

Tema 2

- **Deadline soft:** 22 aprilie 2020, ora 23:55. Primiți un bonus de 10% din punctajul obținut pentru trimiterea temei înainte de 22 aprilie 2020, ora 23:55.
- **Deadline hard:** 29 aprilie 2020, ora 23:55. Veți primi o depunere de 10% din punctajul maxim al temei pentru fiecare zi de întârziere, până la maxim 7 zile, adică până pe 29 aprilie 2020, ora 23:55.

Enunț



Se dă următoarea operație cu matrice:

$$C = B \times A^t + A^2 \times B$$

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
- \times este operația de înmulțire
- $+$ este operația de adunare
- A^2 este A ridicat la patrat

Se dorește implementarea operației de mai sus în C/C++ în 5 moduri:

- **blas** - o variantă care folosește una sau mai multe funcții din  BLAS Atlas pentru realizarea operațiilor de înmulțire de matrice. Adunarea matricelor poate fi făcută "de mână". Aceasta implementare va ține cont de faptul că A este o matrice superior triunghiulara.
- **neopt** - o variantă "de mână" fără îmbunătățiri. Aceasta implementare va ține cont de faptul că A este o matrice superior triunghiulara.
- **opt_m** - o variantă îmbunătățită a versiunii **neopt**. Îmbunătățirea are în vedere exclusiv modificarea codului pentru a obține performanțe mai bune.
- **opt_f** - o variantă îmbunătățită obținută prin compilarea codului de la varianta **neopt** cu flag-ul -O3
- **opt_f_extra** - o variantă îmbunătățită obținută prin compilarea codului de la varianta **neopt** cu flag-ul -O3 și alte  flag-uri de optimizare specifice. Se vor avea în vedere flag-uri care acționează asupra unei singure optimizări, nu flag-uri care acționează asupra unui grup de optimizări.

Rulare și testare

Pentru testarea temei vă este oferit un schelet de cod pe care trebuie să-l completați cu implementările pentru cele 4 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 adăugați implementarea variantei **blas**.
- **solver_neopt.c** - în acest fișier trebuie să adăugați implementarea variantei **neopt**.
- **solver_opt.c** - în acest fișier trebuie să adăugați implementarea variantei **opt_m**.
- **Makefile** - Makefile folosit la compilarea cu gcc.
- **input** - fișierul de input care conține 3 teste pentru următoarele valori ale lui N : 400, 800, 1200
- **compare.c** - utilitar ce poate fi folosit pentru a compara două fișiere rezultat. Acest fișier va fi suprascris în timpul corectării și nu trebuie modificat.



Puteți aduce orice modificare scheletului de cod exceptând cele 3 fișiere

Search

Navigare

- Anunțuri
- Regulament
- Echipă
- Orar

Cursuri

- Cursul 01.
- Cursul 02.
- Cursul 03.
- Cursul 04.
- Cursul 05.
- Cursul 06.
- Cursul 07.
- Cursul 08.
- Cursul 09.
- Cursul 10.
- Cursul 11.
- Cursul 12.




Laboratoare

- Laboratorul 01 - Introducere în limbajul Python
- Laboratorul 02 - Fire de execuție în Python
- Laboratorul 03 - Programare concurrentă în Python (continuare)
- Laboratorul 04 - Arhitecturi de Microprocesoare și Sisteme de Calcul
- Laboratorul 05 - Tehnici de Optimizare de Cod – Înmulțirea Matricelor
- Laboratorul 06 - Analiza Performanței 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

-  Cluster cheat-sheet
-  Tutorial video rulare task-uri cluster
-  Tutorial video rulare programe MPI pe cluster



menționate mai sus.

În urma rulării comenzii **make** cu oricare din cele 2 Makefile-uri vor rezulta 5 fișere binare, **tema2_blas**, **tema2_neopt**, **tema2_opt_m**, **tema2_opt_f** și **tema2_opt_f_extra** corespunzătoare celor 5 variante care trebuie implementate.

Rularea se va realiza astfel:

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



unde:

- mod este unul din modulele **blas**, **neopt**, **opt_m**, **opt_f** sau **opt_f_extra**.
- 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 coada **ibm-nehalem.q**. Compilarea se va face folosind **gcc-5.4.0** din modulul **compilers/gnu-5.4.0**. Pentru a încărca modulul pentru GCC trebuie să dați pe una din mașinile din coada **ibm-nehalem.q** următoarea comandă:

```
module load compilers/gnu-5.4.0
```

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 **libs atlas.so.3.10** de pe mașinile din coada **ibm-nehalem.q** disponibile în directorul

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

Fișierele output referință le găsiți aici:

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

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 este cu cel puțin 30% mai mic decât cel al variantei **neopt** pentru testul cu N = 1200
 - 5p pentru descrierea implementării în README
- **20p** pentru implementarea variantei **opt_f_extra** dintre care:
 - 10p dacă implementarea obține rezultate corecte și timpul de calcul este cu cel puțin 5% mai mic decât cel al variantei **opt_f** pentru testul cu N = 1200
 - 10p pentru explicații legate de alegerea flag-urilor și impactul flag-urilor asupra performanței
- **30p** pentru o analiză comparativă a performanței pentru cele 5 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)
 - 15p pentru explicații oferite în README
- **(Bonus)**
 - 20p dacă timpul de calcul al variantei **opt_m** pentru testul cu N = 1200 este cel

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

- [Comutatoare](#)
- [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 și recomandări
 - Resurse

- mult 4 secunde sau
- 10p daca timpul de calcul al variantei **opt_m** pentru testul cu $N = 1200$ este intre 4 si 8 secunde
- Bonusurile se calculeaza doar pe coada ibm-nehalem.q



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

Precizări și recomandări



Timpul maxim pentru rularea celor 3 teste folosind oricare din cele 5 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 vedea ce optimizari sunt prezente in fiecare nivel de optimizare puteti folosi comanda:

```
gcc -Q -Olevel --help=optimizers
```



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, comanda

```
qstat
```



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

```
qdel -f <id-sesiune>
```

unde <id-sesiune> sunt primele cifre din stânga, rezultate după comanda **qstat**.



Sesiunile interactive deschise prin **qlogin** nu sunt permise pe coada **ibm-nehalem.q**. Va trebui sa utilizati **qsub** pentru a folosi aceasta coada.

Resurse

- Cluster cheat sheet
- Schelet de cod

Loading [MathJax]/jax/output/HTML-CSS/jax.js