Przemysław Chlipała
WfiIS, IS III rok

Sprawozdanie z zajęć „Multicore Programming”

Program symulacji pola wektorowego razem z obliczeniami czasu wykonywania kernela dostępny jest na githubie: https://github.com/przennek/multicore_programming.git

1. Hardware:

Wszystkie dane użyte do napisania tego sprawozdania zostały wygenerowane na procesorze Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz i karcie graficznej GeForce GTX 660M:

```
Device 0: "GeForce GTX 660M"
  CUDA Driver Version / Runtime Version          7.5 / 7.5
  CUDA Capability Major/Minor version number:    3.0
  Total amount of global memory:                 2.00 MBytes (2147287040 bytes)
  GPU Clock rate:                                950 MHz (0.95 GHz)
  Memory Clock rate:                             2500 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 262144 bytes
  Max Texture Dimension Size (x,y,z)             1D=(65536), 2D=(65536,65536), 3D=(4096,4096,4096)
  Max Layered Texture Size (dim) x layers        1D=(16384) x 2048, 2D=(16384,16384) x 2048
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:     2147483647 x 65535 x 65535
  Maximum memory pitch:                          2147483647 bytes
```

Rys 1. Rezultat programu checkDeviceInfor.cu

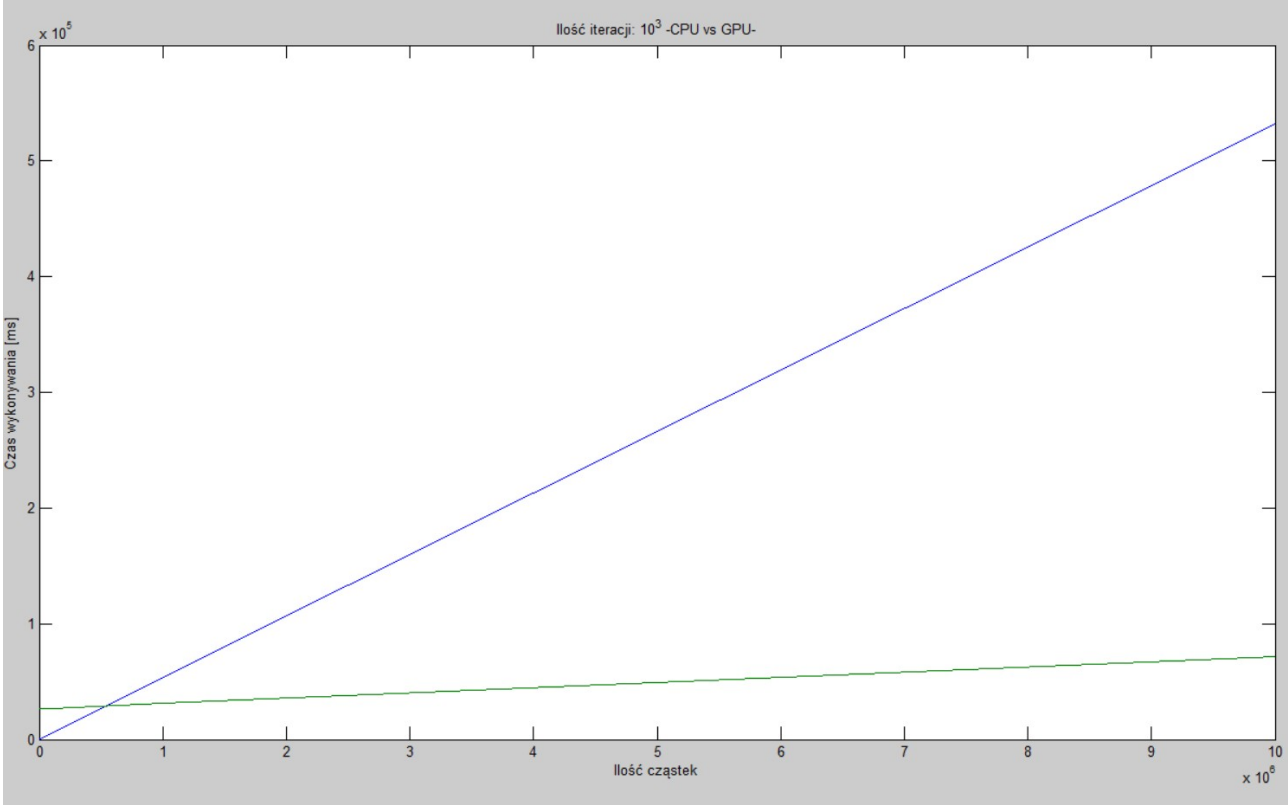
2. Operacje na wektorach

W tym zadaniu miałem napisać kod który wykona operacje na dwóch wektorach przy użyciu CPU oraz GPU i zmierzy czas wykonywania. Postanowiłem pójść krok dalej i zaimplementować operacje potrzebne do symulacji w polu wektorowym i zmierzyć ich czasy wykonywania. Na gpu losuje wektor na podstawie którego będę zmieniał zwrot ruchu cząstek, wyliczam składowe prędkości w danym punkcie i składowe przemieszczenia, na koniec sumuje przemieszczenie w tablicy z przebytym dystansem. Na cpu robię podobnie, za wyjątkiem losowego wektora, ponieważ mogę wygodnie losować liczbę odpowiedzialną za zmianę zwrotu podczas wyliczania składowych prędkości. Złożoność obliczeniowa powinna być identyczna.

Wielkość wektorów zmienia się od 10^2 do 10^8, a ilość iteracji symulacji od 10^3 do 10^5, analizując wyniki zamieszczone poniżej oraz wykresy uzyskane na ich podstawie można dojść do niezbyt zaskakującego wniosku, że GPU zdecydowanie lepiej sobie radzi z większymi wektorami a ilość iteracji nie ma wielkiego wpływu na czas wykonywania kodu dla GPU. Pozytywnie zaskakuje mnie za różnica w czasach wykonywania kodu dla GPU dla konkretnej ilości iteracji. Niezależnie od wielkości wektora operacje wykonywane na GPU nie odbiegają zbytnio jeżeli chodzi o czas wykonywania np. dla 10^3 iteracji, operacje na wektorze o wielkości 10^2 a 10^7 różni się o 3450 ms gdzie dla CPU jest to 53404 ms.

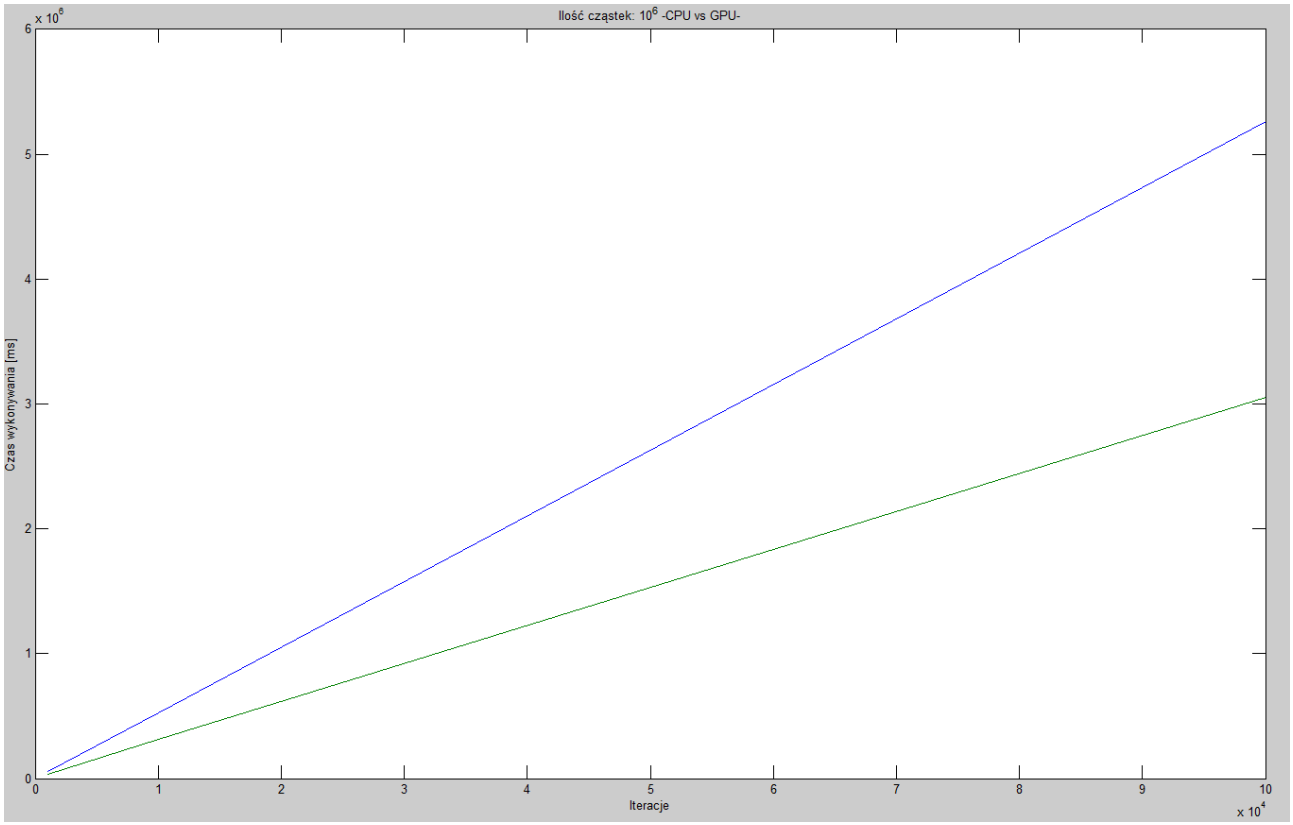
Iteracje	Ilość cząstek	Czas wykonania [ms]				Wygrywa
		CPU	GPU	GPU kopiowanie	Suma GPU	
1.00E+03	1.00E+02	7.00	27505.00	0.07	27505.07	CPU
1.00E+03	1.00E+03	52.00	27033.40	0.24	27033.64	CPU
1.00E+03	1.00E+04	522.46	27065.50	0.37	27065.87	CPU
1.00E+03	1.00E+05	5252.11	27081.70	1.51	27083.21	CPU
1.00E+03	1.00E+06	53411.50	30942.70	6.96	30949.66	GPU
1.00E+03	1.00E+07	532193.00	71314.00	56.86	71370.86	GPU
1.00E+04	1.00E+02	52.42	273840.00	0.12	273840.12	CPU
1.00E+04	1.00E+03	535.73	272081.00	0.17	272081.17	CPU
1.00E+04	1.00E+04	5223.08	270752.00	0.22	270752.22	CPU
1.00E+04	1.00E+05	52330.90	270590.00	1.40	270591.40	CPU
1.00E+04	1.00E+06	527746.00	311180.00	7.48	311187.48	GPU
1.00E+04	1.00E+07	5334100.00	720354.00	61.01	720415.01	GPU
1.00E+05	1.00E+02	511.46	2765510.00	0.14	2765510.14	CPU
1.00E+05	1.00E+03	5254.25	2714110.00	0.15	2714110.15	CPU
1.00E+05	1.00E+04	52593.60	2709760.00	0.21	2709760.21	CPU
1.00E+05	1.00E+05	521572.00	2707100.00	1.33	2707101.33	CPU
1.00E+05	1.00E+06	5262550.00	3049750.00	7.21	3049757.21	GPU

Rys 2. Rezultat programu vector_field_simulation.cu



Rys 3. Ilość czastek a czas wykonywania [ms] dla GPU (na zielono) i CPU (niebiesko), dla 10^3 *iteracji.*

Z powodu mojego doboru wielkości wektorów (a raczej wykładniczego przyrostu ich wielkości), mogłoby się wydawać na pierwszy rzut oka (patrząc tylko na tabelkę), że CPU wygrywa w większości przypadków jeżeli chodzi o obliczenia, powyższy wykres jednak pozbawia złudzeń. Dodatkowo mimo tego co powiedziałem powyżej na temat niskiego wpływu ilości iteracji na czas wykonywania programu, to i tak w tym wypadku GPU wypada lepiej co widać na poniższym wykresie.



Rys 4. Ilość iteracji a czas wykonywania [ms] dla GPU (na zielono) i CPU (niebiesko), dla 10^6 *cząstek.*

Dla pozostałych przypadków uzyskane wykresy powinny wyglądać analogicznie. Do mierzenia czasu wykonywania wykorzystywałem zdarzenia cudy i metodę cudaEventElapsedTime. Dało mi to dokładne wyniki i wolne od błędów od jakich poprzednia wersja tego sprawozdania nie zdołała się ustrzec (wykorzystywałem tam obiekt clock). Moje wnioski ulegają więc zmianie.

Wnioski:

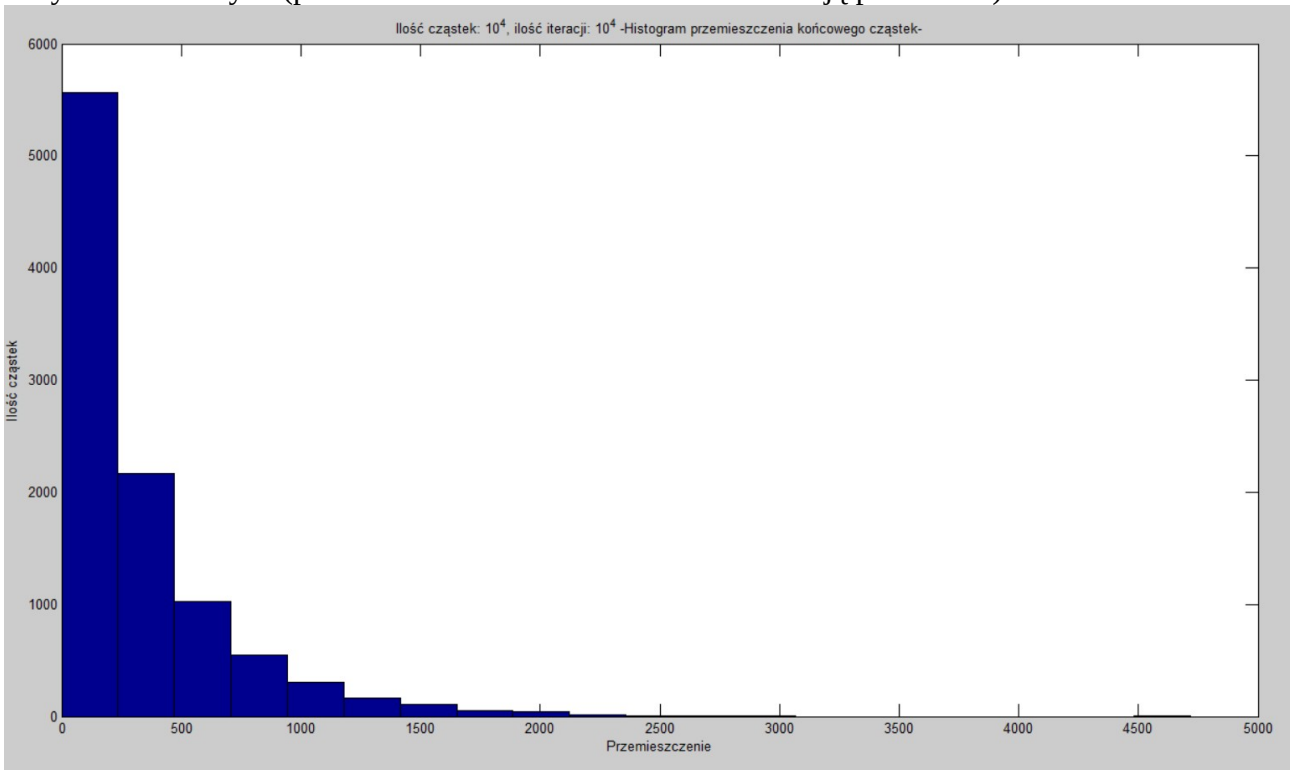
- Czas kopiowania dużych ilości danych z hosta na GPU nie jest tak ważny jak napisałem w poprzedniej wersji sprawozdania a jest pomijalnie mały dla obliczeń tego kalibru.
- Dla wektorów o ilości elementów rzędu 10^6 i więcej, Cuda deklasyfikuje obliczenia na CPU jeżeli chodzi o czas wykonywania.

Spostrzeżenia:

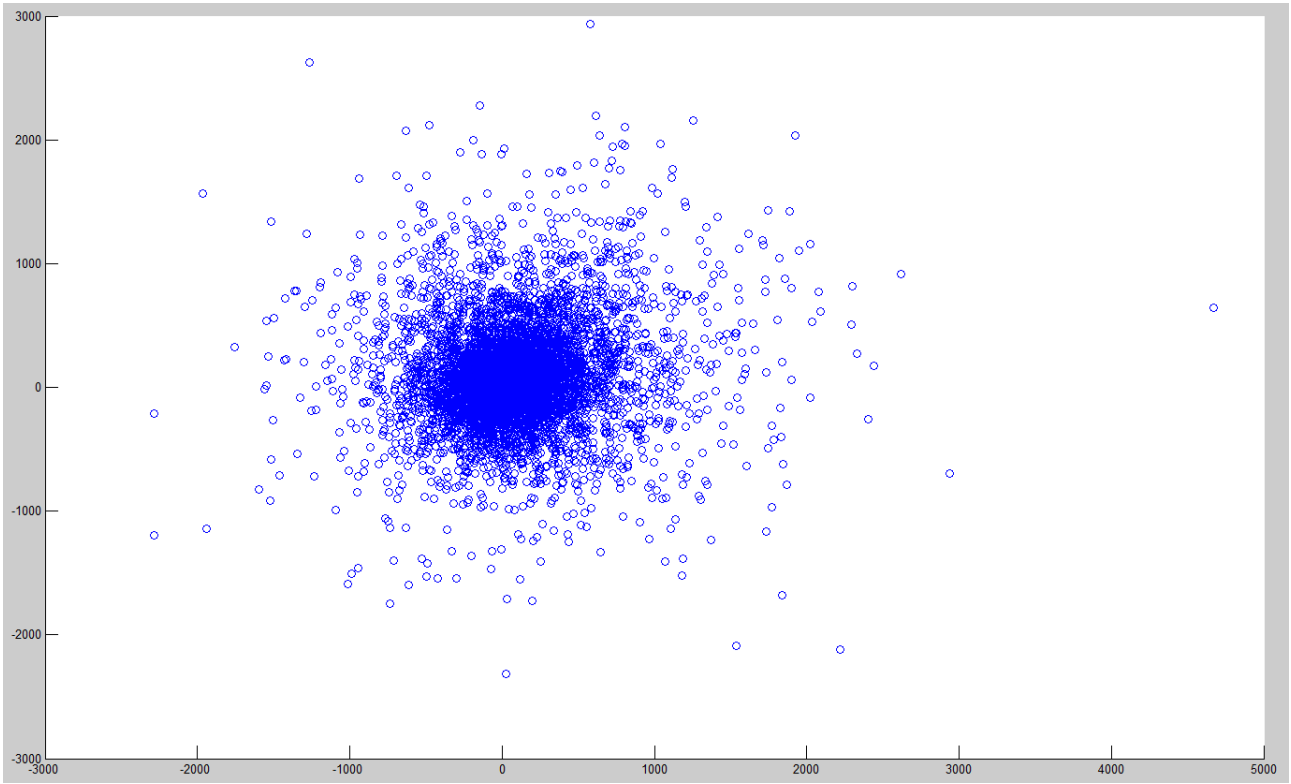
- Program, który dokonał tych symulacji działał nieprzerwanie przez 9 godzin, jednak po zsumowaniu czas wykonywania obliczeń na CPU i GPU dał mi lekko nieco ponad 3,5 godziny. Podejrzewam, że brakujące 4,5 godziny zostały przeznaczone na policzenie skrajnego przypadku dla 10^5 iteracji i 10^8 wielkości wektorów, które zostały przerwane.

3. Symulacja pola wektorowego

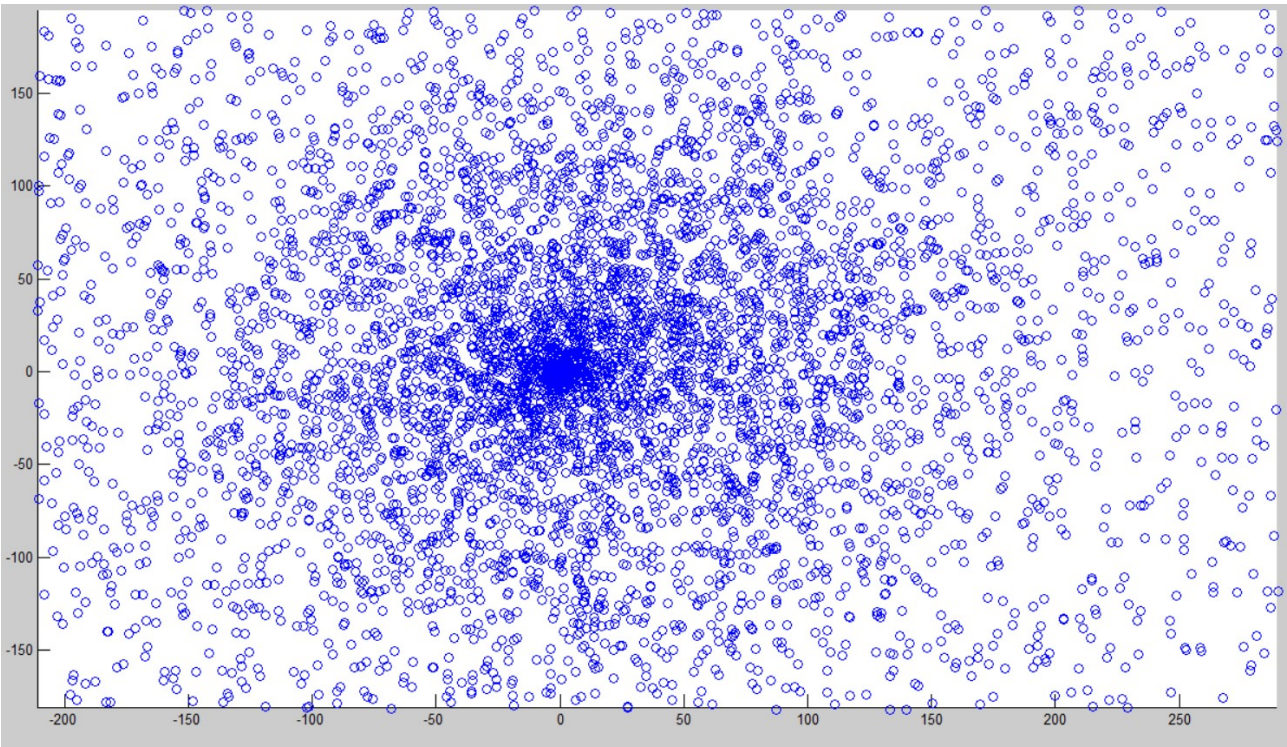
W tej części skupię się na opracowaniu wyników symulacji, której czas wykonywania analizowałem w punkcie 2. Wszystkie dane uzyskane z symulacji zapisywane są w postaci załążka skryptu matlabowego a potem opracowywane w matlabie. Niestety wyniki opracowuję dla przypadków „środkowych” ponieważ mój laptop (i matlab) ma problemy z przetworzeniem takich dużych ilości danych (pliki dla wektorów o wielkości 10^7 mają po 500 mb).



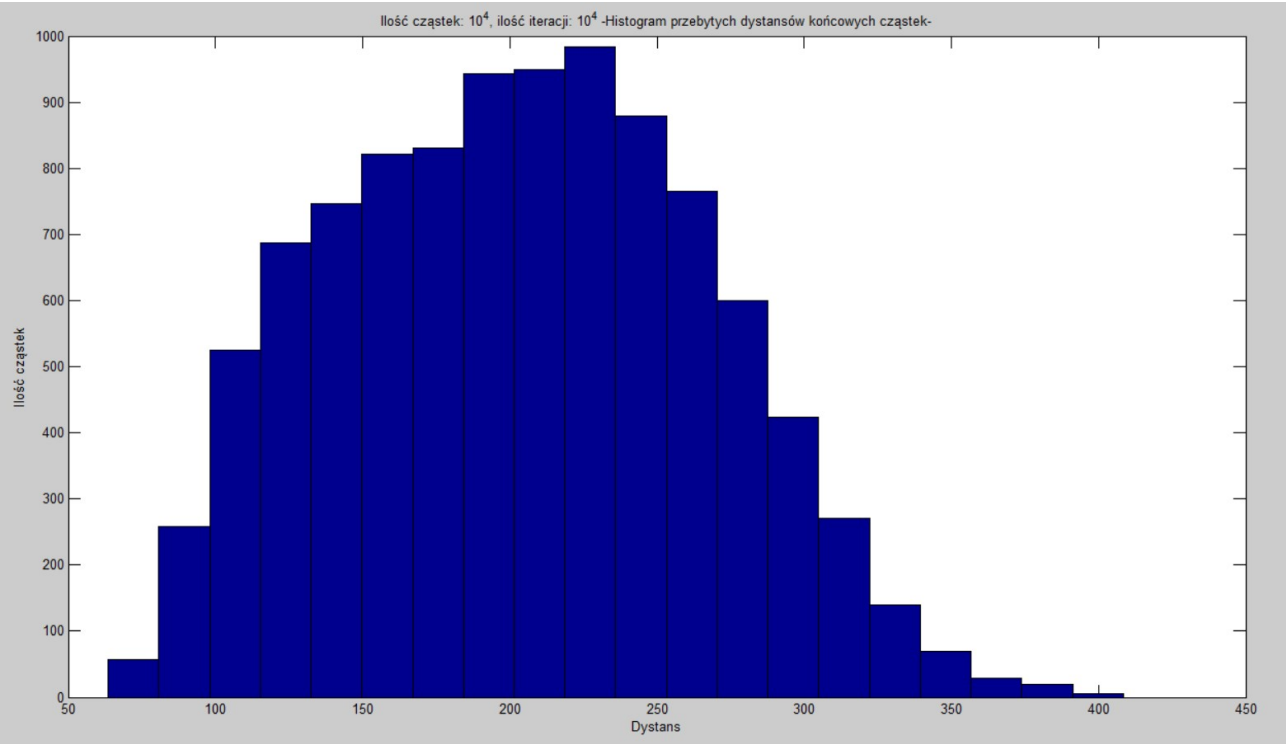
Rys 5. Histogram wartości przemieszczenia dla 10^4 *ilości cząstek i* 10^4 *iteracji (w metrach).*



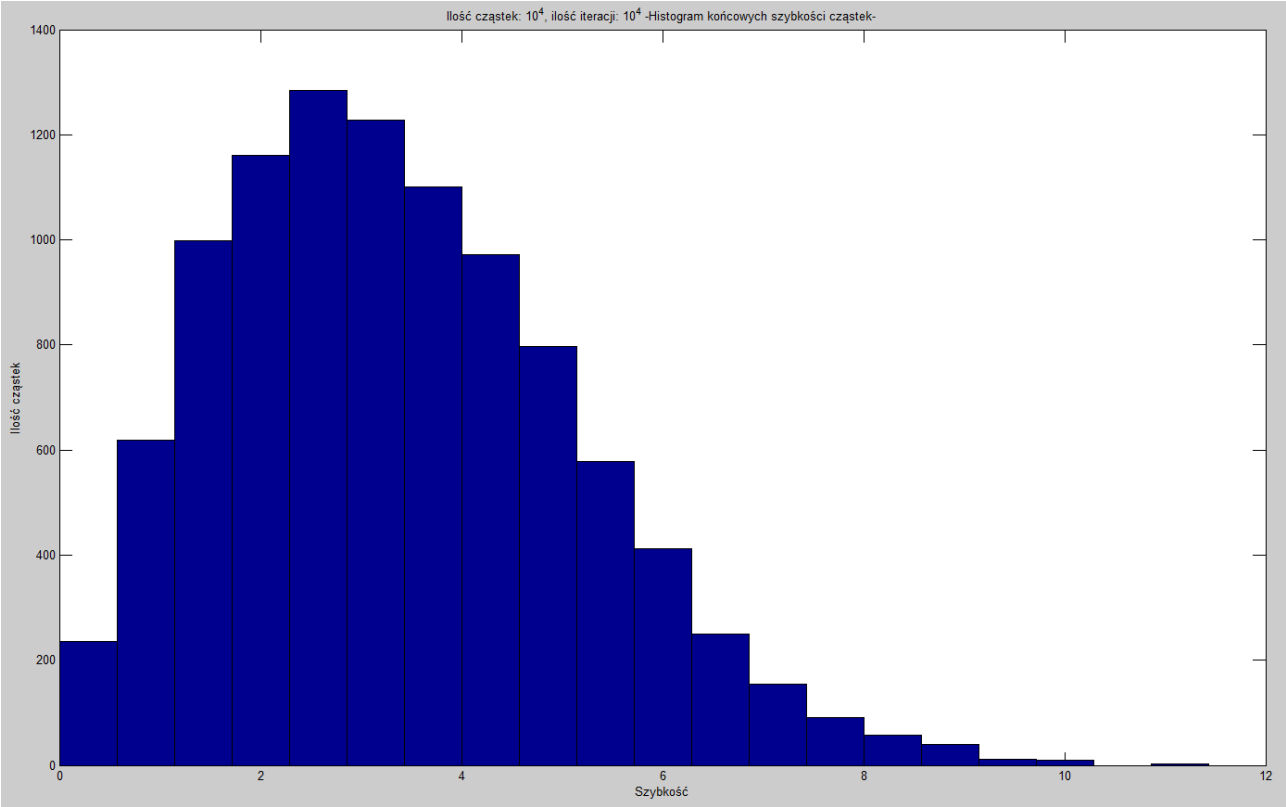
Rys 6. Ostateczne położenie dla 10^4 *ilości cząstek i* 10^4 *iteracji.*



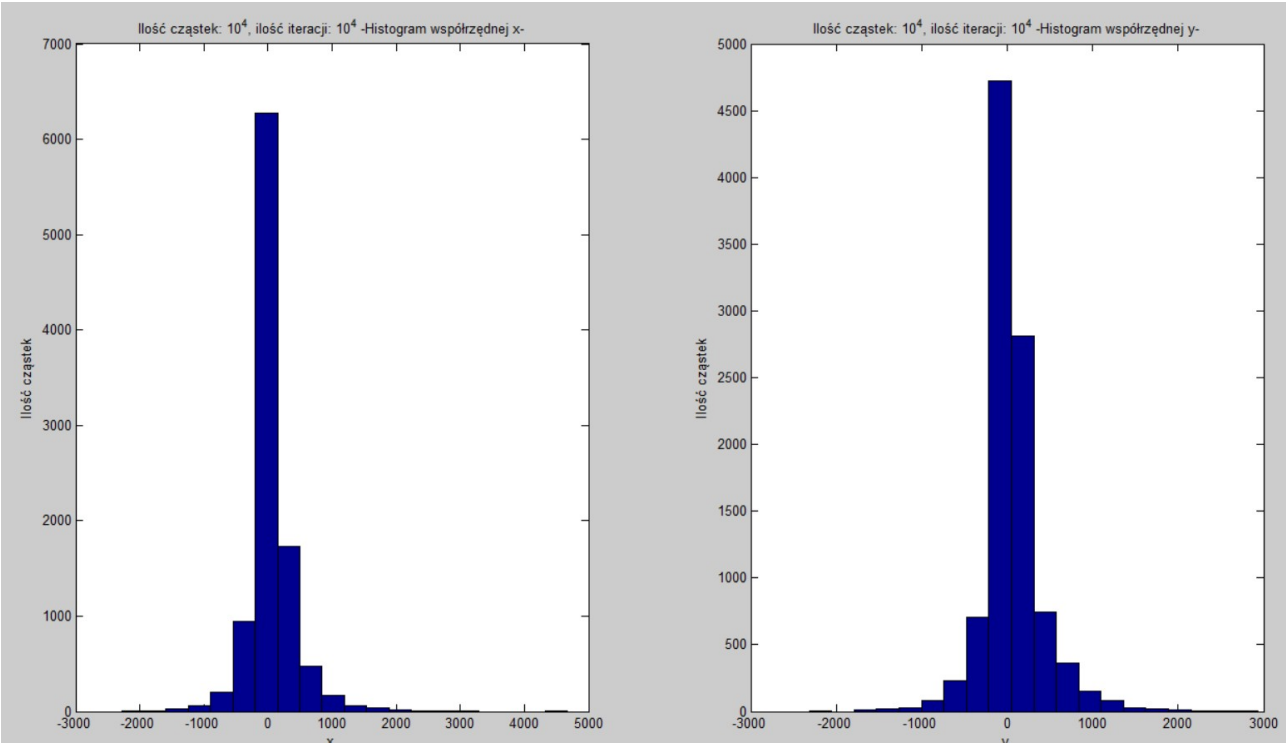
Rys 7. Ostateczne położenie dla 10^4 *ilości cząstek i* 10^4 *iteracji. (przybliżenie)*



Rys 8. Histogram przebytych dystansów dla 10^4 *ilości cząstek i* 10^4 *iteracji. Dystans podaję w metrach.*



Rys 9. Histogram końcowych szybkości dla 10^4 *ilości cząstek i* 10^4 *iteracji. Szybkość podaję w [m/s], gdzie za sekundę przyjmuję krok w iteracji.*



Rys 10. Histogram współrzędnej x i współrzędnej y dla 10^4 *ilości cząstek i* 10^4 *iteracji.*

Wnioski:

- Uzyskane wyniki są zgodne z oczekiwanymi, co zostało uzyskane poprzez zwiększenie ilości cząstek i ilości iteracji w symulacji (względem poprzedniej).
- Korzystanie z cudowskiej funkcji mierzenia czasu daje bardziej wiarygodne i dokładne wyniki niż clock.
- Jako, że przesył danych z karty na hosta zawsze był równy 0, został pominięty w tabeli z wynikami, nie jestem przekonany co do prawidłowości tych pomiarów natomiast jestem przekonany, że suma przesyłu, obliczeń i przesyłu z powrotem jest prawidłowa.

Spostrzeżenia:

- Dla opisanej symulacji cząstki przebyły w sumie 2037700 metrów w ok. 1 dzień i 4 godziny.
- Najbardziej oddalona cząstka od punktu startowego znajduje się 4716 metrów od punktu startu.
- Najszybsza cząstka w momencie końca symulacji poruszała się z v = 11.4343 m/s.

Przykład w jaki sposób opracowałem te wykresy znajduje się w katalogu ze skryptami matlabowymi w pliku oprac105i105w.m, plik ten generuje w/w wykresy. Jest to też najwygodniejsza forma zapoznania się z nimi jednak potrzeba do tego dość mocnej maszyny bo jest dość pamięciożerny. W katalogu ze skryptem znajdują się też wyniki innych pomiarów jednak z powodu rozmiaru niektórych z tych plików musiałem wyciąć te największe (by wygodnie możnabyło je sobie pobrać z githuba).