





## Tema 2

- **Deadline soft:** 04 mai 2022 ~~02-mai~~, ora 23:55. Primiți un bonus de 10% din punctajul obținut pentru trimiterea temei înainte de 29 aprilie 2022, ora 23:55.
- **Deadline hard:** 09 mai 2022, ora 23:55. Veți primi o depunere de 10% din punctajul maxim al temei pentru fiecare zi de întârziere (dupa ~~02-mai~~ 04 mai), până la maxim ~~7-zile~~ 5 zile, adică până pe 09 mai 2022, ora 23:55.
- **Responsabili:**  Vlad Spoiala,  Cosmin-Gabriel Samoila,  Emil Slusanschi,  Radu Petru Daia



- Dată publicare: 16 aprilie 2022
- Dată actualizare enunț: 02 mai 2022

## Enunț


Se dă următoarea operație cu matrice:

$$C = B \times A \times A^t + B^t \times B$$

unde:

- $A$  și  $B$  sunt matrice patratiche de double de dimensiune  $N \times N$
- $A$  este o matrice superior triunghiulara
- $A^t$  este transpusa lui  $A$  și  $B^t$  este transpusa lui  $B$
- $\times$  este operația de înmulțire
- $+$  este operația de adunare

Se dorește implementarea operației de mai sus în C/C++ în 3 moduri:

- **blas** - o variantă care folosește una sau mai multe funcții din  BLAS Atlas pentru realizarea operațiilor de înmulțire și adunare.
- **neopt** - o variantă "de mână" fără îmbunătățiri.
- **opt\_m** - o variantă îmbunătățită a versiunii **neopt**. Îmbunătățirea are în vedere exclusiv modificarea codului pentru a obține performanțe mai bune.



Fiecare din cele 3 implementari de mai sus va tine cont de faptul ca **A este o matrice superior triunghiulara**.

## Rulare și testare

Pentru testarea temei vă este oferit un schelet de cod pe care trebuie să-l completați cu implementările celor 3 variante menționate mai sus. Scheletul de cod este structurat astfel:

- **main.c** - conține funcția main, precum și alte funcții folosite pentru citirea fișierului cu descrierea testelor, scrierea matricei rezultat într-un fișier, generarea datelor de intrare și rularea unui test. Acest fișier va fi suprascris în timpul corectării și nu trebuie modificat.
- **utils.h** - fișier header. Acest fișier va fi suprascris în timpul corectării și nu trebuie modificat.
- **solver\_blas.c** - în acest fișier trebuie să adaugați implementarea variantei **blas**.
- **solver\_neopt.c** - în acest fișier trebuie să adaugați implementarea variantei **neopt**.
- **solver\_opt.c** - în acest fișier trebuie să adaugați implementarea variantei **opt\_m**.
- **Makefile** - Makefile folosit la compilarea cu gcc.
- **compare.c** - utilitar ce poate fi folosit pentru a compara doua fisiere rezultat. Acest fișier va fi suprascris în timpul corectării și nu trebuie modificat.



Puteți aduce orice modificare scheletului de cod exceptând cele 3 fișiere menționate mai sus.

În urma rulării comenzii **make** vor rezulta 3 fișiere binare, **tema2\_blas**, **tema2\_neopt** și **tema2\_opt\_m** corespunzătoare celor 3 variante care trebuie implementate.

Rularea se va realiza astfel:



```
./tema2_<mod> input
```

unde:

- mod este unul din modulele **blas**, **neopt**, **opt\_m**
- input este fișierul ce conține descrierea testelor.

Fișierul **input** este structurat astfel:

- pe prima linie numărul de teste.
- pe următoarele linii descrierea fiecărui test:
  - valoarea lui  $N$ .
  - seed-ul folosit la generarea datelor.
  - calea către fișierul de ieșire ce conține matricea rezultat.

Rularea se va face pe partitia **nehalem**. Compilarea se va face folosind **gcc-8.5.0**. Pentru variantele **blas**, **neopt** și **opt\_m** nu vor fi utilizate flag-uri de optimizare pentru compilare (va fi utilizat -O0). Pentru linkarea cu BLAS Atlas se va folosi versiunea single-threaded **libsatlas.so.3.10** disponibilă în directorul

```
/usr/lib64/atlas
```

de pe masinile din partitia nehallem



Nodurile din partitia nehallem sunt mai lente decat nodurile din alte partitii. Testele voastre de performanta trebuie realizate pe partitia nehallem deoarece evaluarea temelor se va face pe aceasta partitie.

### Navigare

- Regulament
- Echipă
- Orar
- Catalog

### Laboratoare

- Laboratorul 01 - Introducere în limbajul Python
- Laboratorul 02 - Fire de execuție în Python
- Laboratorul 03 - Programare concurentă în Python (continuare)
- Laboratorul 04 - Arhitecturi de Microprocesoare și Sisteme de Calcul
- Laboratorul 05 - Tehnici de Optimizare de Cod - Înmulțirea Matricelor
- Laboratorul 06 - Analiza Performanțelor Programelor
- Laboratorul 07 - Arhitecturi de tip GPGPU
- Laboratorul 08 - Arhitectura GPU NVIDIA CUDA
- Laboratorul 09 - Advanced CUDA
- Exerciții din alți ani

### Teme

- Tema 1
- Tema 2
- Tema 3

### Resurse

- Folosire X11 în WSL2
- Ghid folosire cluster

### GPU related

- CUDA C Programming
- CUDA NVCC compiler
- Visual Profiler
- CUDA 9.1 Toolkit
- NVIDIA Tesla K40M
- NVIDIA Tesla C2070
- Nvidia Tesla 2050/2070
- Nvidia CUDA Fermi/Tesla

### Lecture related

- Computatoare
- Taxonomia Flynn
- Single Board Computers
- Explicitly Parallel Instruction Computing
- Intel Parallel Studio

### Utilitare

- Dinero cache simulator
- Python Visual Interpreter

### Older Labs & Resources

- Cell - Rulare pe CLUSTER
- GDB on Cell BE
- Mailbox Hands-On
- Arhitectura Cell BE
- Kickstart Cell BE
- DMA 101
- Reference Manuals
- Folosirea simulatorului
- Branch Prediction
- Cell Profiler
- Tutorial Cell - Eclipse
- Software Managed Cache
- Liste DMA
- Continut

### Table of Contents

- Tema 2
  - Enunț
  - Rulare și testare
  - Punctaj
  - Precizări încărcare / VMChecker
  - Precizări și recomandări
  - Resurse


Fisierele input ce vor fi folosite pentru testare si fisierele output referință le găsiți pe fep în acest folder:

```
/export/asc/tema2/
```

Fisierul **input** contine 3 teste:

```
3
400 123 out1
800 456 out2
1200 789 out3
```

În cazul fisierului **input** avem 3 teste pentru următoarele valori ale lui N: 400, 800, respectiv 1200. Seed-urile folosite la generarea datelor de intrare sunt 123, 456, respectiv 789. Fisierele de output sunt out1, out2, respectiv out3.



Pentru a fi luată în considerare la punctaj, implementarea trebuie să producă rezultate corecte pe toate cele 3 teste din fisierul **input**.

Fisierul **input\_valgrind** ce va fi folosit pentru rularile de valgrind contine un singur test:

```
1
400 123 out1
```

## Punctaj


Punctajul este împărțit astfel:

- **15p** pentru implementarea variantei **blas** dintre care:
  - 12p dacă implementarea obține rezultate corecte
  - 3p pentru descrierea implementării în README
- **15p** pentru implementarea variantei **neopt** dintre care:
  - 12p dacă implementarea obține rezultate corecte
  - 3p pentru descrierea implementării în README
- **20p** pentru implementarea variantei **opt\_m** dintre care:
  - 15p dacă implementarea obține rezultate corecte și timpul de calcul pe partiția neahalem este mai mic de 14s pentru testul cu N = 1200
  - 5p pentru descrierea implementării în README
- **9p** dacă cele 3 implementări nu prezintă probleme de acces la memorie
  - Pentru a rezolva acest subpunct va trebui să folosiți **valgrind** cu opțiunile **-tool=memcheck -leak-check=full**
  - Veti include 3 fisiere, **neopt.memory**, **blas.memory** și **opt\_m.memory**, cu output-urile rularii valgrind pentru fiecare din cele 3 variante având ca input fisierul **input\_valgrind**
- **17p** pentru analiza celor 3 implementări folosind **cachegrind**
  - 6p pentru includerea în arhivă a 3 fisiere, **neopt.cache**, **blas.cache** și **opt\_m.cache** reprezentând output-urile rularii **valgrind** cu opțiunile **-tool=cachegrind -branch-sim=yes** pe partiția neahalem având ca input fisierul **input\_valgrind**
  - 6p pentru explicații oferite despre valorile obținute (I refs, D refs, Branches etc.)
  - 5p pentru explicații oferite despre efectul optimizărilor făcute de mană în varianta **opt\_m** asupra valorilor obținute
- **24p** pentru o analiză comparativă a performanței pentru cele 3 variante:
  - 15p pentru realizarea unor grafice relevante bazate pe rularea a cel puțin 5 teste (5 valori diferite ale lui N: adică încă cel puțin două valori diferite de 400, 800 și 1200 pentru N)
  - 9p pentru explicații oferite în README
- **(Bonus)**
  - Veti primi un bonus de maxim 10p dacă timpul de calcul pentru varianta opt\_m este mai mic de 9s pentru testul cu N = 1200
  - Consultați main.c pentru mai multe detalii
  - Bonusul se calculează doar pe partiția neahalem

Depunctări posibile:

- **neopt**
  - nu se ține cont de faptul că A este matrice superior triunghiulară (între -3p și -6p)
- **blas:**
  - nu se ține cont de faptul că A este matrice superior triunghiulară (între -3p și -6p)
  - unul sau mai multe calcule sunt realizate de mană, fără a folosi funcții din BLAS (între -3p și -15p)
  - a fost inclus codul BLAS (fisiere .so, .h., .c și altele) în arhivă temei (-15p)
- **opt\_m**
  - nu se ține cont de faptul că A este matrice superior triunghiulară (între -3p și -6p)
  - înmulțirea matricelor se realizează cu o complexitate diferită decât în cazul variantei neopt (ex. Strassen vs înmulțire normală de matrice) (-15p)
  - timpul de calcul este mai mare decât timpul maxim permis - între -5p și -15p
- **analiza comparativă**
  - graficele nu au legendă / unități de măsură (între -2p și -5p)
  - lipsesc parțial sau complet timpurile de rulare (între -1p și -5p)
  - graficele nu conțin toate datele cerute în enunț (între -2p și -5p)
- **generale:**
  - print-uri de debug în cod (între -1p și -10p)
  - blocuri de cod comentate sau nefolosite (-1p)
  - warning-uri la compilare (între -1p și -3p)
  - cod înghesuit/ilizibil (între -1p și -3p)
  - implementare excesivă de funcții în header (-1p)
  - folosirea de constante hardcodate (-1p)
  - publicarea temei pe GitHub (-100p)

## Precizari incarcare / VMChecker

Arhiva temei va fi încărcată pe  vmchecker.

Structura arhivei va fi următoarea:

```
src
  solver_blas.c
  solver_neopt.c
  solver_opt.c
  Makefile
  ...
cache
  blas.cache
  neopt.cache
  opt_m.cache
memory
  opt_m.memory
  neopt.memory
  blas.memory
```

README  
grafice  
...

La încărcarea pe vmchecker vor fi verificate următoarele:

- corectitudinea rezultatelor pentru cele 3 implementari
- lipsa problemelor de acces la memorie pentru cele 3 implementari
- prezenta unor fisiere .memory valide
- prezenta unor fisiere .cache valide

Celelalte aspecte ale temei (timpi limita pentru **opt\_m** si bonus, README, grafice) vor fi verificate ulterior.

## Precizări și recomandări



Timpul maxim pentru rularea celor 3 teste din fisierul **input** pe partitia neahem folosind oricare din cele 3 variante este de 2 minute. Această limită de timp se referă la rularea întregului program, nu doar la partea intensiv computațională.

- Pentru a simplifica implementarea puteți presupune că  $N$  este multiplu de 40 și că este mai mic sau egal cu 1600.
- În compararea rezultatelor se va permite o eroare absolută de maxim  $10^{-3}$ .
- În cazul variantei **opt\_m** complexitatea trebuie să fie aceeași cu cea din varianta **neopt**.
- Formatul arhivei trebuie să fie **zip**.



Pentru a evita aglomerarea cozii se recomanda rularea de teste pentru valori ale lui  $N$  mai mici sau egale cu 1600.



Se recomandă ștergerea fișierelor coredump în cazul rulărilor care se termină cu eroare pentru a evita problemele cu spațiul de stocare.

În cazul în care job-urile vă rămân "agățate", va recomandam să utilizați de pe [fep.grid.pub.ro](http://fep.grid.pub.ro), comanda

squeue



pentru a vedea câte job-uri aveți pornite, și apoi să utilizați comanda

scancel <job\_id>

pentru a opri un job.

## Resurse

- Ghid pentru folosirea gridului instituțional
- Schelet de cod

Logged in as: Bogdan Mihai TUDORACHE (108609) (bogdan.tudorache99)

asc/teme/tema2.txt - Last modified: 2022/05/04 22:09 by vlad.spoiala

Old revisions

Media Manager Manage Subscriptions Back to top