Sprawozdanie zadania Argon

Bartosz Kucypera, bk439964

18 sierpnia 2023

1 Wstęp

Zadanie składało się z trzech części. Pierwsza część polegała na symulacji klastra atomów argonu metodą dynamiki molekularne wykożystując CPU. W części drugiej należało poprawić wydajność rozwiązania, przenosząc część obliczeń na GPU jak i stosując szereg innych optymalizacji. W ostatniej, trzeciej części należało zbadać przeminay fazowe argonu.

2 Implementacja

Poszczególne wersje rozwiązania różnią się tylko obliczaniem sił działających na atomy, poniżej opiszę część roziwązania która jest wspólna dla wszystkich wersji.

2.1 Konfiguracja

Konfiguracja symulacji odbywa się poprzez stałe globlane.

```
/* stałe z treści */
constexpr real_t sig =  0.369;      /* nm */
constexpr real_t eps =  1.19;       /* kJ/mol */
constexpr real_t a   =  0.260922;   /* sig = sqrt(2)*a */
constexpr real_t ts  =  0.001;      /* time stamp w symulcji */
constexpr real_t B   =  1;          /* siła balonu */
constexpr real_t mas = 39.95;       /* masa atomu */
constexpr real_t kB  = 0.00831;     /* stała Boltzmana */
constexpr int MAX_LI = 250;         /* max dlugosc listy sasiadow, z zapasem */

/* wymiary siatki z atomami */
constexpr int siat_X = 6;
constexpr int siat_Y = 6;
constexpr int siat_Z = 6;
constexpr int N = siat_X*siat_Y*siat_Z*4; /*ilość atomów */

/* wymagana konfiguracja */
constexpr int DEBUG         = 1;        /* 0-nic, 1-najważniesze, 2-rozszerzone, 3-wszystko */
constexpr bool VERBOSE      = true;     /* czy wypisywać najważniejsze logi również na stdout */
constexpr int Nterm         = 1000;     /* ilość kroków fazy termalizacji */
constexpr int Ngrz          = 1000;     /* ilość kroków fazy grzania */
constexpr int Nch           = 1000;     /* ilość kroków fazy chłodzenia */
constexpr int ENERG         = 100;      /* częstotliowść zapisu anergi */
constexpr int TRAJE         = 100;      /* częstotliwość zapisu współrzędnych */
constexpr real_t rB         = 20;       /* r balonu w nm */
constexpr real_t rOd        = 1;        /* promień odcięcia */
constexpr real_t rBuff      = 0.2;      /* bufor */
constexpr real_t start_T    = 70;       /* temperatura początkowa */
constexpr real_t cool_T     = 20;       /* temperatura do której chłodzimy */
constexpr real_t heat_T     = 120;      /* temperatura do której ogrzewamy */
```

2.2 Reprezentacja atomów

Informacje o stanie symulacji przechowywane sa w nastepujacych tablicach globalnych.

```
/* tablice na hosice */
real_t cords[3*N];       /* gdzie jest atom */
real_t V[3*N];           /* prędkości atomu względem osi */
real_t F[3*N];           /* siła względem osi */
real_t Epot[2*N];        /* energia potencjalna Lennarda Jonesa, [0] - r12, [1] - r6 */
real_t Ebal[N];          /* enrgia potencjalna - człon balonu */
real_t Ekin[N];          /* energia kinetyczna atomu */

int G[N][MAX_LI];        /* lista sąsiedztwa atomów */
real_t poz0[N][3];       /* pozycja atomu w momencie budowy listy sąsiedztwa */

/* gpu I i gpu II */
real_t* F_gpu;           /* siła, gpu */
real_t* Epot_gpu;        /* energia potencjalna, gpu */
real_t* cords_gpu;       /* kordynaty, gpu */

/* wersja gpu 3, podzial na atomy w roli hosta i goscia */
real_t* F_gpu_host;
real_t* F_gpu_guest;
real_t* Epot_gpu_host;
real_t* Epot_gpu_guest;

/* wersja gpu 4, listy sasiedztwa */
int* G_gpu;
real_t poz0_cpu[N*3];
```

2.3 Krok symulacji

Do integracji równań ruchu używamy algorytmu leap-frog w postaci kick-drift-kick.

$$v_{i+1/2} = v_i + a_i + \frac{\Delta t}{2},$$

$$x_{i+1} = x_i + v_{i+1/2} \cdot \Delta t,$$

$$v_{i+1} = v_{i+1/2} + a_{i+1} \cdot \frac{\Delta t}{2}$$

Pętla główna symulacji wygląda następująco.

```
for (int nr = 0; nr < Nterm+Ngrz+Nch; nr++) {
    up_V();
    up_cords();

    up_forces(config);
    up_ballon();

    up_V();

    up_kin();
    up_temp(nr);

    up_logi(nr);
}
```

Po kolei:

(up_V)	Aktualizujemy predkosc (krok pierwszy leap-frog)
(up_cords)	Aktualizujemy polozenie atomow (krok drugi leap-frog)
(up_forces)	Aktualizujemy sily dzialajace na atomy (sposobem zaleznym od config)
(up_ballon)	Aktualizujemy czlon sily balonu
(up_V)	Aktualizujemy predkosc (krok trzeci leap-frog)
(up_kin)	Aktualizujemy energie kinetyczna
(up_temp)	Skalujemy predkosc zeby wplynac na temperature
(up_logi)	Aktualizujemy pliki logow

2.4 Aktualizacja predkosci

Podstawiamy $a = \frac{f}{m}$ do leap-frog, (ts to Δt).

```
inline void up_V() {
    for (int i = 0; i < N; i++)
        for (int k = 0; k < 3; k++)
            V[i*3 + k] += F[i*3 + k]/mas*ts/2.0;
}
```

2.5 Aktualizacja polozenia atomow

Wzor na droge w ruchu jednostajnym prostoliniowym $s = v \cdot t$.

```
inline void up_cords() {
    for (int i = 0; i < N; i++)
        for (int k = 0; k < 3; k++)
            cords[i*3 + k] += V[i*3 + k]*ts;
}
```

2.6 Aktualizacja energi kinetycznej

Korzystamy z wzrou $E_k = \frac{1}{2}m|v|^2$.

```
inline void up_kin() {
    for (size_t i = 0; i < N; i++)
        Ekin[i] = kinetyczna(i);
}
inline real_t kinetyczna(int i) {
    real_t res = 0;
    for (int k = 0; k < 3; k++)
        res += sq(V[i*3 + k]);
    return res*mas/2.0;
}
```

Gdzie *sq* to podniesienie do kwadratu.

2.7 Uwzglednienie dzialania balonu

Aktualizujemy wektory sil dzialajacych na atomy, oraz czlon energi wynikajacej z dzialania balonu.
B to sila sprerzystosci balonu, rB to promien balonu.

```
inline void up_ballon() {
    for (int i = 0; i < N; i++) {
        real_t r = 0;
        for (int k = 0; k < 3; k++)
            r += sq(cords[i*3 + k]);
        r = sqrt(r);

        if (r <= rB)
            return;

        real_t mno = B*(r-rB)/r;
        for (int k = 0; k < 3; k++)
            F[i*3 + k] -= mno*cords[i*3 + k];
        Ebal[i] += B*sq(r-rB)/2.0;
    }
}
```

2.8 Zmiana temperatury

Temperature zmieniamy skalujac predkosci atomow.

```
inline void up_temp(int nr) {
    if (nr < Nterm)
        return;

    real_t mno;

    /* chłodzenie */
    if (nr >= Nterm and nr < Nterm+Nch) {
        mno = scale_T(cur_temperatura(), cool_T, Nterm+Nch-nr);
    }
    /* grzanie */
    else {
        mno = scale_T(cur_temperatura(), heat_T, Nterm+Ngrz+Nch-nr);
    }

    for (int i = 0; i < N; i++)
        for (int k = 0; k < 3; k++)
            V[i*3 + k] *= mno;
}
```

Gdzie *scale_T* oblicza przez co trzeba przemnazac predkosci atomow by z danej temperatury poczatkowej dojsc do danej temperatury koncowej w danej liczbie krokow,

```
constexpr inline real_t scale_T(real_t pocz_T, real_t konc_T, real_t kroki = 1) {
    return sqrt(1.0 + (konc_T-pocz_T)/pocz_T/kroki);
}
```

cur_temperatura oblicza temperature ukladu,

```
inline real_t cur_temperatura() {
    real_t sum = 0;
    for (size_t i = 0; i < N; i++)
        sum += Ekin[i];
    return temperatura(sum);
}
```

temperatura przyjmuje calkowita energie kinetyczna ukladu i zwraca srednia temperature atomu

```
inline constexpr real_t temperatura(real_t kin) { return kin/N*2.0/3.0/kB; }
```

3 Badanie poprawnosci

Wszystkie pomiary byly wykonywane na symulacji o 10'000 krokach.

3.1 Poprawnosc wersji bez promienia odciecia

$\Delta t = 0.001ps$			
rodzaj energi	srednia	wariancja	odchylenie std.
kinetyczna	415.353793	13295.608949	115.306587
potencjalna	-6565.019095	13296.003279	115.308297
calkowita	-6149.665302	0.000005	0.002140
Bezwzgledna roznica calkowitej energi poczatkowej i koncowej = 0.003746			

$\Delta t = 0.002ps$			
rodzaj energi	srednia	wariancja	odchylenie std.
kinetyczna	414.784172	6998.312010	83.655914
potencjalna	-6564.438732	6999.164543	83.661010
calkowita	-6149.654560	0.000041	0.006431
Bezwzgledna roznica calkowitej energi poczatkowej i koncowej = 0.013607			

$\Delta t = 0.005ps$			
rodzaj energi	srednia	wariancja	odchylenie std.
kinetyczna	413.626971	3395.617382	58.271926
potencjalna	-6563.206356	3398.238114	58.294409
calkowita	-6149.579385	0.000851	0.029179
Bezwzgledna roznica calkowitej energi poczatkowej i koncowej = 0.089974			

$\Delta t = 0.010ps$			
rodzaj energi	srednia	wariancja	odchylenie std.
kinetyczna	414.513856	2131.072219	46.163538
potencjalna	-6563.825136	2137.813790	46.236498
calkowita	-6149.311280	0.009664	0.098306
Bezwzgledna roznica calkowitej energi poczatkowej i koncowej = 0.341988			

$\Delta t = 0.020ps$			
rodzaj energi	srednia	wariancja	odchylenie std.
kinetyczna	415.123586	1482.735668	38.506307
potencjalna	-6563.357016	1502.039471	38.756154
calkowita	-6148.233430	0.119225	0.345290
Bezwzgledna roznica calkowitej energi poczatkowej i koncowej = 1.425067			

$\Delta t = 0.050ps$			
rodzaj energi	srednia	wariancja	odchylenie std.
kinetyczna	410.270381	1051.504381	32.426908
potencjalna	-6548.686890	1133.001339	33.660085
calkowita	-6138.416509	8.139457	2.852973
Bezwzgledna roznica calkowitej energi poczatkowej i koncowej = 9.441853			

Symulacja spelnia warunek poprawnosci, energia calkowita jest prawie stala.
Gdy zwiekszamy Δt dokladnosc maleje, ale nawet dla $\Delta t = 0.050ps$ blad jest maly.

3.2 Wplyw promienia odciecia na dokladnosc obliczen

Wszystkie pomiary wykonane z $\Delta t = 0.001ps$, oraz rozmiarem bufora $0.2nm$.

promien odciecia = $1nm$			
rodzaj energi	srednia	wariancja	odchylenie std.
kinetyczna	405.969270	12883.481542	113.505425
potencjalna	-6434.775838	13911.779320	117.948206
calkowita	-6028.806568	45.052419	6.712110
Bezwzgledna roznica calkowitej energi poczatkowej i koncowej = 6.537997			

promien odciecia = $1.3nm$			
rodzaj energi	srednia	wariancja	odchylenie std.
kinetyczna	411.735610	13157.873468	114.707774
potencjalna	-6515.878892	13614.238444	116.680069
calkowita	-6104.143282	40.556854	6.368426
Bezwzgledna roznica calkowitej energi poczatkowej i koncowej = 3.108414			

promien odciecia = $1.5nm$			
rodzaj energi	srednia	wariancja	odchylenie std.
kinetyczna	413.685796	13313.460769	115.383971
potencjalna	-6539.407757	13455.558777	115.998098
calkowita	-6125.721961	4.910431	2.215949
Bezwzgledna roznica calkowitej energi poczatkowej i koncowej = 1.427013			

brak promienia odciecia			
rodzaj energi	srednia	wariancja	odchylenie std.
kinetyczna	415.353793	13295.608949	115.306587
potencjalna	-6565.019095	13296.003279	115.308297
calkowita	-6149.665302	0.000005	0.002140
Bezwzgledna roznica calkowitej energi poczatkowej i koncowej = 0.003746			

Zgodnie z oczekiwaniami, im mniejszy promien odciecia tym mniej dokladne obliczenia.
Blad jest jednak stosunkowo maly.

4 Liczenie sil dzialajacych na atomy i badanie wydajnosci

W tej czesci przedstawie po kolei coraz lepsze wersje symulacji i zbadam ich wydajnosc.

4.1 CPU I

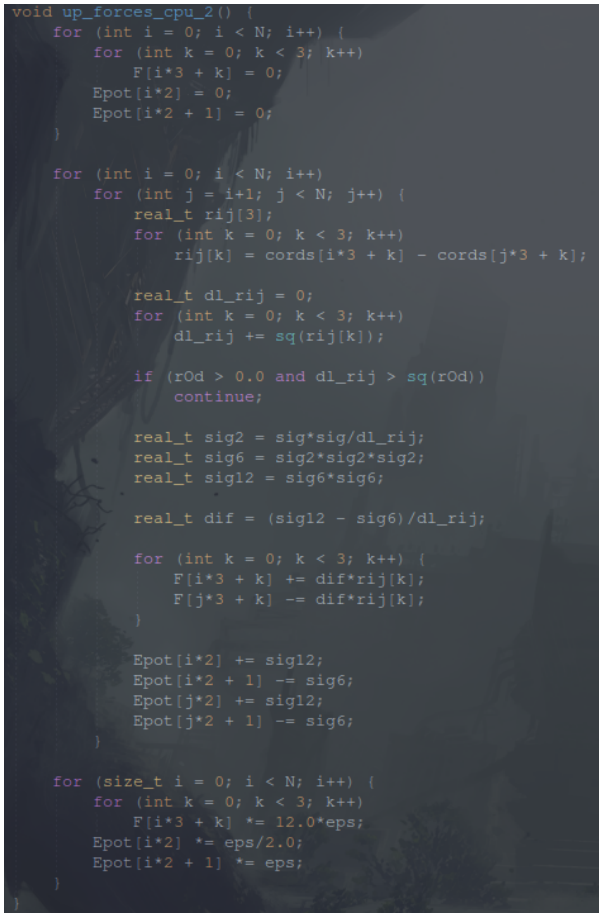
Pierwsza, najprostsza wersja cpu. Dla kazdego atomu iterujemy sie po wszystkich innych atomach i wyliczamy dzialajace na niego sily.

```
void up_forces_cpu_1() {
    for (int i = 0; i < N; i++) {
        for (int k = 0; k < 3; k++)
            F[i*3 + k] = 0;
        Epot[i*2] = 0;
        Epot[i*2 + 1] = 0;

        for (int j = 0; j < N; j++) {
            if (i == j)
                continue;

            real_t rij[3];
            for (int k = 0; k < 3; k++)
                rij[k] = cords[i*3 + k] - cords[j*3 + k];

            real_t dl_rij = 0;
            for (int k = 0; k < 3; k++)
                dl_rij += sq(rij[k]);
            if (rOd > 0.0 and dl_rij > sq(rOd))
                continue;

            real_t sig2 = sig*sig/dl_rij;
            real_t sig6 = sig2*sig2*sig2;
            real_t sig12 = sig6*sig6;

            real_t dif = (sig12 - sig6)/dl_rij;

            for (int k = 0; k < 3; k++)
                F[i*3 + k] += dif*rij[k];

            Epot[i*2] += sig12;
            Epot[i*2 + 1] -= sig6;
        }

        for (int k = 0; k < 3; k++)
            F[i*3 + k] *= 12.0*eps;

        Epot[i*2] *= eps/2.0;
        Epot[i*2 + 1] *= eps;
    }
}
```

Ta wersja jest niezwykle wolna, przeprowadzenie calej symulacji zajmuje nam 167821.64ms czyli okolo 168 sekund.

4.2 CPU II

W tej wersji wykozystujemy III zasade dynamiki Newtna by zmniejszyc ilosc obliczen o polowe.



```
void up_forces_cpu_2() {
    for (int i = 0; i < N; i++) {
        for (int k = 0; k < 3; k++)
            F[i*3 + k] = 0;
        Epot[i*2] = 0;
        Epot[i*2 + 1] = 0;
    }

    for (int i = 0; i < N; i++)
        for (int j = i+1; j < N; j++) {
            real_t rij[3];
            for (int k = 0; k < 3; k++)
                rij[k] = cords[i*3 + k] - cords[j*3 + k];

            real_t dl_rij = 0;
            for (int k = 0; k < 3; k++)
                dl_rij += sq(rij[k]);

            if (rOd > 0.0 and dl_rij > sq(rOd))
                continue;

            real_t sig2 = sig*sig/dl_rij;
            real_t sig6 = sig2*sig2*sig2;
            real_t sig12 = sig6*sig6;

            real_t dif = (sig12 - sig6)/dl_rij;

            for (int k = 0; k < 3; k++) {
                F[i*3 + k] += dif*rij[k];
                F[j*3 + k] -= dif*rij[k];
            }

            Epot[i*2] += sig12;
            Epot[i*2 + 1] -= sig6;
            Epot[j*2] += sig12;
            Epot[j*2 + 1] -= sig6;
        }

    for (size_t i = 0; i < N; i++) {
        for (int k = 0; k < 3; k++)
            F[i*3 + k] *= 12.0*eps;
        Epot[i*2] *= eps/2.0;
        Epot[i*2 + 1] *= eps;
    }
}
```

Wersja ta dziala okolo 2 razy szybciej. Cala symulacja zajmuje nam 78311.49ms czyli okolo 78 sekund.