Μετρικές & Επιδόσεις

Κεφάλαιο 6

Χρόνος εκτέλεσης & επιτάχυνση

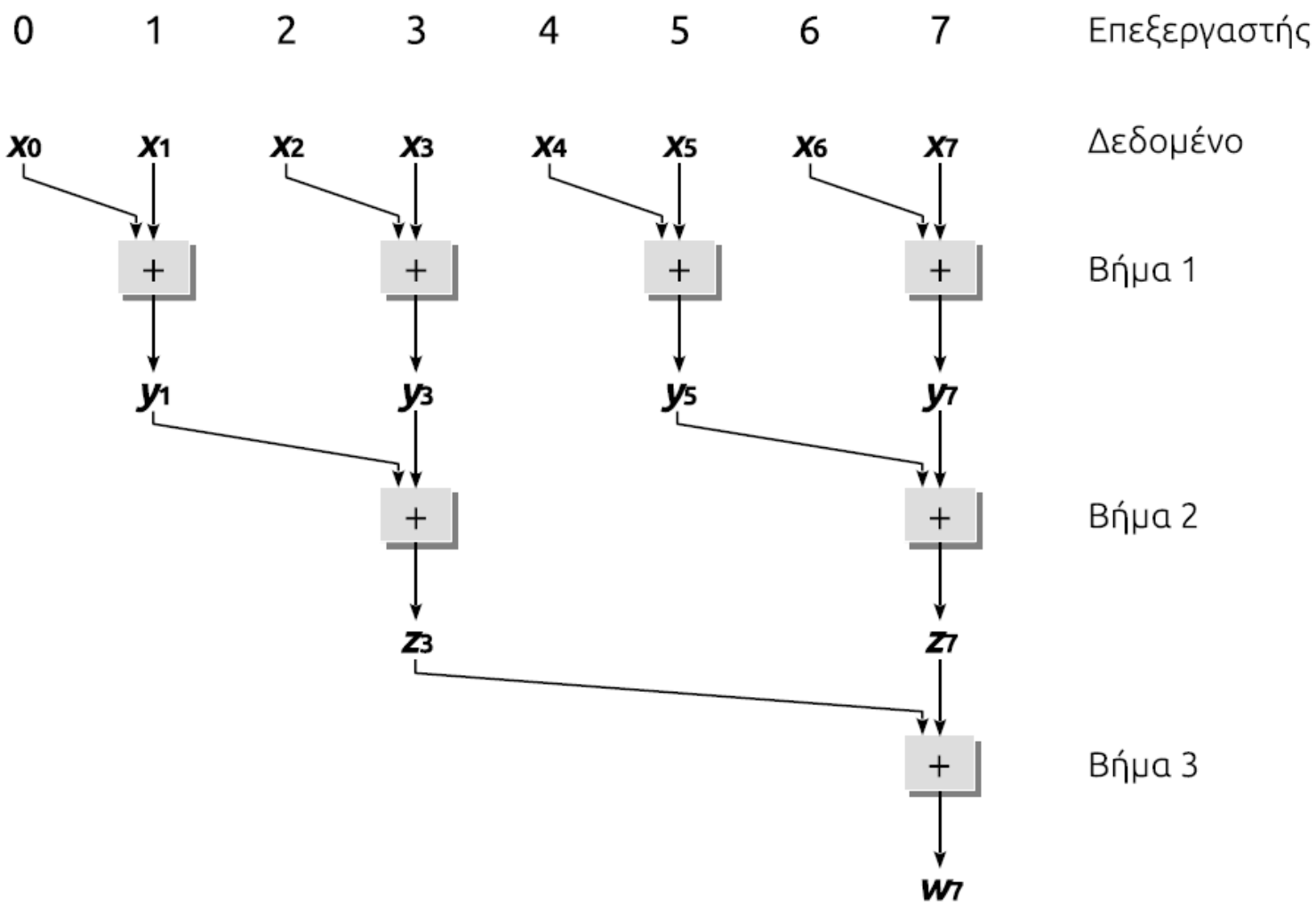
- Σειριακός χρόνος εκτέλεσης:
  - T_1 (για τον καλύτερο σειριακό αλγόριθμο)

- Παράλληλος χρόνος εκτέλεσης:
  - T_n (με n επεξεργαστές)

Επιτάχυνση (speedup):

$$S_n = \frac{T_1}{T_n}$$

Παράδειγμα: πρόσθεση αριθμών



Παράδειγμα: ανάλυση

- *Κατά το βήμα j κάθε επεξεργαστής i, όπου:*

  $i = \{1\times2^{j-1}-1 , 3\times2^{j-1}-1, 5\times2^{j-1}-1, \ldots\}$,

  στέλνει τον αριθμό που διαθέτει, στον επεξεργαστή:

  $i+2^{j-1}$.

- Σε κάθε βήμα, οι αριθμοί μειώνονται κατά το ½, άρα:

$$T_n = \Theta(\log n)$$

- Σειριακός αλγόριθμος για το ίδιο πρόβλημα απαιτεί $T_1 = \Theta(n)$ βήματα. Επομένως,

$$S_n = \Theta\left(\frac{n}{\log n}\right)$$

Ιδεώδης επιτάχυνση & περίεργα

- Με n επεξεργαστές, στην καλύτερη περίπτωση θα χρειαστούμε το 1/n του χρόνου, επομένως το πολύ *γραμμική* (ιδεώδης) επιτάχυνση:

$$S_n \leq n$$

- Έχει, όμως παρατηρηθεί *υπεργραμμική* (superlinear) επιτάχυνση, όπου $S_n > n$ (!)

- Κυριότεροι λόγοι:
  - Το μέγεθος του συστήματος (π.χ. μνήμη) επηρεάζει την ταχύτητα
  - Η τυχαία φύση του αλγορίθμου

Αποδοτικότητα και κόστος

- Η επιτάχυνση δεν μας λέει τίποτε για το πόσο καλά χρησιμοποιούμε το σύστημα
  - (π.χ. επιτάχυνση 10 με 1000 CPUs δεν είναι και τόσο καλό)

Αποδοτικότητα

$$e_n = \frac{S_n}{n} = \frac{T_1}{nT_n} \qquad \left(e_n \le 1\right)$$

- Αν $S_n = n$ ή ισοδύναμα $e_n = 1$ (100%), το πρόγραμμα ονομάζονται *εντελώς παραλληλοποιήσιμο.*

- Στο παράδειγμά μας, $S_n = \Theta\left(\frac{n}{\log n}\right)$ και άρα $e_n = \Theta\left(\frac{1}{\log n}\right)$

Αποδοτικότητα και κόστος

Κόστος

$$c_n = nT_n \qquad \left(c_1 = T_1\right)$$

- Δείχνει *αθροιστικά* πόσος χρόνος αφιερώθηκε από όλους τους επεξεργαστές
- Στην ιδεώδη περίπτωση, πρέπει να αφιερωθεί τόσος χρόνος όσο απαιτεί και η σειριακή έκδοση αλλά στην πράξη για διάφορους λόγους χρειαζόμαστε παραπάνω χρόνο

- Ένα πρόγραμμα θα έχει ***βέλτιστο κόστος*** όταν το κόστος του διαιρούμενο με το σειριακό κόστος είναι σταθερός αριθμός:

$$\frac{c_n}{T_1} = \Theta(1)$$

Εξάρτηση από την αρχιτεκτονική

- Σημαντικός χρόνος που πάει χαμένος και δεν εμφανίζεται στη σειριακή εκτέλεση είναι ο χρόνος που σπαταλιέται σε επικοινωνίες / αλληλεπίδραση μεταξύ των επεξεργαστών:
  - για πολυεπεξεργαστές κοινής μνήμης: αμοιβαίος αποκλεισμός και συγχρονισμός
  - για πολυεπεξεργαστές κατανεμημένης μνήμης: αποστολή / λήψη μηνυμάτων

Παράδειγμα: πρόσθεση αριθμών σε γραμμικό γράφο

Υπενθύμιση:

στο βήμα j αν ο επεξεργαστής i συμμετέχει πρέπει να στείλει τον αριθμό του στον επεξεργαστή $i+2^{j-1}$

- Οι επεξεργαστές i και $i+2^{j-1}$ πόσο απέχουν στον γραμμικό γράφο; Απάντηση: 2^{j-1} ακμές. Άρα:

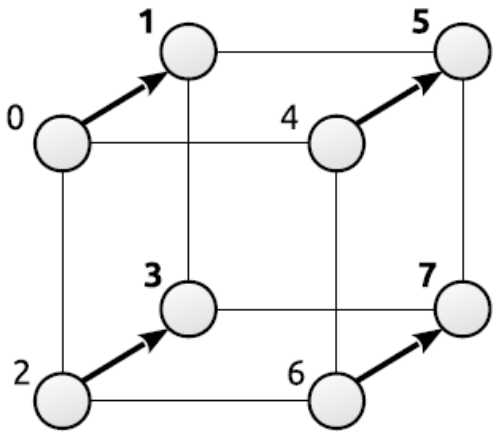
$$T_n = \sum_{j=1}^{\log n} (1 + 2^{j-1})$$

- το οποίο δίνει:

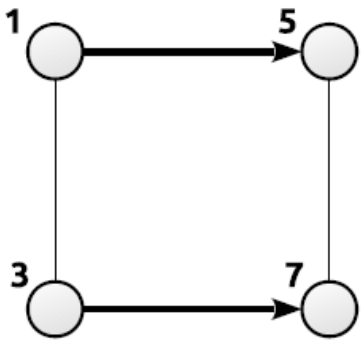
$$T_n = n + \log n - 1$$

- Προσέξτε ότι $T_n > T_1 = n - 1$ (!!)

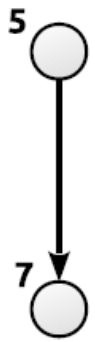
Παράδειγμα: πρόσθεση αριθμών σε υπερκύβο



Βήμα 1



Βήμα 2



Βήμα 3

- Σε κάθε βήμα, οι επεξεργαστές i και $i+2^{j-1}$ *είναι γειτονικοί*.

- Άρα έχουν απόσταση 1 και επομένως, σε κάθε βήμα θέλουμε 1 χρονική μονάδα για επικοινωνία και 1 για υπολογισμό:

$$T_n = 2\log n$$

Λίγοι επεξεργαστές – αύξηση κόκκου παραλληλίας

- Συνήθως οι αλγόριθμοι σχεδιάζονται χωρίς να λαμβάνεται υπόψη το μέγεθος του συστήματος που θα φιλοξενήσει το πρόγραμμα.
  - Γίνεται η υπόθεση ότι για οποιοδήποτε μέγεθος προβλήματος / εισόδου (N) θα διαθέτουμε όσους επεξεργαστές επιθυμούμε ή «μας βολεύει»
  - Για μέγεθος εισόδου N (π.χ. πρόσθεση N αριθμών) συνήθως υποθέτουμε ότι υπάρχουν τουλάχιστον N επεξεργαστές
  - Όταν φτάσουμε στην πράξη, όμως, διαθέτουμε συνήθως πολύ λιγότερους ($n < N$)! Πρέπει να ξανασχεδιάσουμε τον αλγόριθμο;
    - Απάντηση: Όχι, υπάρχει απλός τρόπος να το αποφύγουμε.
- Ας υποθέσουμε ότι μέγεθος εισόδου = N και σχεδιάσαμε τον αλγόριθμό μας για N επεξεργαστές ενώ στην πράξη διαθέτουμε μόνο $n < N$ επεξεργαστές.
  - Οι N είναι οι ***εικονικοί επεξεργαστές***.
  - Οι n είναι οι ***πραγματικοί επεξεργαστές***.
- Μέθοδος: κάθε *πραγματικός* επεξεργαστής αναλαμβάνει την εξομοίωση N/n εικονικών (αύξηση κόκκου παραλληλίας)

Τι να περιμένουμε από αυτό;

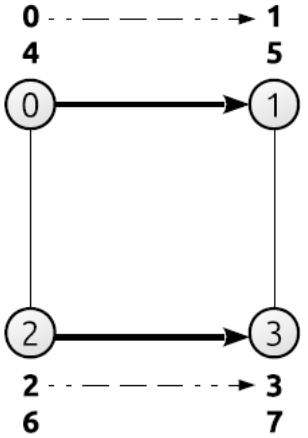
- Αναμενόμενος χρόνος εκτέλεσης:
  - Χρόνοι υπολογισμών αυξημένοι κατά N/n το πολύ
  - Χρόνοι επικοινωνίας αυξημένοι κατά N/n το πολύ, άρα:

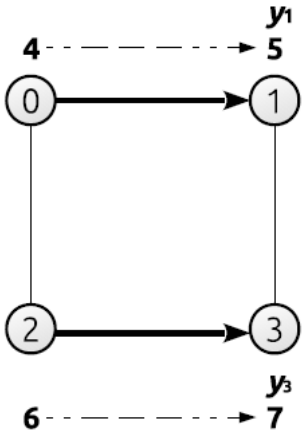
$$T_n = \Theta\left(\frac{N}{n}T_N\right)$$

$$c_n = nT_n = n\Theta\left(\frac{N}{n}T_N\right) = \Theta(NT_N) = \Theta(c_N).$$

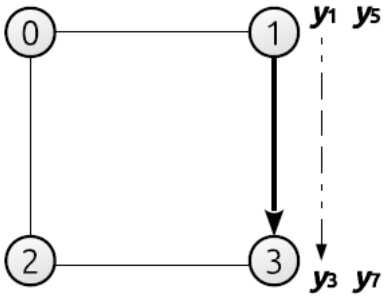
  - Συμπέρασμα: δεν μειώνεται η αποδοτικότητα και δεν αυξάνει το κόστος

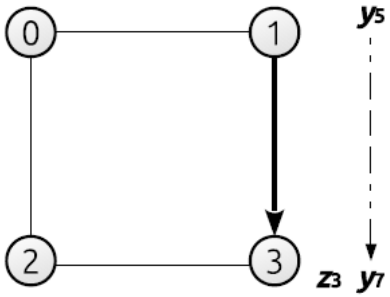
Παράδειγμα: πρόσθεση σε υπερκύβο με κυκλική ανάθεση



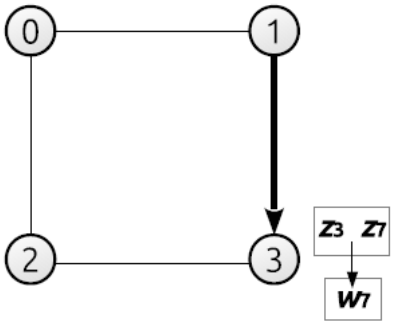


1ο βήμα του αλγορίθμου
(*2 πραγματικά βήματα*)





2ο βήμα του αλγορίθμου
(*2 πραγματικά βήματα*)



3ο βήμα του αλγορίθμου
(*μόνο πρόσθεση*)

Παράδειγμα: ανάλυση

- Για τα πρώτα log n βήματα του αρχικού αλγόριθμου, κάθε επεξεργαστής κάνει τη δουλειά N/n εικονικών, άρα
  - χρόνος εκτέλεσης: Θ((N/n) logn)
- Για τα υπόλοιπα, όλοι οι αριθμοί έχουν μαζευτεί σε έναν επεξεργαστή
  - Άρα χρόνος: Θ(N/n)
- Συνολικά:
  - T_n = Θ((N/n) logn).
- Κόστος
  - $c_n = nT_n$ = Θ(Nlogn),

  το οποίο δεν είναι βέλτιστο, όπως συνέβαινε και στον αρχικό αλγόριθμο.

Λεπτομέρεια

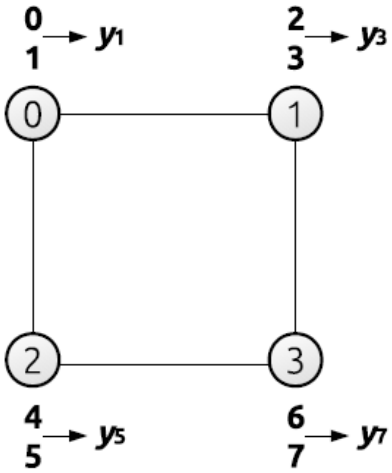
- Με αυτό που κάναμε, ουσιαστικά *αυξήσαμε τον κόκκο παραλληλίας*
  - Κάθε επεξεργαστής εκτελεί περισσότερα πράγματα πλέον
  - Ταυτόχρονα όμως, κάποιες επικοινωνίες πλέον δεν χρειάζεται να γίνουν και έτσι κερδίζουμε κάποιο χρόνο

- Πώς;
  - Περιμέναμε ότι ο χρόνος θα ήταν $\Theta((N/n)T_n) = \Theta((N/n)\ \underline{\log N})$.
  - Όμως είδαμε ότι βγήκε μικρότερος $T_n = \Theta((N/n)\ \underline{\log n})$.

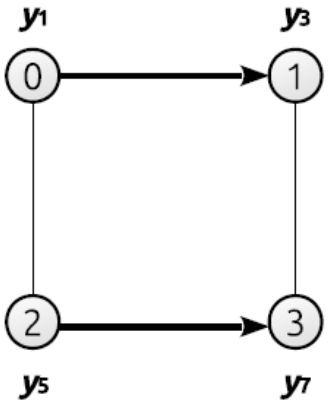
- Η αύξηση του κόκκου στην πράξη μειώνει τις επικοινωνίες
  - Έτσι, μπορεί να μειωθεί και το κόστος, και μάλιστα να γίνει βέλτιστο!

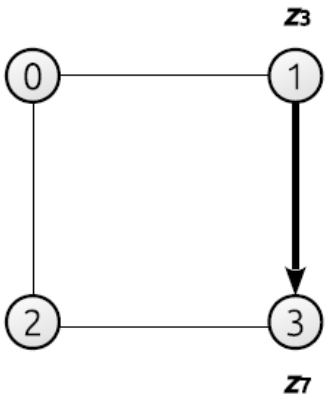
Παράδειγμα: πρόσθεση σε υπερκύβο με βέλτιστο κόστος

- Αλλαγή της ανάθεσης: τμηματική αντί κυκλικής



1η φάση
(*μόνο τοπικές προσθέσεις*)





2η φάση
(*αλγόριθμος σε 2-κύβο*)

Παράδειγμα: ανάλυση

- Στα πρώτα βήματα προσθέσεις τοπικές
  - Θα μείνει 1 αριθμός σε κάθε πραγματικό επεξεργαστή
  - $\Theta(N/n)$
- Στη συνέχεια αλγόριθμος για n αριθμούς
  - $\Theta(\log n)$

- Συνολικά, ο απαιτούμενος χρόνος θα διαμορφωθεί ως:
  - $T_n = \Theta(N/n) + \Theta(\log n) = \Theta(N/n + \log n)$.
- Κόστος
  - $c_n = nT_n = \Theta(N + n\log n)$.

ΣΥΜΠΕΡΑΣΜΑ:

Όσο ο όρος N επικρατεί, δηλαδή $N = \Omega(n\log n)$, το κόστος είναι $\mathbf{c_n = \Theta(N)}$ – δηλαδή βέλτιστο!

Overheads: επιπρόσθετοι χρόνοι λόγω παραλληλισμού

Είναι «άχρηστοι» χρόνοι που δεν εμφανίζονται στη σειριακή εκτέλεση

- Λόγω επικοινωνιών (T_{comm})
  - Στο άθροισμα στον υπερκύβο κάθε βήμα του αλγόριθμου είχε και 1 χρονική μονάδα επικοινωνίας, με logn μονάδες συνολικά. Άρα σε όλο τον υπερκύβο, ο χρόνος που αφιερώθηκε σε επικοινωνίες ήταν $T_{comm} = n\log n$

- Λόγω ανισοκατανομής φόρτου (T_{idle})
  - Στο άθροισμα των αριθμών, ξεχνώντας τις επικοινωνίες, υπάρχουν logn βήματα υπολογισμών. Συνολικά οι επεξεργαστές αφιέρωσαν χρόνο $n\,T_n = n\log n$. Όμως, έγιναν ακριβώς $n-1$ προσθέσεις και άρα τα υπόλοιπα $T_{idle} = n\log n - n + 1$ βήματα χάθηκαν λόγω ανισοκατανομής.

Υπολογισμοί overheads

$$T_{ovh} = T_{comm} + T_{idle}.$$

- Για καθαρούς υπολογισμούς αφιερώθηκε χρόνος:

$$W_n = nT_n - T_{ovh}$$

- Όμως $W_n = T_1$. Επομένως, $T_1 = nT_n - T_{ovh}$, που δίνει:

$$T_n = \frac{T_1 + T_{ovh}}{n}$$

$$S_n = \frac{T_1}{T_n} = n\frac{1}{1 + T_{ovh}/T_1}$$

$$e_n = \frac{S_n}{n} = \frac{1}{1 + T_{ovh}/T_1}$$

$$c_n = nT_n = T_1 + T_{ovh}$$

Κόστος

$$c_n = nT_n = T_1 + T_{ovh}$$

- Αν το πρόγραμμά μας απαιτείται να έχει βέλτιστο κόστος, θα πρέπει:

$$\frac{c_n}{T_1} = 1 + \frac{T_{ovh}}{T_1} = \Theta(1)$$

που σημαίνει ότι *για ένα πρόγραμμα με βέλτιστο κόστος, θα πρέπει να ισχύει*:

$$\boxed{\frac{T_{ovh}}{T_1} = \Theta(1)}$$

Παράδειγμα στον υπερκύβο

- Είδαμε ήδη ότι στον υπερκύβο χρειαζόμαστε χρόνο $T_n = 2\log n$, πράγμα που σημαίνει ότι θα πρέπει:

$$T_{ovh} = nT_n - (n - 1) = 2n\log n - n + 1,$$

αφού γνωρίζουμε ότι απαιτούνται συνολικά $T_1 = n - 1$ προσθέσεις.

- Παρατηρούμε ότι:

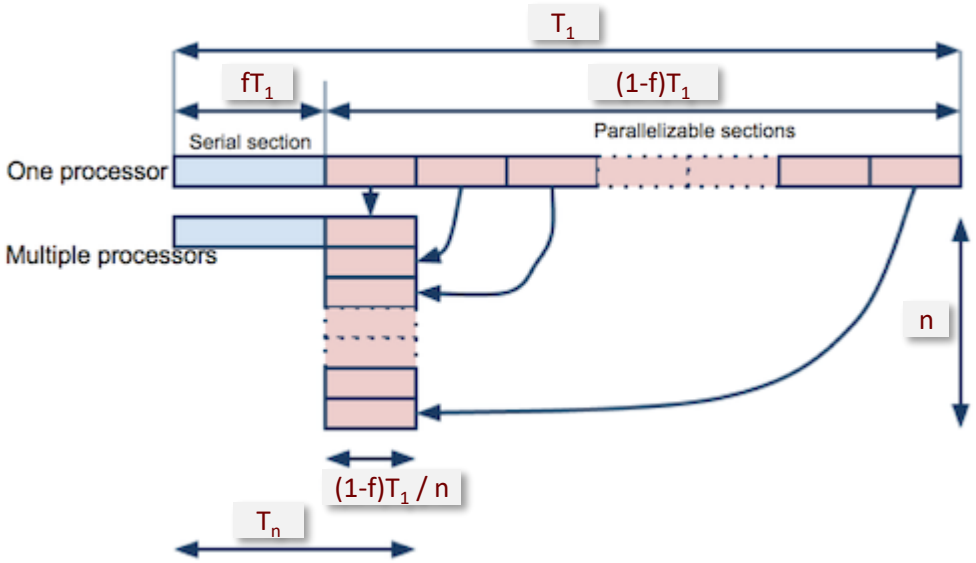
$$T_{ovh} / T_1 = \Theta(n\log n) / \Theta(n) = \Theta(\log n).$$

- Ο αλγόριθμος δεν έχει βέλτιστο κόστος

«Νόμοι» και κλιμακωσιμότητα

Ο νόμος του Amdahl

- Έστω ότι διαθέτουμε n επεξεργαστές και ότι το πρόγραμμά μας έχει τα εξής χαρακτηριστικά:
  - Ένα ποσοστό f των εντολών μπορεί να εκτελεστεί μόνο σειριακά.
  - Το υπόλοιπο ποσοστό (1-f) των εντολών είναι εντελώς παραλληλοποιήσιμο.



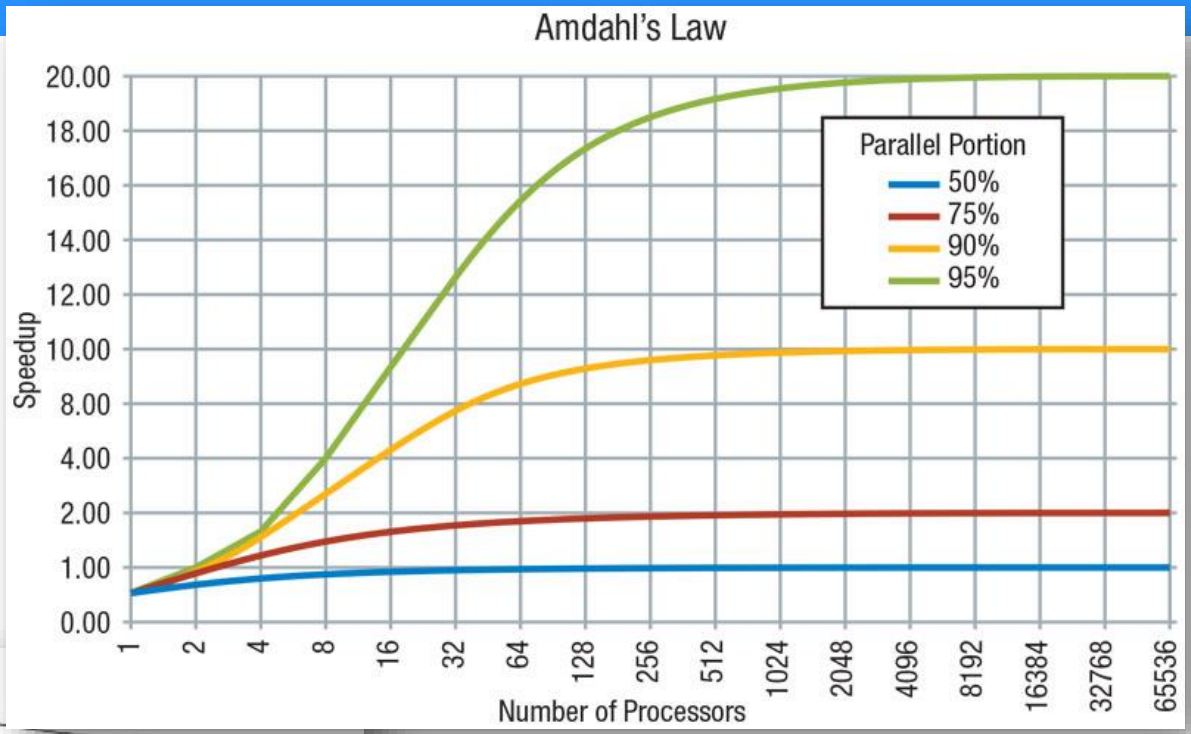
- Τότε:

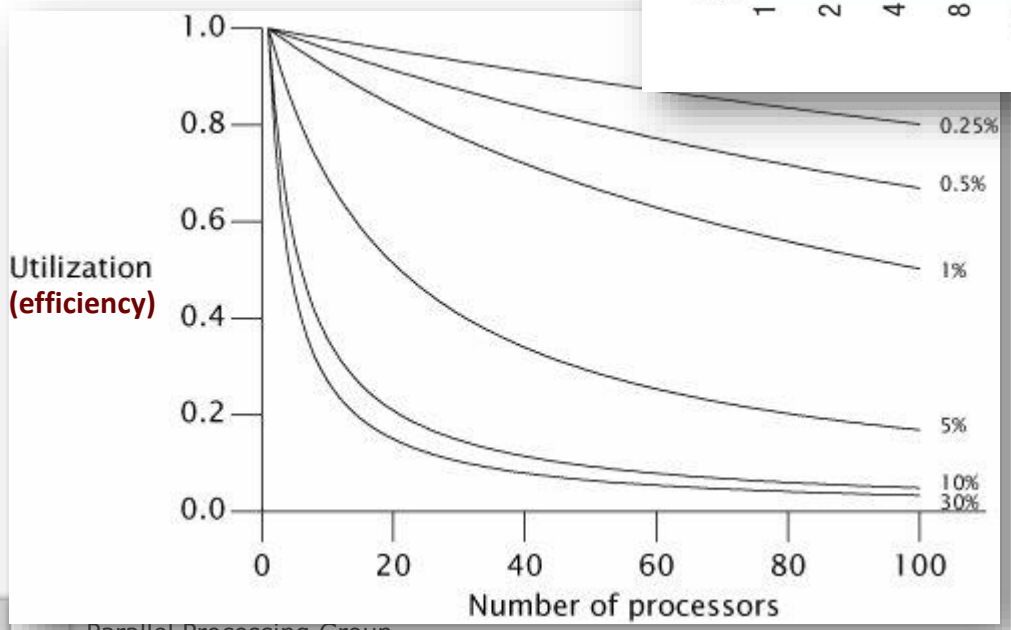
$$T_n = fT_1 + (1-f)\frac{T_1}{n}$$

$$S_n = \frac{T_1}{T_n} = \frac{n}{nf + (1-f)} \leq \frac{1}{f}$$

Ο νόμος του Amdahl γραφικά

- 1967
- Αρνητικός «νόμος»!



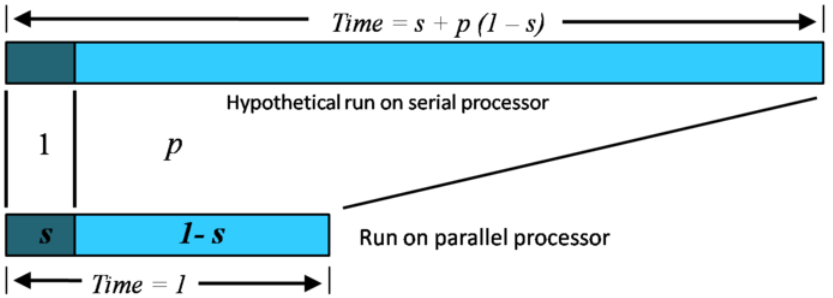


Ο νόμος του Gustafson

- Έστω ότι διαθέτουμε n επεξεργαστές οι οποίοι εκτελούν ένα πρόγραμμα παράλληλα με τα εξής χαρακτηριστικά:
  - Ένα ποσοστό f' των εντολών εκτελείται μόνο του, σειριακά, από έναν μόνο επεξεργαστή.
  - Το υπόλοιπο ποσοστό (1-f') των εντολών εκτελείται εντελώς παράλληλα.
  - Ποια είναι η επιτάχυνση;
- Απάντηση:

  Αν κατά την εκτέλεση του παράλληλου προγράμματος τα f' βήματα ήταν το ποσοστό του σειριακού κομματιού και το υπόλοιπο 1 – f' εκτελέστηκε παράλληλα από τους n επεξεργαστές, τότε σε έναν σειριακό υπολογιστή θα χρειαζόμασταν χρόνο:

$$T_1 = f'T_n + (1-f')nT_n$$

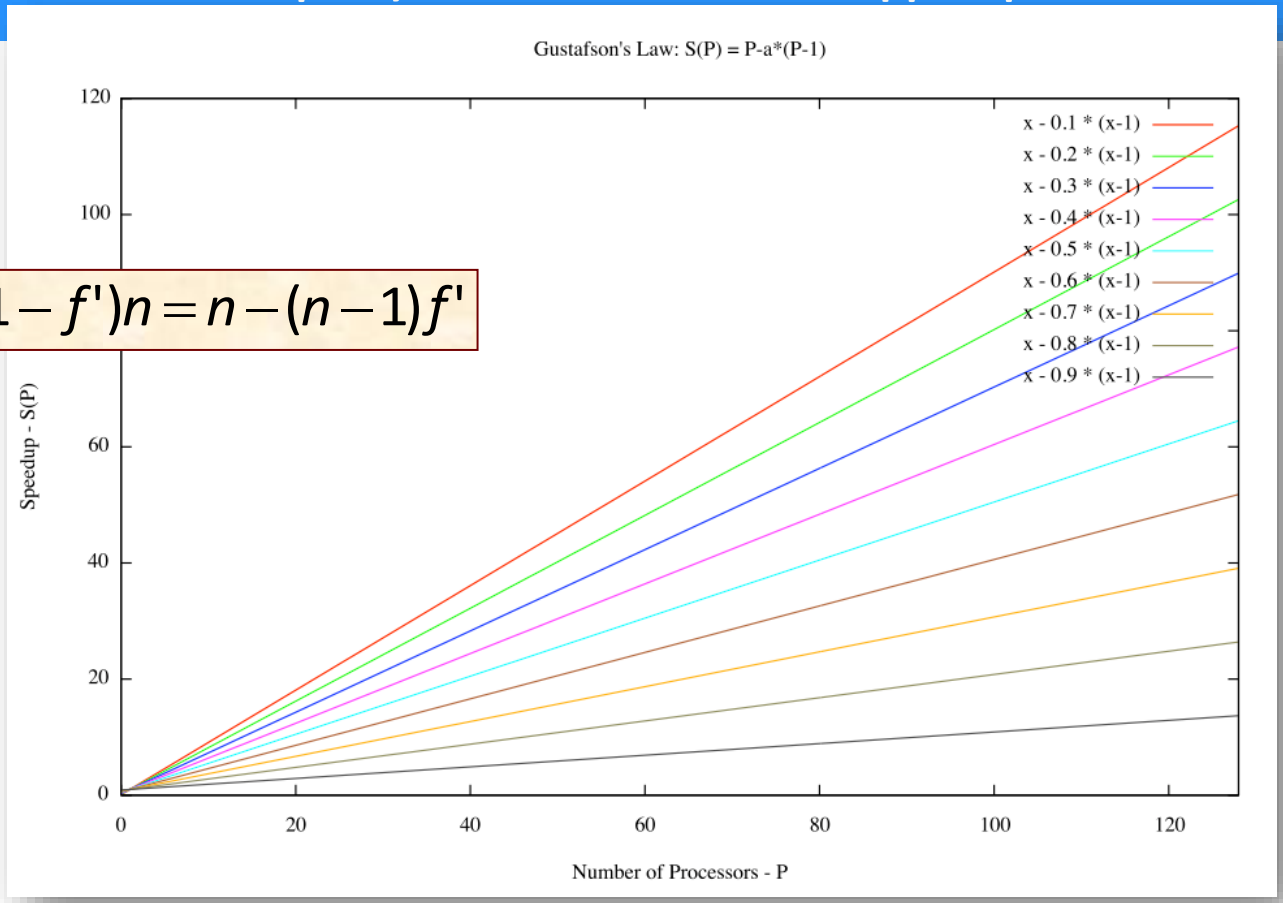


- Επομένως:

$$S_n = f'+(1-f')n = n-(n-1)f'$$

Ο νόμος του Gustafson γραφικά

$$S_n = f' + (1 - f')n = n - (n-1)f'$$



- Μπορούμε να επιτύχουμε μεγάλη επιτάχυνση και κατά συνέπεια αποδοτικότητα (ανάλογα βέβαια με την τιμή του f').
  - Ακόμα και αν f' = 50%, πετυχαίνουμε επιτάχυνση > n/2 !
- Ο νόμος του Amdahl συνεχίζει να ισχύει (!!??).

Σημαντική παρατήρηση

- Πολλές φορές ενδιαφερόμαστε, με αύξηση των επεξεργαστών, μέσα στον ίδιο χρόνο να εκτελέσουμε το πρόγραμμά μας με μεγαλύτερη ακρίβεια => *μεγαλύτερο* πρόβλημα
  - Π.χ. πρόγνωση καιρού
- Πρέπει επομένως να λαμβάνουμε υπόψη μας και το μέγεθος εισόδου του προβλήματος (N) λοιπόν
- Ορθότερη διατύπωση των μέχρι τώρα σχέσεων:

$$S_n(N) = \frac{T_1(N)}{T_n(N)}$$

$$e_n(N) = \frac{S_n(N)}{n} = \frac{T_1(N)}{nT_n(N)}$$

$$c_n(N) = nT_n(N)$$

$$S_n(N) = \frac{n}{nf(N) + (1 - f(N))} \leq \frac{1}{f(N)} \quad \text{(Amdahl)}$$

$$S_n(N) = n - (n-1)f'(N) \quad \text{(Gustafson)}$$

Κλιμακωμένη επιτάχυνση

$$S_n(N) = \frac{n}{nf(N) + (1 - f(N))} \le \frac{1}{f(N)} \Rightarrow e_n(N) \le \frac{1}{nf(N)} \qquad \text{(Amdahl)}$$

- Σωστός ο νόμος του Amdahl: όντως, αν το πρόβλημα παραμείνει ίδιο (N), όσο αυξάνει ο αριθμός των επεξεργαστών (n) σίγουρα θα μειώνεται και η αποδοτικότητα
  - Π.χ. πρόσθεση N = 100 αριθμών σε n = 100000 cores έχει σχεδόν μηδενική αποδοτικότητα
- Τι γίνεται όμως αν **ΜΕΓΑΛΩΣΩ** το πρόβλημα (N) όταν μου δοθούν παραπάνω επεξεργαστές (n), π.χ. για βελτίωση πρόγνωσης καιρού;
  - Συνήθως η αύξηση («κλιμάκωση») του N οδηγεί σε βελτιωμένη παράλληλη εκτέλεση (το σειριακό κομμάτι δεν αλλάζει πολύ, αυξάνει κυρίως το παραλληλοποιήσιμο) και άρα μικραίνει το ποσοστό f του Amdahl, ενώ αυξάνει το ποσοστό f' του Gustafson.

$$S_n(N) = n - (n-1)f'(N) \qquad \text{(Gustafson)}$$

- *Gustafson: μπορούμε να επιτύχουμε οποιαδήποτε επιτάχυνση και κατά συνέπεια οποιαδήποτε αποδοτικότητα 'ρυθμίζοντας' κατάλληλα το f'(N).*
  - Η ρύθμιση αυτή φυσικά γίνεται μέσω του N, δηλαδή βρίσκοντας το κατάλληλο μέγεθος εισόδου το οποίο επιφέρει το επιθυμητό αποτέλεσμα.
  - Έτσι αν αυξηθεί ο αριθμός των επεξεργαστών (γίνει 'κλιμάκωση' προς τα πάνω, του n) πρέπει να γίνει κλιμάκωση και του προβλήματος (N) προκειμένου να επιτευχθεί η απαιτούμενη επιτάχυνση.
  - Για το λόγο αυτό η σχέση αναφέρεται και ως *κλιμακωμένη επιτάχυνση* (*scaled speedup*).

Κλιμάκωση κλπ

- **κλιμάκωση** (προς τα πάνω / κάτω) – scaling
  - Αύξηση / μείωση κάποιου χαρακτηριστικού (π.χ. # CPUs)

- **Κλιμακωσιμότητα ή ικανότητα κλιμάκωσης** (scalability)
  - Ικανότητα να *διατηρείται κάποια ιδιότητα σταθερή* όταν κλιμακώνεται κάποια άλλη

- Συστήματα (αλγόριθμος + αρχιτεκτονική) **ικανά κλιμάκωσης** (scalable systems)
  - *Ένα παράλληλο σύστημα (αλγόριθμος + αρχιτεκτονική) είναι ικανό κλιμάκωσης αν για κάθε κλιμάκωση του αριθμού επεξεργαστών υπάρχει κλιμάκωση του μεγέθους εισόδου έτσι ώστε η αποδοτικότητα να παραμένει σταθερή.*

Παράδειγμα στον υπερκύβο

- Πρώτη μέθοδος (κυκλική ανάθεση εικονικών επεξεργαστών)
  - Χρόνος εκτέλεσης $T_n = \Theta((N/n)\log n)$,
  - Αποδοτικότητα $e_n = \Theta(1/\log n)$.
- Αν αυξηθεί ο αριθμός των επεξεργαστών, η αποδοτικότητα θα μειωθεί, ανεξάρτητα από οποιαδήποτε αύξηση στο μέγεθος εισόδου. *Το σύστημα αυτό ΔΕΝ είναι ικανό κλιμάκωσης.*

- Δεύτερη μέθοδος (τμηματική ανάθεση)
  - Χρόνος εκτέλεσης $T_n = \Theta(N/n + \log n)$
  - Αποδοτικότητα $e_n = \Theta(N / (N + n\log n))$.
  - Όσο ο όρος N επικρατεί στον παρανομαστή, δηλαδή $N = \Omega(n\log n)$, η αποδοτικότητα είναι: $e_n = \Theta(1)$.
- Επομένως, για κάθε αριθμό επεξεργαστών (n), υπάρχει ένα μέγεθος εισόδου (N) έτσι ώστε η αποδοτικότητα να παραμένει σταθερή. *Αυτό το σύστημα έχει ικανότητα κλιμάκωσης.*

Ποιότητα της ικανότητας κλιμάκωσης

- Χαμηλή ικανότητα κλιμάκωσης (poorly scalable):
  - μικρή αύξηση των επεξεργαστών απαιτεί μεγάλη αύξηση στο μέγεθος εισόδου του προβλήματος προκειμένου να διατηρηθεί σταθερή η αποδοτικότητα.

- Υψηλή ικανότητα κλιμάκωσης (highly scalable)
  - μικρή αύξηση των επεξεργαστών απαιτεί και μικρή αύξηση στο μέγεθος εισόδου.

Βελτιστότητα κόστους και ικανότητα κλιμάκωσης

- Τα συστήματα που είναι ικανά κλιμάκωσης, μπορούν πάντα να γίνουν βέλτιστου κόστους με κατάλληλη επιλογή των n και N, αφού
  - σταθερή αποδοτικότητα σημαίνει σταθερός λόγος T_1 / nT_n,
  - που αντίστοιχα σημαίνει σταθερός λόγος T_1 / c_n, δηλαδή βέλτιστο κόστος.