

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

Rybski Arkadiusz

2020

Spis treści

1	Wprowadzenie	3
1.1	Metoda Lattice Boltzmann	3
1.2	Skalowalność kodu	4
2	Analiza	6
2.1	Cel analizy	6
2.2	Opis klastrów obliczeniowych	6
2.3	Badane modele	6
3	Wyniki	7
3.1	Prezentacja wyników	7
3.2	Analiza wyników	7
4	Wnioski	7

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.

Równanie Boltzmanna

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

gdzie

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

x oznacza położenie

ξ oznacza prędkości cząstek

t oznacza czas

Okazuje się, że w skali makroskopowej z równań Boltzmanna łatwo można przejść na równania Naviera-Stokesa. Niestety nie rozwiązuje to problemu analitycznego rozwiązywania równania. Natomiast okazuje się, iż mimo skomplikowanej formy, równanie Boltzmanna w prosty sposób można zaimplementować. W ten sposób możemy otrzymać równanie kratowe Boltzmanna.

Boltzmann Lattice equation

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

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, ale każdy węzeł musi zawierać 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:

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

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

Faza propagacji

Równanie propagacji

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

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

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 SRT(single relaxation time).

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/kodów(?): silne i słabe skalowanie.

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 (Czy to dobry polski odpowiednik?)

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.

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.

Skalowanie superliniowe

Czasami zachodzi, podać przykłady 1pamięć cache

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

Prometheus

System operacyjny

Linux CentOS 7

Konfiguracja

HP Apollo 8000, HPE ProLiant DL360 Gen10

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

2.3 Badane modele

Po prostu, dać tu informacje jakie symulacje były prowadzone. Rozmiary siatek, rodzaje kernela, rodzaj kraty uporządkowane żeby łatwo było odczytać.

Kraty

Siatki

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.