Universitatea din București
Facultatea de Matematică și Informatică
Specializarea Tehnologia Informației

Proiect Algoritmi Paraleli

Studenți:

Bahrim Dragoș

Ionescu Alexandru-Theodor

Maxim Tiberiu

București

2023

Cuprins

1. Descrierea problemei

Sistemele de ecuații liniare reprezintă o problemă matematică fundamentală în multe domenii, precum ingineria sau științele aplicate. Acestea apar atunci când trebuie să se rezolve un set de ecuații liniare simultane, unde necunoscutele sunt variabilele sistemului. Pentru sisteme mari și complexe, soluțiile secvențiale pot fi costisitoare din punct de vedere al timpului de calcul și pot necesita resurse computaționale semnificative. De aceea, se recurge la algoritmi paraleli pentru a rezolva aceste sisteme într-un mod mai eficient și rapid.

Datele problemei includ matricea coeficienților sistemului de ecuații liniare și vectorul termenilor liberi. Acestea sunt date de intrare și trebuie procesate pentru a găsi soluția sistemului. De asemenea, numărul de ecuații și necunoscutele sistemului sunt importante pentru dimensionarea algoritmului și distribuirea datelor pe diferite procese paralele, însă acestea sunt ușor de obținut din datele inițiale și nu necesită input suplimentar.

Scopul rezolvării sistemelor de ecuații liniare prin algoritmi paraleli constă în găsirea soluției sistemului într-un timp mai scurt, utilizând în mod eficient resursele disponibile. Prin paralelizare, se urmărește accelerarea procesului de rezolvare, creșterea eficienței și împărțirea încărcării de calcul între diferitele procesoare.

Soluțiile secvențiale pentru sistemele de ecuații liniare pot deveni **impracticabile** pentru sisteme mari sau atunci când se dorește obținerea unor rezultate într-un timp scurt. Algoritmii secvențiali, cum ar fi *Metoda Eliminării Gaussiene* pot necesita o complexitate computațională ridicată și pot fi dificil de implementat pe un singur procesor.

Pentru cele trei metode de rezolvare a sistemelor de ecuații liniare care vor fi studiate în această lucrare, *Gaussian Elimination*, *Jacobi Iterations* și *Gauss-Seidel*, următoarele **măsuri de performanță** pot fi luate în considerare:

- **Accelerarea** se referă la cât de mult mai rapidă este soluția paralelă în comparație cu soluția secvențială. Se poate calcula raportul dintre timpul de execuție al soluției secvențiale și timpul de execuție al soluției paralele pentru același set de date. O accelerare mai mare indică o eficiență mai mare a algoritmului paralel.
- **Eficiența** măsoară utilizarea eficientă a resurselor de calcul disponibile într-un sistem paralel. Se poate defini drept raportul între accelerare și numărul de procesoare utilizate.

O eficiență mai mare indică o utilizare mai optimă a resurselor și, implicit, o performanță mai bună.

- **Balansarea** sarcinii se referă la distribuirea egală a sarcinilor de calcul între procesele paralele. Este important ca toate procesele să lucreze în mod echitabil și să finalizeze sarcinile aferente acestora aproximativ în același timp. Unele metode pot necesita o împărțire inițială a datelor în bucăți egale, în timp ce altele pot necesita adaptări în timpul execuției pentru a menține o distribuție echitabilă a sarcinilor.
- **Scalabilitatea** se referă la capacitatea algoritmului paralel de a se adapta și de a funcționa eficient pe un număr variabil de procesoare. O soluție paralelă scalabilă ar trebui să ofere o performanță similară pe măsură ce numărul de procesoare crește sau scade.
- Algoritmii paraleli implică de obicei comunicare între procese pentru a sincroniza și a distribui datele. **Overhead**-ul de comunicare se referă la timpul și resursele necesare pentru a efectua această comunicare. Este important să se minimizeze overhead-ul de comunicare pentru a obține o performanță eficientă a algoritmului paralel.
- Alegerea **topologiei soluției paralele** poate influența performanța și comunicarea între procesoare. Diferite topologii, cum ar fi topologia liniară, topologia în cerc sau topologia grilă, pot afecta latența, overhead-ul de comunicare și balansarea sarcinii. Este important să se analizeze caracteristicile problemei și ale algoritmului pentru a alege topologia potrivită care să maximizeze performanța soluției paralele.

2. Specificația soluției

Una din problemele cu care ne putem confrunta este reprezentată de alegerea numărului optim de procesoare. Astfel, atunci când construim o soluție, aceasta va necesita alegerea unei topologii specifice ce ar implica un număr fix de procesoare.

În mod normal, crearea unei constrângeri asupra numărului de procesoare duce la posibilitatea de a optimiza operațiile de lucru, dar și de comunicare, întrucât se cunosc detalii despre cu cine și cu ce se lucrează.

Totuși, acest lucru reprezintă o constrângere pe care am ales să o evităm, astfel dorind să oferim o flexibilitate asupra numărului de procesoare. Prin acest fapt, se limitează posibilitatea utilizării topologiilor puternic structurate de tip grid sau hipercub, în timp ce topologii de tipul inel sau mesh pot accepta această variabilitate.

Limitarea acestei implementări duce la formarea unor limite practice asupra numărului de procesoare. Spre exemplu, dacă ar trebui să calculată suma a două matrice, numărul maxim de procesoare care ar putea să execute această operație, într-un mod paralel, este egal cu numărul de elemente ale matricei, întrucât, dacă ar fi folosite mai multe, nu s-ar putea efectua o distribuire de sarcini către acestea.

Dependențele minimale necesare execuției programelor includ sisteme de calcul ce au un număr de minim un nucleu pe procesor pentru a obține o eficiență maximă a paralelizarii sau clustere formate din sisteme de calcul ale căror agregare duce la obținerea unui grup de procesoare.

Aceste sisteme trebuie să fie compatibile cu o versiune recentă de Python 3 *(versiunea 3.11 a fost testată),* însă, deoarece nu sunt dependențe specifice de această versiune, se pot folosi și versiuni anterioare. Pentru versiunea de Python aleasă sunt esențiale dependențele necesare rulării algoritmilor în mod eficient, aceasta incluzând **numpy**, pentru ușurarea calculului numeric, **mpi4py**, pentru comunicarea prin protocolul MPI, și **simpy**, pentru eficientizarea realizării calculului simbolic. Acestea pot fi instalate folosind **pip**, package managerul pentru Python. De asemenea, sistemul are nevoie de disponibilitatea comenzii **mpirun**.

Din punct de vedere hardware, obiectivul a fost reprezentat de eliminarea oricărei restricții asupra numărului de procesoare. Din acest motiv nu există o limită inferioară pentru

majoritatea algoritmilor, iar cea superioară este dată de limitarea teoretică a algoritmului - nu mai mult decât ce se poate paraleliza.

Totuși, pentru unii algoritmi, în crearea soluțiilor, astfel încât să îndeplinească acest obiectiv, sunt necesare cantități ridicate de memorie. Întrucât nu ne putem baza pe o structură specifică, asta necesită o comunicare integrală a datelor. Eficientizarea pe anumite distribuții poate fi făcută adoptând diferite topologii, în care ne putem aștepta la calcularea datelor de care avem nevoie.

Limitările soluțiilor sunt reprezentate în principal de limitări de calcul, adică de problemele apărute atunci când se operează cu numere foarte mici sau cu multe zecimale. Din acest motiv, până și *Metoda Gauss prin Eliminare* poate duce la erori de calcul ce duc la imposibilitatea obținerii unei soluții sau a unei soluții precise.

Evaluarea soluției poate fi efectuată prin compararea diferiților timpi de execuție pentru diferite numere de procesoare în comparație cu dimensiunea datelor de intrare, ajungând chiar și la compararea codului serial sau cu alte tipuri de topologii specifice.

Constrângerile ce pot apărea sunt declanșate de alegerea arhitecturală de a avea flexibilitate în numărul de procesoare. Astfel comunicările sunt mai mari și mai lente, fapt ce duce la îngreunarea programului și la creșterea costul de rulare.

Interacțiunea cu programul se va putea face printr-o interfață web ce dă acces la parametrii programului și permite trimiterea inputului către program. De asemenea va avea o metodă utilitară de generare de sisteme de ecuații de diferite dimensiuni pentru a avea la îndemână date de test pentru program.

3. Design

3.1. Metoda Gauss prin Eliminare

Metoda Gauss prin eliminare (cunoscută și sub numele de *eliminare Gauss-Jordan*) este un algoritm utilizat pentru rezolvarea sistemelor de ecuații liniare. Scopul metodei este de a transforma sistemul de ecuații inițial într-unul echivalent, dar într-o formă simplificată numită formă echivalentă escalonată.

Pașii principali ai metodei Gauss prin eliminare:

- *Reprezentarea sistemului de ecuații:* Sistemul de ecuații liniare este reprezentat sub forma unei matrice extinse, care include coeficienții variabilelor și termenii liberi (constantele) într-o singură matrice.
- *Etapa de eliminare:* Pentru fiecare coloană, începând cu prima, se aplică operații pe rânduri pentru a obține zerouri sub elementele de pe diagonală. Aceasta implică împărțirea unei ecuații cu coeficientul corespunzător de pe diagonală și scăderea acestei ecuații din celelalte ecuații pentru a elimina variabila curentă.
- *Etapa de pivotare, opțională:* În timpul etapei de eliminare, poate apărea necesitatea de a efectua operații de pivotare pentru a evita împărțirea la zero sau pentru a evita erori de precizie în calcule. Aceasta implică interschimbarea rândurilor pentru a aduce un element nonzero pe diagonală.
- *Forma echivalentă:* După finalizarea etapei de eliminare, sistemul de ecuații se află într-o formă simplificată. Aceasta constă într-o matrice triunghiular superioară, în care toate elementele de sub diagonală principală sunt zero.
- *Etapa de retrosubstituție:* Se rezolvă sistemul de ecuații obținut în forma echivalentă prin aplicarea retrosubstituției. Aceasta implică substituirea valorilor cunoscute ale variabilelor în ecuații pentru a determina valorile necunoscute.

După cum se poate observa, metoda Gauss prin eliminare reprezintă o metodă inerent secvențială întrucât necesită linia precedentă cu care se generează zerouri sub diagonala principală.

Totuși, au existat dorințe de a paraleliza acest algoritm. Din [1] se poate extrage un algoritm ce încearcă să paralelizeze obținerea ecuațiilor într-o metodă bazată pe divide et impera. Dezavantajul acestei metode implică efectuarea unui număr ridicat de calcule.

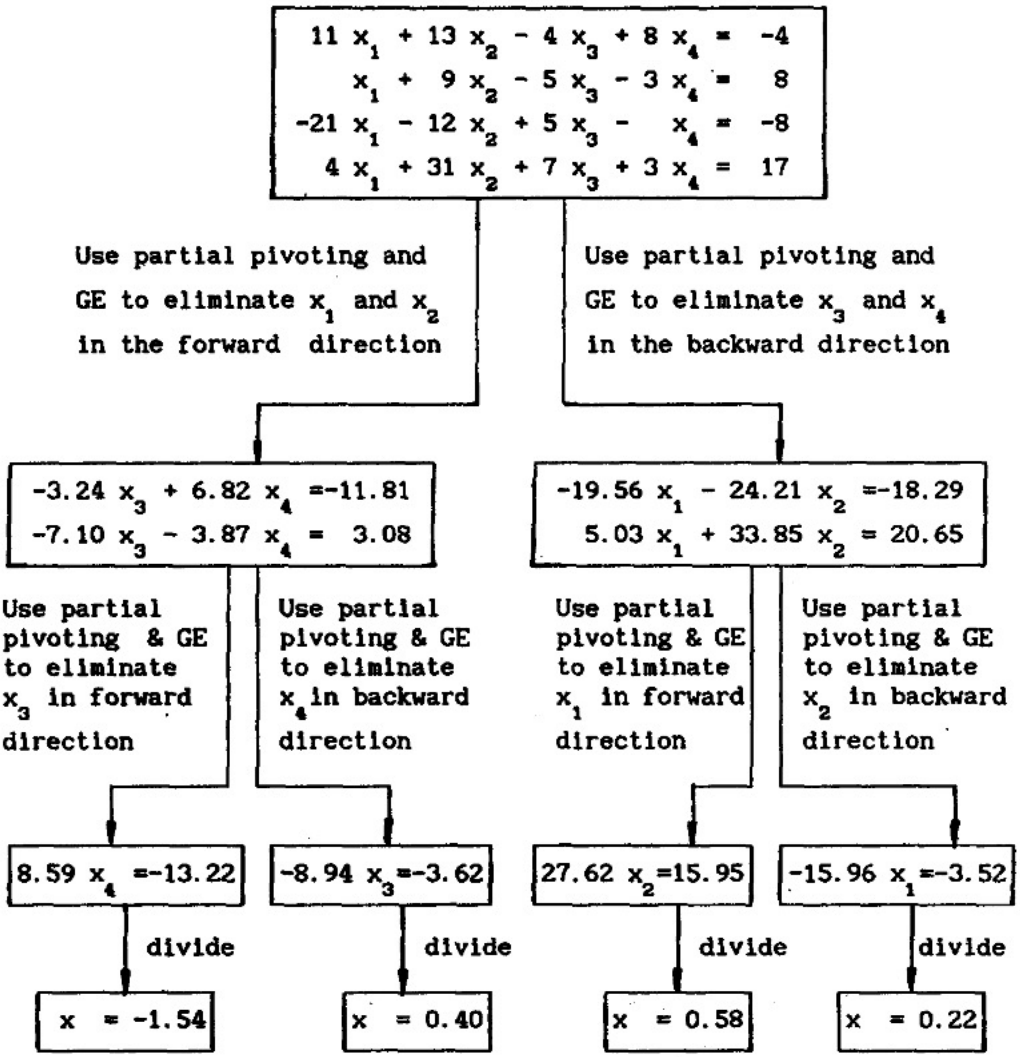

Figure 1. Solving a 4 × 4 system using the SGE algorithm.

Figura 3.1. Algoritm de paralelizare bazat pe divide et impera.

O altă metodă poate fi extrasă din [2]. Aceasta include maparea fiecărui element din matrice asupra unui grid, fiecare element reprezentând o operațiune din cadrul procesului de eliminare ce poate fi executat în paralel. Problema care apare cu această implementare este legată de comunicările frecvente ce apar din cauza necesității operațiilor anterioare pentru element.

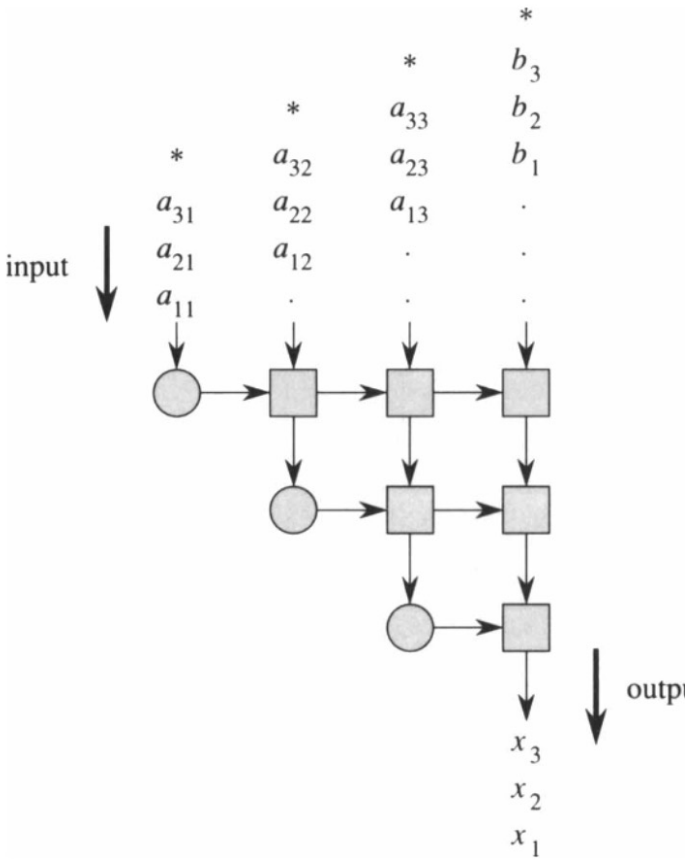

Figura 3.2. Metoda de funcționare a algoritmului propus de Leighton.

Niciuna din aceste metode nu ar îndeplini obiectivul propus, astfel a fost necesară o implementare particularizată bazată pe această ultimă descriere. Astfel, se păstrează algoritmul de calcul însă aplicat pentru un număr variabil de procesoare. Se poate ajunge la acest fapt prin sincronizarea elementelor calculate după fiecare iterație. Dezavantajul acestei metode reprezintă costul de memorie asociat, întrucât fiecare procesor trebuie să rețină matricea creată anterior.

$$C_{jk}^{(i)} = C_{jk}^{(i-1)} - \sum_{\ell=1}^{n} N_{j\ell}^{(i)} C_{\ell k}^{(i-1)} = C_{jk}^{(i-1)} - N_{ji}^{(i)} C_{ik}^{(i-1)} = C_{jk}^{(i-1)} - \frac{C_{ji}^{(i-1)}}{C_{ii}^{(i-1)}} C_{ik}^{(i-1)}.$$

Figura 3.3. Formula de calcul a elementelor din fiecare iterație.

Fiecare procesor are asociat un număr de elemente distribuite, în mod egal, asupra tuturor procesoarelor, acesta efectuând calculele și, la final, creându-se sincronizarea totală a matricei C pe toate procesoarele. Din acest motiv, o topologie de tip *Fully Connected Mesh* este cea mai convenabilă întrucât reduce costul de comunicare de la un procesor la altul.

La final, un singur procesor va efectua *back substitution* pentru a afla soluția sistemului. Acest lucru se întâmplă deoarece nu există un beneficiu în a paraleliza acest task (se putea ca fiecare procesor să își cacheze calculele, însă nu a fost necesar).

3.2. Metoda Jacobi

Metoda Jacobi reprezintă o metodă iterativă utilizată pentru rezolvarea sistemelor de ecuații liniare și se bazează pe ideea de a aproxima soluția sistemului prin iterații succesive, actualizând valorile necunoscutelor până când se obține o soluție convergentă.

Pentru a aplica metoda Jacobi, se presupune că se dă sistemul de ecuații liniare cu forma următoare: $Ax = b$, (A reprezentând matricea coeficienților, x - vectorul de necunoscute, iar b vectorul termenilor liberi). Pașii necesari rezolvării sunt:

- Se va nota cu $x(t)$ valoarea necunoscutelor, unde t este un număr întreg ce reprezintă valoarea iterației curente.
- Se începe prin a aproxima soluțiile $x(0)$ cu valori aleatoare, de obicei acesta reprezentând un vector de zerouri. În concluzie, $x_i(t) = 0$ pentru orice $i \in [0, n)$, unde n este numărul de necunoscute.
- Se începe iterarea și valorile soluțiilor vor fi actualizate după următoarea formulă:

  $x_i(t+1) = (b_i - \sum_{j \neq i} (A_{ij} * x_j(t)))/A_{ii}$, unde i reprezintă rândul curent al sistemului, iar j este coloana.
- Procesul iterativ se va continua până când este îndeplinit criteriul de convergență. Prin acest criteriu de convergență se poate înțelege, de exemplu, diferența dintre două iterații consecutive să fie mai mică decât un anumit prag ε.

Metoda Jacobi prezintă un avantaj semnificativ, și anume acela de a fi ușor de înțeles și implementat, dar în anumite cazuri poate avea o convergență lentă, ceea ce va duce la un număr foarte mare de iterații pentru a atinge o precizie cât mai bună.

Este de precizat faptul că o matrice diagonal dominantă (o matrice pătratică ce prezintă proprietatea că valoarea absolută a elementului de pe poziția a_{ii} aflat pe diagonala principală a matricei, este cel puțin egal sau mai mare decât suma valorilor absolute ale celorlalte elemente ale matricei aflate pe același rând; $|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \; for \; all \; i$) asigură convergența soluției și, respectiv o face mult mai rapidă. În cazul matricelor care nu sunt diagonal dominante, convergența nu este asigurată (de cele mai multe ori soluția poate diverge - adică valorile lui x vor oscila în jurul soluției, sau nu se apropie de aceasta) sau poate fi o convergență foarte lentă, de aceea se preferă utilizarea altor metode, ca spre exemplu Gauss-Seidel. Totuși, unei matrice care nu este strict diagonal dominantă i se poate asigura convergența prin reordonarea ecuațiilor, sau, respectiv prin factorizarea matricei (exemplu: factorizarea LU sau factorizarea Cholesky pentru matrice simetrice definite pozitiv).

Pentru îmbunătățirea metodei Jacobi, paralelizarea algoritmului reprezintă una din cele mai bune opțiuni.

3.3. Metoda Gauss-Seidel

Metoda Gauss-Seidel este o tehnică iterativă utilizată pentru rezolvarea sistemelor de ecuații liniare, dezvoltată de matematicianul Carl Friedrich Gauss și de matematicianul Philipp Ludwig von Seidel.

Aceasta este folosită pentru a găsi soluțiile sistemelor de ecuații liniare prin aproximări succesive. Se bazează pe principiul că valorile variabilelor necunoscute sunt actualizate în mod iterativ, folosind valorile aproximative calculate anterior. Acest proces continuă până când se atinge o precizie dorită sau până când numărul maxim de iterații este atins. Metoda este asemănătoare celei menționate anterior, iar pașii necesari rezolvării sunt:

- Se va nota cu $x(t)$ valoarea necunoscutelor, unde t este un număr întreg ce reprezintă valoarea iterației curente.

- Se începe prin a aproxima soluțiile $x(0)$ cu valori aleatoare, de obicei acesta reprezentând un vector de zerouri. În concluzie, $x_i(t) = 0$ pentru orice $i \in [0, n)$, unde n este numărul de necunoscute.
- Se începe iterarea și valorile soluțiilor vor fi actualizate după următoarea formulă:

  $x_i(t+1) = (b_i - \sum_{j<i}(A_{ij} * x_j(t+1) - \sum_{j>i}(A_{ij} * x_j(t))/A_{ii}$, unde i reprezintă rândul curent al sistemului, iar j este coloana.

Se poate observa faptul că, dacă o iterație Jacobi se execută secvențial, în metoda Gauss-Seidel, la momentul evaluării $x_i(t+1)$ există deja câteva valori estimate noi $x_j(t),\ j < i$.

Metoda Gauss-Seidel poate converge către soluția corectă a sistemului de ecuații liniare, dar acest lucru nu este garantat în toate cazurile. Convergența depinde de proprietățile matematice ale sistemului de ecuații, cum ar fi coeficienții acestuia sau structura matricei asociate.

Aceasta metodă, în general, converge mai rapid către soluția sistemului decât metoda Jacobi. Deoarece metoda Gauss-Seidel actualizează variabilele necunoscute pe măsură ce avansează în iterații, aceasta poate beneficia de informațiile actualizate pentru a obține o convergență mai rapidă.

4. Implementare

4.1. Metoda Gauss prin Eliminare

Metoda Gauss prin eliminare este implementată folosind formula din curs, cea în care la fiecare iterație se elimină câte un rând din matrice bazată pe versiunea de grid. Pentru a îndeplini obiectivul propus, algoritmul a fost modificat. Un proces va fi declarat ca *MASTER*, acesta fiind procesul cu rank-ul 0. Acesta citește matricea din fișier și stabilește, în funcție de câte procesoare sunt folosite, ce elemente trebuie să calculeze fiecare. Se face *scatter* la elemente, astfel fiecare procesor va cunoaște ce elemente din matrice trebuie să calculeze. Se face *broadcast* la matrice. Începe iterația unde fiecare proces calculează în funcție de indecșii elementelor asociați din formula precedentă. La final de iterație toate procesele se vor sincroniza, astfel formând matricea C finală.

După ce se termină numărul de iterații, procesul *MASTER* va face back substitution pentru a obține soluția. Acest pas nu este paralelizat din cauză că este secvențial din natură, deci overhead-ul comunicației ar fi creat delay-uri mai mari. Topologia ce funcționează cel mai bine pentru o astfel de implementare este un *Fully Connected Mesh* întrucât sunt prezente broadcast-uri frecvente.

Algoritmul este blocat de faptul că la fiecare iterație trebuie o resincronizare. Nu există sarcini suficiente pentru a paraleliza total soluția pentru a lăsa procesoarele să lucreze independent.

4.2. Metoda Jacobi

Drept date de intrare, programul paralel va primi matricea A și vectorul termenilor liberi b (sau, matematic, dacă avem sistemul Ax=b , programul va primi extinderea matricea A, adică Ā). Procesul *MASTER* va fi cel care realizează citirea datelor, respectiv împărțirea lor și distribuirea lor către celelalte procese. De asemenea, procesul *MASTER* va determina, de asemenea, dacă matricea este sau nu diagonal dominantă.

Fiecare proces va avea o parte din matricea A și din vectorul termenilor liberi. În funcție de dimensiunea matricei și numărul de procese solicitat împărțirea se poate face diferit. Pentru

exemplificare, se dă o matrice de dimensiune 4x4. Pentru aceasta se întâlnesc patru cazuri posibile:

- Un singur proces - în acest caz, programul este serial, procesul MASTER va primi toată matricea A și se va realiza algoritmul descris mai sus.
- Două procese - în acest caz, programul va prezenta două procese, iar fiecare dintre acestea va primi câte două linii din matricea A, respectiv două valori ai termenilor liberi.
- Trei procese - în cazul acesta, împărțirea programului va fi diferită, primele două procese primind doar o linie a matricea A (și câte un termen liber), iar ultimul proces va avea de prelucrat ultimele două linii ale matricei împreună cu ultimii doi termeni liberi.
- Patru procese - fiecare proces va primi câte o linie a matricei și un singur element liber.

Urmează calculul iterativ Jacobi propriu-zis, în care, pentru fiecare iterație, procesele vor actualiza pentru fiecare linie a matricei și element liber primit, valorile corespunzătoare soluției. După acest calcul, fiecare rezultat este colectat de la fiecare proces și trimis către toate procesele în vederea folosirii lor la o nouă iterație. Astfel, toate procesele primesc valori actualizate ale vectorului de soluții. Acest lucru este realizat din cauza necesității valorilor lui x(t) pentru iterația următoare (t+1). La final, procesul *MASTER* va fi cel care afișează soluția finală cu x-ul final rezultat după realizarea tuturor iterațiilor.

4.3. Metoda Gauss-Seidel

Procesul *MASTER* execută operația de citire a datelor de intrare. Tot acesta va calcula, pe baza inputului primit, variabila *SPLIT_FACTOR*. Această variabilă are menirea de a defini numărul de linii care vor fi atribuite fiecărui proces implicat în algoritmul paralel. Pe baza acestui factor, procesul de rank 0 va trimite liniile aferente proceselor de tip slave. De menționat este și faptul că, însuși procesul *MASTER* are funcționalități specifice proceselor de tip slave. Distribuția datelor între procese a fost descrisă în [3]. Autorul propunea o împărțire a liniilor între procese într-un mod bine definit, care să ducă și la o oarecare randomizare a acestora. Însă, în soluția prezentată, în cazul în care procesele stochează mai mult de o linie, acestea vor fi linii consecutive, tocmai pentru a elimina anumite comunicații care ar fi trebuit să se întâmple între linia i din procesul j și linia $i + 1$ din procesul k.

Fiecare proces are acces la două variabile în care se stochează soluția curentă și soluția precedentă.

Procesul iterativ de calcul estimativ al soluției, se realizează astfel:

- Fiecare proces *i* (diferit de *MASTER*) primește de la procesul *i-1* cele două variabile cu soluțiile, aflate în diferite etape de calcul. Procesul *MASTER* va primi aceste date de la procesul cu rank-ul cel mai mare.
- Cu datele primite, se realizează calculele necesare estimării valorilor x aferente procesului curent.
- Noua componentă cu valoare estimată se adaugă la variabila soluției curente.
- Cele două variabile cu soluțiile sunt trimise mai departe procesului *i+1*. Acest lucru se întâmplă doar în cazul în care procesul curent nu este, de fapt, chiar ultimul proces folosit în algoritmul paralel. În cazul în s-a ajuns la ultimul proces, după calculele estimărilor, se v-a suprascrie vechea soluție estimată cu soluția curentă, iar variabila soluției curente se va inițializa, din nou, cu o listă vidă. Aceste date vor fi trimise procesului *MASTER*.
- La final, procesul *MASTER* va afișa soluția găsită.

Se poate observa faptul că soluția descrisă anterior respectă o structură specifică unei topologii liniare.

5. Experimente

Mașina de test pentru programe este un *MacBook Pro cu Apple M1 Pro CPU* cu **10 nuclee**, din care **8 nuclee de performanță** și **2 nuclee de eficiență**, deci, în total, 10 procesoare fizice pe care se poate rula în mod implicit. Acest număr poate fi extins folosind un fișier de hosts file în care se precizează un număr care să depășească această valoare, iar sistemul de operare sau implementarea de MPI va decide modul în care dă schedule la procese.

De asemenea, se menționează următoarele:

- Versiunea de sistem de operare 13.1 (22C65).
- Versiunea de MPI folosită este mpirun (Open MPI) 4.1.5 pentru MacOS distribuită prin brew.
- Versiunea de Python 3.11.2.

Au fost realizate două tipuri de experimente - compararea metodelor în funcție de durata de procesare a soluțiilor, variind numărul de procesoare și compararea metodelor în funcție de durata de procesarea a soluțiilor, variind numărul ecuații ce alcătuiesc sistemul. De asemenea, am calculat accelerația pentru fiecare algoritm.

Pentru primul experiment, au fost executate câte trei teste pentru fiecare măsurătoare, astfel încât să se obțină o medie a timpului în care se generează soluțiile. Au fost efectuate teste pe un sistem cu 100 de ecuații și 500 de iterații. Numărul de procesoare a fost variabil: 1, 3, 5, 7 și 10.

Au fost obținute următoarele rezultate:

Număr procesoare	Măsurătoare	Gauss Elimination	Jacobi	Gauss-Seidel
1	1	9415	9995	4076
	2	9374	10095	4044
	3	9531	9983	4114
3	1	4070	3438	8387
	2	4153	3384	8378
	3	4175	3433	8613
5	1	3231	2247	10426
	2	3298	2129	10318
	3	3263	2063	9966
7	1	4383	3008	16138
	2	4436	2949	15268
	3	4220	2916	15049
10	1	9968	14537	27100
	2	9233	14493	27449
	3	9743	14346	28857

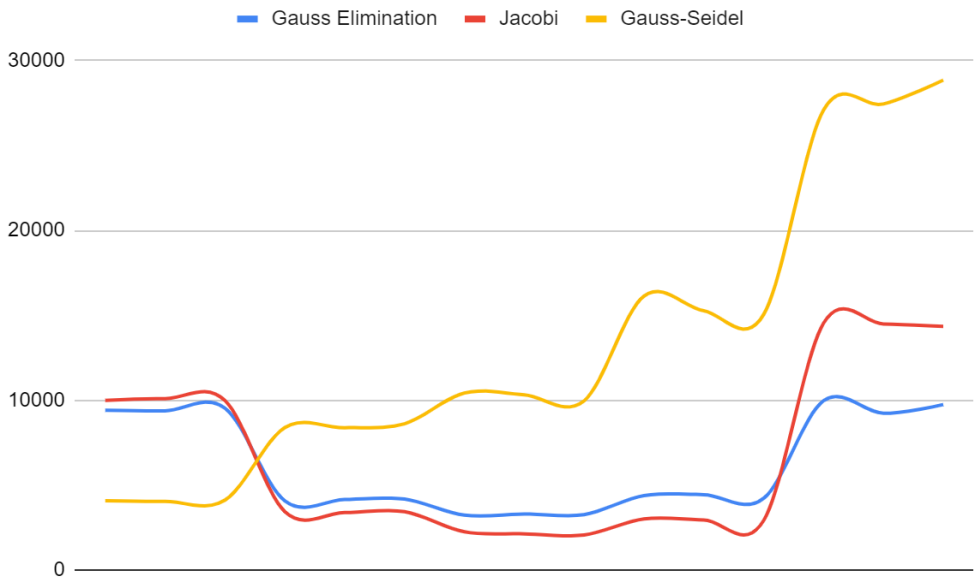

Figura 5.1. Evoluția performanței la fiecare metodă (toate măsurătorile).

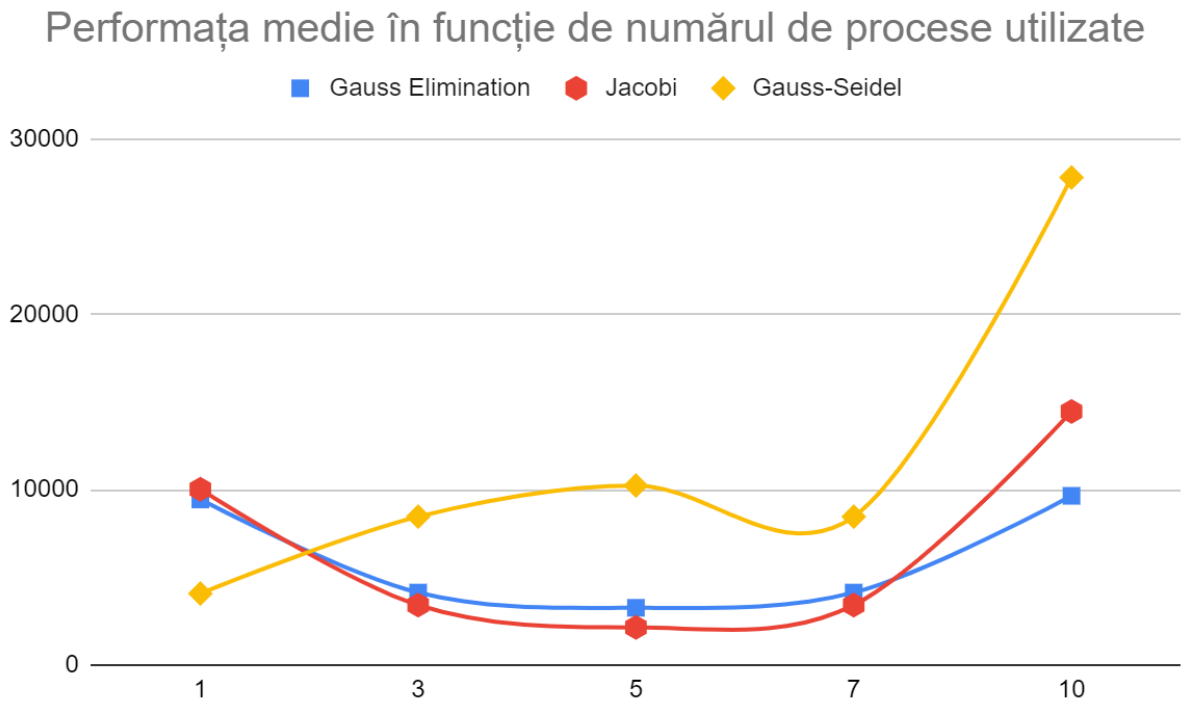

Figura 5.2. Performanța medie pentru fiecare tip de metodă, în funcție de numărul de procese utilizate.

În urma acestui experiment, se poate observa fenomenul de bottleneck în dreptul fiecărei metode, în ultimele măsurători (7-10 procese) fiind creșteri în timp al procesului de rezolvare. Acest fapt poate fi cauzat de arhitectura hardware folosită, unde doar opt din cele zece procesoare fiind pentru performanță. De asemenea, se remarcă o performanță mai scăzută a metodei Gauss-Seidel, acest lucru datorându-se întârzierilor de comunicare între procese.

Pentru al doilea experiment, au fost folosite următoarele date: numărul de ecuații variabil (25, 50, 75, 100, 125) și o configurație care a performat bine pe toate metodele (trei procesoare și 500 de iterații). S-au obținut următoarele:

Număr ecuații	Gauss Elimination	Jacobi	Gauss-Seidel
10	3	34	82
25	31	250	524
50	263	662	1416
75	891	1250	3097
100	2147	2141	5436
125	4151	3379	8527

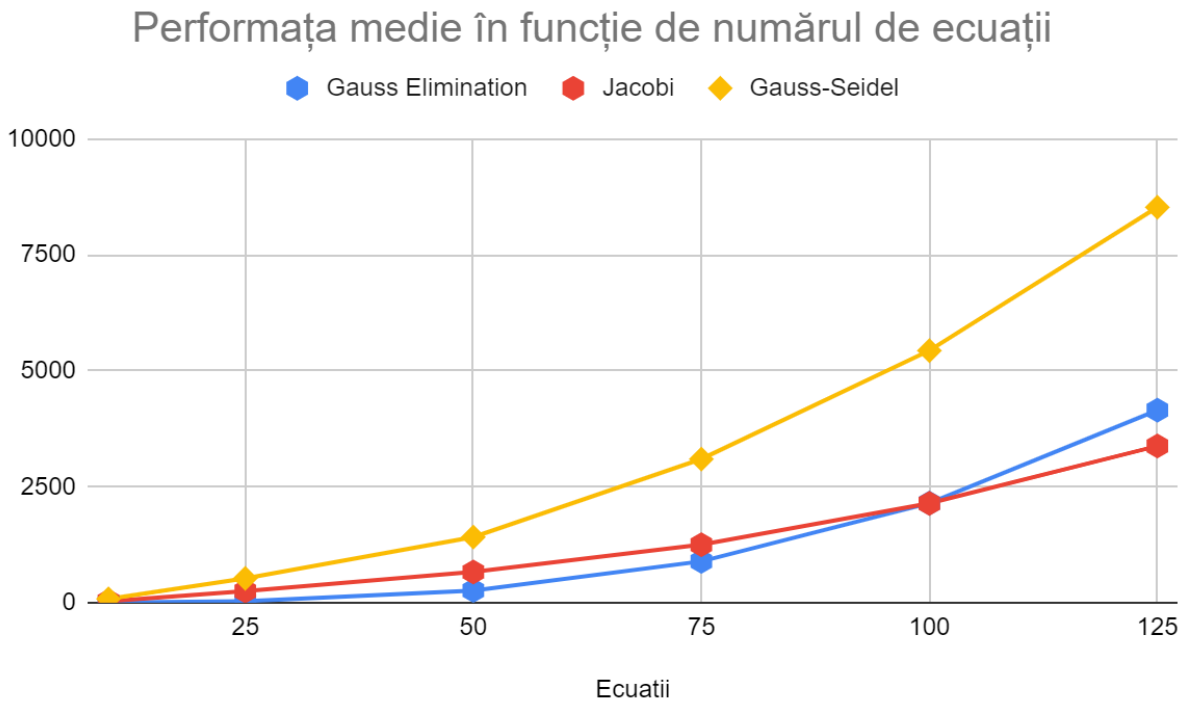

Figura 5.3. Performanța medie pentru fiecare tip de metodă, în funcție de numărul de ecuații din sistem.

În urma acestui experiment, s-a putut constata faptul că durata de execuție a algoritmilor e direct proporțională cu numărul de ecuații care compun sistemul.

Pentru calcularea accelerației, s-a măsurat în timp eficiența algoritmilor secvențiali și s-a raportat la valorile măsurate în experimentul numărul doi. De menționat este faptul că, pentru metoda Gauss-Seidel, măsurătorile algoritmului secvențial au depășit cu mult timpii medii pentru ceilalți doi algoritmi (peste două minute), în cazul în care se primea un sistem cu 50 de ecuații.

Număr ecuații	Gauss Elimination (S)	Gauss Elimination (P)	Jacobi (S)	Jacobi (P)	Gauss-Seidel (S)	Gauss-Seidel (P)
10	4	3	66	34	60	82
25	72	31	410	250	341	524
50	579	263	1634	662	**120000>**	1416
75	2048	891	3913	1250	-	3097
100	4851	2147	6473	2141	-	5436
125	9396	4151	10159	3379	-	8527

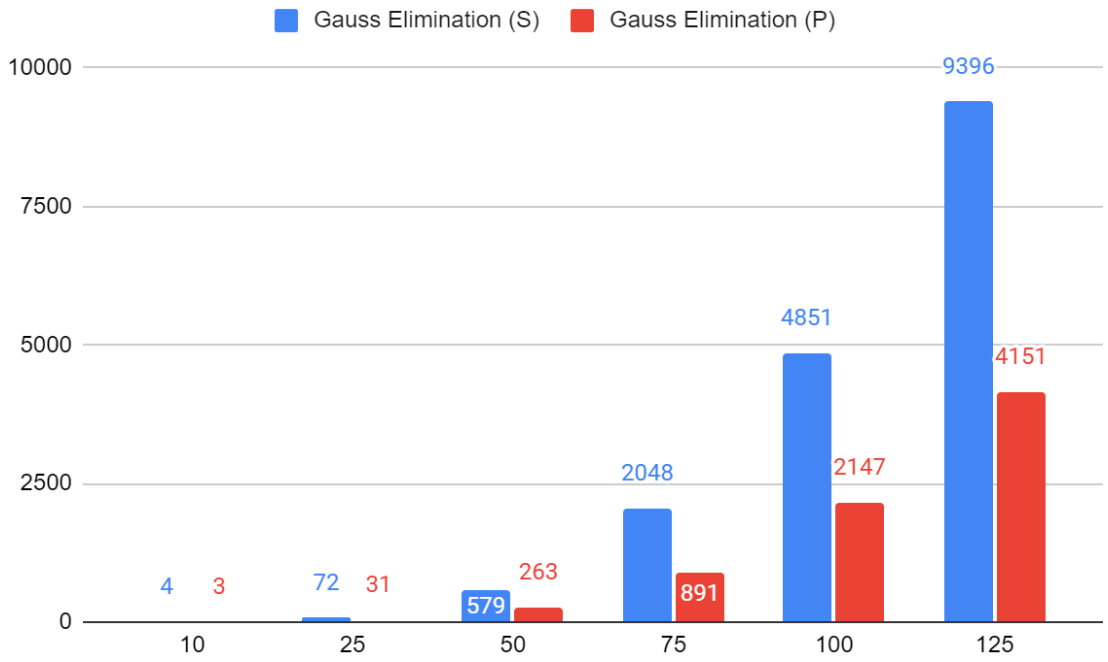

Figura 5.4. Compararea timpilor de execuție pentru metoda eliminării Gauss secvențială (S) și paralel (P).

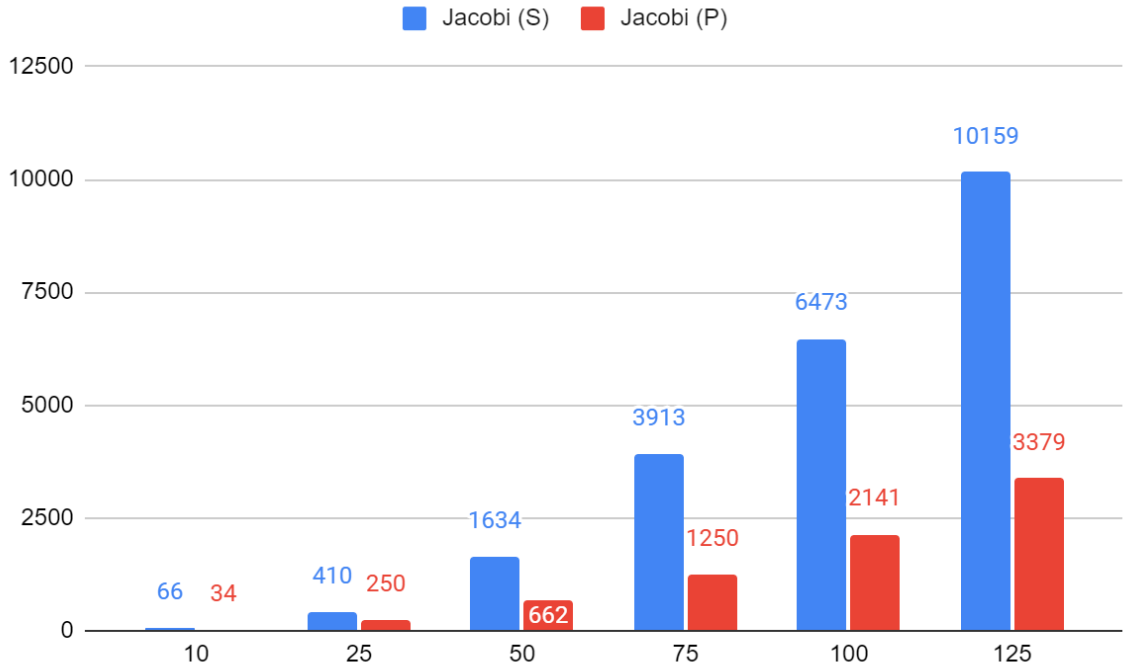

Figura 5.5. Compararea timpilor de execuție pentru metoda eliminării Jacobi secvențială (S) și paralel (P).

Pe baza informațiilor colectate, se obțin următoarele valori pentru speed-up:

Număr ecuații	Gauss Elimination	Jacobi	Gauss-Seidel
10	1.333333333	1.941176471	0.7317073171
25	2.322580645	1.64	0.6507633588
50	2.201520913	2.468277946	-
75	2.298540965	3.1304	-
100	2.259431765	3.023353573	-
125	2.263550952	3.006510802	-

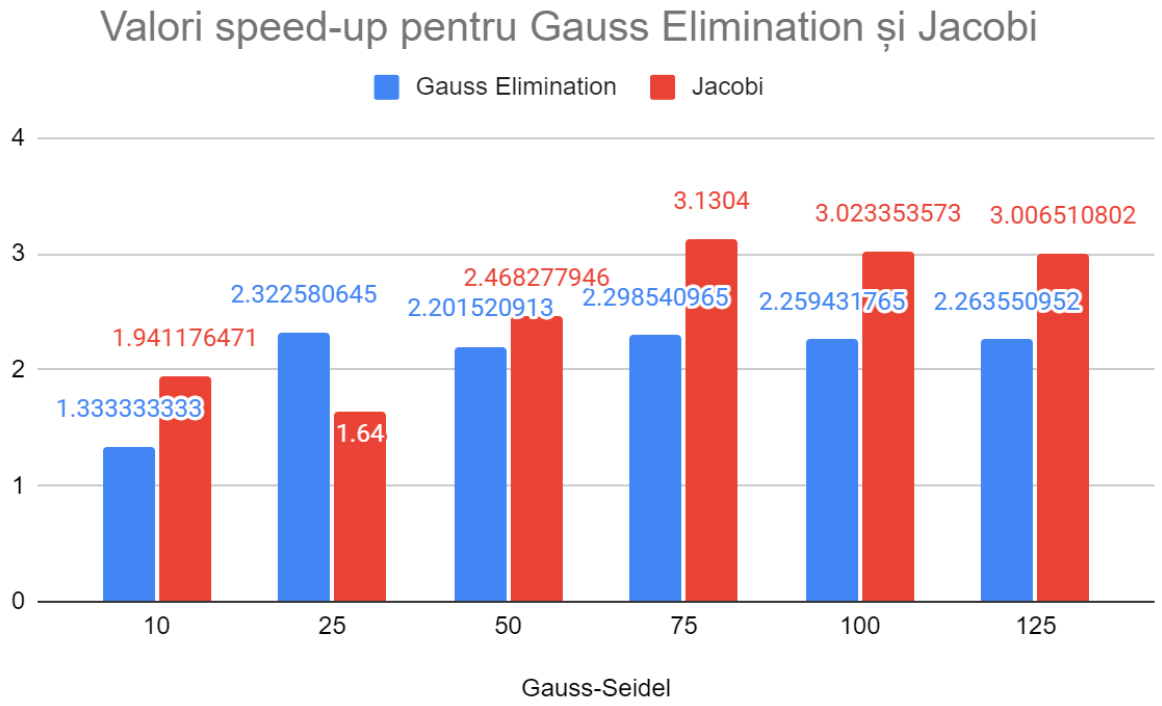

Figura 5.6. Valorile de speed-up aferente fiecărei metode pentru care s-a putut efectua acest calcul.

6. Concluzii

Problema rezolvării unui sistem de ecuații liniar este una inerent secvențială ce implică costuri de comunicare ridicate. Întrucât paralelizarea calculelor duce la ridicarea timpilor de execuție, se pune problema randamentului și a eficienței, dar și a nevoii unui astfel de algoritm.

Am observat că, în multe situații, creșterea numărului de procese nu a dus la o îmbunătățire a rezultatelor din cauza necesității de comunicare frecventă. Astfel, problema rezolvării unui sistem de ecuații în mod paralel se poate pune în alte contexte, ci nu cea a calculului, precum rezolvarea unor sisteme de ecuații ce depășesc memoria asignată unui singur procesor, însă pentru a rezolva un astfel de caz este necesară definirea unor topologii formate dintr-un număr de procese exacte pentru a reliefa o structură concretă.

Dificultățile care au apărut constau în probleme de calcul cu numere în virgulă mobilă ce devin dificile din cauza preciziei necesare ce duc la pierderea de informație și obținerea de rezultate incorecte, deși cazul este posibil.

Observăm că există un punct la care creșterea numărului de procesoare aduce chiar o reducere a performanțelor, ceea ce face alegerea de topologii și mai importantă. Acest lucru se poate face după mai multe experimentări. Pot fi aduse îmbunătățiri la modul de comunicare prin crearea unor topologii specifice, dar și prin adoptarea unor topologii specializate pe această problemă.

Un alt obstacol reprezintă obținerea setului de date pentru testare, întrucât paralelizarea devine o opțiune atunci când avem volum ridicat de date. În absența acesteia, nu se poate concluziona impactul sarcinilor distribuite. Paralelizarea unei sarcini necesită cunoștințe în domeniul problemei pentru a găsi soluția eficientă, dar și pentru a avea metode alternative.

7. Bibliografie

[1] K.N.Balasubramanya Murthy, and C.Siva Ram Murthy 1995. *A new gaussian elimination-based algorithm for parallel solution of linear equations*. Computers & Mathematics with Applications, 29(7), p.39-54.

[2] F.Thomson Leighton 1992. *Introduction to Parallel Algorithms and Architectures*. Arrays, Trees, Hypercubes. Elsevier Inc.

[3] Yueqiang Shang 2009. *A distributed memory parallel Gauss–Seidel algorithm for linear algebraic systems.* Computers & Mathematics with Applications, 57(8), p.1369-1376.