

# KDMLR<sup>1</sup>에 기반을 둔 Multiple Linear Regression

## 병렬 알고리즘의 설계 및 구현

이승일, 우경태, 전성익

한국정보통신대학원대학교

gaialee@icu.ac.kr, woockt@ssky.co.kr, sijun@etri.re.kr

### 요약

MRL(Multiple Linear Regression)은 데이터 사이의 관계를 파악하고, 수집된 자료를 기반으로 미래의 데이터를 예측하는 과정에서 많이 사용되고 있는 통계적 방법이다. 본 프로젝트는 KDMLR은 독립 변수의 수가 많을 때 기존의 통계적인 방법으로는 정확한 결과를 생성할 수 없는 단점을 극복하는 새로운 접근 방식이다. 이 KDMLR은 많은 독립 변수들에 대한 관련성을 분석하기 위하여 반복법을 사용하는 특징이 있다. 본 프로젝트에서는 KDMLR이 반복법으로 인해 많은 계산 시간을 필요로 하는 것을 병렬 알고리즘으로 개선하여 보다 빠른 예측이 가능하도록 하였다.

## 1. 서론

기존의 통계 패키지에 도입한 MLR 알고리즘으로는 독립 변수의 수가 주어진 일차 방정식의 수 보다 많은 경우에는 문제를 해결하지 못했다. KIST의 Doping Control Center에서 고안한 알고리즘은 이 점을 보완하여 독립 변수의 수가 주어진 일차 방정식의 수 보다 큰 경우에도 MLR의 방법을 사용할 수 있도록 한 것이다. 이 알고리즘에 대한 공식적인 명칭은 아직 주어지지 않았지만, 편의상 본 보고서에서는 이 순차 처리 알고리즘을 KDMLR이라 부르겠다.

MLR은 통계적 데이터 처리의 한 가지 방법으로 그 응용 범위가 크다. 따라서, MLR 순차 처리 알고리즘의 병렬화는 그 의미가 크다고 할 수 있다. 그러나, 순차 처리 프로그래밍과 병렬 처리 알고리즘은 패러다임이 다르기 때문에 순차 처리 알고리즘의 단순 병렬화는 어느 정도의 한계를 가질 것이다. 본 병렬 프로그래밍 프로젝트에서는 KDMLR 알고리즘의

---

<sup>1</sup> KDMLR은 Kist Doping control center가 설계한 순차 MLR 알고리즘이다.

단순 병렬화가 아닌 문제의 성격을 파악하여 하나 이상의 프로세서에서 통계적으로 의미 있는 정확한 결과를 도출해 낼 수 있도록 병렬 알고리즘을 설계하고 구현하여 보았다. 본 보고서의 2 절에서는 KDMLR 순차 처리 알고리즘에 대하여 소개하고, 3 절에서는 이를 기반으로 설계한 병렬 알고리즘에 대하여 설명한다. 4 절에서는 3 절에서 설명된 병렬 알고리즘을 구현한 프로그램의 성능을 보여주며 5 절에서는 본 프로젝트를 수행하면서 앞으로 논의 될 수 있는 주제에 대하여 간략히 논하고 6 절에서 결론의 글로 마무리 짓는다.

## 2. 순차 알고리즘 - KDMLR

MLR[1]은 목적 변수(target variable) Y 를 가장 잘 설명하는 독립 변수 X 를 찾는 방법으로 설명력은 multiple determination 계수,  $R^2$ ,의 값으로 나타내어 진다. 일반적으로  $R^2$  의 값이 클수록 설명력이 높다. 가장 이상적인 방법은 모든 독립 변수 X 의 조합을 구하여 그 중에 제일 높은  $R^2$  값을 갖는 독립 변수의 집합을 구하면 된다. 그러나 이 경우 독립 변수의 수가 증가하면 계산 시간이 기하 급수적으로 증가한다. 따라서, KIST Doping control center 에서는 계산 시간의 문제를 어느 정도 해결하고 통계적으로 의미 있는 데이터를 얻어 낼 수 있는 새로운 알고리즘, KDMLR,을 고안하였다. 다음은 KDMLR 알고리즘에 대한 설명이다.

먼저  $R^2$  값이 높은 두 개의 독립 변수로 이루어진 집합을 만들고 이 집합에 나머지 독립 변수를 하나씩 첨가하여 3 개의 독립 변수로 이루어진 집합을 만든다. 이 중에 다시  $R^2$  값이 큰 세 개의 독립 변수로 이루어진 집합을 골라낸다. 이런 방법으로 관심을 가지는 일정 수 (m)의 독립 변수로 이루어진 집합 중에  $R^2$  값이 가장 큰 것을 찾아내는 알고리즘이다.

예를 들어, N 개의 독립 변수가 있는 경우를 생각해 보자. 이 때,  $N \geq 0$  이다. 먼저 N 개의 독립 변수로 이루어진 집합에서 두 개의 원소로 이루어진 부분 집합을 구한다[그림 1]. 각 부분 집합에 대해서  $R^2$  값을 계산한 후에  $R^2$  값이 큰 순서대로 일정 수의 부분 집합을 선택한다.

$$\{X_1, X_2, \dots, X_i, \dots, X_N\}$$

$$\underbrace{\hspace{10em}}_{\{X_i, X_j\}}$$

$$N \geq 0, i \neq j, 1 \leq i, j \leq N$$

[그림 1]

선택되어진 부분 집합에  $k \neq i, j$  이면서  $1 \leq k \leq N$  인 원소를 첨가하여 세 개의 원소로 이루어진 부분 집합을 만든다[그림 2]. 그림 2 에 보여지는 각 부분 집합에 대해서 다시  $R^2$  값을 계산한다. 계산 후에 다시  $R^2$  값이 큰 순서대로 다시 일정 수의 부분 집합을 선택한다.

$$\{X_i, X_j, X_k\} \quad k \neq i, j, \quad 1 \leq k \leq N$$

[그림 2]

초기에 목적 변수(target variable) Y를 가장 잘 설명하는 독립 변수 X를 몇 개 얻어낼 것인가를 결정한다. 위와 같은 반복 과정은 이 때 정해진 독립 변수의 개수로 이루어진 부분 집합이 얻어질 때까지 반복한다. KDMLR의 PSEUDO 순차 처리 코드를 나타내면 소스 1과 같다.

```

Program SQ_MLR(X, Y)
  Caculate2ComponentList
    nC2 ← select_2_componentList(X)
    R2 ← calculateR2(nC2, Y)           # LAPACK 패키지 사용
    Sort(R2)                          # 손실 발생 가능 부분
  CalculateNComponentList
    Do I=3, Max_Component
      CI ← make_I_componentList(I)
      RI ← calculateR2(CI, Y)
      Sort(RI)
    EndDo
  EndOfProgram
  
```

[소스 1] 순차 처리 PSEUDO 코드

### 3. 병렬 알고리즘 - PKDMLR

순차 처리 알고리즘인 KDMLR을 병렬화 하기 위하여 두 가지를 고려 하였다. 첫 째는 순차처리 알고리즘이 계산 시간 단축을 위하여 취하고 있는 방법을 병렬 처리에 사용하는 방안을 고려 했고, 두 번째는 순차 처리 알고리즘이 가지고 있는 특징을 이용하는 방안을 생각했다.

순차 처리 알고리즘은 계산 시간 단축을 위하여 종속 변수 Y를 가장 잘 설명하는 독립 변수 X의 집합을 구하기 위해서 먼저  $R^2$  값이 높은 순서로 두 개의 독립 변수로 이루어진 일정 수의 집합을 계산하여 다음의 계산에 사용하고 있다. 모든 경우의 수를 고려 한다면  $O(N^4)$ 의 복잡도가 된다. 이 과정에서  $R^2$  값이 다른 집합보다 상대적으로 작은 집합들은 다음의 계산에서 제외된다. 즉,  ${}_N C_2$  개 중에서 K 개만 선택한다. 그러나 이 과정에서 제외된 두 개의 원소로 이루어진 집합 중에서 다른 독립 변수가 더해지면  $R^2$  값이 높아져 제외되지 않은 다른 집합보다 더 큰  $R^2$  값을 가질 수 있는 집합이 통계적으로 있을 수 있다[1]. 이런 관점에서 볼 때, KDMLR 알고리즘은 계산 시간 단축을 위해서 이러한 손실을 감수하고 있는 것이다.

본 프로그램의 실험을 위해 TEST 데이터로 사용하기 위해 데이터에서 독립 변수의 종류는 여러 가지인 집합을 준비하였다. 여기서 23 개의 독립 변수를 갖는 집합의 예를 들면 이 데이터를 기준으로 두 개의 원소를 가지는 부분 집합의 개수는 253 가지가 나온다. KDMLR 알고리즘은 253 가지 가능한 모든 부분집합을 가지고 계산을 했을 때와 K 개의 부분 집합만을 가지고 계산했을 때 같은 결과를 보여 주었다. 이 것이 의미하는 것은 KDMLR의 알고리즘으로 MLR의 문제를 해결할 경우에 가능한 두 개의 원소를 가지는 부분 집합에 대하여 계산할 필요가 없다는 의미를 가진다. 같은 결과를 얻기 위해서 필요한 최소 부분 집합의 개수는 다르지만 다른 데이터를 가지고 실험을 한 결과도 비슷한 결과를 얻었다.

위의 두 가지를 고려하여 KDMLR에 기반하여 고안된 병렬 알고리즘을 소스 2와 같다. KDMLR을 단순히 병렬화했을 경우에는  $R^2$  값이 높은 부분 집합을 일정 수 골라내기 위한 `sort(R2)`과정[소스 1]에서 각 프로세스간 데이터 전달이 매번 이루어져야 한다. 그러나, 위에서 언급한 두 가지 면을 고려하면 각 프로세스마다  $nC2 / Pe$  개수의 부분 집합만을 가지고 계산하고 마지막 계산 결과에 대해서만 전체 정렬을 해주면 될 것이다. 이는 모든 경우의 수를 고려하는 것으로 확장된 것이다. 고안된 병렬 알고리즘은 각 프로세스가 가지고 있는  $nC2 / Pe$  개의 부분 집합에 대한 정렬을 하여 계산을 진행하므로 `calculateR2(nC2/Pe, Y)`에 의해 계산되어 R2 안에 들어 있는 부분집합 리스트의  $R^2$  값이 다른 프로세스의 R2 안에 들지 못한 부분 집합의  $R^2$  값보다 작을 수 있다. 그러나 이런 측면은 계산 시간 단축을 위해서 순차 알고리즘이 손실을 보고 있는 영역과 같은 의미로 이해 될 수 있을 것이다.

```

Program P_MLR(X, Y)
  Do I = nStartOfPi, nEndOfPi
    Caculate2ComponentList # STEP 1
      nC2/Pe ← select_2_componentList(X)
      R2 ← calculateR2(nC2/Pe, Y) # LAPACK 패키지 사용
      Sort(R2) # K 개만 선택
    CalculateNComponentList # STEP 2
      Do I=3, Max_Component
        CI ← make_I_componentList(I)
        RI ← calculateR2(CI, Y)
        If I == Max_Component
          Begin
            mergeSort() # 모든 프로세스의 RI 를 정렬
            printResult()
            ExitOfProgram # 프로그램 종료
          EndOfBegin
        Sort(RI)
      EndDo
    EndDo
  EndOfProgram

```

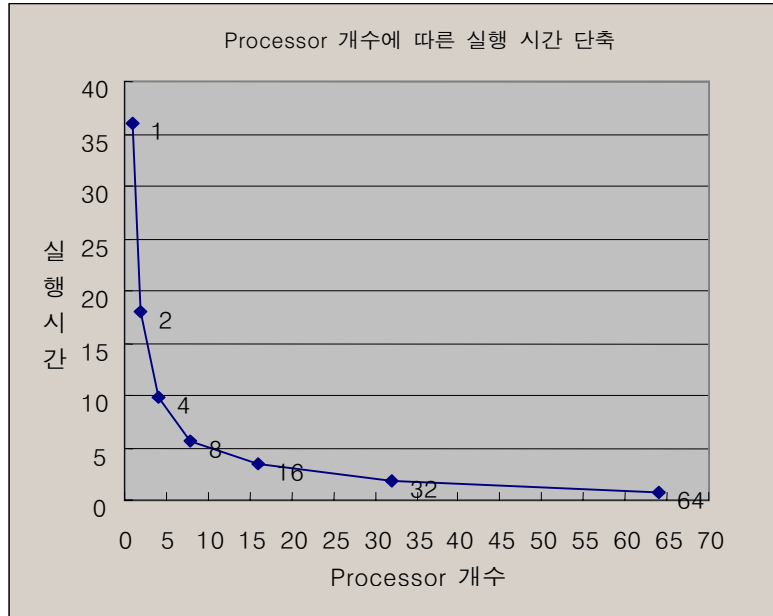
[소스 2] 병렬 처리 PSEUDO 코드

#### 4. 결 과

고안된 병렬 알고리즘으로 구현한 코드로 실험용 데이터를 계산하여 표 1, 표 2와 같은 수행시간을 얻었다. 실험은 두 가지 경우에 대하여 수행되었다. 먼저 프로세스 개수에 따른 속도 증가를 측정하였으며, 두 번째로는 독립 변수의 수에 따른 수행 시간을 측정하였다.

프로세서 개수에 대한 실험에서 그림 3에 보이듯이 SpeedUp이 거의 선형으로 나타났다. 이 실험에 사용된 데이터는 34개의 독립 변수와 10개의 데이터 집합(set)으로 이루어진 것이다. 이 실험 결과는 순차 프로그램에 비해 두 개의 프로세서만 사용해도 MPI로 약 절반의 시간 단축으로 정확한 결과를 볼 수 있다는 것을 보여 준다.

독립 변수의 수행 시간은 그림 4에 보이듯이 독립 변수의 수가 증가하면 기하 급수적으로 증가한다. 따라서, 많은 수의 독립 변수에 대해서는 병렬 알고리즘이 더욱 효과적이라고 볼 수 있다.

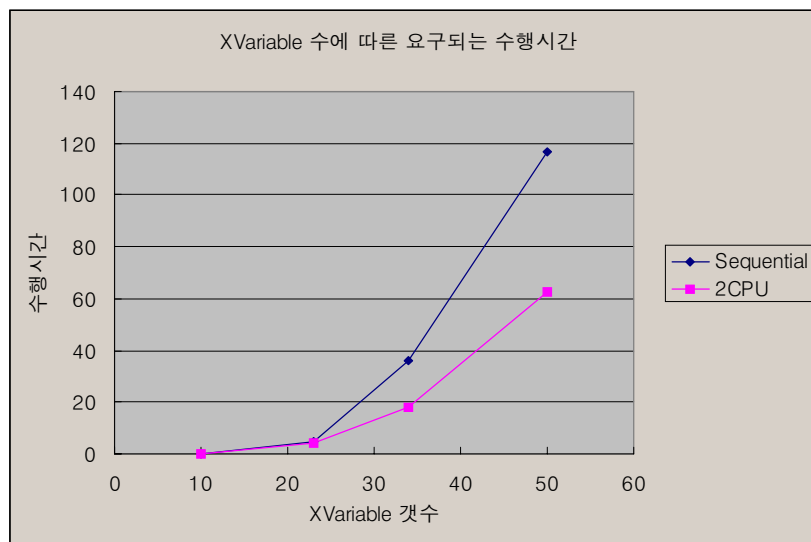


[그림 3] 프로세서 개수에 따른 실행 시간 단축

표 1. 프로세서 수당 수행시간 결과

	1	2	4	8	16	32	64
실행시간	36.0	17.9	9.78	5.70	3.38	1.82	0.81
Offset	561	280	140	70	35	17	8

# 이 실험에 사용된 데이터는 34 개의 독립 변수와 10 개의 데이터 집합(set)으로 이루어졌다. offset 은 한 프로세서가 계산에 사용하고 있는 두 개의 원소로 이루어진 부분 집합의 개수다. 소스 2 에서 nC2/Pe 에 해당하는 값이다.



[그림 4] 독립 변수의 수에 따른 수행 시간

[표 2] 독립 변수의 수에 따른 수행 시간

노드	개수	10	23	34	50
1		0.2	5.02	36	116.62
2		0.175	4.448	17.9361	62.34

## 5. 토 의

KDMLR 알고리즘은 종래의 MLR 과 달리 반복법으로 계산하므로 알고리즘과 tolerance criteria 에 따른 성능 차이가 심하게 발생할 수 있다. 따라서, 사전에 결정 계수값의 허용 오차를 예상할 수 있으면 반복을 적정선에서 중단할 수 있을 것이다.

KDMLR 알고리즘은 embarrassingly parallel 알고리즘 범주에 속해서 초기에 주어지는 관측 데이터를 functionally parallel 하게 사용하지만 서로 communication 을 하지 않는 특징으로 프로세서 수에 따른 성능 향상 효과가 크다.

MLR 에서 주어진 데이터에 의존하여  $R^2$  값이 결정되므로 주어진 데이터에서  $Y$  에 영향이 큰  $X$  의 독립 변수를 포함하는 집합을 초기에 결정할 수 있다면 STEP 2 이후에서 모든 변수에 대하여 고려하므로 선택  $K$  의 크기를 3 ~ 4 정도를 줄여서 동일 결과를 생성 할 수 있을 것이다. 따라서,  $O(N^2)$ 에 가능한 획기적인 방법도 가능할 수 있을 것 같다.

주어진 초기 데이터가 그 이전부터 알려져 있다면  $R^2$  의 upper bound 를 모르는 상황에서는 KDMLR 의 경우는 계산 시간과  $R^2$  의 허용 범위를 모르기 때문에 반복 수행 및 시행 착오를 반복하여야 한다. 따라서, upper bound 를 확인할 수 있는 통계적 방법의 고안은 무엇보다 중요하다고 보아 진다.

## 6. 결 론

본 프로젝트는 순차 MLR 알고리즘의 하나인 KDMLR 의 병렬 알고리즘을 설계하고 구현하는 것이었다. 본 프로젝트의 결과는 KDMLR 의 병렬화 프로그램을 이용하여 순차 프로그램과 동일한 정확성을 유지하면서 실행시간을 대폭 단축할 수 있었다. 이런 결과를 얻을 수 있었던 이유는 KDMLR 알고리즘의 성격이 embarrassingly parallel 하여 반복 계산에 있어서 서로 다른 프로세서 간에 거의 통신 없이 가능했기 때문이었다. 따라서, 대규모의 데이터 처리를 위하여 순차 KDMLR 알고리즘을 사용하는 것보다 PKDMLR 을 사용하는 것이 효과적일 것이다.

본 프로젝트에서는 MPI 로 PKDMLR 의 설계 및 구현을 하였지만, pThread 등으로 구현하여도 동일한 효과를 기대할 수 있을 것으로 보인다. 그리고 앞으로 충분히 많은 데이터와 관련연구를 통해서 성능 비교 및 정확성 검증을 해야 할 것이다.

## 7. 참고 문헌

- [1] Scheffer, McClave, *Probability and Statistics for Engineers*, Thomson Information, 1990
- [2] Peter S. Pacheco, *A User's Guide to MPI*, Department of Mathematics, University of San Francisco, March 30, 1998
- [3] Ian Foster, *Designing and Building Parallel Programs*, Addison-Wesley, 1995
- [4] Barry Wilkinson, Michael Allen, *Parallel Programming*, Prentice Hall, 1999
- [5] Alberto Leon-Garcia, *Probability and Random Processes for Electrical Engineering*, Addison-Wesley, 1994