

Praca Przejściowa Inżynierska
Analiza skalowalności kodu metodą Lattice
Boltzman, wpływ kernela oraz kraty na
efektywność obliczeń

Rybski Arkadiusz
Mechanika i Projektowanie Maszyn
numer indeksu 292784

2020

Spis treści

1	Wprowadzenie	3
1.1	Metoda Lattice Boltzmann	3
1.1.1	Rodzaje krat	3
1.1.2	Algorytm	5
1.1.3	Rodzaje kernela	5
1.2	Skalowalność kodu	6
1.2.1	Silne skalowanie	7
1.2.2	Słabe skalowanie	8
1.2.3	Skalowanie superliniowe	9
2	Analiza	10
2.1	Cel analizy	10
2.2	Opis klastrów obliczeniowych	10
2.2.1	Prometheus	10
2.2.2	Rysy	10
2.3	Badane modele	11
3	Wyniki	11
3.1	Prezentacja wyników	11
3.2	Analiza wynikow	11
4	Wnioski	11

1 Wprowadzenie

1.1 Metoda Lattice Boltzmann

Metoda Lattice Boltzmann jest metodą numeryczną służącą do rozwiązywania równań z zakresu mechaniki płynów. Metoda kratowa Boltzmanna oparta jest na równaniu Boltzmanna([Rownanie 1](#)).

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

gdzie

$f(x, \xi, t)$ oznacza funkcję rozkładu

x oznacza położenie

ξ oznacza prędkości cząstek

t oznacza czas

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

$$f_i(x + c_i * \Delta t, t + \Delta t) = f_i(x, t) + \Omega(x, t) \quad (2)$$

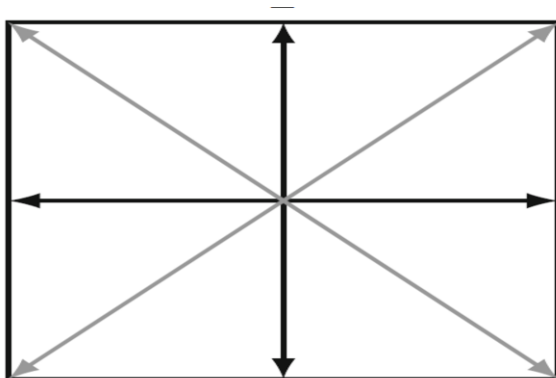
1.1.1 Rodzaje krat

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.

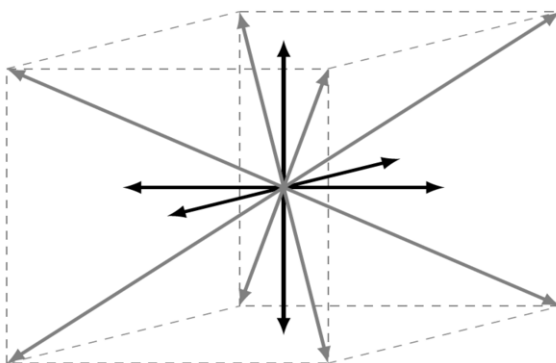
$$D_n Q_m$$

gdzie n oznacza ilość wymiarów, natomiast m oznacza ilość możliwych kierunków prędkości.

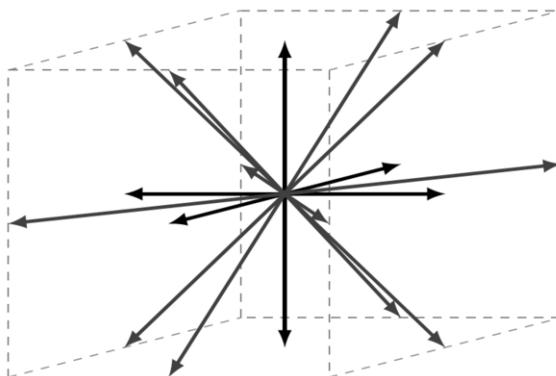
Przykładowe kraty zostały zamieszczone na poniższych obrazkach.



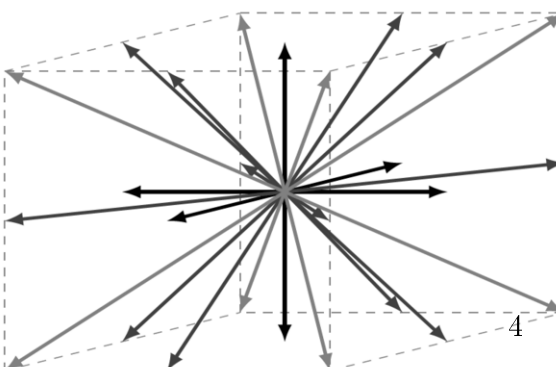
(a) D2Q9



(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

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

Faza propagacji

Równanie propagacji

$$f_i(x + c_i * \Delta t, t + \Delta t) = f_i^*(x, t) \quad (4)$$

Całość mechanizmu można podsumować w następujących 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 algorytmie operator kolizji BGK(Bhatnagar-Gross-Krook) to jeden z wielu możliwych operatorów kolizji. Inne z nich to *MRT(multiple relaxation time)* czy *Cumulant Collision Operator*.

Operator kolizji musi spełniać równania zachowania masy(Równanie 5), momentów (Równanie 6), energii (Równanie 7), a także zasadę zachowania entropii.

$$\int \Omega(f) d^3\xi = 0 \quad (5)$$

$$\int \Omega(f) \xi d^3\xi = 0 \quad (6)$$

$$\int \Omega(f) \xi^2 d^2 \xi = 0 \quad (7)$$

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

BGK operator

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

Multiple relaxation time collision operator

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

1.2 Skalowalność kodu

Dzięki nieustannemu rozwojowi technicznemu współcześnie jesteśmy w stanie rozwiązywać większe problemy obliczeniowe za pomocą wielu procesorów. Co zdecydowanie skraca czas wykonywania obliczeń. Skalowalność określa nam efektywność kodu w przypadku użycia zwiększonych zasobów komputerowych. 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 wykonania 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, którą wprowadzimy 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 część kodu obliczeniowego, którą nie jesteśmy w stanie zrównoleglić. Istnieją dwa główne podejścia w zakresie badania skalowalności 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

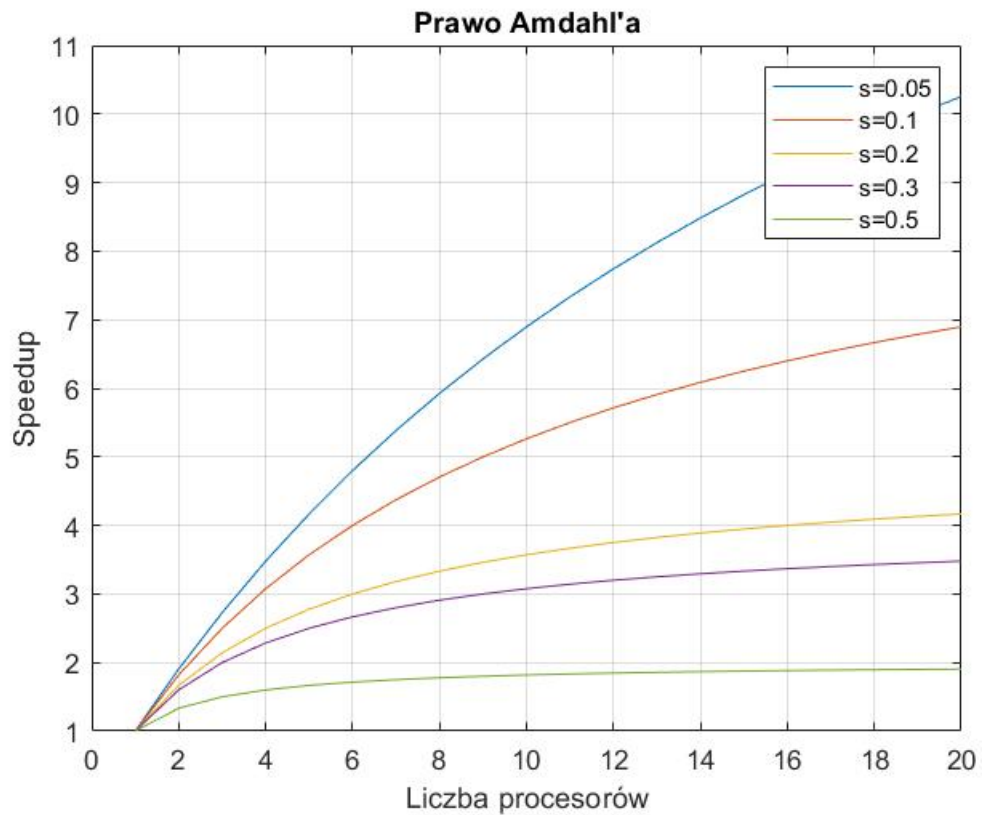
s oznacza część kodu, która nie może zostać zrównoleglona

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.



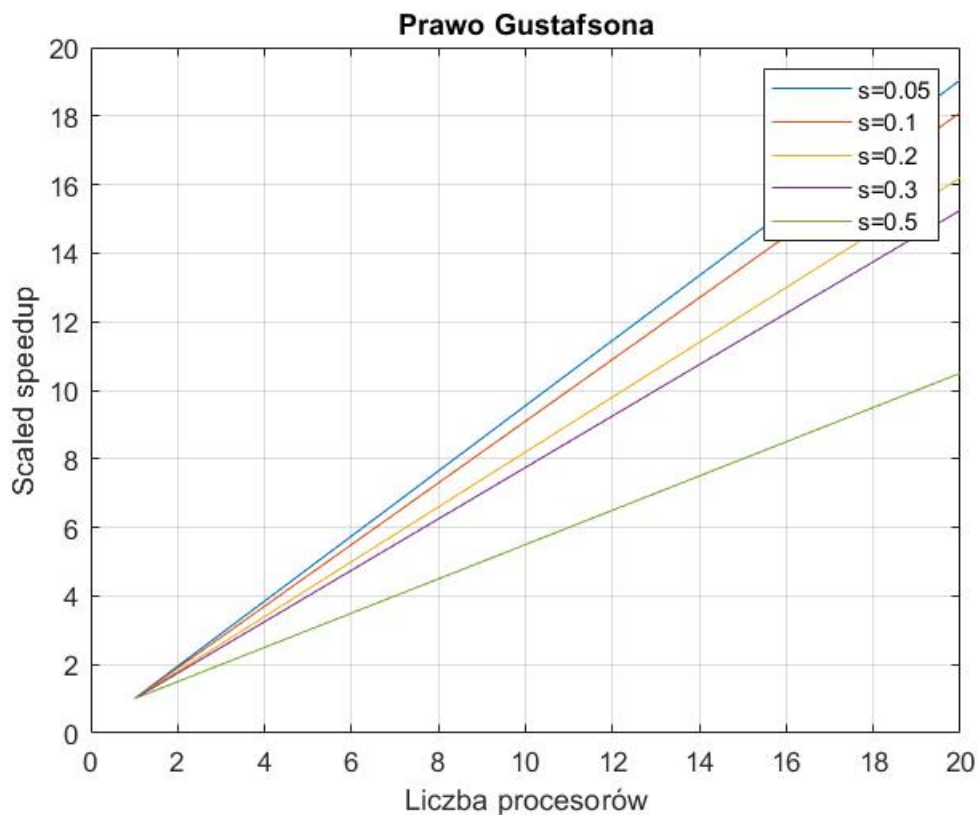
Rysunek 2: Prawo Amdahl'a

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.

$$scaled\ speedup \leq s + pN$$

oznaczenia jak wyżej.



Rysunek 3: Prawo Gustafsona

1.2.3 Skalowanie superliniowe

Należy także wspomnieć o zjawisku zwanym *superliniowym 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 może być superoptymalny algorytm sekwencyjny. Gdy koszt pojedynczego 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. Informacja o skalowalności kodu obliczeniowego W jakim celu to było badane. jakie może to dać nam korzyści(informacja o tym jakie symulacje są efektywne), ewentualnie nad czym pracować by uoefektywnić metodę

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)

Pamięć 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 informacje jakie symulacje były prowadzone. Rozmiary siatek, rodzaje kernela, rodzaj kraty uporządkowane żeby łatwo było odczytać.

Kraty W symulacji użyto dwóch modeli krat, dwuwymiarową kratę D2Q9 ([Figure 1a](#)) oraz trójwymiarową kratę D3Q27([Figure 1d](#))

Kernele

3 Wyniki

3.1 Prezentacja wyników

tutaj wykresy najpierw, słabe skalowanie, potem silne

3.2 Analiza wyników

Jak działa skalowanie, a potem spróbować coś wnioskować, z zależności tej powierzchni przepływu informacji do rozmiarów siatki

4 Wnioski

Co działa dobrze, co działa źle.