Radu Emanuel Ioan 336 CC

Tema 2 Arhitectura Sistemelor de Calcul

Tema a avut ca scop realizarea de optimizări ce pot fi aduse unei expresii ce conține operații cu matrici. Am implementat fiecare cerință din team după cum urmează:

1. **Blas** – am folosit funcție blas din biblioteca 'BLAS Level 3 Routines', 'cblas_dtrmm' care realizează înmulțirea a două matrici luând în considerare că minim una dintre ele este triunghiulară. Funcția are de asemenea facilitatea de a face înmulțiri de matrici, transpunând o una din ele data ca parametru. Adunarea se realizează de mână.

2. **Varianta neoptimizată** – operațiile se fac în forma clasică, transpusa lui A, ridicarea la pătrat și mai apoi calculul de sume parțiale în C folosind 3 foruri după i, j și k. Pentru calcului lui A^2 iau în considerare că matricea A este superior triunghiulară și j-ul în for pornește mereu de la valoarea lui i.

3. **Varianta optimizată** – prima optimizare a fost folosirea registrilor pentru înmulțire. Memoria din matricea C se accesează doar o data la n operații folosindu-se variabila suma. De asemenea, folosesc pointeri pentru accesarea valorilor din matrici. Nu se mai folosesc accese vectoriale prin dereferențiere. Îmbunătățirea timpului față de varianta neoptimizată pentru testul 3 (N = 1200) este de aproximativ 46%.

4. **Varianta optimizată cu flags** – s-au folosit următoarele flaguri:

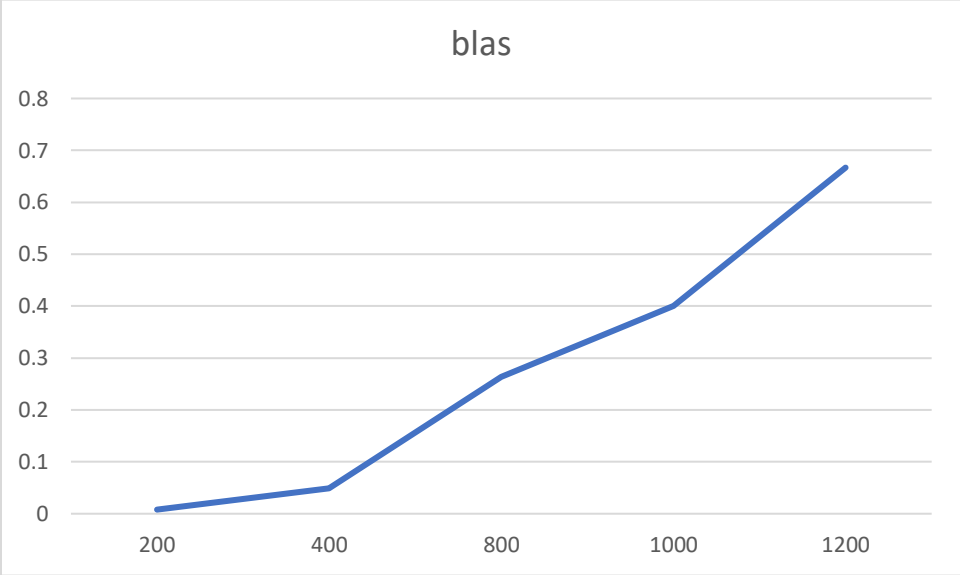
    - fno-trapping-math – codul se compilează având garanția că nu există cazuri de împărțire la 0/overflow/underflow/rezultate inexacte și operații invalide
    - fno-rounding-math – nu se rotunjesc valorile. Se folosesc valori double și acest flag trebuie folosit.
    - floop-interchange – permite interschimbarea loopurilor – în cazul nostru al forurilor pentru optimizarea codului.
    - fno-math-errno- nu seteaza errno dupa ce apeleaza o functie matematica care executa o singură instrucțiune
    - ffinite-math-only – permite oprimizarea pentru float-uri presupund ca valorile nu sunt NULL sau infinit
    -

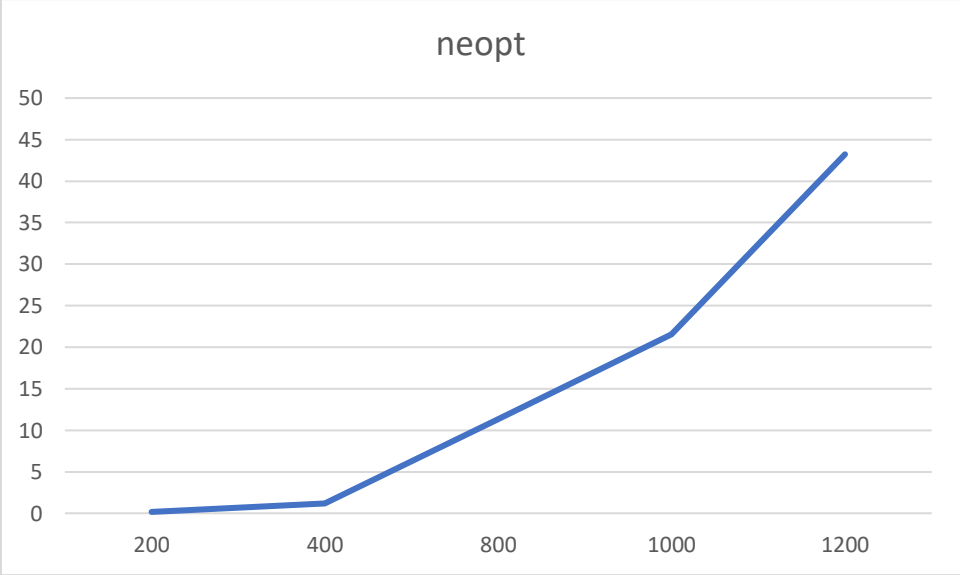
    Peformanța diferă de la o rulare la alta. Cea mai mare diferență față de varianta opt_f a fost de 4.8775%. Sunt cazuri în care timpul rezultat este mai mare față de opt_f.

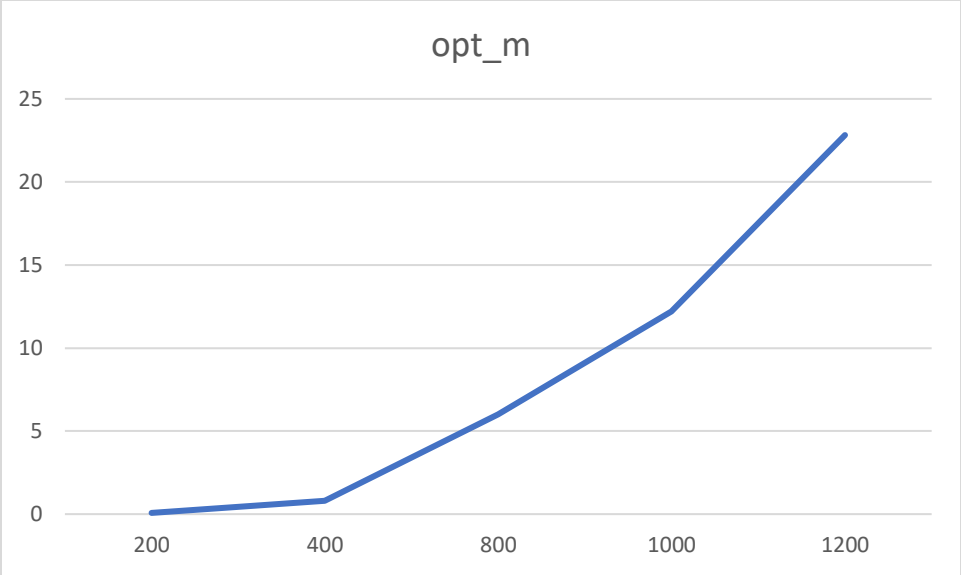
    Pentru testare se folosește fișierul test.sh din arhivă. Se înlocuiește în el numele variantei care se vrea fi testată. Pentru rulare folosesc comanda:

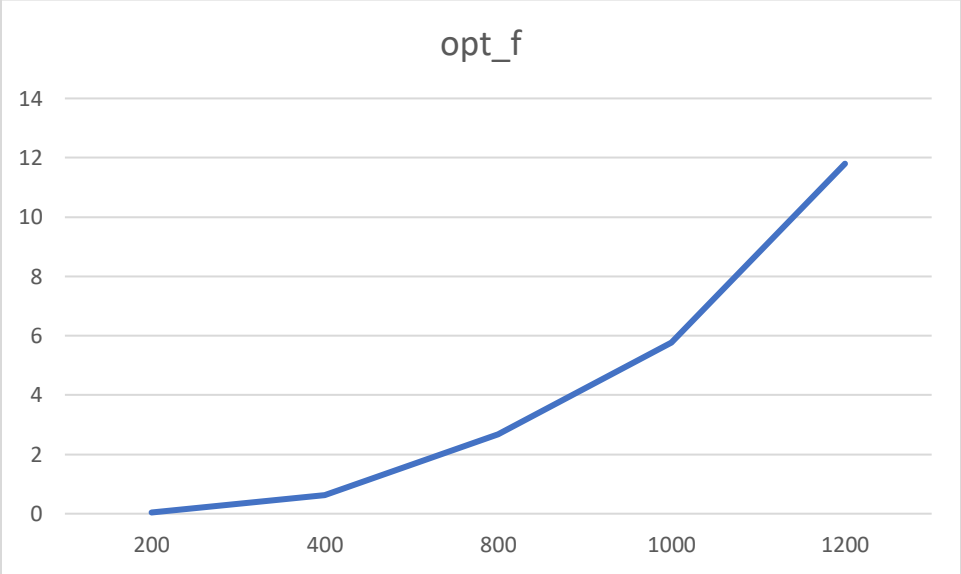
    qsub -cwd -q ibm-nehalem.q -b y ./test.sh

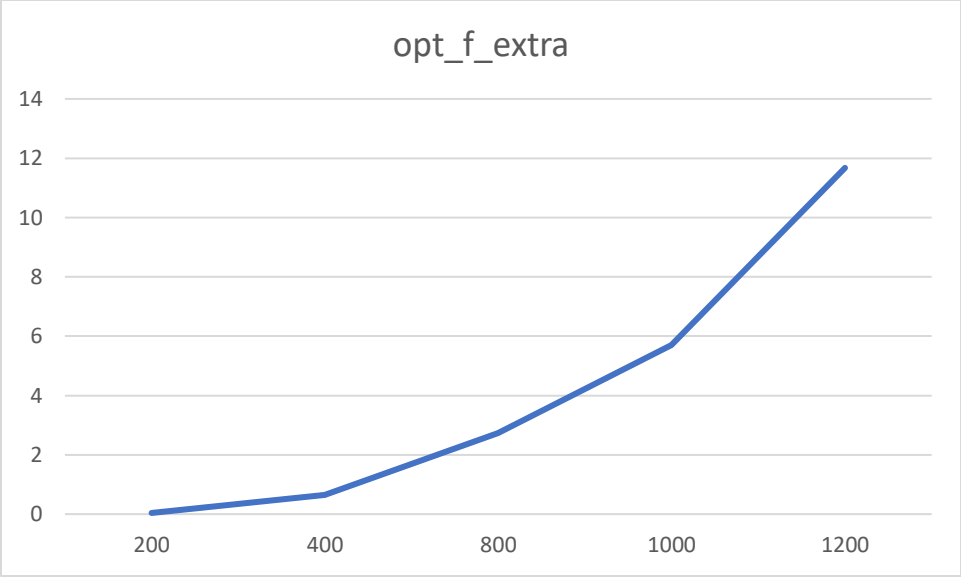
Grafice

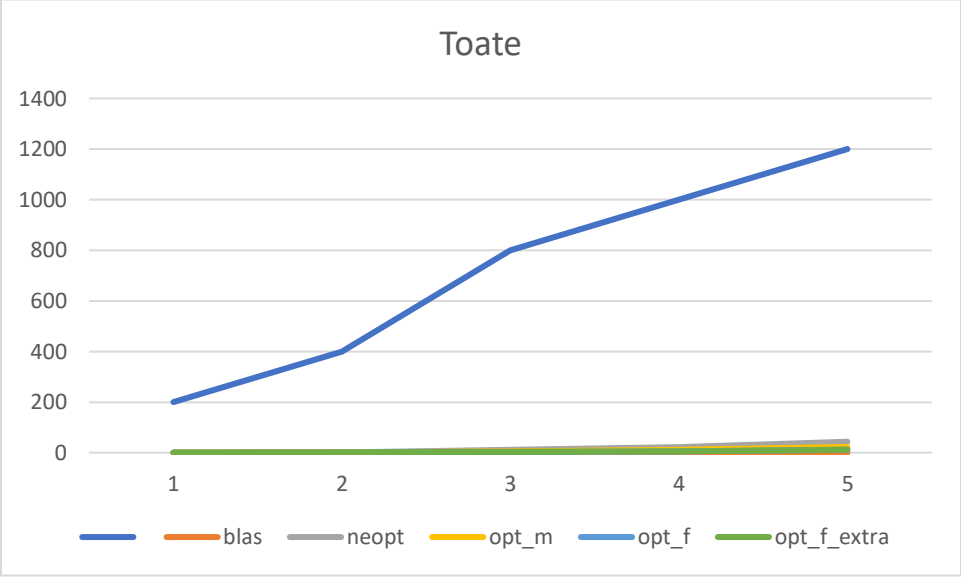

blas
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
200
400
800
1000
1200


neopt
50
45
40
35
30
25
20
15
10
5
0
200
400
800
1000
1200


opt_m
25
20
15
10
5
0
200
400
800
1000
1200


opt_f
14
12
10
8
6
4
2
0
200
400
800
1000
1200





Se observă că cele mai bune performanțe se obțin pentru blas, urmat de opt_f_extra care e foarte aproape de opt_f, iar mai apoi opt_m și neopt.

Diferența de performanță dintre neopt și opt este de aproximativ 46%, în timp ce cea pentru opt_f și opt_f_extra nu diferă cât ar trebui să difere pentru cazurile de mai sus.

Față de celelalte optimizări, -O3, prezent atât în opt_f cât și în opt_f_extra activează următoarele flaguri: -fgcse-after-reload**,** -finline-functions**,** -fipa-cp-clone**,** -fpredictive-commoning**,** -ftree-vectorize**,** -funswitch-loops care sporesc viteza de execuție, dar cresc și dimensiunea obiectului rezultat.

Tabelul cu timpii programelor

Variantă	1	2	3	4	5
	200	400	800	1000	1200
blas	0.00792	0.04903	0.26404	0.40018	0.66682
neopt	0.16326	1.16342	11.3057	21.5637	43.2232
opt_m	0.07395	0.80373	6.00992	12.2076	22.8206
opt_f	0.04015	0.62767	2.67132	5.75951	11.8003
opt_f_extra	0.0398	0.64585	2.72977	5.70395	11.6742