Game of Life

Autorzy:

- Filip Nowicki
- Maciej Sawicki
- Krystian Żabicki

Gra w życie została wykonana w języku C++. Ponadto Dzieli się na 4 moduły.

1. GameOfLife - główna logika gry, obsługa plików. Budowana jest jako biblioteka .lib.
2. ParallelColumnsLauncher - definiuje logikę Master i Slave oraz komunikację między nimi. Budowana jest jako plik wykonywalny .exe.
3. SerialLauncher - definiuje logikę wykonywaną na jednym wątku. Budowana jest jako plik wykonywalny .exe.
4. UnitTesting - testy jednostkowe.

Uruchomienie

ParallelColumnsLauncher

Program przyjmuje 5 argumentów:

1. boardSize: int - matrix size
2. numberOfIterations: int - number of iterations
3. boardInitPattern: string - allowed values = 'random', 'verticalStripes'
4. (optional) saveImages: bool - should save images after each iteration
5. (optional) debugEnabled: bool - should enable debug logging

Przykład:

Program, dla macierzy o wielkości 500 powinien przeprowadzić 1000 iteracji i użyć do tego 20 procesów. Początkowa macierz powinna być losowa. Zapisywanie obrazów powinno być włączone, a logi do debugowania wyłączone.

```
mpiexec -n 20 ParallelColumnsLauncher 500 1000 random true false
```

SerialLauncher

Program przyjmuje 5 argumentów:

1. boardSize: int - matrix size
2. numberOfIterations: int - number of iterations
3. boardInitPattern: string - allowed values = 'random', 'verticalStripes'
4. (optional) saveImages: bool - should save images after each iteration
5. (optional) debugEnabled: bool - should enable debug logging

Przykład:

Program, dla macierzy o wielkości 728 powinien przeprowadzić 40 iteracji. Początkowa macierz powinna składać się z pionowych pasków o naprzemiennych wartościach. Zapisywanie obrazów powinno być włączone, a logi do debugowania wyłączone.

```
SerialLauncher 728 40 verticalStripes true false
```

Wyniki

Całkowity czas wykonania wyświetlony zostanie tuż przed zakończeniem działania programu. Obrazki wynikowe zostaną zapisane w folderze *./screens*, który zostanie utworzony w katalogu plików wykonywalnych programu. Po uruchomieniu programu wynik każdej iteracji, w postaci obrazka, zostanie zapisany w katalogu o nazwie, która jest datą jego uruchomienia. Nazwa obrazka ma format *iteration_xxxxxx.pgm*, gdzie *xxxxxx* to numer iteracji. Ponadto utworzony zostanie jeszcze jeden plik o nazwie *begin.pgm*, który reprezentować będzie początkowy stan macierzy.

Tworzenie filmiku

Aby utworzyć filmik ze wszystkich zdjęć należy użyć następującej komendy:

```
ffmpeg -r 10 -pattern_type glob -i "*.pgm" a_video.mp4
```

Komunikacja wieloprocesorowa

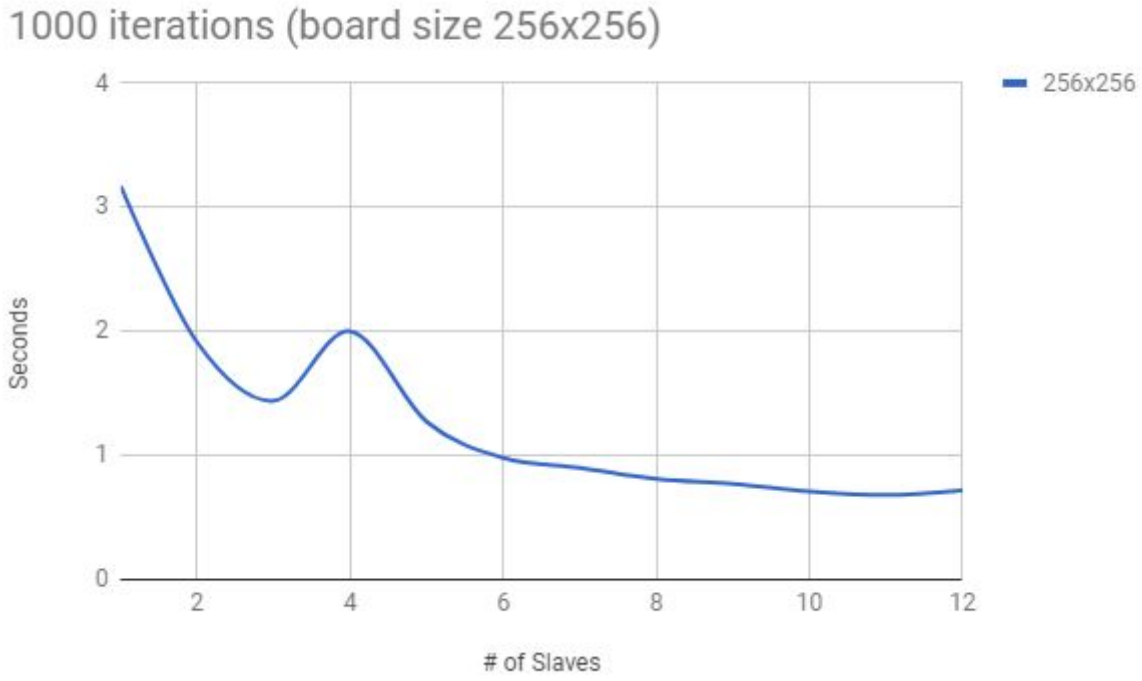
Program dzieli się na 2 typy: Master oraz Slave. Master po rozpoczęciu programu tworzy początkową macierz i rozsyła jej kolumny do Slave-ów. W wysyłanych danych zawarta jest nie tylko kolumna, ale też jej index w macierzy, identyfikatory lewego oraz prawego sąsiada. W tym samym czasie wszystkie Slave-y nasłuchują na komunikaty od Mastera na tym samym kanale. Po rozdaniu wszystkich kolumn Master wysyła do wszystkich Slave-ów, ciągle na tym samym kanale, pusty komunikat. Oznacza on, że Slave-y przestają nasłuchiwać na kolejne kolumny i rozpoczynają właściwe obliczenia.

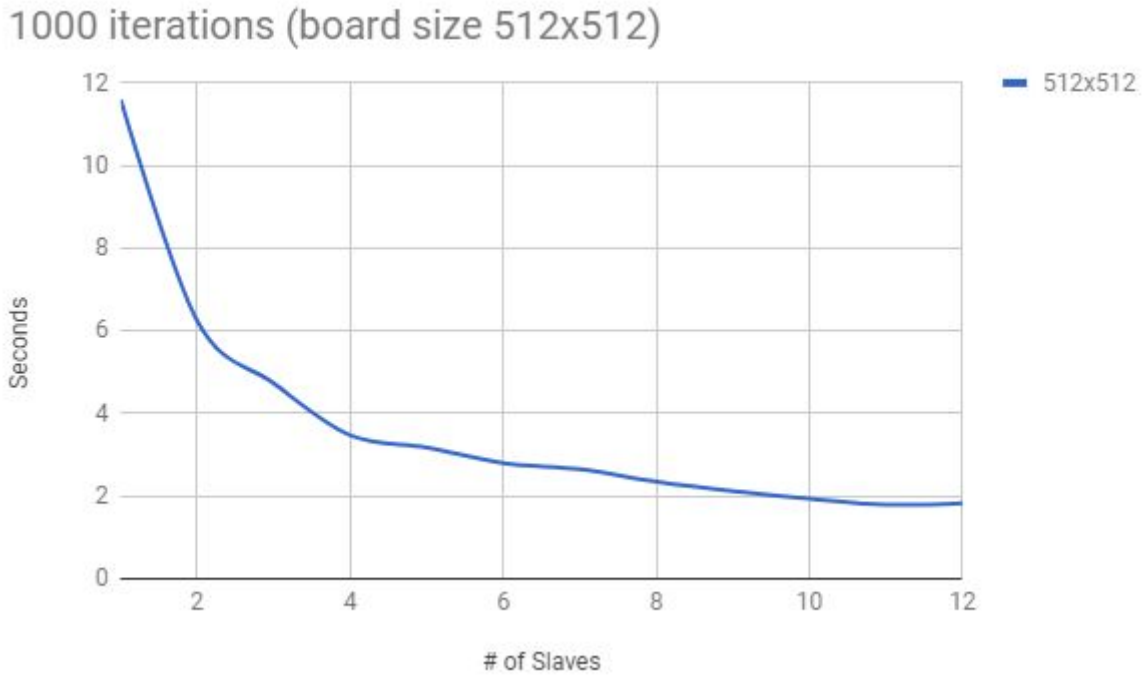
Następnie jeśli włączone jest zapisywanie obrazów, to Master zaczyna nasłuchiwać na przychodzące kolumny z kolejnych iteracji, a następnie je zapisuje do formatu *pgm*.

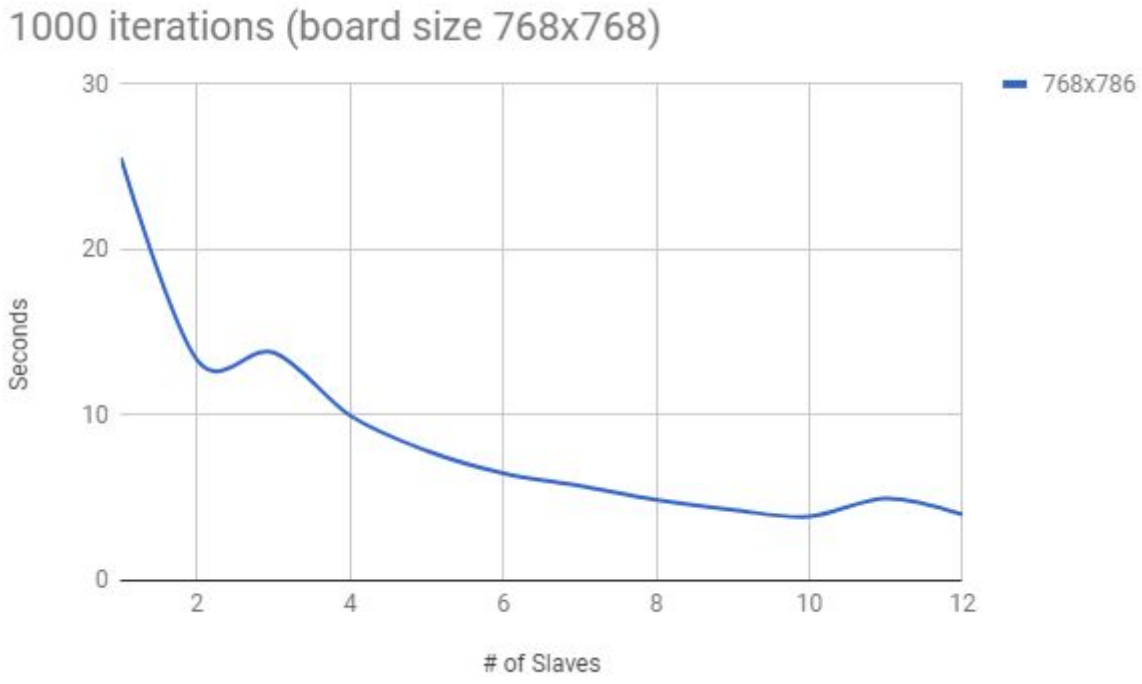
Slave-y w tym samym momencie sprawdzają, który z sąsiadów będzie potrzebował ich kolumn i jeśli wystąpi taka potrzeba, to wyśle je **asynchronicznie**. Proces ten dotyczy wszystkich kolumn w pojedynczej iteracji naraz. Następnie Slave sprawdza, czy musi pobrać kolumnę od swojego sąsiada i jeśli taka potrzeba nastąpi, to pobiera kolumnę **synchronicznie**. Następnie zebrawszy wszystkie wymagane kolumny (lewa i prawa) oblicza wartość swojej kolumny dla nowej iteracji i w zależności, czy zapisywanie obrazków jest włączone wysyła je do Mastera. Po tym czyści zbędne zasoby i proces powtarza się od początku aż do momentu ukończenia ostatniej iteracji.

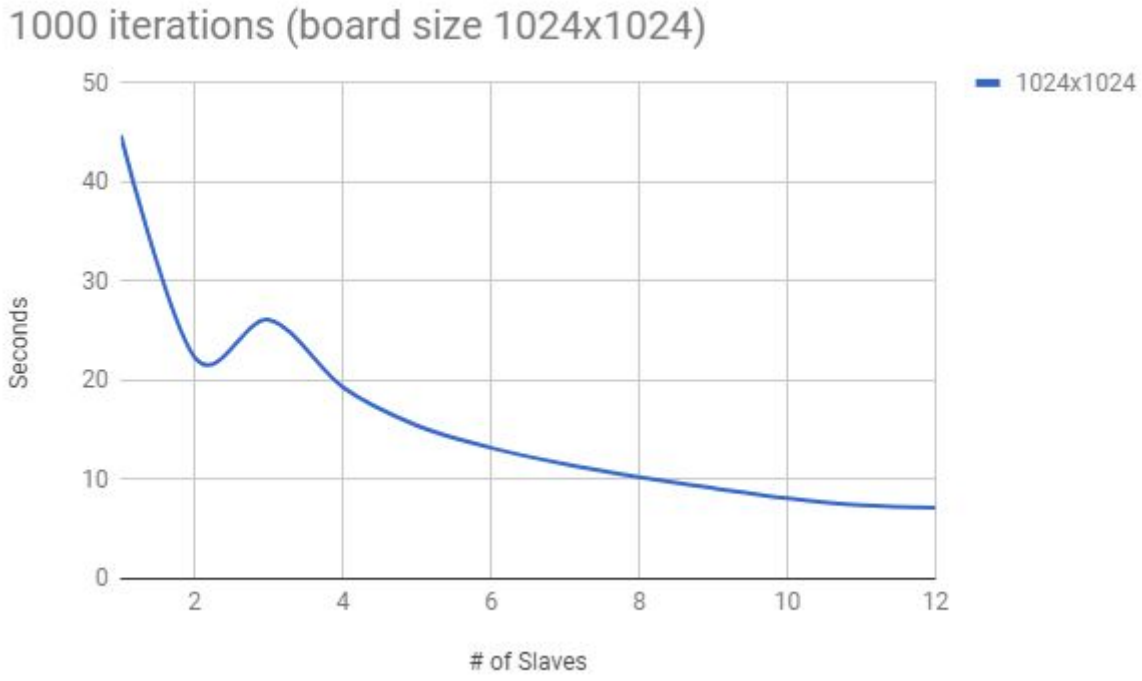
Benchmarki:

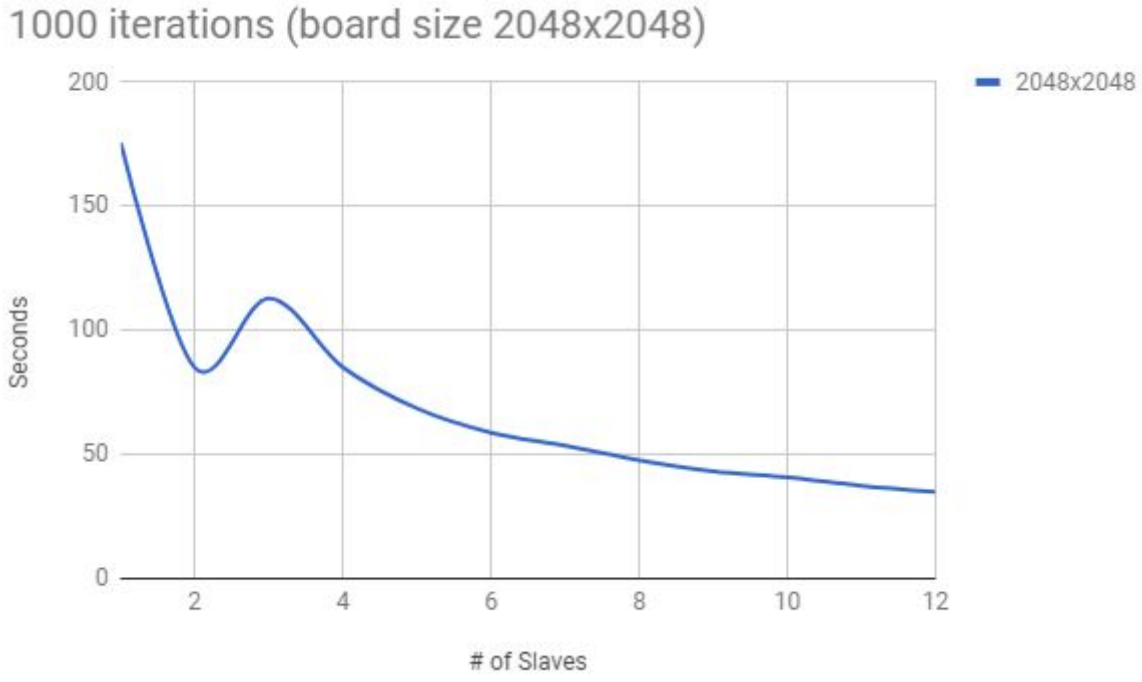
Testy wykonano na 6 rdzeniowym, 12 wątkowym procesorze Intel Core i7-8700k. Przetestowano czasy wykonania, wyrażone w sekundach, różnych rozmiarów plansz (256, 512, 768, 1024 i 2048) dla 1000 iteracji. Wyniki przedstawiono na poniższych wykresach:




1000 iterations (board size 512x512)
512x512
Seconds
# of Slaves
0
2
4
6
8
10
12


1000 iterations (board size 768x768)
768x786
Seconds
# of Slaves
0
10
20
30
2
4
6
8
10
12





Wnioski

Z przeprowadzonych badań wynika, że używanie większej liczby wątków do obliczeń obniża czas wykonywania się programu. Całkowity czas ‘gry o życie’ maleje wykładniczo osiągając 6-krotnie krótszy czas wykorzystując 12 wątków. Należy jednak dobrać odpowiednią liczbę wątków do określonego zadania, gdyż nadmierna

ich ilość może przekładać się na większe koszta przy zastosowaniach komercyjnych przy nieznacznym lub nawet braku skróceniu czasu wykonywania. Dla większości badanych przypadków stwierdzono anomalię zwiększenia czasu wykonania dla 3-4 Slave-ów. Przyczyny takiego rozwoju wydarzeń pozostają jednak zagadką.