

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

Rybski Arkadiusz  
Mechanika i Projektowanie Maszyn  
numer indeksu 292784

2020

# Spis treści

<b>1 Wprowadzenie</b>	<b>3</b>
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 . . . . .	8
<b>2 Analiza</b>	<b>9</b>
2.1 Cel analizy . . . . .	9
2.2 Opis klastrów obliczeniowych . . . . .	9
2.2.1 Prometheus . . . . .	9
2.2.2 Rysy . . . . .	9
2.3 Badane modele . . . . .	10
<b>3 Wyniki</b>	<b>10</b>
3.1 Prezentacja wyników . . . . .	10
3.1.1 Silne Skalowanie . . . . .	11
3.1.2 Słabe Skalowanie . . . . .	12
3.1.3 Wykresy przepływów informacji . . . . .	20
3.2 Analiza wyników . . . . .	23

# 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 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

$\xi$  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 Boltzmana w prosty sposób można zaimplementować. W ten sposób możemy otrzymać równanie kratowe Boltzmanna([Równanie 2](#)).

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

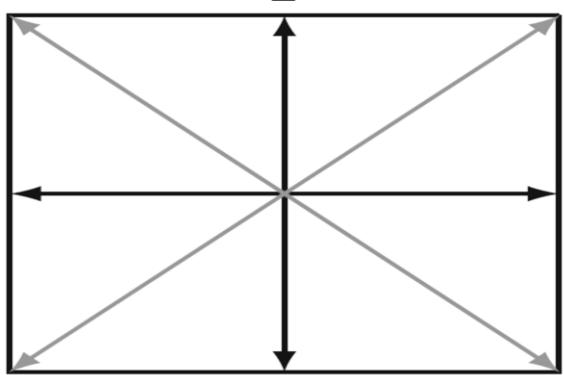
### 1.1.1 Rodzaje krat

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.

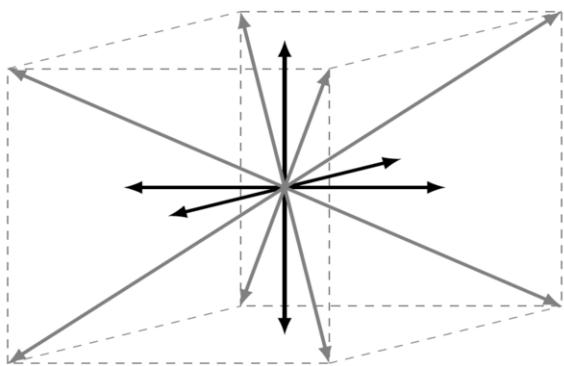
$$D_n Q_m$$

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

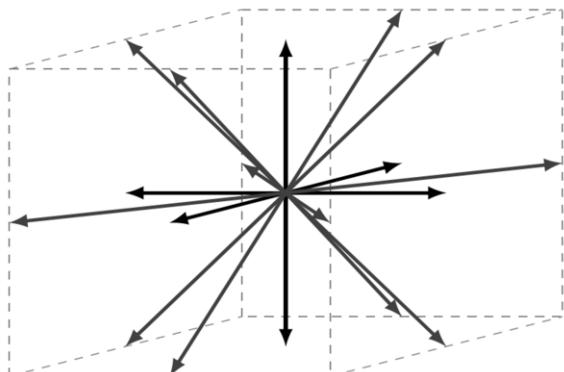
Przykładowe kraty zostały zamieszczone na poniższych obrazkach.



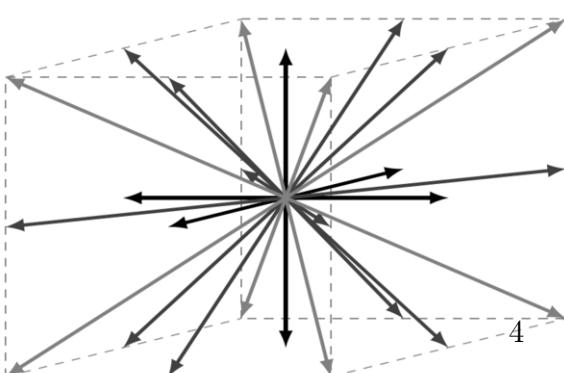
(a) D2Q9



(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.

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

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

$$\int \Omega(f) \xi^2 d^3\xi = 0 \quad (7)$$

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 colission operator**

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

## 1.2 Skalowalność kodu

Dzięki nieustannemu rozwojowi technicznemu wspólnie jesteśmy w stanie rozwijać większe problemy obliczeniowe za pomocą wielu procesorów. Co zdecydowanie skracą czas wykonywania obliczeń. Skalowalność określa nam efektywność kodu w przypadku użycia zwiększych 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, która 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 skalowania 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

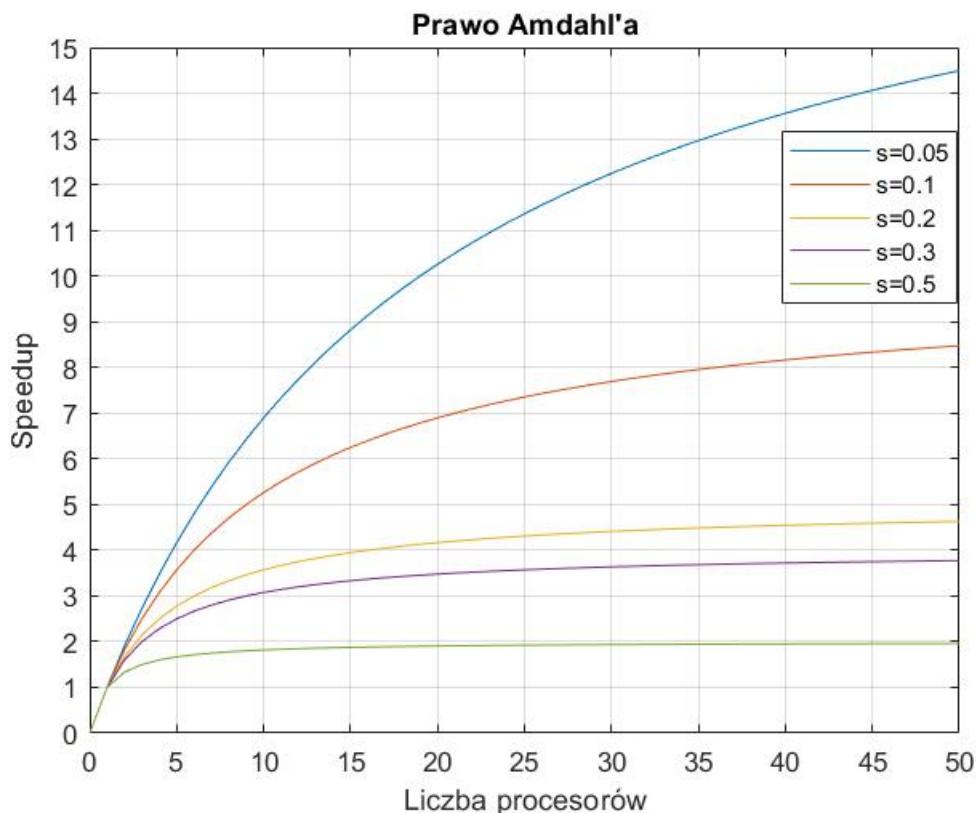
$s$  oznacza cześć kodu, która nie może zostać zrównoleglona

$p$  cześć 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.



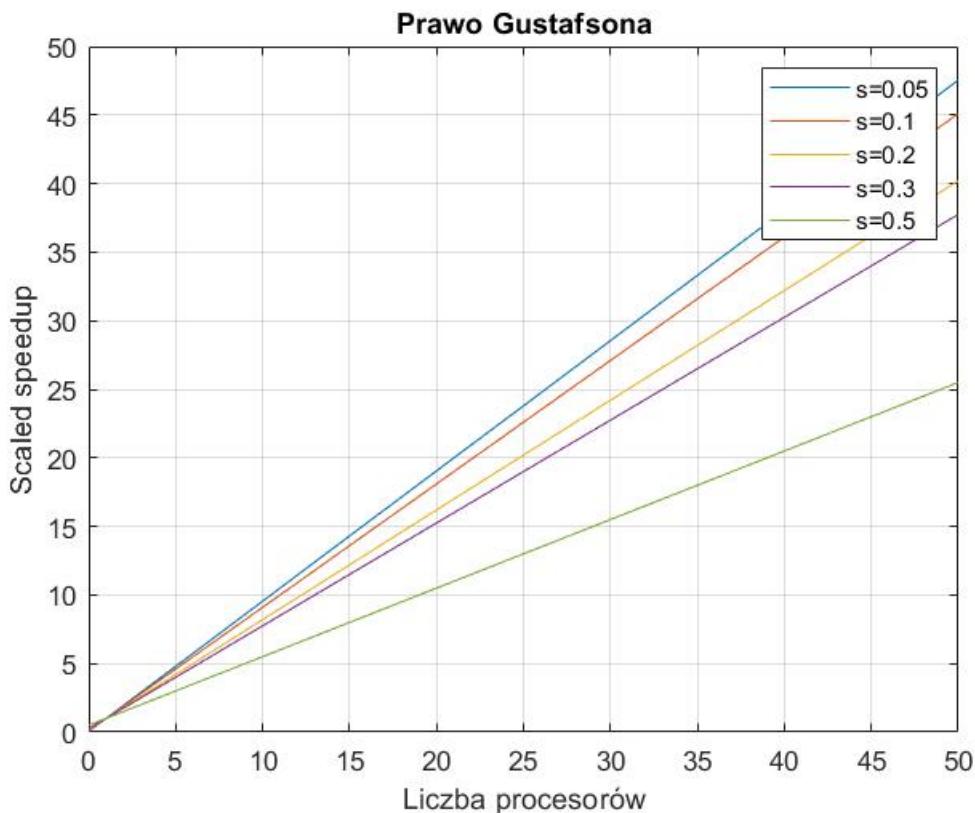
Rysunek 2: Prawo Amdahl'a

### 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.

$$\text{scaled speedup} \leq s + pN$$

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 sup optymalny 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 obliczeniowego kodu metody Lattice Boltzmann w celu określenia jej efektywności. Zbadano również wpływ powierzchni przepływu informacji do objętości całego problemu obliczeniowego. Dzięki zrealizowanym badaniom będziemy mogli określić optymalne wykorzystanie zasobów obliczeniowych.

### 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

Badanym modelem testowym był przepływ z wymianą ciepła wokół walca.

**Krata** W symulacji użyto trójwymiarowej kraty typu D3Q27([Figure 1d](#))

**Kernel****Schematy obliczeniowe**

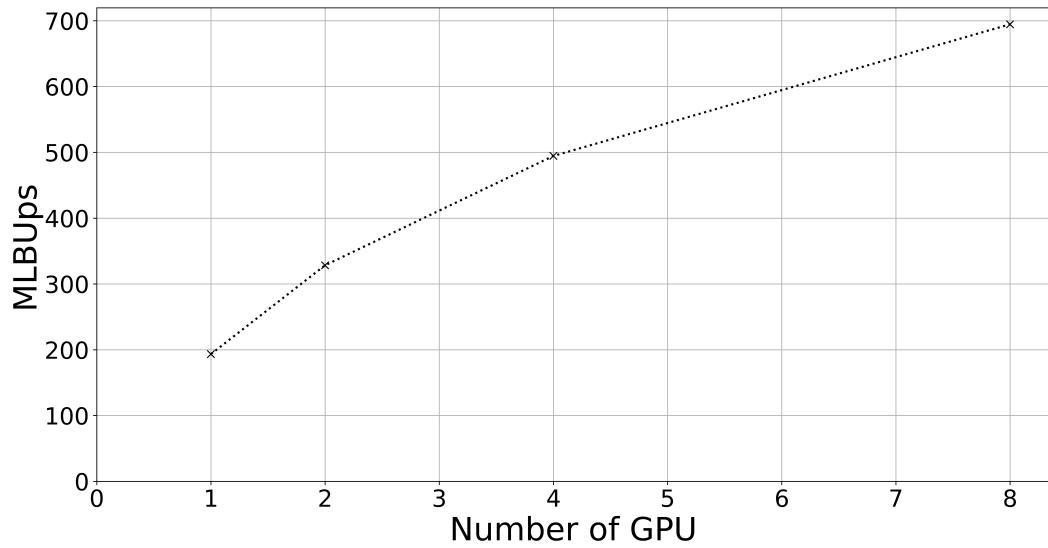
## 3 Wyniki

### 3.1 Prezentacja wyników

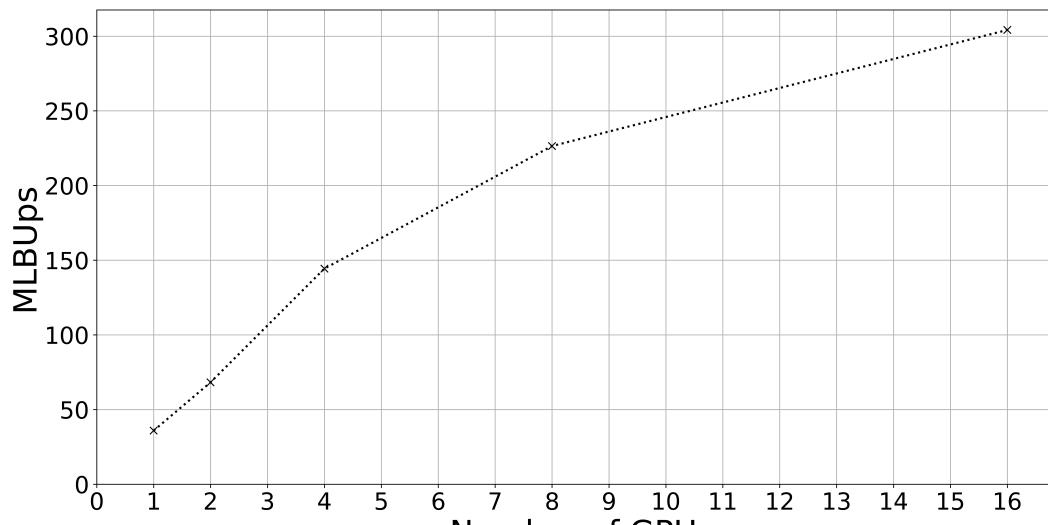
Poniżej zostały przedstawione wyniki symulacji.

### 3.1.1 Silne Skalowanie

Wykresy dot. silnego skalowania zostały stworzone w wyniku symulacji powyżej opisanego modelu na siatce o rozmiarze  $4096 \times 640 \times 3$ , skutkowało to obciążeniem karty GPU  $\sim 10Gb$ .

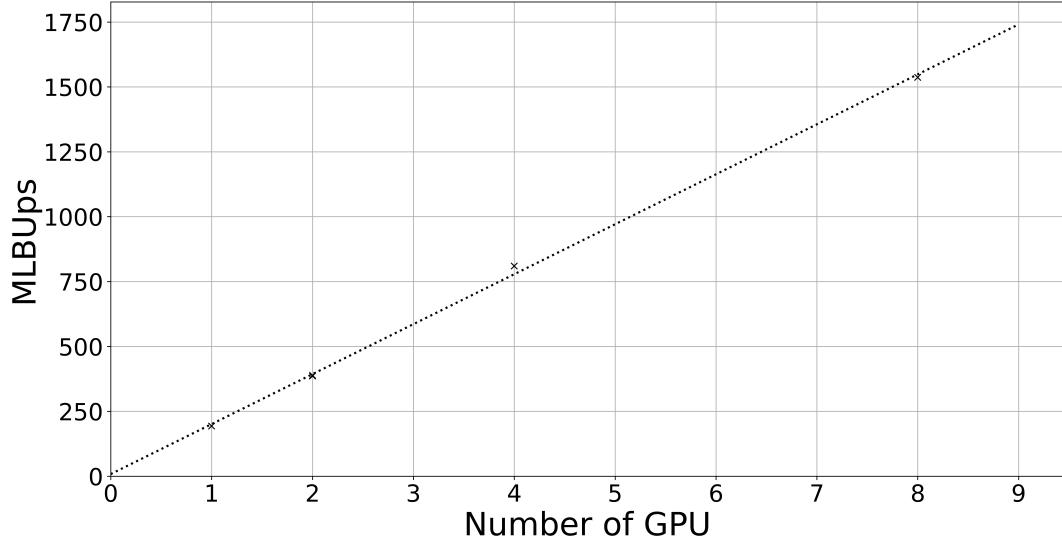


(a) Rysy

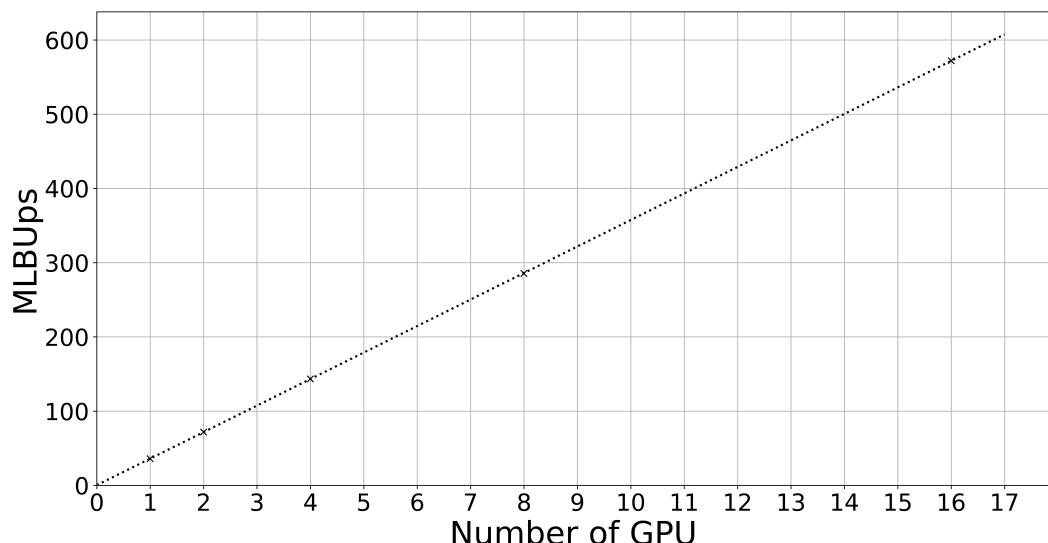


(b) Prometheus

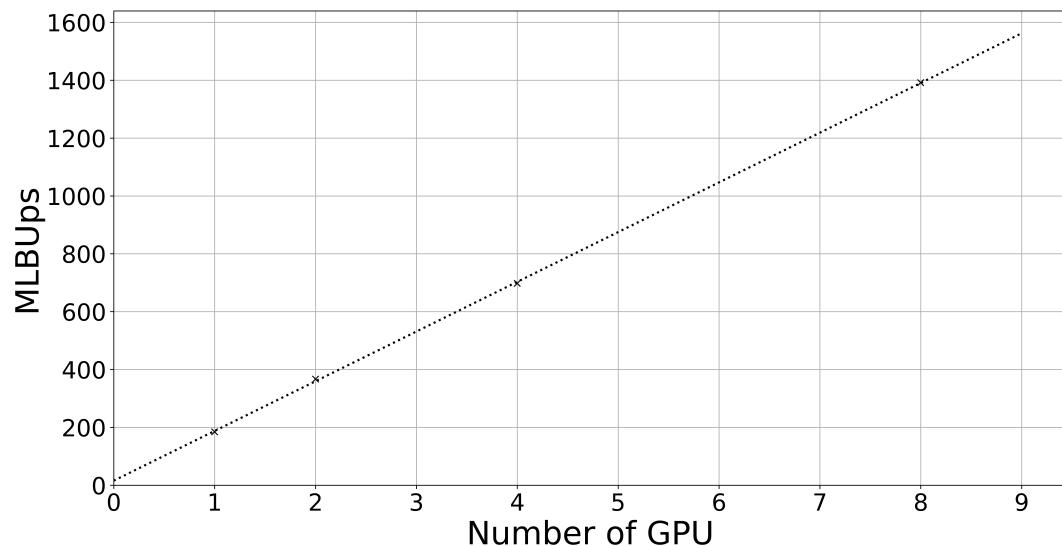
### 3.1.2 Słabe Skalowanie



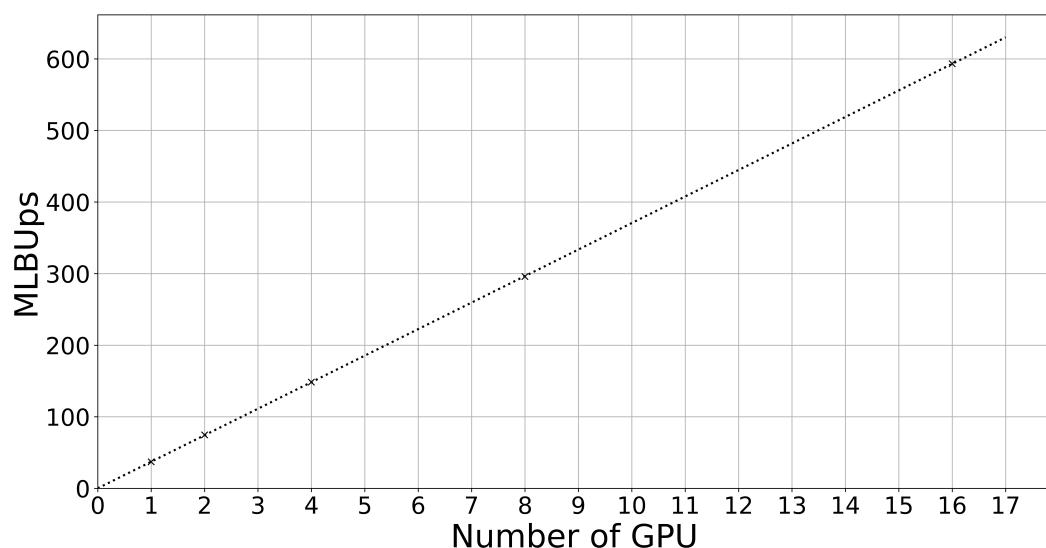
(a) Rysy: 4096 x 640 x 3



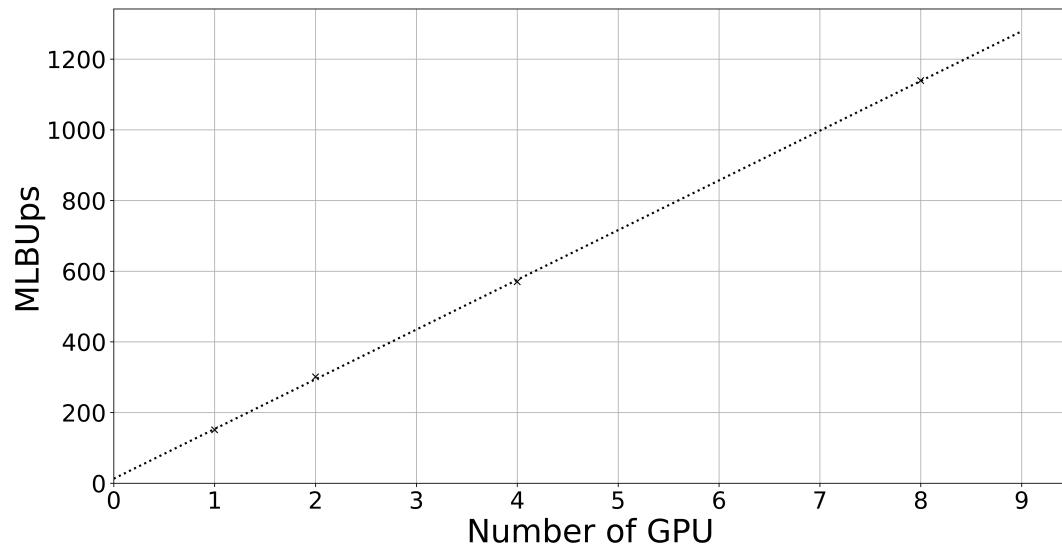
(b) Prometheus: 4096 x 640 x 3



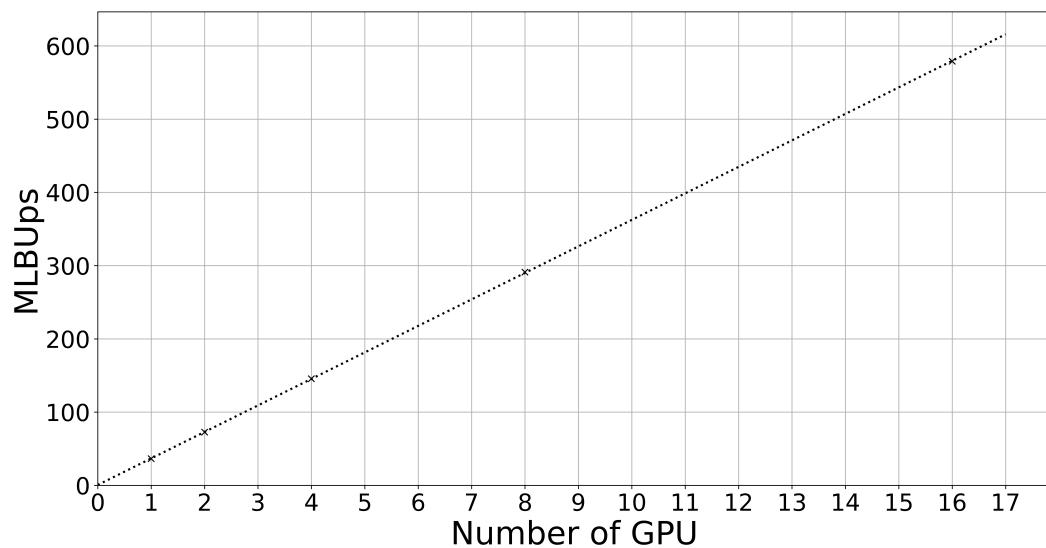
(c) Rysy: 2816 x 440 x 3



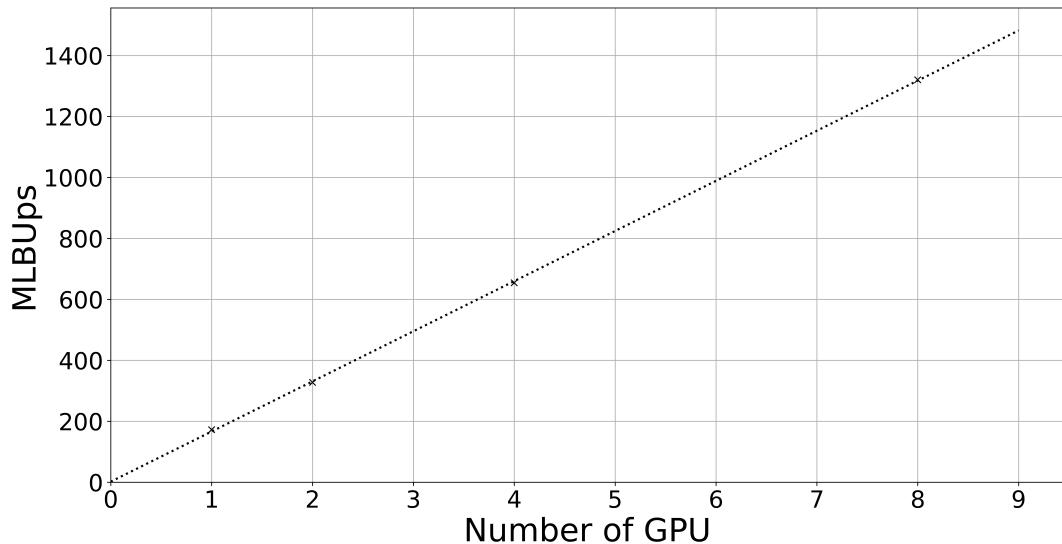
(d) Prometheus: 2816 x 440 x 3



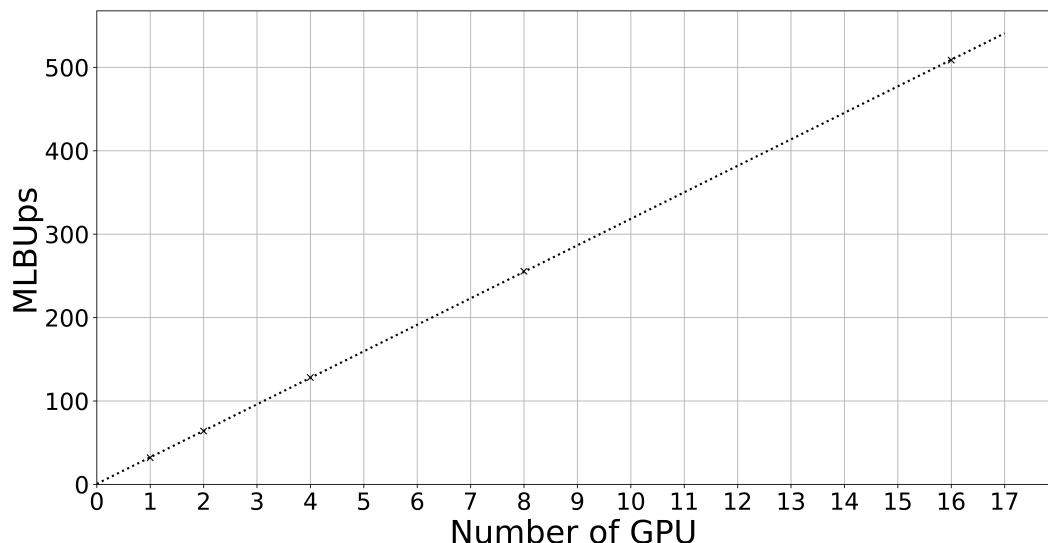
(e) Rysy: 5632 x 220 x 3



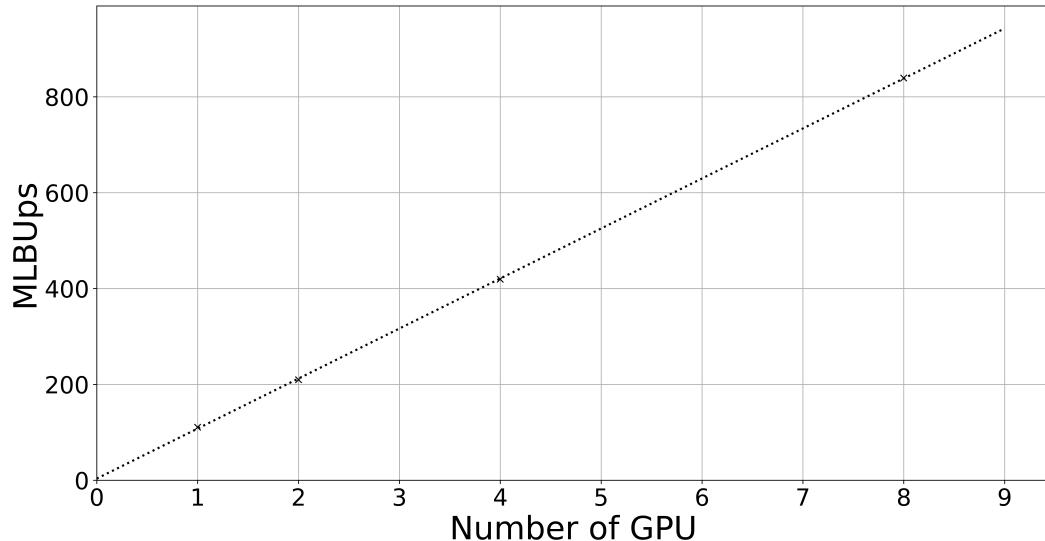
(f) Prometeusz: 5632 x 220 x 3



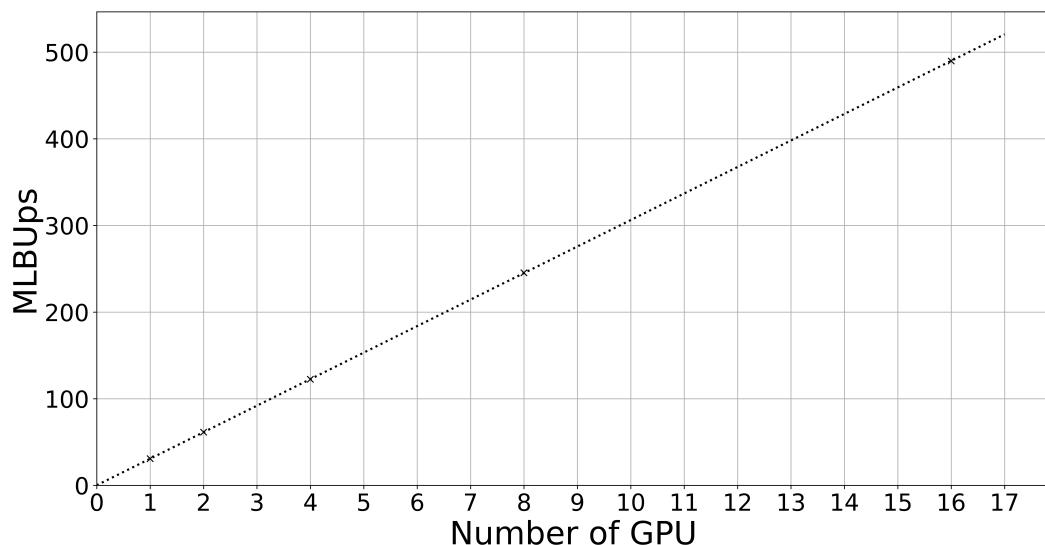
(g) Rysy: 8192 x 320 x 3



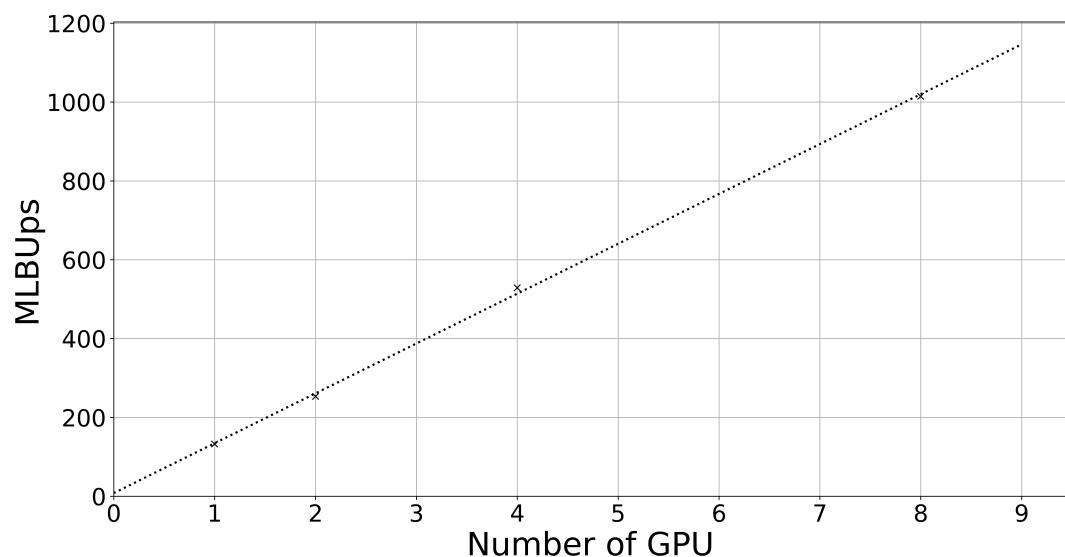
(h) Prometeusz: 8192 x 320 x 3



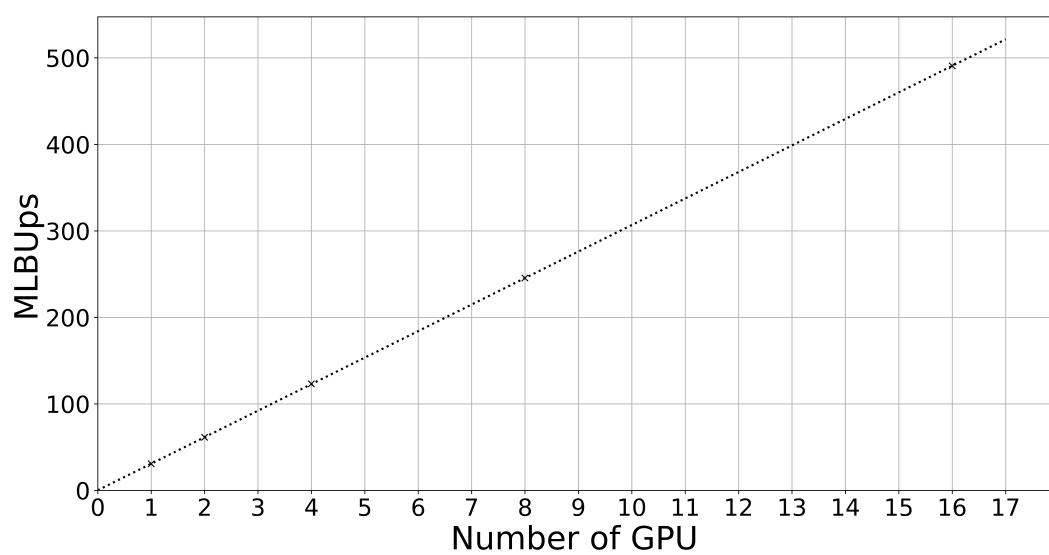
(i) Rysy: 11264 x 110 x 3



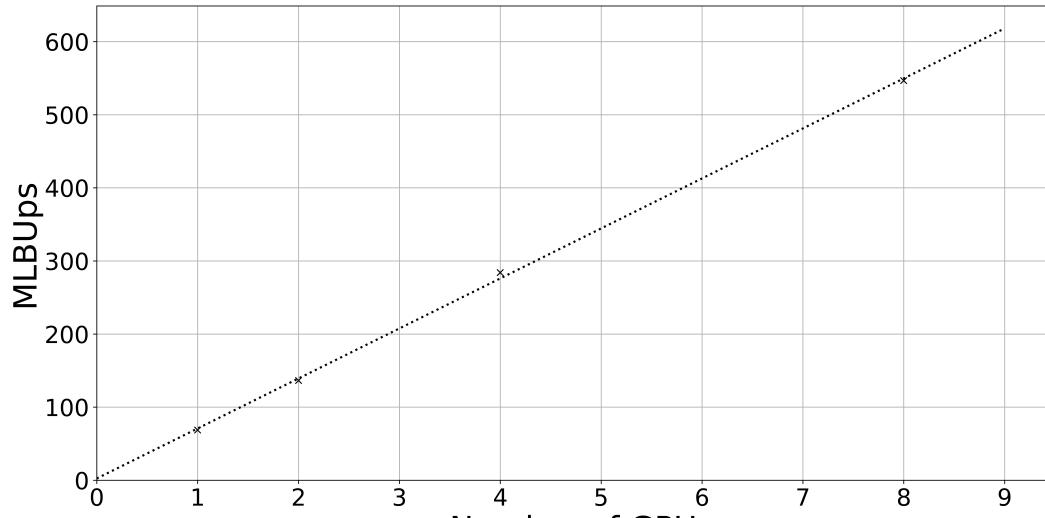
(j) Prometheus: 11264 x 110 x 3



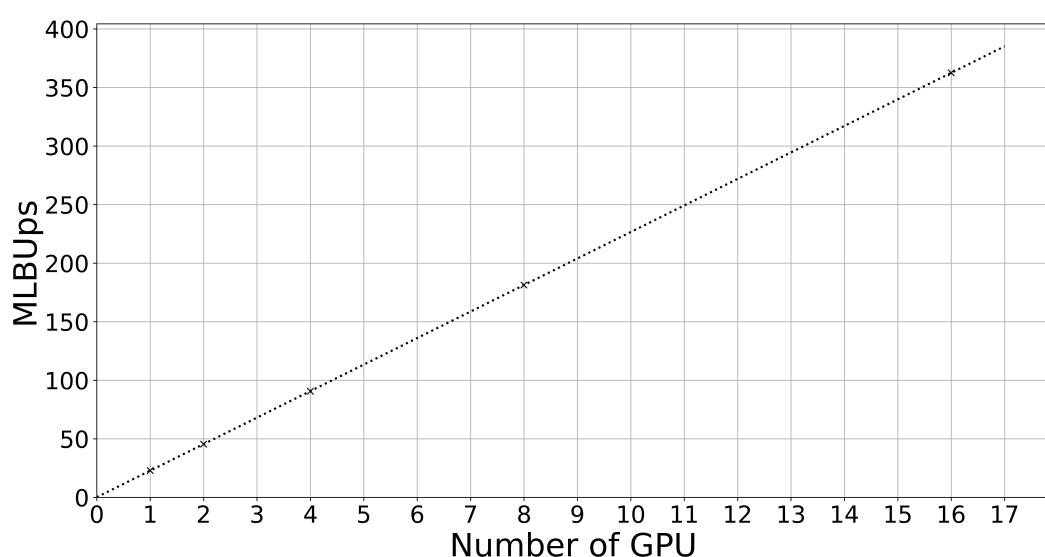
(k) Rysy: 16384 x 160 x 3



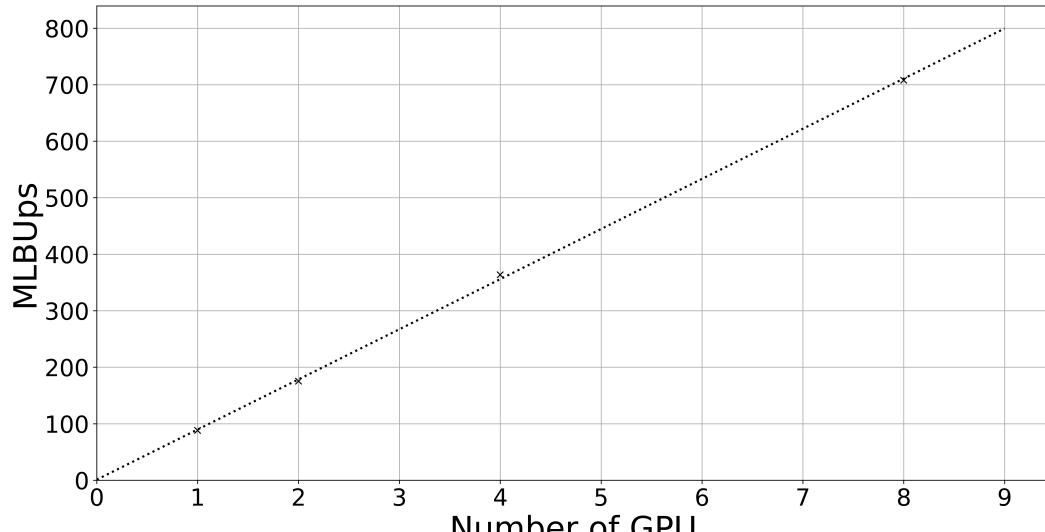
(l) Prometheus: 16384 x 160 x 3



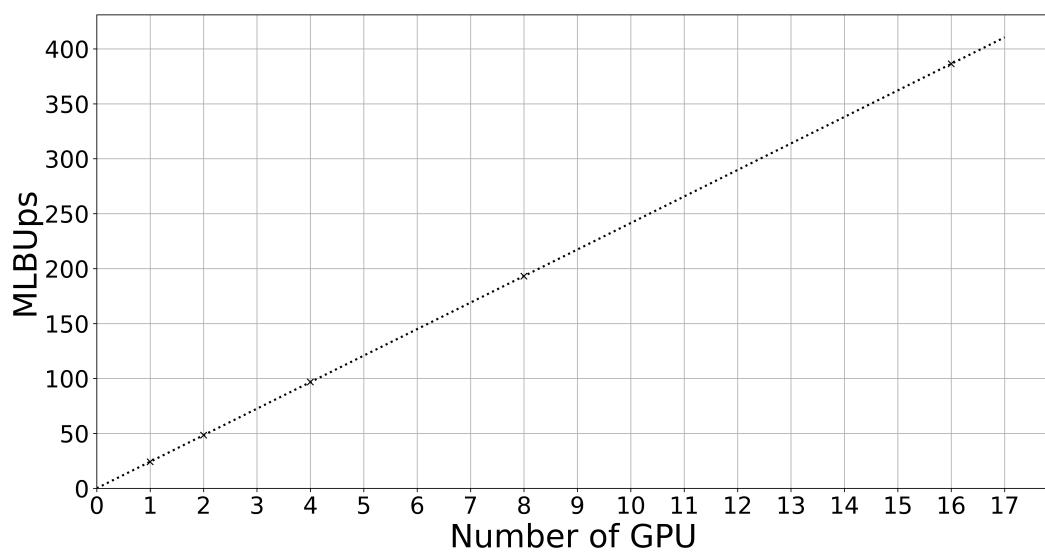
(m) Rysy: 22528 x 55 x 3



(n) Prometheus: 22528 x 55 x 3



(o) Rysy: 32768 x 80 x 3

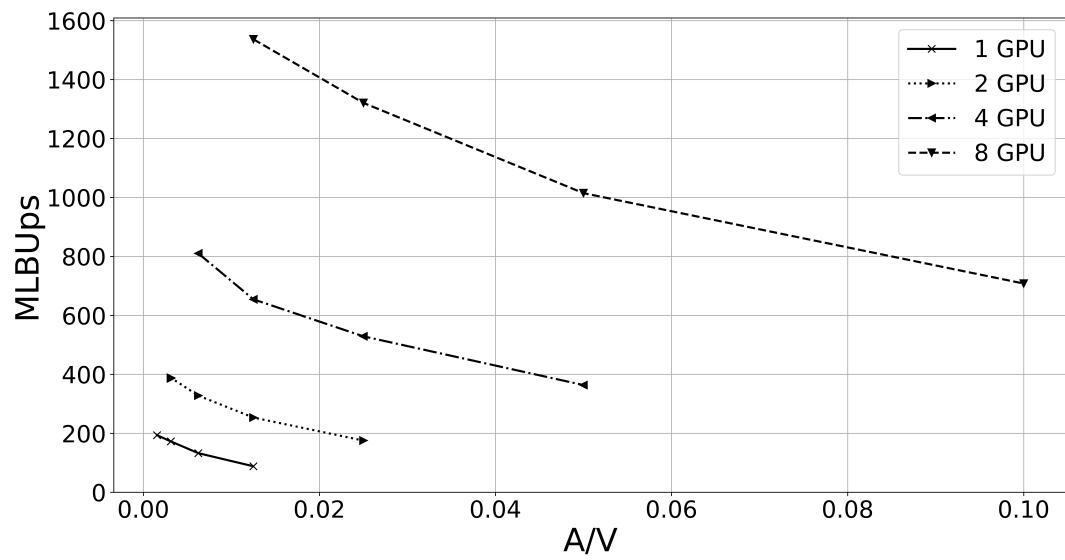


(p) Prometheus: 32768 x 80 x 3

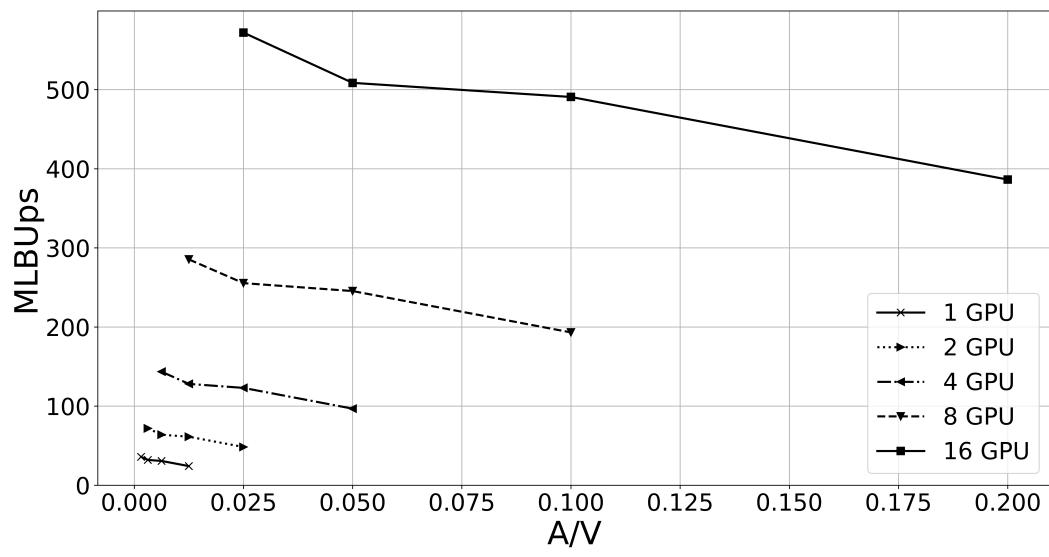
Rysunek 5: Weak scaling

### **3.1.3 Wykresy przepływów informacji**

Jedynym z istotnych paramterów wpływającym na szybkość kodu obliczeniowego metody Lattice Boltzmann jest stosunek powierzchni przepływu informacji do całkowitego rozmiaru siatki obliczeniowej. Poniżej przedstawiono wykresy mające na celu zbadać te zależność.

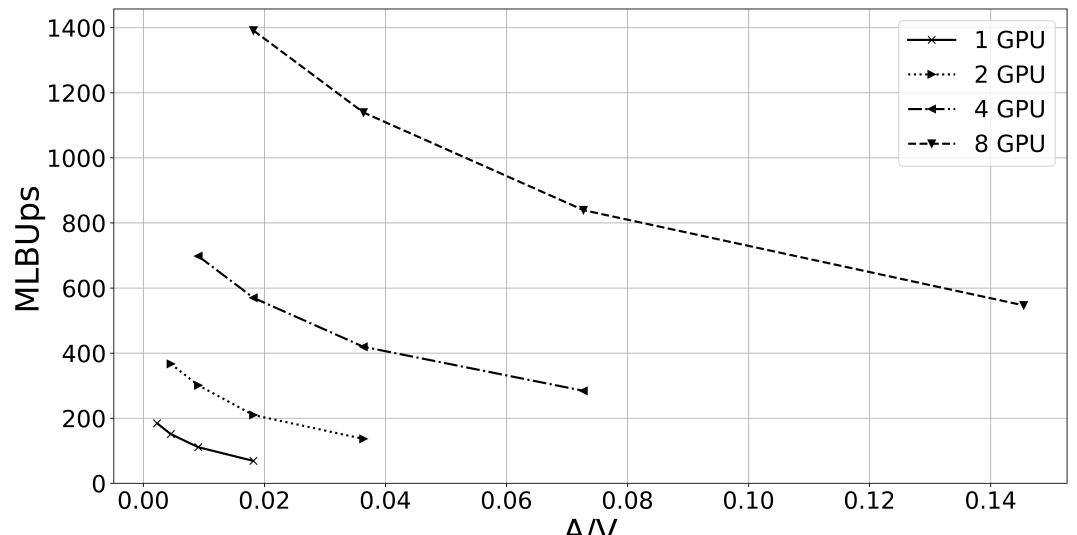


(a) Rysy

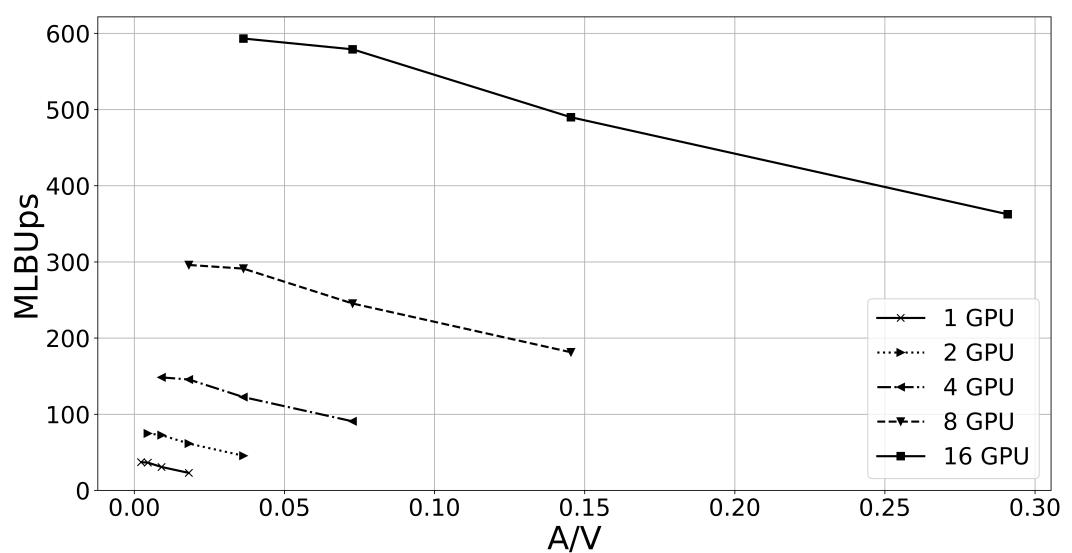


(b) Prometheus

Rysunek 6: A/V plots 10Gb load



(a) Rysy



(b) Prometheus

Rysunek 7: A/V plots 5Gb load

### 3.2 Analiza wyników

Z wykresów silnego skalowania możemy zauważać, że dla stałego rozmiaru problemu optymalnym rozwiązaniem jest prowadzenie obliczeń przy użyciu 4 kart GPU (uwzględniając czas trwania oraz koszt obliczeń).

W przypadku wykresów słabego skalowania możemy zauważać, że punkty układają się w linię. Co zgadza się z prawem Gustafsona.

Z wykresów [Figure 6](#) możemy zauważać, że przy nieodpowiednim doborze siatki obliczniowej mimo zwiększonej ilości procesorów działających możemy uzyskać mniejszą szybkość obliczeń. Co więcej na wykresach Speed(A/V) ([Figure 7](#), [Figure 6](#)) widać, że kształt lini nie zależy od ilości procesorów(kart GPU). Jest to z pewnością duże ograniczenie kodu obliczeniowego metody Lattice Boltzmann, który mimo dobrego skalowania się jest zdecydowanie wrażliwy na konfigurację siatki obliczniowej.