Rozproszona gra w życie w Erlangu

Jan Bieroń
Kamil Kopczyk
Michał Pierzchała
Robert Przystasz
Przemysław Szczepaniak

1. **Przyjęte rozwiązanie**

Nasz projekt oparliśmy o wzorzec OTP i behaviour gen_server. Logika programu rozbita jest na następujące moduły:

- starter — moduł ułatwiający pracę z programem
- master_bula — program „glowny" żądający obliczeń na fragmentach tablicy od slave'ów
- slave_bula — servery obliczeniowe spawnowane przez mastera
- lifeio — wiadomy modul rozbudowany o konieczne funkcje konwertujące plik na krotkę krotek i vice versa
- benchmark — moduł testujący czas działania

Master spawnuje N slave'ów i wysyła każdemu fragment tablicy możliwie równo podzielonej w pasy (interpretowane u nas poziomo), wraz z informacją o adresie slave'a posiadającego pas wyżej i niżej. Slave'y wymieniają się między sobą (bez udziału mastera) informacjami o stycznych wierszach za każdym razem gdy liczą kolejną iterację. Dopiero po ukończeniu wszystkich zadanych iteracji odsyłają swoje fragmenty do mastera, który blokuje się synchronicznym callem w oczekiwaniu na nie. Wtedy je skleja zgodnie z zapamiętanym przez siebie słownikiem slave → fragment.

2. **Użycie programu**

```
erl -sname ...
> l(starter).
> starter:start(N).                %gdzie dim(plansza) =:= 2**N , tworzy losowo plansze
> starter:read_and_start(Filename). %zamiast powyższego!
> starter:next().                  %zwroc kolejna iteracje planszy
> starter:next(L).                 %zwroc L-tą iteracje planszy
> benchmark:test_time([X],Y). % gdzie      X: ilosc iteracji do obliczenia przez slave'y
                                           Y: ilosc liczonych prob przez benchmark
> benchmark:test_time([X]) → test_time([X],1).
> benchmark:test_time() → test_time([10],10).
> master_bula:say("~p~n", ["to jest życie!"]).
> master_bula:save().              %zapisuje bieżącą, wyliczoną tablicę do fff.gz w working-dir
```

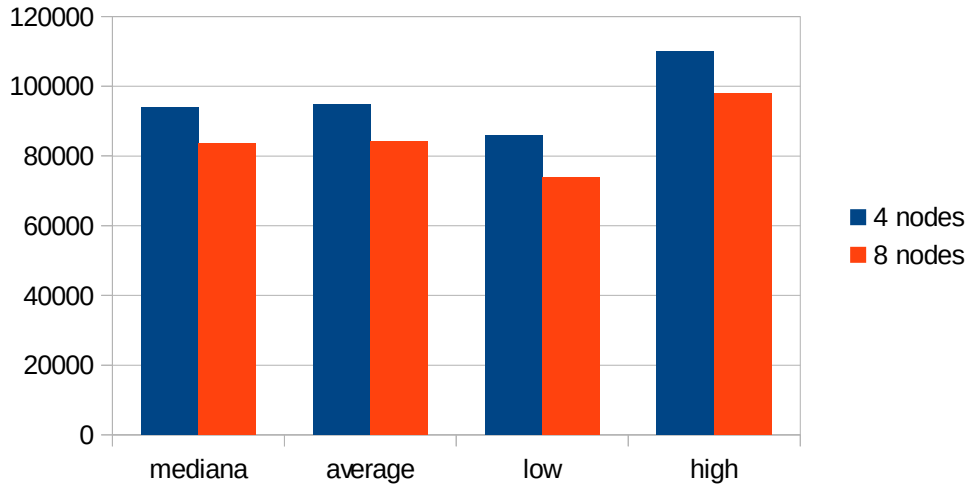
3. **Napotkane problemy**

Nie byliśmy w stanie korzystać z dwóch ostatnich nodów (l@le9,l@le10). Chociaż są zwracane przez discover_nodes, nie odpowiadają na call mastera, co skutkuje timeoutem (a infinity mija się z celem). Dlatego ograniczyliśmy liczbę możliwych nodów do 8.

3.Opis testu skalowalności (czy użycie kolejnych węzłów poprawia wynik)
> benchmark:test_time([1],100).

%Rozmiar 512 x 512, 4 nodów
Range: 85910 - 110123 mics
Median: 93919 mics
Average: 94864 mics

%Rozmiar 512 x 512, 8 nodów
Range: 74015 - 98038 mics
Median: 83645 mics
Average: 84220 mics



Dwukrotne zwiększenie ilości węzłów skutkuje niewiele ponad 10% efektywnością działania. Niewykluczone, że dla większych danych zysk jest większy, gdyż stosunek narzutu działań stałych do faktycznych obliczeń maleje. Pozostaje pytanie , czy przesyłanie fragmentów tablicy do innych węzłów jest mniej czasochłonne od obliczania z mniejszą dostępną ilością procesorów.