Analiza skalowalnośći kodu metodą Lattice Boltzman, wpływ kernela oraz kraty na efektywność obliczeń

Rybski Arkadiusz

2020

Spis treści

1 Wprowadzenie

1.1 Metoda Lattice Boltzmann

Metoda Lattice Boltzmann jest metodą numeryczną służacą do rozwiązywania równań z zakresu mechaniki płynów. Metoda kratowa Boltzmanna oparta jest na równaniu Boltmanna(Rownanie 1).

$$\frac{\partial f}{\partial t} + \xi_\beta * \frac{\partial f}{\partial x_\beta} + \frac{F_\beta}{\rho} * \frac{\partial f}{\partial \xi_\beta} = \Omega(f) \tag{1}$$

gdzie
$f(x, \xi, t)$ oznacza funkcję rozkładu
x oznacza położenie
ξ oznacza prędkości cząstek
t oznacza czas

Można pokazać, że rozwiązania mezoskopwych równań Boltzmanna zbiega się do równań Naviera Stokesa. Niestety nie rozwiązuje to problemu analitycznego rozwiązania równania. Natomiast okazuję się, iż mimo skomplikowanej formy, równanie Boltzmana w prosty sposób można zaimplementować. W ten sposób możemy otrzymać równanie kratowe Boltzmanna(Rownanie 2).

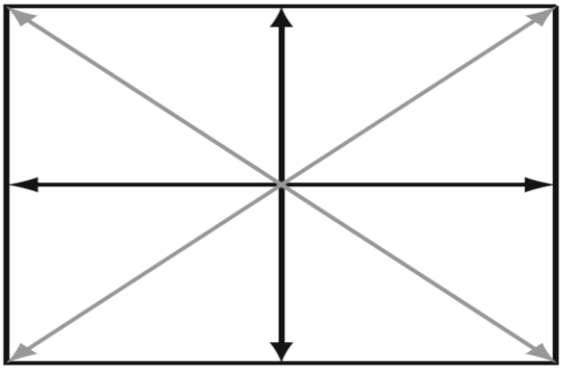
$$f_i(x + c_i * \Delta t, t + \Delta t) = f_i(x, t) + \Omega(x, t) \tag{2}$$

1.1.1 Rodzaje krat

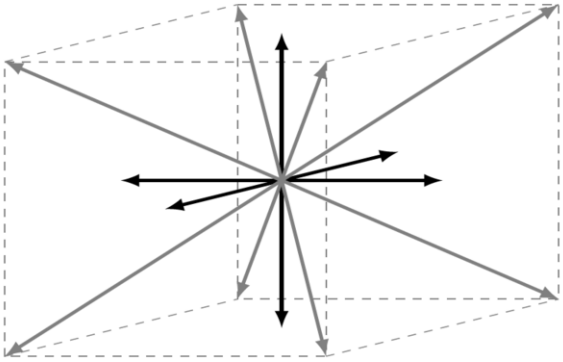
Ze względu na to, że funkcja dystrybucji jest zależna nie tylko od czasu, położenia ale też i prędkości do implementacji potrzebujemy siatki zawierającej nie tylko położenie geometryczne. Wymagana jest dyskretyzacja czasu, przestrzeni ale także i prędkości. W literaturze przyjęto następujące oznaczenia krat.

$$D_n Q_m$$

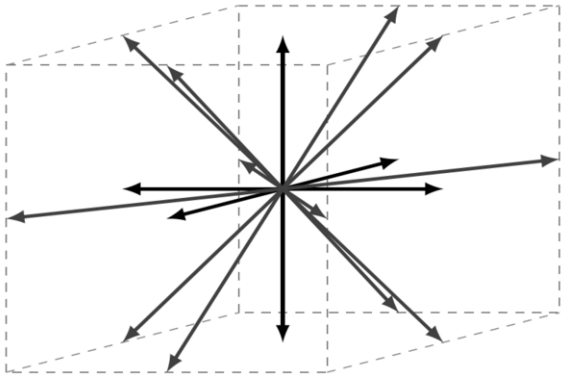
gdzie n oznacza ilość wymiarów, natmoiast m oznacza ilość możliwych kierunków prędkości.
Przykładowe kraty zostały zamieszczone na poniższych obrazkach.



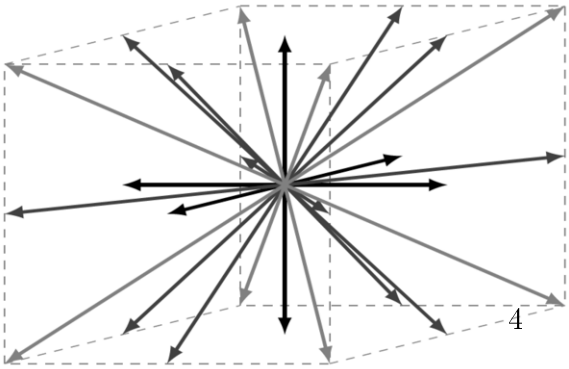
(a) D2Q9



(b) D3Q15



(c) D3Q19



(d) D3Q27

Rysunek 1: Przykładowe kraty

1.1.2 Algorytm

Zasada działania metody kratowej Boltzmanna oparta jest na dwóch fazach: propagacji i kolizji. Poniższe równania będą przedstawione dla operatora kolizji BGK.

Faza kolizji

Równanie kolizji

$$f_i^*(x + c_i * \Delta t, t + \Delta t) = f_i(x,t) - \frac{\Delta t}{\tau} * (f_i(x,t) - f_i^{eq}(x,t)) \tag{3}$$

Faza propagacji

Równanie propagacji

$$f_i(x + c_i * \Delta t, t + \Delta t) = f_i^*(x,t) \tag{4}$$

Całość mechanizmu można podsumować w nastepujących krokach.

1. Wybór lokalizacji
2. Rejestracja informacji o nadchodzących cząsteczkach
3. Kolizja
4. Dystrybucja po kolizji
5. Wybór kolejnej lokalizacji

1.1.3 Rodzaje kernela

Przedstawiony w powyższym alogrytmie operator kolizji BGK(Bhatnagar-Gross-Krook) to jeden z wielu możliwych operatorów kolizji. Inne z nich to *MRT(multiple relaxation time)* czy *Cumulant Colission Operator*.
Operator kolizji musi spełniać równania zachowania masy(Rownanie 5), momentów (Rownanie 6), energii (Rownanie 7), a także zasadę zachowania entropii.

$$\int \Omega(f) d^3\xi = 0 \tag{5}$$

$$\int \Omega(f) * \xi d^3\xi = 0 \tag{6}$$

$$\int \Omega(f) * \xi^2 d^2\xi = 0 \tag{7}$$

Operator kolizji może być realizowany na wiele sposobów, dwa z nich zamieszczam poniżej.

BGK operator

$$\Omega(t) = -\frac{\Delta t}{\tau} * (f_i(x,t) - f_i^{eq}(x,t)) \tag{8}$$

Multiple relaxation time colission operator

$$-M^{-1} * S * M * [f(x,t) - f^{eq}(x,t)] * \Delta t \tag{9}$$

1.2 Skalowalność kodu

Dzięki nieustannnemu rozwojowi technnicznemu współcześnie jesteśmy w stanie rozwiązywać większe problemy obliczeniowe za pomocą wielu proceserów. Co zdecydowanie skraca czas wykonywania obliczeń. Skalowalność określa nam efektywność kodu w przypadku użycia zwiększonych zasobów komputeroywch. W celu zbadania efektywności obliczeń równoległych wprowadźmy wielkość zwana dalej *przyspieszeniem*(z ang. *speedup*).

$$speedup = \frac{t_1}{t_N}$$

gdzie
t_1 oznacza czas wykonanaia procesu przy użyciu 1-ego procesora
t_N oznacza czas wykonania procesu przy użyciu N procesorów.

W idealnym przypadku wykres $speedup(N)$ byłby wykresem liniowym.

Drugą wielkością, która wporwadzimy będzie tzw. *skalowane przyspieszenie*(z ang. *scaled speedup*).

$$scaled\ speedup = \frac{t_1}{\frac{t_N}{N}}$$

gdzie
t_1 oznacza czas wykonywania procesu przy użyciu 1 procesora

t_N oznacza czas wykonywania procesu przy użyciu N procesorów, pod warunkiem stałej wielkości $\frac{work}{processor}$

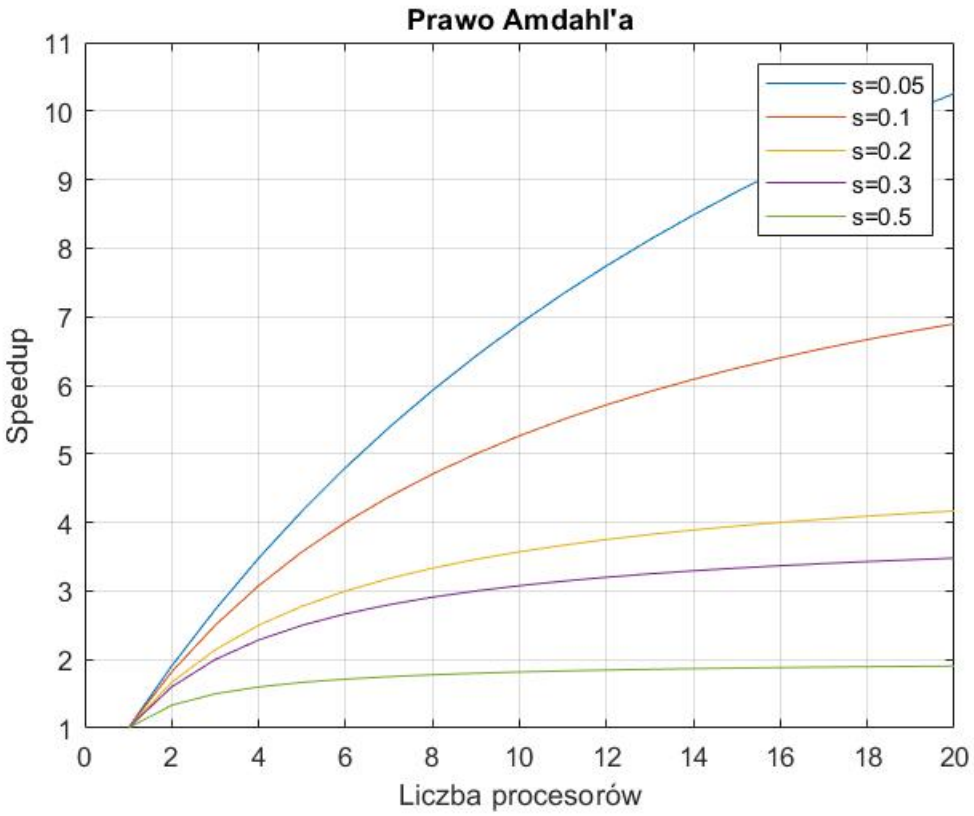
Przyspieszenie jest limitowane przez cześć kodu obliczeniowego, którą nie jesteśmy w stanie zrównoleglić. Istnieją dwa główne podejścia w zakresie badania skalowaności programów: silne i słabe skalowanie.

1.2.1 Silne skalowanie

Idea silnego skalowania jest prosta: przy zachowaniu stałego rozmiaru programu zwiększamy ilość procesorów pracujących nad jego rozwiązaniem. *Przyspieszenie* jest limitowane prawem Amdahl'a:

$$speedup \leq \frac{1}{s + \frac{p}{N}} < \frac{1}{s}$$

gdzie
s oznacza cześć kodu, która nie może zostać zrównoleglona
p część kodu zrównoleglona
N ilość procesorów.
Uwaga $s + p = 1$
Przykładowo jeśli 10% kodu obliczeniowego nie nadaje się do zrównoleglenia to maksymalnie możemy uzyskać 10-krotne przyspieszenie procesu.



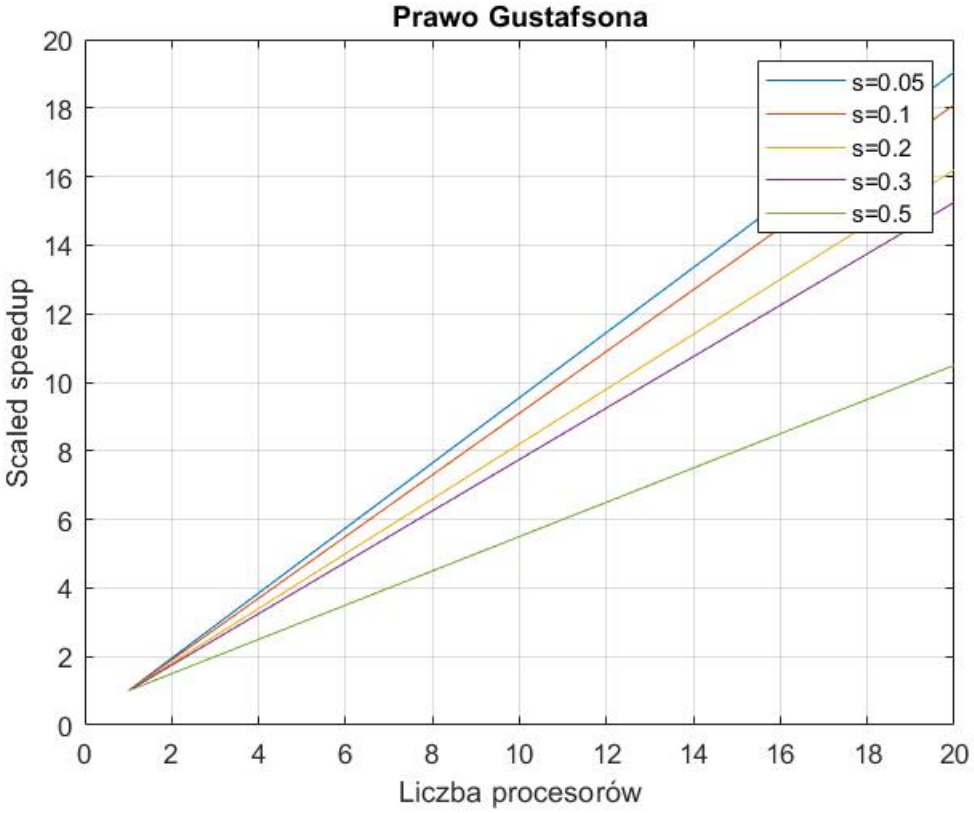
Rysunek 2: Prawo Amdahl'a

1.2.2 Słabe skalowanie

Drugim podejściem do badania skalowalności jest słabe skalowanie(z ang.*weak scaling*). W tym przypadku zwiększany jest równocześnie rozmiar problemu jak i ilość procesorów pracujących nad jego rozwiązaniem. *Przyspieszenie skalowane* limitowane jest prawem Gustafson-a.

$$scaled\ speedup \leq s + p * N$$

oznaczenia jak wyżej.



Rysunek 3: Prawo Gustafsona

1.2.3 Skalowanie superliniowe

Należy także wspomnieć o zjawisku zwanym *superlinowym skalowaniem*. Jest to zjawisko ,w którym w miarę zwiększania liczby procesorów czas wykonywania zmniejsza się bardziej niż liniowo. Istnieją różne wyjaśnienia tego zjawiska. Jednym z nich jest dostęp do pamięci cache. Innym wyjaśnieniem zjawiska superliniowego skalowania możeb być sup optymalny algorytm sekwencyjny. Gdy koszt pojedynczeo algorytmu wynosi $O(n^2)$ to w przypadku podzieleniu tego procesu na dwa procesory koszt wyniesie $O((0.5n)^2 = O(0.25n^2)$.

2 Analiza

2.1 Cel analizy

Celem pracy było zbadanie skalowalności kodu metody Lattice Boltzmann. Informacaja o skalowalności kodu obliczeniowego W jakim celiu to było badane. jakie może to dać nam korzysci(informacja o tym jakie symulacje sa efektywe), ewentualnie nad czym pracowac by uefektywnić metode

2.2 Opis klastrów obliczeniowych

2.2.1 Prometheus

System operacyjny
Linux CentOS 7

Konfiguracja
HP Apollo 8000, HPE ProLiant DL360 Gen10

Architektura/Procesory
Intel Xeon (Haswell / Skylake)

Liczba rdzeni obliczeniowych
53604

Liczba kart GPGPU
144 (Nvidia Tesla K40 XL)

Pamięc operacyjna
282 TB

Pamięć dyskowa
10 PB

Moc obliczeniowa
2403 TFlops

2.2.2 Rysy

Typ
Klaster GPU

Architektura
Intel Skylake NVIDIA Volta

Liczba węzłów obliczeniowych
6

Parametry węzła
36 rdzeni 380 GB pamięci RAM 4 GPU

2.3 Badane modele

Po prostu, dać tu infrmacje jakie symulacje byly prowadzone. Rozmiary siatek, rodzaje kernela, rodzaj kraty uporzadkowane zeby latwo bylo odczytac.

Kraty W symulacji użyto dwóch modelów krat, dwuwymiarową kratę D2Q9 (Figure 1a) oraz trójwymiarową kratę D3Q27(Figure 1d)

Kernele

3 Wyniki

3.1 Prezentacja wyników

tutaj wykresy najpierw, slabe skalowanie, potem silne

3.2 Analiza wynikow

Jak dziala skalowanie, a potem sprobowac cos wnioskowac, z zaleznosci tej powierzchni przeplywu informacji do rozmiarow siatki

4 Wnioski

Co dziala dobrze, co dziala zle.