

Praca Przejściowa Inżynierska
Analiza skalowalności kodu TCLB opartego o
metodę Lattice Boltzmann

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	9
2.1 Cel analizy	9
2.2 Opis klastrów obliczeniowych	10
2.2.1 Prometheus	10
2.2.2 Rysy	10
2.3 Badane modele	11
3 Wyniki	12
3.1 Prezentacja wyników	12
3.1.1 Silne Skalowanie	13
3.1.2 Słabe Skalowanie	14
3.1.3 Wykresy przepływ informacji	22
3.1.4 Wydajność GPU	27
3.2 Analiza wynikow	28

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

ξ - oznacza prędkości cząstek

t - oznacza czas

$\Omega(f)$ - oznacza operator kolizji([1.1.3](#)).

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

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

,gdzie:

$c_i = (c_{ix}, c_{iy}, c_{iz})$ - oznacza prędkość cząstki w i-tym węźle.

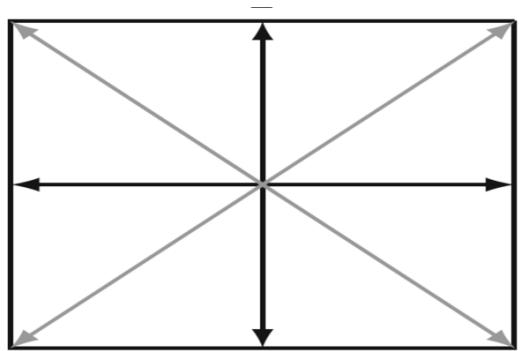
1.1.1 Rodzaje krat

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 prędkości. W literaturze przyjęto następujące oznaczenia krat:

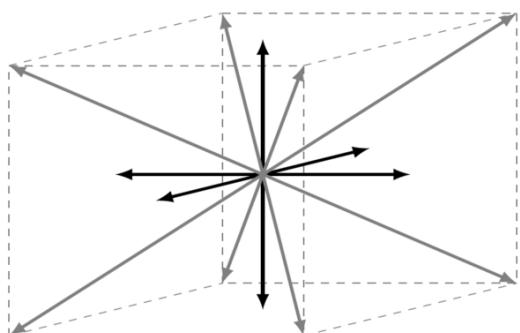
$$D_n Q_m$$

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

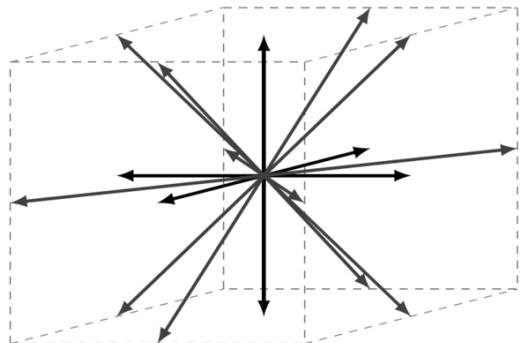
Przykładowe kraty zostały zamieszczone na poniższych obrazkach ([2], [4]).



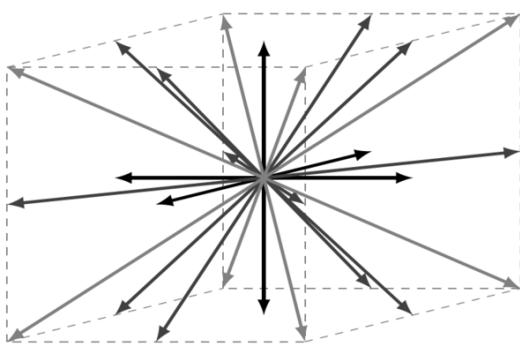
(a) D2Q9



(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 ([4]).

Faza kolizji

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

Faza 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*), *Central Moment* [1] czy *Cumulant Collision Operator* [3].

Operator kolizji musi spełniać równania zachowania masy ([Równanie 5](#)), momentów ([Równanie 6](#)), energii ([Równanie 7](#)), a także zasadę zachowania entropii[4].

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

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

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

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

BGK operator [4]

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

,gdzie:

τ - oznacza stałą czasową, nazywaną czasem relaksacji

f^{eq} - oznacza funkcję rozkładu w stanie równowagi (*equilibrium*).

Multiple relaxation time colission operator [4]

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

,gdzie:

M - oznacza macierz transformacji

S - oznacza macierz relaksacji (kolizji).

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 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, której nie jesteśmy w stanie zrównoleglić. Istnieją dwa główne podejścia w zakresie badania skalalności 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:

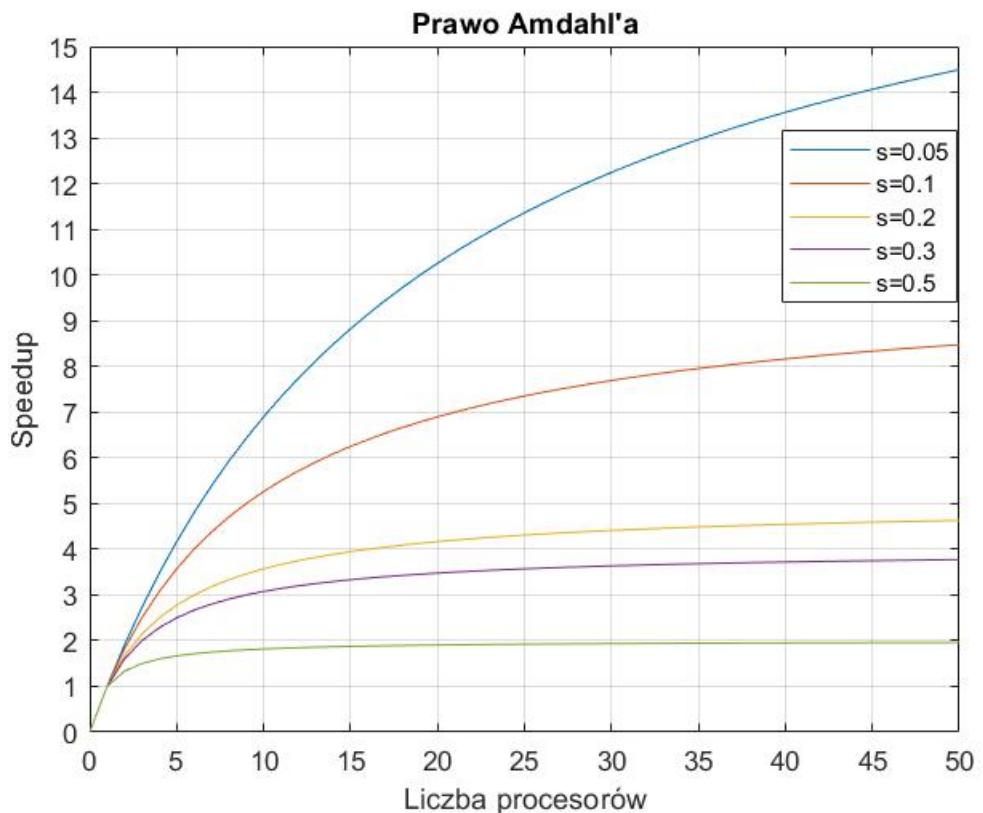
s - oznacza część kodu, która nie może zostać zrównoleglona

p - część kodu zrównoleglona

N - oznacza liczbę procesorów.

Uwaga: $s + p = 1$.

Przykładowo jeśli 10% kodu obliczeniowego nie nadaje się do zrównoleglenia to maksymalnie możemy uzyskać 10-krotne przyspieszenie procesu.



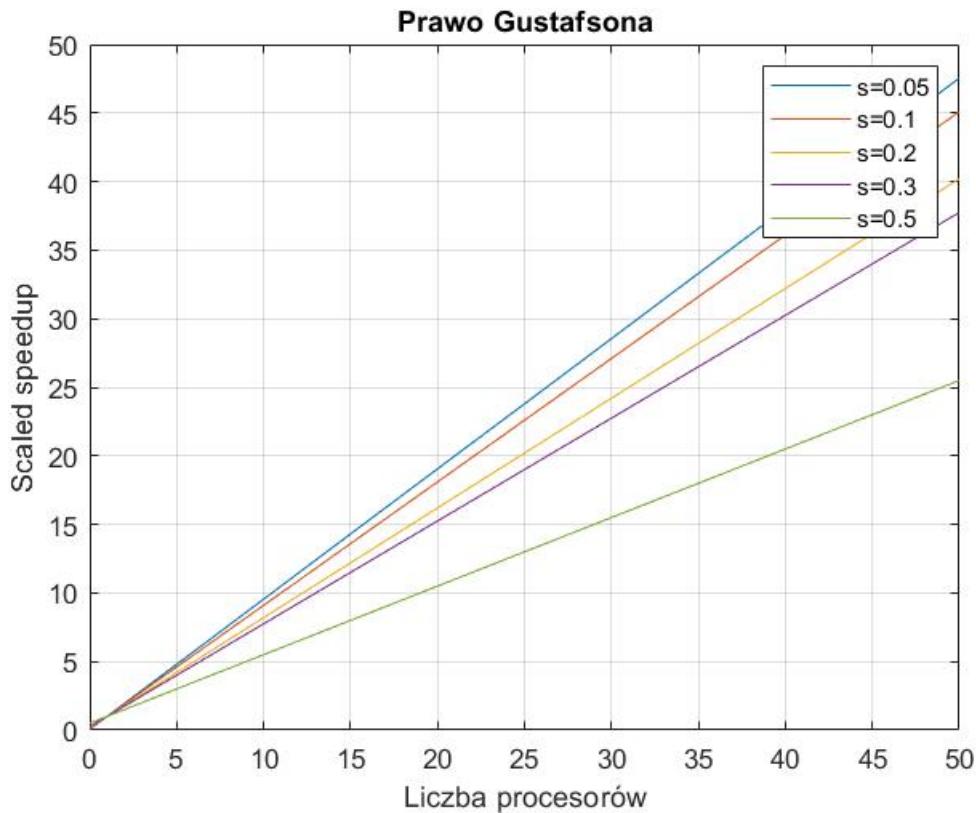
Rysunek 2: Prawo Amdahl'a

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 metody Lattice Boltzmann w celu określenia jej efektywności. Kluczowym parametrem metod numerycznych jest stosunek ilości informacji przekazywanych między

procesorami do całej objętości problemu. W metodzie Lattice Boltzmann informacja jest przekazywana na granicy styku subdomen przypisanych do kart GPU. Dlatego w tym przypadku liczy się stosunek 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)

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 (Nvidia V100 32GB)

2.3 Badane modele

Badanym modelem testowym był przepływ z wymianą ciepła wokół walca. Przestrzeń obliczeniowa została zdefiniowana następująco:

- długość przestrzeni na X - L
- długość przestrzeni na Y - H
- średnica cylindra $d = 0.1*H$
- położenie środka cylindra
 - $y_{cylindra} = 0.5H$
 - $x_{cylindra} = 0.3L$

, gdzie środek układu współrzędnych to lewy dolny róg prostokąta widocznego na załączonym zdjęciu([Rysunek 4](#)).



Rysunek 4: Przepływ wokół walca

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

Kernel Zastosowane zostały dwa kernele:

- kernel hydrodynamiczny - *Cumulant Collision Operator* [3]
- kernel populacji termicznych - *Central Moment Operator* [1]

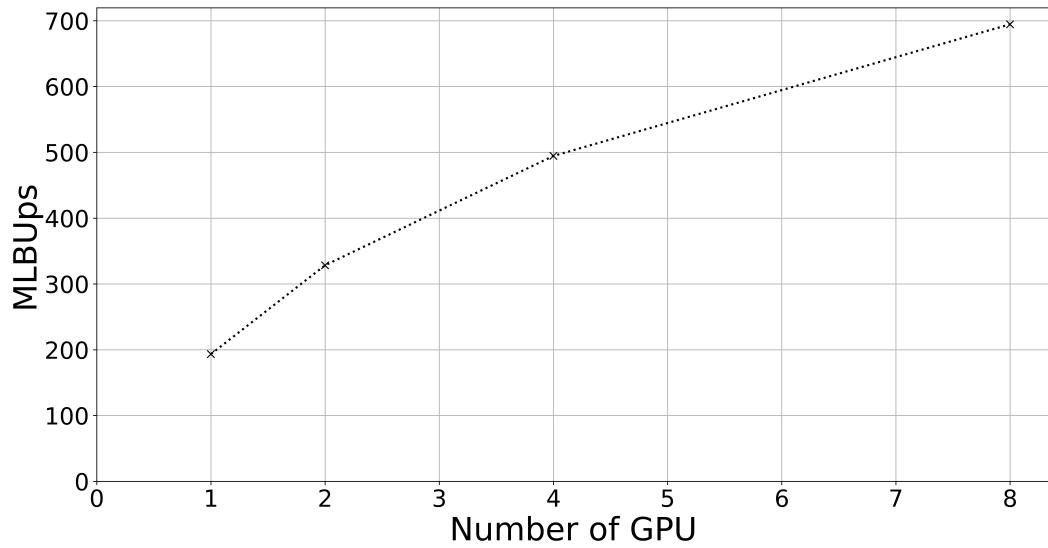
3 Wyniki

3.1 Prezentacja wyników

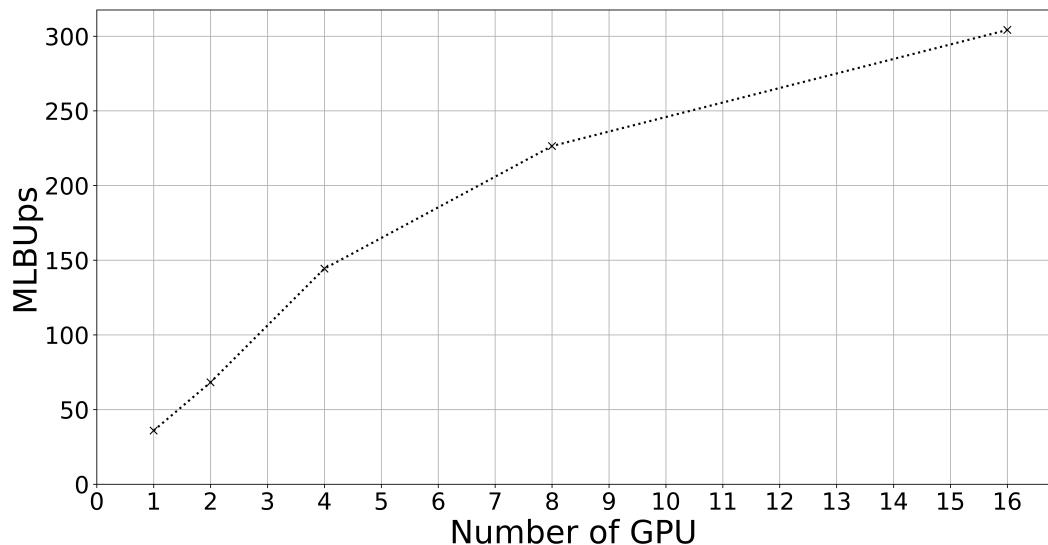
Wszystkie symulacje zostały przeprowadzone w oparciu o solver TCLB ([Solver TCLB](#)). Wyniki symulacji zostały przedstawione poniżej.

3.1.1 Silne Skalowanie

Wykresy dot. silnego skalowania zostały stworzone w wyniku symulacji powyżej opisanego modelu na siatce o rozmiarze $4096 \times 640 \times 3$, co zajmowało $\sim 10GB$ pamięci karty GPU.



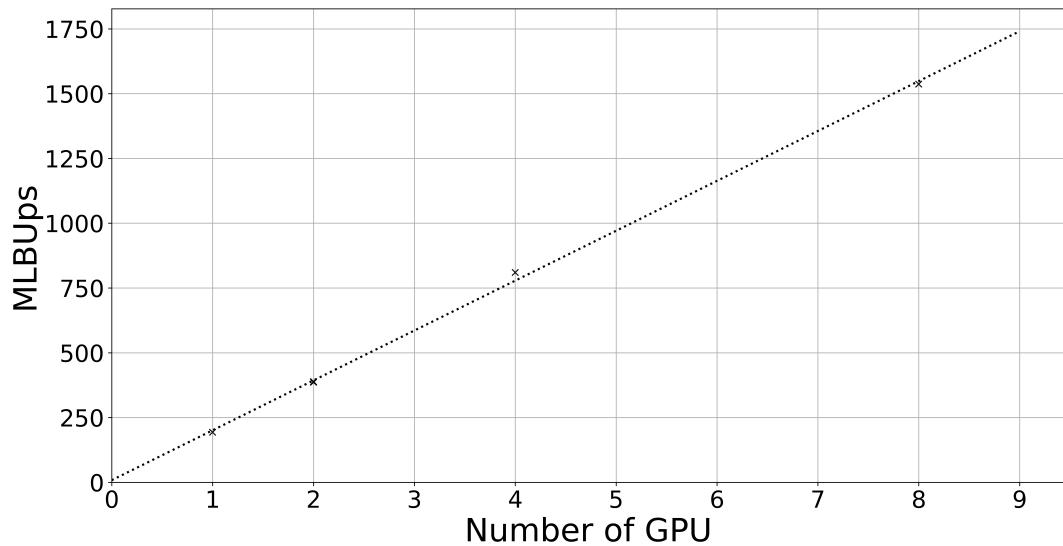
(a) Rysy



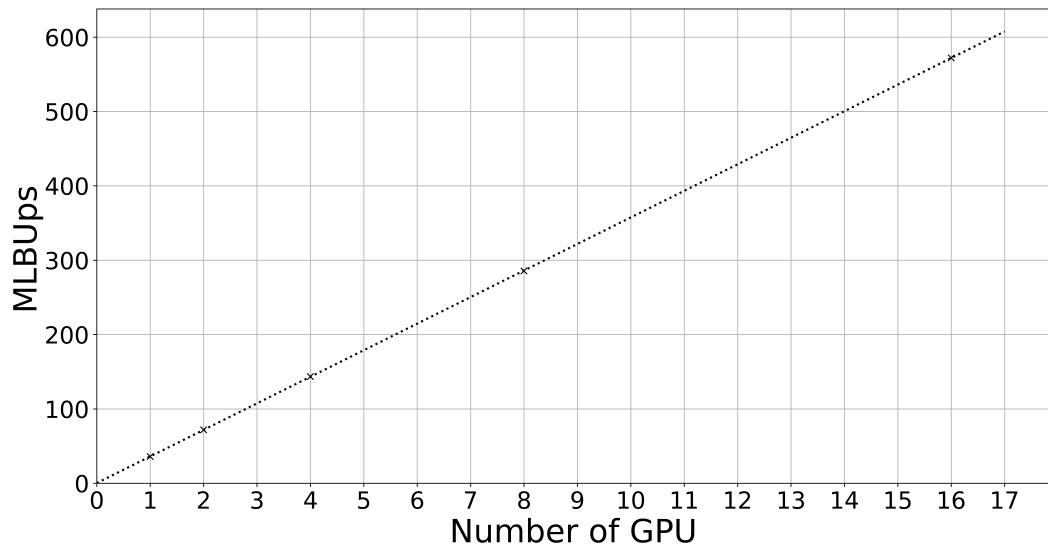
(b) Prometheus

3.1.2 Słabe Skalowanie

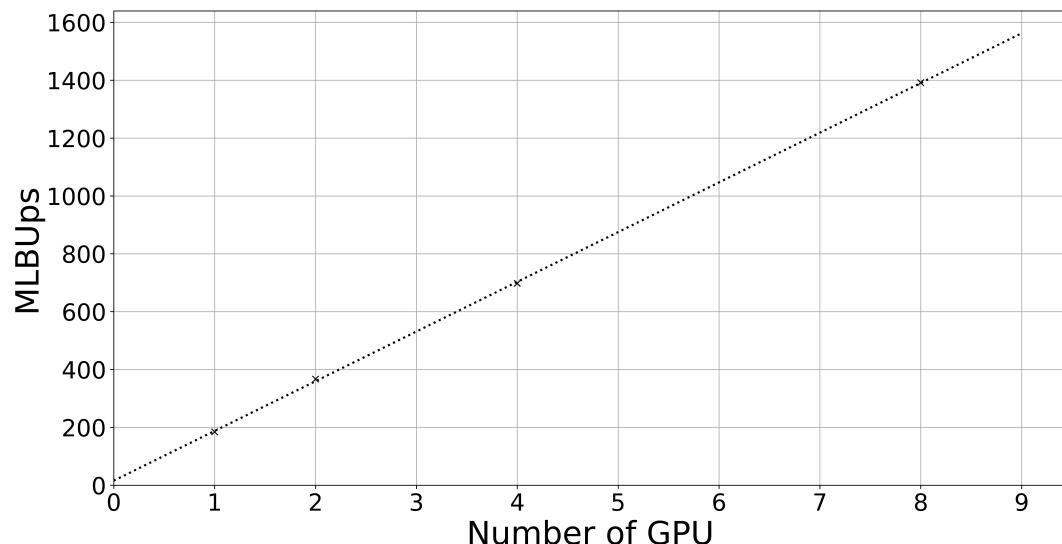
Podpis pod wykresem informuje nas o tym na jakim klastrze były przeprowadzone obliczenia, oraz rozmiar siatki przypadającej na 1 kartę GPU.



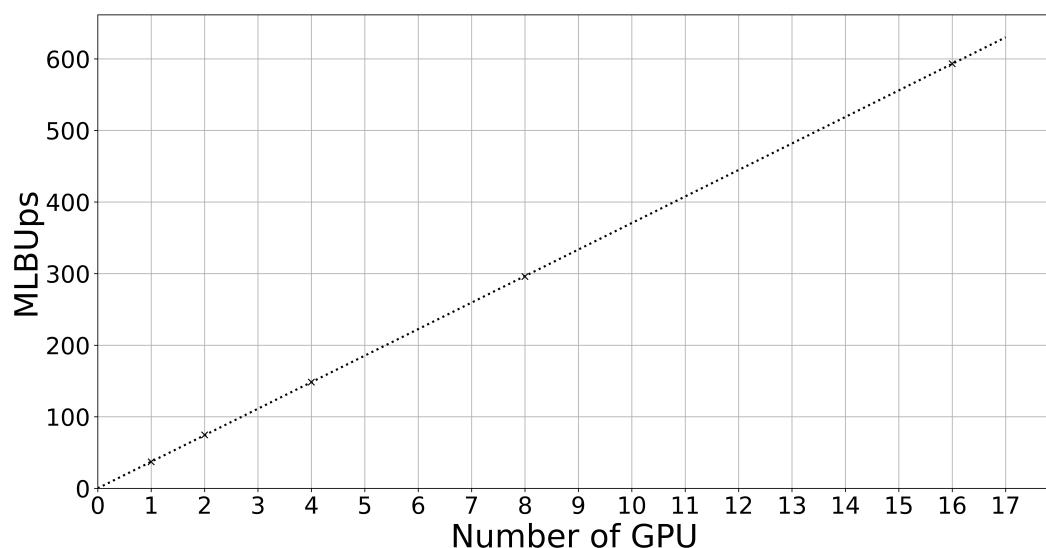
(a) Rysy: 4096 x 640 x 3



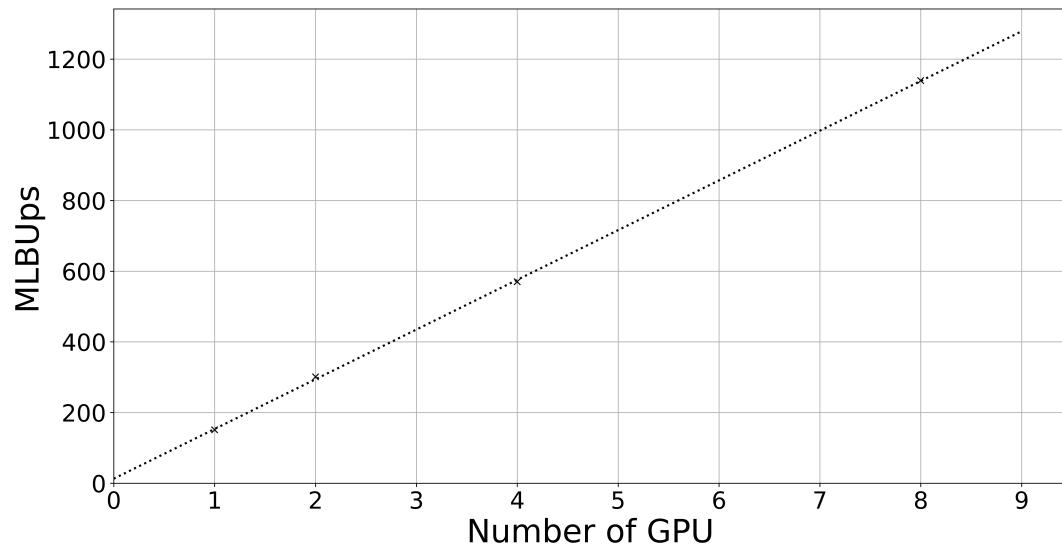
(b) Prometheus: 4096 x 640 x 3



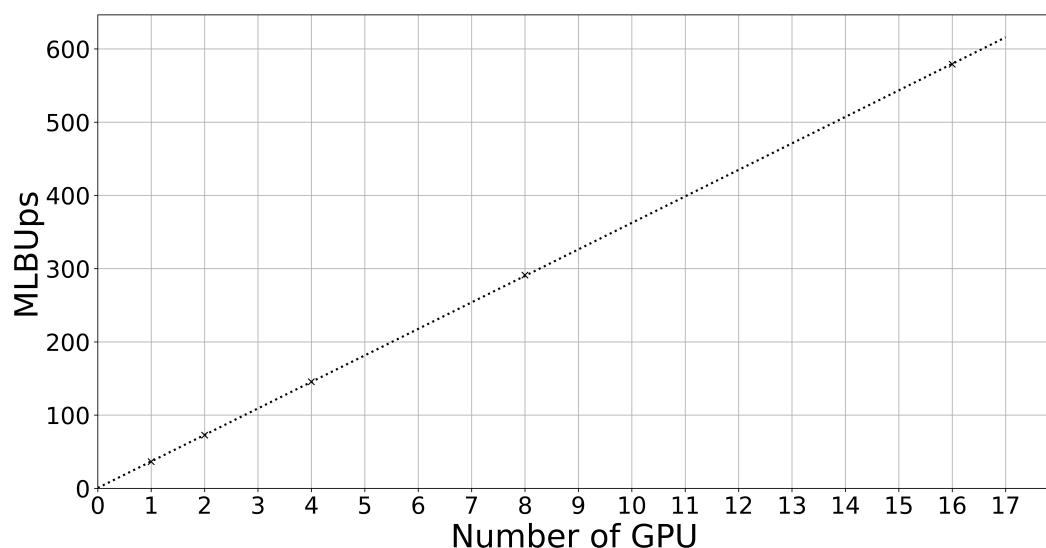
(c) Rysy: 2816 x 440 x 3



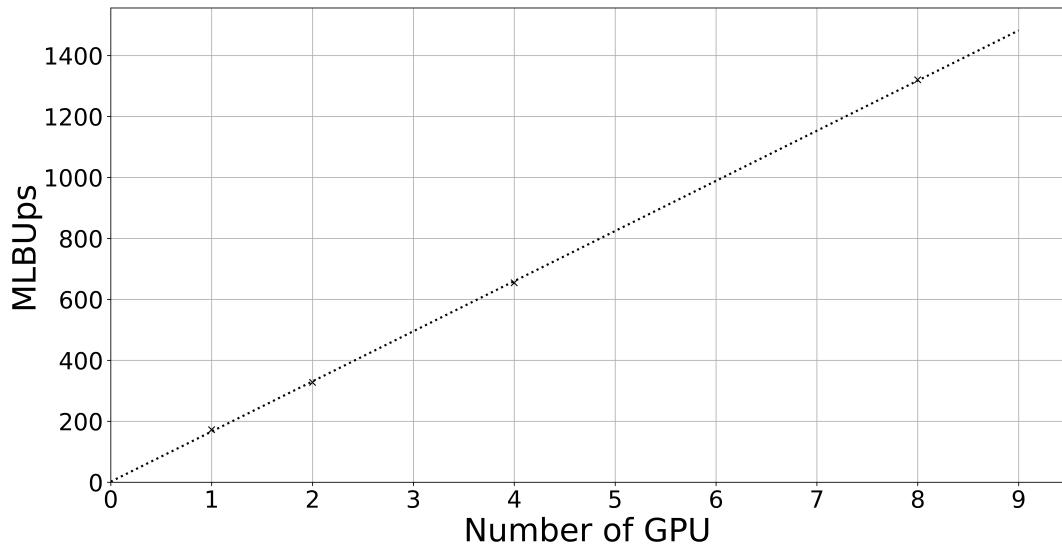
(d) Prometheus: 2816 x 440 x 3



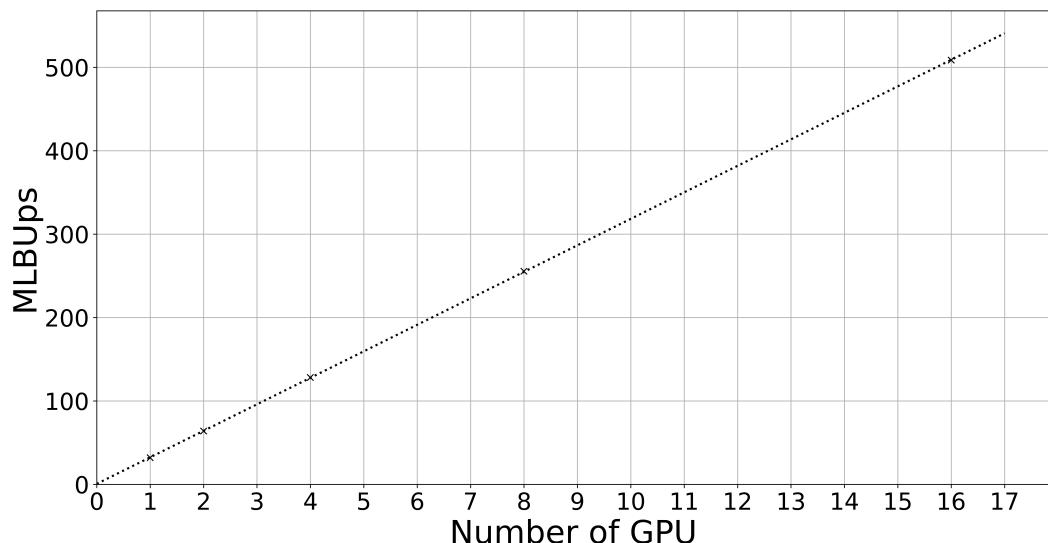
(e) Rysy: 5632 x 220 x 3



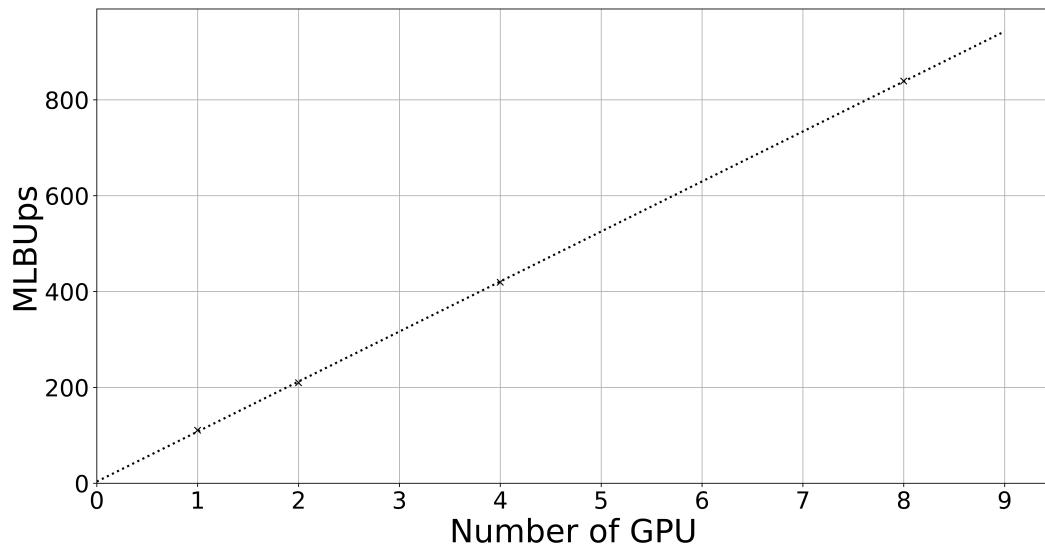
(f) Prometeusz: 5632 x 220 x 3



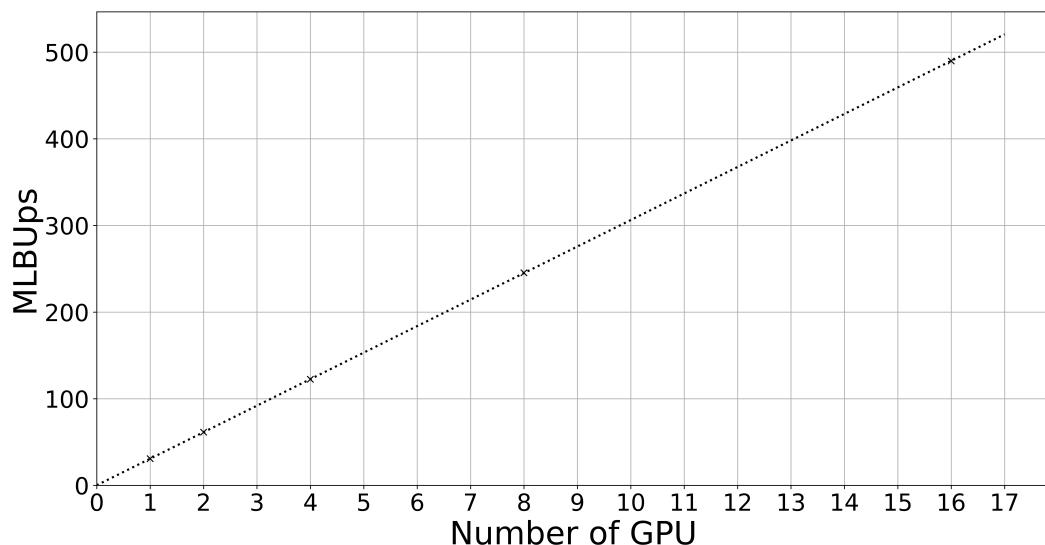
(g) Rysy: 8192 x 320 x 3



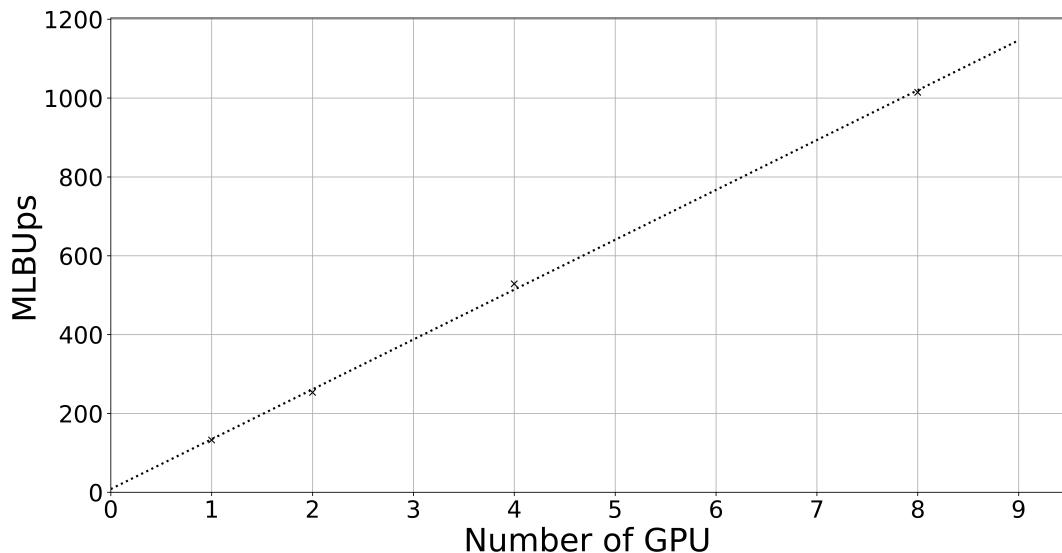
(h) Prometeusz: 8192 x 320 x 3



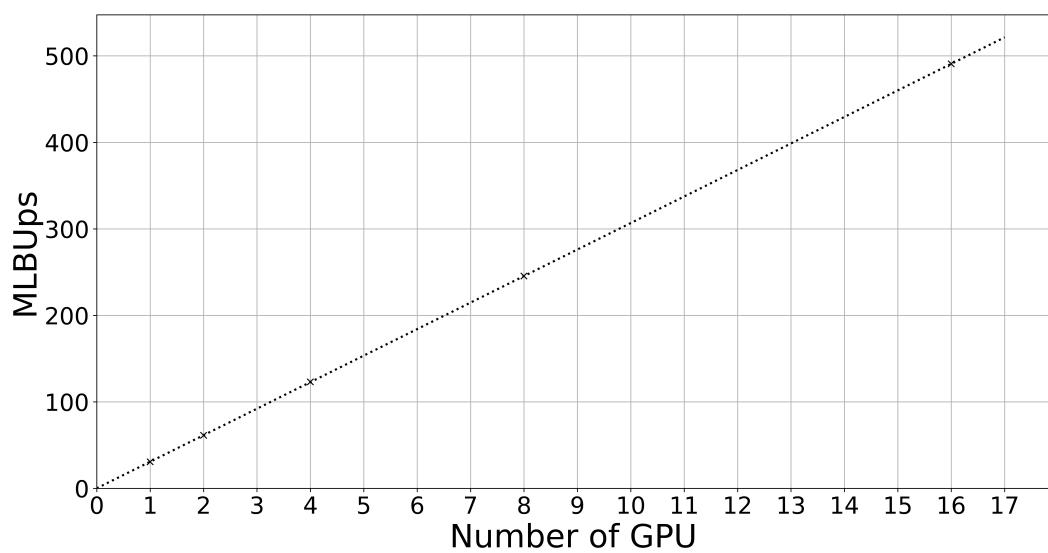
(i) Rysy: 11264 x 110 x 3



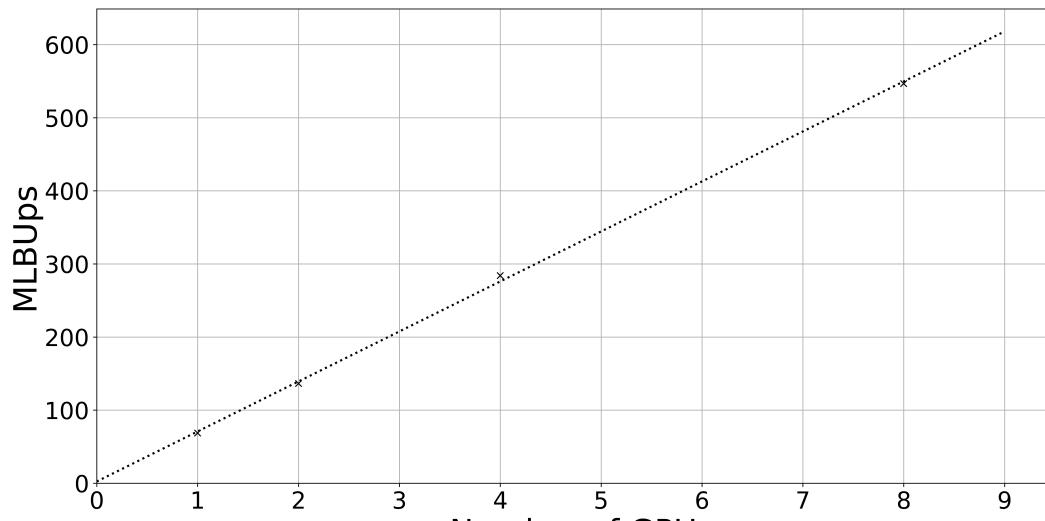
(j) Prometheus: 11264 x 110 x 3



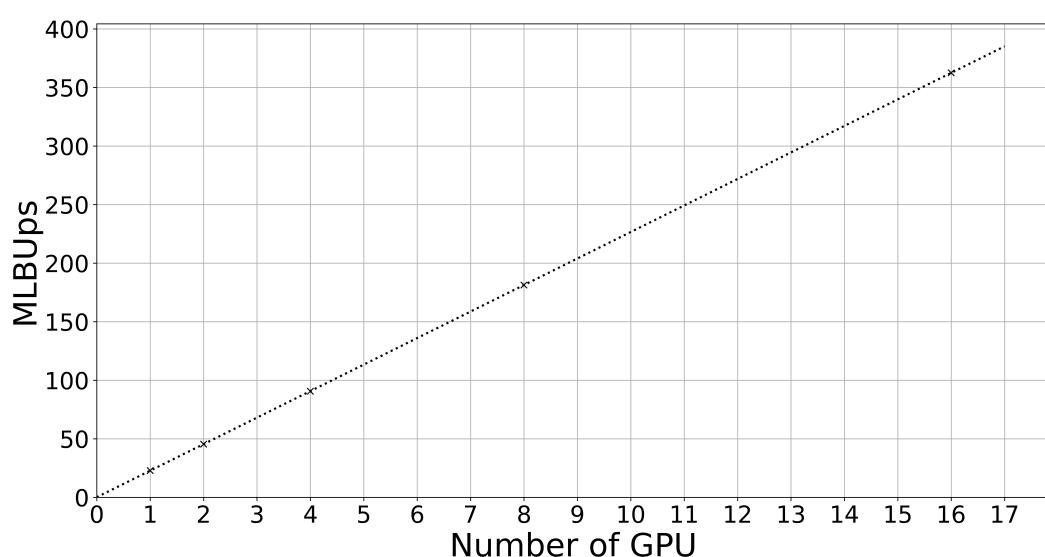
(k) Rysy: 16384 x 160 x 3



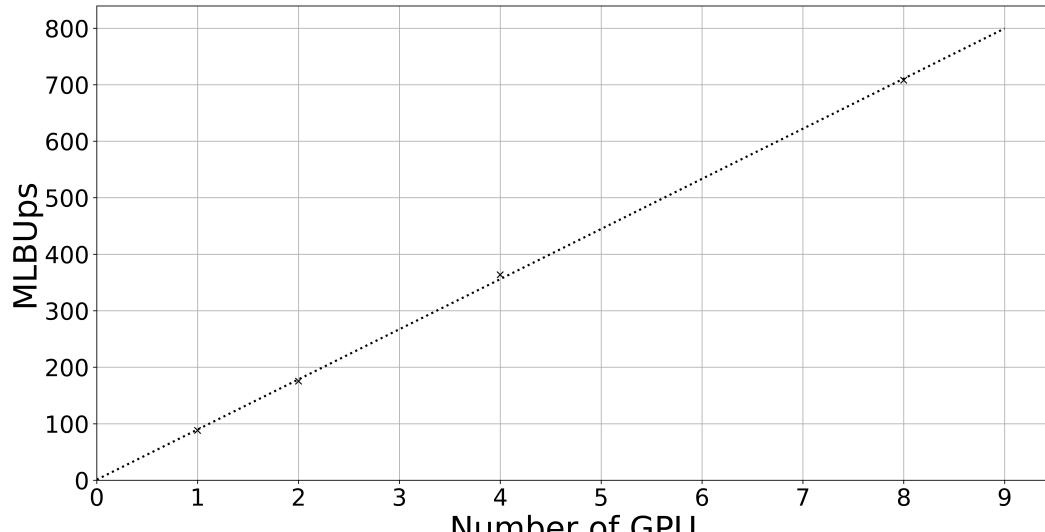
(l) Prometheus: 16384 x 160 x 3



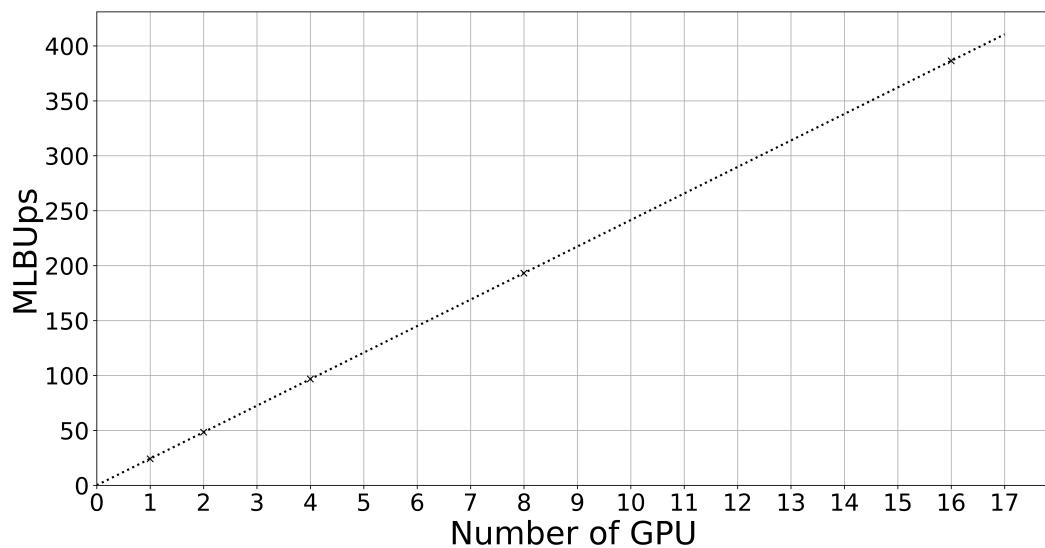
(m) Rysy: 22528 x 55 x 3



(n) Prometheus: 22528 x 55 x 3



(o) Rysy: 32768 x 80 x 3



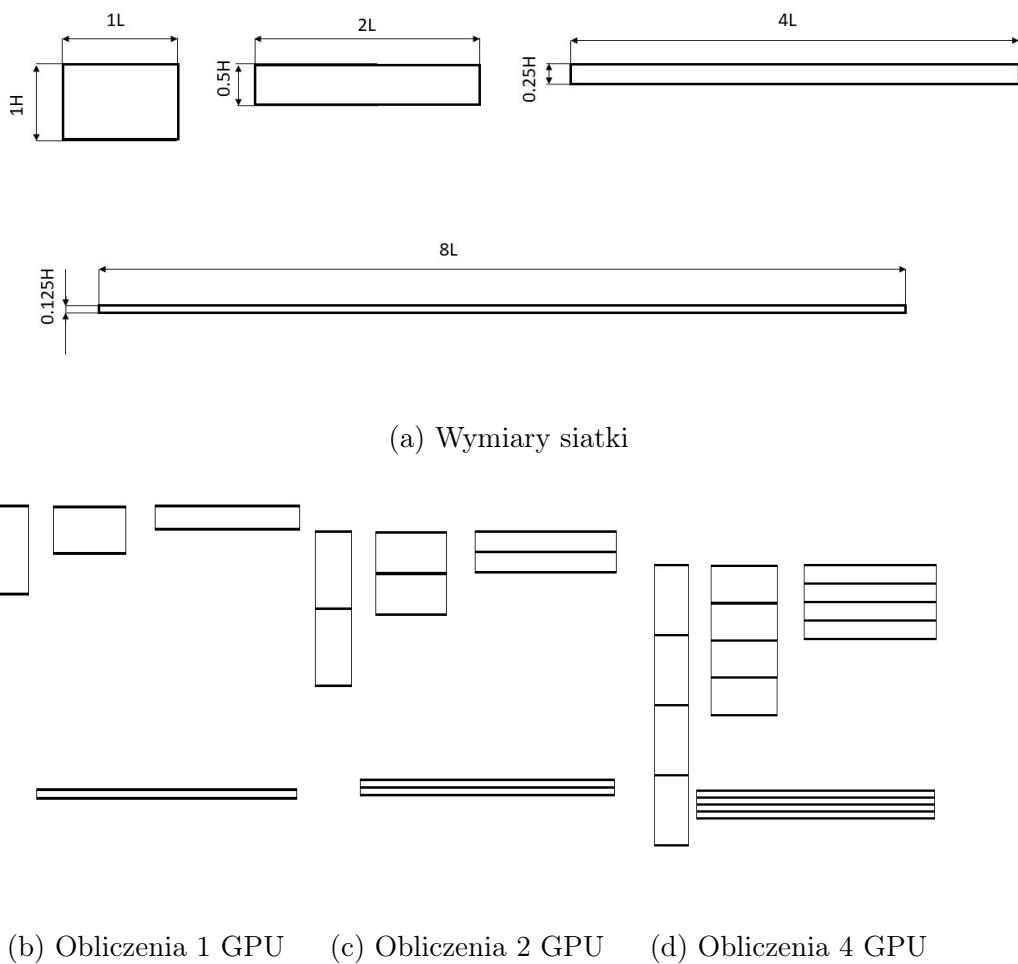
(p) Prometheus: 32768 x 80 x 3

Rysunek 6: Weak scaling

3.1.3 Wykresy przepływu informacji

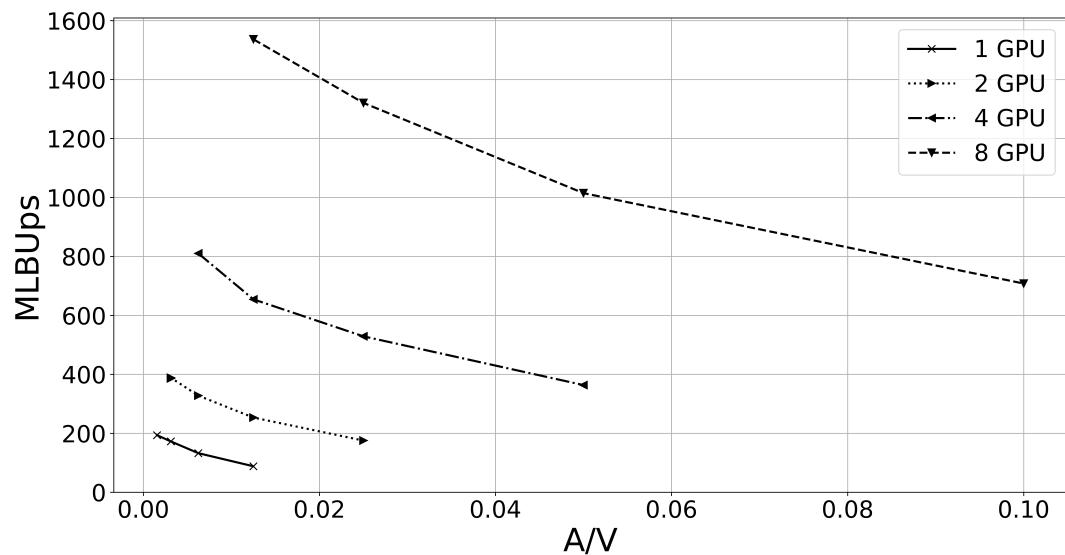
Kluczową rolę w skalowalności kodu obliczeniowego metody Lattice Boltzmann odgrywa stosunek ilości tzw. *ghost cells* (węzły siatki obliczeniowej na granicy obszaru obliczeniowego znajdującego się na karcie) do objętości domeny na kartę.

W celu zbadania wpływu stosunku powierzchni przepływu informacji do objętości domeny obliczeniowej na kartę zastosowano schemat obliczeń widoczny na poniżej ilustracji ([Rysunek 7](#)). Obliczenia na 8 i 16 kartach przeprowadzono analogicznie.

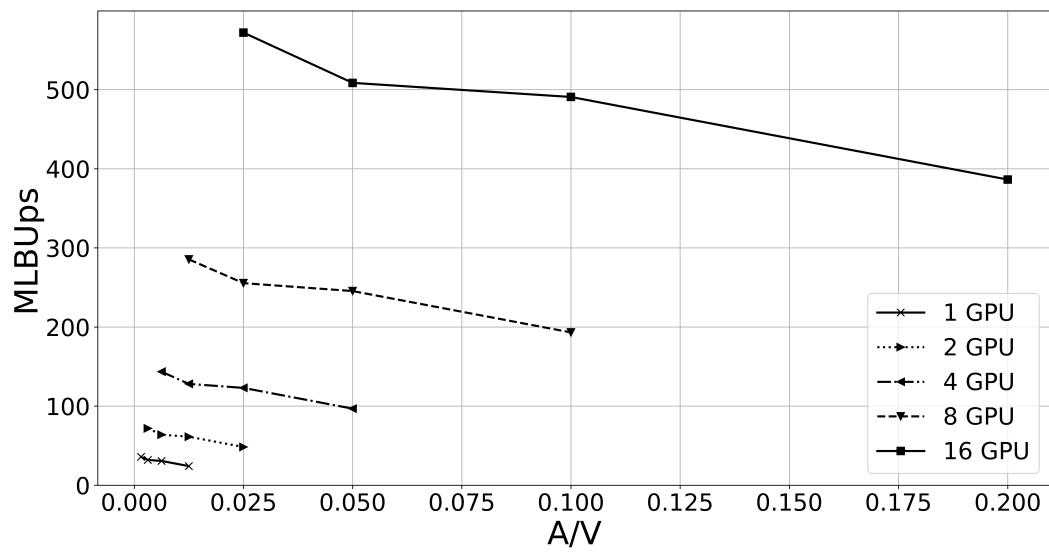


Rysunek 7: Schemat obliczeń

Poniżej przedstawiono wykresy oraz tabele mające na celu zbadać te zależność.

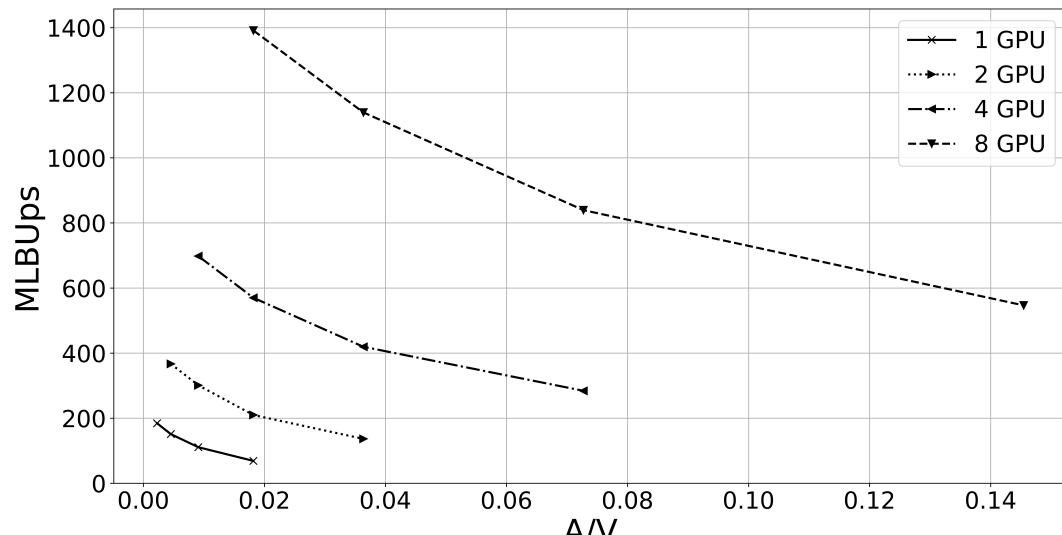


(a) Rysy

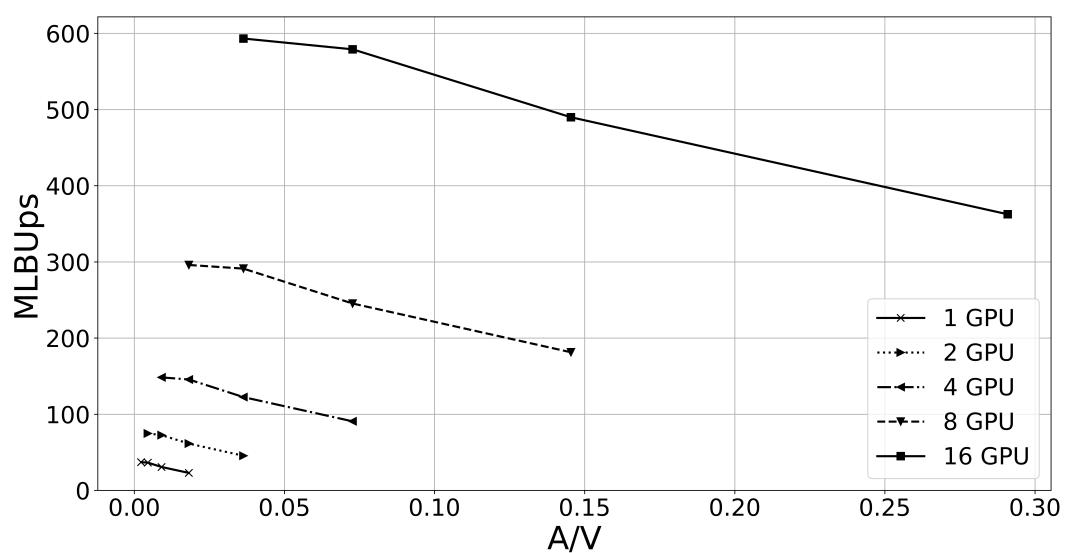


(b) Prometheus

Rysunek 8: A/V plots 10GB load



(a) Rysy



(b) Prometheus

Rysunek 9: A/V plots 5GB load

Tablica 1: Siatka - $\sim 10GB$

A	V	Liczba kart GPU	Speed	Klaster
12288	7864320	1	193.38 35.902	Rysy Prometheus
24576	7864320	1	172.41 32.013	Rysy Prometheus
49152	7864320	1	132.49 30.687	Rysy Prometheus
98304	7864320	1	88.13 24.188	Rysy Prometheus
24576	15728640	2	388.41 71.874	Rysy Prometheus
49152	15728640	2	327.67 63.825	Rysy Prometheus
98304	15728640	2	253.39 61.33	Rysy Prometheus
196608	15728640	2	175.32 48.361	Rysy Prometheus
393216	31457280	4	363.53 96.801	Rysy Prometheus
98304	31457280	4	654.36 128.08	Rysy Prometheus
49152	31457280	4	810.04 143.473	Rysy Prometheus
196608	31457280	4	528.87 123.024	Rysy Prometheus
196608	62914560	8	1320.70 255.415	Rysy Prometheus
-				
Kontynuacja na następnej stronie				

Tablica 1 – kontynuacja z poprzedniej strony

A	V	Liczba kart GPU	Speed	Klaster
786432	62914560	8	707.97 193.158	Rysy Prometheus
98304	62914560	8	1536.52 285.424	Rysy Prometheus
393216	62914560	8	1014.58 245.414	Rysy Prometheus
196608	125829120	16	572.06	Prometheus
393216	125829120	16	508.53	Prometheus
786432	125829120	16	490.70	Prometheus
1572864	125829120	16	386.33	Prometheus

Tablica 2: Siatka - $\sim 5GB$

A	V	Liczba kart GPU	Speed	Klaster
8448	3717120	1	184.309 37.051	Rysy Prometheus
16896	3717121	1	150.881 36.359	Rysy Prometheus
33792	3717122	1	110.693 30.694	Rysy Prometheus
67584	3717123	1	68.683 22.888	Rysy Prometheus
16896	15728640	2	366.83 74.707	Rysy Prometheus
33792	15728640	2	301.008 72.506	Rysy Prometheus
67584	15728640	2	209.717 61.404	Rysy Prometheus
Kontynuacja na następnej stronie				

Tablica 2 – kontynuacja z poprzedniej strony

A	V	Liczba kart GPU	Speed	Klaster
135168	15728640	2	136.349	Rysy
			45.406	Prometheus
33792	31457280	4	697.886	Rysy
			148.29	Prometheus
67584	31457280	4	569.919	Rysy
			145.573	Prometheus
135168	31457280	4	419.359	Rysy
			122.435	Prometheus
270336	31457280	4	284.071	Rysy
			90.612	Prometheus
540672	62914560	8	546.567	Rysy
			181.307	Prometheus
67584	62914560	8	1391.463	Rysy
			295.928	Prometheus
135168	62914560	8	1139.458	Rysy
			291.114	Prometheus
270336	62914560	8	839.171	Rysy
			245.308	Prometheus
135168	59473920	16	593.22	Prometheus
270336	59473920	16	579.05	Prometheus
540672	59473920	16	489.86	Prometheus
1081344	59473920	16	362.48	Prometheus

3.1.4 Wydajność GPU

W poniższych tabelach przedstawiono dane techniczne dotyczące kart GPU, oraz ich wydajności w trakcie wykonywania symulacji na jednej karcie. Efektywność zostało obliczona wg poniższego wzoru ([Rownanie 10](#)).

$$\text{efektywnosc} = \frac{\text{speed}}{\text{Ilosc CUDA CORE * Taktowanie}} \quad (10)$$

Tablica 3: Wydajność GPU - siatka $4096 \times 640 \times 3 \sim 10GB$

	Model Karty	Taktowanie[MHz]	Cuda-Cores	MLUPs	Efektywnosc
Rysy	NVIDIA Tesla V100	1380	5120	193.38	2.74E-05
Prometheus	NVIDIA Tesla K40 XL	928	2880	35.90	1.34E-05

Tablica 4: Wydajność GPU - siatka $2816 \times 440 \times 3 \sim 5GB$

	Model Karty	Taktowanie[MHz]	Cuda-Cores	MLUPs	Efektywnosc
Rysy	NVIDIA Tesla V100	1380	5120	184.31	2.61E-05
Prometheus	NVIDIA Tesla K40 XL	928	2880	37.05	1.39E-05

3.2 Analiza wynikow

Z wykresów silnego skalowania możemy zauważyc, że dla stałego rozmiaru problemu wykres załamuje się po przekroczeniu 4 kart GPU.

Liniowy rozkład punktów na wykresach słabego skalowania jest zgodny z prawem Gustafsona.

Z wykresów [Rysunek 8](#) możemy zauważyc, że przy nieodpowiednim doborze siatki obliczeniowej mimo zwiększonej ilości procesorów działających możemy uzyskać mniejszą szybkość obliczeń. Co więcej na wykresach Speed(A/V) ([Rysunek 8](#), [Rysunek 9](#)) widać, że kształt linii nie zależy od ilości procesorów(kart GPU). Potwierdza to zależność szybkości kodu obliczeniowego opartego o metodą Lattice Boltzmann, od udziału *ghost cells* w całej objętości problemu obliczeniowego. Należy zatem zwracać uwagę na odpowiednią konfigurację siatki obliczeniowej w celu uzyskania optymalnej efektywności kodu obliczeniowego.

Bibliografia

- [1] Martin Geier, Andreas Greiner i Jan G Korvink. “A factorized central moment lattice Boltzmann method”. W: *The European Physical Journal Special Topics* 171.1 (2009), s. 55–61.
- [2] Erlend Magnus Viggen. “The lattice Boltzmann method: Fundamentals and acoustics”. W: (2014).
- [3] Martin Geier i in. “The cumulant lattice Boltzmann equation in three dimensions: Theory and validation”. W: *Computers & Mathematics with Applications* 70.4 (2015), s. 507–547.
- [4] Krüger Timm i in. “The lattice Boltzmann method: principles and practice”. W: *Springer International Publishing AG Switzerland, ISSN* (2016), s. 1868–4521.