAL) | GWmodel | spgwr |
|---|---|---|---|---|
| Intercept | 23.075 | 23.075 | 23.075 | 23.075 |
| | (4.105) | (4.105) | (4.105) | (4.105) |
| PctPov | −0.263 | −0.263 | −0.263 | −0.263 |
| | (0.088) | (0.088) | (0.088) | (0.088) |
| PctRural | −0.118 | −0.118 | −0.118 | −0.118 |
| | (0.037) | (0.037) | (0.037) | (0.037) |
| PctBlack | 0.045 | 0.045 | 0.045 | 0.045 |
| | (0.058) | (0.058) | (0.058) | (0.058) |

**Table 4.** Comparison of memory usage for *FastGWR* with memory complexity of O(*nk*) against other packages with memory complexity of O($n^2$).

| Number of Data Points | FastGWR O(nk) | Others O($n^2$) |
|---|---|---|
| 1000 | 19KB | 3.8MB |
| 10,000 | 0.19MB | 380MB |
| 100,000 | 1.9MB | 38GB |
| 1,000,000 | 19MB | 3.8TB |

parallel framework, the *FastGWR* algorithm scales linearly with the number of available cores.

To demonstrate the utility of *FastGWR*, we conducted an empirical study using a hedonic house price model based on a Zillow dataset containing 1.28 million single-family residential properties in the city of Los Angeles. To our knowledge, this is the first time a GWR model has been calibrated for over a million locations (and possibly the first time such a model has been calibrated for more than 100,000 locations). The bandwidth selection procedure for 1.28 million locations took only 4–5 h using a 512-core computing cluster, while the calibration of parameter estimates and computation of standard errors using a known bandwidth took less than 10 min. Furthermore, based on a sample of 10,000 data points, *FastGWR* is approximately 12 times faster than *MGWR(PySAL)*, approximately 160 times faster than *GWmodel*, and approximately 3400 times faster than *spgwr*, and these speed-ups are even greater as the number of observations are increased.

Consequently, *FastGWR* potentially enables GWR to be unrestricted in scale and across large study areas, avoiding the need to sacrifice data quality and resolution by either data aggregation or sampling. One implication of this is that GWR can now be more easily applied to remote sensing data or social media data where a large number of geo-referenced observations are increasingly available. The challenges in the application of GWR to very large datasets are no longer computational but are in the realms of visualization and interpretation – the ability to estimate large numbers of local parameters, associated standard errors, t statistics, and so on. raise important new challenges for how to efficiently interpret model results. Future work will also focus on extending the principles in *FastGWR* to the recently developed and more computationally intensive Multi-scale GWR (Fotheringham *et al.* 2017).

In summary, the advantages of *FastGWR* are:

1) It speeds up the GWR calibration routine by several orders of magnitude, especially when data size is large.

2) It lowers memory complexity from O($n^2$) down to O($nk$) to allow a single desktop to run GWR on tens of thousands of observations within a reasonable time.
3) It is an open-sourced, parallel, and portable implementation based on Python and MPI and can be used on either a desktop or within an HPC environment.
4) It opens up opportunities for applying GWR to very large datasets, previously thought impossible.

## Notes

1. https://www.open-mpi.org/.
2. http://mpi4py.scipy.org.
3. https://github.com/Ziqi-Li/FastGWR.
4. https://www.kaggle.com/c/zillow-prize-1.
5. *gwrr* is also an open-source GWR package. It is only optimizing cross-validation instead of AICc in bandwidth searching, which cannot produce comparable results.
6. https://github.com/pysal/mgwr.
7. https://cran.r-project.org/web/packages/GWmodel/index.html.
8. https://cran.r-project.org/web/packages/spgwr/index.html.
9. Computation times are based on an average of five iterations to account for any small variations that can occur due to passive consumption of computing resources by the operating system.

## Disclosure statement

No potential conflict of interest was reported by the authors.

## Funding

This work was supported by the National Science Foundation [1455349,1758786].

## Notes on contributors

***Ziqi Li*** is a PhD Student in the School of Geographical Sciences and Urban Planning, Arizona State University, Tempe, USA. His current research interest is developing geographically weighted statistical methods and tools in better understanding spatial non-stationary processes. He is also one of the developers of mgwr open-source python package for calibrating multi-scale and traditional GWR models.

***A. Stewart Fotheringham*** is Professor of computational spatial science in the School of Geographical Sciences and Urban Planning, Arizona State University, Tempe, USA. He is also a Distinguished Scientist in the Julie Ann Wrigley Global Institute of Sustainability. His research interests are in the analysis of spatial data sets using statistical, mathematical, and computational methods. He is well known in the fields of spatial interaction modeling and local statistical analysis, the latter as one of the developers of geographically weighted regression (GWR). He has substantive interests in health data, crime patterns, retailing, and migration.

***Wenwen Li*** is Associate Professor and Director of the Cyberinfrastructure and Computational Intelligence (CICI) Lab in the School of Geographical Sciences and Urban Planning at Arizona State University. Her research interest is cyberinfrastructure, space-time big data analytics and machine learning. She led the team who developed PolarHub - a large-scale web crawling engine for distributed geospatial data and PolarGlobe - a web-based scientific visualization tool for Earth science data.

***Taylor Oshan*** is an assistant professor in the Center for Geospatial Information Science within the Department of Geographical Science at the University of Maryland. His current research is focused on adapting spatial analysis methods for large heterogenous datasets and applying them to reveal how complex relationships change over space and time, especially within the context of cities. He is also broadly interested in spatial statistics, spatial data science, geocomputation, and the development of open source tools.

## ORCID

Ziqi Li http://orcid.org/0000-0003-2237-9499
Wenwen Li http://orcid.org/0000-0003-2237-9499

## References

Atkinson, P.M., *et al*., 2003. Exploring the relations between riverbank erosion and geomorphological controls using geographically weighted logistic regression. *Geographical Analysis*, 35 (1), 58–82. doi:10.1111/gean.2003.35.issue-1

Bivand, R., *et al*., 2017. Package 'spgwr'. *R Software Package*.

Brown, S., *et al*., 2012. Assessment of spatiotemporal varying relationships between rainfall, land cover and surface water area using geographically weighted regression. *Environmental Modeling & Assessment*, 17 (3), 241–254. doi:10.1007/s10666-011-9289-8

Brunsdon, C., Fotheringham, A.S., and Charlton, M.E., 1996. Geographically weighted regression: a method for exploring spatial nonstationarity. *Geographical Analysis*, 28 (4), 281–298. doi:10.1111/j.1538-4632.1996.tb00936.x

Cahill, M. and Mulligan, G., 2007. Using geographically weighted regression to explore local crime patterns. *Social Science Computer Review*, 25 (2), 174–193. doi:10.1177/0894439307298925

Cardozo, O.D., García-Palomares, J.C., and Gutiérrez, J., 2012. Application of geographically weighted regression to the direct forecasting of transit ridership at station-level. *Applied Geography*, 34, 548–558. doi:10.1016/j.apgeog.2012.01.005

da Silva, A.R. and Fotheringham, A.S., 2016. The multiple testing issue in geographically weighted regression. *Geographical Analysis*, 48 (3), 233–247. doi:10.1111/gean.2016.48.issue-3

Dalcín, L., *et al*., 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

Feuillet, T., *et al*., 2018. A massive geographically weighted regression model of walking-environment relationships. *Journal of Transport Geography*, 68, 118–129. doi:10.1016/j.jtrangeo.2018.03.002

Fotheringham, A.S., Brunsdon, C., and Charlton, M., 2002. *Geographically weighted regression: the analysis of spatially varying relationships*. New York: John Wiley & Sons.

Fotheringham, A.S., Charlton, M.E., and Brunsdon, C., 1996. The geography of parameter space: an investigation into spatial non-stationarity. *International Journal of Geographic Information Systems*, 10, 605–627. doi:10.1080/026937996137909

Fotheringham, A.S., Crespo, R., and Yao, J., 2015. Geographical and temporal weighted regression (GTWR). *Geographical Analysis*, 47 (4), 431–452. doi:10.1111/gean.2015.47.issue-4

Fotheringham, A.S., Yang, W., and Kang, W., 2017. Multiscale geographically weighted regression (MGWR). *Annals of the American Association of Geographers*, 107 (6), 1247–1265. doi:10.1080/24694452.2017.1352480

Gabriel, E., *et al*. (2004). Open MPI: goals, concept, and design of a next generation MPI implementation. *In*: D. Kranzlmüller, P. Kacsuk, J. Dongarra, eds. *Recent Advances in Parallel Virtual Machine and Message Passing Interface,EuroPVM/MPI 2004*. Lecture Notes in Computer Science, 3241. Berlin: Springer.

Gollini, I., *et al*. (2013). *GWmodel: an R package for exploring spatial heterogeneity using geographically weighted models*. arXiv preprint arXiv:1306.0413.

Griffith, D.A., 2008. Spatial-filtering-based contributions to a critique of geographically weighted regression (GWR). *Environment and Planning A*, 40 (11), 2751–2769. doi:10.1068/a38218

Gropp, W., *et al*., 1996. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22 (6), 789–828. doi:10.1016/0167-8191(96)00024-5

Harris, R., *et al*., 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/tgis.2010.14.issue-1

McMillen D. P., McDonald, J. F. (2004). Locally Weighted Maximum Likelihood Estimation: Monte Carlo Evidence and an Application. In: L. Anselin, R. J. G. M. Florax, R. S.Jey, eds. *Advances in Spatial Econometrics. Advances in Spatial Science*. Berlin: Springer.

Neese, F., 2012. The ORCA program system. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 2 (1), 73–78.

Osborne, P.E. and Suárez-Seoane, S., 2002. Should data be partitioned spatially before building large-scale distribution models? *Ecological Modelling*, 157 (2–3), 249–259. doi:10.1016/S0304-3800(02)00198-9

Pozdnoukhov, A. and Kaiser, C., 2011. Scalable local regression for spatial analytics. *In*: Divyakant Agrawal, Isabel Cruz, Christian S. Jensen, Eyal Ofek, and Egemen Tanin, eds. *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '11)*, ACM, New York, NY, USA, 361–364. doi:10.1145/2093973.2094023

Schmidt, H.A., *et al*., 2002. TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, 18 (3), 502–504.

Tran, H.T., Nguyen, H.T., and Tran, V.T., 2016. Large-scale geographically weighted regression on Spark. *In*: *Knowledge and Systems Engineering (KSE), 2016 Eighth International Conference on*, October. IEEE.,127–132. doi: 10.1177/1753193416669263

Wheeler, D.C., 2007. Diagnostic tools and a remedial method for collinearity in geographically weighted regression. *Environment and Planning A*, 39 (10), 2464–2481. doi:10.1068/a38325

Wheeler, D.C., 2009. Simultaneous coefficient penalization and model selection in geographically weighted regression: the geographically weighted lasso. *Environment and Planning A*, 41 (3), 722–742. doi:10.1068/a40256

Wolf, L.J., Oshan, T.M., and Fotheringham, A.S., 2018. Single and multiscale models of process spatial heterogeneity. *Geographical Analysis*, 50 (3), 223–246. doi:10.1111/gean.v50.3

Wu, Y., *et al*., 2013. Parallelization of a hydrological model using the message passing interface. *Environmental Modelling & Software*, 43, 124–132. doi:10.1016/j.envsoft.2013.02.002

Yu, D., 2007. Modeling owner-occupied single-family house values in the city of Milwaukee: a geographically weighted regression approach. *GIScience & Remote Sensing*, 44 (3), 267–282. doi:10.2747/1548-1603.44.3.267

Yu, H., *et al*., 2018. *Inference in multi-scale geographically weighted regression*. Open Science Framework. doi: osf.io/g4pbz

Zhang, J., 2010. Towards personal high-performance geospatial computing (HPC-G): perspectives and a case study. *In*: *Proceedings of the ACM SIGSPATIAL international workshop on high performance and distributed geographic information systems*. ACM, 3–10. doi: 10.1002/bit.22603