Raport z prac nad projektem.

Skład grupy:

Paweł Brańka

Maciej Bruno-Kamiński

Krzysztof Grzyb

Mateusz Juraszek

Mirosław Małek

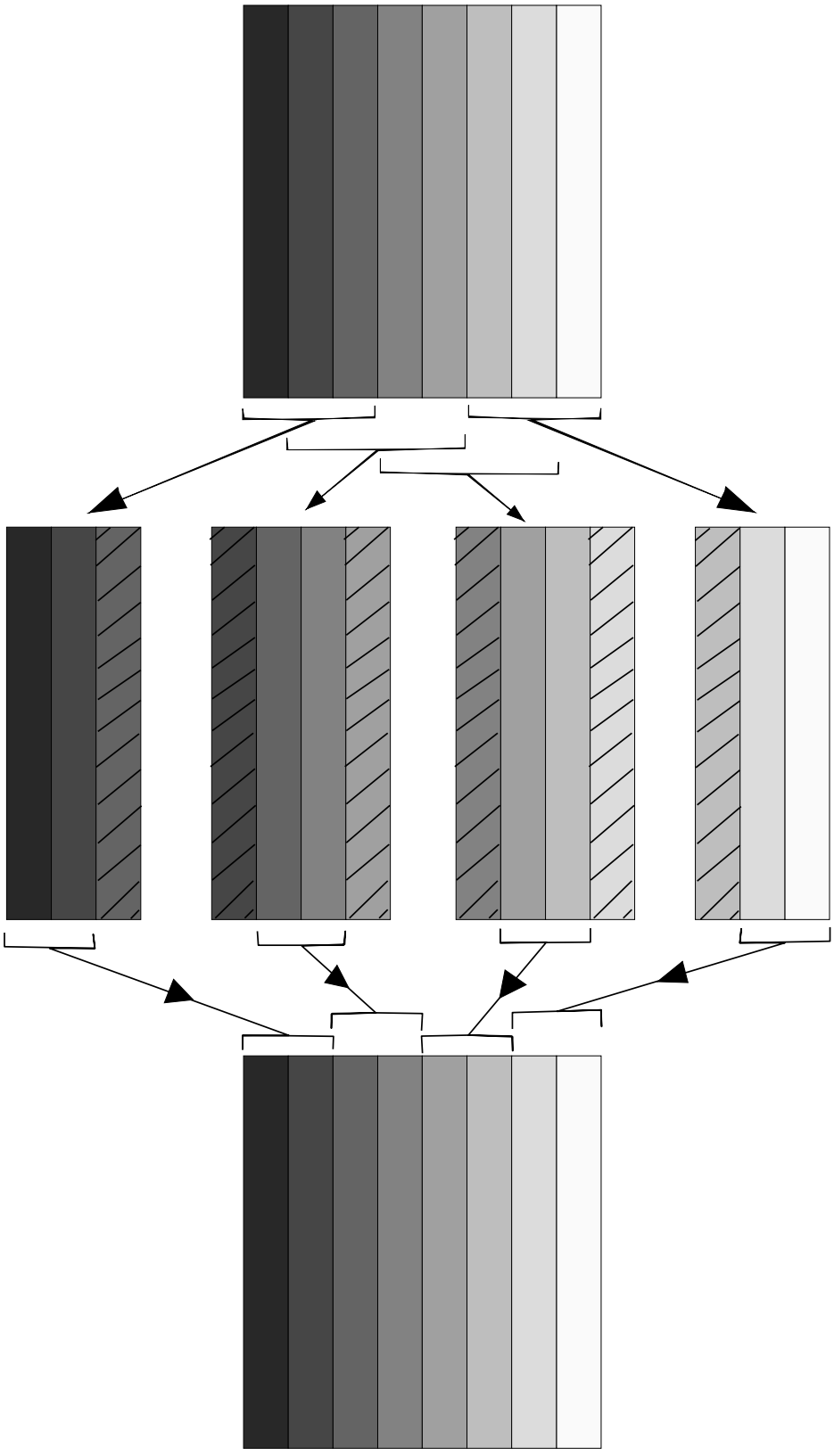
Opis przyjętego rozwiązania:

Moduły:

- tab2d – implementacja tablic dwuwymiarowych (tablice tablic) na podstawie Erlangowego modułu array. Zawiera podstawowe funkcje do pracy z tablicami (create, set,get,size) oraz funkcje umożliwiające na generowanie losowych tablic, ich dziele i łączenie.
- file_io – operacje zapisu i odczytu tablic ze skompresowanych plików
- worker – moduł węzła obliczeniowego, zapewnia wymianę informacji oraz silnik gry life
- life – główny moduł programu zawierający wszystkie pozostałe funkcje zawarte w opisie wymagań.

Podstawą naszego programu jest dwuwymiarowa tablica komórek, która w każdym kroku iteracji jest dzielona, w zależności od ilości dostępnych nodów, na 2,4 lub 8 mniejszych tablic, i rozsyłana do węzłów. Każda z tak uzyskanych podtablic oprócz kolumn właściwych zawiera dodatkowe kolumny – sąsiednie do zewnętrznych. Nie są one aktualizowane ale zawarta w nich informacja jest niezbędna do prawidłowego działania programu. Po otrzymaniu tablicy, każdy węzeł przystępuje do obliczania swojej części i zwraca wynik wraz ze swoim identyfikatorem. Po otrzymaniu wyników od wszystkich nodów są one scalane z powrotem w jedną tablice, dzięki otrzymanym identyfikatorom możliwe jest określenie miejsca pojedynczego wyniku w tablicy końcowej. Dodatkowe kolumny opisane powyżej zostają odrzucone.

Przykładowy schemat podziału dla 4 węzłów zobrazowany jest poniżej.



Wyniki i skalowalność

Mimo iż sama iteracja po tablicy i uaktualnianie stanów komórek odbywa się dość szybko algorytm jako całość nie prezentuje najlepszych czasów działania. Przyjęta metoda – dzielenie i scalanie tablic jest proste do zrozumienia i intuicyjne lecz niestety okazało się nieoptymalne. Wykonanie tych dwóch funkcji zajmuje najwięcej czasu. Nie bez znaczenia pozostaje też kwestia ilości danych przesyłanych pomiędzy węzłami, która w tym przypadku jest dość duża. Wciąż można jednak zaobserwować korzyści płynące ze zrównoleglenia obliczeń i wykorzystania większej ilości węzłów. Przy niewielkich tablicach (256x256) różnice pomiędzy zastosowaniem 2,4 i 8 węzłów są pomijalne (rzędu dziesiątych/setnych sekundy) jednak stają się coraz bardziej zauważalne wraz ze wzrostem tablic (parenaście sekund dla 4096x4096 i parędziesiąt dla większych). Poniższa tabela obrazuje czasy działania dla niektórych rozmiarów tablic, niestety z powodów kłopotów z dostępnością węzłów nie udało nam się przeprowadzić obliczeń dla największych rozmiarów. Przy rozmiarze 13 (8192x8192) należy spodziewać się jednak wyników oscylujących w granicach 500-550 sekund dla 4 i 8 węzłów, oraz nawet 600 sekund dla dwóch węzłów

Rozmiar tablicy	Czas dla 2 węzłów	Czas dla 4 węzłów	Czas dla 8 węzłów
2^8	0,450 s	0,342 s	0,321 s
2^9	1,877 s	1,401 s	1,204 s
2^{10}	8,653 s	6,630 s	6,311 s
2^{11}	39,218 s	28,038 s	26,740 s
2^{12}	158,494 s	122,262 s	105,378 s

Ważniejsze funkcje

tab2d		
Funkcja	Parametry	Opis
randomArr(S)	Wielkość tablicy (2^S)	Funkcja generuje tablice wypełnioną losowymi wartościami o podanym rozmiarze.
patternArr(S)	Wielkość tablicy (2^S)	Generuje tablicę wypełnioną charakterystycznymi dla gry Life strukturami komórek (glider, lwss, block, boat, blinker, frog).
split2(Arr)	Tablica do podziału	Dzieli podaną jako parametr tablicą na dwie podtablice zwracane jako krotka.
split4(Arr)	Tablica do podziału	Dzieli podaną jako parametr tablicę na cztery podtablice zwracane jako krotka.
split8(Arr)	Tablica do podziału	Dzieli podaną jako parametr tablicę na osiem podtablic zwracanych jako krotka.
join2(A1,A2)	Tablice do połączenia	Scala dwie podane tablice w jedną i zwraca ją.
join4(A1,A2,A3,A4)	Tablice do połączenia	Scala cztery podane tablice w jedną i zwraca ją.
join8(A1,A2,A3,A4,A5, A6,A7,A8)	Tablice do połączenia	Scala osiem podanych tablic w jedną i zwraca ją.
worker		
main(NR)	Numer węzła obliczającego	Główna funkcja węzła, pobierająca swoją część tablicy, dokonująca obliczeń i zwracająca ja do nadawcy.
iterate(Arr)	Tablica	Dokonuje iteracji po podanej tablicy zmieniająca wartości komórek zgodnie z zasadami gry Life.
file_io		
readFile(FileName)	Nazwa pliku	Wczytuje zawartość skompresowanego pliku do tablicy i zwraca ją.
write2file(FileName,Arr, S)	Nazwa pliku, Tablica, Rozmiar (2^S)	Zapisuje do pliku o podanej nazwie tablicę o rozmiarze 2^S.
life		
run()	-	Kompiluje pozostałe moduły, zwraca listę dostępnych nodów.
next({T,N})	Krotka: {Tablica, Lista nodów}	Oblicza jedną iterację programu, zwraca wyliczoną tablice. W zależności od liczby nodów wykonuje rozproszenie na 2,4 lub 8 nodów.

next({T,N},IT)	Krotka: {Tablica, Lista nodów}, Ilość iteracji	Wykonuje określoną liczbę funkcji next({T,N}).
test_time(P,N)	P = Krotka: {Tablica, Lista nodów}, N = Ilość iteracji	Funkcja testująca czas działania programu.

Przykładowe uruchomienia programu

Kompilacja, ładowanie modułów, tworzenie listy nodów

c(life). -kompilacja głównego modułu

L = life.run(). -kompilacja pozostałych modułów, zwrócenie listy aktualnych węzłów

nl(worker). -wysłanie modułów na węzły

nl(tab2d).

Z zapisem do pliku

T = tab2d:randomArr(10). -wygenerowanie losowej tablicy o rozmiarze 2^{10}

W = next({T,L},5). -wykonanie pięciu iteracji

write2file('plik.gz',W,10). -zapisanie wyniku do pliku

Z wczytaniem z pliku

T = readFile('plik.gz'). -odczyt tablicy z pliku

W = next({T,L},5). -wykonanie pięciu iteracji

Testowanie czasu działania

T = tab2d:randomArr(10). -wygenerowanie losowej tablicy o rozmiarze 2^{10}

test_time({T,L},5). -funkcja testująca czas pracy