Krzysztof Dąbrowski 293101

PRIR Specyfikacja wstępna - Gra w życie

19 maja 2020

Spis treści

1. Opis zadania

Projekt polega na zrealizowaniu równoległej symulacji *Gry w życie*. W tym celu zostanie rozbudowany projekt z przedmiotu JIMP2. W przeciwieństwie do większości zadań realizowanych na uczelni, w ramach projektu zostanie rozbudowany już istniejący kod. Zadanie to będzie wymagało innego podejścia, co moim zdaniem dobrze pasuje do aspektów programowania równoległego, które pozwala na przyśpieszenie już istniejących rozwiązań.

Projekt zakłada przygotowanie równoległej symulacji w językach Java oraz C. Dzięki temu możliwe będzie porównanie użytych bibliotek, architektur kodu oraz wydajności otrzymanych rozwiązań.

2. Opis ogólny problemu

Gra w życie jest automatem komórkowym wymyślonym przez brytyjskiego matematyka John Horton Conway w 1970 roku. Polega na symulacji kolejnych pokoleń życia komórek według następujących zasad.

2.1. Symulacja

Stany Komórka może znajdować się w jednym z dwóch stanów:

- żywa,
- martwa.

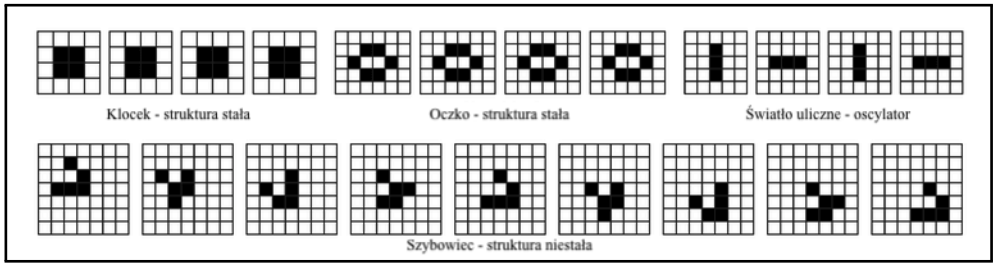
Pokolenie to stan wszystkich komórek w danej chwili. Gdy stan pokolenia jest ustalony, możliwe jest utworzenie nowego (potomnego) pokolenia komórek, powstających według poniższych zasad.

Reguły Następne pokolenie generowane jest zgodnie z regułami:

- Jeżeli komórka była martwa i miała dokładnie 3 żywych sąsiadów, w następnym pokoleniu staje się żywa,
- Jeżeli komórka była żywa to pozostaje żywa jeśli miała dwóch lub trzech żywych sąsiadów. W przeciwnym razie staje się martwa.

2.2. Struktury

Symulacja przeprowadzona zgodnie z powyższymi regułami może prowadzić do powstania ciekawych obiektów zwanych strukturami.



Rysunek 1. Przykłady struktur

Reguły symulacji umożliwiają również tworzenie dużo bardziej skomplikowanych struktur (jak na przykład maszyna Turinga – https://youtu.be/My8AsV7bA94).

3. Opis funkcjonalności

Ponieważ rozbudowywane są dwa różne programy to funkcjonalności będą zależne od programu.

3.1. Rozwiązanie w języku Java

Program wyświetla interface graficzny pozwalający na wykonywanie symulacji automatów komórkowych *Gra w życie* oraz *Wireworld*. Użytkownik może wygenerować losowy automat komórkowy lub stworzyć własny przy pomocy narzędzi pozwalających na modyfikację stanu wybranych komórek. Możliwa jest automatyczna symulacja lub ręczne przechodzenie między kolejnymi stanami. Dany stan automatu może zostać zapisany do pliku oraz później wczytany.

3.2. Rozwiązanie w języku C

Program działa w trybie wsadowym. Użytkownik przy pomocy argumentów wiersza poleceń może ustalić opisane poniżej parametry symulacji.

Argumenty

- `-h / --help`
  Wyświetlenie pomocy,
- `-f [nazwa pliku] / --file plik [nazwa pliku]`
  Łańcuch znaków będący nazwą pliku z wejściowym stanem planszy zgodny z formatem; wyklucza się z flagą -s
- `-o [ścieżka] / --output_dest [ścieżka]`
  Łańcuch znaków będący ścieżką do folderu, w którym zostaną zapisane wyniki symulacji. Domyślnie pliki będą generowane w folderze o nazwie będącej aktualną datą i godziną wywołania programu,
- `-t [gif | png | txt] / --type [gif | png | txt])`
  Łańcuch znaków reprezentujący typ generowanych rezultatów. Domyślnie gif,

- `-n [liczba] / --number_of_generations [liczba]`
  Liczba pokoleń do wygenerowania. Domyślnie 15,
- `-p [liczba] / --step [liczba]`
  Liczba decydująca o tym, co który stan symulacji będzie zapisywany. Domyślnie 1,
- `-s [LICZBAxLICZBA] / --size [LICZBAxLICZBA]`
  Łańcuch znaków o formacie "XxY" (X – szerokość planszy, Y – wysokość planszy), będący wymiarami losowo generowanej planszy początkowej. Wyklucza się z -f,
- `-d [liczba] / --delay [liczba]`
  Podanie tego argumentu spowoduje wyświetlanie w konsoli kolejnych pokolenie symulacji. *Liczba* ta będzie oznaczać czas w milisekundach między wyświetleniem poszczególnych pokoleń. Wartość -1, co oznacza manualne przechodzenie do następnego pokolenia klawiszem ENTER lub zapisanie wyświetlanego pokolenia w pliku.txt wpisując „save".

Wywołanie programu bez żadnego argumentu przyjmuje flagę –size 10, i wartości domyślne innych parametrów.

Wyniki symulacji są wyświetlana na konsolę lub zapisywane do plików o wybranym formacie.

4. Analiza możliwości zrównoleglenia programu

Liczenie stanu każdej komórki w danym pokoleniu zależy tylko od stanu komórek sąsiednich w poprzednim pokoleniu. Dzięki temu możliwe jest niezależne symulowanie każdej komórki nowego pokolenia. Nie da się jednak symulować przyszłych pokoleń nie znając stanu pokolenia poprzedniego.

Do symulacji tak zadanego problemu bardzo dobrze nadaje się zastosowanie wątków oraz pamięci współdzielonej.

Teoretyczne przyśpieszenie jakie można uzyskać poprzez zrównoleglenie programu jest równe liczbie zastosowanych wątków.

5. Wybór technologii

Do zrównoleglenia aplikacji w języku Java zastosowane będzie współczesne *Stream API* z wykorzystaniem *parallel stream*. Ponieważ na wcześniejsze zadania laboratoryjne wykorzystywałem klasy `Tread` to takie podejście pozwoli na zaznajomienie się z innym sposobem zrównoleglania obliczeń w tym języku.

Do projektu w C zastosowane będą *posix thread* lub ewentualnie *OpenMP*. Z uwagi na potencjalną trudność dodania *OpenMP* do już istniejącego projektu bardziej prawdopodobne jest zastosowanie wątków.

6. Opis sposobu zrównoleglenia

Ponieważ stan kolejnego pokolenia bazuje na pokoleniu poprzednim możliwe jest tylko zrównoleglenie symulacji jednego pokolenia w przód.

Zrównoleglenie symulacji danego pokolenia może być wykonane na dwa sposoby. Można wykonać zrównoleglenie wykonania poprzez podział całej planszy na obszary, które będą oddzielnie symulowane przez poszczególne wątki. Alternatywnie możliwe jest obliczanie przez wątki kolejnych pól po kolei przy pomocy schematu *round-robin*.

Zastosowanie przydziału *round-robin* pozwoli na zminimalizowanie oczekiwania na najwolniejszy wątek, może jednak powodować narzuty związane z pobieraniem kolejnych danych przez wątki. Optymalnym podejściem może być połączenie obu metod, polegające na cyklicznym przetwarzaniu bloków po kila komórek przez poszczególne wątki.