**TEMA 2 - CODE OPTIMIZATION**

A. Descrierea implementarii:

---------------------------------

1. Varianta neoptimizata

Calculez transpusele matricilor A si B, apoi calculez A tranpus X B si B transpus X A, pun adunarea celor 2 inmultiri intr-un array auxiliar doar deasupra sau pe diagonala principala (unde i <= j) si returnez matricea obtinuta prin ridicarea la patrat a acelei matrici auxiliare. Complexitatea aceste variante este O(n ^ 3).

2. Varianta optimizata prin cod

Optimizez fiecare inmultire de matrici folosind blocked matrix multiplication astfel: de fiecare data calculez cate bs termeni, unde am luat bs egal cu 40 pentru ca N este sigur multiplu de 40; aceasta varianta are aceeasi complexitate cu varianta neoptimizata (O(n ^ 3 / bs ^ 3) (for-urile exterioare) \* O(bs ^ 3) (for-urile interioare) = O(n ^ 3)). Diferenta este faptul ca aici sunt mai putine cache miss-uri pentru ca datorita lui bs calculam blocuri care intra exact in cache.

3. Varianta optimizata prin flag-uri

Folosesc flag-urile -O3 si -Ofast la gcc si la icc. -Ofast face toate optimizarile pe care le face si -O3 si seteaza pe 1 flag-ul --fast-math pentru a imbunatati viteza in cazul operatiilor matematice.

4. Varianta blas

Folosesc functia blas dgemm care realizeaza inmultirea a 2 matrici. Pentru a fi mai eficient, la primele 2 inmultiri dam al doilea argument CblasTrans pentru a nu face inainte de apelurile lui dgemm transpunerile lui A, respectiv B. Dimensiunea fiecarei matrici folosite este N si in toate cazurile alfa este 1 si beta este 0 (intrucat formula generica pentru dgemm este C = alfa \* A \* B + beta \* C si de fiecare data noi avem nevoie sa calculam o simpla inmultire de matrici pe al carei rezultat il retinem in alta variabila).

B. Analiza performantei:

--------------------------

Variantele sunt puse in ordine crescatoare a timpilor obtinuti.

Compilare cu gcc:



1. Varianta blas (colorata cu portocaliu)

N: 400, 1000, 1600

Timpi: 0.05, 0.77, 3.02

2. Varianta optimizata prin flag-uri (galben)

Timpi: 0.52, 17.34, 71.46

3. Varianta optimizata prin cod (verde)

Timpi: 1.47, 24.30, 103.82

4. Varianta neoptimizata (albastru)

Timpi: 1.51, 32.18, 153.29

Compilare cu icc:



1. Varianta blas (colorata cu portocaliu)

Timpi: 0.05, 0.78, 3.10

2. Varianta optimizata prin flag-uri (galben)

Timpi: 0.17, 9.97, 40.86

3. Varianta optimizata prin cod (verde)

Timpi: 1.72, 26.92, 113.57

4. Varianta neoptimizata (albastru)

Timpi: 1.50, 31.24, 159.52

C. Analiza comparativa gcc versus icc:

------------------------------------------------

Grafic pentru varianta BLAS



Cu portocaliu sunt timpii obtinuti cu gcc, iar cu galben sunt timpii obtinuti cu icc.

Pentru varianta BLAS timpii obtinuti cu gcc si icc sunt aproximativ aceiasi.

Grafic pentru varianta optimizata prin flag-uri



Cu portocaliu sunt timpii obtinuti cu gcc, iar cu galben sunt timpii obtinuti cu icc.

Pentru varianta optimizata prin flag-uri (-Ofast) timpii obtinuti cu icc sunt mult mai mici decat cei obtinuti cu gcc.

Grafic pentru varianta optimizata prin cod



Cu portocaliu sunt timpii obtinuti cu gcc, iar cu galben sunt timpii obtinuti cu icc.

Pentru varianta optimizata prin cod timpii obtinuti cu gcc sunt putin mai mici decat cei obtinuti cu icc.

Variatia timpilor pentru icc si gcc in cazul optimizarii prin flag-uri este mai mare decat variatia timpilor pentru gcc si icc in cazul optimizarii prin cod.

Grafic pentru varianta neoptimizata



Cu portocaliu sunt timpii obtinuti cu gcc, iar cu galben sunt timpii obtinuti cu icc.

La varianta neoptimizata timpii obtinuti cu gcc si icc sunt aproximativ aceiasi.

Daca dorim sa facem o medie intre cele 4 variante pentru fiecare compilator observam ca icc este mai eficient decat gcc intrucat folosind optimizarea cu flag-uri timpii pentru icc sunt mult mai mici decat timpii pentru gcc.

Constantin Ioan, 335 CA