بسمه تعالی

عنوان درس: مباحث پیشرفته در زبانهای برنامه نویسی موازی

استاد: آقای دکتر سوادی

پروژه : پیاده سازی و موازی سازی الگوریتم Gradient descend linear regression

اعضاء گروه : محمد دانش آموز – مهسا زاهدی

توضیحات فاز موازی سازی:

در موازی سازی این الگوریتم با چالشهای مختلفی روبرو شدیم که مهمترین آنها بحث false sharing بود که باعث میشد در حالت موازی زمان اجرای الگوریتم بیشتر شود، از دیگر موارد زمان و هزینه ایجاد thread در سیستم عاملهای مختلف بود که در لینوکس این زمان خیلی کمتر از ویندوز است و نهایتا اجرای برنامه نهایی در ویندوز و لینوکس ، نتایج مختلفی داشتند. همچنین به دلیل اینکه کل این الگوریتم تشکیل شده از ضربهای ماتریس در بردار است ، موازی سازیم منجر به اجراهای کوتاه در هر thread می گردید و عملا امکان استفاده از حالت اتمیک و منطقه بحرانی نبود (زمان اجرا بشدت تحت تاثیر قرار میگرفت و حتی طولانی تر میشد). از این رو با بررسی هایی متوجه شدیم که بیش از 90 درصد پردازنده های اخیر دارای کش لاین 64 بایتی هستند که چون متغیرهایی از نوع double که 8 بایتی هستند را در حلقه ها استفاده میکردیم ، بهترین حالت موازی سازی هر حلقه در این برنامه ، استفاده از حالت schedule (static,8) است.

لینک پروژه :

https://github.com/mohammaddan/ferdowsi-gradient-descend-regression.git

برنامه را در چند حالت مختلف از جمله سریال، موازی سازی با ۱،۲،۳،۴،۵،۶،۷،۸ هسته اجرا کردیم.

نهایتا برای بهینه سازی زمان اجرا در حالت موازی (کاستن زمانهای سربار جهت ایجاد تردها)، برنامه را تغییراتی دادیم و تاحد امکان، حلقه های تکرار را در هم ادغام کردیم تا در هر ترد، کارهای بیشتری انجام شود. تابع think3 پیاده سازی نهایی و بهینه این الگوریتم است.

نتایج اجرای پیاده سازی های مختلف با تعداد تردهای مختلف به شکل زیر میباشد:

CPU : AMD 8350 8 core – RAM 16GB DDR3 freq: 4Ghz – cache size: 16MB – cache line size : 64Byte

OS : Ubuntu 20.04.2

```
md@md-desktop ~/Projects/C++/gradient-descend-regression ᚠ main ± g++ ./main.cpp -o main.out -std=c++14 -fopenmp && ./main.out
Sequense first implement (tink0 function)
theta={ 99.8513,2.02146,4.02466,0.515641,1.01715,4.01338,3.02082,2.01877,1.02,2.028,-1.98196,5.0235,7.01634,14.0264,23.0218,37.0233 }
parallel seconad implement (tink2 function)
theta={ 99.8513,2.02146,4.02466,0.51564,1.01715,4.01338,3.02082,2.01876,1.02,2.028,-1.98196,5.02349,7.01634,14.0264,23.0218,37.0233 }
parallel third implement (tink3 function)
theta={ 99.8513,2.02145,4.02465,0.515639,1.01715,4.01338,3.02082,2.01876,1.02,2.02799,-1.98197,5.02349,7.01633,14.0264,23.0217,37.0233 }
Implements/Number of threads                    1      2      3      4      5      6      7      8
Sequense first implement (tink0 function)     5.614
parallel seconad implement (tink2 function)   5.987   4.35  3.856  3.901  3.816  3.905  4.778  4.898
parallel third implement (tink3 function)       5.8  4.117  3.212  2.433  2.149  1.907  1.853  1.605
md@md-desktop ~/Projects/C++/gradient-descend-regression ᚠ main ±
```

همانطور که در تصویر بالا قابل مشاهده است، در پیاده سازی موازی اولیه، به علت کمتر بودن عملیات هر ترد، در صورت افزایش تعداد تردها، نتایج بدتر میشوند. ولی در پیاده سازی بهینه، با افزایش تعداد تردها (حداکثر به اندازه تعداد هسته های پردازنده که در اینجا ۸ هسته است) نتایج بهتر میشوند.

CPU : Intel 7700 4 core – 8 hyper thread - freq: 4Ghz – cache size: 16MB – cache line size : 64Byte – RAM 16GB DDR4

OS : Windows 10

```
Sequense first implement (tink0 function)
theta={ 99.8626,2.01773,4.05266,0.518423,1.01826,4.01394,3.0183,2.02074,1.02327,2.01397,-1.97593,5.0212,7.01557,14.0219,23.008,37.0191 }
Parallel seconad implement (tink2 function)
theta={ 99.8626,2.01773,4.05266,0.518422,1.01826,4.01394,3.0183,2.02073,1.02327,2.01397,-1.97593,5.0212,7.01557,14.0219,23.008,37.0191 }
Parallel third implement (tink3 function)
theta={ 99.8626,2.01773,4.05266,0.518422,1.01826,4.01394,3.0183,2.02073,1.02327,2.01397,-1.97593,5.0212,7.01557,14.0219,23.008,37.0191 }
Implements/Number of threads                    1      2      3      4      5      6      7      8
Sequense first implement (tink0 function)     2.364
Parallel seconad implement (tink2 function)   4.245  3.101  2.903  2.814  3.019  4.879  5.732  6.003
Parallel third implement (tink3 function)     3.775  2.485  1.719   1.03   1.22  0.963  1.118  1.219
```

CPU : Intel 7700 4 core – 8 hyper thread - freq: 4Ghz – cache size: 16MB – cache line size : 64Byte – RAM 16GB DDR4

OS : Ubuntu 20.04.2

```
md@md-Aspire-A715-71G ~/Projects/ferdowsi-gradient-descend-regression ᚠ main ± g++ ./main.cpp -o main.out -std=c++14 -fopenmp && ./main.out
Sequense first implement (tink0 function)
theta={ 99.8409,2.01733,4.02033,0.517593,1.021,4.01134,3.02563,2.02475,1.02704,2.02681,-1.98135,5.02449,7.02152,14.021,23.0282,37.024 }
Parallel seconad implement (tink2 function)
theta={ 99.8409,2.01733,4.02033,0.517594,1.02101,4.01134,3.02563,2.02475,1.02704,2.02681,-1.98135,5.02449,7.02152,14.021,23.0282,37.024 }
Parallel third implement (tink3 function)
theta={ 99.8409,2.01732,4.02033,0.517592,1.021,4.01134,3.02562,2.02474,1.02703,2.02681,-1.98135,5.02449,7.02151,14.021,23.0282,37.024 }
Implements/Number of threads                    1      2      3      4      5      6      7      8
Sequense first implement (tink0 function)     4.459
Parallel seconad implement (tink2 function)   5.275  3.984  3.725  3.395  3.167  3.093  3.144  6.196
Parallel third implement (tink3 function)     4.951   3.31  2.497  1.966  1.965  1.826  1.739  3.145
md@md-Aspire-A715-71G ~/Projects/ferdowsi-gradient-descend-regression ᚠ main ±
```

```
void tink3(double *theta,int thread_num) {
  double alpha = 0.01;
  double err_0[n];

  for (int i = 0; i < p; i++)
    theta[i] = (rand() % 10) / 1000.0;

  double delta[p] = {0};
  double y_pred[n] = {0};
  int cnt = 2000000, it = 0;
  double last_error, error = 100;
  do {
    last_error = error;
    int j;
    double error2 = 0;

    #pragma omp parallel for private(j) num_threads(thread_num) schedule(static,8) reduction(+:error2)
    for (int i = 0; i < n; i++) {
      y_pred[i] = 0;
      for (j = 0; j < p; j++) {
        y_pred[i] += theta[j] * train_x[i][j];
      }
      err_0[i] = y_pred[i] - train_y[i];
      error2 += err_0[i] * err_0[i];
    }

    #pragma omp parallel for num_threads(thread_num) schedule(static,1)
    for (int j = 0; j < p; j++){
      double temp=0;
      for(int i=0;i<n;i++)
        temp += train_x[i][j] * err_0[i];
      theta[j] -= alpha * temp / n;
    }

    error = sqrt(error2);
    if (error > last_error)
      alpha /= 1.2;

  } while (it++ < cnt && fabs(error - last_error) > epsilon);

}
```