Algorytm Brandesa

Filip Binkiewicz 332069

1. Operacje wejścia-wyjścia

Wczytywanie danych oraz zapis do pliku wykonywane są przez główny wątek. Implementacja tych operacji znajduje się w `in-out.cpp`

2. Sekwencyjny algorytm Brandesa

W pliku `brandes-sequential.cpp` zaimplementowany jest algorytm Brandesa w wersji jednowątkowej. Jest to w zasadzie pseudokod przepisany na C++. Polecenie

```
./brandes 1 file_in file_out
```

spowoduje wywołanie funkcji `sequential_brandes`. Funkcja ta ma jedynie pomóc w zrozumieniu algorytmu oraz przy testach wydajnościowych oraz poprawnościowych.

3. Zrównoleglony algorytm Brandesa

Implementacja zrównoleglonego algorytmu Brandesa znajduje się w pliku `brandes-parallel.cpp`. Poniżej opis algorytmu:

- Wątek główny tworzy stos ze wszystkich wierzchołków
- Tworzonych jest `num_threads` identycznych wątków
- Każdy z nich wykonuje w pętli następujący schemat operacji:
  - Jeśli stos jest pusty, wątek kończy pracę
  - Wątek pobiera ze stosu identyfikator wierzchołka
  - Wątek wykonuje dla tego wierzchołka odpowiedni obrót pętli sekwencyjnego algorytmu
- Synchronizacja dostępu do danych współdzielonych jest realizowana przy użyciu mutexa

4. Wydajność

Wydajność testowana była na komputerze Dell Inspiron z procesorem Intel i3 (poniżej wynik polecenia `lscpu`)

```
$ lscpu | head
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    2
Core(s) per socket:    2
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
```

Dane do testowania wydajności pochodzą ze strony polecanej w treści zadania. Przykładowe wywołania:

Jeden wątek

```
$ time ./brandes 1 ../../data/facebook/107.edges fb_out.txt

real    0m11.867s
user    0m11.856s
sys 0m0.008s
```

Cztery wątki

```
$ time ./brandes 4 ../../data/facebook/107.edges fb_out.txt

real    0m5.355s
user    0m20.728s
sys 0m0.292s
```

Dziesięć wątków

```
$ time ./brandes 10 ../../data/facebook/107.edges fb_out.txt

real    0m5.551s
user    0m21.668s
sys 0m0.076s
```

Sto wątków

```
$ time ./brandes 100 ../../data/facebook/107.edges fb_out.txt

real    0m6.903s
user    0m25.656s
sys 0m0.452s
```

5. Przyspieszenie

Niestety podczas pisania tego raportu nie mam dostępu do szybkiego połączenia z Internetem, wobec czego zamiast na wymaganych przez zadanie danych (Wikipedia) test przyspieszenia wykonałem na załączonym w archiwum pliku 107.edges, pochodzącym z danych publikowanych przez Facebooka.

Do testowania przyspieszenia służy skrypt `acceleration.sh`. Należy go wywoływać z parametrami `liczba-watkow` oraz `plik-wejsciowy` Poniżej jego przykładowe wywołanie oraz wynik:

```
$ ./acceleration.h 5 107.edges

Brandes algorithm executed by 1 thread:
Time: 12.82 seconds

Brandes algorithm executed by 2 threads:
Time: 6.94 seconds
Acceleration: 1.8472622473861671469

Brandes algorithm executed by 3 threads:
Time: 6.10 seconds
Acceleration: 2.1016393442622950819 6

Brandes algorithm executed by 4 threads:
Time: 5.41 seconds
Acceleration: 2.3696857670979667282 8

Brandes algorithm executed by 5 threads:
Time: 5.61 seconds
Acceleration: 2.2852049910873440285 2
```

6. Optymalizacja przy kompilacji

Przy kompilacji program `cmake` nakazuje optymalizować kod, wykorzystując opcję `-O2`