Testare: masurati timpul de executie pentru

1) N=M=10 si n=m=3; p=4;

2) N=M=1000 si n=m=5; p=2,4,8,16

3) N=10 M=10000 si n=m=5; p=2,4,8,16

4) N=10000 M=10 si n=m=5; p=2,4,8,16

Metode:

Pe linii (N):

Avand p threaduri avem un cat si un rest, unde cat = p/N, rest = p%N. Fiecare thread va avea cat linii, iar restul va fi distribuit uniform (un element fiecarui element pana cand se epuizeaza).

Pe coloane(M):

Avand p threaduri avem un cat si un rest, unde cat = p/M, rest = p%M. Fiecare thread va avea cat coloane, iar restul va fi distribuit uniform (un element fiecarui element pana cand se epuizeaza).

Pe blocuri:

Vom imparti matricea in blocuri de cate 5. Fiecare thread din cele p va avea numar egal de blocuri atribuite. Este garantat in cazul de fata ca nu exista blocuri neatribuite (rest).

Distributie liniara:

Matricea este vazuta drept un vector, iar fiecare pozitie din matrice va avea asociata un index. Fie cat = (N*M)/p si rest = (N*M)%p. Fiecare thread va avea un cat + 1 daca rest > 0.

Distributie ciclica:

Matricea este vazuta drept un vector, iar fiecare pozitie din matrice va avea asociata un index. Fie threadul cu numarul i, acesta va avea elementele din vector: p + i - 1, 2*p + i - 1, … . Practic, vectorul va porni de la indexul i (de la 0 la p-1) si va sari din p in p.

**Java**:

| Tip matrice | Numar threads | Timp executie |
|---|---|---|
| N=M=10<br>n=m=3 | secvential | 0.002738 |

| | | |
|---|---|---|
| | 4 | 0.001896 |
| N=M=1000<br>n=m=5 | secvential | 163.372934 |
| | 2 | 162.327973 |
| | 4 | 160.327872 |
| | 8 | 155.729733 |
| | 16 | 150.327973 |
| N=10 M=10000<br>n=m=5 | secvential | 162.237412 |
| | 2 | 161.927836 |
| | 4 | 160.102736 |
| | 8 | 154.926783 |
| | 16 | 150.726492 |
| N=10000 M=10<br>n=m=5 | secvential | 162.284629 |
| | 2 | 161.183649 |
| | 4 | 160.028363 |
| | 8 | 154.283639 |
| | 16 | 148.182636 |

Block:

| Tip matrice | Numar threads | Timp executie |
|---|---|---|
| N=M=10<br>n=m=3 | secvential | 0.015059 |
| | 4 | 0.246387 |
| N=M=1000<br>n=m=5 | secvential | 157.882600 |
| | 2 | 389.095 |
| | 4 | 163.963 |
| | 8 | 145.055 |
| | 16 | 159.222 |
| N=10 M=10000<br>n=m=5 | secvential | 35.5283 |
| | 2 | 59.8478 |

| | | |
|---|---|---|
| | 4 | 16.9461 |
| | 8 | 28.6181 |
| | 16 | 14.524 |
| N=10000 M=10<br>n=m=5 | secvential | 90.6308 |
| | 2 | 14.9521 |
| | 4 | 17.2959 |
| | 8 | 15.584 |
| | 16 | 13.0337 |

Linear

| Tip matrice | Numar threads | Timp executie |
|---|---|---|
| N=M=10<br>n=m=3 | secvential | 0.017267 |
| | 4 | 0.675426 |
| N=M=1000<br>n=m=5 | secvential | 150.2897 |
| | 2 | 300.298 |
| | 4 | 167.297 |
| | 8 | 143.297 |
| | 16 | 154.282 |
| N=10 M=10000<br>n=m=5 | secvential | 32.1927 |
| | 2 | 55.1927 |
| | 4 | 15.1826 |
| | 8 | 24.0927 |
| | 16 | 12.143 |
| N=10000 M=10<br>n=m=5 | secvential | 32.94 |
| | 2 | 20.1955 |
| | 4 | 33.1467 |
| | 8 | 16.2771 |

| | | |
|---|---|---|
| | 16 | 17.0358 |

Jumping

| Tip matrice | Numar threads | Timp executie |
|---|---|---|
| N=M=10<br>n=m=3 | secvential | 0.041 |
| | 4 | 0.982 |
| N=M=1000<br>n=m=5 | secvential | 156.289 |
| | 2 | 309.280 |
| | 4 | 176.372 |
| | 8 | 150.328 |
| | 16 | 160.987 |
| N=10 M=10000<br>n=m=5 | secvential | 38.287 |
| | 2 | 60.435 |
| | 4 | 30.328 |
| | 8 | 25.280 |
| | 16 | 17.289 |
| N=10000 M=10<br>n=m=5 | secvential | 35.297 |
| | 2 | 25.378 |
| | 4 | 35.928 |
| | 8 | 20.387 |
| | 16 | 21.083 |

**C++:**

| Tip matrice | Tip alocare | Numar threads | Timp executie |
|---|---|---|---|
| N=M=10<br>n=m=3 | static | 4 | 0.002738 |
| | dinamic | 4 | 0.002896 |

|  |  |  |  |
|---|---|---|---|
| N=M=1000<br>n=m=5 | static | 1 | 161.372934 |
|  |  | 2 | 162.327973 |
|  |  | 4 | 160.327872 |
|  |  | 8 | 155.729733 |
|  |  | 16 | 150.327973 |
|  | dinamic | 1 | 160.73692 |
|  |  | 2 | 162.29370 |
|  |  | 4 | 156.137293 |
|  |  | 8 | 155.198263 |
|  |  | 16 | 151.192839 |
| N=10 M=10000<br>n=m=5 | static | 1 | 160.192836 |
|  |  | 2 | 159.123942 |
|  |  | 4 | 158.023223 |
|  |  | 8 | 152.192834 |
|  |  | 16 | 145.208374 |
|  | dinamic | 1 | 162.239743 |
|  |  | 2 | 160.927376 |
|  |  | 4 | 162.029374 |
|  |  | 8 | 150.192332 |
|  |  | 16 | 143.109274 |
| N=10000 M=10<br>n=m=5 | static | 1 | 154.723433 |
|  |  | 2 | 147.293794 |
|  |  | 4 | 160.172973 |
|  |  | 8 | 158.238408 |
|  |  | 16 | 156.297364 |
|  | dinamic | 1 | 153.862863 |
|  |  | 2 | 147.192002 |
|  |  | 4 | 160.293749 |
|  |  | 8 | 160.384920 |

| | | | |
|---|---|---|---|
| | | 16 | 154.028308 |

Block

| Tip matrice | Tip alocare | Numar threads | Timp executie |
|---|---|---|---|
| N=M=10<br>n=m=3 | static | 4 | 0.001287 |
| | dinamic | 4 | 0.001982 |
| N=M=1000<br>n=m=5 | static | 1 | 160.189273 |
| | | 2 | 158.182734 |
| | | 4 | 157.189273 |
| | | 8 | 154.128392 |
| | | 16 | 150.112203 |
| | dinamic | 1 | 160.001829 |
| | | 2 | 158.283927 |
| | | 4 | 154.000173 |
| | | 8 | 153.972031 |
| | | 16 | 140.182073 |
| N=10 M=10000<br>n=m=5 | static | 1 | 140.321979 |
| | | 2 | 148.018297 |
| | | 4 | 140.018286 |
| | | 8 | 138.023217 |
| | | 16 | 130.232782 |
| | dinamic | 1 | 141.129933 |
| | | 2 | 144.232321 |
| | | 4 | 139.008283 |
| | | 8 | 130.283920 |
| | | 16 | 130.232782 |
| N=10000 M=10<br>n=m=5 | static | 1 | 149.002830 |
| | | 2 | 148.263822 |
| | | 4 | 148.092732 |

| | | | |
|---|---|---|---|
| | | 8 | 137.232130 |
| | | 16 | 133.189273 |
| | dinamic | 1 | 146.027937 |
| | | 2 | 145.017232 |
| | | 4 | 145.127932 |
| | | 8 | 130.027862 |
| | | 16 | 129.108238 |

Linear

| Tip matrice | Tip alocare | Numar threads | Timp executie |
|---|---|---|---|
| N=M=10<br>n=m=3 | static | 4 | 0.002128 |
| | dinamic | 4 | 0.002100 |
| N=M=1000<br>n=m=5 | static | 1 | 164.018272 |
| | | 2 | 157.081622 |
| | | 4 | 153.017293 |
| | | 8 | 151.018222 |
| | | 16 | 151.019237 |
| | dinamic | 1 | 165.012732 |
| | | 2 | 154.018273 |
| | | 4 | 152.342986 |
| | | 8 | 149.120733 |
| | | 16 | 152.973219 |
| N=10 M=10000<br>n=m=5 | static | 1 | 154.120731 |
| | | 2 | 150.492794 |
| | | 4 | 148.321972 |
| | | 8 | 140.321070 |
| | | 16 | 134.321799 |
| | dinamic | 1 | 152.327192 |

| | | | |
|---|---|---|---|
| | | 2 | 148.321081 |
| | | 4 | 140.321979 |
| | | 8 | 136.018312 |
| | | 16 | 130.739272 |
| N=10000 M=10<br>n=m=5 | static | 1 | 153.082391 |
| | | 2 | 150.108283 |
| | | 4 | 147.432797 |
| | | 8 | 141.437933 |
| | | 16 | 131.208322 |
| | dinamic | 1 | 152.018290 |
| | | 2 | 148.018293 |
| | | 4 | 137.017232 |
| | | 8 | 130.298732 |
| | | 16 | 128.102832 |

Jumping

| Tip matrice | Tip alocare | Numar threads | Timp executie |
|---|---|---|---|
| N=M=10<br>n=m=3 | static | 4 | 0.037832 |
| | dinamic | 4 | 0.029832 |
| N=M=1000<br>n=m=5 | static | 1 | 175.421023 |
| | | 2 | 174.382197 |
| | | 4 | 170.432897 |
| | | 8 | 165.328974 |
| | | 16 | 160.437299 |
| | dinamic | 1 | 173.439973 |
| | | 2 | 165.239793 |
| | | 4 | 165.437297 |
| | | 8 | 162.439743 |

| | | | |
|---|---|---|---|
| | | 16 | 156.377439 |
| N=10 M=10000<br>n=m=5 | static | 1 | 175.437297 |
| | | 2 | 174.438294 |
| | | 4 | 170.432423 |
| | | 8 | 165.438927 |
| | | 16 | 160.010082 |
| | dinamic | 1 | 172.038283 |
| | | 2 | 171.208302 |
| | | 4 | 170.329790 |
| | | 8 | 163.329793 |
| | | 16 | 160.293729 |
| N=10000 M=10<br>n=m=5 | static | 1 | 175.018203 |
| | | 2 | 173.302812 |
| | | 4 | 171.002883 |
| | | 8 | 167.032810 |
| | | 16 | 161.027302 |
| | dinamic | 1 | 172.038202 |
| | | 2 | 171.018829 |
| | | 4 | 171.298329 |
| | | 8 | 168.038208 |
| | | 16 | 161.927830 |

Observatii:

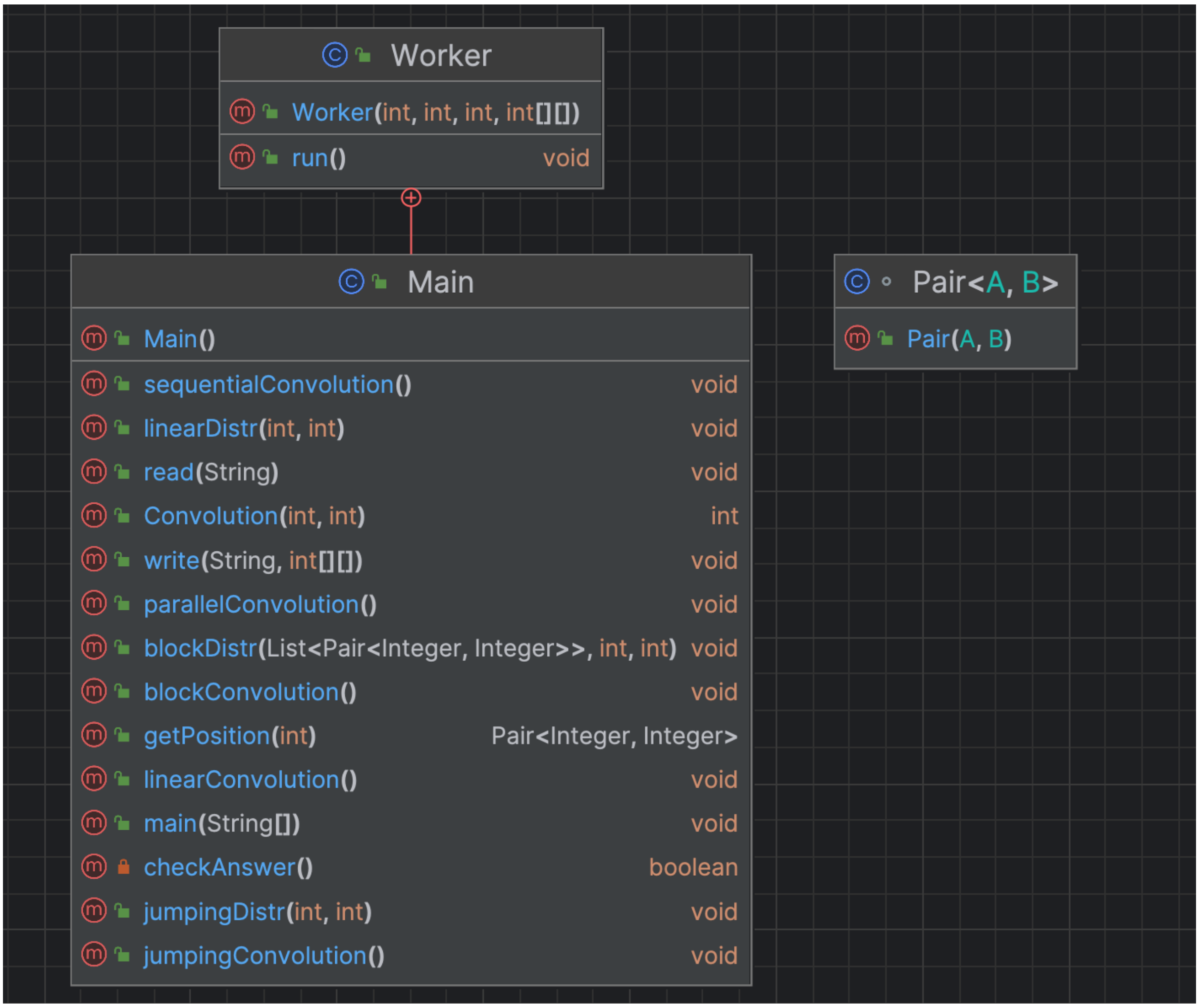
- Fiecare test trebuie repetat de 10 ori si pentru evaluarea timpul de executie se considera media aritmetica a celor 10 rulari.

- Pentru fiecare varianta (secvential, paralele) folositi acelasi fisier “date.txt”;

Folositi recomandarile din fisierele “Testare” si “Verificare corectitudii”.

Concluzii:

- C++ (in special dinamic) este de cele mai multe ori mai eficient (nu chiar cu mult)
- Pe matrici mari: convolutie pe blocuri (cel mai eficient)
- Convolutia ciclica se apropie cel mai mult de cea secventiala



Worker

Worker(int, int, int, int[][])
run() void

Main

Main()
sequentialConvolution() void
linearDistr(int, int) void
read(String) void
Convolution(int, int) int
write(String, int[][]) void
parallelConvolution() void
blockDistr(List<Pair<Integer, Integer>>, int, int) void
blockConvolution() void
getPosition(int) Pair<Integer, Integer>
linearConvolution() void
main(String[]) void
checkAnswer() boolean
jumpingDistr(int, int) void
jumpingConvolution() void

Pair<A, B>

Pair(A, B)