Έτσι για μεγάλο n, $p = \alpha n$ και $\frac{1}{4} \leq \alpha \leq \frac{1}{2}$

προκύπτει:

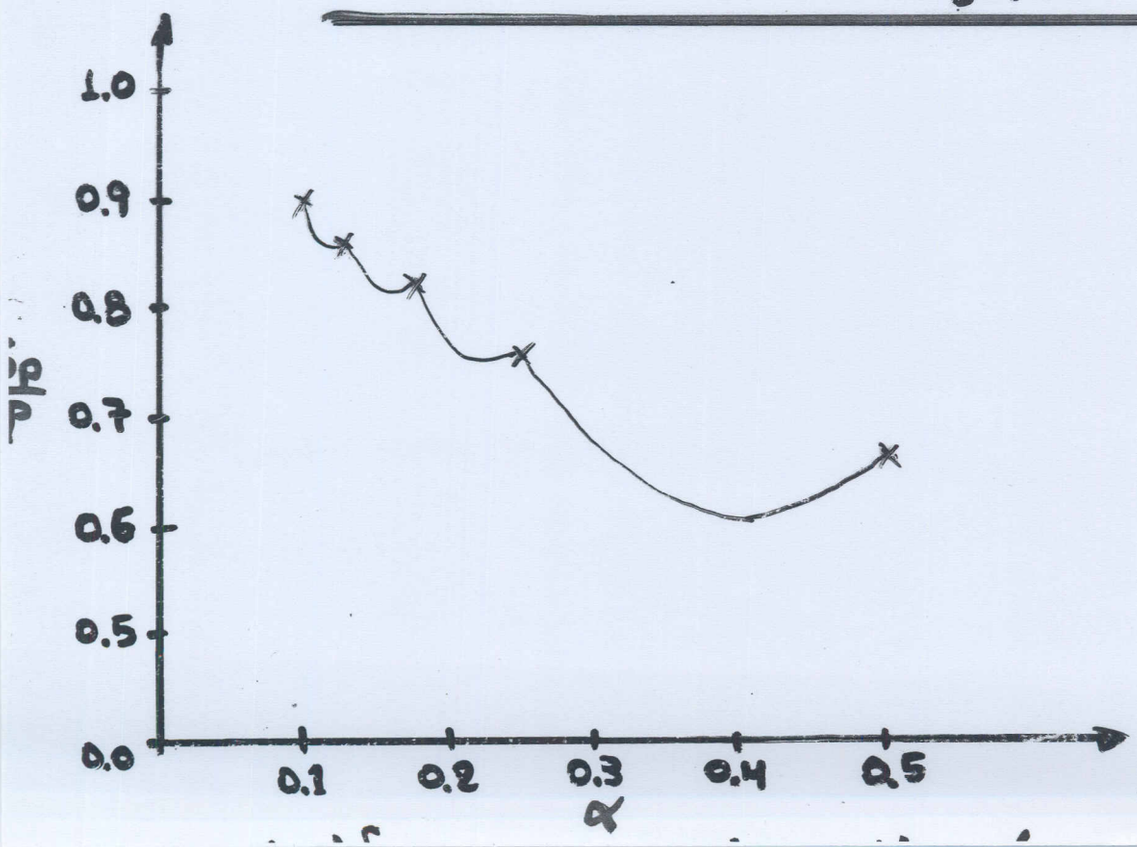
$$\frac{S_p}{p} = \frac{1}{3(2\alpha - 4\alpha^3)}$$

Με παρόμοια ανάλυση προκύπτουν:

$$\frac{S_p}{p} = \begin{cases} \frac{1}{3(3\alpha - 20\alpha^3)}, & \frac{1}{6} \leq \alpha \leq \frac{1}{4} \\ \frac{1}{3(4\alpha - 56\alpha^3)}, & \frac{1}{8} \leq \alpha \leq \frac{1}{6} \end{cases}$$

και γενικά:

$$\frac{S_p}{p} = \frac{1}{3\left((k+1)\alpha - 4\alpha^3 \cdot \sum_{j=1}^{k} j^2\right)}, \quad \frac{1}{2(k+1)} \leq \alpha \leq \frac{1}{2k}$$



ΠΡΑΓΜΑΤΙΚΗ ΚΑΙ ΠΡΟΒΛΕΠΟΜΕΝΗ ΑΠΟΔΟΣΗ

n \ P	2	3	4	5	6	7	8
15	0.888 0.900	0.794 0.815	0.740 0.766	0.651 0.679	0.618 0.652	0.625 0.681	0.581 0.633
20	0.921 0.931	0.843 0.863	0.774 0.798	0.758 0.789	0.670 0.703	0.623 0.656	0.605 0.640
25	0.934 0.944	0.878 0.896	0.830 0.855	0.763 0.739	0.755 0.788	0.692 0.726	0.642 0.675
30	0.942 0.949	0.892 0.911	0.844 0.863	0.818 0.843	0.757 0.783	0.744 0.777	0.710 0.745
35	0.948 0.956	0.901 0.918	0.862 0.880	0.819 0.843	0.790 0.827	0.747 0.779	0.741 0.769

* Οι μικρές διαφορές μεταξύ προβλεπομένης και πραγματικής απόδοσης E_p οφείλονται στους εξής παράγοντες :

1) Ο χρόνος που χρειάζεται η μηχανή για τη δημιουργία statements.

2) Ο συγχρονισμός των μεταβλητών του προγράμματος

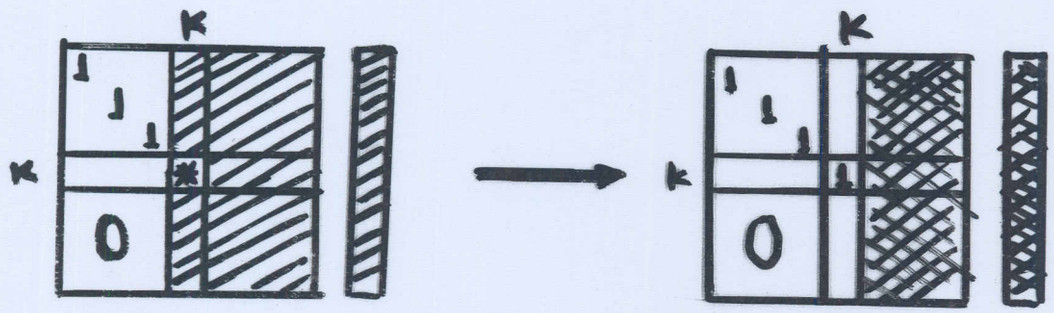
3) Ο έλεγχος των do-loops

4) Άλλες εντολές προγράμματος που δεν λαμβάνονται υπόψη στον τύπο μας.

Β) Αλγόριθμος GAUSS-JORDAN

```
For  κ:=1 to n
     For j:=κ+1 to n+1
         α_κj := α_κj / α_κκ
     For i:=1 to n με i≠κ
         For j:=κ+1 to n+1
             α_ij := α_ij − α_iκ * α_κj
```

} Διαγωνοποίηση



κ βήμα κ+1 βήμα

Αν τώρα η προσπέλαση στα δεδομένα γίνεται κατά γραμμές, προκύπτει άμεσα ο παρακάτω διαχωρισμός σε tasks:

Β1) JORDAN by rows

```
For κ:=1 to n
    T_κκ : { For j:=κ+1 to n+1
                 α_κj := α_κj / α_κκ }
    For i:=1 to n με i≠κ
        T_κi : { For j:=κ+1 to n+1
                     α_ij := α_ij − α_iκ * α_κj }
```

Αν θέλουμε να έχουμε προσπέλαση στα δεδομένα κατα στήλες αντιστρέφουμε τη σειρά των Do loops και προκύπτει :

B2) JORDAN by Columns

For k:=1 to n

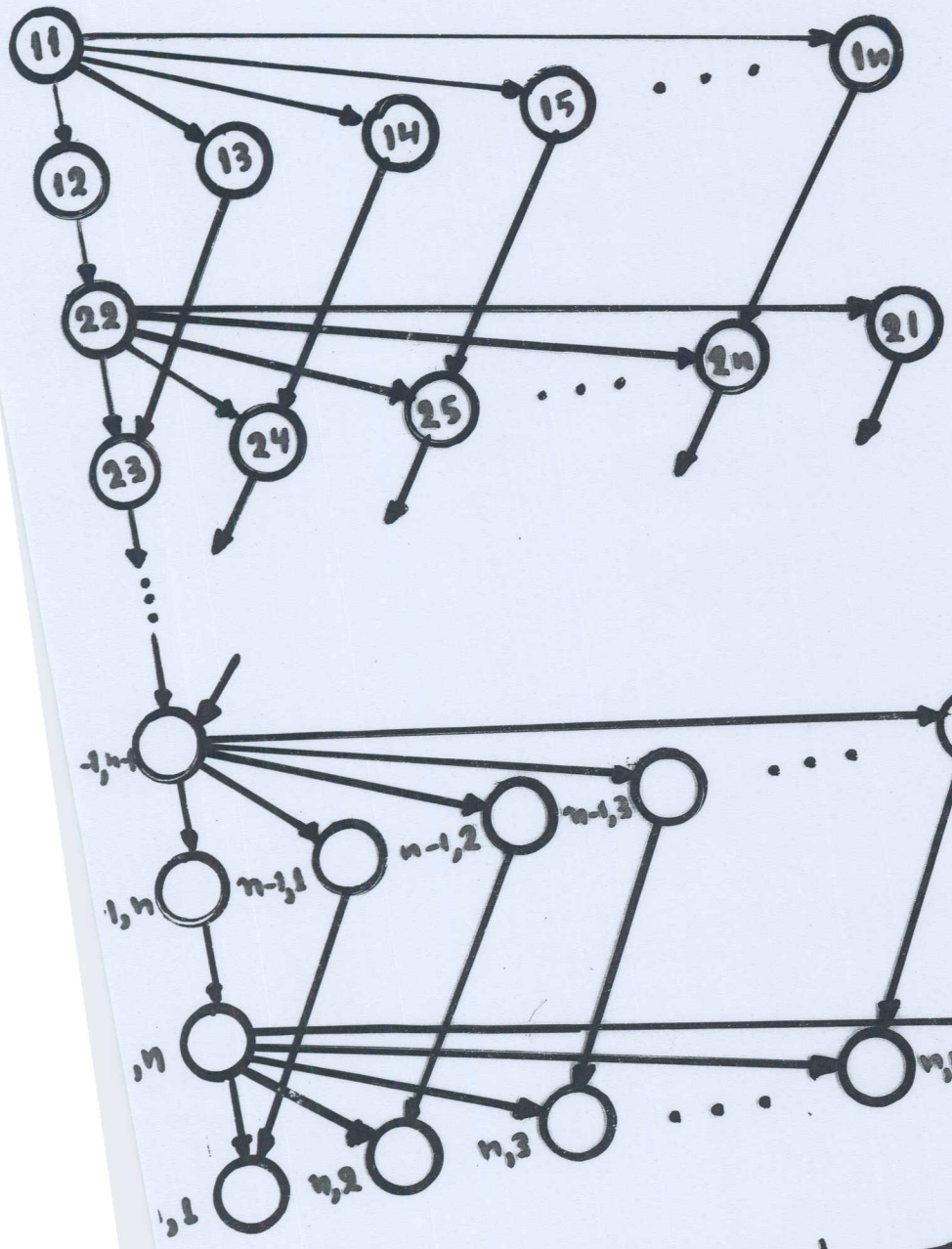
For j:=k+1 to n+1

T_{kj}: { $\alpha_{kj} := \alpha_{kj} / \alpha_{kk}$

For i:=1 to n με $i \neq k$

$\alpha_{ij} := \alpha_{ij} - \alpha_{ik} * \alpha_{kj}$ } } 2n-1 Πράξεις

B1) JORDAN by rows



Για τη μέθοδο αυτή προκύπτουν:

$$t_{opt} = \frac{3n^2}{2} + \frac{3n}{2} \quad , P_{opt} = \frac{2n}{3} + O(1)$$

$$t_{opt}(p) = \frac{n^3}{p} + O(n)$$

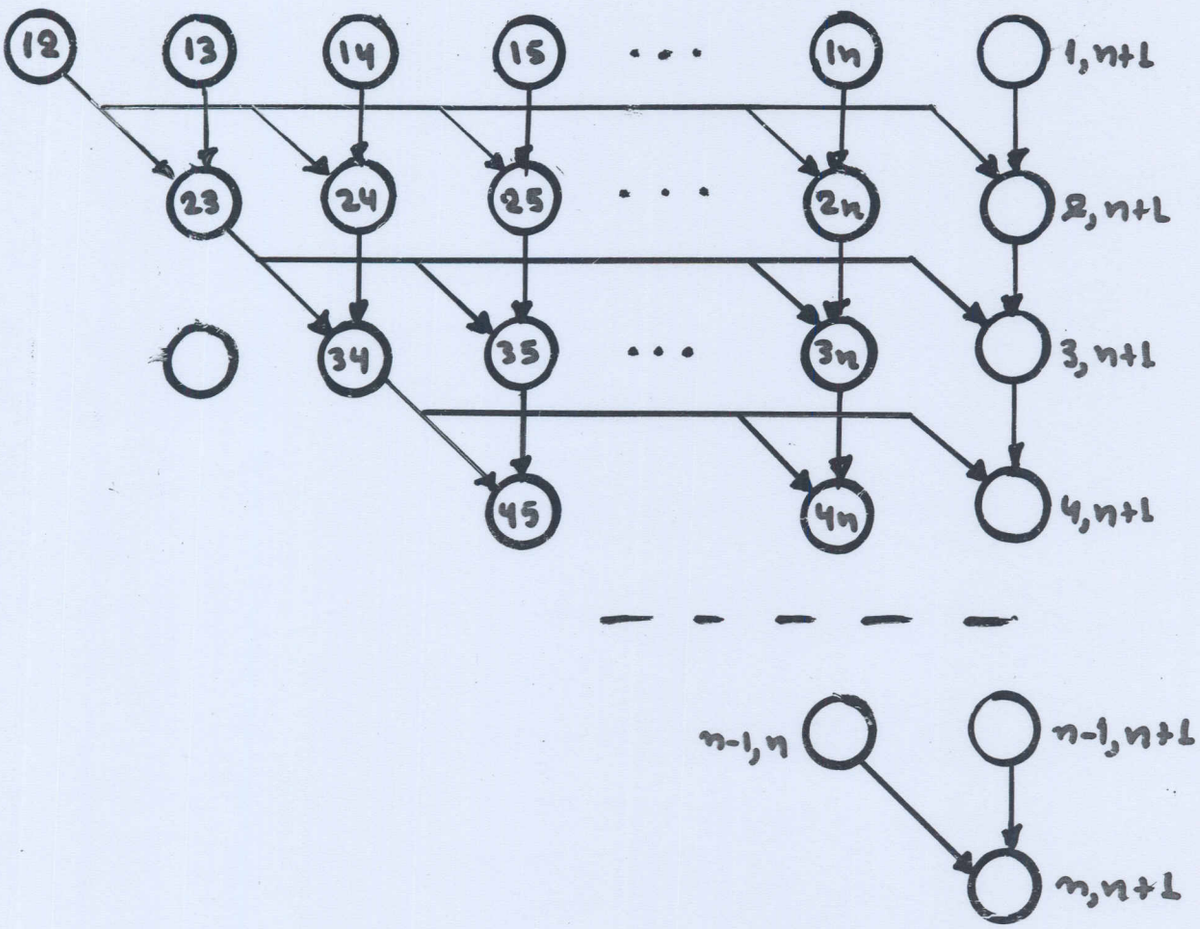
Αν $p = \alpha \cdot n$ τότε:

για $\alpha \leq \frac{2}{3}$ υπάρχει αλγόριθμος με

χρόνο εκτέλεσης $\frac{n^3}{\alpha}$ ασυμπτωτική απόδοση $= 1$

και

B2) JORDAN by columns



Για τη μέθοδο αυτή έχουμε:

$$t_{opt} = 2n^2 - n \qquad p_{opt} = n$$

Για σταθερό p, ο greedy αλγόριθμος είναι optimal και:

$$t_{opt}(p) = \left(\left\lceil \frac{(n-p)(n+p+1)}{2p} \right\rceil + p \right)(2n-1)$$

Συμπέρασμα

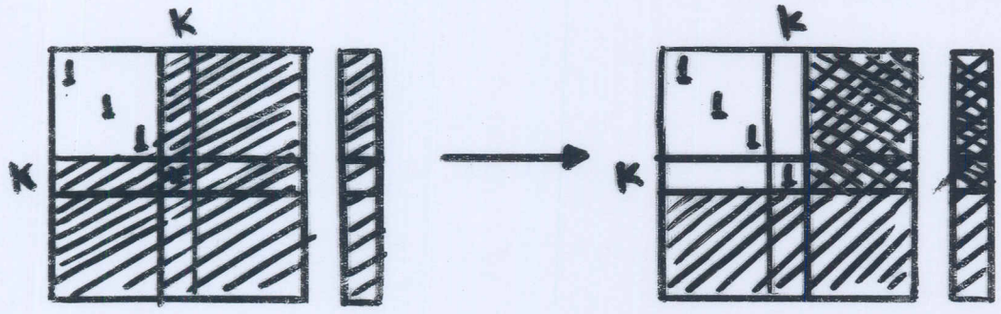
Για οποιοδήποτε α ισχύει:

$$t_{JR}(p) \leq t_{JC}(p) \quad \delta\eta\lambda.$$

η παράλληλη JORDAN by rows είναι πάντα

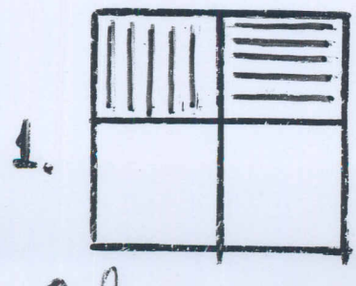
C) Αλγόριθμος HUARD

```
For   k:=1 to n
      (* απαλοιφή της γραμμής k *)
      For i:=1 to k-1
          For j:=k+1 to n+1
              α_kj = α_kj - α_ki * α_ij
      (* ενημέρωση της γραμμής k *)
      c := 1/α_kk
      For j:=k+1 to n+1
          α_kj := α_kj * c
      (* μηδενισμός της στήλης k μέχρι το διαγώνιο όρο *)
      For i:=1 to k-1
          For j:=k+1 to n+1
              α_ij := α_ij - α_ik * α_kj
```

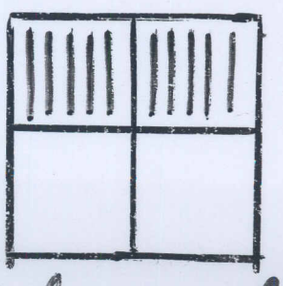


k βήμα k+1 βήμα

Αν τώρα θεωρήσουμε δύο τρόπους προσπέλασης στα δεδομένα :



Columns - rows



Columns - columns

C1) HUARD Columns-rows (HCR)

(* κάθοδος στηλών – άνοδος γραμμών)

```
For k:=1 to n
    (* κάθοδος στηλών *)
    For j:=k to n+1
    Dkj: { For i:=1 to k-1
             αkj := αkj - αki * αij
    (* προετοιμασία της γραμμής k *)
    For j:=k+1 to n+1
    Ckj: { αkj := αkj / αkk
    (* άνοδος γραμμών *)
    For i:=1 to k-1
    Rik: { For j:=k+1 to n+1
             αij := αij - αik * αkj
```

↓ το αντίστοιχο γράφημα των tasks

```
C12  C13  C14  ...  C1,n+1
D22  D23  D24  ...  D2,n+1
     C23  C24  ...  C2,n+1
R12
...  ...  ...  ...  ...
     Dkk  Dk,k+1 ...  Dk,n+1
          Ck,k+1 ...  Ck,n+1
R1k  R2k  ...  Rk-1,k
...  ...  ...  ...
```

C2) HUARD Columns – Columns (HCC)

(* κίνδυνος στηλών – άνοδος στηλών *)

For $k := 1$ to n

For $j := k+1$ to $n+1$

P_{kj} :
- $\alpha_{kj} := \alpha_{kj} / \alpha_{kk}$
- For $i := 1$ to $k-1$
  - $\alpha_{ij} := \alpha_{ij} - \alpha_{ik} * \alpha_{kj}$
- For $i := 1$ to k
  - $\alpha_{k+1,j} := \alpha_{k+1,j} - \alpha_{k+1,i} * \alpha_{ij}$

↓ Το αντίστοιχο γράφημα των tasks

P_{12} P_{13} P_{14} ... $P_{1,n+1}$

P_{23} P_{24} ... $P_{2,n+1}$

...

$P_{n,n+1}$

Συμπέρασμα

Όταν διατίθενται $p = \alpha n$ processors $(0 < \alpha \leq 1)$ τότε για οποιοδήποτε α ισχύει $t_{HCC}(p) \leq t_{HCR}(p)$

Σύγκριση **Parallel JORDAN** και **Parallel HUARD**

Για $\alpha \leq 0.44 \Rightarrow t_{HCC}(p) < t_{JR}(p)$

Για $\alpha \geq 0.45 \Rightarrow t_{JR}(p) < t_{HCC}(p)$

D) MATRIX FACTORIZATION

D1) Αλγόριθμος LU (Doolittle's)

$$A = L \cdot U$$

Στο i-βήμα υπολογίζονται πρώτα τα στοιχεία της i-γραμμής του U από τον τύπο:

$$u_{ij} = \alpha_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \;,\; j = i(1)n$$

και έπειτα τα στοιχεία της i-στήλης του L από τον τύπο:

$$l_{ji} = \left(\alpha_{ji} - \sum_{k=1}^{i-1} l_{jk} u_{ki} \right) / u_{ii} \;,\; j = i+1(1)n$$

Υποθέτοντας

1 time step = 1 διαίρεση

ή 1 αφαίρεση + 1 πολ/σμός

$$\Rightarrow T_1 = \frac{n^3}{3} - \frac{n}{3} \quad \text{steps}.$$

Τώρα ορίζουμε τα tasks όπως φαίνεται στο παρακάτω σχήμα:

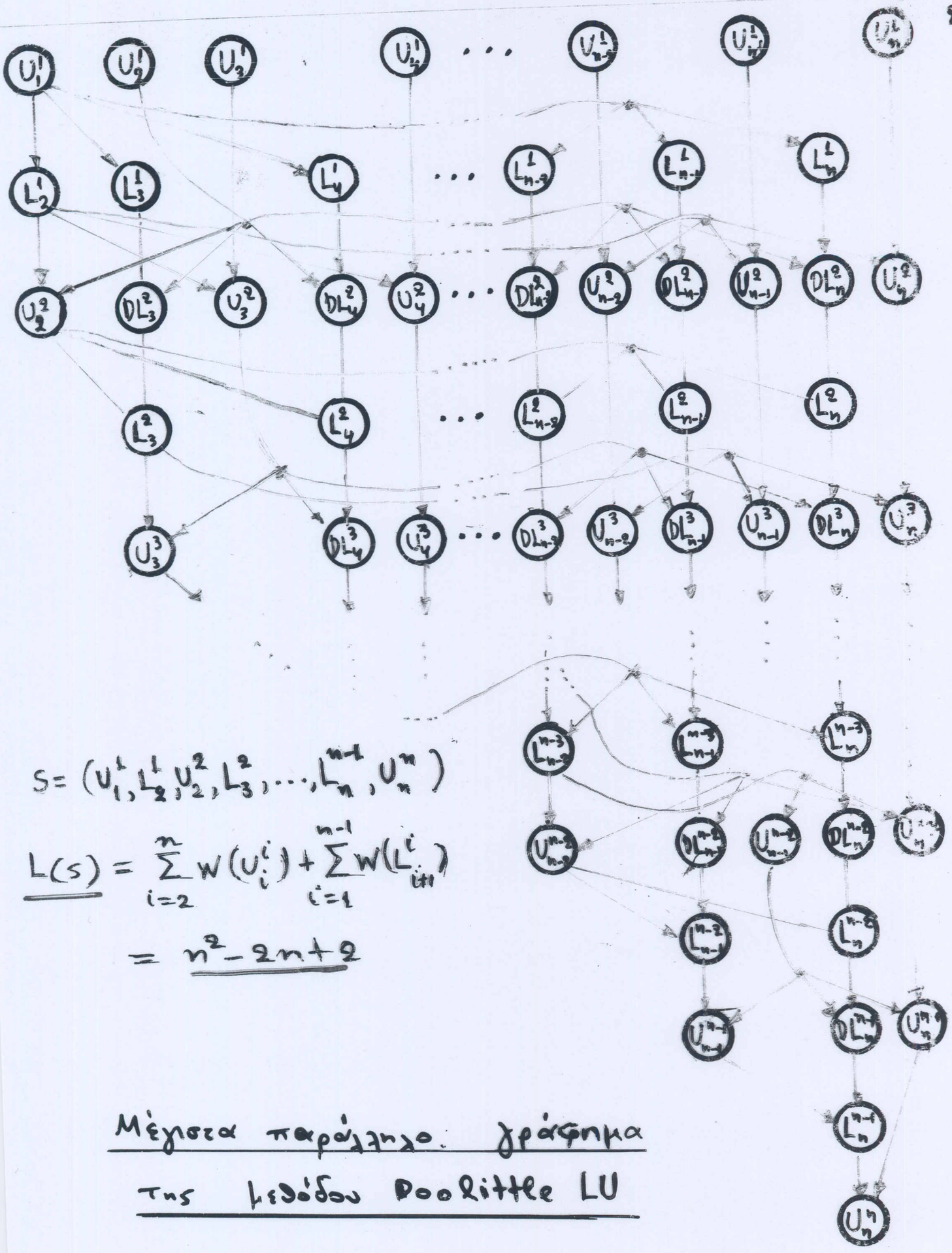
Doolittle αλγόριθμος

U_j^1 {For j:=1 to n
$u_{1j} = \alpha_{1j}$ }

L_j^1 {for j:=2 to n
$l_{j1} := \alpha_{j1} / u_{11}$ }

for i:=2 to n
begin
$U_i^i \{ u_{ii} := \alpha_{ii} - \sum_{k=1}^{i-1} l_{ik} u_{ki} \}$
for j:=i+1 to n
begin
$U_j^i \{ u_{ij} := \alpha_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \}$
$DL_j^i \{ l_{ji} := \alpha_{ji} - \sum_{k=1}^{i-1} l_{jk} u_{ki} \}$
$L_j^i \{ l_{ji} := l_{ji} / u_{ii} \}$
end
end

Το αντίστοιχο μίγματα παράλληλο γράφημα φαίνεται στο παρακάτω σχήμα:



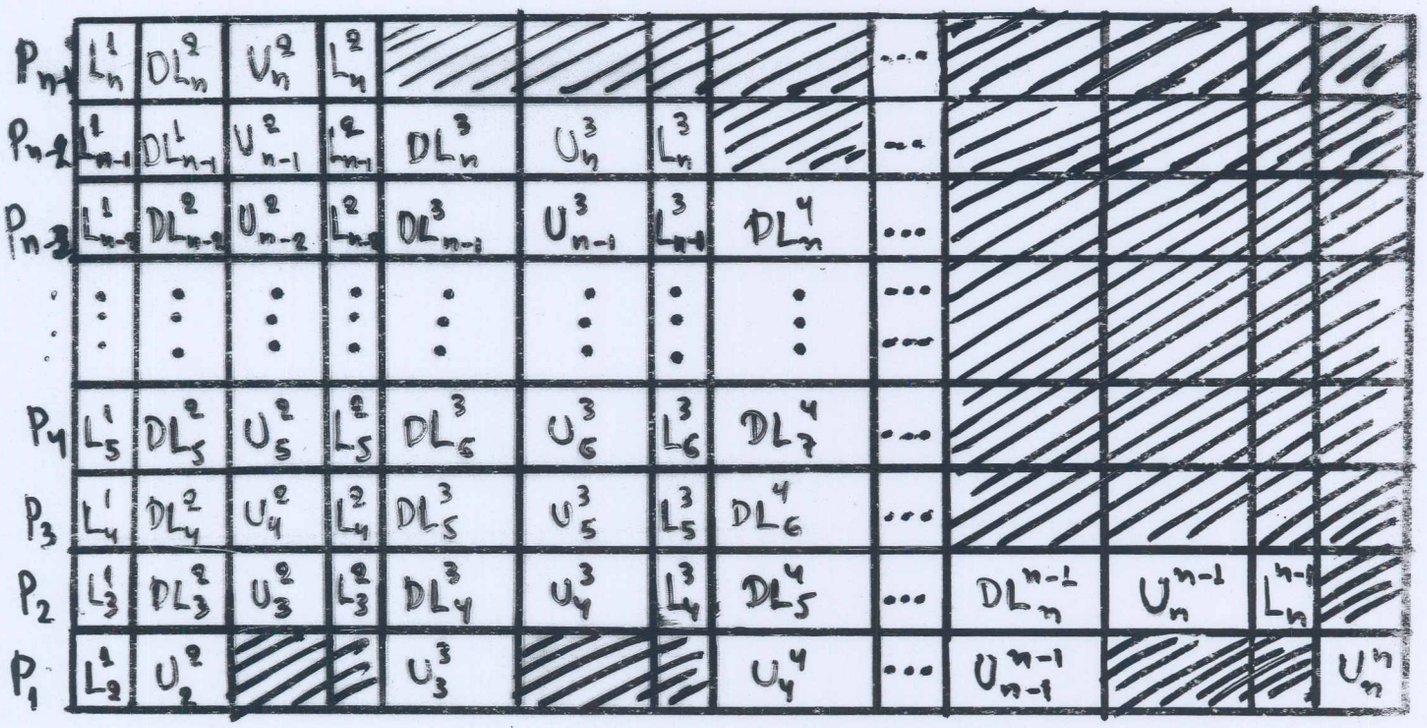
$$S = (U_1^1, L_2^1, U_2^2, L_3^2, \ldots, L_n^{n-1}, U_n^n)$$

$$\underline{L(S)} = \sum_{i=2}^{n} W(U_i^i) + \sum_{i=1}^{n-1} W(L_{i+1}^i)$$

$$= \underline{n^2 - 2n + 2}$$

Μέγιστα παράλληλο γράφημα της μεθόδου Doolittle LU

Χρησιμοποιώντας $p \sim O(n)$ processors μια δρομολόγηση μήκους $L(S)$ είναι η παρακάτω:



Η επιτάχυνση S_p και η απόδοση E_p του αλγορίθμου είναι:

$$S_p = \frac{T_1}{T_p} = \frac{\frac{n^3}{3} - \frac{n}{3}}{n^2 - 2n + 2} \simeq \frac{n}{3}$$ για μεγάλο n

και $$E_p = \lim_{n \to \infty} \frac{S_p}{p} = \lim \frac{\frac{n^3}{3} - \frac{n}{3}}{(n^2 - 2n + 2)(n-1)} = \frac{1}{3} \simeq 0.3333$$

Στη συνέχεια θα εξετάσουμε τη λύση ενός κάτω τριγωνικού συστήματος $\underline{Cu = d}$.

Αλγόριθμος

```
for k:=1 to n
    begin
    T_k^k{u_k = d_k/u_kk}
      for i:=k+1 to n
      T_i^k{d_i:=d_i - c_ik u_k}
```

$$S = (T_1^1, T_2^1, T_2^2, T_3^2, \ldots, T_{n-1}^{n-1}, T_n^{n-1}, T_n^n)$$

$$\underline{L(S)} = \sum_{i=1}^{n} w(T_i^i) + \sum_{i=1}^{n-1} w(T_{i+1}^i) = \underline{2n-1}$$

Μια δρομολόγηση γι'αυτό το πρόβλημα είναι η παρακάτω:

$\underline{\text{Τα tasks του } S \longrightarrow P_1}$

$\underline{T_3^1, T_4^1, T_4^2, T_5^2, \ldots, T_n^{n-2} \longrightarrow P_2}$

και πιό γενικά:

$\underline{T_{2j-1}^1, T_{2j}^1, T_{2j}^2, T_{2j+1}^2, \ldots, T_n^{n-2(j-1)} \longrightarrow P_j}$

Θέτοντας $A = L \cdot U$ στο σύστημα $Au = b$ προκύπτουν:

$L \cdot v = b$ (κάτω τριγωνικό σύστημα) χρόνος εκτέλεσης $n-1$

και $Uu = v$ (πάνω τριγωνικό σύστημα) χρόνος εκτέλεσης n

Άρα ο συνολικός χρόνος εκτέλεσης T_p^D της μεθόδου Doolittle είναι:

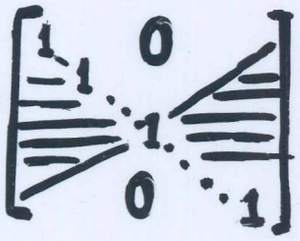
$$T_p^D = n^2 + n$$

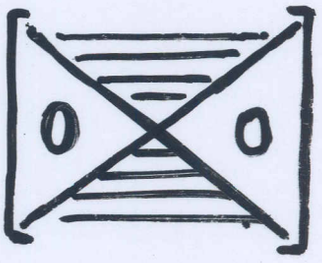
Έτσι η επιτάχυνση S_p και η απόδοση E_p του αλγορίθμου μας δεν επηρεάζεται για μεγάλες τιμές του n.

Παρόμοια ανάλυση γίνεται και για τον αλγόριθμο WZ.

Δ2) Αλγόριθμος WZ

$$A = W \cdot Z$$

όπου $W =$ 

και $Z =$ 

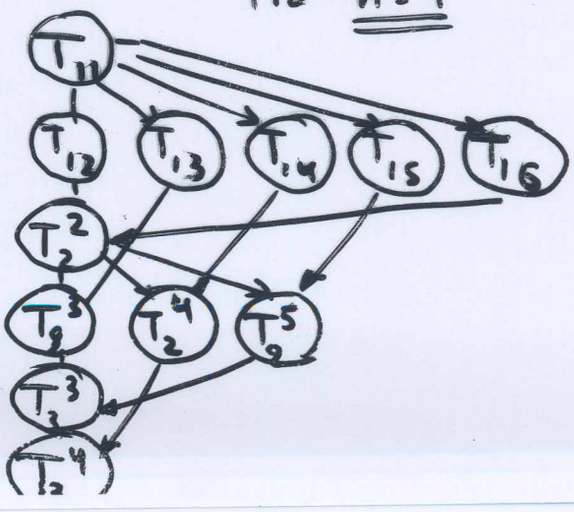

```
For κ:=1 to ⌈(n-1)/2⌉ do
    begin
    T_κ^κ { m:= α_{κ,n-κ+1} / α_{κκ}
            m_1:= α_{n-κ+1,n-κ+1} - m α_{n-κ+1,κ}
            for j:=κ+1 to n-κ do
                begin
                    α_{j,n-κ+1}:= (α_{j,n-κ+1} - m·α_{jκ}) / m_1
                    α_{jκ} := (α_{jκ} - α_{n-κ+1,κ}·α_{j,n-κ+1}) / α_{κκ}
                end }
            for j:= κ+1 to n-κ do
    T_κ^j { for i:=κ+1 to n-κ do
                    α_{ij}:= α_{ij} - α_{iκ} α_{κj} - α_{i,n-κ+1} α_{n-κ+1,j} }
    end.
```

$$W(T_k^j) = \begin{cases} 4(n-2k)+2 & \text{αν } k=j \\ 2(n-2k) & \text{αν } k \neq j \end{cases}$$

$$L(S) = \sum_{i=1}^{m} W(T_i^i) + \sum_{i=1}^{m} W(T_i^{i+1}) \leq \frac{3}{2}n^2 - 2n + \frac{1}{2}$$

$n = \lceil \frac{n-1}{2} \rceil$

Για $n=7$



Μια optimal προσέγγιση είναι να δώσουμε τα tasks του S στον P_1 και τα υπόλοιπα στους $\lceil \frac{n}{2} \rceil - 1$ υπολοίπους processors.

Έτσι προκύπτει: $S_p = \frac{T_1}{T_p} = \frac{n^3/3 + O(n^2)}{3n^2/2 - 2n + 1/2} \simeq \frac{2n}{3}$

και για τείνον n $E_p = \lim_{n \to \infty} \frac{S_p}{p} = \frac{4}{9} \simeq 0.444$

Σύγκριση **Parallel LU**

και **Parallel WZ**

Η μέθοδος WZ είναι πιο αποδοτική από την LU όταν χρησιμοποιήσουμε $p = \lceil n/2 \rceil$ processors.