Raport timpi executie

Metoda	Tip matrice	Nr. threads	Timpi		
			Java	C++ static alc.	C++ dyn. alc.
Pe linii	N=M=10 n=m=3	Sequencial	0.7ms	0.04366ms	0.04644ms
		2			
		4	0.2ms	0.04051ms	0.03883ms
		8			
		16			
	N=M=1000 n=m=5	Sequencial	172.7ms	855.0386000000002ms	1408.118ms
		2	159.7ms	837.0911ms	1415.479ms
		4	171.7ms	921.4137999999999ms	1412.0890000000002ms
		8	188.2ms	1413.0819999999999ms	1409.91ms
		16	177.5ms	1387.5680000000002ms	1406.8249999999998ms
	N=10, M=10000 n=m=5	Sequencial	66.6ms	141.4809ms	141.4907ms
		2	71.4ms	140.39000000000001ms	142.7225ms
		4	68.4ms	142.3937ms	144.5376ms
		8	54.4ms	141.505ms	141.8767ms
		16	57.7ms	141.7174ms	143.0705ms
	N=10000, M=10 n=m=5	Sequencial	52.2ms	143.8905ms	143.9432ms
		2	47.5ms	142.6981ms	143.4584ms
		4	41.7ms	143.42260000000002ms	143.0679ms
		8	48.9ms	145.03650000000002ms	144.36489999999998ms
		16	63.6ms	146.3693ms	143.45719999999997ms
Pe coloane	N=M=10 n=m=3	Sequencial	0.6ms	0.06123ms	0.04536ms
		2			
		4	0.1ms	0.05031ms	0.04269ms
		8			
		16			
	N=M=1000 n=m=5	Sequencial	292.6ms	1410.2689999999998ms	1413.1490000000001ms
		2	298.2ms	1411.0919999999999ms	1408.417ms
		4	287.1ms	1411.9769999999999ms	1423.9829999999997ms
		8	282.0ms	1409.012ms	1404.9959999999996ms
		16	285.2ms	1413.4450000000002ms	1401.7910000000002ms
	N=10, M=10000 n=m=5	Sequencial	78.8ms	141.0983ms	142.1872ms
		2	70.3ms	142.07670000000002ms	143.73180000000002ms
		4	69.1ms	142.3546ms	139.7724ms
		8	75.2ms	141.17659999999998ms	141.93019999999999ms
		16	70.4ms	141.7462ms	140.8524ms
	N=10000, M=10 n=m=5	Sequencial	62.8ms	145.16479999999999ms	143.14190000000002ms
		2	58.4ms	143.3789ms	143.6478ms
		4	61.4ms	143.8575ms	145.3441ms
		8	56.9ms	144.0623ms	144.7795ms
		16	60.8ms	145.0118ms	144.3375ms

Pe blocuri	N=M=10 n=m=3	Sequencial	0.7ms	0.05317ms	0.0486ms
		2			
		4	0.9ms	0.04581ms	0.04438ms
		8			
		16			
	N=M=1000 n=m=5	Sequencial	311.7ms	1417.442ms	1413.783ms
		2	257.4ms	1413.08ms	1404.96ms
		4	281.4ms	1407.841ms	1093.1325000000002ms
		8	284.9ms	929.2028000000003ms	868.3705999999999ms
		16	309.3ms	1428.48ms	854.4012ms
	N=10, M=10000 n=m=5	Sequencial	70.8ms	140.00900000000001ms	88.32522999999999ms
		2	70.0ms	142.1274ms	85.71130000000001ms
		4	71.0ms	140.4049ms	94.9155ms
		8	74.2ms	143.22320000000002ms	134.1874ms
		16	71.4ms	137.6026ms	141.10510000000002ms
	N=10000, M=10 n=m=5	Sequencial	65.7ms	143.3011ms	143.50800000000004ms
		2	56.0ms	141.85309999999998ms	143.8367ms
		4	55.7ms	144.19379999999998ms	142.05960000000002ms
		8	54.3ms	142.6292ms	141.69729999999998ms
		16	59.6ms	145.84539999999998ms	144.6449ms
D. liniara	N=M=10 n=m=3	Sequencial	0.5ms	0.05491ms	0.04468ms
		2			
		4	0.7ms	0.05096ms	0.04507ms
		8			
		16			
	N=M=1000 n=m=5	Sequencial	275.0ms	1400.8529999999998ms	961.5847000000001ms
		2	303.7ms	1405.2090000000003ms	869.3103000000001ms
		4	303.0ms	1395.8280000000002ms	855.3788000000001ms
		8	314.5ms	1410.1830000000002ms	1305.7482999999997ms
		16	296.6ms	1405.0839999999998ms	1406.4869999999999ms
	N=10, M=10000 n=m=5	Sequencial	71.6ms	144.0958ms	142.0539ms
		2	73.6ms	140.5615ms	141.1959ms
		4	71.1ms	140.9006ms	140.3759ms
		8	69.7ms	141.3623ms	142.25089999999997ms
		16	72.1ms	149.86669999999998ms	143.81170000000003ms
	N=10000, M=10 n=m=5	Sequencial	53.4ms	123.45992000000001ms	141.69109999999998ms
		2	57.6ms	138.13320000000002ms	118.16906000000002ms
		4	57.3ms	144.1587ms	94.67733999999999ms
		8	58.7ms	143.3248ms	91.9037ms
		16	52.5ms	143.2853ms	91.85178ms
D. ciclica	N=M=10 n=m=3	Sequencial	0.4ms	0.04306ms	0.04604ms
		2			
		4	0.5ms	0.04623ms	0.04268ms
		8			
		16			
	N=M=1000	Sequencial	276.9ms	905.7324999999998ms	874.8035ms

	n=m=5	2	297.4ms	949.592ms	953.9918000000001ms
		4	306.4ms	1406.092ms	956.7807999999999ms
		8	307.3ms	1409.7649999999999ms	869.7274ms
		16	281.7ms	1411.836ms	869.2369999999999ms
	N=10, M=10000 n=m=5	Sequencial	68.9ms	141.2677ms	88.82759999999999ms
		2	73.0ms	140.32139999999998ms	85.60920999999999ms
		4	71.1ms	143.56679999999997ms	110.45194ms
		8	74.5ms	138.6771ms	139.62920000000003ms
		16	71.6ms	142.3922ms	140.2942ms
	N=10000, M=10 n=m=5	Sequencial	58.6ms	126.69532ms	142.5922ms
		2	57.4ms	90.10504ms	144.7217ms
		4	61.5ms	109.3886ms	143.26569999999998ms
		8	57.1ms	96.12807000000001ms	144.6641ms
		16	59.9ms	89.50988000000001ms	145.2568ms

Concluzii

- In unele cazuri, in C++ varianta cu alocare dinamica poate fi chiar cu pana la 40% mai rapida decat cea statica
- In C++, este mult mai vizibil raportul timp/cantitate de date (daca X date se proceseaza in T secunde, atunci 10X date se proceseaza in 10T secunde). Aceasta corelatie nu este insa valabila in Java.
- Varianta in Java este mai rapida decat cea in C++, posibil datorita JIT-ului si optimizarilor facute de JRE, dar si din cauza faptului ca programul C++ a fost compilat in Debug mode.
- Secvential vs parallel
    - Pe dispozitivul folosit pentru testare, paralelizarea nu aduce un bonus considerabil la performanta fata de varianta secventiala. Folosirea threadurilor in Java aduce un boost mediu de pana la 15%, dar exista cazuri in care poate incetini executia. In C++, diferentele de timp sunt de pana in 5-10%.
- Paralelism pe numar diferit threaduri
    - In general, utilizarea a 2-4 threaduri pot da timpi de executie mai buni.
    - In timp ce 8-16 threaduri pot aduce un mic overhead. In putine cazuri 16 threaduri se pot comporta mai bine decat toate celelalte cazuri.
- Comparatie metode
    - Parcurgerea pe linii este mai rapida decat cea pe coloane, cel mai probabil datorita cache-ului de processor care este mai efficient cand accesam memoria continua. Distributia liniara si cea ciclica au performante asemanatoare, in timp ce mai costisitoare este impartirea pe blocuri.

Descrierea algoritmilor

Fie p numarul de threaduri si definim $L(S) = \bigcup_{i=0}^{\infty} \prod_{j=0}^{i} S$ multmea listelor de elemente din S.

Considerăm două funcții $intsplit_p: \mathbb{N} \to L(\mathbb{N}^2)$ si $blocksplit_p: \mathbb{N}^4 \to L(\mathbb{N}^4)$ care impart un interval $[0, n)$, respective un block (i, j, n, m) in p subsegmente/subblocuri de marimi aproximativ egale.

Pseudo pentru calculul $intsplit_p(n)$:

Fie $q = \left[\frac{n}{p}\right]$ si $r = n - q * p$

$i \coloneqq 0.$

$start \coloneqq 0.$

for $i = 0..(p-1)$ do

$end \coloneqq q + (i < r)$

$(start, end) \in intsplit_p(n)$

$start \coloneqq end$

end

Pentru calculul $blocksplit_p(i, j, n, m)$ folosim un algoritm divide et impera care creeaza p subblocuri:

daca p=1 atunci

$(i, j, n, m) \in blocksplit_p(i, j, n, m)$

return

end

daca $n > m$ atunci

$blocksplit_{\frac{p}{2}+p\%2}\left(i, j, \frac{n}{2}, m\right) \subseteq blocksplit_p(i, j, n, m)$

$blocksplit_{\frac{p}{2}}\left(i + \frac{n}{2}, j + \frac{m}{2}, n - \frac{n}{2}, m\right) \subseteq blocksplit_p(i, j, n, m)$

altfel

$blocksplit_{\frac{p}{2}+p\%2}\left(i, j, n, \frac{m}{2}\right) \subseteq blocksplit_p(i, j, n, m)$

$blocksplit_{\frac{p}{2}}\left(i + \frac{n}{2}, j + \frac{m}{2}, n, m - \frac{m}{2}\right) \subseteq blocksplit_p(i, j, n, m)$

end

Impartirea elementelor pe threaduri

Pentru impartirea pe linii, se imparte intervalul $[0, N)$ in p subintervale folosind $intsplit_p(N)$, iar pentru fiecare subinterval se itereaza indicia coloanelor si se aplica o operatie atomica $conv(K, A, R, i, j)$ care scrie in $R_{i,j}$ rezultatul convolutiei matricei A cu nucleul K in pozitia (i, j).

Analog pentru impartirea pe coloane, doar ca indicii coloanelor se impart pe threaduri, in timp ce liniile sunt iterate complet.

In impartirea pe blocuri, se aplica $blocksplit_p(0,0,N,M)$ si pentru fiecare pozitie (i_1, j_1) a fiecarei submatrici dreptunghiulare (i, j, n', m'), se aplica operatia $conv(K, A, R, i_1, j_1)$.

Pentru functiile de distributie, mai intai se considera variant linearizata a matricei. Pentru un indice $(i, j), i = \overline{0..N-1}, j = \overline{0..M-1}$, consideram un indice $k = i * M + j$ corespunzator pozitiei elementului in matricea linearizata. Pentru delinearea matricei cunoscand $k = \overline{0..NM-1}$, se obtin indicii bidimensionali in matrice folosind formulele $i = \frac{k}{M}, j = k\%M$.

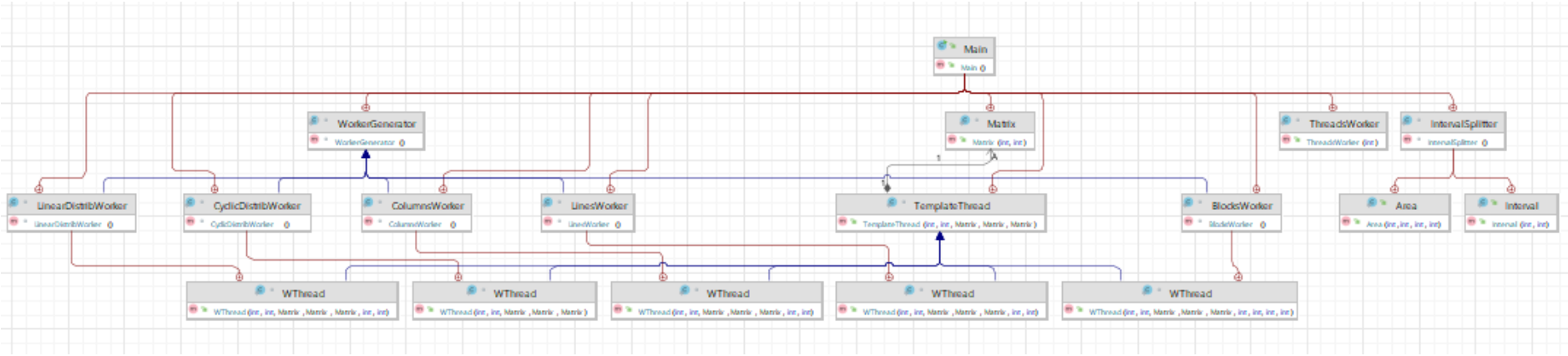
Consideram acum sirul A' de lungime MN rezultat in urma linearizarii matricei A.

In distributia lineara, se aplica $intsplit_p(MN)$ pentru a obtine intervalele atribuite fiecarui thread ce sunt iterate in mod obisnuit:

pentru $k = s \ldots e$ do $conv(K, A, R, \frac{k}{M}, k\%M)$.

In distributia ciclica, fiecare thread isi foloseste un id $id = \overline{0..p-1}$ si parcurge sirul A' incepand de la i cu pasul p:

pentru $k = id, k < MN, k \coloneqq k + p$ do $conv(K, A, R, \frac{k}{M}, k\%m)$


Main
Main ()
WorkerGenerator
WorkerGenerator ()
Matrix
Matrix (int, int)
ThreadsWorker
ThreadsWorker (int)
IntervalSplitter
IntervalSplitter ()
LinearDistribWorker
LinearDistribWorker ()
CyclicDistribWorker
CyclicDistribWorker ()
ColumnsWorker
ColumnsWorker ()
LinesWorker
LinesWorker ()
TemplateThread
TemplateThread (int, int, Matrix, Matrix, Matrix)
BlocksWorker
BlocksWorker ()
Area
Area (int, int, int, int)
Interval
Interval (int, int)
WThread
WThread (int, int, Matrix, Matrix, Matrix, int, int)
WThread
WThread (int, int, Matrix, Matrix, Matrix)
WThread
WThread (int, int, Matrix, Matrix, Matrix, int, int)
WThread
WThread (int, int, Matrix, Matrix, Matrix, int, int)
WThread
WThread (int, int, Matrix, Matrix, Matrix, int, int, int, int)