Raport

Maciej Gług
Mateusz Jabłoński
Szymon Konicki
Adam Musiał

Opis rozwiązania

W celu osiągnięcia wysokiej wydajności programu została użyta struktura danych bitboard (czyli najprościej mówiąc tablica bitów). Struktura ta jest bardo efektywna przy przechowywaniu informacji które swoim wyglądem naturalnie odpowiadają planszy - jak np. tablica gry w życie. Bitboardy są często wykorzystywane w grach takich jak szachy i warcaby. Zamiast przetwarzać po jednej komórce program przetwarza równocześnie 64 komórki zapisane na 64 bitach. Mechanizm działa w następujący sposób:

1. tablica danych jest wczytywana z pliku za pomocą funkcji readDataToColumns(FileName, ColumnsCount) w module lifeio
2. dane są wczytywane do zadanej liczby kolumn
3. dane są zamieniane na tablicę bitów (bitboard)
4. tablica bitów jest obudowywana zerami zarówno z góry jak i dołu
5. pierwsza i ostatnia kolumna są obudowywane zerami z lewej lub odpowiednio z prawej strony, a pozostałe wymieniają się krawędziami
6. wyliczany jest następny stan gry w następujący sposób
    a. dla każdego bitu równego 1 sprawdzani są wszyscy sąsiedzi operacją logiczną AND (w zależności od wyniku komórka umiera lub pozostaje żywa)
    b. dla każdego bitu równego 0 sprawdzani są wszyscy sąsiedzi i w zależności od wyniku jesli komórka ma dokładne trzech sąsiadów staje się żywa
7. odpowiednia pętla powtarza iterację po rozbudowaniu i obudowaniu tablicy

- opisanie mechanizmu podziału danych na węzły i procesy

Główną funkcją programu jest test_timer/2 pobierająca jako argument liczbę iteracji oraz wartość logiczną mówiącą o chęci zapisania końcowego wyniku do pliku. W pierwszej kolejności określany jest rozmiar tablicy startowej. Następnie na jego podstawie wyznaczana jest najlepsza mozliwa konfiguracja tzn. liczba dodatkowych węzłów oraz liczba kolumn na które zostaje podzielona tablica startowa (funkcja: lifeconc:getBestConfiguration/1). Następnie wczytywany jest plik ze startową tablicą który od razu dzielony jest na wyznaczoną wcześniej liczbę kolumn.

1. Liczba węzłów dodatkowych równa zero

W przypadku gdy wyznaczona liczba dodatkowych węzłów równa jest zero wywołana zostaje funkcja lifemain:iterateSingleMachine/9

Działanie funkcji lifemain:iterateSingleMachine/9

Funkcja ta odpowiada za wykonanie wszystkich iteracji. Zwraca całkowity czas i końcową listę kolumn (w odpowiedniej kolejności). Dla każdej iteracji wywoływana jest funkcja iterateLocal/8 przy pomocy list:foldl (co zapewnia, że lista kolumn po każdej iteracji jest aktualizowana).

Działanie funkcji iterateLocal/8

Funkcja ta w pierszej kolejnosci wywołuje funkcję prepareColumnTuples/6 która z listy kolumn tworzy listę krotek, zwanych dalej krotkami kolumny.
Krotka kolumny zawiera: prawą krawędz poprzedniej kolumny (lub zero, jesli to pierwsza kolumna), daną kolumne, lewa krawędź nastepnej kolumny (lub zero, jesli to ostatnia kolumna).
Aby stworzyć te krotki w pierwszej kolejności tworzona jest lista krotek zawierajacych 3 kolumny: poprzednią, bierzącą i następną (borderTuplesToColumnTriples/1). Na podstawie takich krotek tworzone są krotki kolumny (columnTripleToTuple/3)
Dla każdego elementu listy krotek kolumny wywoływana jest funkcja calculateSingleColumn/3 przy pomocy funkcji rpc:pmap, zapewniającej równoległość obliczeń. Funkcja calculateSingleColumn/3 na podstawie krotki kolumny tworzy kolumnę rozszerzoną o krawedzie poprzedniej i nastepnej kolumny, następnie przelicza stany wszystkich komórek kolumny. Na końcu kolumny prawa i lewa są obcinane. Po wykonaniu funkcji calculateSingleColumn/3 dla wszystkich krotek nowo powstałe krotki są zapisywane w odpowiednij kolejności w liście krotek kolumny.
Jest to koniec działania metody iterateLocal/8 (przeliczenia jednej iteracji).

2. Liczba węzłów dodatkowych różna od zera

W przypadku gdy wyznaczona liczba dodatkowych węzłów różna jest od zera wywołana zostaje funkcja lifeconc:mainController/7. Jest ona odpoweidzialna za uruchamianie, synchronizację i konczenie procesów na węzłach. Na początku wywoływana jest funkcja initializeNodesSupervisors/6.
Funkcja initializeNodesSupervisors/6.
Jako argument dostaje m.in. listę dostępnych węzłów. Obliczana jest jej długość a następnie wyznaczana jest liczba kolumn która ma przypadać na jeden węzeł.
Dla każdgo węzła (przy pomocy lists:map) przydzielane sa odpowiednie kolumny (po kolei), tworzony jest proces wywołujący funkcję nodeSupervise/7 i tworzona jest krotka zawierajacych lewą krawedź wezla, dane wezla, prawa krawedz wezla (dane węzła to krotka zawierająca numer węzła, nazwa węzła i pid procesu).
Funkcja zwraca listę tych krotek.
Po zakończeniju działania funkcji initializeNodesSupervisors/7 rozpoczynany jest pomiar czasu.
Następnie wynik wywołania funkcji initializeNodesSupervisors/7 przekazywany jest do funkcji nodeNext/3.
Funkcja nodeNext/3
Funkcja wymienia krawedzie pomiedzy wezlami i wywoluje nastepna iteracje.
Funkcja do każdego procesu (stworzonego w initializeNodesSupervisors/6) przesyła kolumne jaka jest na lewo i na prawo od kolumn przetwarzanych przez proces węźle.

Dla każdego procesu, w funkcji nodeSupervise/7 odbierana jest ta informacja i analogicznie jak w przypadku jednego węzła wykonywana jest iteracja. Po wykonaniu iteracji na danym węźle wysyłana jest wiadomość do procesu wywołującego funkcje nodeListener a funkcja nodeSupervise wywoływana jest ponownie(z nowymi argumentami). Funkcja nodeListener po otrzymaniu wiadomosci od wszystkich węzłów sortuje kolumny (na podstawie numeru weżłą) a następnie wywołuje funkcję nodeNext z wskaźnikiem iteracji o 1 mniejszym. Gdy wszystkie iteracje się wykonają, wtedy funkcja nextNode wywoływana jest z argumentem 0 i kończy swoje działanie.
Po skończeniu działania powyższej funkcji następuje koniec pomiaru czasu.
Następnie zostaje wywołana dodatkowa funkcja callFinishOnNodes/1 konczęca procesy na wszystkich węzłach oraz funkcja getFinalColumns/1 sklejająca kolumny z węzłów w jedną tablicę.