**Laborator 1 PPD – Documentatie**

**Tema:**

Considerand ca se da o matrice V(n,m) si o matrice de convolutie C(k,k) se cere sa se calculeze matricea V(n,m) rezultata in urma aplicarii convolutiei cu matricea de convolutie C pe matricea F.

A) Program secvential

B) Program paralel: folositi p threaduri pentru calcul.

Obiectiv: Impartire cat mai echilibrata si eficienta a calculul pe threaduri!

Pentru impartirea sarcinilor de calcul (taskuri) se va folosi descompunere geometrica cu urmatoarele 2 variante:

- Pe orizontala (mai multe linii alocate unui thread)

si

- Pe verticala (mai multe coloane alocate unui thread)

**Proiectare:**

**Structuri de Date:**

1. **matrix** - o matrice pentru datele de intrare, matrice NxM
2. **kernel** - un kernel folosit pentru convoluție, matrice nxm
3. **result** - matricea rezultat după aplicarea convoluției, matrice NxM
4. **THREAD\_COUNT** - numărul de fire de execuție disponibile
5. **m**, **n**, **N**, **M**, **lineOffset**, **columnOffset** - variabile pentru dimensiunile matricei și offset-urile necesare pentru convoluție

**Partiționare pe Fir de Execuție:**

1. O execuție secvențială pentru calculul convoluției.
2. O execuție paralelă, unde matricea este împărțită pe linii între firele de execuție pentru a calcula convoluția în paralel.

**Clase:**

1. **Main** - clasa principală care conține metodele **main**, **read**, **write**, **sequential**, și **parallel**.
2. **MyThread** - o clasă internă care extinde **Thread** și reprezintă un fir de execuție pentru calculul convoluției pe o parte a matricei.

**Funcții:**

1. **read(String filename)** - citirea datelor de intrare dintr-un fișier.
2. **write(String output)** - scrierea matricei rezultat într-un fișier de ieșire.
3. **sequential()** - calculul convoluției în mod secvențial.
4. **parallel()** - calculul convoluției în mod paralel folosind mai multe fire de execuție.
5. **convolution(int x, int y)** - calculul convoluției pentru un punct dat.

**Relații:**

* Metoda **main** inițializează parametrii și alege între execuția secvențială și cea paralelă.
* Firul de execuție **MyThread** realizează convoluția pe o parte din matrice.

**Detalii de Implementare:**

* Citirea și scrierea datelor sunt realizate din și în fișiere text.
* Algoritmii pentru calculul convoluției sunt implementați în metoda **convolution**.
* Operațiile de citire și scriere folosesc fluxuri de caractere pentru manipularea fișierelor.
* Numărul de fire de execuție este ales în funcție de parametrul **p** din linia de comandă.

**Analiza:**

Conform rezultatelor obtinute in outJ.xlsx si outC.xlsx, programul ruleaza mai rapid folosind threaduri in C++, mai ales cand modul de alocare al matricilor este static.

**Dimensiuni Matrice și Kernel:**

**N=M=10, n=m=3**

* Pentru o matrice mică, performanța este mai bună atunci când se folosește o singură secvență (un fir de execuție).
* Cu 4 fire de execuție, performanța este mai slabă decât cea secvențială. Aceasta poate fi datorată overhead-ului adăugat de paralelizare.

**N=M=1000, n=m=5**

* Cu o matrice de dimensiuni mai mari, paralelizarea aduce în general beneficii, în special cu 4 fire de execuție.
* Cu 16 fire de execuție, performanța este apropiată de cea cu 2 fire, ceea ce poate indica o saturare a resurselor de calcul.

**N=10000, M=10, n=m=5**

* Pentru o matrice de dimensiuni neobișnuite, performanța este cea mai bună cu 4 fire de execuție, în special în cazul alocării statice.
* Cu mai multe fire de execuție, performanța scade semnificativ.

**N=10, M=10000, n=m=5**

* Această configurație arată că performanța depinde de modul în care datele sunt alocate.
* Alocarea statică pare să ofere performanțe mult mai bune decât alocarea dinamică în acest caz.

**Tipuri de Alocare**

* Alocarea statică pare să ofere în general performanțe mai bune decât alocarea dinamică pentru majoritatea configurațiilor