Δημοκρίτειο Πανεπιστήμιο Θράκης

Τμήμα: Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εργαστήριο: Εργαστήριο Αρχιτεκτονικής Υπολογιστών και Συστημάτων Υψηλών Επιδόσεων

Μάθημα: Τεχνολογία Παράλληλης Επεξεργασίας

Ακαδημαϊκό έτος: 2023

Θέμα: Επίλυση συστήματος γραμμικών εξισώσεων με τη μέθοδο απαλοιφής Gauss

Ομάδα 1

Αντωναράκος Δημήτριος

Μωραΐτη Παναγιώτα

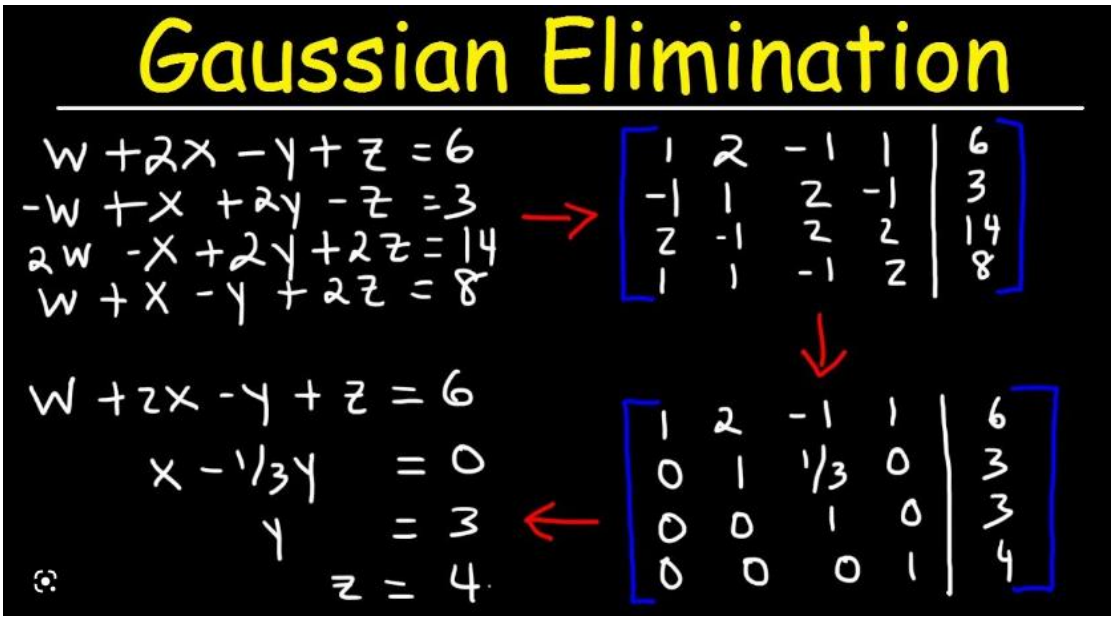
Σούλη Ευθυμία

<u>Θεωρητική Παρουσίαση Αλγορίθμου:</u>

Σκοπός της μεθόδου απαλοιφής Gauss είναι να μετατρέψει ένα σύστημα γραμμικών εξισώσεων σε έναν άνω τριγωνικό πίνακα από τον οποίο θα προκύψουν οι λύσεις με τη χρήση της μεθόδου αντικατάστασης προς τα πίσω (**Backward Substitution Method**).

Ο αλγόριθμος που ακολουθήσαμε για την υλοποίηση της μεθόδου έχει τα εξής στάδια:

1. **Μετατροπή του επαυξημένου πίνακα σε άνω τριγωνικό**: Προσθέτει κάθε γραμμή στις επόμενες πολλαπλασιασμένη κάθε φορά κατά μια ποσότητα τέτοια ώστε το στοιχείο της γραμμής που ανήκει στη διαγώνιο να μηδενίζει τα επόμενα στοιχεία της στήλης του. Αν το στοιχείο αυτό είναι 0, τότε τερματίζει τον αλγόριθμο.
2. **Εντοπισμός λύσεων με τη μέθοδο αντικατάστασης προς τα πίσω:** Αρχικά βρίσκει τη λύση της τελευταίας εξίσωσης διαιρώντας το τελευταίο στοιχείο της τελευταίας γραμμής με το προτελευταίο. Στη συνέχεια, για κάθε άγνωστο που απομένει, πολλαπλασιάζει τις λύσεις που έχουν βρεθεί με τον συντελεστή τους στη γραμμή του αγνώστου και τις αφαιρεί από το τελευταίο στοιχείο της γραμμής. Τέλος διαιρεί το τελευταίο στοιχείο με τον συντελεστή του αγνώστου και βρίσκει τη λύση.



Ο κώδικας:

Σειριακός

Στον κώδικα αρχικά ορίζουμε τις βιβλιοθήκες και τις μεταβλητές που θα χρησιμοποιήσουμε. Η μεταβλητή n είναι ο αριθμός των αγνώστων, οι i, j και k θα χρησιμοποιηθούν ως μετρητές και η ratio θα εξηγηθεί παρακάτω. Ο πίνακας a θα είναι ο επαυξημένος πίνακας me n+1Xn στοιχεία, στους xs και x θα αποθηκευτούν οι λύσεις του σειριακού και του παραλλήλου αντίστοιχα και ο A_copy θα χρησιμοποιηθεί στον παράλληλο κώδικα. Στη συνέχεια θέτουμε τον αριθμό των αγνώστων, δεσμεύουμε την μνήμη για τους πίνακες a, x, xs και A_copy με τη χρήση της malloc και ορίζουμε τις τιμές των στοιχείων των a και A_copy.

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<omp.h>

#define EPSILON 0.000001

int main(int argc, char *argv[])
{
    float ratio;
    int i, j, k, n;
    double x_i;

     // Dimensions of the matrix
    printf("Enter number of unknowns: ");
    scanf("%d", &n);

    double *x, *xs;
    float **a, **A_copy;
    x = malloc((n+1)*sizeof(double));
    xs = malloc((n+1)*sizeof(double));
    a = malloc((n+1)*sizeof(double*));
    A_copy = malloc((n+1)*sizeof(double*));

    for (i=0; i<n+1; i++)
    {
        a[i] = malloc(n*sizeof(double));
        A_copy[i] = malloc(n*sizeof(double));
    }

     // Put the elements of the Augmented Matrix
    for (i = 0; i < n; i++)
    {
        for (j = 0; j <= n; j++)
        {
            a[i][j] = rand();
            A_copy[i][j] = a[i][j];
        }
    }
```

Στο σημείο αυτό ξεκινά ο σειριακός κώδικας. Αρχικά ορίζουμε τις μεταβλητές start_time, end_time και elapsed_time_s που θα χρησιμοποιηθούν για την σύγκριση του χρόνου υλοποίησης του σειριακού αλγορίθμου με τον παράλληλο.

```
// Gauss Elimination serial
double start_time, end_time, elapsed_time_s;

start_time = omp_get_wtime();
for (i = 0; i < n-1; i++)
{
    if (a[i][i] == 0.0)
    {
        printf("Error, Division by zero!");
        exit(0);
    }

    for (j = i+1; j < n; j++)
    {
        ratio = a[j][i] / a[i][i];

        for (k = 0; k <= n; k++)
        {
            a[j][k] = a[j][k] - ratio * a[i][k];
        }
    }
}
```

Χρησιμοποιούμε τρεις εμφωλιασμένες for για την υλοποίηση του πρώτου βήματος. Η πρώτη for ξεκινά από την πρώτη γραμμή και τελειώνει στην προτελευταία ώστε να μηδενιστούν όλα τα στοιχεία κάτω από την διαγώνιο (του αρχικού πίνακα). Πριν χρησιμοποιηθούν στις επόμενες for, κάθε στοιχείο της διαγώνιου ελέγχεται για το αν είναι ίσο με 0.

Η επόμενη for ξεκινά από τη δεύτερη γραμμή και τελειώνει στην τελευταία. Σε αυτήν υπολογίζεται η μεταβλητή ratio η οποία είναι σε κάθε επανάληψη:

ratio = τιμή επόμενων στοιχείων στήλης/τιμή στοιχείου διαγώνιου

Τέλος, η τελευταία for χρησιμοποιείται για να αφαιρέσει τα στοιχεία της γραμμής που δείχνει ο δείκτης i πολλαπλασιασμένα με το ratio από τα στοιχεία της γραμμής που δείχνει ο δείκτης j.

Στη συνέχεια βρίσκουμε την λύση της τελευταίας εξίσωσης του συστήματος και την αποθηκεύουμε στην τελευταία θέση του πίνακα x.

```
xs[n-1] = a[n-1][n] / a[n-1][n-1];

for (i = n-2; i >= 0; i--)
{
    xs[i] = a[i][n];
    for (j = i+1; j < n; j++)
    {
        xs[i] = xs[i] - a[i][j] * xs[j];
    }
    xs[i] = xs[i] / a[i][i];
}
```

Για να βρούμε τις υπόλοιπες λύσεις χρησιμοποιούμε δύο εμφωλιασμένες for.Η πρώτη ξεκινά από την προτελευταία γραμμή και τελειώνει στην πρώτη. Αρχικά ορίζουμε ότι το στοιχείο i του x είναι το τελευταίο στοιχείο και στη συνέχεια, στη δεύτερη for, αφαιρούμε από αυτό τις λύσεις που έχουμε βρει μέχρι στιγμής. Τέλος το διαιρούμε τον συντελεστή του ώστε να βρούμε την τελική λύση.

Στο τέλος του σειριακού κώδικα βρίσκουμε τον χρόνο υλοποίησης του προγράμματος και τον τυπώνουμε μαζί με τις λύσεις.

```
end_time = omp_get_wtime();

// Print the solution
/*printf("\nSolution:\n");
for (i = 0; i < n; i++)
{
    printf("xs[%d] = %f\n", i, xs[i]);
}*/

elapsed_time_s = end_time - start_time;
printf("\nElapsed time Serial: %fs\n\n", elapsed_time_s);
```

Παράλληλος

Ο παράλληλος κώδικας εκτελείται για πυρήνες P = 1, 2, 4, 8, 32, 64. Ο κώδικας ξεκινά με τον ορισμό των μεταβλητών P, speedup, elapsed time και efficiency. Επίσης αντιγράφει τα στοιχεία του A_copy πίσω στον πίνακα a και εμφανίζει το αριθμό των πυρήνων στους οποίους τρέχει το πρόγραμμα στη συγκεκριμένη λούπα.

```
// Gauss Elimination parallel
int P;
double speedup, elapsed_time, efficiency;
for (P = 1; P <= 64; P = P*2)
{
    // Restore the initial values in the a matrix
    for(i=0; i<n; i++)
    {
        for(j=0; j<=n; j++)
        {
            a[i][j] = A_copy[i][j];
        }
    }
    printf("P = %d\n", P);
```

Από αυτό το σημείο και μετά ο παράλληλος κώδικας είναι σχεδόν πανομοιότυπος με τον σειριακό. Η διαφορά βρίσκεται στο ότι παραλληλοποιούμε την εμφωλιασμένη for που υπολογίζει το ratio και την for που εκτελεί την αντικατάσταση προς τα πίσω. Η παραλληλοποίηση της 1ης for για την κατασκευή του άνω τριγωνικού πίνακα δεν μπορεί να υλοποιηθεί λόγω εξαρτήσεων του αλγορίθμου. Συγκεκριμένα, η σειρά με την οποία επεξεργαζόμαστε τις γραμμές είναι συγκεκριμένη και δεν μπορεί να γίνει τυχαία.

```
start_time = omp_get_wtime();

for(i=0; i<n-1; i++)
{
    if(a[i][i] == 0.0)
    {
        printf("Error, Division by zero!");
        exit(0);
    }
  # pragma omp parallel for private(j,k,ratio) num_threads(P)
    for(j=i+1; j<n; j++)
    {
        ratio = (a[j][i]/a[i][i]);
        for(k=0; k<n+1; k++)
        {
            a[j][k] = a[j][k] - ratio*a[i][k];
        }
    }
}
```

Για την αντικατάσταση προς τα πίσω μπορούμε να παραλληλοποιήσουμε είτε την εξωτερική είτε την εσωτερική for. Εκτελέσαμε το πρόγραμμα και με τους δύο τρόπους ώστε να διαπιστώσουμε ποιος είναι καλύτερος.

Εσωτερική for:

```
x[n-1] = a[n-1][n]/a[n-1][n-1];
//# pragma omp parallel for private(i, x_i, j) num_threads(P)
for(i=n-2; i>=0; i--)
{
    x_i = a[i][n];
# pragma omp parallel for private(j) reduction(+: x_i) num_threads(P)
    for(j=i+1; j<n; j++)
    {
        x_i += -a[i][j]*x[j];
    }
    x[i] = x_i/a[i][i];
}
```

Εξωτερική for:

```
        x[n-1] = a[n-1][n]/a[n-1][n-1];
        # pragma omp parallel for private(i, x_i, j) num_threads(P)
        for(i=n-2; i>=0; i--)
        {
            x_i = a[i][n];
            //# pragma omp parallel for private(j) reduction(+: x_i)
num_threads(P)
            for(j=i+1; j<n; j++)
            {
                x_i += -a[i][j]*x[j];
            }
            x[i] = x_i/a[i][i];
        }
```

Στην συνέχεια εκτυπώνουμε τα αποτελέσματα και υπολογίζουμε τα elapsed_time, speedup και efficiency.

```
        end_time = omp_get_wtime();

        // Print the solution
        /*printf("\nSolution:\n");
        for(i=0; i<n; i++)
        {
            printf("x[%d] = %f\n",i, x[i]);
        }*/

        elapsed_time = end_time - start_time;
        printf("Elapsed time Parallel: %fs\n", elapsed_time);
        speedup = elapsed_time_s/elapsed_time;
        printf("Speedup: %f\n", speedup);
        efficiency = speedup/P;
        printf("Efficiency: %f\n", efficiency);
```

Το πρόγραμμα αυτό χρησιμοποιείται για να λύσουμε μεγάλα γραμμικά συστήματα (μεγέθους 1000X1000). Για αυτό το λόγο ο έλεγχος των λύσεων μέσω εκτύπωσης καθίσταται δύσκολος. Οπότε για μεγάλα συστήματα, αντί να εκτυπώνουμε τις λύσεις, συγκρίνουμε απλά τις λύσεις που υπολογίστηκαν με τον σειριακό κώδικα με αυτές του παραλλήλου. Αν η διαφορά κάποιων λύσεων είναι μεγαλύτερη του EPSILON = 0.000001 τότε η μεταβλητή flag γίνεται 1 και εκτυπώνεται το μήνυμα Wrong Answer.

```
        // Check the results
        int flag = 0;
        for (i=0; i<n; i++)
        {
            if (fabs(x[i] - xs[i]) <= EPSILON)
            {
                flag = 0;
            }
            else
            {
                flag = 1;
                break;
            }
        }

        if (flag == 0)
        {
            printf("Correct answer\n\n");
        }
        else
        {
            printf("Wrong answer\n\n");
        }
    }
```

Τέλος απελευθερώνομε την μνήμη για τους πίνακες a, A_copy, xs και x.

```
    // Free memory
    for(i=0; i<n+1; i++)
    {
        free(a[i]);
        free(A_copy[i]);
    }
    free(x);
    free(xs);
    return(0);
}
```

Αποτελέσματα:

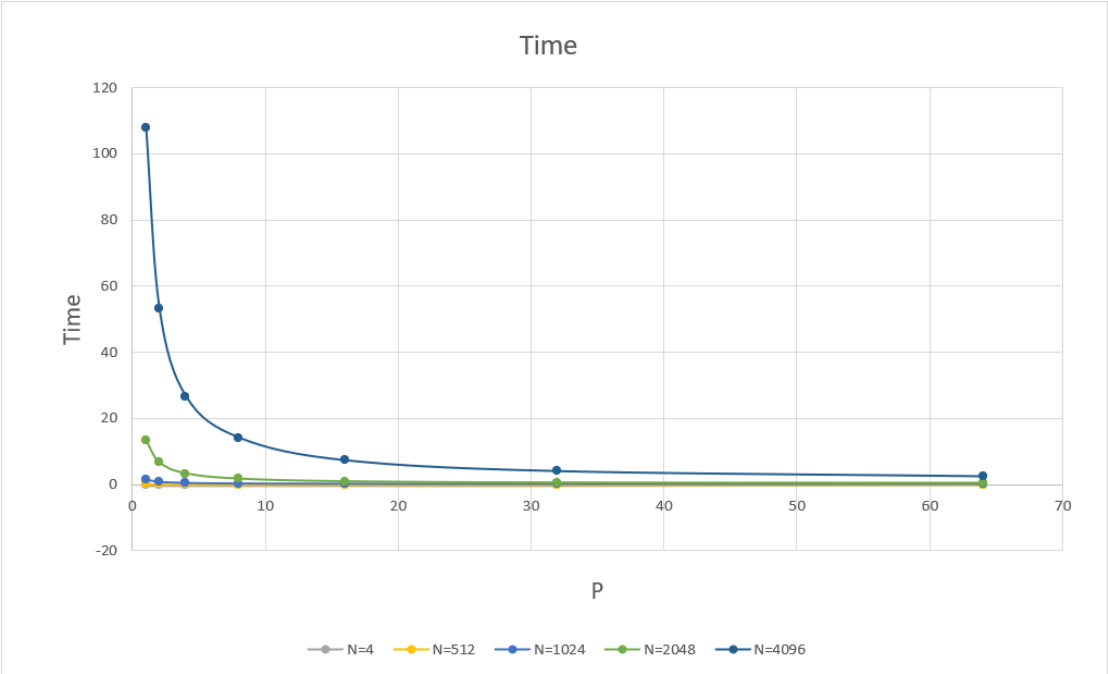
Παραπάνω φαίνονται τα αποτελέσματα του κώδικα και ενός calculator για ένα σύστημα 4X4. Από την εικόνα μπορούμε να παρατηρήσουμε πως το πρόγραμμα λύνει το σύστημα με τον σωστό τρόπο.

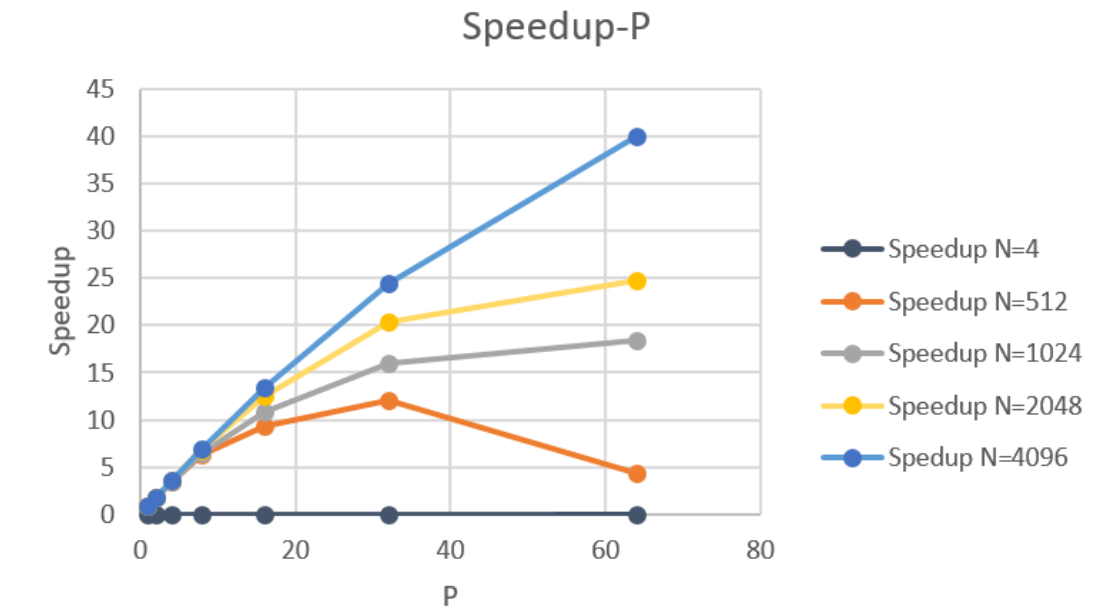
Για Εσωτερική for:

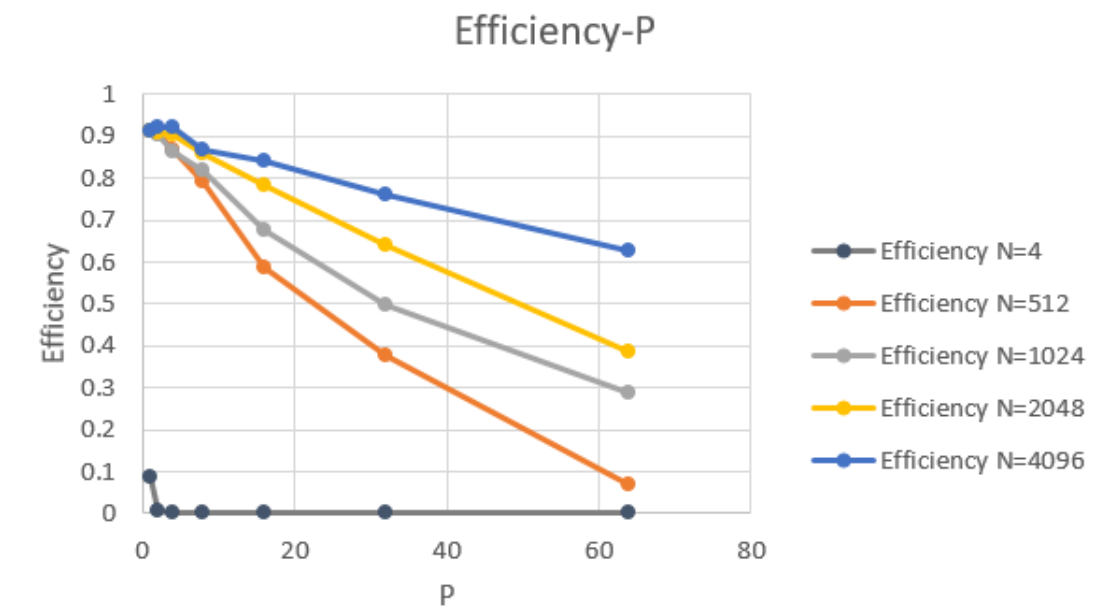
N	Metric	Serial	P=1	P=2	P=4	P=8	P=16	P=32	P=64
4	Time	0.000002s	0.000033s	0.000165s	0.000199s	0.000306s	0.000410s	0.000763s	0.048727s
	Speedup		0.059634	0.011985	0.009945	0.006470	0.004829	0.002599	0.000041
	Efficiency		0.059634	0.005993	0.002486	0.000809	0.000302	0.000081	0.000001
512	Time	0.192727s	0.210334s	0.107001s	0.057118s	0.035211s	0.026224s	0.026695s	0.059470s
	Speedup		0.916289	1.801164	3.374173	5.473517	7.349373	7.219506	3.240732
	Efficiency		0.916289	0.900582	0.843543	0.684190	0.459336	0.225610	0.050636
1024	Time	1.533818s	1.696357s	0.845984s	0.447474s	0.240157s	0.154219s	0.122867s	0.256460s
	Speedup		0.904184	1.813059	3.427723	6.386728	9.945695	12.483520	5.980726
	Efficiency		0.904184	0.906529	0.856931	0.798341	0.621606	0.390110	0.093449
2048	Time	12.130293s	13.393849s	6.674169s	3.383771s	1.765271s	0.944782s	0.625972s	0.481407s
	Speedup		0.905661	1.817499	3.584844	6.871630	12.839251	19.378331	25.197576
	Efficiency		0.905661	0.908749	0.896211	0.858954	0.802453	0.605573	0.393712
4096	Time	98.113870s	108.373497s	53.897047s	26.743582s	13.759371s	7.273141s	4.094508s	2.851055s
	Speedup		0.905331	1.820394	3.668688	7.130694	13.489890	23.962310	34.413183
	Efficiency		0.905331	0.910197	0.917172	0.891337	0.843118	0.748822	0.537706

Για Εξωτερική for:

N	Metric	Serial	P=1	P=2	P=4	P=8	P=16	P=32	P=64
4	Time	0.000002s	0.000021s	0.000183s	0.000187s	0.000260s	0.000382s	0.000686s	0.040701s
	Speedup		0.085328	0.009745	0.009529	0.006867	0.004678	0.002603	0.000044
	Efficiency		0.085328	0.004873	0.002382	0.000858	0.000292	0.000081	0.000001
512	Time	0.192957s	0.211541s	0.106542s	0.055737s	0.030474s	0.020574s	0.015948s	0.043697s
	Speedup		0.912150	1.811097	3.461914	6.331935	9.378882	12.099404	4.415777
	Efficiency		0.912150	0.905548	0.865478	0.791492	0.586180	0.378106	0.068997
1024	Time	1.523144s	1.673794s	0.841685s	0.441062s	0.232800s	0.140868s	0.095510s	0.082424s
	Speedup		0.909995	1.809637	3.453355	6.542711	10.812541	15.947471	18.479432
	Efficiency		0.909995	0.904818	0.863339	0.817839	0.675784	0.498358	0.288741
2048	Time	12.104099s	13.281563s	6.656703s	3.342983s	1.764660s	0.964965s	0.592855s	0.487868s
	Speedup		0.911346	1.818333	3.620748	6.859166	12.543563	20.416624	24.810183
	Efficiency		0.911346	0.909166	0.905187	0.857396	0.783973	0.638019	0.387659
4096	Time	98.182695s	107.715889s	53.335160s	26.652197s	14.151880s	7.307675s	4.023906s	2.454131s
	Speedup		0.911497	1.840862	3.683850	6.937785	13.435558	24.399849	40.007116
	Efficiency		0.911497	0.920431	0.920963	0.867223	0.839722	0.762495	0.625111







Όπως είναι λογικό παρατηρούμε πως ο σειριακός κώδικας είναι καλύτερος για μικρό αριθμών αγνώστων. Ο παράλληλος δαπανά πολύ χρόνο στην επικοινωνία μεταξύ των πυρήνων και της μνήμης καθώς και στην ίδια την υλοποίηση της παραλληλοποίησης. Ωστόσο, όσο το n αυξάνεται, ο σειριακός χρόνος αυξάνει εκθετικά και ο παράλληλος γίνεται όλο και πιο αποδοτικός. Συγκεκριμένα, το speedup του παραλλήλου για n>=1024 σχεδόν διπλασιάζεται σε κάθε διπλασιασμό των πυρήνων. Το efficiency είναι συνήθως μεγαλύτερο όταν χρησιμοποιείται μικρότερος αριθμός πυρήνων αλλά αυξάνεται ανάλογα με τον αριθμό των δεδομένων. Τέλος παρατηρούμε πως η

εξωτερική και εσωτερική for έχουν παρόμοιες επιδόσεις, με την εξωτερική να είναι λίγο καλύτερη όταν χρησιμοποιούμε μεγαλύτερο αριθμό πυρήνων (P>=32).

Με βάση τα παραπάνω, καταλήγουμε στο συμπέρασμα ότι η παραλληλοποίηση του κώδικα βελτιστοποιεί τον υπολογισμό των λύσεων μεγάλων γραμμικών συστημάτων (>1000 εξισώσεων) με την μέθοδο απαλοιφής Gauss.

Πολυπλοκότητα:

Η πολυπλοκότητα του σειριακού αλγορίθμου απαλοιφής Gauss είναι O(n^3), όπου n είναι ο αριθμός των αγνώστων. Αυτό συμβαίνει επειδή ο αλγόριθμος περιλαμβάνει τρεις ένθετους βρόχους πάνω από τον επαυξημένο πίνακα, ο καθένας με μέγιστο n επαναλήψεις. Επομένως, η χρονική πολυπλοκότητα είναι ανάλογη του n^3. Το βήμα αντικατάστασης προς τα πίσω, το οποίο επίσης εμπλέκεται στην επίλυση του συστήματος γραμμικών εξισώσεων, έχει πολυπλοκότητα O(n^2), η οποία είναι πολύ μικρότερη από την πολυπλοκότητα της απαλοιφής Gauss. Συνολικά, η πολυπλοκότητα του αλγορίθμου κυριαρχείται από την πολυπλοκότητα της απαλοιφής Gauss, καθιστώντας τον αλγόριθμο O(n^3).

Η πολυπλοκότητα του παράλληλου κώδικα εξαρτάται από τον αριθμό των νημάτων P που χρησιμοποιούνται και το μέγεθος του πίνακα εισόδου n. Υποθέτοντας ότι το n είναι αρκετά μεγάλο και το P <= n, η χρονική πολυπλοκότητα του παραλληλισμένου κώδικα μπορεί να προσεγγιστεί ως O(n^3/P), που αντιπροσωπεύει τον μειωμένο χρόνο υπολογισμού λόγω της παραλληλοποίησης.

Βιβλιογραφία:

1. https://www.youtube.com/watch?v=eDb6iugi6Uk’
2. Διαφάνειες θεωρίας και εργαστηρίου του μαθήματος
3. https://www.cliffsnotes.com/study-guides/algebra/linear-algebra/linear-systems/gaussian-elimination
4. https://mathworld.wolfram.com/GaussianElimination.html
5. https://matrix.reshish.com/gauss-jordanElimination.php