Calcularea inversei unei matrice folosind descompunerea LU

Studenți: Dumitru Bianca Ștefania 343C1 & Dumitrescu Bogdan 343C1

Îndrumător: Tudor Calafeteanu

Ianuarie 2025 - Arhitecturi și Prelucrări Paralele

Descrierea problemei

Inversa unei matrice

$A * A^{-1} = I$

Factorizarea LU

Anumite matrice pătratice suporta factorizarea în 2 matrice distincte (L - lower triangular, U - upper triangular) → A = L * U

Calcularea inversei folosind LU

A * X = I → L * U * X = I

2*n ecuatii liniare:

- L * Y = I (folosire forward substitution pentru a determina Y)
- U * X = Y (folosire backward substitution pentru a determina X)

Factorizarea LU

```
for i in range(0, n):
  for j in range(0, n):
    for k in range(0, i):
      compute L[j][i]
      compute U[i][j]
```

Complexitate timp $O(n^3)$
Complexitate spațiu $O(n^2)$

Calcularea inversei

```
def forward_substitution(l, e, y):
  for i in range(0, n):
    for j in range (0, i):
      compute y[i]

def backward_substitution(u, y, x):
  for i in range(n-1, 0):
    for j in range (i+1, n):
      compute x[i]

for i in range(0. n):
  forward_substitution(l, e, y);  // Solve L * y = e
  backward_substitution(u, y, x); // Solve U * x = y
  for j in range(0, n):
    a_inv[j][i] = x[j]
```

Complexitate timp $O(n^3)$
Complexitate spațiu $O(n^2)$

Inputs

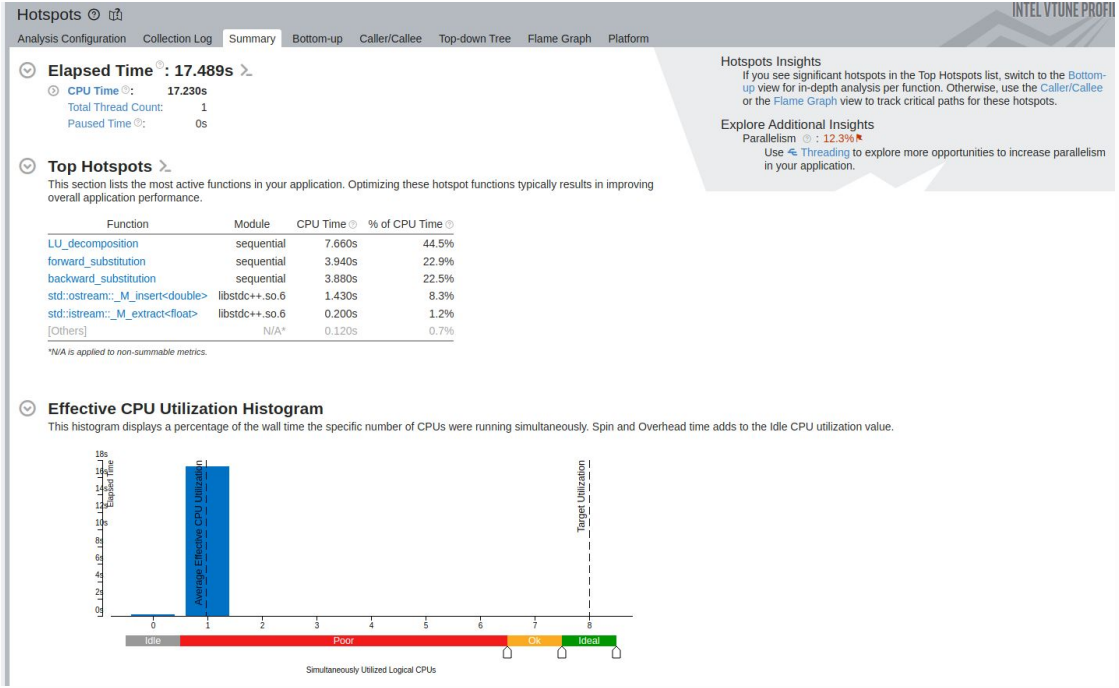
Trei dimensiuni ale matricelor pentru inputuri:

- Small - 700 x 700
- Medium - 1100 x 1100
- Large - 1500 x 1500

Profiling

Am folosit Intel VTune, dar și măsurarea efectivă a timpului de rulare pentru realizarea diverselor grafice comparative.

Rezultatele algoritmului secvențial



Hotspots

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Flame Graph | Platform

INTEL VTUNE PROFI

Elapsed Time: 17.489s

CPU Time: 17.230s
Total Thread Count: 1
Paused Time: 0s

Hotspots Insights

If you see significant hotspots in the Top Hotspots list, switch to the Bottom-up view for in-depth analysis per function. Otherwise, use the Caller/Callee or the Flame Graph view to track critical paths for these hotspots.

Explore Additional Insights

Parallelism: 12.3%

Use Threading to explore more opportunities to increase parallelism in your application.

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
LU_decomposition	sequential	7.660s	44.5%
forward_substitution	sequential	3.940s	22.9%
backward_substitution	sequential	3.880s	22.5%
std::ostream::_M_insert<double>	libstdc++.so.6	1.430s	8.3%
std::istream::_M_extract<float>	libstdc++.so.6	0.200s	1.2%
[Others]	N/A*	0.120s	0.7%

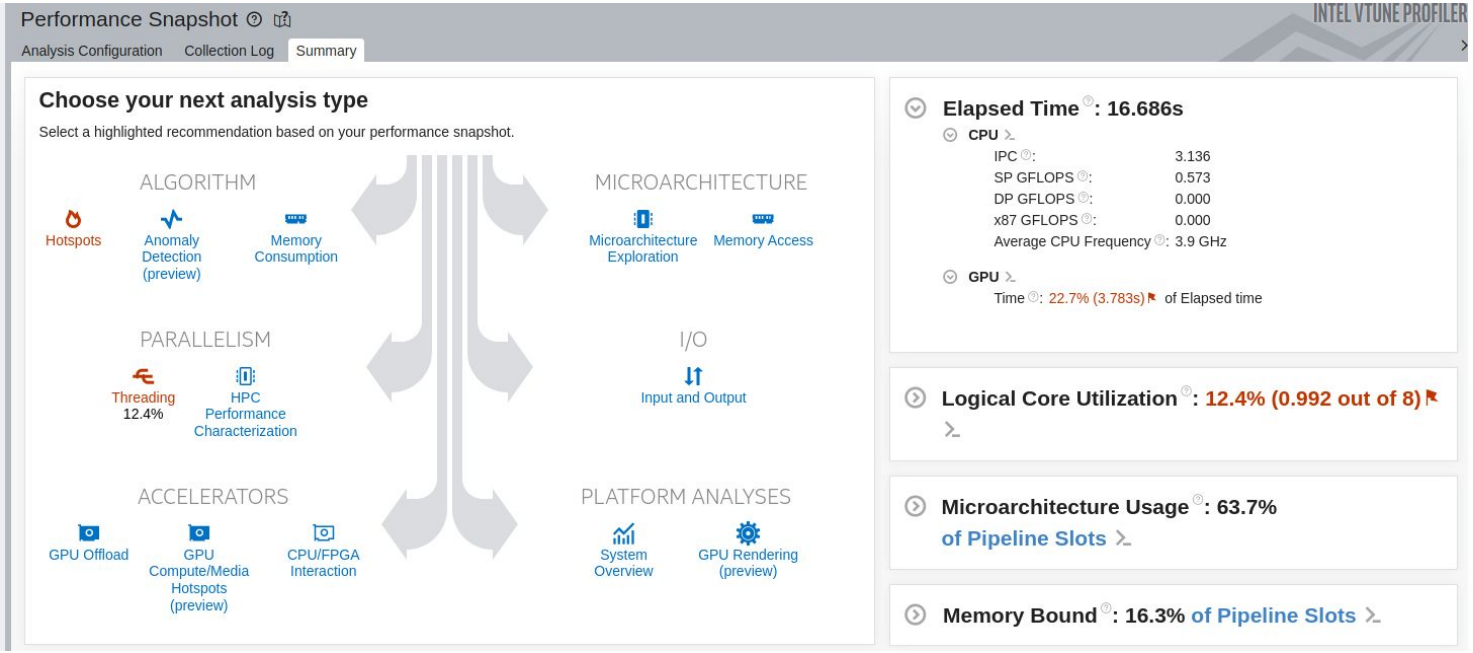
*N/A is applied to non-summable metrics.

Effective CPU Utilization Histogram

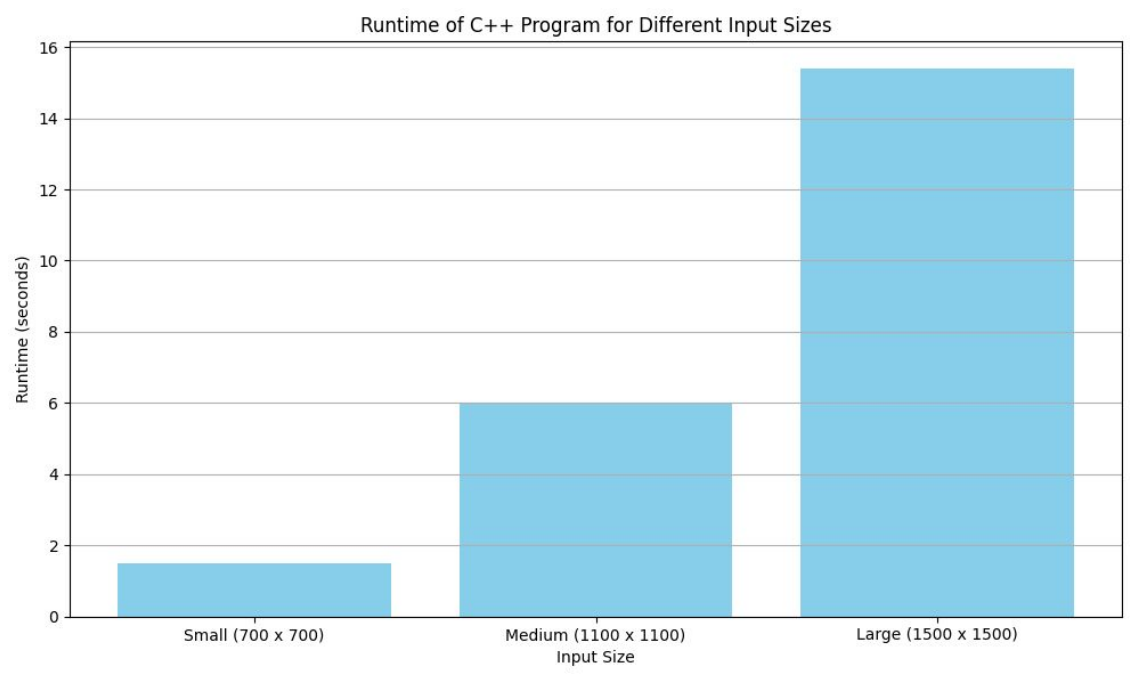
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

Rezultatele algoritmului secvențial

Chiar si profiler-ul sugereaza folosirea threadingului:



Rezultatele algoritmului secvențial

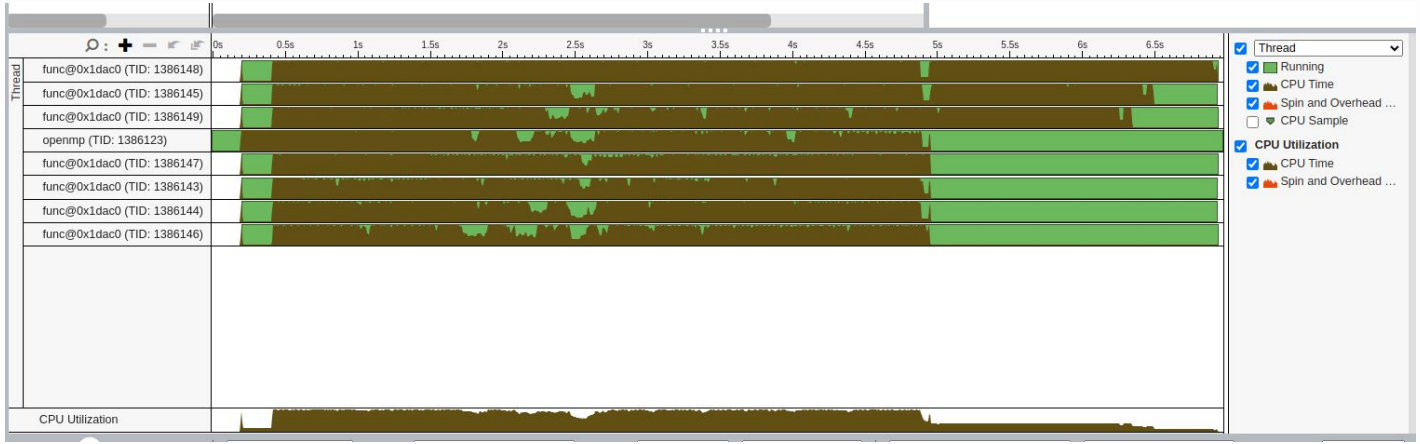


Paralelizarea cu OpenMP

- Paralelizare pe **rândurile matricei L** și **coloanele matricei U** folosind **scheduler dynamic** cu chunk size 1 pentru a echilibra sarcinile, deoarece unele iterații sunt mai complexe (mai multe operații în bucla k).
- **Fiecare coloană a matricii inverse** este calculată independent de un thread (folosind scheduler static).
- **Inițializare Matrice** & **Scriere Fișiere** → **Paralelizarea pentru alocarea liniilor** matricei **si scrierea paralelă** a matricilor L, U și A^{-1} folosind directive OpenMP reduce timpul de executie
- **Număr Threaduri** → Setat la **numărul de nuclee fizice disponibile (8)** pentru a evita supraîncărcarea și comutarea excesivă între threaduri.

Paralelizarea cu OpenMP

- **CPU utilization** este aproape maximal pe parcursul executiei programului



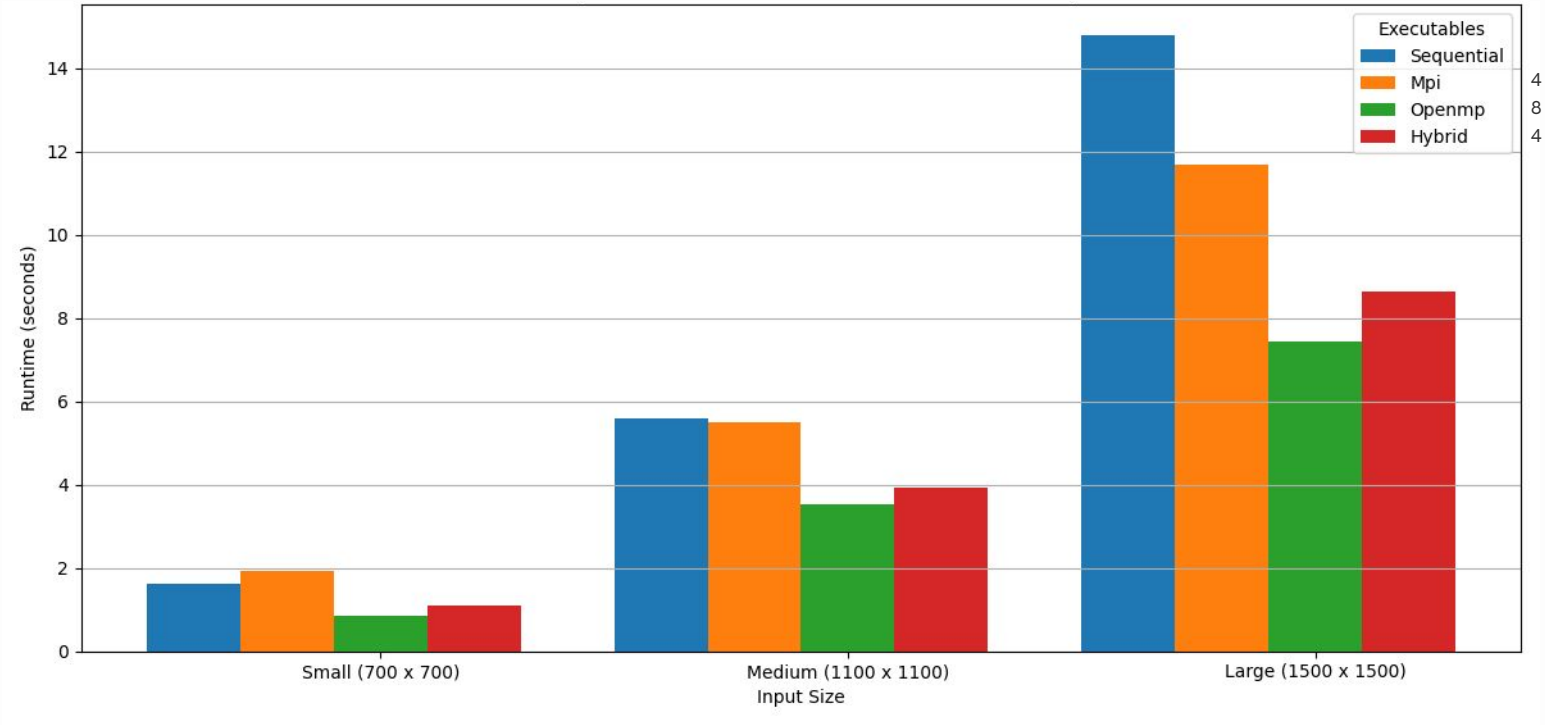
Paralelizarea cu MPI

- **Algoritmul original păstrat** → S-au evitat tehnici avansate (pivotare, blocuri) pentru a menține logica algoritmului inițial.
- **Paralelizarea LU Decomposition** → Doar **bucla interioară** (k) a fost paralelizată, cu un **prag de 500** pentru a reduce overhead-ul pe matrici mici.
- **Calculul inversei** → Fiecare proces calculează **coloane diferite** din matricea inversă, rezultatele fiind centralizate la procesul root.
- **Limitări la I/O** → Doar procesul root scrie rezultatele pentru a evita transferurile costisitoare între procese.

Paralelizarea hibrida (MPI + OpenMP)

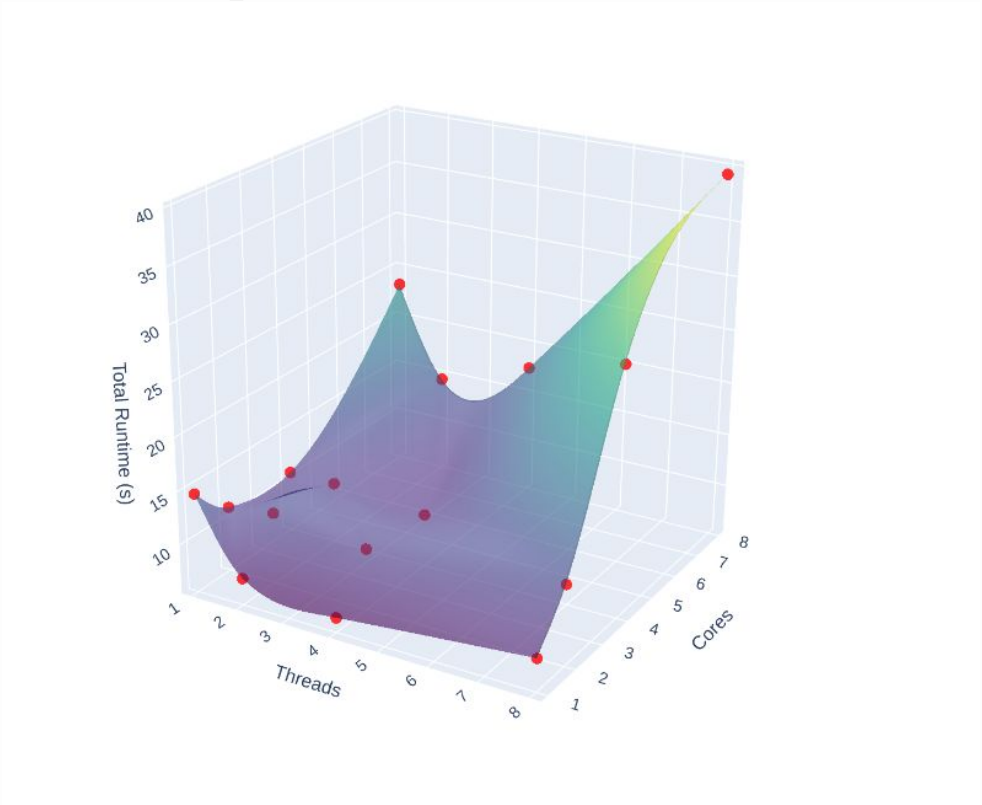
- **Algoritmul original păstrat** → S-au evitat tehnici avansate (pivotare, blocuri) pentru a menține logica algoritmului inițial.
- **Paralelizarea LU Decomposition** → Paralelizare asemănătoare cu OpenMP
- **Calculul inversei** → Fiecare proces calculează **coloane diferite** din matricea inversă, rezultatele fiind centralizate la procesul root.
- **Limitări la I/O** → Doar procesul root scrie rezultatele pentru a evita transferurile costisitoare între procese.

Evaluare comparativă



4 procese
8 threaduri
4 procese, 4 threaduri

Evaluare comparativă



Challenges

Complexitatea Configurării VTune

- Necesitatea unei configurări meticuloase și înțelegerii detaliate a parametrilor.
- Setările inițiale au fost consumatoare de timp din cauza documentației incomplete, fiind necesare soluții găsite pe forumuri precum Stack Overflow.

Paralelizare Eficientă

- Echilibrarea între optimizarea performanței și menținerea structurii originale a algoritmului a fost o provocare.
- Încercarea de a optimiza algoritmul MPI pentru un input size compromitea performanța pentru alte inputuri.

Sensibilitatea Performanței la Hardware

- Diferite variante (hibrid, secvențial, OpenMP) au avut performanțe variate pe mașini diferite.
- De exemplu, versiunea OpenMP a fost cea mai rapidă pe laptopul Stefaniei, dar o variantă cu 8 nuclee și 2 fire a fost mult mai rapidă pe laptopul lui Bogdan.
- Evidentă dependență de hardware și necesitatea testării pe sisteme diverse.

Multumim!