The Korean Journal of Applied Statistics (2018) DOI: http://dx.doi.org/10.5351/KJAS.2018.31.3.383
31(3), 383–396

Parallelism point selection in nested parallelism situations with focus on the bandwidth selection problem

Gayoung Cho[a] · Hohsuk Noh[b,1]

[a]Department of Statistics, Sookmyung Women's University;
[b]Department of Statistics, The Research Institute of Natural Sciences,
Sookmyung Women's University

(Received March 29, 2018; Revised May 8, 2018; Accepted May 14, 2018)

Abstract

Various parallel processing R packages are used for fast processing and the analysis of big data. Parallel processing is used when the work can be decomposed into tasks that are non-interdependent. In some cases, each task decomposed for parallel processing can also be decomposed into non-interdependent subtasks. We have to choose whether to parallelize the decomposed tasks in the first step or to parallelize the subtasks in the second step when facing nested parallelism situations. This choice has a significant impact on the speed of computation; consequently, it is important to understand the nature of the work and decide where to do the parallel processing. In this paper, we provide an idea of how to apply parallel computing effectively to problems by illustrating how to select a parallelism point for the bandwidth selection of nonparametric regression.

Keywords: bandwidth selection, big data, parallel computing, parallelism point selection

1. 서론

기술 발전에 따른 디지털화의 급진전으로 인해 많은 정보들이 지속적으로 축적되면서 빅데이터의 시대가 열렸다. 빅데이터는 기존의 데이터베이스 관리도구의 한계를 넘어서는 대량의 정형데이터 또는 데이터베이스 형태가 아닌 비정형 데이터의 집합을 의미한다. 빅데이터 시대가 열림에 따라 데이터의 빠른 처리와 분석을 위한 다양한 연구가 매우 활발히 진행되고 있다. 이와 연관된 대표적인 연구로, 하둡(Hadoop)과 같이 대용량의 자료를 여러 곳에 나누어 보관하면서 필요한 연산을 효율적으로 수행하는 것에 대한 연구와 처리 시간이 오래 걸리는 작업을 여러 개의 컴퓨터 또는 CPU 코어(core)로 분산하여 처리하여 연산속도를 향상시키는 기술인 병렬컴퓨팅에 대한 연구를 들 수 있다.

이러한 경향과 연관하여 통계학에서 데이터 분석을 위해 흔히 사용하는 R 프로그램의 커뮤니티에서도 대용량 자료의 처리와 통계분석을 위한 다양한 연구와 관련된 프로그램 패키지 계발이 활발하게 이루어

This research was supported by the 'Basic science research program' through the National Research Foundation of Korea funded by the Ministry of Education (NRF-2017R1D1A1A09000804).
[1]Corresponding author: Department of Statistics, The Research Institute of Natural Sciences, Sookmyung Women's University, Cheongpa-ro 47-gil 100, Yongsan-gu, Seoul 04310, Korea.
E-mail: word5810@gmail.com

지고 있다. Rossini 등 (2003)에 의해 R에서 병렬컴퓨팅을 가능하게 하는 패키지인 snow가 소개되었으며, Schmidberger 등 (2009)은 다양한 R에서의 병렬처리를 위한 패키지를 각각의 특성과 함께 소개하고 그 성능을 비교하는 모의실험을 진행하였다. 국내에서 Park 등 (2012)는 병렬 컴퓨팅을 위한 R 패키지를 소개하고 모의실험을 통한 성능평가를 통해 병렬처리를 위한 클러스터에 연결된 컴퓨터의 수가 늘어날수록, 각 컴퓨터에서 사용되는 CPU 코어수가 증가할수록 연산속도가 빨라짐을 확인하였다.

병렬처리는 수행하려는 작업이 상호의존적이지 않은 작업들로 분해될 수 있을 때 사용하게 되는데, 경우에 따라서는 병렬처리를 위해 분해된 각각의 작업들이 또 다시 상호의존적이지 않은 작업으로 분해되기도 한다. 이와 같은 중첩병렬화(nested parallelism) 상황에서는 처음 단계에서 분해된 작업들에 대해 병렬처리를 할 것을 선택하든지, 두 번째 단계에서 세분화되는 작업들에 대해 병렬처리를 할 것인지를 선택하게 된다. 일반적으로 처음 단계에서 분해된 작업들이 나중에 분해된 작업들에 비교하여 작업의 개수도 상대적으로 적고, 소요시간도 어느 정도 걸리는 경우가 많아 통신로드(communication load)를 최적화한다는 측면에서 처음에 분해된 작업에 병렬처리를 적용하는 경우가 많다. 여기서 통신로드란 마스터에서 각각의 워커에게 작업을 배분하고 처리된 결과를 모으는 일련의 과정을 의미한다. 하지만 소요시간이 큰 작업들에 대해 병렬처리를 진행하는 경우 작업 간 소요시간의 차이 역시 크다는 특성 때문에 워커들에 비슷한 양의 작업을 할당하기 어려운 문제가 발생한다. 그로 인해 적은 양의 작업을 할당받은 워커가 많은 양의 작업을 할당받은 워커의 작업이 끝날 때까지 기다리는 비효율성이 증가할 위험이 있다. 따라서 수행하고자 하는 작업의 상황에 맞게 병렬처리를 실시할 곳을 잘 결정하는 것이 계산속도 향상에 중요하다. 본 논문에서는 이러한 병렬화 포인트 선택이라는 문제에 대한 이해를 돕고 자신의 문제에 효과적으로 병렬컴퓨팅을 적용하려는 사람들에게 필요한 아이디어를 제공하려는 시도의 하나로 비모수적 함수 추정의 평활량(bandwidth) 선택이라는 구체적인 통계 문제에 대해 효율적 병렬처리를 위한 병렬화 포인트 선택과정을 제시하였다.

본 논문의 구성은 다음과 같다. 2장에서는 통계분석을 위한 흔한 R 프로그램 환경인 멀티코어 프로세서(multi-core processor) 환경에서 사용할 수 있는 병렬컴퓨팅을 위한 R 패키지를 소개하고자 한다. 3장에서는 본 논문에서 병렬화 포인트 선택이라는 문제에 대한 예시로 사용할 비모수 함수추정에서 평활량 선택문제를 소개한다. 4장에서는 비모수함수추정의 평활량 선택문제에서 자료의 수를 변화시켜 가며 병렬화 포인트를 다르게 선택하는 두 가지 계산방법의 성능을 비교하는 모의 실험결과를 제시한다. 5장에서는 논문에 사용된 병렬처리 프로그램을 소개하고 6장에서는 논문의 결론을 제시하였다.

2. 병렬처리

이 절에서는 병렬처리의 개념을 설명하고 멀티코어 프로세서 환경에서 사용할 수 있는 병렬컴퓨팅을 위한 R 패키지를 소개하고자 한다.

2.1. R의 병렬처리 패키지

통계분석을 위한 병렬컴퓨팅은 크게는 한 대의 컴퓨터 안에 있는 멀티코어 프로세서를 이용한 병렬 컴퓨팅 환경과 여러 대의 컴퓨터를 하나의 클러스터(cluster)로 묶음으로서 실시되는 병렬컴퓨팅 환경에서 주로 실시된다고 할 수 있다. 본 논문에서는 한 대의 컴퓨터 안에 있는 멀티코어 프로세서를 이용한 병렬 컴퓨팅 환경이 보다 흔히 접하는 통계분석을 위한 컴퓨팅 환경임을 고려하여 그러한 환경에서의 병렬컴퓨팅을 주 논의의 대상으로 하고자 한다. 병렬 처리는 마스터(master), 워커(worker) 구조를 통해 이해될 수 있다. 여기서 마스터는 워커에 작업을 분배하고 작업에 대한 명령을 내리는 역할을 하고, 워커는 분배된 작업을 수행한 후 수행결과를 마스터에 보내는 역할을 한다. Figure 2.1을 보면 순차 처리 방

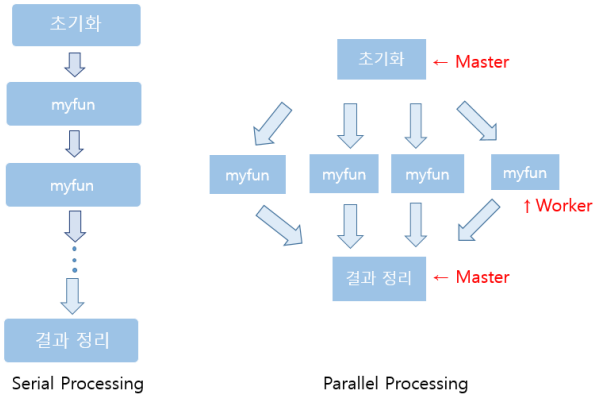

Figure 2.1. Serial processing and parallel processing.

법과 병렬 처리 방법의 차이점을 알 수 있다.

순차 처리는 Figure 2.1의 왼쪽과 같이 초기화 단계에서 연산을 순차적으로 처리한다. 반면, 병렬 처리는 Figure 2.1의 오른쪽과 같이 초기화 단계에서 여러 연산 작업을 여러 개의 워커로 나누어 동시에 처리한 뒤 마지막에 마스터가 그 결과를 정리한다.

병렬 처리를 시작하기 위해서는 마스터와 워커 사이의 통신방법을 선택해야한다. 통신방법의 종류는 SOCKET, MPI, PVM, NetWorkSpace 방식 등이 있다. 통계분석을 위한 흔한 컴퓨터 환경인 Windows를 기반으로 하는 멀티코어 프로세서 환경에서는 SOCKET 방식을 선택하면 병렬컴퓨팅을 손쉽게 할 수 있다.

최근 R에서는 많은 병렬 패키지들이 개발되었으며, 멀티코어 프로세서 환경에서 사용할 수 있는 R 패키지와 그 특징을 요약하면 아래와 같다.

(1) snow: Anthony Rossini, Luke Tierney, Na Li에 의해 처음 개발하였고, 현재까지 병렬 처리를 위해 개발된 R 패키지 중 가장 잘 알려져 있는 패키지이다. 이 패키지는 2003년에 CRAN에 등록되어 현재까지 꾸준히 발전해왔으며 모든 OS 환경에서 사용할 수 있는 장점이 있다.

(2) multicore: Simon Urbanek이 개발하였고 2009년에 CRAN에 등록되었다. Linux나 Mac OS X 등의 Posix system에서는 사용할 수 있으나 윈도우 OS를 지원하지 않는다는 단점이 있다.

(3) parallel: snow와 multicore를 합친 패키지로, parallel 패키지를 설치하면 두 패키지에 대한 별도의 설치작업을 하지 않아도 된다. 윈도우와 리눅스 모두에서 사용할 수 있지만 윈도우에서는 multicore 패키지의 함수를 사용할 수 없다.

(4) snowFT: Hana Ševčíková와 Anthony Rossini에 의해 개발한 snowFT는 snow의 기능을 확장한 R 패키지로 병렬 컴퓨팅에 있어서 난수의 재현성을 보장한 특징이 있다. 아울러 병렬컴퓨팅의 인터페이스를 snow보다 단순화하였다.

(5) foreach: 기본적으로 반복수행을 지원하며 수행결과를 벡터나 행렬로 정리하여 출력해줄 수 있게 해 준다. parallel backend를 설정하여 설정된 backend에 따라 병렬컴퓨팅을 수행시켜 주는 특징이 있다.

본 논문에서는 위의 여러 패키지 중 가장 잘 알려져 있고 모든 OS 환경에서 사용할 수 있는 snow 패키지를 중심으로 설명하고자 한다.

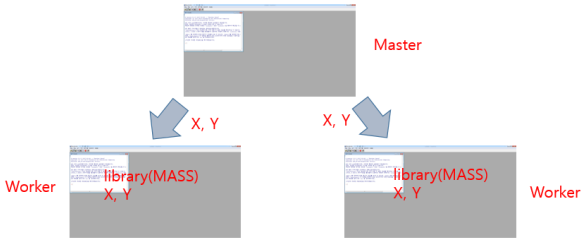

Figure 2.2. Conceptual illustration of worker setting for parallel processing.

2.2. snow 패키지

snow 패키지에서 병렬 프로그래밍을 사용하는 방법은 클러스터 객체 설정 단계, 데이터 및 작업을 전송하는 단계, 연산을 실행하는 단계, 클러스터 작업을 종료하는 총 4단계로 구성된다. 각 단계에 대한 설명과 예제 프로그램은 다음과 같다.

(1) 클러스터 객체: snow를 사용하여 병렬 처리할 때, 가장 먼저 해야 할 일은 병렬처리에 사용되는 워커들로 구성된 클러스터 객체를 만드는 것이다. 클러스터 객체는 클러스터 워커(cluster worker)와 상호작용하는데 필요하다.

Program 1 클러스터 객체 설정

```
library(snow)
cl<-makecluster(n,type="sock")
```

[프로그램 1]을 살펴보면, snow 패키지를 불러들인 뒤, 클러스터 생성함수인 makecluster를 사용하여 클러스터 객체를 설정한다. 첫 번째 인수인 n은 사용할 워커의 수이고, type은 통신방법을 의미한다.

(2) 데이터 및 작업 전송: 기본적으로 병렬처리를 위한 각각의 worker는 어떤 library나 데이터 object가 loading되어 있지 않는 fresh한 R 세션에 해당한다. 따라서 병렬처리를 실시하기 위해서는 워커에 필요한 R library를 loading시켜 놓거나 데이터 개체, 함수 등을 마스터에서부터 각각의 워커로 전달하는 워커 설정 작업이 필요하다. 워커에서 필요한 R 패키지를 loading하는데는 clusterEvalQ 함수를 사용하며, 각 워커에서 필요한 연산에 필요한 데이터 개체를 마스터로부터 워커에 전송하는 것은 clusterExport 함수를 사용한다.

Program 2 데이터 및 작업 전송 함수

```
clusterEvalQ(cl, {library(MASS); NULL})
clusterExport(cl,c("X","Y"))
```

예를 들어 [프로그램 2]를 수행하면 Figure 2.2에서와 같이 워커의 수만큼 열린 R 세션에 MASS 패키지를 loading하고, 데이터 X, Y를 전송하게 된다.

(3) 연산 실행 및 결과 전송: 클러스터의 각 워커가 마스터에서 전송받은 데이터와 정보를 이용해 작업을 수행한 후 그 결과를 마스터에 전송하는 과정이다. 이 때, 마스터에서 클러스터내의 워커에 작업을 배분하고 결과를 받아오는 방식은 [프로그램 3]의 연산실행 및 결과전송 함수에 따라 결정된다.

Program 3 연산실행 및 결과전송 함수의 다양한 예

```
clusterApply(cl,X,myfun)
clusterApplyLB(cl,X,myfun)
parLapply(cl, x, myfun)
```

clusterApply와 clusterApplyLB는 snow 패키지에서 여러 개의 워커를 사용하여 lapply 함수를 실행하는 것이라고 이해할 수 있다. clusterApply와 달리 clusterApplyLB는 워커에 작업을 배분할 때 모든 워커가 부여된 작업을 완료할 때까지 기다리는 것이 아니라 각각의 워커가 이전의 작업을 완료하자마자 새로운 작업을 부여하는 라운드 로빈(round-robin)방식을 사용한다는 점에서 차이가 있다. parLapply는 clusterApply와 clusterApplyLB에서와 같이 x의 원소 하나하나를 task로 인식하는 것과 달리, x를 워커의 수만큼의 task vector로 분할하고 각각의 task vector를 각각의 워커에서 처리하게 함으로써 마스터와 워커 사이의 통신횟수를 줄이는 병렬처리 함수라 이해할 수 있다.

(4) 클러스터 종료: 워커로부터 마스터에 결과가 전송된 이후 클러스터 종료 함수를 통해 컴퓨터 클러스터 내에 연결된 작업을 종료하는 것이 필요하다. cl이 워커 클러스터 객체라고 했을 때 stopcluster(cl)이라는 명령을 실행하면 병렬처리를 위해 구성된 클러스터 내의 워커에 해당하는 R세션이 모두 사라지게 된다.

3. 비모수 회귀함수 추정의 평활량 선택

본 절에서는 본 논문에서 병렬처리의 활용 예제로 고려하는 비모수 회귀함수의 평활량 선택문제에 대해 설명하고자 한다. 두 변수 X, Y의 관계를 파악하기 위해 조건부 평균함수 $m(x) = E(Y|X = x)$를 자료로부터 추정하는 것을 일반적으로 회귀함수 추정이라고 한다. 회귀함수를 추정할 때 특정한 회귀함수의 형태를 가정하지 않고 데이터로부터 직접 회귀함수를 추정하는 방법을 비모수적 회귀함수 추정법이라고 한다. 비모수적 회귀함수 추정법에는 스플라인(spline) 방법, 국소다항식(local polynomial) 방법, 웨이블렛(wavelet) 방법 등이 있는데 본 논문에서는 국소다항회귀를 이용한 회귀함수 추정에 병렬 처리를 활용하고자 한다. 국소다항회귀란 회귀함수를 추정하고자 하는 목표점에서 가까운 자료에 가중치를 더 많이 주어 국소적으로 다항식을 적합함으로써 목표점에서 회귀함수 값을 계산하는 방법이다. 일반적으로 국소적으로 일차식을 적합시키는 국소선형회귀(local linear regression)가 분석에 많이 사용되고 있다. 가중치를 부여하는 방식은 커널함수에 의해 결정된다. 커널 함수는 Epanechnikov 커널, triangular 커널, normal 커널, uniform 커널, Laplace 커널 등이 있으며 본 논문에서는 normal 커널함수를 사용하였다.

Figure 3.1에서는 빨간색 점으로 표시된 위치에서 국소선형회귀를 통해 어떻게 회귀함수가 추정되는지 보여주고 있다. 빨간색으로 표시된 자료가 빨간색 점에서 회귀함수 추정을 위한 국소회귀에 사용되는 자료이다. 이와 같이 국소다항회귀에서 회귀함수를 추정하고자 하는 점 주변의 어느 범위까지의 자료까지 국소회귀에 활용할 것인지를 결정하는 양을 평활량이라고 한다. 국소회귀법을 이용할 때 적절한 평활량을 결정하는 것이 회귀함수 추정 성능에 매우 중요하다.

구체적으로 국소선형회귀 추정법은 목표점 근처의 회귀 추정값 $\hat{m}(x; h)$를 식 (3.1)를 최소화하는 a, b 중 a로 추정한다.

$$\sum_{i=1}^{n}(Y_i - (a + b(X_i - x)))^2 K_h(X_i - x), \tag{3.1}$$

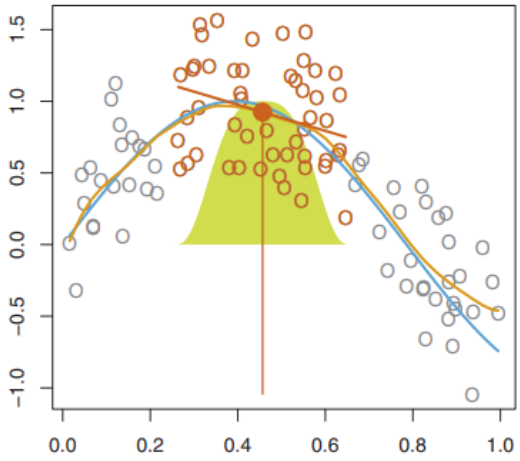

Figure 3.1. Example of local linear regression (Figure 7.9 of James *et al.* (2013)).

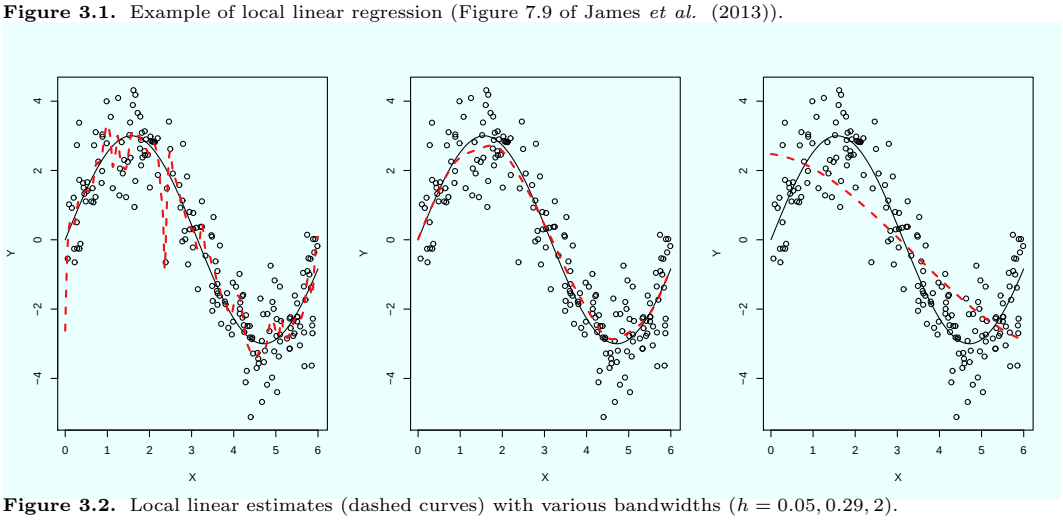

Figure 3.2. Local linear estimates (dashed curves) with various bandwidths ($h = 0.05, 0.29, 2$).

여기서 $K(\cdot)$는 Gaussian 커널함수이며, $K_h(\cdot)$는 Gaussian 커널함수를 평활량 값 h로 스케일한 함수로 $K_h(\cdot) = K(\cdot/h)/h$이다. Figure 3.2는 X_i가 균일분포 $[0, 6]$를 따르고 오차 ϵ_i가 표준정규분포를 따르고 $Y_i = 3\sin(X_i) + \epsilon_i$인 모형으로부터 얻어진 200개의 자료 $(X_1, Y_1), \ldots, (X_{200}, Y_{200})$에 대해 각각 평활량 h의 값을 0.05, 0.29, 2로 설정하여 국소선형회귀 방법으로 추정한 평균회귀 함수의 추정값이다.

Figure 3.2에서 알 수 있는 것처럼 평활량이 적절하게 선택된 경우(가운데) 참회귀함수에 가깝게 회귀함수가 추정된 반면 지나치게 작거나 크게 선택된 경우 필요 이상으로 구불구불하거나 평평하게 회귀함수가 추정되는 것을 볼 수 있다. 적절한 평활량 값 0.29는 주어진 데이터를 이용하여 고려하는 있는 평활량 후보값에 대해 식 (3.2)의 generalized cross validation (GCV)값을 계산하고 GCV값이 가장 작아지게 하는 평활량을 선택한 결과이다.

$$\mathrm{GCV}(h) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{m}(x_i; h)}{1 - L_{ii}}\right)^2, \tag{3.2}$$

여기에서 L_{ii}는 L의 i번째 대각원소이며,

$$X_x = \begin{pmatrix} 1 & x_1 - x \\ \vdots & \vdots \\ 1 & x_n - x \end{pmatrix}, \qquad W_{x,h} = \operatorname{diag}\left(K\left(\frac{x_1 - x}{h}\right), \ldots, K\left(\frac{x_n - x}{h}\right)\right), \qquad e_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

$$L = \begin{pmatrix} e_1^T (X_{x_1} W_{x_1,h} X_{x_1})^{-1} X_{x_1} W_{x_1,h} \\ \vdots \\ e_1^T (X_{x_n} W_{x_n,h} X_{x_n})^{-1} X_{x_n} W_{x_n,h} \end{pmatrix}$$

이다. 단, $\operatorname{diag}(w_1, \ldots, w_n)$는 $w_1, \ldots, w_n$를 대각원소로 가지는 대각행렬이다. GCV방법을 사용하여 주어진 평활량에 대한 GCV값을 구하려면 자료의 수가 n인 국소회귀를 적합을 n번 하는 만큼의 계산을 수행해야 한다. 따라서 자료의 수가 증가함에 따라 계산량이 자료의 수에 제곱에 비례하여 증가하게 된다. 본 논문에서는 GCV값을 이용하여 평활량을 선택하는 것을 고려하고자 한다.

4. 병렬처리 포인트 선택에 대한 모의실험

본 절에서는 3절에서 소개한 GCV값을 이용한 비모수 함수추정의 평활량 선택문제에 병렬컴퓨팅을 적용할 때 병렬처리를 적용할 포인트에 따라 계산시간이 어떻게 달라지는지를 확인하고자 한다. GCV값을 이용한 평활량 선택은 기본적으로 평활량 후보 값들의 리스트 $\{h_1, \ldots, h_N\}$에 있는 각각의 평활량에 대해 GCV(h)를 계산하는 작업이라고 할 수 있고 이 작업들은 상호의존적이지 않기에 병렬처리가 가능하다. 또한,

$$\mathrm{GCV}(h) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{m}(x_i; h)}{1 - L_{ii}} \right)^2 \equiv \frac{1}{n} \sum_{i=1}^{n} \{R_i(h)\}^2$$

에서 알 수 있듯이 주어진 평활량 h에 대해 GCV(h)를 계산하는 작업은 각각의 자료 (x_i, y_i)에 대해 $R_i(h)$를 계산하는 작업이라 볼 수 있기에 이차적인 병렬처리 작업이 가능하다. 따라서 Figure 4.1에 나와 있는 것 같이 평활량 선택문제에서는 각각의 평활량에 대해 GCV(h)를 계산하는 것(A)을 병렬처리할지 아니면 각각의 자료에 대해 $R_i(h)$를 계산하는 것(B)을 병렬처리할지 결정해야 한다.

4.1. 평활량에 대한 GCV값 계산 병렬처리 선택

평활량에 대한 GCV값 계산을 병렬처리 한다는 것은 Figure 4.1에서 B부분은 순차처리를 하고 A부분에 대해 병렬처리를 하는 것을 의미한다. Figure 4.2에서는 워커가 2개일 때, 마스터에서 연산에 필요한 데이터 X와 Y를 보내주고 다양한 평활량 값에 대한 GCV(h)값을 각각의 워커에서 병렬로 처리하여 마스터로 보내주는 과정을 보여주고 있다.

4.2. 자료에 대한 $R_i(h)$값 계산 병렬처리 선택

자료에 대한 $R_i(h)$값을 계산하는 부분을 병렬처리 한다는 것은 Figure 4.1에서 B부분은 병렬처리를 하고 A부분에 대해 순차처리를 하는 것을 의미한다. 이 경우 Figure 4.3에 나와 있는 바와 같이 각각의 평활량에 대해 자료의 수만큼의 세부작업을 여러 개의 워커로 나누어 병렬처리를 수행하게 된다. 이러한 병렬화 작업은 평활량에 대한 GCV값 계산에 대한 병렬처리보다 병렬처리하는 작업 개수가 훨씬 많아져 마스터와 워커 사이의 통신로드를 많이 증가시키며 그로 인해 계산속도가 저하될 위험성이 있다.

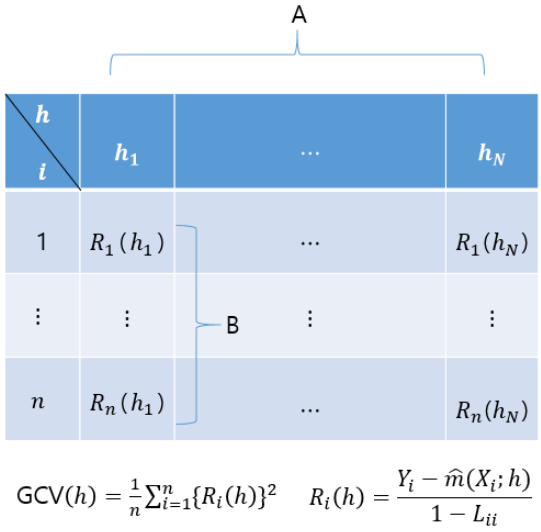

Figure 4.1. Two parallelizable points for bandwidth selection problem.

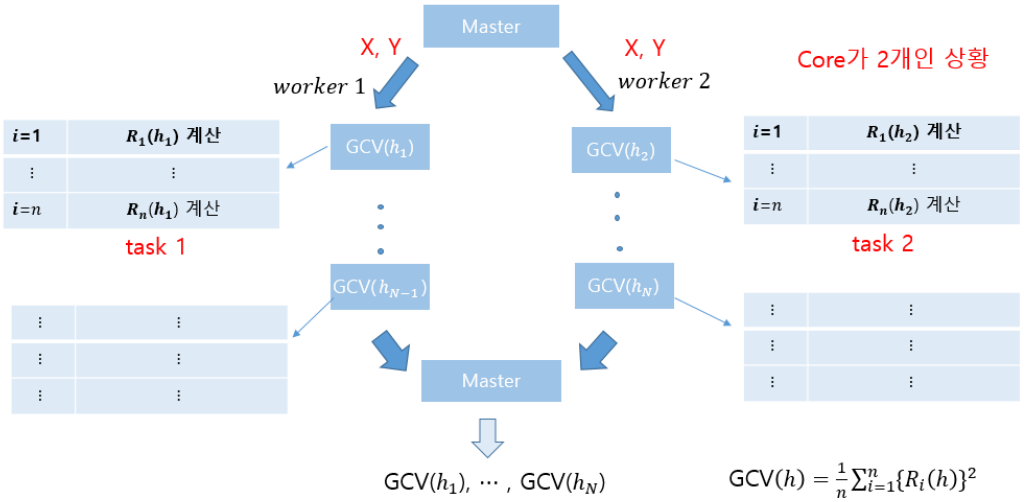

Figure 4.2. Conceptual diagram of parallel processing of the GCV(h) calculation for bandwidth selection when there are two workers.

4.3. 모의실험 결과

병렬화 포인트 선택에 따라 계산성능이 어떻게 차이가 나게 되는지를 파악하기 위해 평균회귀분석에서 X_i가 구간 $[0, 6]$에서의 균일분포, 오차 ϵ_i가 표준정규분포를 따르고 $Y_i = 3\sin(X_i) + \epsilon_i$로 주어지는 모형으로부터 얻어진 각각 자료의 수 n이 $200, 400, 600, 800, 1000$인 자료를 생성하였다. 주어진 자료로부터 회귀함수를 비모수적으로 추정하기 위한 평활량 h를 0.05부터 3까지 등간격의 16개의 그리드 포인트에서 GCV방법에 근거하여 선택하는 상황을 고려하였다.

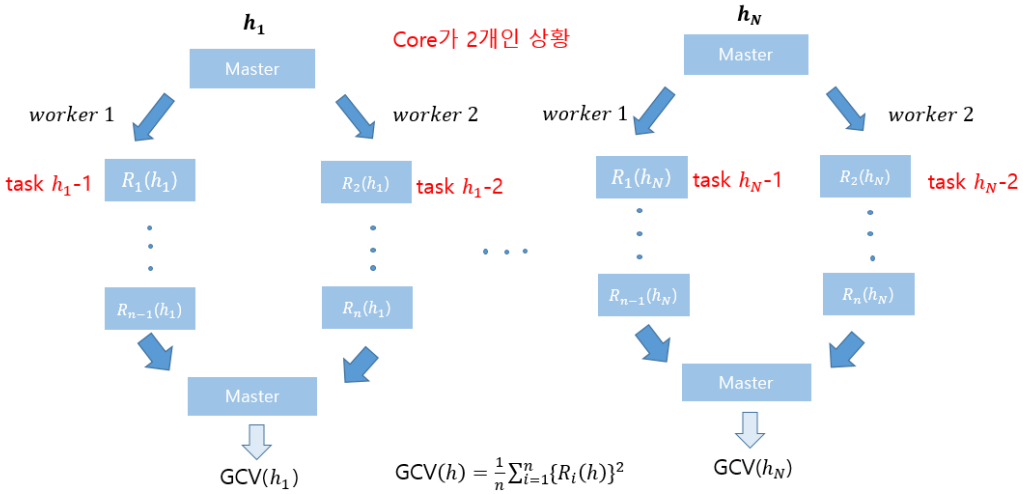

Figure 4.3. Conceptual diagram of parallel processing of the $R_i(h)$ calculation for bandwidth selection when there are two workers.

Table 4.1. Comparison of computation time depending on the parallelism points

	Sequential processing	Parallel processing			
		GCV(h) calculation		$R_i(h)$ calculation	
		2 workers	4 workers	2 workers	4 workers
$n = 200$	2.07(0.19)	1.06(0.07)	0.58(0.04)	1.97(0.17)	1.38(0.13)
$n = 400$	9.50(0.31)	4.43(0.08)	2.38(0.04)	6.18(0.06)	3.29(0.03)
$n = 600$	33.96(1.41)	14.81(0.63)	7.99(0.38)	17.13(0.44)	9.34(0.16)
$n = 800$	108.24(4.92)	48.09(0.69)	30.04(0.46)	50.41(0.67)	30.69(0.22)
$n = 1000$	188.84(7.41)	152.58(8.58)	110.56(4.23)	149.36(2.85)	107.61(1.66)

Table 4.1은 평균회귀 분석의 평활량 선택을 위해 각각의 평활량에 대해 GCV(h)값을 계산하는 부분에 병렬 처리를 했을 때와 평활량이 주어져있을 때 각각의 자료에 대한 $R_i(h)$값을 계산하는 부분에 병렬처리를 했을 때 자료의 수에 따라, 워커의 수에 따라 어떻게 계산시간의 차이가 났는지를 보여주고 있다. 참고로 평활량 선택을 위한 핵심 계산은 크게 L 행렬의 계산에 필요한 행렬 생성과 L 행렬의 계산수행 두 부분으로 나눌 수 있는데 후자의 계산이 계산시간의 대부분을 차지하는 것으로 나타났다. 각 셀에 표시된 값은 주어진 조건하에서의 계산시간을 100번 반복 측정한 것에 대한 평균값이며 괄호안의 값은 반복 측정된 계산시간의 표준편차이다. 단위는 초이다. 계산을 위해 사용된 컴퓨터의 CPU는 Intel(R) Xeon(R) CPU E5-2697 v3 2.60GHz(×2)이며 메모리는 256GB였다. 전체 56개의 core 중에 최대 4개까지의 core를 사용하여 병렬계산을 수행하였다.

Table 4.1의 결과를 보면 쉽게 예상할 수 있듯이 자료의 수가 고정되어 있을 때 병렬처리를 실시하는 포인트의 차이에 관계없이 워커(또는 코어)의 수가 늘어남에 따라 계산시간이 감소하는 경향이 관찰됨을 볼 수 있다. $R_i(h)$를 계산하는 부분에 병렬처리를 하게 되면 GCV(h)를 계산하는 부분에 병렬처리를 하는 것보다 훨씬 많은 수의 작업들에 대해 병렬처리를 실시하는 것이어서 마스터와 워커 사이의 통신로드가 상대적으로 더 많이 발생하게 된다. 그 뿐 아니라 자료의 수가 상대적으로 적을 때에는 $R_i(h)$를 계산하는데 소요되는 시간이 매우 짧기 때문에 워커에 작업을 할당하자마자 거의 동시에 작업이 완료되어

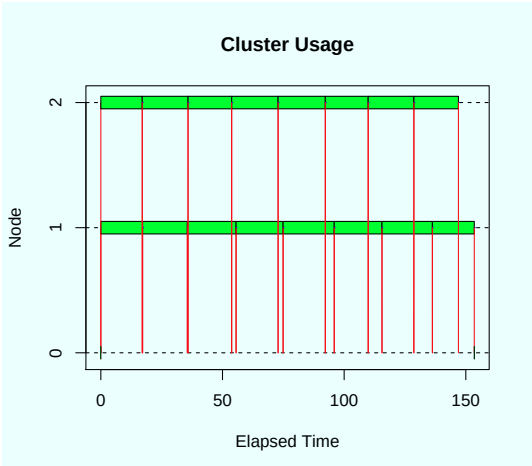

Figure 4.4. Example of the work allocation when applying parallel computing to GCV(h) calculation with two workers ($n = 1000$). A block represents one allocated workload and a red line represents communication between the master (Node 0) and the worker (Node 1, 2).

워커들이 작업을 할당받기 위해 대기하는 상황이 매우 빈번하게 발생하여 각각의 워커에 작업을 배분하고 수행결과를 받는 것이 비효율적으로 진행되게 된다. 그 결과 자료의 수가 적을 때에는 $R_i(h)$ 병렬처리 방식이 GCV(h) 병렬처리 방식보다 계산시간 면에서 상당히 비효율적인 방식으로 나타났다.

하지만 자료의 수가 커져서 고정된 평활량에 대해 $R_i(h)$를 계산하는 것이 약간의 계산시간이 걸리는 작업이 되면 자료의 수가 적었을 때 나타났던 각 워커에 작업을 부여하자 작업이 종료되어 작업을 다시 할당해야 하는 상황이 발생함으로써 생기는 마스터·워커 통신상의 비효율성이 개선됨에 따라 $R_i(h)$ 병렬처리 방식과 GCV(h) 병렬처리 방식의 계산시간 차이가 줄어드는 것으로 나타났다. 자료의 수가 800개가 되었을 때부터는 병렬처리 포인트 차이에 따라 계산시간의 유의미한 차이가 나타나지 않았다. 흥미로운 것은 자료의 수가 매우 커져서 1,000개가 되었을 때에는 근소하기 하지만 $R_i(h)$ 병렬처리 방식이 GCV(h) 병렬처리 방식보다 계산시간이 덜 걸리는 것으로 나타났다. 또한, 계산시간의 표준편차도 $R_i(h)$ 병렬처리 방식이 GCV(h) 병렬처리 방식보다 훨씬 작은 것을 볼 수 있다. 이는 병렬처리 하는 작업의 크기가 크고 작업들 간의 소요시간 차이가 어느 정도 있을 때 그것을 각 워커들에게 균등하게 배분하기 어려운 현상과 연관이 있다.

자료의 수가 커지면 하나의 평활량에 대해 GCV 값을 계산하는 것이 시간이 상당히 걸리는 (약 16초에서 20초 정도) 큰 작업이 되고 평활량에 따라 계산소요시간의 차이도 상대적으로 커지게 된다. 그러한 작업들을 워커들에 분배하다 보면 각각의 워커에 비슷한 양의 작업이 부여될 때도 있지만 때로는 부여되는 양의 차이가 생길 때도 있다. 그러한 경우 적은 양의 작업을 부여받은 워커는 자신보다 상대적으로 많은 양의 작업을 부여받은 워커가 작업을 완료할 때까지 기다리게 되는 일이 발생하게 된다. Figure 4.4은 자료의 수가 1,000개일 때 평활량 선택을 위해 2개의 워커에 16개의 평활량에 대한 GCV 값을 계산하는 작업이 어떻게 분배되었는지를 snow패키지의 snow.time이라는 함수를 이용해서 나타낸 예이다. Figure 4.4에서 볼 수 있듯이 작업의 불균등분배로 인해 워커2가 워커1이 작업을 종료할 때까지 약 6초(= 153.45초 − 146.94초)를 기다리는 상황이 발생하였다. 그에 비해 주어진 평활량에 대해 $R_i(h)$를 계산하는 과정에 두 개의 워커로 병렬처리를 하게 되면 각각의 워커에 계산시간이 상대적으로 훨씬 짧은 작업들이 계속 분배하는 것이 되어 Figure 4.5에서와 같이 워커에 따른 작업량 배분 불균등이 훨씬 덜 발생하게 되고 두 워커가 거의 동시에 작업을 완료하게 된다. 이러한 점을 살펴볼 때 자료의 수가 클 때에는 자료의 수가 작을 때와는 정반대로 GCV(h) 계산하는 부분보다는 $R_i(h)$를 계산하는 부분에 병렬

Figure 4.5. Example of the work allocation when applying parallel computing to $R_i(h)$ calculation with two workers ($n = 1000$ and $h = 0.05$). The difference between the computing time of two workers was 0.05 second. A block represents one allocated workload and a red line represents communication between the master (Node 0) and the worker (Node 1, 2).

처리를 적용하는 것이 평활량 선택을 위한 계산시간을 줄일 수 있다고 할 수 있다.

4.4. 병렬컴퓨팅을 위한 시뮬레이션 코드 설명

이 절에서는 본 논문에서 사용한 병렬처리 시뮬레이션 코드를 간단하게 제시하였다. [프로그램 6]은 자료의 수가 정해져 있을 때 순차처리, 워커가 2개일 때와 4개일 때 본 논문에서 고려하는 두 가지 병렬처리방식에 의한 평활량선택 시간을 측정하는 프로그램의 예시이다. [프로그램 4]의 gcvh, [프로그램 5]의 gcvhpara 두 함수 모두 주어진 평활량에 대해 GCV값을 구하는 함수이다. gcvh 함수는 $R_i(h)$값을 구하는 함수를 순차적으로 적용하여 GCV값을 계산하도록 되어 있는 반면 gcvhpara 함수는 $R_i(h)$값을 구하는 함수를 설정된 워커의 수만큼 병렬로 적용하여 GCV값을 계산하도록 되어 있다.

Program 4 gcvh함수

```
gcvh<-function(h)
{
  h_in<-h
  R_i_sq<-function(i) # R_i 값의 제곱 계산
  {
    x0 <- X[i]; w_x0 <- dnorm((X - x0)/h_in);
    WM_x0 <- diag(w_x0); Xnew <- X - x0; Des_x0 <- cbind(1, Xnew);
    Ltemp <- as.vector(c(1, 0) %*% solve(t(Des_x0) %*%
                      WM_x0 %*% Des_x0) %*% t(Des_x0) %*% WM_x0)
    mhat <- Ltemp %*% Y; temp <- (((Y[i] - mhat)/(1 - Ltemp[i])))^2;
    return(temp)
  }
  return(sum(sapply(n_arr,R_i_sq)))  # R_i값 구하는 함수를 순차적용하여 GCV값 산출
}
```

Program 5 gcvhpara함수

```
gcvhpara<-function(h)
{
  h_in<-h
  clusterExport(cl,"h_in",envir=environment())
  R_i_sq<-function(i) # R_i 값의 제곱 계산
  {
    x0 <- X[i]; w_x0 <- dnorm((X - x0)/h_in);
    WM_x0 <- diag(w_x0); Xnew <- X - x0; Des_x0 <- cbind(1, Xnew);
    Ltemp <- as.vector(c(1, 0) %*% solve(t(Des_x0) %*%
                        WM_x0 %*% Des_x0) %*% t(Des_x0) %*% WM_x0)
    mhat <- Ltemp %*% Y; temp  <- (((Y[i] - mhat)/(1 - Ltemp[i])))^2;
    return(temp)
  }
   return(sum(unlist(clusterApplyLB(cl,n_arr,fun=R_i_sq))))
   # R_i값 구하는 함수를 워커의 수만큼 병렬처리하여 GCV값 산출
}
```

Program 6 소요시간 측정하는 프로그램

```
library(snow)
# 데이터 생성
n<-200 # 자료의 수
h_arr<-seq(0.05,3,length=16) # 평활량 후보 벡터
n_arr<-seq(1,n,1); X<-runif(n,0,6); e<-rnorm(n,0,1); Y<-3*sin(X)+e;

# 병렬처리 없이 평활량 선택 소요시간
time_seq<-system.time(res1<-sapply(h_arr,gcvh))

# 워커를 2개로 설정
cl<-makeCluster(2,type = "SOCK"); clusterExport(cl,c("X","Y","n_arr"));
# GCV값 병렬처리(워커 2개)에 의한 평활량 선택 소요시간
time_par2_Ri<-system.time(res2<-sapply(h_arr,gcvhpara))
# R_i값 병렬처리(워커 2개)에 의한 평활량 선택 소요시간
time_par2_GCV<-system.time(res3<-clusterApplyLB(cl,h_arr,fun=gcvh))
stopCluster(cl) # 워커 개수 변경을 위해 기존 워커 클러스터 종료

# 워커를 4개로 설정
cl<-makeCluster(4,type = "SOCK"); clusterExport(cl,c("X","Y","n_arr"));
# GCV값 병렬처리(워커 4개)에 의한 평활량 선택 소요시간
time_par4_Ri<-system.time(res4<-sapply(h_arr,gcvhpara))
```

```
# R_i값 병렬처리(워커 4개)에 의한 평활량 선택 소요시간
time_par4_GCV<-system.time(res5<-clusterApplyLB(cl,h_arr,fun=gcvh))
stopCluster(cl) # 워커 클러스터 종료
```

5. 결론

본 논문에서 통계처리를 위한 흔한 환경(multi-processor, windows)에서 병렬처리를 하려고 할 때 발생할 수 있는 병렬화 포인트 선택이라는 이슈를 비모수 함수추정의 평활량 선택이라는 예제를 통해 살펴보았다. 일반적으로 병렬처리를 위해 분해된 각각의 작업들이 또 다시 상호의존적이지 않은 세부작업으로 분해되는 경우, 처음 단계에서 분해된 작업들에 대해 병렬처리를 하는 것이 두 번째 단계에서 추가적으로 분해되는 상대적으로 크기가 더 작은 작업들에 대해 병렬처리를 하는 것보다 계산효율면에서 더 좋다고 알려져 있다. 하지만, 평활량 선택 예제에서 알 수 있는 것은 그러한 선택이 항상 효율적인 선택이 아니라는 것이다. 두번째 단계에서 세분화되는 작업의 크기가 매우 작은 경우에는 마스터,워커 사이의 통신로드등을 고려할 때 그러한 선택이 적절한 선택이지만, 두 번째 단계에서 분해되는 작업의 크기가 증가함에 따라 두 병렬화 포인트 처리 방식에 따른 계산시간의 차이가 줄어드는 것을 확인할 수 있었다. 또한, 두 번째 세분화되는 작업의 크기가 어느 정도 커지게 되면 첫 번째 단계에서 병렬화하려는 작업들이 상당한 계산 소요시간을 요구하는 작업이 되게 되는데 그러한 경우는 각 워커에게 균일하게 작업을 할당하는 것이 어려워지는 면이 있어 효율적인 병렬처리가 되지 않을 수 있는 단점이 생긴다. 따라서 일반적인 가이드라인과는 다르게 두 번째 단계에서 분해되는 작업들에 대해 병렬처리를 하는 것이 더 효율적인 병렬처리가 될 수 있다.

References

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R*, Springer-Verlag, New York.

Park, Y. M., Ko, Y. J., and Kim, J. S. (2012). R Packages for parallel computing and their performance evaluation. *Journal of the Korean Data Analysis Society*, **14**, 1951–1961.

Rossini, A., Tierney, L., and Li, N. (2003). Simple parallel statistical computing in R, Technical Report.

Schmidberger, M., Morgan, M., Eddlebuettel, D., Yu, H., Tierney, L., and Mansmann, U. (2009). State of the art in parallel computing with R, *Journal of Statistical Software*, **31**, 1–27.

Ševčíková, H. and Rossini, A. J. (2004). Pragmatic parallel computing, Technical Report.

평활량 선택문제 측면에서 본 중첩병렬화 상황에서 병렬처리 포인트선택

조가영[a] · 노호석[b,1]

[a]숙명여자대학교 통계학과; [b]숙명여자대학교 통계학과, 자연과학연구소

(2018년 3월 29일 접수, 2018년 5월 8일 수정, 2018년 5월 14일 채택)

요 약

빅데이터의 시대가 열림에 따라 데이터의 빠른 처리와 분석을 위한 방법의 하나로 R 프로그램 기반의 다양한 병렬처리 패키지가 사용되고 있다. 병렬처리는 수행하려는 작업이 상호의존적이지 않은 작업들로 분해될 수 있을 때 사용하게 되는데, 경우에 따라서는 병렬처리를 위해 분해된 각각의 작업들이 또 다시 상호의존적이지 않은 세부작업으로 분해되기도 한다. 이러한 중첩병렬화 상황에서는 일반적으로 처음 단계에서 분해된 작업들에 대해 병렬처리를 할지, 두 번째 단계에서 세분화되는 작업들에 대해 병렬처리를 할지 선택하게 된다. 그러한 선택이 계산 속도에 상당한 영향을 주는 경우가 많기 때문에 수행하고자 하는 작업의 상황에 따라 병렬처리를 실시할 곳을 잘 결정하는 것이 중요하다. 본 논문에서는 이러한 병렬화 포인트 선택이라는 문제에 대한 이해를 돕고 자신의 문제에 효과적으로 병렬컴퓨팅을 적용하려는 사람들에게 필요한 아이디어를 제공하려는 시도의 하나로 비모수적 함수 추정의 평활량 선택이라는 구체적인 통계문제에 대해 효율적인 계산을 위한 병렬화 포인트 선택 과정을 제시하였다.

주요용어: 병렬처리, 병렬화 포인트 선택, 빅데이터, 평활량선택

이 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (NRF-2017R1D1A1A09000804).

[1]교신저자: (04310) 서울특별시 용산구 청파로47길 100, 숙명여자대학교 통계학과, 자연과학연구소.
E-mail: word5810@gmail.com