Raport
Zrównoleglenie Heat Transfer 2D przy użyciu PCJ
Franciszek Grzędzicki grzedzicki@mat.umk.pl

Celem niniejszego raportu jest przedstawienie problemu rozchodzenia ciepła oraz zaimplementowanie jego rozwiązania wykorzystując bibliotekę PCJ w języku Java.

1. Abstrakt

Do rozwiązania problemu wykorzystano równanie przewodnictwa cieplnego metodą różnic skończonych dzięki czemu w prosty sposób otrzymamy wzór możliwy do zaimplementowania do siatki 2D. Przy niewielkiej wielkości siatki zyski z korzystania z wielowątkowości są nikłe, lecz przy odpowiednich ustawieniach możemy zyskać nawet 15 minut czasu co zostanie opisane w dalszej części tekstu.

2. Wzór

Podstawowy wzór na przewodnictwo cieplne: $\frac{\partial u}{\partial t} - \alpha \nabla u = 0$, aby móc wykorzystać wzór w

przestrzeni 2D wzór ten zapisujemy jako: $\frac{\partial u}{\partial t} - \alpha \left(\frac{\partial^2 u}{\partial x} + \frac{\partial^2 u}{\partial y}\right) = 0$, gdzie t to jednostka czasu, x oraz y to współrzędne w przestrzeni, a α to współczynnik wyrównania temperatur.

Wykorzystując metodę różnic skończonych wyprowadzoną z szeregu Taylora otrzymujemy $f'(a) \approx \frac{f(a+h)-f(a)}{h}$. Następnie zapiszemy nasze zmienne jako: $x_i = i\Delta x, y_j = j\Delta y, t_k = k\Delta t$.

Tym czego szukamy jest $u(x, y, t) = u_{i,j}^k$. Dzięki temu otrzymujemy

$$\frac{u_{i,j}^{k+1} - u_{i,j}^k}{\Delta t} - \alpha \left(\frac{u_{i+1,j}^k - 2u_{i,j}^k + u_{i-1,j}^k}{\Delta x^2} + \frac{u_{i,j+1}^k - 2u_{i,j}^k + u_{i,j-1}^k}{\Delta y^2}\right) = 0$$

Zakładamy, że siatka jest kwadratowa, więc $\Delta x = \Delta y$, wtedy mamy

$u_{i,j}^{k+1} = \gamma\left(u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k - 4u_{i,j}^k\right) + u_{i,j}^k$, gdzie $\gamma = \alpha \frac{\Delta t}{\Delta x^2}$.

3. Maszyna testowa

Testy przeprowadzona na maszynie:

- CPU: Intel Core i5-6600K, 3.5GHz, @4.7GHz OC
- RAM: Kingston HyperX FURY 4GB DDR4 2133MHz CL14 @2800MHz OC x2 (16GB)
- Disk: Crucial MX450 240GB

4. Implementacja sekwencyjna

```
for (int i = 1; i < fileSize - 1; i += delta_x)
    for (int j = 1; j < fileSize - 1; j+= delta_x) {
        newArray1D[i][j] =  (gamma * (myArray[i + 1][j] + myArray[i - 1][j] + myArray[i][j + 1] + myArray[i][j - 1] - 4 * myArray[i][j])) + myArray[i][j];
    }
```

Powyższa implementacja została wykorzystana przy użyciu biblioteki PCJ z jednym wątkiem.

5. Implementacja wielowątkowa

- Dla wątku 0

```
if(PCJ.myId() == 0){
        for (int i = 1; i < filelengthdown; i += delta_x) {
            if (i == filelengthdown - 1) {hArray = PCJ.get( threadId: PCJ.myId() + 1, Shared.myArray,  ...indices: 0);
            }
            for (int j = 1; j < filelength - 1; j += delta_x)
            {
                if (i == filelengthdown - 1)
                    newArrayPCJ[i][j] = (gamma * (hArray[j] + myArray[i - 1][j] + myArray[i][j + 1] + myArray[i][j - 1] - 4 * myArray[i][j])) + myArray[i][j];
                else newArrayPCJ[i][j] = (gamma * (myArray[i+1][j] + myArray[i - 1][j] + myArray[i][j + 1] + myArray[i][j - 1] - 4 * myArray[i][j])) + myArray[i][j];
            }
        }
```

- Dla wątków >0 oraz < PCJ.threadCount-1

```
for(int i = 0; i < filelengthdown; i += delta_x) {
    if(i == 0){
        hArray = PCJ.get( threadId: PCJ.myId() - 1, Shared.myArray,  ...indices: filelengthdown - 1);
    } else if (i == filelengthdown - 1) {
        hArray = PCJ.get( threadId: PCJ.myId() + 1, Shared.myArray,  ...indices: 0);
    }
    for (int j = 1; j < filelength - 1; j += delta_x) {
        if (i==0) newArrayPCJ[i][j] = (gamma * (myArray[i+1][j] + hArray[j] + myArray[i][j + 1] + myArray[i][j - 1] - 4 * myArray[i][j])) + myArray[i][j];
        else if (i==filelengthdown-1) newArrayPCJ[i][j] = (gamma * (hArray[j] + myArray[i-1][j] + myArray[i][j + 1] + myArray[i][j - 1] - 4 * myArray[i][j])) + myArray[i][j];
        else newArrayPCJ[i][j] = (gamma * (myArray[i-1][j] + myArray[i+1][j] + myArray[i][j + 1] + myArray[i][j - 1] - 4 * myArray[i][j])) + myArray[i][j];
```

- Dla ostatniego wątku

```
for(int i = 0; i < filelengthdown + rest - 1; i += delta_x) {
    if (i == 0) hArray = PCJ.get( threadId: PCJ.myId() - 1, Shared.myArray,  ...indices: filelengthdown - 1);
    for (int j = 1; j < filelength - 1; j += delta_x) {
        if(i == 0) newArrayPCJ[i][j] = (gamma * (myArray[i + 1][j] + hArray[j] + myArray[i][j + 1] + myArray[i][j - 1] - 4 * myArray[i][j])) + myArray[i][j];
        else newArrayPCJ[i][j] = (gamma * (myArray[i + 1][j] + myArray[i - 1][j] + myArray[i][j + 1] + myArray[i][j - 1] - 4 * myArray[i][j])) + myArray[i][j];
```

6. Sposób implementacji wielowątkowej oraz wyjaśnienie wzoru

Zgodnie ze wzorem, posiadając siatkę 2D obliczeń dokonuje się nie zmieniając wartości brzegowych. Mając to na uwadze przy implementacji jednowątkowej obliczeń dokonujemy od pierwszego wiersza oraz pierwszej kolumny, nie dotykamy 0, a kończmy na przedostatniej linii oraz przedostatniej kolumnie. Do obliczenia temperatury danej komórki musimy znać temperatury z poprzedniego kroku jej sąsiada z góry, dołu, lewej strony oraz prawej. Jak widać w tym wzorze bardzo opieramy się na komunikacji między komórkami, dlatego przy małych siatkach lepiej używać jednego wątku.

Do implementacji wielowątkowej uznałem, że plik z temperaturami zostanie podzielony na bloki w taki sposób, że każdy wątek dostaje kolumny od 0 do końca, a wiersze na podstawie dzielenia długości pliku przez ilość wątków, w przypadku, gdy nie uda się podzielić pliku przez ilość wątków, to wątek ostatni otrzymuje swój przydział oraz resztę wierszy. Dzięki takiej implementacji jedyna komunikacja między wątkami następuje, gdy wątki działają nad wierszem 0 i ostatnim.

Podczas implementacji początkowo w drugiej pętli for, gdzie poruszamy się po kolumnach za każdym razem wywoływałem funkcję, która pytała sąsiedni wątek o temperaturę jego komórki, która jest sąsiadem komórki, dla której liczymy temperaturę. Podczas testów zauważyłem, że rozwiązanie to jest słabe, ponieważ to co zyskujemy wielowątkowością tracimy na komunikację. Ponieważ w bibliotece PCJ poza pobieraniem jednej wartości można dzielić się całymi tablicami, postanowiłem, że gdy wartości i (w przypadku wątków >0) wynosi 0, dokonuje jednej komunikacji z wątkiem posiadającym blok nad nami, aby pobrać od niego cały ostatni wiersz, a następnie już w pętli j, po prostu pobierać z tej tablicy wartości. Zostało to zaimplementowane dla wszystkich wątków, w przypadku wątku 0, jedyna komunikacja to zapytanie do wątku 1, gdy działamy na ostatnim wierszu, wątki od 1 do przedostatniego, komunikują się między sobą dwa razy, gdy i = 0 oraz i = ostatni wiersz im przydzielony, a wątek ostatni w przeciwieństwie do 0, gdy i =0. Dzięki takiemu rozwiązaniu czasy obliczeń się znacząco poprawiły przy dużej ilości danych.

7. Wyniki czasowe

	Sekwencyjnie	**2**	**4**	**8**	**20**
1000x1000 1S	**15ms**	27ms	28ms	**39ms**	35ms
1000x1000 100S	476ms	448ms	459ms	**423ms**	**540ms**
1000x1000 500S	**1.842s**	**1.438s**	1.5s	1.695s	1.531s
1000x1000 1.000S	**3.321s**	2.599s	**2.445s**	2.473s	2.688s
1000x1000 10.000S	**29.745s**	22.043s	20.837s	**20.636s**	21.327s
1000x1000 50.000S	**2.43min**	2.197min	**1.669min**	1.755min	1.8min
1000x1000 100.000S	**4.673min**	3.444min	**3.284min**	3.405min	3.351min
1000x1000 1.000.000S	**53min**	**38.840min**	40min	42min	38.979min

8. Podsumowanie

Po dostosowaniu algorytmu do wielowątkowości w postaci bloków dzięki ograniczeniu komunikacji zgodnie z założeniem można zauważyć, że obliczenia dokonują się szybciej w postaci wielowątkowej przy większej ilości obliczeń. Uważam więc, że algorytm został poprawnie zaimplementowany choć zapewne można by coś jeszcze poprawić, lecz obecnie nie zauważyłem takiej opcji.

9. Bibliografia

- https://www.youtube.com/watch?v=LJPSRp0ZeJo&list=PLZOZfX_TaWAHZOgn8CRjpqRElp5Dd-GaY&index=13&ab_channel=CPPMechEngTutorials
- https://en.wikipedia.org/wiki/Heat_equation
- https://studyres.com/doc/16038790/finite-difference-methods-in-2d-heat-transfer
- https://levelup.gitconnected.com/solving-2d-heat-equation-numerically-using-python-3334004aa01a