Programowanie równoległe i rozproszone

Politechnika Krakowska

Laboratorium 1

Paweł Suchanicz,
Rafał Niemczyk

15 października 2019

Spis treści

1 Wstęp

1.1 Opis laboratorium

Celem laboratorium było wykorzystanie interfejsu OpenMP w celu zrównoleglenia kodu C++. Interfejs OpenMP składa się głównie z dyrektyw preprocesora a także z zmiennych środowiskowych i funkcji bibliotecznych. W laboratorium wykorzystywany będzie głównie do zrównoleglania pętli.

Algorytmy, które są implementowane a następnie zrównoleglane w ramach laboratorium to normalizacja min-max, standaryzacja rozkładem normalnym i klasyfikacja KNN (k-najbliższych sąsiadów). Zaimplementowany KNN uwzględnia jednego sąsiada i używa metryki euklidesowej.

Szybkość działania każdego algorytmu została zmierzona dla implementacji w C++, implementacji w C++ po zrównolegleniu dla różnej ilości wątków (1-4) oraz impelmentacji w Python (ze skorzystaniem z funkcji z pakietu scikit-learn).

1.2 Specyfikacja sprzętowa

Przy pomiarach szybkości wykonywania algorytmów wykorzystany był sprzęt w konfiguracji (maszyna wirtualna):

- Procesor: Intel Core i7-4712MQ 4 x 2.30GHz
- Ram: 2GB DDR3
- System: Linux (Fedora 22)

1.3 Zbiór danych

Wykorzytany został zbiór obrazów ręcznie pisanych cyfr MNIST. Wykorzytany zbiór ma format .csv i zawiera 60000 rekordów, gdzie każdy rekord odpowiada za jeden obrazek 28x28 pikseli w skali szarości. Pierwsza wartość w rekordzie jest cyfrą która widnieje na obrazku, a kolejne to wartości pikseli obrazka.

Dla zadań postawionych w laboratorium zbiór danych jest dość duży, więc został on obcięty do pierwszych 6000 rekordów, z czego 4500 przeznaczono do trenowania, a pozostałe 1500 do testowania.

2 Wyniki

2.1 Normalizacja min-max

Wzór:

$x^* = \frac{x - min(x)}{max(x) - min(x)}$

2.1.1 Implementacja

W C++ normalizacja została samodzielnie zgodnie z podanym powyżej wzorem. W pętli przechodzącej tablicy (po kolumnach) wyszukiwane są wartości minimum i maxium dla każdej kolumny a następnie wyliczana nowa wartość dla każdego z elementów tablicy. Zrównoleglenie pętli za pomocą dyrektyw:

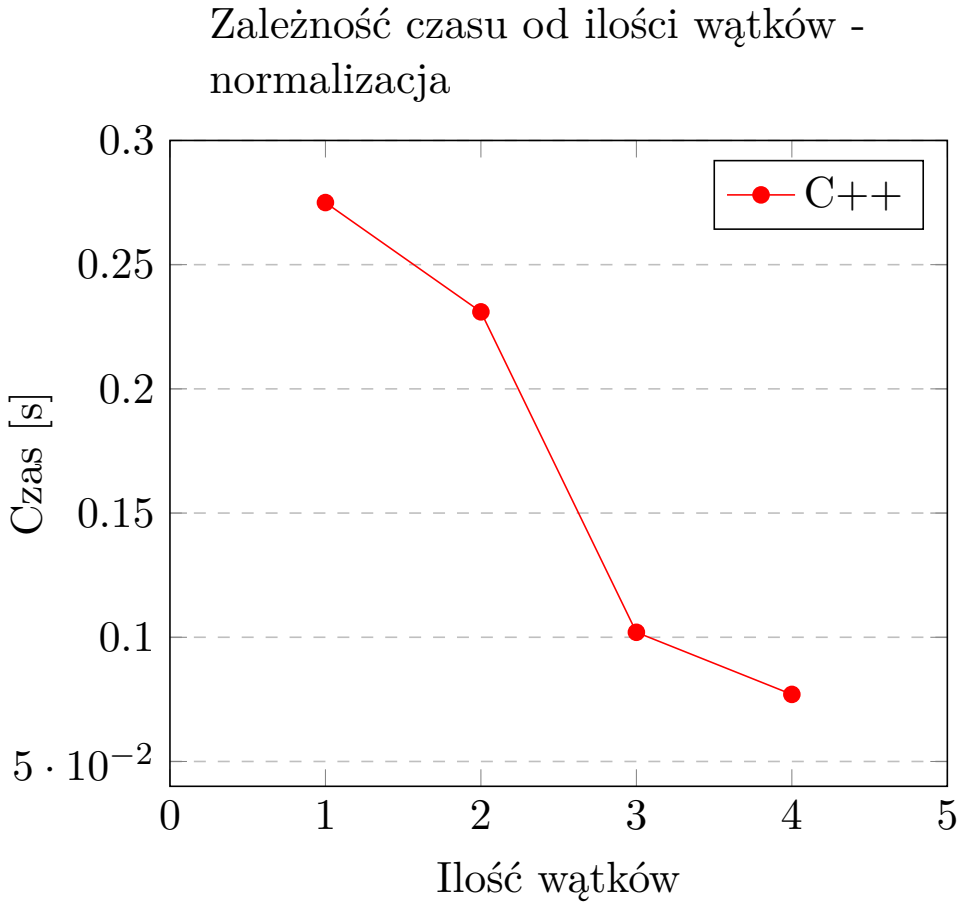
```
# pragma omp parallel default(none) private(i, j, min, max)
  shared(data, rows, columns, nr_threads) num_threads(nr_threads)
# pragma omp for schedule(dynamic, nr_threads)
```

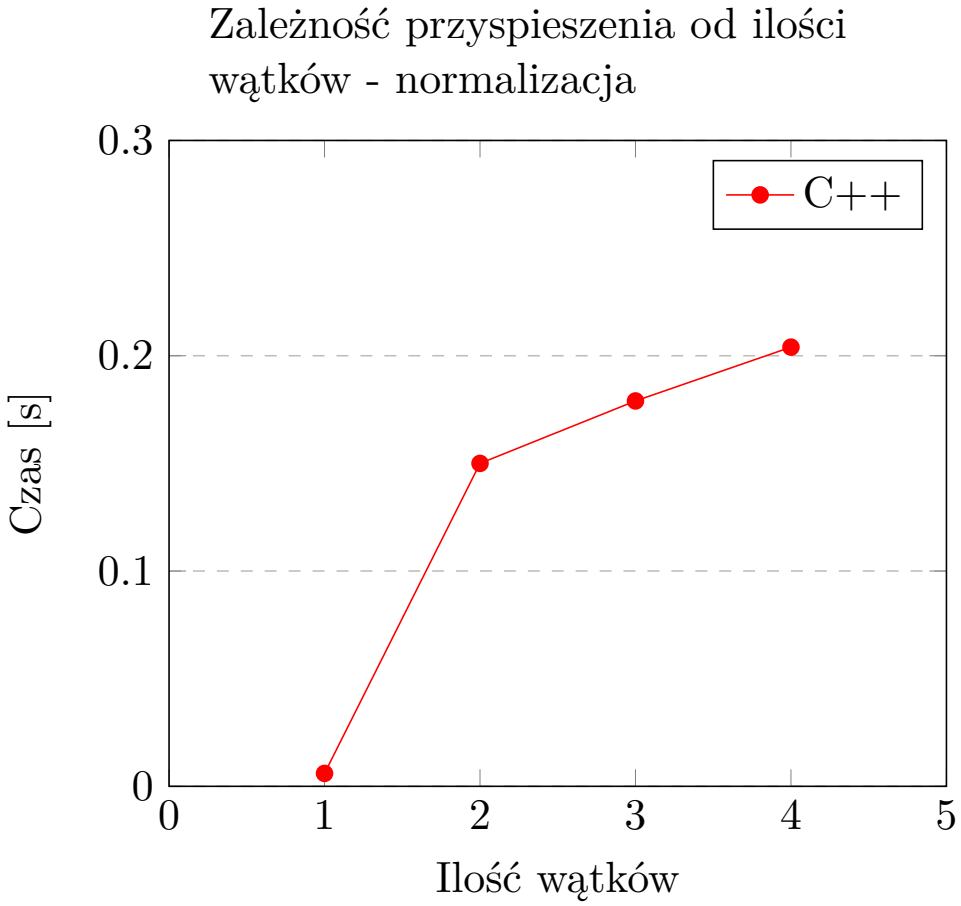
W Pythonie użyta została funkcja MinMaxScaler z pakietu sklearn .

2.1.2 Porównanie wyników

Parametry	Czas [s]
C++	0.281
C++ OpenMP 1 wątek	0.275
C++ OpenMP 2 wątki	0.131
C++ OpenMP 3 wątki	0.102
C++ OpenMP 4 wątki	0.077
Pyhon sklearn	0.037

Po zastosowaniu OpenMP i zwiększeniu ilości używanych wątków widać znaczącą poprawę czasu wykonania. Czas zmniejsza się proporcjonalnie wraz ze zwiększanie liczby wątków.


Zależność czasu od ilości wątków - normalizacja
0.3
0.25
0.2
0.15
0.1
$5 \cdot 10^{-2}$
Czas [s]
C++
0
1
2
3
4
5
Ilość wątków


Zależność przyspieszenia od ilości wątków - normalizacja
0.3
0.2
0.1
0
Czas [s]
C++
0
1
2
3
4
5
Ilość wątków

2.2 Standaryzacja rozkładem normalnym

Wzór:

$x^* = \frac{x-\mu}{\sigma}$

2.2.1 Implementacja

W C++ standaryzacja została samodzielnie zgodnie z podanym powyżej wzorem. Przechodzimy w pętli po kolumnach i dla każdej kolumny szukamy wartości średniej i wariancji, a następnie wyliczamy nowe wartości dla każdego elementu tablicy. Zrównoleglenie pętli za pomocą dyrektyw:

```
#pragma omp parallel default(none) private(i, j, amo, var, ave)
shared(data, rows, columns, nr_threads) num_threads(nr_threads)
#pragma omp for schedule(dynamic, nr_threads)
```

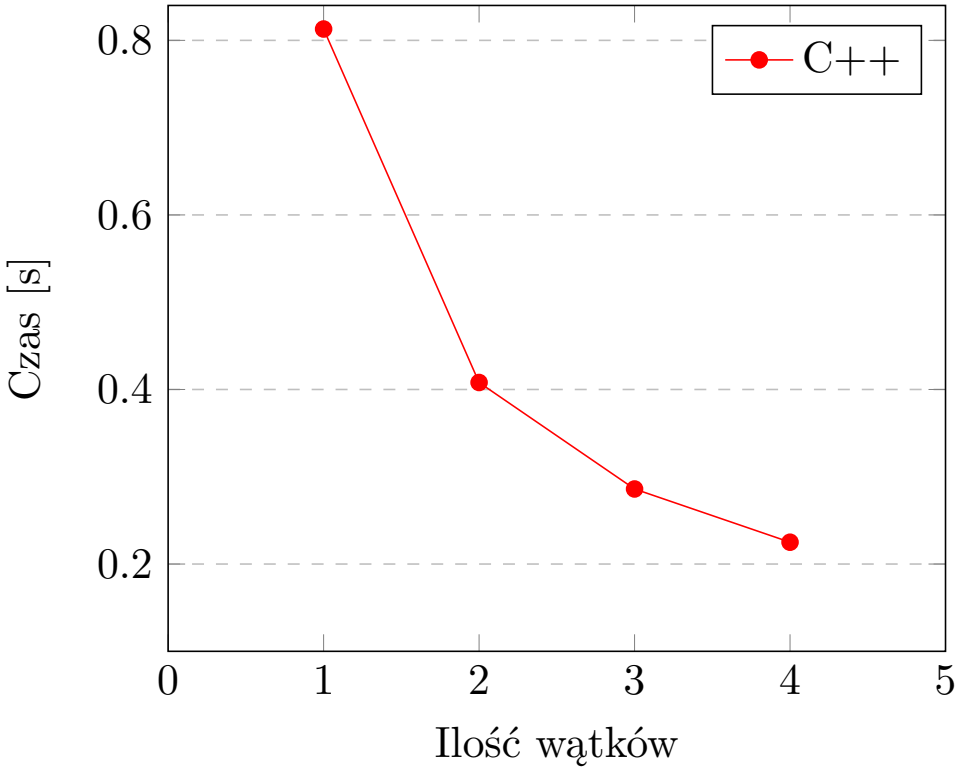
W Pythonie użyta została funkcja StandardScaler z pakietu sklearn.

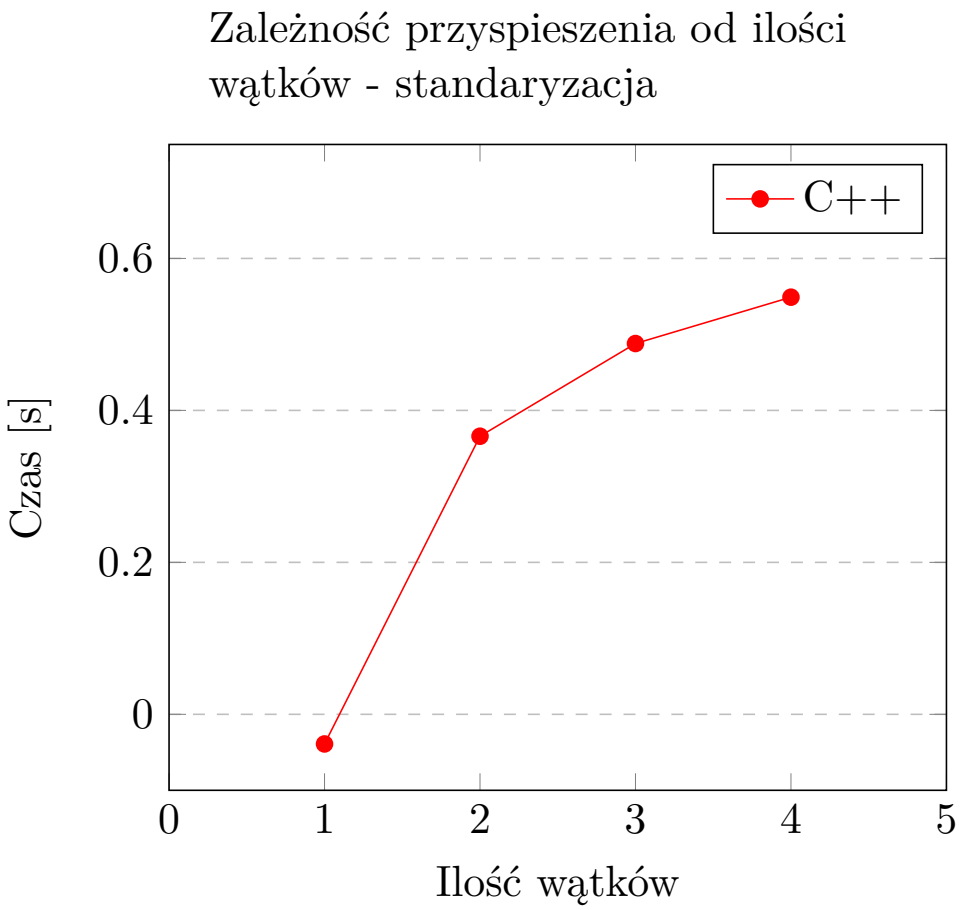
2.2.2 Porównanie wyników

Parametry	Czas [s]
C++	0,774s
C++ OpenMP 1 wątek	0.813
C++ OpenMP 2 wątki	0.408
C++ OpenMP 3 wątki	0.286
C++ OpenMP 4 wątki	0.225
Pyhon sklearn	0.086

Zrównoleglenie w C++ przyniosło pozytywne sutki. Czas wykonania spadł dwukrotnie przy użyciu dwóch wątków i prawie czterokrotnie przy użyciu czterech.

Zależność czasu od ilości wątków - standaryzacja




Zależność przyspieszenia od ilości wątków - standaryzacja
C++
0.6
0.4
0.2
0
Czas [s]
0
1
2
3
4
5
Ilość wątków

2.3 Klasyfikacja KNN

2.3.1 Implementacja

W C++ algorytm k najbliższych sąsiadów zaimplementowany samodzielnie. Algorytm uwzględnia tylko najbliższego sąsiada i korzysta z metryki euklidesowej. Zrównoleglenie za pomocą dyrektyw:

```
#pragma omp parallel default(none) private(i, j, metric, amount, searchRo
shared(trainData, dataTrainRows, columns, nr_threads) num_threads(nr_thr
#pragma omp for schedule(dynamic, nr_threads)
```

W Pythonie użyta została funkcja KNeighborsClassifier z pakietu sklearn z parametrami:

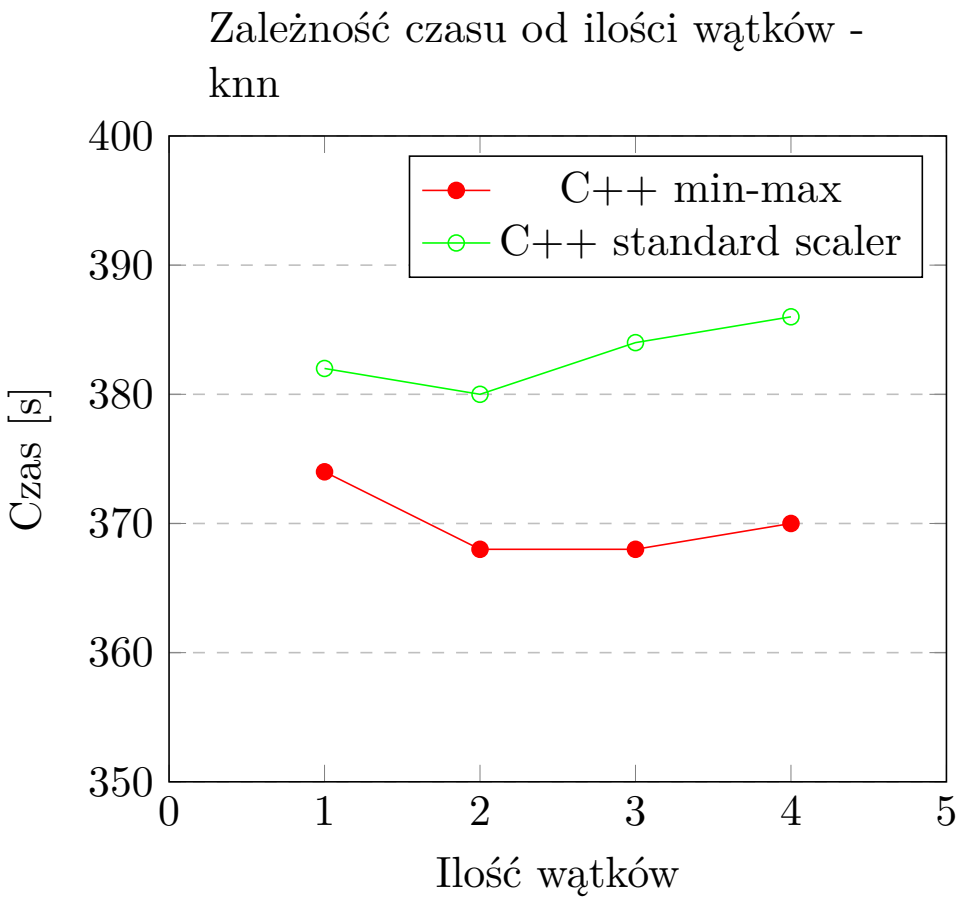
```
KNeighborsClassifier(n_neighbors=1, algorithm='brute', p=2, metric='minkowski',
n_jobs=app_conf['jobs_number'])
```

Czasy były mierzone dla wartości njobs od 1 do 4.
Dokładność accuracy wyniosła 71% dla danych po standard scalerze oraz 66% dla danych po min-max scalarze. Dla c++ w przypadku normalizacji min-max otrzymano dokładność. W przypadku normalizacji w c++ otrzymano dokładność 75%. Inaczej okazało się w przypadku standaryzacji, gdzie otrzymywano bardzo niskie wartości dokładności. Prawdopodobnie spowodowane błędem w implementacji algorytmu.

2.3.2 Porównanie wyników

Parametry	Czas [s]
C++ OpenMP 1 wątek min-max	374
C++ OpenMP 2 wątki min-max	368
C++ OpenMP 3 wątki min-max	368
C++ OpenMP 4 wątki min-max	370
Pyhon sklearn 1 wątek min-max	0.215
Pyhon sklearn 2 wątki min-max	0.323
Pyhon sklearn 3 wątki min-max	0.455
Pyhon sklearn 4 wątki min-max	0.386
C++ OpenMP 1 wątek standard-scaler	382
C++ OpenMP 2 wątki standard-scaler	380
C++ OpenMP 3 wątki standard-scaler	384
C++ OpenMP 4 wątki standard-scaler	386
Pyhon sklearn 1 wątek standard-scaler	0.208
Pyhon sklearn 2 wątki standard-scaler	0.326
Pyhon sklearn 3 wątki standard-scaler	0.329
Pyhon sklearn 4 wątki standard-scaler	0.328

Zrównoleglenie nie dało żadnych pozytywnych skutków. W przypadku c++ czas wykonania był bardzo duży i nie zmieniał i pozostawał stały przy próbach zrównoleglania i zwiększania ilości wątków. W przypadku Python zwiększanie parametru njobs algorytmu KNN przynosiło odwrotny skutek do oczekiwanego - czas wykonania wydłużał się.


Zależność czasu od ilości wątków - knn
400
390
380
370
360
350
Czas [s]
0
1
2
3
4
5
Ilość wątków
C++ min-max
C++ standard scaler

Zależność czasu od ilości wątków - knn