Tema 2

- **Deadline soft:** 05 mai 2021, ora 23:55. Primiți un bonus de 10% din punctajul obtinut pentru trimiterea temei înainte de 02 mai 2021, ora 23:55.
- **Deadline hard:** 12 mai 2021, ora 23:55. Veți primi o depunctare de 10% din punctajul maxim al temei pentru fiecare zi de întârziere (dupa 05 mai), până la maxim 7 zile, adică până pe 12 mai 2021, ora 23:55.
- **Responsabili:** Vlad Spoiala, Cosmin-Gabriel Samoila, Emil Slusanschi, Andrei-Stefan Trifu, Damian Dinoiu

- Dată publicare: 16 aprilie 2021
- Dată actualizare enunț: 29 aprilie 2021

Enunț

Se dă următoarea operație cu matrice:

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

unde:

- A si B sunt matrice patratice de double de dimensiune N x N
- **A este o matrice superior triunghiulara**
- A^t este transpusa lui A si B^t este transpusa lui B
- $\times$ este operația de înmulțire
- $+$ este operatia de adunare

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

- **blas** - o variantă care folosește una sau mai multe functii din BLAS Atlas pentru realizarea operatiilor de inmultire si 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 implementarile 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** cu oricare din cele 2 Makefile-uri vor rezulta 3 fișere binare, **tema2_blas**, **tema2_neopt** si **tema2_opt_m** corespunzătoare celor 3 variante care trebuie implementate.

Rularea se va realiza astfel:

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

unde:

- mod este unul din modurile **blas**, **neopt**, **opt_m**
- input este fișierul ce contine descrierea testelor.

Fișierul **input** este structurat astfel:

- pe prima linie numărul de teste.
- pe următoarele linii descrierea fiecarui 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 incarca modulul pentru GCC trebuie sa dati pe una din masinile din coada ibm-nehalem.q urmatoarea comanda:

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

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

Pentru variantele **blas**, **neopt** si **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** de pe masinile din coada ibm-nehalem.q disponibile in directorul

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

Nodurile de pe coada Nehalem sunt mai lente decat nodurile de pe alte cozi ce permit acces interactiv. Testele voastre de performanta trebuie realizate pe coada Nehalem deoarece evaluarea temelor se va face pe aceasta coada.

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

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

Fisierul **input** contine 3 teste:

```
3
400 123 out1
800 456 out2
1200 789 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 impărțit astfel:

- **15p** pentru implementarea variantei **blas** dintre care:
  - 12p daca implementarea obtine rezultate corecte
  - 3p pentru descrierea implementarii in README
- **15p** pentru implementarea variantei **neopt** dintre care:
  - 12p daca implementarea obtine rezultate corecte
  - 3p pentru descrierea implementarii in README
- **20p** pentru implementarea variantei **opt_m** dintre care:
  - 15p daca implementarea obtine rezultate corecte si timpul de calcul este mai mic de 12s pentru

testul cu N = 1200
  - 5p pentru descrierea implementarii in README
- **10p** daca cele 3 implementari nu prezinta probleme de acces la memorie
  - Pentru a rezolva acest subpunct va trebui sa folositi **valgrind** cu optiunile **–tool=memcheck –leak-check=full**
  - Veti include 3 fisiere, **neopt.memory**, **blas.memory** si **opt_m.memory**, cu output-urile rularii valgrind pentru fiecare din cele 3 variante avand ca input fisierul **input_valgrind**
- **15p** pentru analiza celor 3 implementari folosind **cachegrind**
  - **5p** pentru includerea in arhiva a 3 fisiere, **neopt.cache**, **blas.cache** si **opt_m.cache** reprezentand output-urile rularii **valgrind** cu optiunile **–tool=cachegrind –branch-sim=yes** avand ca input fisierul **input_valgrind**
  - **10p** pentru explicatii oferite in README
- **25p** pentru o analiza comparativa a performantei pentru cele 3 variante:
  - 15p pentru realizarea unor grafice relevante bazate pe rularea a cel putin 5 teste (5 valori diferite ale lui N: adica inca cel putin doua valori diferite de 400, 800 si 1200 pentru N)
  - 10p pentru explicatii oferite in README
- **(Bonus)**
  - Veti primi un bonus de maxim 10p daca timpul de calcul pentru varianta opt_m este mai mic de 8s pentru testul cu N = 1200
  - Consultati main.c pentru mai multe detalii
  - Bonusul se calculeaza doar pe coada ibm-nehalem.q

Depunctări posibile:

- **neopt**
  - nu se tine cont de faptul ca A este matrice superior triunghiulara (intre -2p si -6p)
- **blas**:
  - nu se tine cont de faptul ca A este matrice superior triunghiulara (intre -2p si -6p)
  - unul sau mai multe calcule sunt realizate de mana, fara a folosi functii din BLAS (intre -3p si -15p)
  - a fost inclus codul BLAS in arhiva temei (-15p)
- **opt_m**
  - nu se tine cont de faptul ca A este matrice superior triunghiulara (intre -2p si -6p)
  - inmultirea matricelor se realizeaza cu o complexitate diferita decat in cazul variantei neopt (ex. Strassen vs inmultire normala de matrice) (-15p)
- **analiza comparativa**
  - graficele nu au legenda / unitati de masura (intre -2p si -5p)
  - lipsesc partial sau complet timpii de rulare (intre -1p si -5p)
  - graficele nu contin toate datele cerute in enunt (intre -2p si -5p)
- **generale**:
  - print-uri de debug in cod (intre -1p si -10p)
  - blocuri de cod comentate sau nefolosite (-1p)
  - arhiva nu are formatul .zip (-2p)
  - warning-uri la compilare (intre -1p si -3p)
  - cod inghesuit/ilizibil (intre -1p si -3p)
  - implementare excesiva de functii in headere (-1p)
  - folosirea de constante hardcodate (-1p)
  - Makefile gresit (intre -5p si -10p)
  - publicarea temei pe GitHub inainte de expirarea deadline-ului hard (-100p)

Precizări și recomandări

Timpul maxim pentru rularea celor 3 teste din fisierul **input** 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, 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**.

Resurse

- Cluster cheat sheet
- Schelet de cod