TFG - Benchmark

Mª Dolors Pelegrí

9/3/2019

Producte de matrius

Producte de matrius per blocs

El rendiment en el producte de matrius per blocs s'ha fet de dues formes diferents, per una banda, s'ha tingut en compte el tamany de bloc a l'hora de calcular el rendiment i posteriorment s'ha tingut en compte el tamany de la matriu, per a calcular el rendiment del tamany de matriu s'han tingut en compte els resultats del tamany del bloc obtinguts previament per agafar un tamany més òptim.

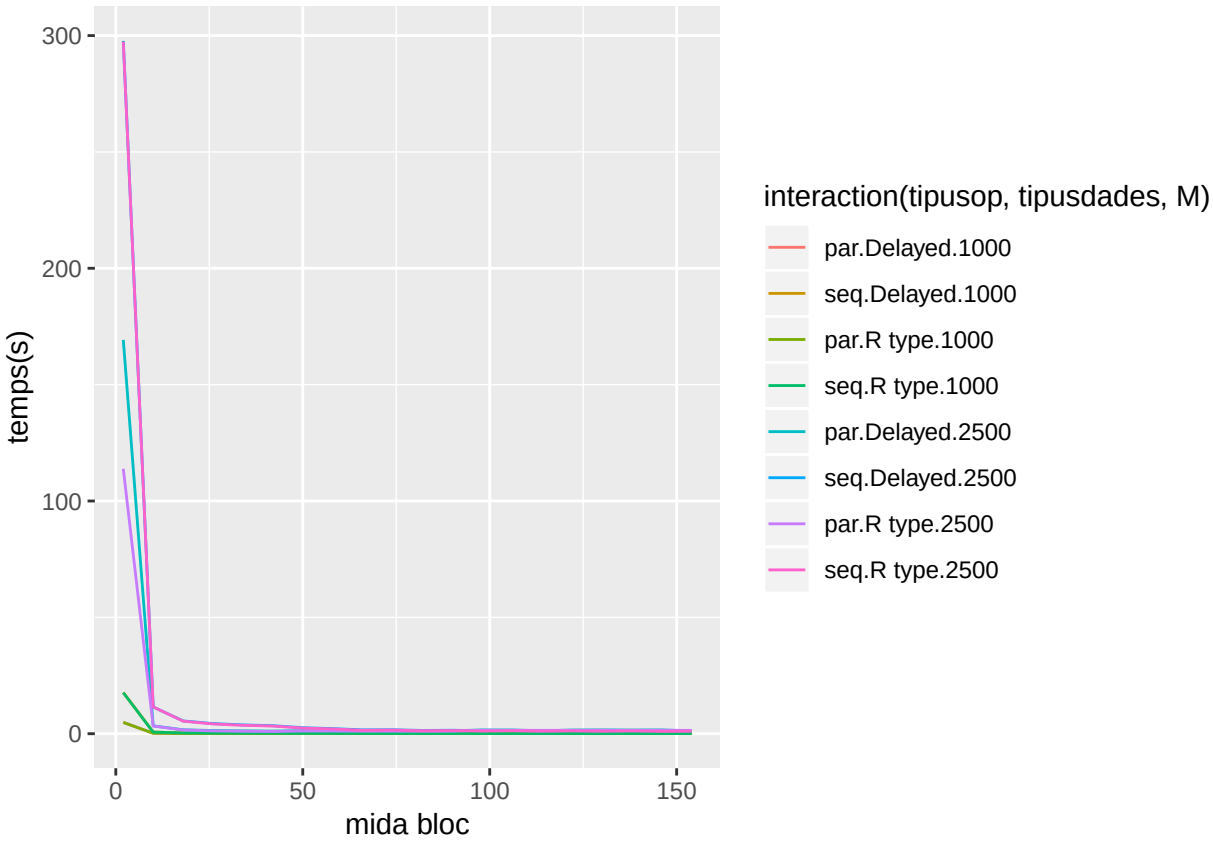
Rendiment obtingut en el càlcul de producte de matrius tenint en compte el tamany del bloc, les proves s'han realitzat amb matrius quadrades (nxn) amb n=1000 i n=2500.

Producte de matrius per mida de blocs

Els càlculs s'han realitzat en blocs quadrats de (k x k) agafant k de 2 a 160 amb increments de 8 (n+8,k+8). A la funció blockmult, el paràmetre bparal indica si l'execució s'ha de fer en paral · lel o no, AD i BD indiquen dades del tipus DelayedArray.

Table 1: Temps mínims registrats amb producte matrius per mida matriu i mida de bloc

	expr	min	mean	max	tipusdades	bloc	M	ncores
65	blockmult(A, B, i, bparal)	0.0572	0.0572	0.0572	R type	130	1000	4
66	blockmult(AD, BD, i, bparal)	0.0780	0.0780	0.0780	Delayed	130	1000	4
75	blockmult(A, B, i, bparal)	0.0686	0.0686	0.0686	R type	146	1000	4
76	blockmult(AD, BD, i, bparal)	0.0917	0.0917	0.0917	Delayed	146	1000	4
101	blockmult(A, B, i, bparal)	0.9620	0.9620	0.9620	R type	42	2500	4
102	blockmult(AD, BD, i, bparal)	1.1288	1.1288	1.1288	Delayed	42	2500	4
155	blockmult(A, B, i, bparal)	1.0014	1.0014	1.0014	R type	146	2500	4
156	blockmult(AD, BD, i, bparal)	1.1674	1.1674	1.1674	Delayed	146	2500	4



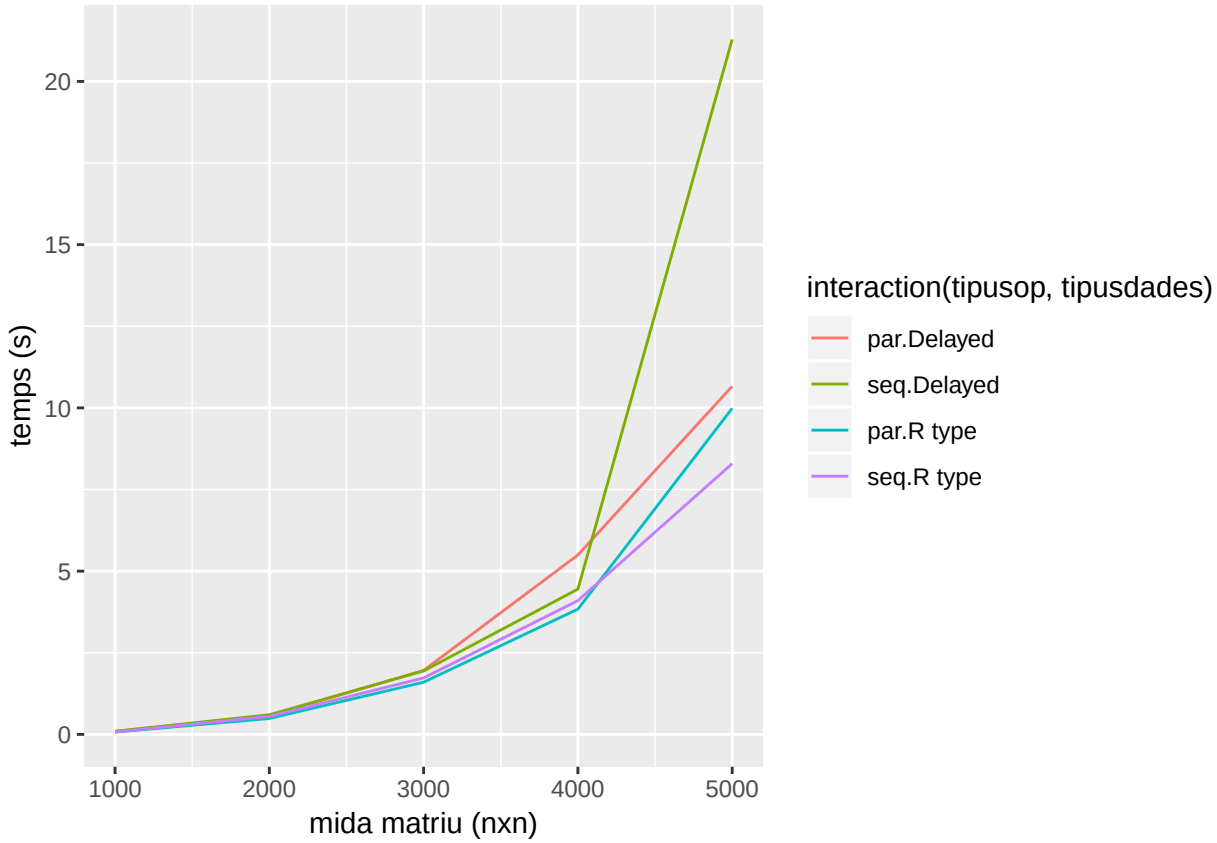
Producte de matrius per mida total de les matrius

En aquest cas s'han tingut en compte blocs d'un tamany de 128 per a realitzar els càlculs, els resultats, depenent de la mida del bloc poden variar molt, un dels grans problemes es trobar la mida òptima del bloc per poder obtenir un rendiment eficient en les operacions amb matrius.

Table 2: Temps registrats producte matrius mida bloc = 128

expr	min	mean	max	tipusdades	bloc	M	ncores
blockmult(A, B, bloc_size, bparal)	0.0588	0.0703	0.0818	R type	1000	1000	4
blockmult(AD, BD, bloc_size, bparal)	0.0782	0.0890	0.0997	Delayed	1000	1000	4
blockmult(A, B, bloc_size, bparal)	0.0689	0.0735	0.0781	R type	1000	1000	4
blockmult(AD, BD, bloc_size, bparal)	0.0888	0.0955	0.1022	Delayed	1000	1000	4
blockmult(A, B, bloc_size, bparal)	0.4179	0.4871	0.5564	R type	2000	2000	4
blockmult(AD, BD, bloc_size, bparal)	0.5124	0.5770	0.6417	Delayed	2000	2000	4
blockmult(A, B, bloc_size, bparal)	0.5217	0.5514	0.5811	R type	2000	2000	4
blockmult(AD, BD, bloc_size, bparal)	0.5941	0.6011	0.6080	Delayed	2000	2000	4
blockmult(A, B, bloc_size, bparal)	1.3889	1.5979	1.8068	R type	3000	3000	4
blockmult(AD, BD, bloc_size, bparal)	1.9421	1.9587	1.9752	Delayed	3000	3000	4
blockmult(A, B, bloc_size, bparal)	1.7222	1.7282	1.7342	R type	3000	3000	4
blockmult(AD, BD, bloc_size, bparal)	1.9420	1.9429	1.9439	Delayed	3000	3000	4
blockmult(A, B, bloc_size, bparal)	3.2731	3.8368	4.4005	R type	4000	4000	4
blockmult(AD, BD, bloc_size, bparal)	4.6598	5.4949	6.3300	Delayed	4000	4000	4
blockmult(A, B, bloc_size, bparal)	4.0687	4.0964	4.1241	R type	4000	4000	4
blockmult(AD, BD, bloc_size, bparal)	4.4444	4.4492	4.4540	Delayed	4000	4000	4

expr	min	mean	max	tipusdades	bloc	M	ncores
blockmult(A, B, bloc_size, bparal)	8.1119	9.9871	11.8623	R type	5000	5000	4
blockmult(AD, BD, bloc_size, bparal)	8.6879	10.6616	12.6353	Delayed	5000	5000	4
blockmult(A, B, bloc_size, bparal)	8.2195	8.2916	8.3636	R type	5000	5000	4
blockmult(AD, BD, bloc_size, bparal)	19.1128	21.2864	23.4600	Delayed	5000	5000	4



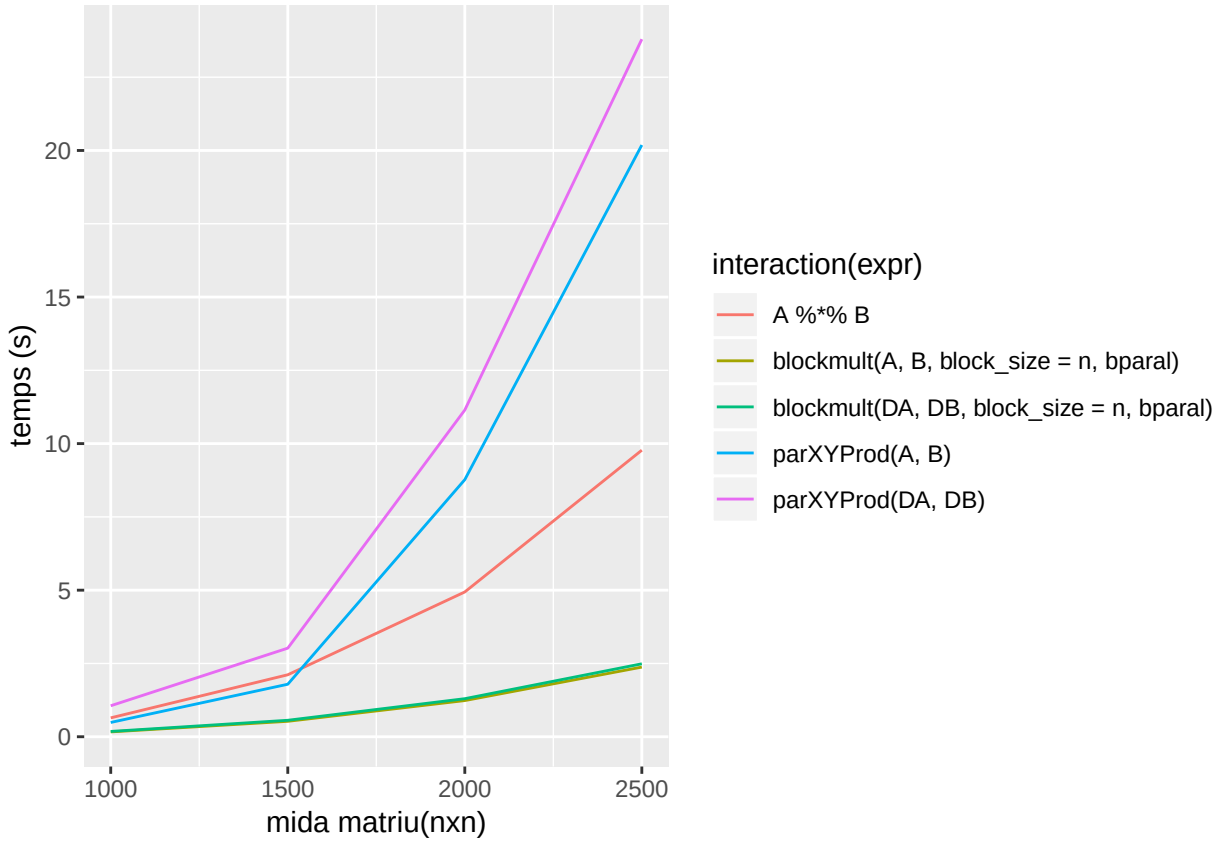
Producte de matrius sense blocs :

Per a realitzar el càlculs s'han utilitzat la funció de producte de matrius pròpia de R (%*%) que només funciona amb tipus de dades d'r, una funció realitzada a partir de la llibreria RcppParallel (parXYprod), pot utilitzar DelayedArrays o RObjects (tipus de dades de R), i la funció realitzada utilitzant omp per paral · lelitzar (blockmult), aquesta funció també pot treballar amb RObjects i DelayedArrays, amb aquesta funció es realitzant multiplicacions per blocs però si la mida del bloc=n, llavors tracta la matriu com un únic bloc i no paral · lelitza per blocs.

Table 3: Temps registrats producte matrius sense blocs

expr	min	mean	max	tipusdades	M	ncores
parXYProd(A, B)	0.4568	0.4871	0.5174	R type	1000	4
blockmult(A, B, block_size = n, bparal)	0.1574	0.1651	0.1728	R type	1000	4
parXYProd(DA, DB)	0.9649	1.0571	1.1494	Delayed	1000	4
blockmult(DA, DB, block_size = n, bparal)	0.1775	0.1808	0.1841	Delayed	1000	4
A %*% B	0.6382	0.6436	0.6490	R type	1000	4
parXYProd(A, B)	1.7618	1.7914	1.8210	R type	1500	4

expr	min	mean	max	tipusdades	M	ncores
blockmult(A, B, block_size = n, bparal)	0.5198	0.5267	0.5336	R type	1500	4
parXYProd(DA, DB)	2.9577	3.0216	3.0855	Delayed	1500	4
blockmult(DA, DB, block_size = n, bparal)	0.5573	0.5612	0.5650	Delayed	1500	4
A %*% B	2.1076	2.1145	2.1213	R type	1500	4
parXYProd(A, B)	8.7278	8.7711	8.8145	R type	2000	4
blockmult(A, B, block_size = n, bparal)	1.2320	1.2346	1.2372	R type	2000	4
parXYProd(DA, DB)	11.1029	11.1461	11.1894	Delayed	2000	4
blockmult(DA, DB, block_size = n, bparal)	1.2918	1.2977	1.3035	Delayed	2000	4
A %*% B	4.9347	4.9386	4.9425	R type	2000	4
parXYProd(A, B)	20.0929	20.1860	20.2791	R type	2500	4
blockmult(A, B, block_size = n, bparal)	2.3746	2.3759	2.3771	R type	2500	4
parXYProd(DA, DB)	23.7258	23.7970	23.8681	Delayed	2500	4
blockmult(DA, DB, block_size = n, bparal)	2.4870	2.4877	2.4883	Delayed	2500	4
A %*% B	9.7728	9.7764	9.7800	R type	2500	4

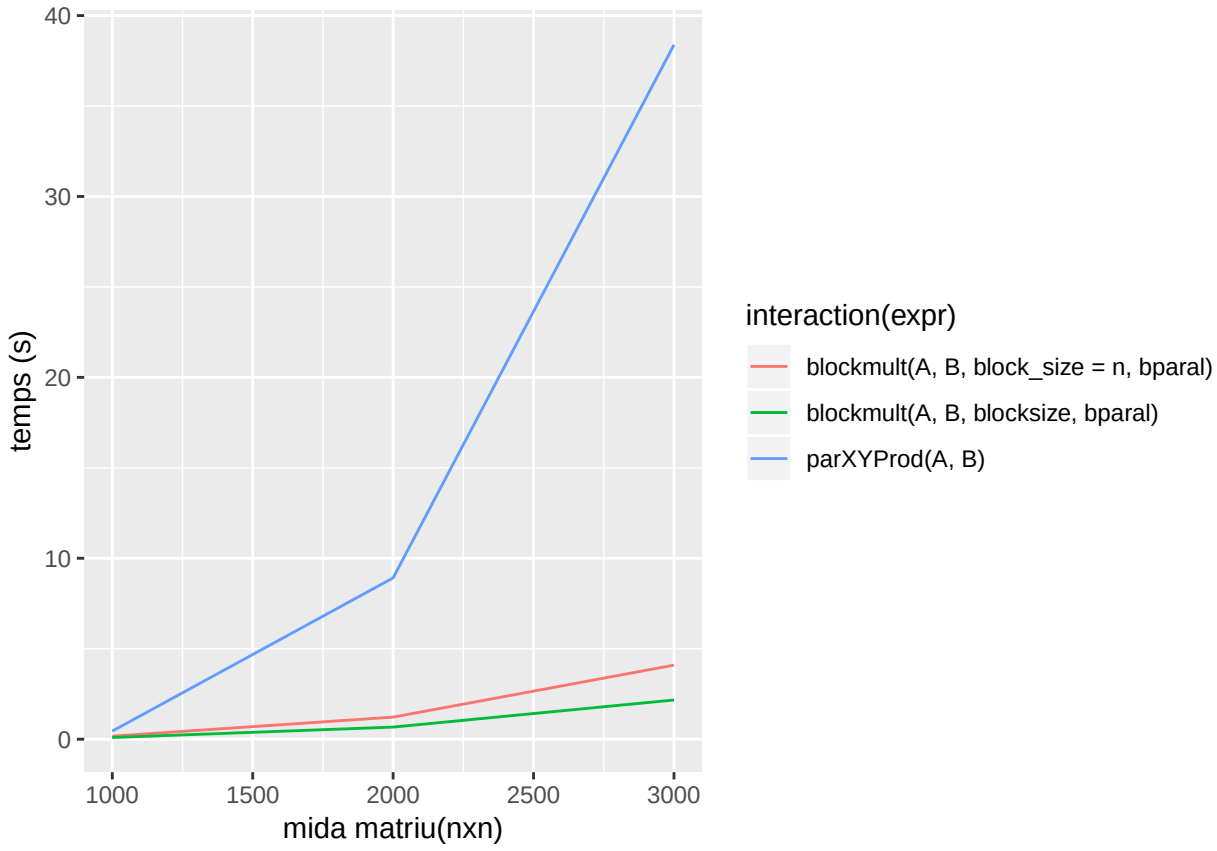


Producte de matrius en paral · lel per blocs vs no blocs :

En aquest cas s'han fet proves de rendiment multiplicant matrius de 1000x1000, 2000x2000 i 3000x3000 amb càlculs paral · lelitzats utilitzant funcions utilitznat Rcppparallel (funció parXYProd) i omp (funció blockmult()), s'han utilitzat blocs de mida 128 i blocs de mida 0 (mida bloc = n) per realitzar les operacions sense blocs.

Table 4: Temps registrats producte matrius per blocs i sense blocs

expr	min	mean	max	tipusop	tipusdades	bloc	M	ncores
blockmult(A, B, blocksize, bparal)	0.0793	0.0910	0.1027	par	R type	128	1000	4
parXYProd(A, B)	0.4218	0.4448	0.4679	par	R type	0	1000	4
blockmult(A, B, block_size = n, bparal)	0.1648	0.1659	0.1669	par	R type	0	1000	4
blockmult(A, B, blocksize, bparal)	0.5578	0.6677	0.7776	par	R type	128	2000	4
parXYProd(A, B)	8.8707	8.9148	8.9588	par	R type	0	2000	4
blockmult(A, B, block_size = n, bparal)	1.2177	1.2184	1.2191	par	R type	0	2000	4
blockmult(A, B, blocksize, bparal)	1.7726	2.1648	2.5569	par	R type	128	3000	4
parXYProd(A, B)	38.2629	38.3835	38.5041	par	R type	0	3000	4
blockmult(A, B, block_size = n, bparal)	3.9786	4.0924	4.2062	par	R type	0	3000	4



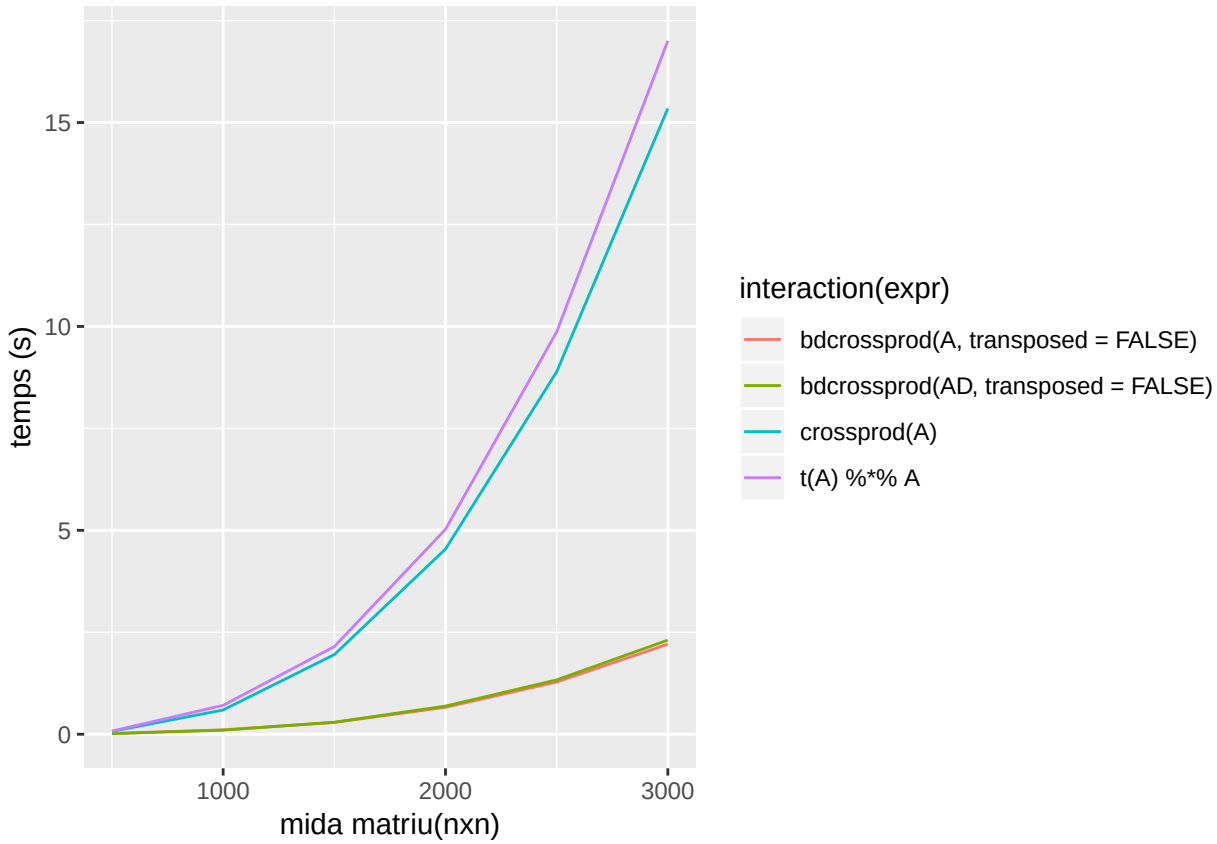
Crossprod i tCrossprod

Els càlculs s'han realitzat per diferents mides de matrius, tant per tipus de dades de R com per DelayedArray. Per realitzar la comparativa, tant per crossprod com per al tcrossprod s'han utilitzat les funcions de R, crossprod / tcrossprod i l'operador de producte de matrius i s'han comparat amb la funció bdcrossprod creada a partir de les llibreries d'Eigen, aquesta funció accepta RObjects i DelayedArrays per a fer els càlculs i calcula el crossprod i el tcrossprod a partir del paràmetre transposed, si transpoded = TRUE calcula tcrosspod, si transposed=FALSE calcula el crossprod.

Crossprod

Table 5: Temps registrats càlcul crossprod

expr	min	mean	max	tipusdades	M	ncores
bdcrossprod(A, transposed = FALSE)	0.0137	0.0166	0.0212	R type	500	4
crossprod(A)	0.0675	0.0695	0.0716	R type	500	4
t(A) %*% A	0.0739	0.0766	0.0797	R type	500	4
bdcrossprod(AD, transposed = FALSE)	0.0141	0.0147	0.0151	Delayed	500	4
bdcrossprod(A, transposed = FALSE)	0.0970	0.1073	0.1248	R type	1000	4
crossprod(A)	0.5783	0.5940	0.6052	R type	1000	4
t(A) %*% A	0.6582	0.7082	0.8415	R type	1000	4
bdcrossprod(AD, transposed = FALSE)	0.0992	0.1000	0.1020	Delayed	1000	4
bdcrossprod(A, transposed = FALSE)	0.2819	0.2889	0.2978	R type	1500	4
crossprod(A)	1.8980	1.9517	2.0242	R type	1500	4
t(A) %*% A	2.1061	2.1483	2.2458	R type	1500	4
bdcrossprod(AD, transposed = FALSE)	0.2911	0.2941	0.2957	Delayed	1500	4
bdcrossprod(A, transposed = FALSE)	0.6408	0.6605	0.6827	R type	2000	4
crossprod(A)	4.5311	4.5404	4.5605	R type	2000	4
t(A) %*% A	4.9746	5.0250	5.1396	R type	2000	4
bdcrossprod(AD, transposed = FALSE)	0.6831	0.6899	0.6974	Delayed	2000	4
bdcrossprod(A, transposed = FALSE)	1.2555	1.2801	1.3081	R type	2500	4
crossprod(A)	8.8659	8.8864	8.9080	R type	2500	4
t(A) %*% A	9.8514	9.8652	9.8907	R type	2500	4
bdcrossprod(AD, transposed = FALSE)	1.3257	1.3315	1.3398	Delayed	2500	4
bdcrossprod(A, transposed = FALSE)	2.1820	2.2073	2.2421	R type	3000	4
crossprod(A)	15.3255	15.3456	15.3617	R type	3000	4
t(A) %*% A	16.9925	17.0024	17.0237	R type	3000	4
bdcrossprod(AD, transposed = FALSE)	2.2926	2.3074	2.3255	Delayed	3000	4

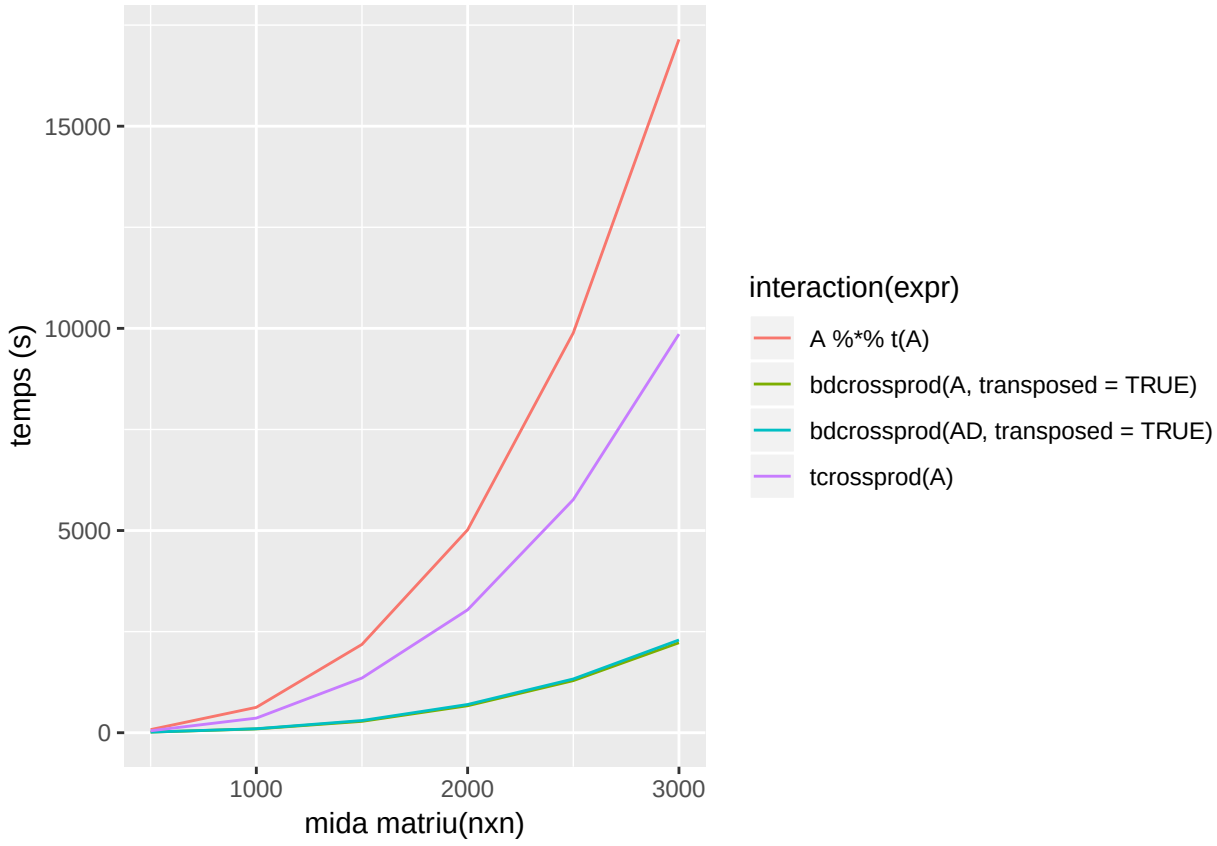


tCrossprod

Table 6: Temps registrats càlcul tcrossprod

expr	min	mean	max	tipusdades	M	ncores
bdcrossprod(A, transposed = TRUE)	13.1586	15.9712	21.9401	R type	500	4
tcrossprod(A)	42.6117	48.2438	60.6567	R type	500	4
A %*% t(A)	75.2437	77.2839	79.5016	R type	500	4
bdcrossprod(AD, transposed = TRUE)	15.0471	15.6180	15.8664	Delayed	500	4
bdcrossprod(A, transposed = TRUE)	89.7039	94.2731	98.4204	R type	1000	4
tcrossprod(A)	349.7606	361.2944	372.1673	R type	1000	4
A %*% t(A)	618.0770	627.7003	637.2499	R type	1000	4
bdcrossprod(AD, transposed = TRUE)	95.0993	98.4650	100.4347	Delayed	1000	4
bdcrossprod(A, transposed = TRUE)	271.0843	281.6320	292.3987	R type	1500	4
tcrossprod(A)	1254.3056	1352.6447	1635.7674	R type	1500	4
A %*% t(A)	2107.2427	2183.8993	2384.3398	R type	1500	4
bdcrossprod(AD, transposed = TRUE)	292.7833	301.4990	315.6320	Delayed	1500	4
bdcrossprod(A, transposed = TRUE)	639.7904	667.2225	711.7308	R type	2000	4
tcrossprod(A)	2979.8720	3039.7023	3144.9475	R type	2000	4
A %*% t(A)	4988.1765	5018.0478	5064.8900	R type	2000	4
bdcrossprod(AD, transposed = TRUE)	695.7982	698.5771	705.5663	Delayed	2000	4
bdcrossprod(A, transposed = TRUE)	1263.3234	1288.5586	1314.5370	R type	2500	4
tcrossprod(A)	5757.3092	5765.4323	5787.0203	R type	2500	4
A %*% t(A)	9810.3975	9888.4310	9985.7562	R type	2500	4
bdcrossprod(AD, transposed = TRUE)	1324.8928	1329.6482	1342.5978	Delayed	2500	4

expr	min	mean	max	tipusdades	M	ncores
bdcrossprod(A, transposed = TRUE)	2167.5651	2225.6863	2252.0050	R type	3000	4
tcrossprod(A)	9811.5347	9858.4695	9922.3834	R type	3000	4
A %*% t(A)	16984.6352	17146.5941	17338.3354	R type	3000	4
bdcrossprod(AD, transposed = TRUE)	2287.7379	2293.7331	2304.0454	Delayed	3000	4



Crossprod i tCrossprod amb pesos (xtwx / xwxt)

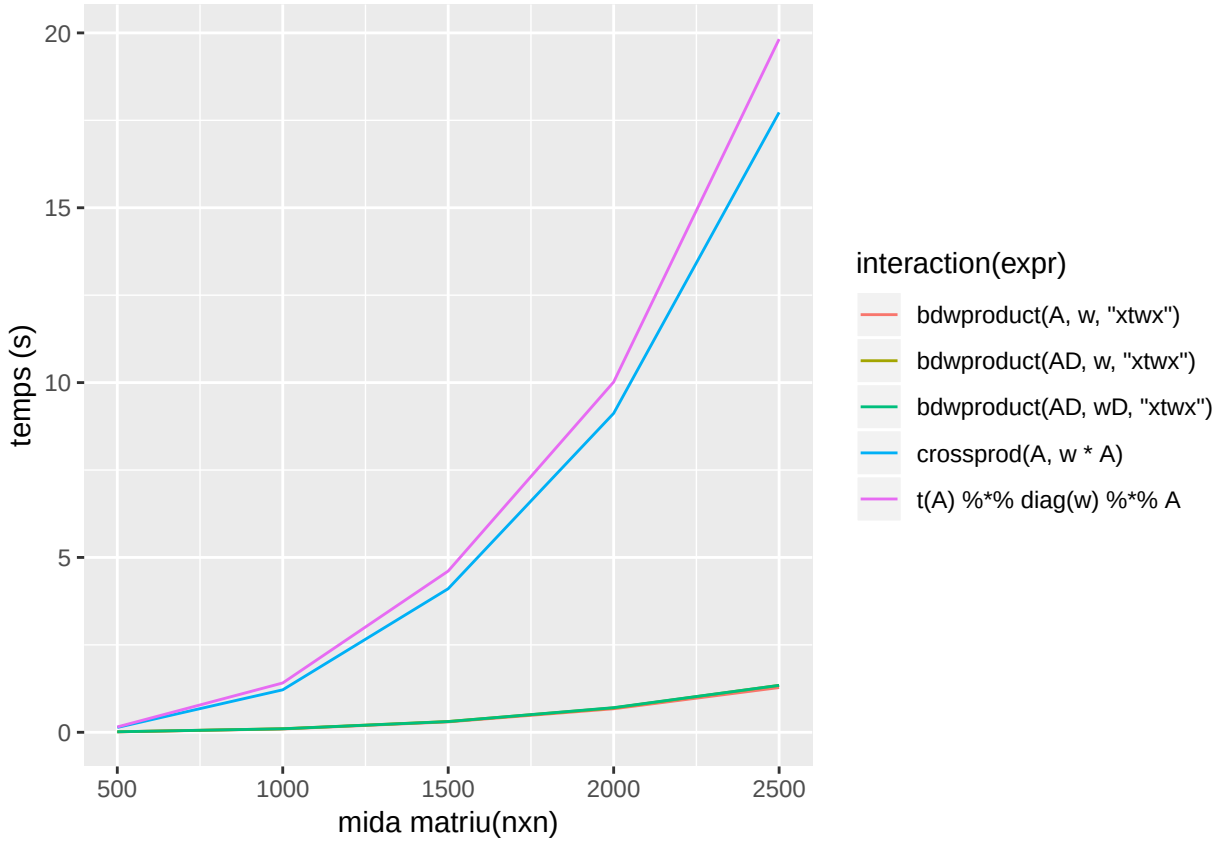
En aquest cas es calculen els temps de càlcul per al producte d'una matriu per una matriu diagonal de pesos, les funcions que es comparen son les funcions de r de producte de matrius %*% i la funció bdwproduct programada amb les llibreries d'Eigen, aquesta funció, accepta tipus de dades RObject i DelayedArray i pot fer els dos càlculs passant com a paràmetre el tipus d'operació que es vol realitzar amb la matriu i el vector de pesos (xwxt o xtwx).

Crossprod amb pesos (xtwx)

Table 7: Temps registrats càlcul tcrossprod per matriu diagonal de pesos

expr	min	mean	max	tipusdades	M	Y	ncores
bdwproduct(A, w, "xtwx")	0.0115	0.0136	0.0153	R type	500	500	4
crossprod(A, w * A)	0.1339	0.1377	0.1476	R type	500	500	4

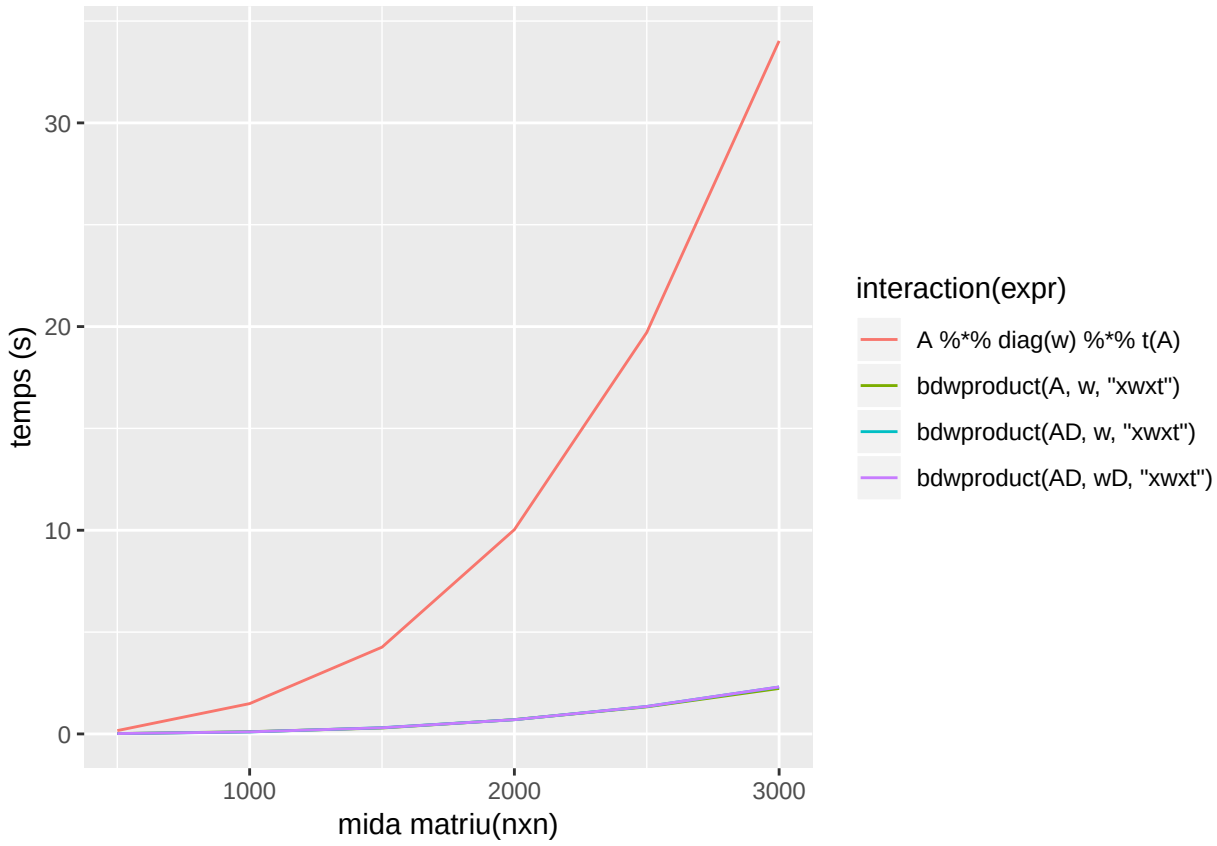
expr	min	mean	max	tipusdades	M	Y	ncores
t(A) %% *diag(w)* %% A	0.1450	0.1533	0.1621	R type	500	500	4
bdwproduct(AD, w, “xtwx”)	0.0131	0.0134	0.0139	Delayed	500	500	4
bdwproduct(AD, wD, “xtwx”)	0.0147	0.0151	0.0158	Delayed	500	500	4
bdwproduct(A, w, “xtwx”)	0.0836	0.0955	0.1052	R type	1000	1000	4
crossprod(A, w * A)	1.1265	1.2150	1.4631	R type	1000	1000	4
t(A) %% *diag(w)* %% A	1.2523	1.4106	1.7607	R type	1000	1000	4
bdwproduct(AD, w, “xtwx”)	0.0941	0.1014	0.1187	Delayed	1000	1000	4
bdwproduct(AD, wD, “xtwx”)	0.0954	0.1000	0.1086	Delayed	1000	1000	4
bdwproduct(A, w, “xtwx”)	0.2759	0.2974	0.3137	R type	1500	1500	4
crossprod(A, w * A)	3.8501	4.1060	4.3864	R type	1500	1500	4
t(A) %% *diag(w)* %% A	4.3045	4.6107	5.1300	R type	1500	1500	4
bdwproduct(AD, w, “xtwx”)	0.2975	0.3086	0.3268	Delayed	1500	1500	4
bdwproduct(AD, wD, “xtwx”)	0.2984	0.3064	0.3167	Delayed	1500	1500	4
bdwproduct(A, w, “xtwx”)	0.6469	0.6727	0.6974	R type	2000	2000	4
crossprod(A, w * A)	9.0203	9.1216	9.4020	R type	2000	2000	4
t(A) %% *diag(w)* %% A	9.9089	10.0173	10.1359	R type	2000	2000	4
bdwproduct(AD, w, “xtwx”)	0.6893	0.6975	0.7159	Delayed	2000	2000	4
bdwproduct(AD, wD, “xtwx”)	0.6907	0.7044	0.7373	Delayed	2000	2000	4
bdwproduct(A, w, “xtwx”)	1.2632	1.2790	1.3159	R type	2500	2500	4
crossprod(A, w * A)	17.6512	17.7270	17.9009	R type	2500	2500	4
t(A) %% *diag(w)* %% A	19.5934	19.8228	20.3304	R type	2500	2500	4
bdwproduct(AD, w, “xtwx”)	1.3308	1.3424	1.3594	Delayed	2500	2500	4
bdwproduct(AD, wD, “xtwx”)	1.3326	1.3417	1.3597	Delayed	2500	2500	4



tCrossprod amb pesos (xwxt)

Table 8: Temps registrats càlcul tcrossprod per matriu diagonal de pesos

expr	min	mean	max	tipusdades	M	Y	ncores
bdwproduct(A, w, "xwxt")	0.0148	0.0174	0.0232	R type	500	500	4
A %% *diag(w)* %% t(A)	0.1487	0.1637	0.1788	R type	500	500	4
bdwproduct(AD, w, "xwxt")	0.0143	0.0150	0.0158	Delayed	500	500	4
bdwproduct(AD, wD, "xwxt")	0.0161	0.0170	0.0180	Delayed	500	500	4
bdwproduct(A, w, "xwxt")	0.0964	0.1089	0.1242	R type	1000	1000	4
A %% *diag(w)* %% t(A)	1.2440	1.4881	1.8427	R type	1000	1000	4
bdwproduct(AD, w, "xwxt")	0.0966	0.0976	0.0981	Delayed	1000	1000	4
bdwproduct(AD, wD, "xwxt")	0.0989	0.1022	0.1078	Delayed	1000	1000	4
bdwproduct(A, w, "xwxt")	0.2759	0.2920	0.3060	R type	1500	1500	4
A %% *diag(w)* %% t(A)	4.2107	4.2597	4.3589	R type	1500	1500	4
bdwproduct(AD, w, "xwxt")	0.2987	0.3041	0.3118	Delayed	1500	1500	4
bdwproduct(AD, wD, "xwxt")	0.2998	0.3019	0.3060	Delayed	1500	1500	4
bdwproduct(A, w, "xwxt")	0.6523	0.7032	0.8338	R type	2000	2000	4
A %% *diag(w)* %% t(A)	9.9399	10.0298	10.1258	R type	2000	2000	4
bdwproduct(AD, w, "xwxt")	0.6940	0.7004	0.7186	Delayed	2000	2000	4
bdwproduct(AD, wD, "xwxt")	0.6965	0.6991	0.7048	Delayed	2000	2000	4
bdwproduct(A, w, "xwxt")	1.2762	1.3285	1.4782	R type	2500	2500	4
A %% *diag(w)* %% t(A)	19.6382	19.7111	19.8157	R type	2500	2500	4
bdwproduct(AD, w, "xwxt")	1.3408	1.3478	1.3547	Delayed	2500	2500	4
bdwproduct(AD, wD, "xwxt")	1.3424	1.3484	1.3540	Delayed	2500	2500	4
bdwproduct(A, w, "xwxt")	2.1829	2.2321	2.2985	R type	3000	3000	4
A %% *diag(w)* %% t(A)	33.8590	34.0255	34.2141	R type	3000	3000	4
bdwproduct(AD, w, "xwxt")	2.2928	2.3030	2.3099	Delayed	3000	3000	4
bdwproduct(AD, wD, "xwxt")	2.2962	2.3070	2.3168	Delayed	3000	3000	4



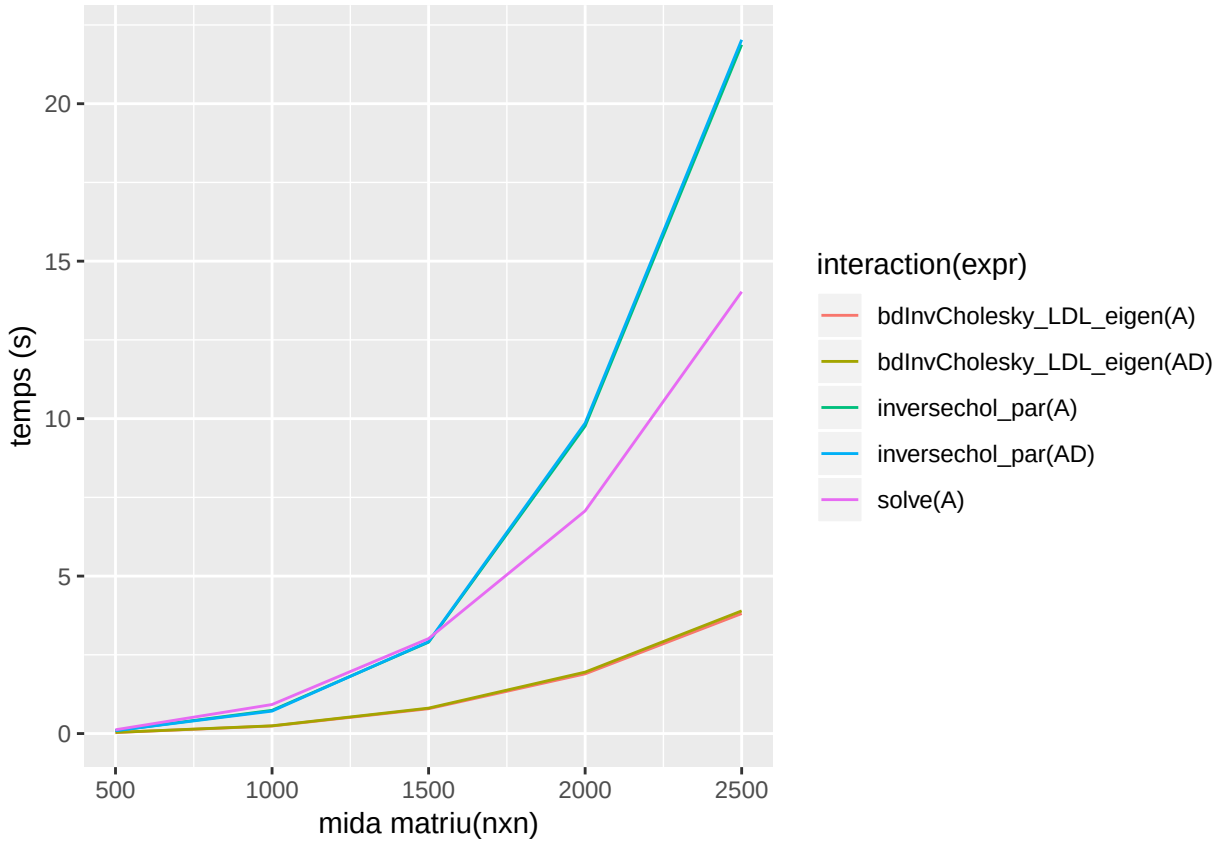
Inversa Cholesky

Les funciones comparades és la funció solve de R i les funcions bdInvCholesky_LDL_Eigen, realitzada amb les funcions de la llibreria Eigen i la funció inverse_chol_par, realitzada amb un algoritme paral · lelitzat amb omp.

Table 9: Temps registrats càlcul matriu Inversa Cholesky

expr	min	mean	max	tipusdades	M	ncores
inversechol_par(A)	0.0876	0.0916	0.1095	R type	500	4
bdInvCholesky_LDL_eigen(A)	0.0311	0.0334	0.0365	R type	500	4
solve(A)	0.1077	0.1195	0.1849	R type	500	4
inversechol_par(AD)	0.0903	0.0915	0.0928	Delayed	500	4
bdInvCholesky_LDL_eigen(AD)	0.0354	0.0360	0.0368	Delayed	500	4
inversechol_par(A)	0.6888	0.7376	0.8276	R type	1000	4
bdInvCholesky_LDL_eigen(A)	0.2302	0.2403	0.2514	R type	1000	4
solve(A)	0.8173	0.9222	1.0566	R type	1000	4
inversechol_par(AD)	0.6853	0.7124	0.8279	Delayed	1000	4
bdInvCholesky_LDL_eigen(AD)	0.2348	0.2450	0.2902	Delayed	1000	4
inversechol_par(A)	2.8530	2.9009	3.0595	R type	1500	4
bdInvCholesky_LDL_eigen(A)	0.7734	0.7880	0.8172	R type	1500	4
solve(A)	2.9633	3.0148	3.0982	R type	1500	4
inversechol_par(AD)	2.8692	2.9184	2.9894	Delayed	1500	4
bdInvCholesky_LDL_eigen(AD)	0.7970	0.8074	0.8302	Delayed	1500	4
inversechol_par(A)	9.6903	9.7622	9.8509	R type	2000	4

expr	min	mean	max	tipusdades	M	ncores
bdInvCholesky_LDL_eigen(A)	1.8846	1.8955	1.9039	R type	2000	4
solve(A)	7.0343	7.0700	7.1651	R type	2000	4
inversechol_par(AD)	9.6387	9.8587	10.3300	Delayed	2000	4
bdInvCholesky_LDL_eigen(AD)	1.9297	1.9507	2.0063	Delayed	2000	4
inversechol_par(A)	21.3968	21.8740	23.1140	R type	2500	4
bdInvCholesky_LDL_eigen(A)	3.7365	3.8094	4.0714	R type	2500	4
solve(A)	13.6488	14.0291	14.8522	R type	2500	4
inversechol_par(AD)	21.5569	22.0298	22.8496	Delayed	2500	4
bdInvCholesky_LDL_eigen(AD)	3.7927	3.8946	4.0795	Delayed	2500	4

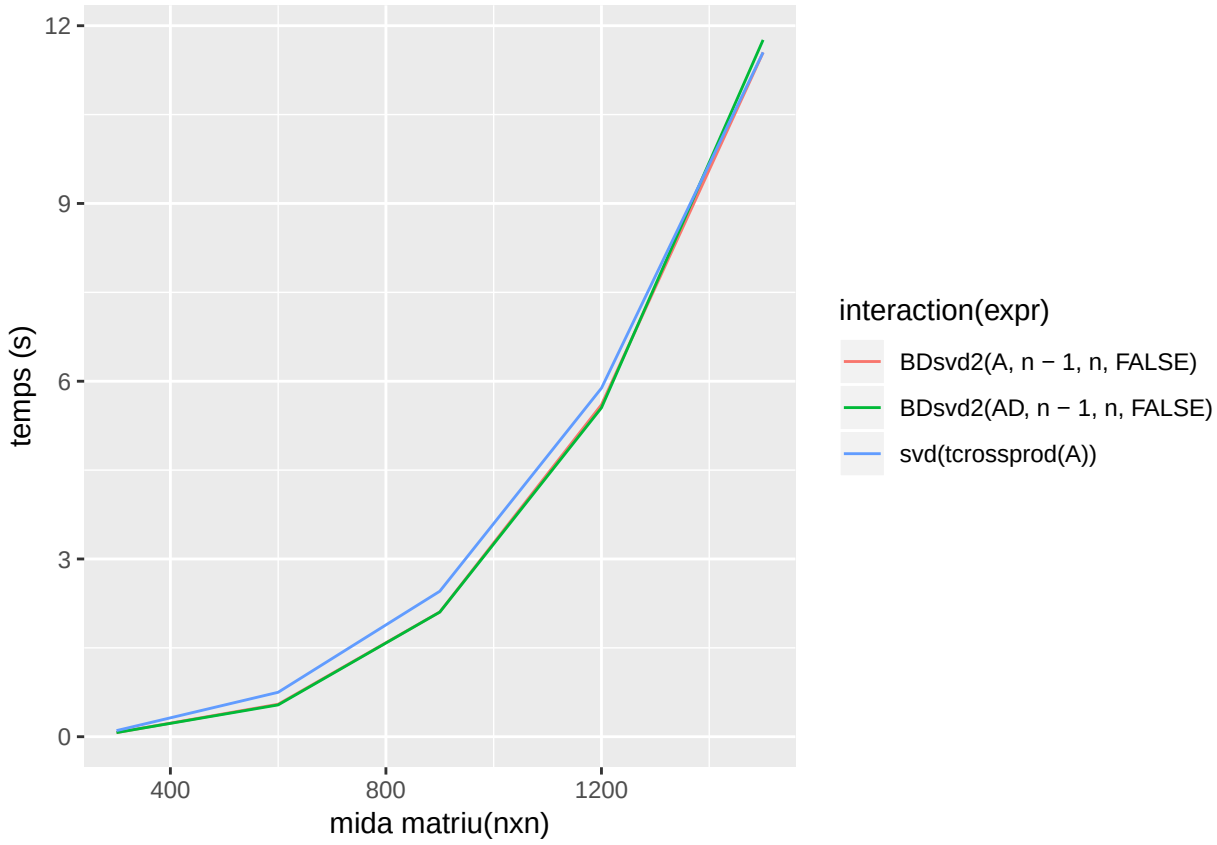


Descomposició SVD

Table 10: Temps registrats càlcul matriu Inversa Cholesky

expr	min	mean	max	tipusdades	M	ncores
BDsvd2(A, n - 1, n, FALSE)	0.0665	0.0699	0.0733	R type	300	4
svd(tcrossprod(A))	0.0995	0.1033	0.1071	R type	300	4
BDsvd2(AD, n - 1, n, FALSE)	0.0685	0.0694	0.0703	Delayed	300	4
BDsvd2(A, n - 1, n, FALSE)	0.5427	0.5482	0.5537	R type	600	4
svd(tcrossprod(A))	0.7499	0.7501	0.7502	R type	600	4
BDsvd2(AD, n - 1, n, FALSE)	0.5310	0.5372	0.5435	Delayed	600	4
BDsvd2(A, n - 1, n, FALSE)	2.0937	2.1050	2.1163	R type	900	4

expr	min	mean	max	tipusdades	M	ncores
svd(tcrossprod(A))	2.4522	2.4554	2.4587	R type	900	4
BDsvd2(AD, n - 1, n, FALSE)	2.0985	2.1034	2.1083	Delayed	900	4
BDsvd2(A, n - 1, n, FALSE)	5.5505	5.6034	5.6563	R type	1200	4
svd(tcrossprod(A))	5.8738	5.8829	5.8920	R type	1200	4
BDsvd2(AD, n - 1, n, FALSE)	5.5417	5.5513	5.5609	Delayed	1200	4
BDsvd2(A, n - 1, n, FALSE)	11.5003	11.5479	11.5955	R type	1500	4
svd(tcrossprod(A))	11.4955	11.5507	11.6060	R type	1500	4
BDsvd2(AD, n - 1, n, FALSE)	11.6634	11.7606	11.8577	Delayed	1500	4



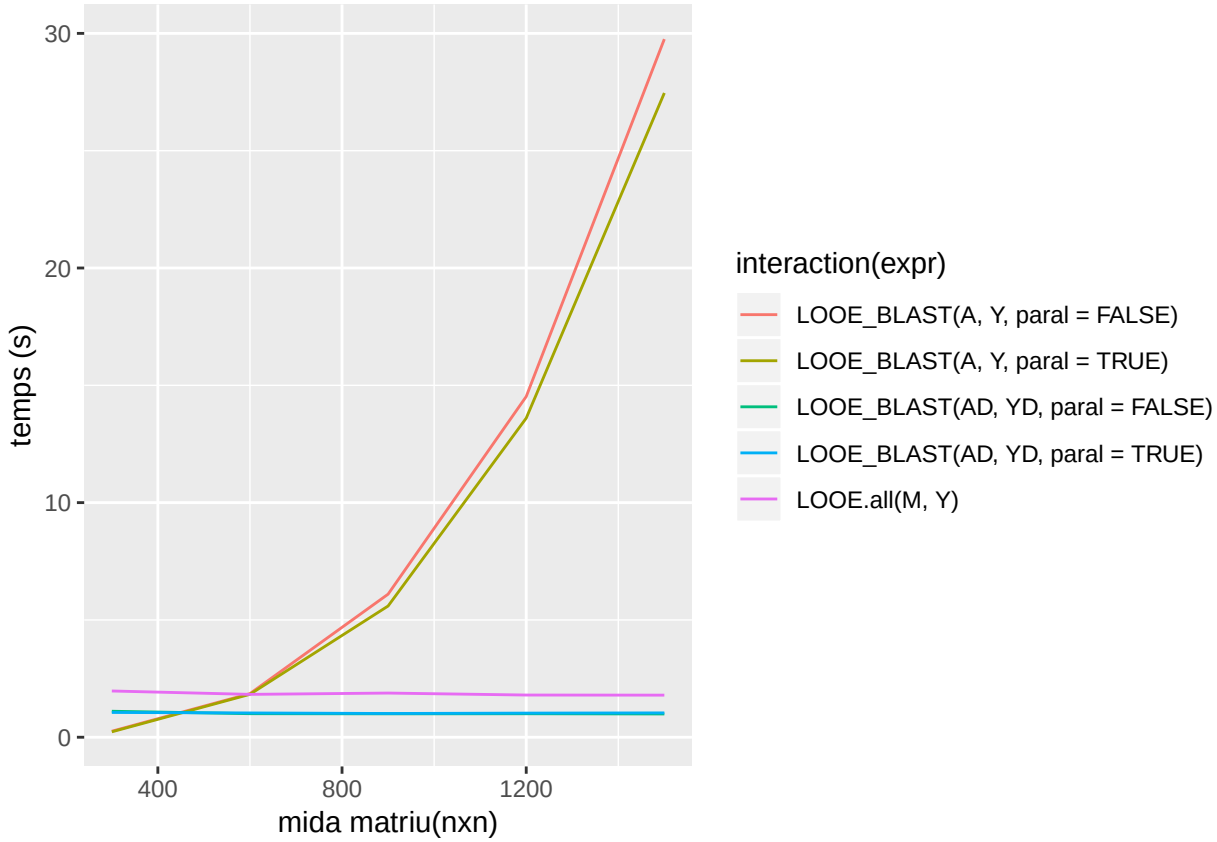
Càlcul LOOE

Les funcions utilitzades per avaluar el rendiment són les funcions ealitzades directament en R LOOE.all(M, Y) que agrupa les funcions LOOE() i solveEigen_orig(), en aquest cas s'han agrupat per poder tenir el temps total de comput. D'altra banda s'ha utilitzat la funció LOOE_BLAST que accepta tipus de dades DelayedArray i RObjects i es pot executar amb part del codi en paral · lel utilitzant les funcions testejades anteriorment.

Table 11: Temps registrats càlcul matriu Inversa Cholesky

expr	min	mean	max	tipusdades	M	Y	ncores
LOOE_BLAST(A, Y, paral = TRUE)	0.2204	0.2311	0.2417	R type	300	500	4
LOOE_BLAST(A, Y, paral = FALSE)	0.2300	0.2557	0.2814	R type	300	500	4

expr	min	mean	max	tipusdades	M	Y	ncores
LOOE.all(M, Y)	1.8640	1.9710	2.0780	R type	300	500	4
LOOE_BLAST(AD, YD, paral = TRUE)	1.0239	1.0562	1.0885	Delayed	300	500	4
LOOE_BLAST(AD, YD, paral = FALSE)	1.0106	1.1072	1.2039	Delayed	300	500	4
LOOE_BLAST(A, Y, paral = TRUE)	1.7566	1.8298	1.9030	R type	600	500	4
LOOE_BLAST(A, Y, paral = FALSE)	1.8380	1.8588	1.8797	R type	600	500	4
LOOE.all(M, Y)	1.8215	1.8273	1.8330	R type	600	500	4
LOOE_BLAST(AD, YD, paral = TRUE)	1.0091	1.0345	1.0598	Delayed	600	500	4
LOOE_BLAST(AD, YD, paral = FALSE)	0.9990	1.0042	1.0094	Delayed	600	500	4
LOOE_BLAST(A, Y, paral = TRUE)	5.5858	5.5932	5.6006	R type	900	500	4
LOOE_BLAST(A, Y, paral = FALSE)	6.0895	6.0975	6.1056	R type	900	500	4
LOOE.all(M, Y)	1.8069	1.8828	1.9587	R type	900	500	4
LOOE_BLAST(AD, YD, paral = TRUE)	1.0114	1.0125	1.0136	Delayed	900	500	4
LOOE_BLAST(AD, YD, paral = FALSE)	0.9942	0.9981	1.0020	Delayed	900	500	4
LOOE_BLAST(A, Y, paral = TRUE)	13.5638	13.6003	13.6367	R type	1200	500	4
LOOE_BLAST(A, Y, paral = FALSE)	14.5181	14.5218	14.5255	R type	1200	500	4
LOOE.all(M, Y)	1.7902	1.7990	1.8079	R type	1200	500	4
LOOE_BLAST(AD, YD, paral = TRUE)	1.0170	1.0277	1.0384	Delayed	1200	500	4
LOOE_BLAST(AD, YD, paral = FALSE)	1.0023	1.0024	1.0024	Delayed	1200	500	4
LOOE_BLAST(A, Y, paral = TRUE)	27.4351	27.4593	27.4835	R type	1500	500	4
LOOE_BLAST(A, Y, paral = FALSE)	29.7035	29.7558	29.8081	R type	1500	500	4
LOOE.all(M, Y)	1.7928	1.7946	1.7964	R type	1500	500	4
LOOE_BLAST(AD, YD, paral = TRUE)	1.0052	1.0370	1.0687	Delayed	1500	500	4
LOOE_BLAST(AD, YD, paral = FALSE)	0.9858	0.9923	0.9988	Delayed	1500	500	4



Observacions

Amb la funció LOOE s'observa una clara diferència de rendiment amb la utilització de dades DelayedArray, s'observa com els temps inicials amb mides de dades petites són molt grans respecte als temps obtinguts amb la mateixa funció utilitzant dades RObject però conforme augmenta el tamany de les dades, el rendiment es manté mentre que amb el tipus de dades RObject, el temps necessari per a realitzar els càlculs es dispara. Amb la resta de funcions on es mou una quantitat de dades gran, s'observa el mateix, el tipus de dades Delayed Array continua funcionant adeqüadament mentre el tipus de dades RObjects dispara el temps d'execució.

Codi R utilitzat

```
library(microbenchmark)
library(ggplot2)
library(DelayedArray)
library(BigDataStatMeth)


# --    PRODUCTE DE MATRIUS x BLOCS   --
# ------------------------------------

# Creem data.frame per emmagatzemar els resultats
result.df <- data.frame( expr = character(),min=numeric(),lq=numeric(),mean=numeric(),median=numeric(),u
                         max=numeric(),tipusop=character(),tipusdades=character(),bloc_size=numeric(), K
                         K=numeric(), N=numeric(), ncores = numeric(), nexec=numeric())
repet <- 1 # Repeticions microbenchmark
tipusop <- c('par','seq') # tipus execució paral·lel, sequencial
tipusdades <- c('R type','Delayed') # tipus execució paral·lel, sequencial
ncores <- detectCores() # Cores de la màquina

# Fem una execució per diferents tamanys de matrius, i per diferents tamanys de blocs
# per saber com actuen els algoritmes en les diferents situacions
matrixsize <- c(1000,2500)
#..# blocksize <- c(2, 4, 8, 16, 32, 64, 128, 256, 512)


for (k in 1:length(matrixsize))
{
  n <- matrixsize[k]
  A <- matrix(rnorm(n*n), nrow=n, ncol=n)
  B <- matrix(rnorm(n*n), nrow=n, ncol=n)
  AD <- DelayedArray(A)
  BD <- DelayedArray(B)

  for ( i in seq(2, 160, by=8)) ## Apliquem blocs quadrats 2x2,4x4,6x6.... fins
  {

     for (j in 1:length(tipusop))
     {
       if(j==1)
         bparal <- TRUE
       else
         bparal <- FALSE
       for(l in 1:length(tipusdades))
       {
         if(tipusdades[l]=="Delayed")
           res <- microbenchmark( blockmult(AD,BD,i, bparal), times = repet, unit = "s")
          else
            res <- microbenchmark( blockmult(A,B,i, bparal), times = repet, unit = "s")

         resdata <- as.data.frame(summary(res)[, c(1:7)])
         resdata <- cbind(resdata,tipusop[j],tipusdades[l],i,M=dim(A)[1], K=dim(A)[2], N=dim(B)[2], nco
         result.df <- rbind(result.df,resdata)
       }
```

```
      }

  }
}

# Guardem les dades per reutilitzar-les si convé.
   write.csv(result.df,"./doc/benchmark/multi5blocksize.csv")

# Readaptem el nom de les columnes del data.frame per tenir-los controlats -- OPCIÓ QUAN LLEGIM DEL FIT
   colnames(result.df) <-  c('id','expr', 'min', 'lq', 'mean', 'median', 'uq', 'max',
                             'tipusop','tipusdades', 'bloc_size','M', 'K', 'N', 'ncores', 'nexec')



colnames(result.df) <-  c('expr', 'min', 'lq', 'mean', 'median', 'uq', 'max',
                          'tipusop','tipusdades', 'bloc_size','M', 'K', 'N', 'ncores', 'nexec')

# Grafiquem els resultats
p <- ggplot(result.df, aes( x = result.df$bloc_size,
                            y = result.df$mean,
                            group= interaction(tipusop,tipusdades,M))) +
  geom_line(aes(color=interaction(tipusop,tipusdades,M))) +
  xlab('mida bloc') +
  ylab('temps(s)')

print(p)

result.par <- result.df[which(result.df[,11]>2),]
p <- ggplot(result.par, aes( x = result.par$bloc_size,
                            y = result.par$mean,
                            group= interaction(tipusop,tipusdades,M))) +
  geom_line(aes(color=interaction(tipusop,tipusdades,M))) +
  xlab('mida bloc') +
  ylab('temps(s)')

print(p)







result.df <- data.frame( expr = character(),min=numeric(),lq=numeric(),mean=numeric(),median=numeric(),u
                         max=numeric(),tipusop=character(),tipusdades=character(),bloc_size=numeric(), M
                         K=numeric(), N=numeric(), ncores = numeric(), nexec=numeric())

repet <- 2

for ( i in seq(1000, 5000, by=1000)) ## Apliquem blocs quadrats 2x2,4x4,6x6.... fins
{
  n <- i
  bloc_size <- 128
```

```
  A <- matrix(rnorm(n*n), nrow=n, ncol=n)
  B <- matrix(rnorm(n*n), nrow=n, ncol=n)
  AD <- DelayedArray(A)
  BD <- DelayedArray(B)

  for (j in 1:length(tipusop))
  {
    if(j==1)
      bparal <- TRUE
    else
      bparal <- FALSE
    for(l in 1:length(tipusdades))
    {
      if(tipusdades[l]=="Delayed")
        res <- microbenchmark( blockmult(AD,BD,bloc_size, bparal), times = repet, unit = "s")
      else
        res <- microbenchmark( blockmult(A,B,bloc_size, bparal), times = repet, unit = "s")

      resdata <- as.data.frame(summary(res)[, c(1:7)])
      resdata <- cbind(resdata, tipusop[j], tipusdades[l], i, M=dim(A)[1], K=dim(A)[2], N=dim(B)[2], nc
      result.df <- rbind(result.df,resdata)
    }

  }

}

write.csv(result.df,"./doc/benchmark/multBLOCKi5matrixsize.csv")
colnames(resultsize.df) <-  c('expr', 'min', 'lq', 'mean', 'median', 'uq', 'max',
                              'tipusop', 'tipusdades','bloc_size','M', 'K', 'N', 'ncores', 'nexec')

# Grafiquem els resultats
p <- ggplot(resultsize.df, aes( x = resultsize.df$M,
                                y = resultsize.df$mean,
                                group= interaction(tipusop,tipusdades))) +
  geom_line(aes(color=interaction(tipusop,tipusdades))) +
  xlab('mida matriu (nxn)') +
  ylab('temps (s)')

print(p)



# --    PRODUCTE DE MATRIUS - NO BLOCS   --
# ---------------------------------------


resultsize.df <- data.frame( expr = character(),min=numeric(),lq=numeric(),mean=numeric(),median=numeri
                         max=numeric(),tipusop=character(),tipusdades=character(),bloc_size=numeric(),M
                         K=numeric(), N=numeric(), ncores = numeric(), nexec=numeric())

repet <- 2
```

```r
for ( i in seq(1000, 2500, by=500)) ## Apliquem blocs quadrats 2x2,4x4,6x6.... fins
{
  n <- i
  A <- matrix(rnorm(n*n), nrow=n, ncol=n)
  B <- matrix(rnorm(n*n), nrow=n, ncol=n)
  DA <- DelayedArray(A)
  DB <- DelayedArray(B)

  for(l in 1:length(tipusdades))
  {
    if(tipusdades[l]=="Delayed")
      res <- microbenchmark(parXYProd(DA,DB),
                            blockmult(DA,DB,block_size = n, bparal), # Forcem un únic bloc per tota la
                            times = repet, unit = "s")
    else
      res <- microbenchmark(parXYProd(A,B),
                            blockmult(A,B,block_size = n, bparal), # Forcem un únic bloc per tota la ma
                            times = repet, unit = "s")

    resdata <- as.data.frame(summary(res)[, c(1:7)])
    resdata <- cbind(resdata, tipusop = 'par',tipusdades = tipusdades[l], 0, M=dim(A)[1], K=dim(A)[2], N
    resultsize.df <- rbind(resultsize.df,resdata)
  }

  res <- microbenchmark(A%*%B, times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = 'seq', tipusdades = 'R type', 0, M = dim(A)[1], K = dim(A)[2], N =
  resultsize.df <- rbind(resultsize.df,resdata)

}

write.csv(resultsize.df,"./doc/benchmark/multi5matrixsize.csv")

# Readaptem el nom de les columnes del data.frame per tenir-los controlats
colnames(resultsize.df) <-  c('expr', 'min', 'lq', 'mean', 'median', 'uq', 'max',
                              'tipusop','tipusdades', 'bloc_size','M', 'K', 'N', 'ncores', 'nexec')


# Grafiquem els resultats
p <- ggplot(resultsize.df, aes( x = resultsize.df$M,
                                y = resultsize.df$mean,
                                group= interaction(expr))) +
  geom_line(aes(color=interaction(expr))) +
  xlab('mida matriu(nxn)') +
  ylab('temps (s)')

print(p)



# --    PRODUCTE DE MATRIUS - BLOCS vs NO BLOCS (PARAL·LEL)   --
# ------------------------------------------------------------
```

```
resultbvsnb.df <- data.frame( expr = character(),min=numeric(),lq=numeric(),mean=numeric(),median=numeri
                              max=numeric(),tipusop=character(),tipusdades=character(),bloc_size=numeric
                              K=numeric(), N=numeric(), ncores = numeric(), nexec=numeric())

repet <- 2
bparal <- TRUE

for ( i in seq(1000, 3000, by=1000)) ## Apliquem blocs quadrats 2x2,4x4,6x6.... fins
{
  n <- i
  A <- matrix(rnorm(n*n), nrow=n, ncol=n)
  B <- matrix(rnorm(n*n), nrow=n, ncol=n)

  for(j in seq(1,2,by=1) )
  {
    if (j==1)
    {
      blocksize = 128
      res <- microbenchmark(blockmult(A,B,blocksize, bparal), times = repet, unit = "s")
    }
    else {
      blocksize = 0
      res <- microbenchmark(parXYProd(A,B),
                            blockmult(A,B,block_size = n, bparal),
                            times = repet, unit = "s")
    }

    resdata <- as.data.frame(summary(res)[, c(1:7)])
    resdata <- cbind(resdata, tipusop = 'par',tipusdades = 'R type', bloc_size = blocksize,
                     M=dim(A)[1], K=dim(A)[2], N=dim(B)[2], ncores, repet)
    resultbvsnb.df <- rbind(resultbvsnb.df,resdata)
  }
}


write.csv(resultbvsnb.df,"./doc/benchmark/multi5blocvsnobloc.csv")

# Grafiquem els resultats
p <- ggplot(resultbvsnb.df, aes( x = resultbvsnb.df$M,
                                 y = resultbvsnb.df$mean,
                                 group= interaction(expr))) +
  geom_line(aes(color=interaction(expr))) +
  xlab('mida matriu(nxn)') +
  ylab('temps (s)')

print(p)



# --    CROSSPROD i TCROSSPROD    --
# ----------------------------------
```

```r
# crossprod

results.df <- data.frame( expr = character(),min=numeric(),lq=numeric(),mean=numeric(),median=numeric()
                          max=numeric(),tipusop=character(),tipusdades=character(),bloc_size=numeric(),
                          K=numeric(), N=numeric(), ncores = numeric(), nexec=numeric())

repet <- 4

for ( i in seq(500, 3000, by=500)) ## Apliquem blocs quadrats 2x2,4x4,6x6.... fins
{
  n <- i
  A <- matrix(rnorm(n*n), nrow=n, ncol=n)
  AD <- DelayedArray(A)


  res <- microbenchmark(bdcrossprod(A, transposed = FALSE), # crossprod
                        crossprod(A),
                        t(A)%*%A,
                        times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'R type', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], N=0, ncores, repet)
  results.df <- rbind(results.df,resdata)

  res <- microbenchmark(bdcrossprod(AD, transposed = FALSE), # crossprod
                        times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'Delayed', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], N=0, ncores, repet)
  results.df <- rbind(results.df,resdata)

}

write.csv(results.df,"./doc/benchmark/crossprodi5.csv")

# Grafiquem els resultats
p <- ggplot(results.df, aes( x = results.df$M,
                                y = results.df$mean,
                                group= interaction(expr))) +
  geom_line(aes(color=interaction(expr))) +
  xlab('mida matriu(nxn)') +
  ylab('temps (s)')

print(p)



#   tcrossprod

results.df <- data.frame( expr = character(),min=numeric(),lq=numeric(),mean=numeric(),median=numeric()
                          max=numeric(),tipusop=character(),tipusdades=character(),bloc_size=numeric(),
```

```
                          K=numeric(), N=numeric(), ncores = numeric(), nexec=numeric())


for ( i in seq(500, 3000, by=500)) ## Apliquem blocs quadrats 2x2,4x4,6x6.... fins
{
  n <- i
  A <- matrix(rnorm(n*n), nrow=n, ncol=n)
  AD <- DelayedArray(A)


  res <- microbenchmark(bdcrossprod(A, transposed = TRUE), # crossprod
                        tcrossprod(A),
                        A%*%t(A),
                        times = repet, unit = "ms")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'R type', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], N=0, ncores, repet)
  results.df <- rbind(results.df,resdata)

  res <- microbenchmark(bdcrossprod(AD, transposed = TRUE), # crossprod
                        times = repet, unit = "ms")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'Delayed', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], N=0, ncores, repet)
  results.df <- rbind(results.df,resdata)

}

write.csv(results.df,"./doc/benchmark/tcrossprodi5.csv")

# Grafiquem els resultats
p <- ggplot(results.df, aes( x = results.df$M,
                             y = results.df$mean,
                             group= interaction(expr))) +
  geom_line(aes(color=interaction(expr))) +
  xlab('mida matriu(nxn)') +
  ylab('temps (s)')

print(p)




# --    CROSSPROD i TCROSSPROD AMB PESOS    --
# ------------------------------------------

# crossprod amb pesos
results.df <- data.frame( expr = character(),min=numeric(),lq=numeric(),mean=numeric(),median=numeric()
                          max=numeric(),tipusop=character(),tipusdades=character(),bloc_size=numeric(),
                          K=numeric(), w=numeric(), ncores = numeric(), nexec=numeric())
```

```
repet <- 10

for ( i in seq(500, 2500, by=500))
{
  n <- i
  A <- matrix(rnorm(n*n), nrow=n, ncol=n)
  u <- runif(n)
  w <- u * (1 - u)
  AD <- DelayedArray(A)
  wD <- DelayedArray(as.matrix(w))

  res <- microbenchmark(bdwproduct(A, w,"xtwx"),
                        crossprod(A, w*A),
                        t(A)%*%diag(w)%*%A,
                        times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'R type', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], w=length(w), ncores, repet)
  results.df <- rbind(results.df,resdata)

  res <- microbenchmark(bdwproduct(AD, w,"xtwx"),
                        bdwproduct(AD, wD,"xtwx"),
                        times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'Delayed', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], w=length(w), ncores, repet)
  results.df <- rbind(results.df,resdata)


}


write.csv(results.df,"./doc/benchmark/crossweightprodi5.csv")

# Grafiquem els resultats
p <- ggplot(results.df, aes( x = results.df$M,
                             y = results.df$mean,
                             group= interaction(expr))) +
  geom_line(aes(color=interaction(expr))) +
  xlab('mida matriu(nxn)') +
  ylab('temps (s)')

print(p)



# tcrossprod amb pesos
results.df <- data.frame( expr = character(),min=numeric(),lq=numeric(),mean=numeric(),median=numeric()
                          max=numeric(),tipusop=character(),tipusdades=character(),bloc_size=numeric(),
                          K=numeric(), N=numeric(), ncores = numeric(), nexec=numeric())
```

```
repet <- 5

for ( i in seq(500, 3000, by=500)) ## Apliquem blocs quadrats 2x2,4x4,6x6.... fins
{
  n <- i
  A <- matrix(rnorm(n*n), nrow=n, ncol=n)
  u <- runif(n)
  w <- u * (1 - u)
  AD <- DelayedArray(A)
  wD <- DelayedArray(as.matrix(w))

  res <- microbenchmark(bdwproduct(A, w,"xwxt"),
                        A%*%diag(w)%*%t(A),
                        times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'R type', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], N=length(w), ncores, repet)
  results.df <- rbind(results.df,resdata)

  res <- microbenchmark(bdwproduct(AD, w,"xwxt"),
                        bdwproduct(AD, wD,"xwxt"),
                        times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'Delayed', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], N=length(w), ncores, repet)
  results.df <- rbind(results.df,resdata)


}


write.csv(results.df,"./doc/benchmark/tcrossweightprodi5.csv")

# Grafiquem els resultats
p <- ggplot(results.df, aes( x = results.df$M,
                             y = results.df$mean,
                             group= interaction(expr))) +
  geom_line(aes(color=interaction(expr))) +
  xlab('mida matriu(nxn)') +
  ylab('temps (s)')

print(p)


# --    INVERSA CHOLESKY    --
# ------------------------------------------

# Genera matrius "definides positives"
Posdef <- function (n, ev = runif(n, 0, 10))
{
  Z <- matrix(ncol=n, rnorm(n^2))
```

```
  decomp <- qr(Z)
  Q <- qr.Q(decomp)
  R <- qr.R(decomp)
  d <- diag(R)
  ph <- d / abs(d)
  O <- Q %*% diag(ph)
  Z <- t(O) %*% diag(ev) %*% O
  return(Z)
}



results.df <- data.frame( expr = character(),min=numeric(),lq=numeric(),mean=numeric(),median=numeric()
                          max=numeric(),tipusop=character(),tipusdades=character(),bloc_size=numeric(),
                          K=numeric(), N=numeric(), ncores = numeric(), nexec=numeric())

repet <- 4

for ( i in seq(500, 2000, by=500))
{

  A <- Posdef(n=i, ev=1:i)
  AD <- DelayedArray(A)

  res <- microbenchmark(inversechol_par(A),
                        bdInvCholesky_LDL_eigen(A),
                        solve(A),
                        times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'R type', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], N=0, ncores, repet)
  results.df <- rbind(results.df,resdata)

  res <- microbenchmark(inversechol_par(AD),
                        bdInvCholesky_LDL_eigen(AD),
                        times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'Delayed', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], N=0, ncores, repet)
  results.df <- rbind(results.df,resdata)

}


write.csv(results.df,"./doc/benchmark/invcholi5.csv")

# Grafiquem els resultats
p <- ggplot(results.df, aes( x = results.df$M,
                             y = results.df$mean,
                             group= interaction(expr))) +
  geom_line(aes(color=interaction(expr))) +
```

```
  xlab('mida matriu(nxn)') +
  ylab('temps (s)')

print(p)



# --    DESCOMPOSICIÓN SVD    --
# ------------------------------



results.df <- data.frame( expr = character(),min=numeric(),lq=numeric(),mean=numeric(),median=numeric()
                          max=numeric(),tipusop=character(),tipusdades=character(),bloc_size=numeric(),
                          K=numeric(), N=numeric(), ncores = numeric(), nexec=numeric())

repet <- 2

for ( i in seq(300, 1500, by=300))
{
  n <- i
  A <- matrix(rnorm(n*n), nrow=n, ncol=n)
  AD <- DelayedArray(A)

  res <- microbenchmark( BDsvd2( A, n-1, n, FALSE), # No normalitza la matriu
                         svd(tcrossprod(A)),
                         times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'R type', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], N=0, ncores, repet)
  results.df <- rbind(results.df,resdata)

  res <- microbenchmark(BDsvd2(AD, n-1,n,FALSE),
                         times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'Delayed', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], N=0, ncores, repet)
  results.df <- rbind(results.df,resdata)

}


write.csv(results.df,"./doc/benchmark/svdi5.csv")

# Grafiquem els resultats
p <- ggplot(results.df, aes( x = results.df$M,
                             y = results.df$mean,
                             group= interaction(expr))) +
  geom_line(aes(color=interaction(expr))) +
  xlab('mida matriu(nxn)') +
  ylab('temps (s)')
```

```r
print(p)



# --    LOOE    --
# ------------------------------



# FUNCIONS R ORIGINALS : (Afegida una funció que crida les 2 funcions necessaries per obtenir els coeff
#                         poder obtenir el temps total.)

inversecpp_orig <- function(X, lambda=1, eigen=TRUE,
                            Lambda, Q){
  if (eigen){
    ee <- eigen(X, symmetric = TRUE)  # mirar la librería BiocSingular
    Lambda <- ee$values
    Lambda[Lambda<0] <- 0
    Q <- ee$vectors
  }
  else
    if(missing(Lambda) | missing(Q))
      stop('SVD results should be provided. \n')

  if (lambda == 1)
    W <-  1/Lambda
  else
    W <- 1/(Lambda + lambda)

  Ginv <- rfunctions::crossprodcpp(t(Q), W)  # implementar xwxt  (en la librería rfunctions está xtwx)
  # Q%*%diag(W)%*%t(Q)
  Ginv
}

solveEigen_orig <- function(X, Y, lambda){   # X DelayedArray (HDF5), Y un vector
  XX <- tcrossprod(X)
  Ginv <- inversecpp_orig(XX, lambda=lambda)  # Con DelayedArray (HDF5)
  coef <- t(Ginv%*%X)%*%Y
  coef
}


LOOE.i_orig <- function(lambda, Lambda, Q, Y){
  Ginv <- inversecpp_orig(lambda=lambda, Lambda=Lambda,
                          Q=Q, eigen=FALSE)
  print(dim(Ginv));
  print(dim(Y));
  cte <- Ginv%*%Y
  ans <- sum((cte/diag(Ginv))^2)
  ans
}
```

```
#
# Compute LOOE
#

LOOE_orig <- function(X, Y, nlambdas=100, max.lambda=1, lambdas){

  if (missing(lambdas)){
    lambdas <- seq(0.01, max.lambda, length=nlambdas)
  }
  ee <- eigen(X, symmetric = TRUE)
  Lambda <- ee$values

  Lambda[Lambda<0] <- 0
  Q <- ee$vectors
  looe <- sapply(lambdas, LOOE.i_orig, Lambda=Lambda, Q=Q, Y=Y)

  lambda.min <- lambdas[which.min(looe)]

  Ginv <- inversecpp_orig(X, lambda.min)

  ans <- list(looe=looe, Ginv=Ginv, lambdas=lambdas,
              lambda.min=lambda.min)
  ans
}

LOOE.all <- function(X, Y){
  sol <- LOOE_orig(tcrossprod(X),Y)
  return(solveEigen_orig(X, as.matrix(Y), lambda=sol$lambda.min))
}

### FI FUNCIONS ORIGINALS R ###



results.df <- data.frame( expr = character(),min=numeric(),lq=numeric(),mean=numeric(),median=numeric()
                          max=numeric(),tipusop=character(),tipusdades=character(),bloc_size=numeric(),
                          K=numeric(), N=numeric(), ncores = numeric(), nexec=numeric())

repet <- 2

for ( i in seq(300, 1500, by=300))
{
  if(i>=900) p <- 500
  else p <- 100
  n <- i
  A <- matrix(rnorm(n*p), nrow=n, ncol=p)

  Y <- 2.4*M[,1] + 1.6*M[,2] - 0.4*M[,5]
  AD <- DelayedArray(M)
  YD <- DelayedArray(as.matrix(Y))
```

```
  res <- microbenchmark( LOOE_BLAST(A,Y,paral=TRUE),
                         LOOE_BLAST(A,Y,paral=FALSE),
                         LOOE.all(M,Y),
                         times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'R type', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], N=length(Y), ncores, repet)
  results.df <- rbind(results.df,resdata)

  res <- microbenchmark( LOOE_BLAST(AD,YD,paral=TRUE),
                         LOOE_BLAST(AD,YD,paral=FALSE),
                         times = repet, unit = "s")

  resdata <- as.data.frame(summary(res)[, c(1:7)])
  resdata <- cbind(resdata, tipusop = '',tipusdades = 'Delayed', bloc_size = 0,
                   M=dim(A)[1], K=dim(A)[2], N=length(Y), ncores, repet)
  results.df <- rbind(results.df,resdata)

}


write.csv(results.df,"./doc/benchmark/looei5.csv")

# Grafiquem els resultats
p <- ggplot(results.df, aes( x = results.df$M,
                             y = results.df$mean,
                             group= interaction(expr))) +
  geom_line(aes(color=interaction(expr))) +
  xlab('mida matriu(nxn)') +
  ylab('temps (s)')

print(p)
```

Informació del sistema

```
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.5
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
## [1] ca_ES.UTF-8/ca_ES.UTF-8/ca_ES.UTF-8/C/ca_ES.UTF-8/ca_ES.UTF-8
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] knitr_1.21           BigDataStatMeth_1.0  ggplot2_3.1.0
## [4] microbenchmark_1.4-6
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.0          highr_0.7           pillar_1.3.1
##  [4] compiler_3.5.1      plyr_1.8.4          tools_3.5.1
##  [7] digest_0.6.18       evaluate_0.13       tibble_2.0.1
## [10] gtable_0.2.0        lattice_0.20-38     pkgconfig_2.0.2
## [13] rlang_0.3.1         Matrix_1.2-15       yaml_2.2.0
## [16] xfun_0.4            withr_2.1.2         stringr_1.4.0
## [19] dplyr_0.8.0.1       grid_3.5.1          tidyselect_0.2.5
## [22] glue_1.3.0          R6_2.4.0            rmarkdown_1.11
## [25] purrr_0.3.0         magrittr_1.5        scales_1.0.0
## [28] htmltools_0.3.6     assertthat_0.2.0    colorspace_1.4-0
## [31] labeling_0.3        stringi_1.3.1       lazyeval_0.2.1
## [34] RcppParallel_4.4.2  munsell_0.5.0       crayon_1.3.4
## [37] RcppEigen_0.3.3.5.0
```