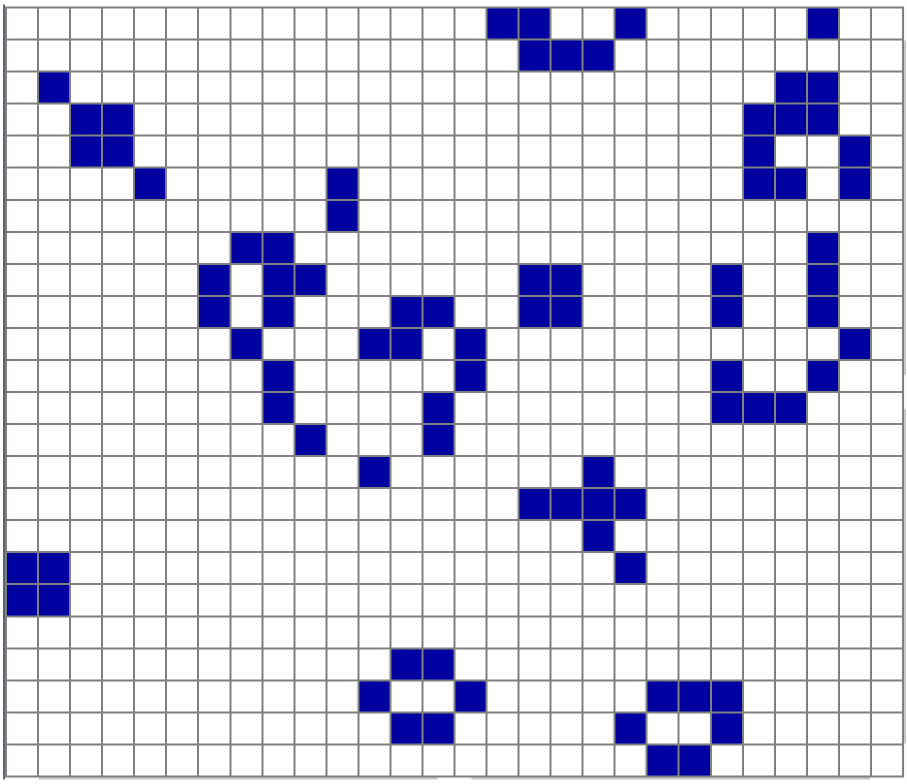
Dokumentacja projektowa

Współbieżna, rozproszona gra life w erlangu



Autorzy:

Małgorzata Maciurzyńska

Rafał Płonka

Konrad Seweryn

Mateusz Stanaszek

Mateusz Ścirka

1. Wprowadzenie teoretyczne

1.1. Cel projektu i wymagania

Celem projektu bylo opracowanie architektury rozproszonego, skalowalnego systemu w erlangu dla gry Life w/g podstawowej reguły Conwaya 23/3.

Rozmiar planszy jest kwadratowy będący potęgą 2 począwszy od 256×256 do 16384×16384 (rozmiar 8-14)

Program wykorzystuje **rozproszenie**.

Program uwzględnia, że nie wszystkie węzły będą zawsze dostępne i nie będą znikać w trakcie obliczeń.

Program posiada możliwość **generowania losowych plansz**.

Program posiada wbudowany **benchmark**.[1]

Program posiada funkcję **„next/0” , która wylicza następną iterację.**

Program ma możliwość **wczytania planszy z pliku i zapisu do pliku**.[2]

1.2. Reguły gry według Conwaya

Martwa komórka, która ma dokładnie 3 żywych sąsiadów, staje się żywa w następnej jednostce czasu (rodzi się)

Żywa komórka z 2 albo 3 żywymi sąsiadami pozostaje nadal żywa; przy innej liczbie sąsiadów umiera (z "samotności" albo "zatłoczenia").

---

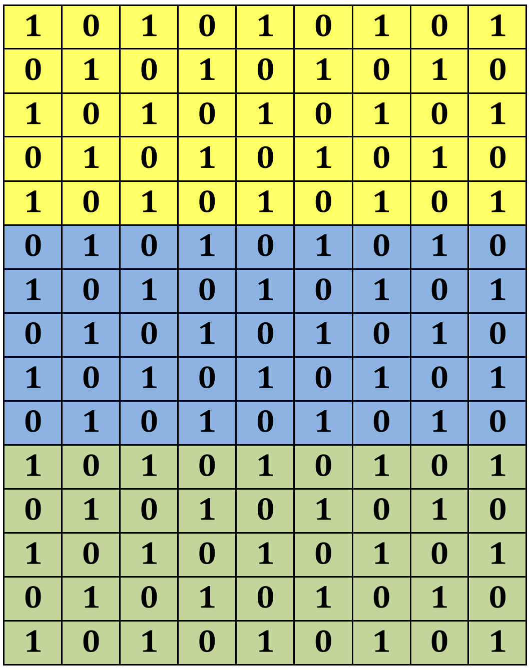
[1] https://erlangcentral.org/wiki/index.php/Measuring_Function_Execution_Ti

[2] Plik jest skompresowanym ciągiem zawierającym rozmiar jako pierwszy bajt (2^X, np. 12 oznacza planszę 2^12 na 2^12) oraz wartości poszczególnych komórek (0 lub 1) wierszami

2. Wykorzystywane algorytm

2.1. Dzielenie tablicy na mniejsze

Na poniższym rysunku przedstawiony jest poglądowy widok tablicy, na którym omówiony zostanie algorytm działania.

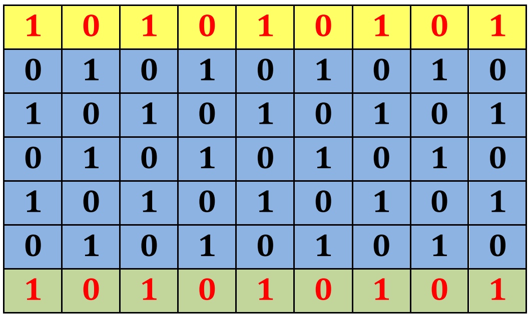


1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1

Tablica dzielona jest według kolorów na mniejsze podtablice. Dodawane są również wiersze w sąsiednich podtablic.

Przykład: niebieska tablica dostaje dodatkowy wiersz z tablicy żółtej i zielonej.

Każda z takich podtablic zostaje przesłana z dodatkową informacją gdzie należy dodać zera. Tak więc przykładowo node z tablicą niebieską otrzymuje:



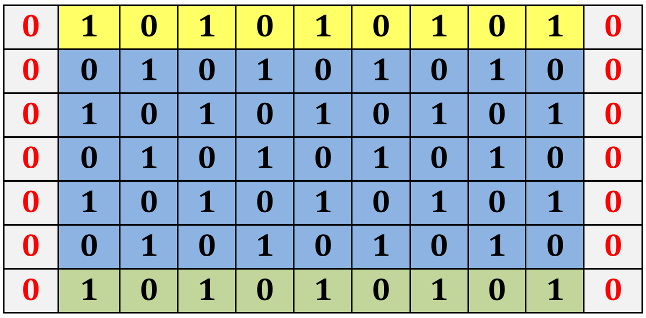
1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1

a także dane {left, right} wskazujące na miejsce umieszczenia zer.

2.2. Algorytm liczenia podtablicy

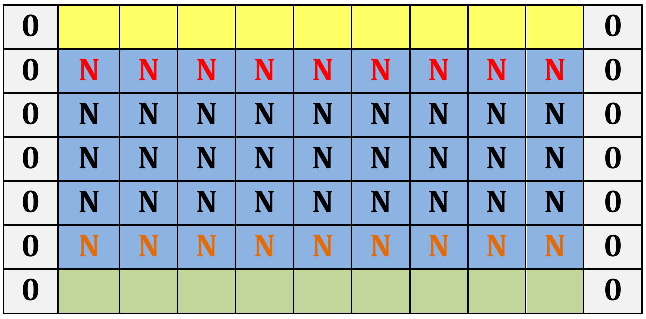
W każdym z node-ów do tablicy która się w nim znajduje dodawane są w odpowiedni sposób zera.

Przykład dla niebieskiej tablicy:



0	1	0	1	0	1	0	1	0	1	0
0	0	1	0	1	0	1	0	1	0	0
0	1	0	1	0	1	0	1	0	1	0
0	0	1	0	1	0	1	0	1	0	0
0	1	0	1	0	1	0	1	0	1	0
0	0	1	0	1	0	1	0	1	0	0
0	1	0	1	0	1	0	1	0	1	0

Kolejnym krokiem jest przeprowadzenie iteracji po wszystkich komórkach w kolorze niebieskim.



0										0
0	N	N	N	N	N	N	N	N	N	0
0	N	N	N	N	N	N	N	N	N	0
0	N	N	N	N	N	N	N	N	N	0
0	N	N	N	N	N	N	N	N	N	0
0	N	N	N	N	N	N	N	N	N	0
0										0

N jest nową wartością.

Node zapamiętuję tą tablicę, żeby nie trzeba było dodawać zer za każdym razem.

Sąsiednie komórki potrzebują jedynie informację z drugiego i przedostatniego wiersza, dlatego dany node wysyła do nadzorcy swoje 2 wiersza: top i down.

0	N	N	N	N	N	N	N	N	N	0

0	N	N	N	N	N	N	N	N	N	0

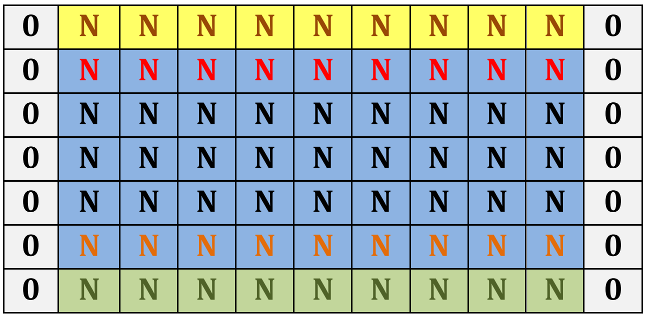
Kiedy nadzorca dostaje od wszystkich swoich node-ów informację o nowych wierszach, a liczba iteracji jeszcze się nie skończyła , przesyła odpowiednie wiersze do odpowiednich node-ów.

Do node-a z niebieską tablicą przesłana zostaje informacja od node-ów zawierającego tablice żółtą oraz zieloną.

0	N	N	N	N	N	N	N	N	N	0

0	N	N	N	N	N	N	N	N	N	0

Następnie na nodzie zostaje podmieniony pierwszy i ostatni wiersz.

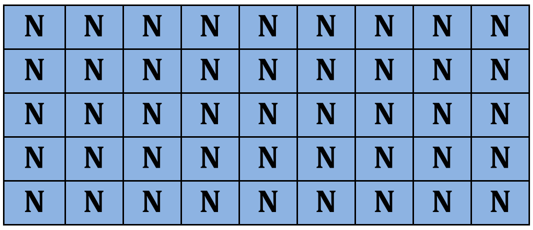


0	N	N	N	N	N	N	N	N	N	0
0	N	N	N	N	N	N	N	N	N	0
0	N	N	N	N	N	N	N	N	N	0
0	N	N	N	N	N	N	N	N	N	0
0	N	N	N	N	N	N	N	N	N	0
0	N	N	N	N	N	N	N	N	N	0
0	N	N	N	N	N	N	N	N	N	0

Przeprowadzona zostaje nowa iteracja i cykl się powtarza.

2.3. Algorytm scalania podtablic

Ostatnim krokiem algorytmu jest scalenie podtablic ze wszystkich node-ów z powrotem w całość. W tym celu nadzorca „woła" wszystkie node-y, aby te zwróciły mu swoje fragmenty tablicy. Zwracane są one w takiej postaci:



N	N	N	N	N	N	N	N	N
N	N	N	N	N	N	N	N	N
N	N	N	N	N	N	N	N	N
N	N	N	N	N	N	N	N	N
N	N	N	N	N	N	N	N	N

Nadzorca scala wszystkie fragmenty w całość i kończy działanie algorytmu.

3. Przebieg algorytmu

3.1. Warunki wstępne

Użytkownik ma możliwość:

- wygenerowania losowej planszy i zapisania do pliku
- wczytania planszy z pliku na podstawie jego nazwy
- wyświetlenia planszy z pliku (działa tylko dla małych tablic)
- rozpoczęcia symulacji (konkretna liczba iteracji) wraz ze zmierzeniem czasu iteracji

3.2. Działanie

Zachodzą odpowiednie operacje:

- sprawdzenie liczby dostępnych węzłów i na tej podstawie wybranie nadzorcy oraz procesów odpowiedzialnych za liczenie części tablicy
- nadzorca dzieli planszę podzielenie na mniejsze fragmenty i wysyła je do procesów liczących
- następuje obłożenie tablic zerami i wymiana odpowiednich wierszy pomiędzy nimi
- nadzorca czeka aż wszystkie procesy przyślą wiadomość z obliczonym fragmentem tablicy i wszystkie fragmenty tablicy zostaną obliczone
- jeżeli ma wykonać kolejną iterację to wraca do punktu 2 w przeciwnym razie idzie dalej
- zwrócenie przez nadzorcę informacji do użytkownika, że zostały wykonane iteracje

3.3. Warunki końcowe

Na koniec zostają wykonane następujące czynności:

- użytkownik zostaje poinformowany o czasie działania algorytmu
- zwracany jest czas działania oraz tablica

3.4. Test skalowalności

- Wpływ kolejnych węzłów na wynik

Używanie kolejnych node-ów poprawia wynik działania programu, jednakże do pewnego momentu. Przy zbyt dużej ilości węzłów szybkość algorytmu spada.

Spis Treści