

# Factorizarea LU

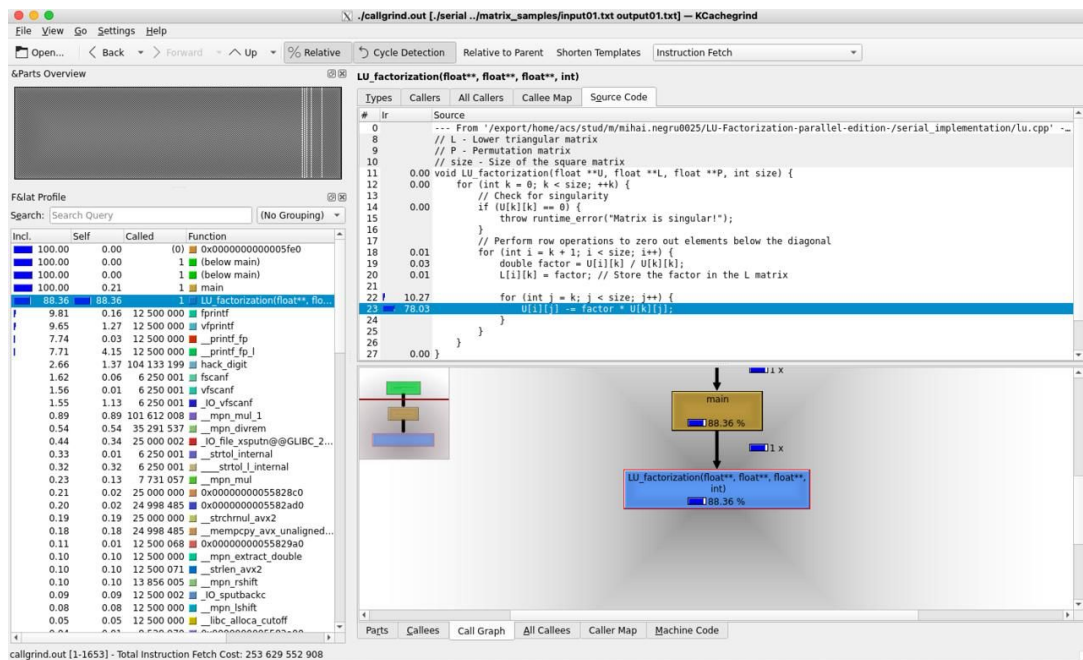
Negru Mihai 343C1  
Ilinca Sebastian 341C1

# Descriere algoritm

- Descompune o matrice pătratică ( $A$ ) în două matrici triunghiulare superior ( $U$ ) și inferior ( $L$ ).
- Matricile  $L$  și  $U$  se construiesc simulatan, parcurgând matricea  $A$ .
- Ulterior matricile triunghiulare sunt folosite pentru rezolvarea sistemelor de ecuații.
- În metodele numerice descompunerea LU se folosește pentru metoda lui Newton (metoda tangentei) și pentru analiza rețelelor electrice (Kirchoff)
- Complexitatea spațială al algoritmului este de  $O(N^2)$ .
- Complexitatea temporală al algoritmului este de  $O(N^3)$ .

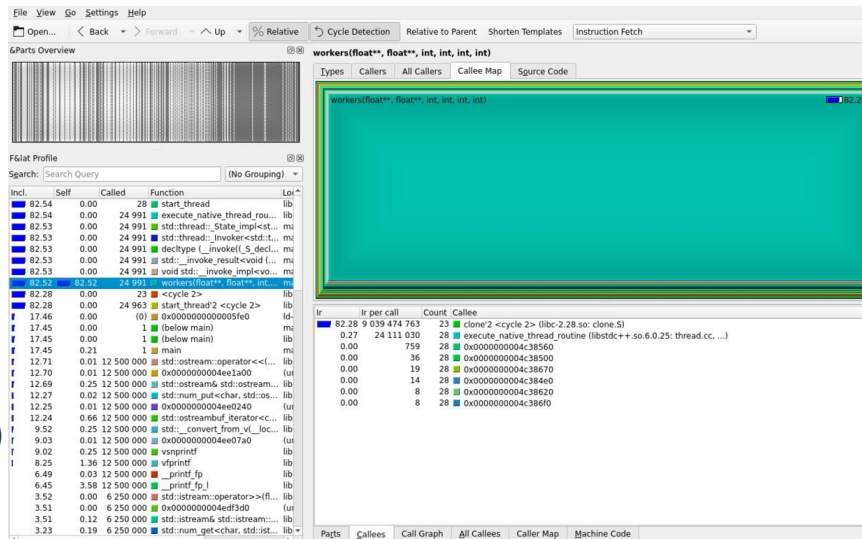
# LU Serial

- Complexitatea temporală  $O(N^3)$ , se va paraleliza doar bucla interioară astfel ajunge la o complexitate teoretică de  $O(N * N^2 / P)$ .
- Pentru testele de dimensiune 2500 s-a obținut un rezultat de **27194742** us.
- Pentru testele de dimensiune 5000 s-a obținut un rezultat de **194177345** us.

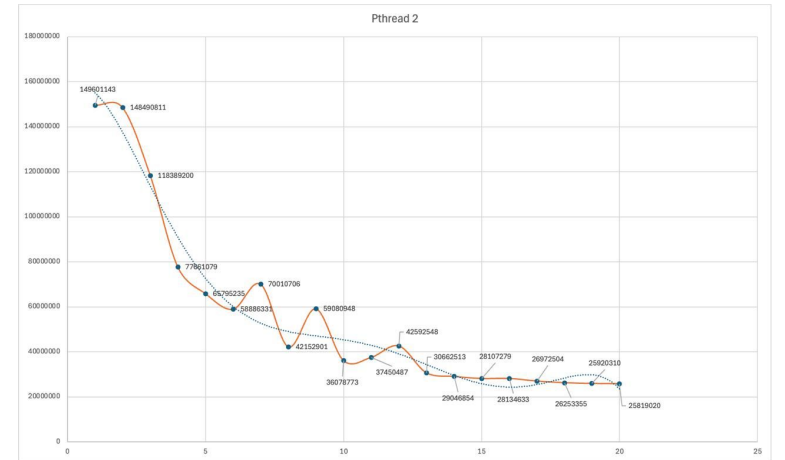
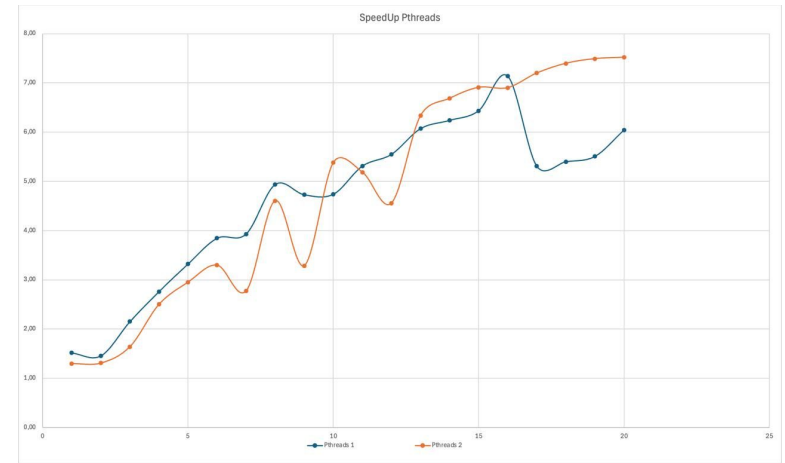
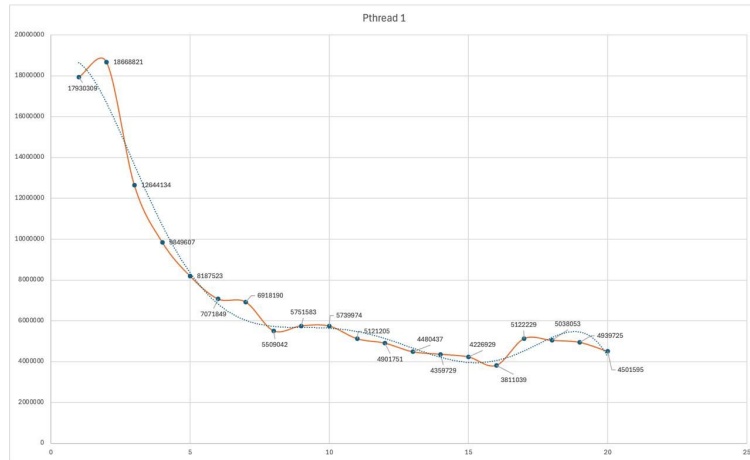


# LU Pthreads

- Cu mențiunea de mai sus pentru un număr de thread-uri folosit  $P$ , se vor porni în  $N$  etape,  $N * P$  thread-uri (numiți workeri).
- Spațiul de lucru al thread-urilor per etapă este intervalul  $[k + 1; N]$
- Dat fiind faptul că nu există comunicare între thread-uri și nici posibilitatea unui data race, se va face o împărțire statică a spațiului de lucru.
- Thread-ul cu index-ul  $i$ , va avea ca spațiu de lucru intervalul  $[k + 1 + i(N - (k + 1) / P); k + 1 + (i + 1)(N - (k + 1) / P)]$
- În caz că numărul  $P$  nu este un divizor al lui  $N$ , ultimul thread, va prelucra cu  $(N - (k + 1)) \% P$ , mai multe elemente



# LU Pthreads

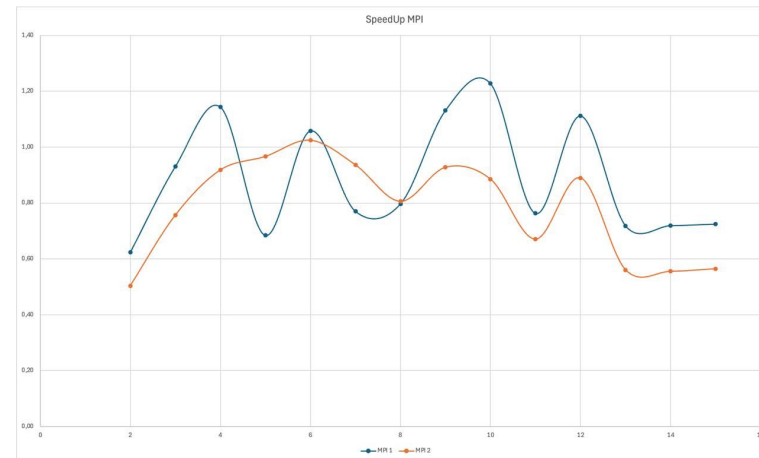
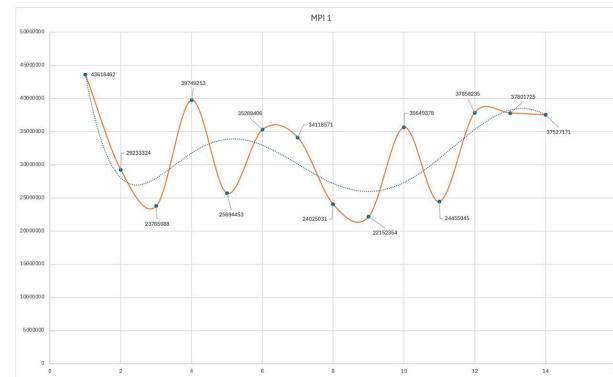
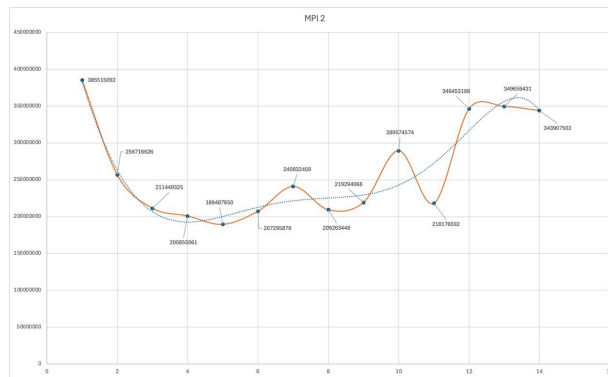


# LU MPI

- Arhitectura folosită pentru implementarea MPI **Boss-Worker**.
- Se folosește aceeași metodologie de împărțire a spațiului de lucru ca la pthreads.
- Fiecare worker calculează subproblema (din factorizarea LU) și comunică rezultatul către **master**.
- Comunicarea dintre master și workeri are loc prin proceduri de MPI asincrone.

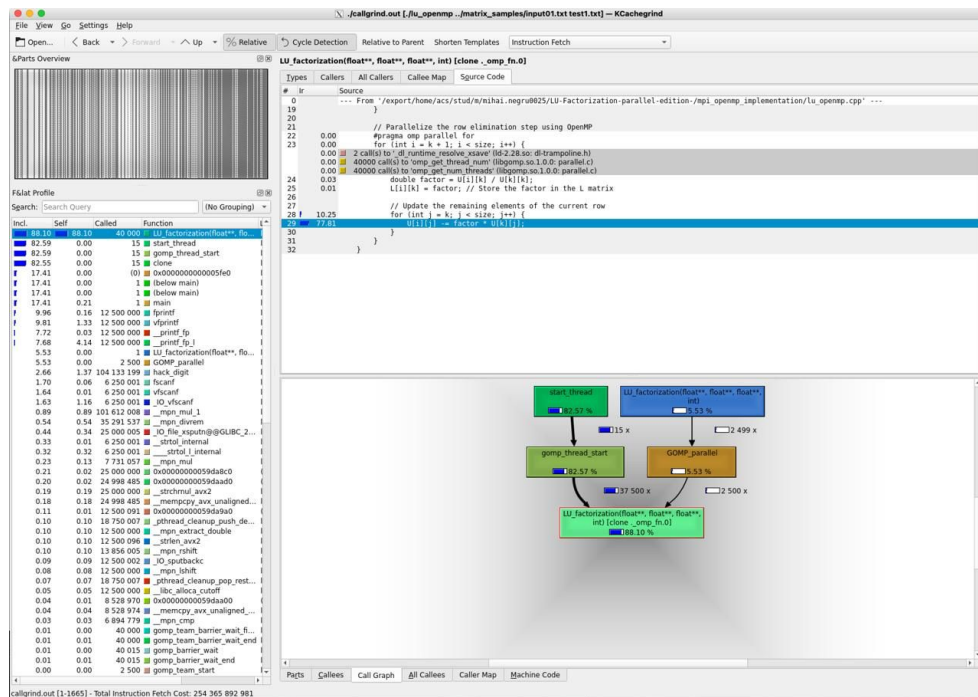
# LU MPI

- În general observăm că nu avem un speedup mai mare decât 1, ceea ce sugerează că timpul algoritmului de MPI durează mai mult decât cel serial.
- Din cauza comunicării dintre workeri și master a matricii rezultante prin apelurile de MPI, acest lucru îngreunează sarcina procesării.



# LU OpenMP

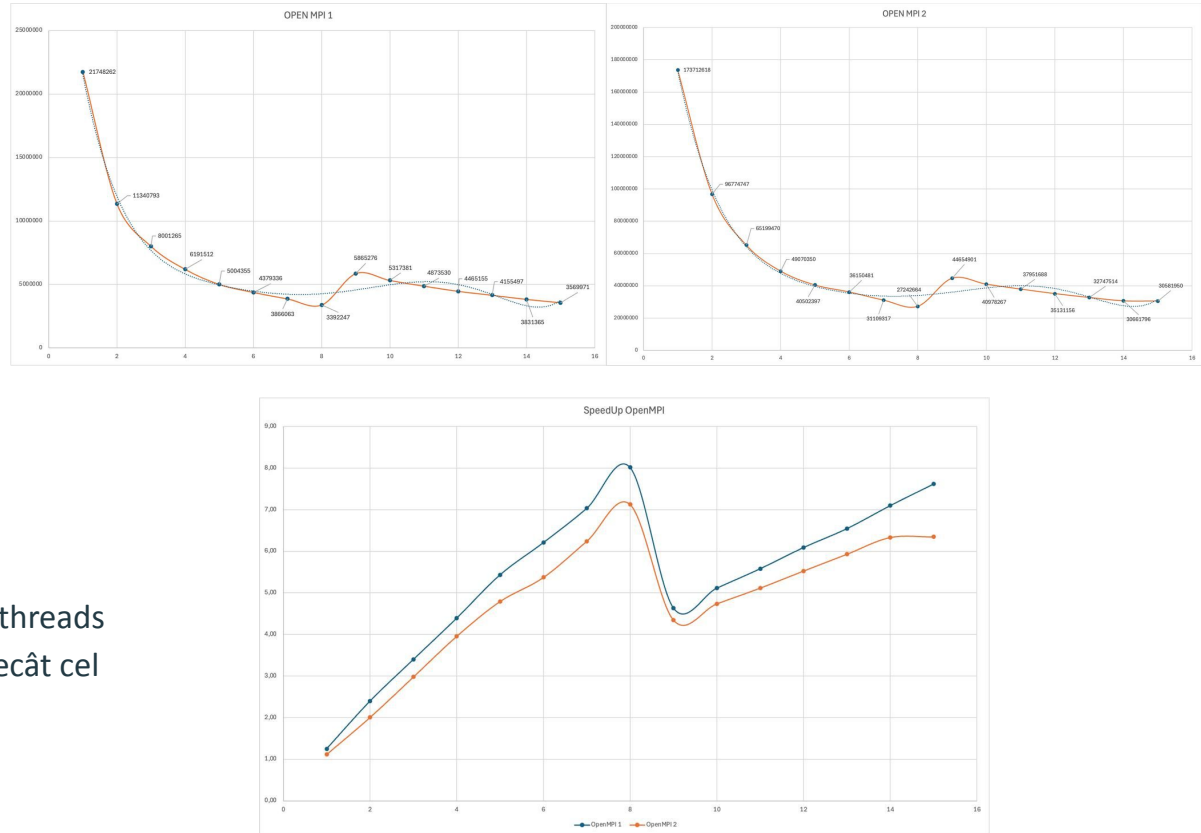
- La fel ca și la pthreads a fost folosită o împărțire statică (default) a mediului de lucru paralelizându-se doar bucla interioară al algoritmului.
- Datorită faptului că paralelizăm doar bucla interioară algoritmul nu necesită sincronizări (mutex, atomic, etc).





# LU OpenMP

- SpeedUp mai mare decât la pthreads
- Un speedup mai consistent decât cel de la pthreads.



# Concluzie

- În urma explorării testelor s-a observat că metoda de **MPI**, nu a funcționat din cauza traficului intensiv între workeri și master.
- Deși varianta de pthreads și openmp au aceeași arhitectură și aceeași împărțire a spațiului de lucru, openmp a adus un speedup mai consistent și mai mare decât varianta de pthreads, astfel pentru acest algoritm rutina de openmp este mai bine implementată decât implementarea specifică cu pthreads.
- În urma testărilor a câte **20** de core-uri pentru openmp și pthreads se observă o apropiere asimptotică inferioară, însă luând în considerare dimensiunile matricilor pentru testare, asimptota nu a fost atinsă.
- Astfel luând în considerare speedup-ul pentru pthreads și openmp de **(8 - 8.5)** paralelizarea a adus un plus valoarea algoritmului serial.
- Toate testele au fost rulate pe coada de **haswell**.



Muhtumim!