06.02.2024 r.

Bartłomiej Baur

# Giga-Argon Crystal Simulator

Raport z projektu zaliczeniowego na przedmiot
*High Performance Computing in Scientific Applications*

## Wstęp

*Giga-Argon Crystal Simulator* (GACS) to program symulujący kryształ argonu z wykorzystaniem klasycznej dynamiki molekularnej. Program został napisany w języku C i wykorzystuje platformę CUDA do przeprowadzania równoległych obliczeń na karcie graficznej.

W niniejszym dokumencie wyjaśniono, jak skompilować i uruchomić program oraz pokrótce opisano działanie programu.

## Instrukcja obsługi

### Kompilacja programu

GACS został przygotowany do pracy na superkomputerze DWARF znajdującym się na Wydziale Fizyki Politechniki Warszawskiej. Po zalogowaniu się na DWARF-ie przez SSH należy przenieść się na ssh68.

*» ssh68*

Kompilacja programu odbywa się za pomocą polecenia:

*» nvcc argon.cu CFG/cfg_parse.c -o argon -lm -lcudart --expt-relaxed-constexpr*

Gdzie *argon.cu* to kod programu, *CFG/cfg_parse.c* to konieczna biblioteka. Pojawi się plik wykonywalny o nazwie *argon*.

### Obsługa programu

Aby uruchomić symulację, należy posłużyć się poleceniem:

*» ./argon <plik konfiguracyjny> <plik wyjściowy parametrów> <plik wyjściowy położeń>*

Program do działania potrzebuje trzech plików:

- *<plik konfiguracyjny>* - plik zawiera informacje o parametrach niezbędnych do prawidłowego funkcjonowania programu.

- *<plik wyjściowy parametrów>* – plik, w którym w równych odstępach czasu symulacji zapisywane będą makroskopowe parametry układu: temperatura *T*, ciśnienie *P*, energia całkowita *E*, energia kinetyczna *Ek*, energia potencjalna *V*.
- *<plik wyjściowy położeń>* – plik, w którym w równych odstępach czasu symulacji zapisywane będą położenia atomów. Plik jest sformatowany tak, że wynik symulacji w postaci animacji można obejrzeć w programie JMol[1].

# Plik konfiguracyjny

Aby program zadziałał poprawnie niezbędnym jest wskazanie mu pliku konfiguracyjnego. Przykładowy plik konfiguracyjny ma następującą postać:

```
n=3
m=40
e=1
R=0.38
f=1e4
a=0.38
T_0=10
tau=1e-3
S=10000
S_out=10
S_xyz=100
```

Oto, co oznaczają kolejne parametry:

| | |
|---|---|
| *n* | Liczba atomów w krawędzi kryształu. $n^3$ to całkowita liczba atomów w symulacji. |
| *m* | Masa atomu. |
| *e* | Stała dielektryczna. |
| *R* | Rozmiar atomu – optymalny dystans między atomami dla siły Van der Vaalsa. |
| *f* | Współczynnik sprężystości dla oddziaływań atomów ze ściankami. |
| *a* | Stała sieci krystalicznej. |
| *T_0* | Początkowa temperatura układu. |
| *tau* | Długość kroku czasowego. |
| *S* | Liczba kroków czasowych. |
| *S_out* | Liczba kroków, co ile parametry symulacji będą zapisane. |
| *S_xyz* | Liczba kroków, co ile położenia atomów będą zapisane. |

1 Link do programu: https://jmol.sourceforge.net/.

# Zasada działania programu

Model teoretyczny wykorzystywany przez program jest opisany w pliku *KMS_lab.pdf* – instrukcji do laboratoriów z przedmiotu KMS – który został umieszczony w repozytorium programu. W tym rozdziale skupimy się na wykorzystaniu biblioteki CUDA.

## Podstawowe elementy symulacji:

Najważniejszymi z punktu widzenia symulacji przechowywanymi wielkościami są położenia **r** *oraz* pędy ***p*** atomów oraz działające na nie siły ***F***. Wszystkie one w programie przechowywane są jako tablice o rozmiarze 3*N*, gdzie *N* to liczba symulowanych atomów. Tablice pędów oraz sił zostały dynamicznie zaalokowane na GPU z użyciem polecenia cudaMalloc, tablica położeń zaś korzysta z tzw. *CUDA Unified Memory* (cudaMallocManaged), która w inteligentny sposób zarządza kopiowaniem danych między GPU a CPU. W podobny sposób zostały zaalokowane pojedyncze zmienne dla makroskopowych parametrów symulowanego układu: temperatura *T,* ciśnienie *P,* energia całkowita *E,* energia kinetyczna *Ek* oraz energia potencjalna *V*. *Unified Memory* znacznie ułatwia zapisywanie powyższych danych do plików, chociaż obliczane są one w GPU.

Działanie programu składa się się z dwóch głównych etapów: inicjalizacja oraz główna pętla symulacji. Wszystkie obliczenia wykonywane są bezpośrednio na GPU. CPU dba jedynie o poprawą kolejność wykonywania operacji oraz zapis wyników do programu.

## Inicjalizacja programu

Inicjalizacja programu wykonywana jest za pomocą kerneli *r_init* oraz *p_init*. Są to proste operacje wykonywane tylko raz, dlatego wykonywane są jednowątkowo. Ich rolą jest wypełnienie przechowywanych w GPU tablic ***r*** oraz ***p*** liczbami.

## Główna pętla symulacji

Każda pętla symulacji wykonuje po sobie kolejne kernele: *update_rp, calc_VFP, update_p, calc_EKET.* W każdym z nich operacje matematyczne wykonywane są równolegle z wykorzystaniem wielu wątków w GPU, jednak każdy kolejny kernel musi być wykonywany po całkowitym ukończeniu pracy poprzedniego. Stąd pomiędzy kolejnymi kernelami znajduje się polecenie *cudaDeviceSynchronize*, przy którym CPU czeka na zakończenie obliczeń w GPU.

### calc_VFP(double* r, double* F, double* V, double* P)

Najważniejszym a zarazem najcięższym elementem symulacji jest wyliczanie sił van der Waalsa spajających atomy w jeden kryształ. Operacja ta wymaga szczególnie dużo mocy obliczeniowej należy bowiem uwzględnić wkład od każdej możliwej pary atomów w systemie Oznacza to, że liczba sił składowych do policzenia wynosi 0,5**N**(*N*-1), gdzie *N* to liczba atomów w krysztale. *N*a szczęście wszystkie siły składowe można z łatwością policzyć równolegle. Drugą, prostszą w obliczeniu siłą jest siła elastycznego odbicia się cząstek od ścianek naczynia, w którym znajduje się kryształ. Tu trzeba policzyć jedynie *N* trójwymiarowych sił.

Kernel calc_VFP wykonuje jednocześnie obliczenia sił van der Waalsa oraz sił odbić elastycznych. Do tego celu wykorzystuje się *macierz wątków* obliczeniowych na GPU. Na przykład w hipotetycznej symulacji 5 atomów, macierz wątków wyglądałaby jak poniżej:

$$\begin{bmatrix} E & E & E & E & E \\ W & 0 & 0 & 0 & 0 \\ W & W & 0 & 0 & 0 \\ W & W & W & 0 & 0 \\ W & W & W & W & 0 \end{bmatrix}$$

Znaki tworzące macierz określają rolę, jaką pełni dany wątek:

- Litera $W$ – oznacza, że wątek liczy siłę van der Waalsa działającą pomiędzy $i$-tym oraz $j$-tym atomem, gdzie $i, j$ to indeksy powyższej macierzy.
- Litera $E$ – oznacza, że wątek liczy siłę elastycznego odbicia się $i$-tego atomu od ścianek, gdzie $i$ to kolumna powyższej macierzy.
- Liczba 0 – oznacza, że wątek, chociaż uruchomiony, nie wykonuje żadnej faktycznej pracy.

Do policzenia sił van der Waalsa wykorzystuje się tylko połowę macierzy dzięki 3. zasadzie dynamiki Newtona – macierz elementarnych oddziaływań atom-atom jest antysymetryczna. Z drugiej strony powołanie nieco większej liczby wątków było łatwiejsze do oprogramowania. Jest to miejsce, gdzie można byłoby wprowadzić poprawki.

## update_p(double* p) i inne kernele

Pozostałe kernele były znacznie prostsze w optymalizacji. W każdym przypadku ich zasadniczą część stanowiła jedna pętla, która wykonuje te same działania dla każdego elementu wektora. Optymalizacja obliczeń polegała jednego wątku na każdy element wektora, aby każdy z nich wykonał tylko jedno działanie.

Czasem wyniki obliczeń w kolejnych iteracjach należy ze sobą zsumować. Robi się to również w przypadku liczenia sił, ale najelegantszymi przykładami są energia kinetyczna i potencjalna do policzenia. Aby sumować wyniki obliczeń poszczególnych obliczeń i uniknąć „racing conditions” zastosowano funkcję atomicAdd.

## Profilowanie kodu:

Profilowanie zostało wykonane dla układu 27 atomów. Do tego celu wykorzystano narzędzie nvprof. Rezultaty przedstawiono poniżej:

```
==7029== Profiling application: ./argon parameters.config energies.out xyz.out
==7029== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   97.58%  1.89520s     10001  189.50us  146.46us  763.38us  calc_VFP(double*, double*,
double*, double*)
                   0.96%  18.616ms      1001  18.597us  17.856us  23.231us  calc_EkET(double*, double*,
double*, double*, double*)
                   0.82%  15.907ms     10000  1.5900us  1.5040us  13.664us  update_rp(double*, double*,
double*)
                   0.63%  12.173ms     10000  1.2170us  1.1510us  2.6560us  update_p(double*, double*)
                   0.01%  160.13us       101  1.5850us  1.4720us  2.0160us  [CUDA memcpy DtoH]
                   0.00%  45.311us         1  45.311us  45.311us  45.311us  p_init(double*)
                   0.00%  15.392us        10  1.5390us  1.4720us  1.8560us  [CUDA memcpy HtoD]
                   0.00%  5.4080us         1  5.4080us  5.4080us  5.4080us  r_init(double*)
      API calls:   71.00%  2.05538s     31003  66.296us  4.2290us  767.24us  cudaDeviceSynchronize
                   12.67%  366.85ms     31004  11.832us  8.4650us  715.27us  cudaLaunch
                   11.96%  346.32ms        10  34.632ms  11.622us  346.21ms  cudaMemcpyToSymbol
                   2.17%  62.778ms     95011     660ns     410ns  687.18us  cudaSetupArgument
                   0.73%  21.158ms     31004     682ns     440ns  17.252us  cudaConfigureCall
                   0.72%  20.786ms         5  4.1573ms  10.389us  20.717ms  cudaMallocManaged
                   0.67%  19.530ms       101  193.37us  53.279us  257.03us  cudaMemcpy
                   0.04%  1.1106ms         1  1.1106ms  1.1106ms  1.1106ms  cuDeviceTotalMem
                   0.02%  496.71us         7  70.959us  10.670us  217.36us  cudaFree
                   0.01%  391.76us         3  130.59us  9.2880us  370.93us  cudaMalloc
                   0.01%  212.27us        94  2.2580us     361ns  78.406us  cuDeviceGetAttribute
                   0.00%  23.604us         1  23.604us  23.604us  23.604us  cuDeviceGetName
                   0.00%  5.0890us         3  1.6960us     410ns  4.1180us  cuDeviceGetCount
                   0.00%  1.3420us         2     671ns     401ns     941ns  cuDeviceGet

==7029== Unified Memory profiling result:
Device "Tesla V100S-PCIE-32GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
    1004  4.2227KB  4.0000KB  60.000KB  4.140625MB  2.929792ms  Host To Device
    1005  4.2227KB  4.0000KB  60.000KB  4.144531MB  1.725280ms  Device To Host
    1001         -         -         -           -  110.8347ms  Gpu page fault groups
     264  4.0000KB  4.0000KB  4.0000KB  1.031250MB           -  Memory thrashes
Total CPU Page faults: 1001
Total CPU thrashes: 264
```

Profilowanie programu wskazuje na dwa najpoważniejsze problemy, które zmniejszają efektywność programu. Jest nim liczenie sił w calc_VFP oraz duża ilość czasu spędzona na synchronizacji wątków z urządzeniem poprzez funkcję cudaDeviceSynchronize(). Synchronizacja wątków z urządzeniem jest konieczna, ponieważ kolejne wykonywane kernele opierają swoje działanie

na wynikach kerneli poprzednich (np. aby policzyć siły trzeba zaktualizować położenia atomów). Aby poprawić działanie programu, należy zredukować liczbę wywołań funkcji cudaDeviceSynchronize. Można to zrobić np. poprzez zastąpienie wielu mniejszych kerneli jednym silnikiem, który wykonuje wszystkie obliczenia związane z danym krokiem czasowym. Drugim największym problemem jest kernel calc_VFP, którego działanie jest bardzo złożone i powinno wykonać się więcej analiz nad sposobem jego pracy.

## Memory leaks

Poniżej przedstawiono wyniki badań wycieków pamięci z użyciem narzędzia valgrind.

```
==7157== Memcheck, a memory error detector
==7157== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7157== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7157== Command: ./argon parameters.config energies.out xyz.out
==7157==
==7157== Warning: noted but unhandled ioctl 0x30000001 with no size/direction hints.
==7157==    This could cause spurious value errors to appear.
==7157==    See README_MISSING_SYSCALL_OR_IOCTL for guidance on writing a proper wrapper.
==7157== Warning: noted but unhandled ioctl 0x27 with no size/direction hints.
==7157==    This could cause spurious value errors to appear.
==7157==    See README_MISSING_SYSCALL_OR_IOCTL for guidance on writing a proper wrapper.
                                              (...)
==7157==    See README_MISSING_SYSCALL_OR_IOCTL for guidance on writing a proper wrapper.
==7157== Warning: noted but unhandled ioctl 0x48 with no size/direction hints.
==7157==    This could cause spurious value errors to appear.
==7157==    See README_MISSING_SYSCALL_OR_IOCTL for guidance on writing a proper wrapper.
==7157== Warning: set address range perms: large range [0x22000000, 0x33fff000) (noaccess)
Zaladowano parametry z pliku parameters.config
Otworzono plik xyz.out do zapisu wspolrzednych symulowanych atomow
Otworzono plik energies.out do zapisu danych wyjsciowych programu
Inicjalizacja symulacji... Gotowe!
Rozpoczeto symulacje...
Obliczenia zakonczone!
==7157==
==7157== HEAP SUMMARY:
==7157==     in use at exit: 9,794,398 bytes in 12,932 blocks
==7157==   total heap usage: 16,291 allocs, 3,359 frees, 68,938,489 bytes allocated
==7157==
==7157== LEAK SUMMARY:
==7157==    definitely lost: 0 bytes in 0 blocks
==7157==    indirectly lost: 0 bytes in 0 blocks
==7157==      possibly lost: 22,424 bytes in 189 blocks
==7157==    still reachable: 9,771,974 bytes in 12,743 blocks
==7157==         suppressed: 0 bytes in 0 blocks
==7157== Rerun with --leak-check=full to see details of leaked memory
==7157==
==7157== For counts of detected and suppressed errors, rerun with: -v
==7157== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 7 from 7)
```