Projekt z przedmiotu

Programowanie współbieżne i rozproszone

GAME OF LIFE

Erlang

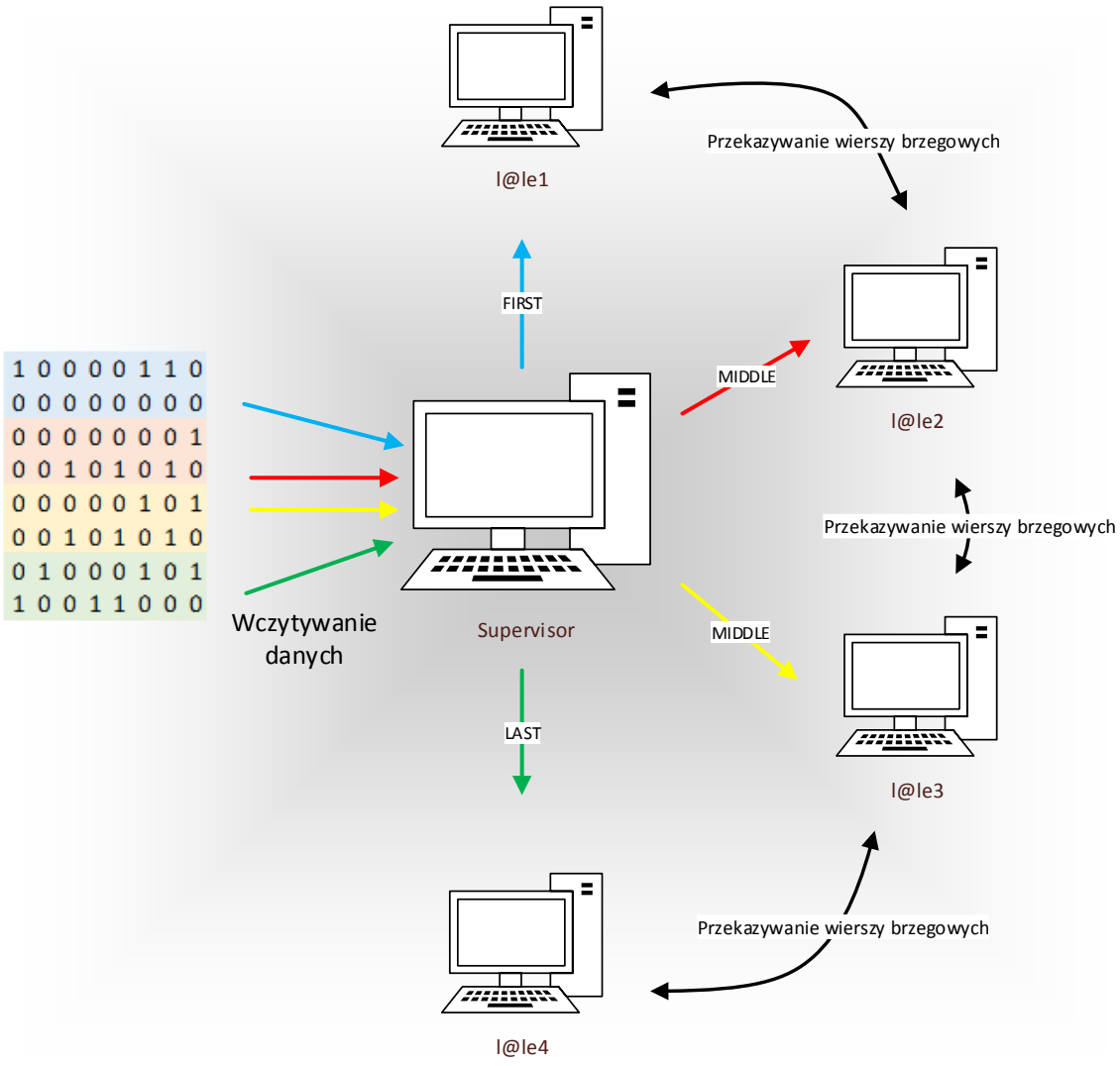
Spis treści

1. Skład osobowy

- Antos Jacek
- Korecki Tomasz
- Pelc Jakub
- Schaefer Maja
- Wicher Łukasz

2. Opis przyjętego rozwiązania

2.1 Algorytm



1. *Supervisor* wczytuje z pliku planszę fragmentami. Znając rozmiar całej planszy oraz ilość węzłów przeznaczonych do obliczeń, *supervisor* wyznacza ile wierszy ma mieć dana cześć.
2. Na zdalnych węzłach inicjalizowane są procesy, które jako argument pobierają część tablicy, na której będą pracować.

3. Węzły oczekują na polecenie *start*, którym *supervisor* daje sygnał do rozpoczęcia obliczeń.
4. *Supervisor* poleceniem *next* wysłanym do węzłów zmusza je do liczenia kolejnej iteracji. Węzły komunikują się między sobą, przesyłając wiersze brzegowe.
5. Program działa aż do momentu wysłania przez *supervisora* polecenia *stop*. Wtedy wszystkie węzły przesyłają swój fragment planszy do *supervisora*, który to zapisuje wynik obliczeń do pliku.

2.2 Opis poszczególnych funkcjonalności

2.2.1 Generowanie losowej tablicy o zadanym rozmiarze

Program posiada funkcję `lifeio:writeRandomBoard` pobierającą argumenty `Filename` oraz `Size`. Funkcja ta generuje oraz zapisuje do pliku tablicę o rozmiarze 2^{Size} wypełnioną losowymi liczbami ze zbioru {0, 1}.

2.2.2 Wczytywanie i dzielenie danych

Dane wczytywane są za pomocą funkcji `lifeio:readPartBoard`, która wczytuje z pliku liczbę wierszy określoną w argumencie `HowManyRows`. Funkcja ta zwraca dwuwymiarową tablicę. Ostatnia funkcja to *readBoardAsList*. Wczytywana plansza dzielona jest na tyle części ile zdalnych węzłów będzie używanych do obliczeń. Plansza dzielona jest wierszami. Przykładowo, jeżeli mamy planszę o wymiarze 1000x1000 i 8 węzłów – każdy z nich będzie pracował na tablicy zawierającej 125 wierszy i 1000 kolumn.

2.2.3 Warunki brzegowe

Dla fragmentu planszy o wymiarze *M x N*, tworzona jest tablica o wymiarach *M+2 x N+2*. Skrajne wiersze i kolumny wypełnione są 0.

2.2.4 Przekazywanie obliczeń do węzłów

Po wczytaniu danej części tablicy następuje przesłanie jej do węzła. *Supervisor* inicjuje procedury obliczające w zdalnych węzłach, przekazując jako argument część tablicy, na której dany węzeł będzie pracował.

2.2.5 Zarządzanie węzłami

Supervisor ma możliwość wysłania do węzłów następujących poleceń:

- *start* – po przesłaniu do wszystkich węzłów danych na których mają pracować, *supervisor* tym poleceniem rozpoczyna działanie algorytmu. Jednocześnie każdy węzeł otrzymuje informację o typie (*first*, *middle* lub *last*). Są to informacje istotne dla komunikacji między węzłami.
- *next* – polecenie wymuszające na węźle wykonanie następnej iteracji
- stop – polecenie kończące pracę. Jeżeli węzeł otrzyma to polecenie przerywa obliczenia, *wysyła* bieżącą tablicę do *supervisora*.

2.2.6 Komunikacja między węzłami

Wszystkie zdalne węzły mają przypisany jeden z trzech typów, które mówią o tym czy dany węzeł oblicza początek tablicy, jeden z elementów środkowych czy element końcowy. Węzeł początkowy potrzebuje jedynie komunikować się z węzłem pracujący na drugiej w kolejności części tablicy, węzeł końcowy wymaga jedynie komunikacji z węzłem liczącym przedostatnią część tablicy, natomiast węzły środkowe muszą porozumiewać się z węzłami pracującymi na części poprzedzającej jak i następującej po części tablicy danego węzła. Komunikacja między węzłami zawarta jest w funkcji `worker:next`.

2.2.7 Obliczanie nowej konfiguracji planszy

Po tym, jak funkcja `worker:next` przekaże do odpowiednich węzłów wiersze brzegowe wywoływana jest funkcja `worker:computeRows`, służąca do obliczania nowej konfiguracji planszy. Zliczanie żywych sąsiadów odbywa się poprzez sumowanie wartości w komórkach – sąsiadach. Gra działa zgodnie z zasadami Conwaya (23/3).

2.2.8 Pobieranie wyników od węzłów i zapis wyników

Kiedy *supervisor* chce zakończyć działanie programu wysyła do wszystkich węzłów polecenie *stop*. Węzeł po otrzymaniu takiego polecenia przerywa pracę i odsyła swoją cześć tablicy. *Supervisor* odbiera poszczególne części tablicy, a następnie w odpowiedniej kolejności zapisuje je do pliku używając funkcji `lifeio:writeParfOfBard`.

3. Opis testu skalowalności

Naszym zamiarem było przedstawienie tabeli z dokładnym porównaniem czasów wykonania iteracji w zależności od rozmiaru planszy i ilość użytych węzłów jednak ze względu na występujące w ostatnich dniach problemy (niedostępność niektórych węzłów oraz problemy z połączeniem z tymi, które są dostępne) nie byliśmy w stanie tego zrobić. Nie mniej w trakcie testów zauważone zostały następujące zależności:

- wraz ze wzrostem liczby węzłów zmniejsza się czas wykonania (nie jest to jednak spadek proporcjonalny, 2 razy więcej węzłów nie oznacza 2 razy krótszego czasu wykonania)
- uzależnione jest to jednak od rozmiaru planszy (górna granica wzrostu wydajności dla planszy 256 x 256 to około 2 – 3 węzły, wraz ze wzrostem rozmiaru planszy granica ta rośnie)
- związane jest to z kosztem wymiany skrajnych wierszy pomiędzy procesami (nie opłaca się dzielić małej planszy na wiele części, gdyż wrasta ilość koniecznych do przesłania informacji, co ostatecznie powoduje spadek wydajności)

4. Opis użycia oraz przykładowe uruchomienie programu

Moduł `life_supervisor` udostępnia następujące funkcje:

- `init(HowManyNodes, Proccesses, Filename)` – funkcja inicjalizująca, próbuje ona znaleźć `HowManyNodes` węzłów (jeśli się nie uda program o tym informuje i kończy działanie), wczytuje planszę z pliku o nazwie `Filename` oraz na każdym z węzłów uruchamia `Proccesses` procesów, plansza jest więc dzielona pomiędzy `Proccesses*Filename` procesów. Funkcja zwraca listę PID'ów, którymi posługujemy się w wywołaniach innych funkcji
- `next(N, Nodes)` – funkcja `N` wykonuje iteracji algorytmu korzystając z listy PID'ów `Nodes`, zwróconych przez funkcję `init`
- `test_time(N, Nodes)` – funkcja wywołuje `N` razy funkcję `next(1, Nodes)`, po czym oblicza statystyki czasowe dotyczące wykonania jednej iteracji algorytmu (w mikrosekundach); `Nodes` to lista PID'ów zwróconych przez funkcję `init`
- `stop(Nodes, Filename)` – funkcja zatrzymuje wszystkie procesy w zdalnych węzłach i zapisuje wynikową planszę w pliku o nazwie `Filename`; `Nodes` to lista PID'ów zwróconych przez funkcję `init`

Kompilacja i przykładowe uruchomienie wyglądają następująco:

```
(jakupel1@borg)3> c(lifeio).
{ok,lifeio}
(jakupel1@borg)4> c(worker).
{ok,worker}
(jakupel1@borg)5> c(life_supervisor).
{ok,life_supervisor}
(jakupel1@borg)6> lifeio:writeRandomBoard("board.gz",8).
Otwieranie pliku board.gz do zapisu...OK
Zapis planszy o rozmiarze 256x256...OK
ok
(jakupel1@borg)7> Nodes = life_supervisor:
init/3         module_info/0  module_info/1  next/2         stop/2
test_time/2
(jakupel1@borg)7> Nodes = life_supervisor:init(2,1,"board.gz").
Otwieranie pliku board.gz do odczytu...OK
Thread created: <6512.24492.0>
Thread created: <6555.32321.0>
Computing started
Computing started
[<6512.24492.0>,<6555.32321.0>]
(jakupel1@borg)8> life_supervisor:test_time(100,Nodes).
Range: 61104 - 66177 mics
Median: 61512 mics
Average: 61577 mics
61577
(jakupel1@borg)9> life_supervisor:stop(Nodes,"final.gz").
Otwieranie pliku final.gz do zapisu...OK
Zapis planszy o rozmiarze 256x256...OK
ok
(jakupel1@borg)10>
```

6. Problemy

Doświadczyliśmy dziwnych problemów z węzłem l@le10. Gdy chcieliśmy go użyć do obliczeń, komunikacja z nim nagle ustawała, zawsze przy próbie wykonania pierwszej iteracji (niezależnie od tego czy pełnił rolę węzła pierwszego, środkowego czy też ostatniego). Wykonaliśmy kilka milionów iteracji algorytmu korzystając z różnych plansz, różnych konfiguracji (od 2 do 9 węzłów) i nigdy nie natrafiliśmy na żaden problem. Dodatkowo analizowaliśmy wiele razy

algorytm nie znajdując błędów, mogących powodować powyższą sytuację. Domyślamy się, że problemy te mogą być powiązane z konfiguracją węzła lub połączeniem (jak w przypadku występujących w ostatnich dniach problemów z innymi węzłami, o których wspomniano wcześniej).