# ASC – Tema 2

Trașcă Andrei-Cătalin – 335CA

1. Varianta **BLAS**

Pentru implementarea folosind BLAS am folosit funcția cblas\_dtrmm. Aceasta funcție înmulțește 2 matrici, A și B, A putând fi triunghiulară. Rezultatul final se pune în B. Deoarece matricile inițiale sunt suprascrise, am folosit 2 matrici auxiliare astfel:

* A2 – copiez inițial în el matricea A, iar apoi înmulțesc A cu A2, la final A2 conținând matricea A2;
* Următoare înmulțire este A2 x B, rezultatul final fiind pus în B;
* La început am copiat în matricea B2 matricea B, pentru a o putea înmulți la final cu At;
* La final mai trebuie sa adun cele 2 matrici rezultate, B și B2, rezultatul final fiind pus în B2.

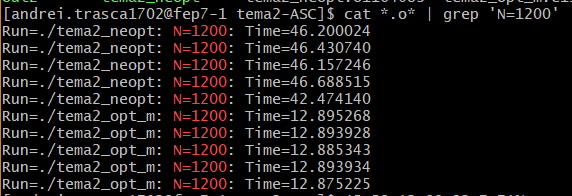
1. Varianta **neopt**

Pentru implementarea variantei neoptime am creat o funcție – multiplyMatrix, care primește ca parametrii 3 matrici de dimensiune N x N, și un întreg care specifică tipul primei matrice, A – inferior, superior sau non triunghiulară, și face operația C = A x B.

1. Varianta **opt\_m**

Plecând de la funcția variantei neopt și de la explicațiile din laboratorul 5 ale algoritmului BMM, am făcut funcția multiplyMatrix2. De asemenea, am înlocuit și accesul la memorie folosind vectori cu pointeri.

Am rulat 5 teste pe cele 2 variante, neopt și opt\_m, iar rezultatele au fost următoarele:



Se poate observa un timp mediu pentru neopt de 45.58 secundem iar pentru opt\_m de 12.88 secunde. Astfel am obținut un speedup mediu de 32.7 secunde, adică peste 70%.

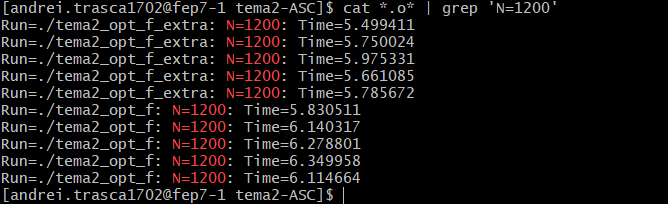
1. Varianta opt\_f\_extra

Flagurile folosite au fost următoarele:

-funroll-loops -ffast-math -mfpmath=sse -funsafe-math-optimizations

* -funroll-loops: acest flag desface loop-urile al căror număr de pași poate fi determinat la compilare. Singurul dezavantaj al său îl reprezintă faptul că sursa va fi la final mai mare, chiar daca îmbunătățește sau nu rezultatul. În cazul de față eu nu am loop-uri al căror număr de pași poate fi determinat înainte de citire, dar neștiind cum este implementat main-ul l-am inclus oricum, neavând nimic de pierdut.
* -mfpmath=sse: procesorul va folosi setul de instrucțiuni de tip sse – single instruction, multiple data, schimbând resursele utilizate de acesta. Este un set de instrucțiuni ce este folosit în aplicații ce fac intens operații cu matrice(cum ar fi GPU și cum este și cazul de față).
* -funsafe-math-optimizations: acest flag determină programul să presupună că întotdeauna argumentele și rezultatele sunt valide. Deși nu este recomandat pentru că poate duce la rezultate greșite, în cazul de față se poate garanta întotdeauna corectitudinea datelor, iar în urma testelor am observat că rezultatele sunt în continuare corecte.
* --ffast-math: asemănător cu funsafe-math-optimizations, acesta activează mai multe flaguri ce pot duce la un speedup al programului, dar nu mai garantează rezultate valide la final.

Am rulat 5 teste pe cele 2 variante, opt\_f și opt\_f\_extra, iar rezultatele au fost următoarele:

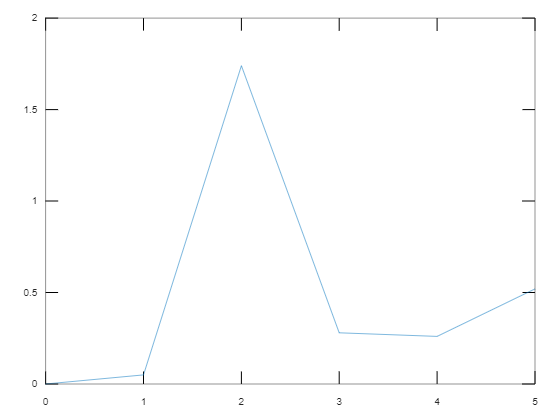
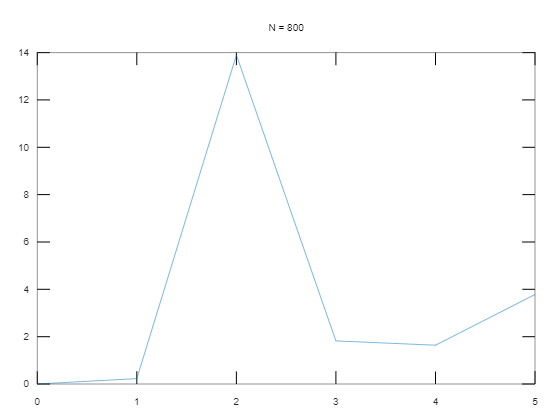
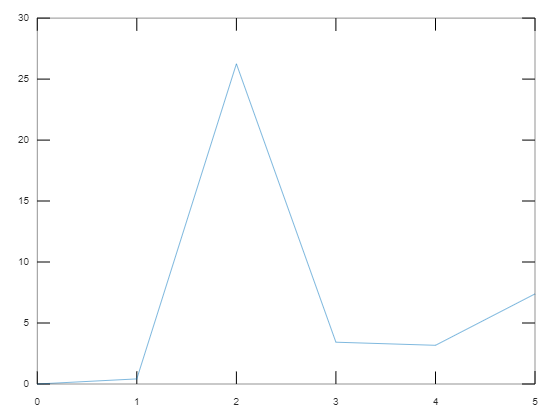
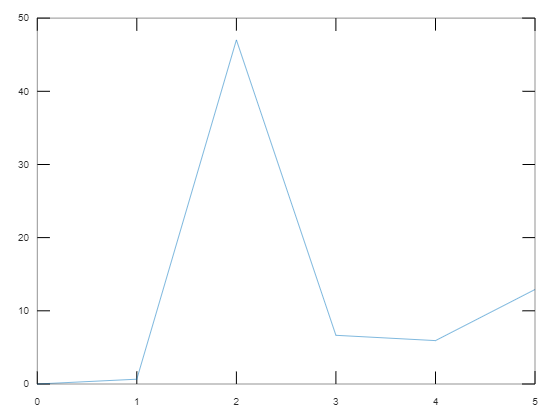
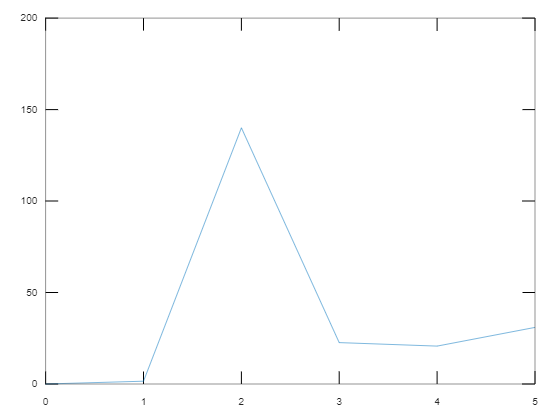


Se poate observa un timp mediu pentru opt\_f de 6.14 secunde iar pentru opt\_f\_extra de 5.73 secunde. Astfel am obținut un speedup mediu de 0.41 secunde, adică peste 6%.

1. Analiza performanței

Cele 5 grafice care urmează a fi prezentate au următoarea legendă:

* Pe axa Ox  se află cei 5 algoritmi, numerotați de la 1 la 5 astfel:
  + 1. BLAS
  + 2. Neopt
  + 3. Opt\_f
  + 4. Opt\_f\_extra
  + 5. Opt\_m
* Pe axa Oy  se află timpii obținuți de fiecare algoritm
* Fiecare grafic reprezintă rezultatul pentru un anume N, N = [400, 800, 1000, 1200, 1600]

1. N = 400
2. N = 800
3. N = 1000
4. N = 1200
5. N = 1600

Se poate observa din cele 5 grafice că varianta BLAS este fără discuție cea mai optimă, având rezultate net superioare oricărei alte variante. Varianta neoptimă are cel mai rău timp, ajungând la un rezultat de 160 de ori mai mare decât varianta BLAS. Între variantele opt\_f și opt\_f\_extra se poate observa în fiecare caz un speed-up micuț. Varianta optimizată are un timp mult mai bun fața de variantă neoptimizată(de 10 ori mai mic), însă în niciun punct nu poate să se apropie măcar putin de variantele compilate cu O3, dovedind astfel faptul că în acest moment compilatoarele au ajuns la un nivel foarte înalt de optimizare. Deși nici ele nu sunt apropiate ca timp de varianta BLAS, reușesc o variantă de 2 ori mai bună pe un cod neoptimizat, față de un cod optimizat ce este compilat cu flagul O3.

Datele pe care au fost făcute graficele sunt următoarele:

