6/6/2023
Μετρικές και επιδόσεις

Λ8

Συστήματα & Λογισμικό Υψηλών Επιδόσεων

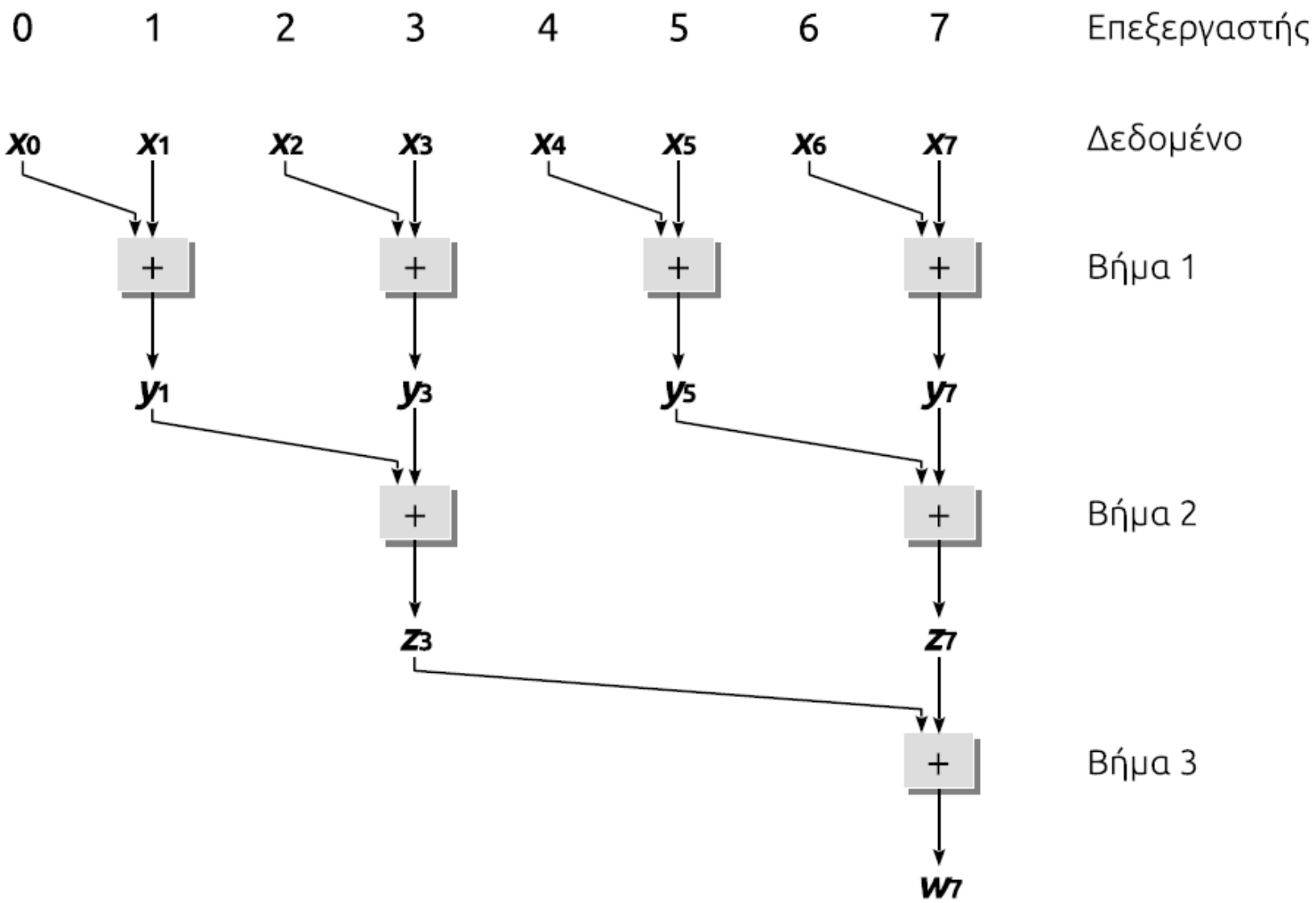
Χρόνος εκτέλεσης & επιτάχυνση

- 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

$$e_n = \frac{S_n}{n} = \frac{T_1}{nT_n} \qquad (e_n \leq 1)$$

- 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:

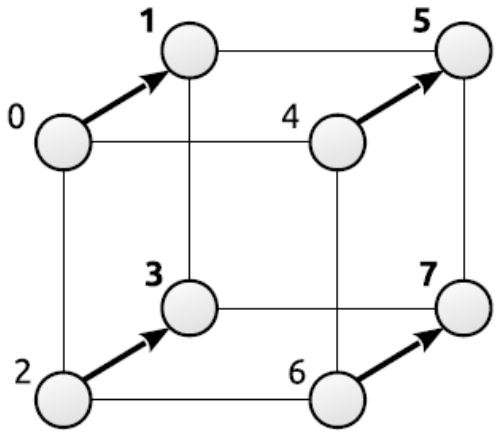
$$T_n = \sum_{j=1}^{\log n} (1 + 2^{j-1})$$

- το οποίο δίνει:

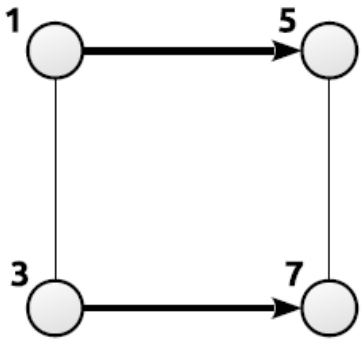
$$T_n = n + \log n - 1$$

- 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:

$$T_n = 2\log n$$

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:

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

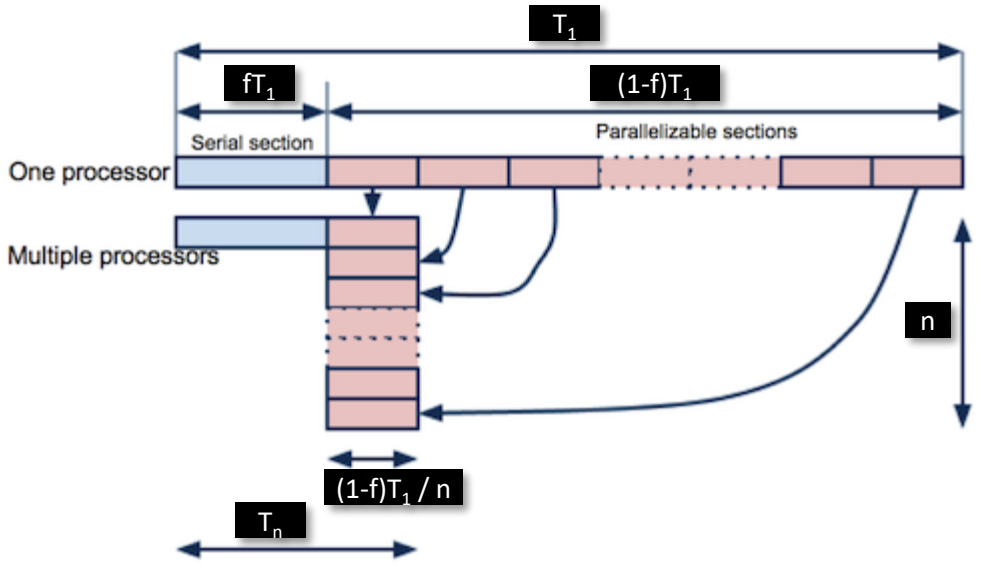
που σημαίνει ότι *για ένα πρόγραμμα με βέλτιστο κόστος, θα πρέπει να ισχύει*:

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

Οι δύο νόμοι

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.



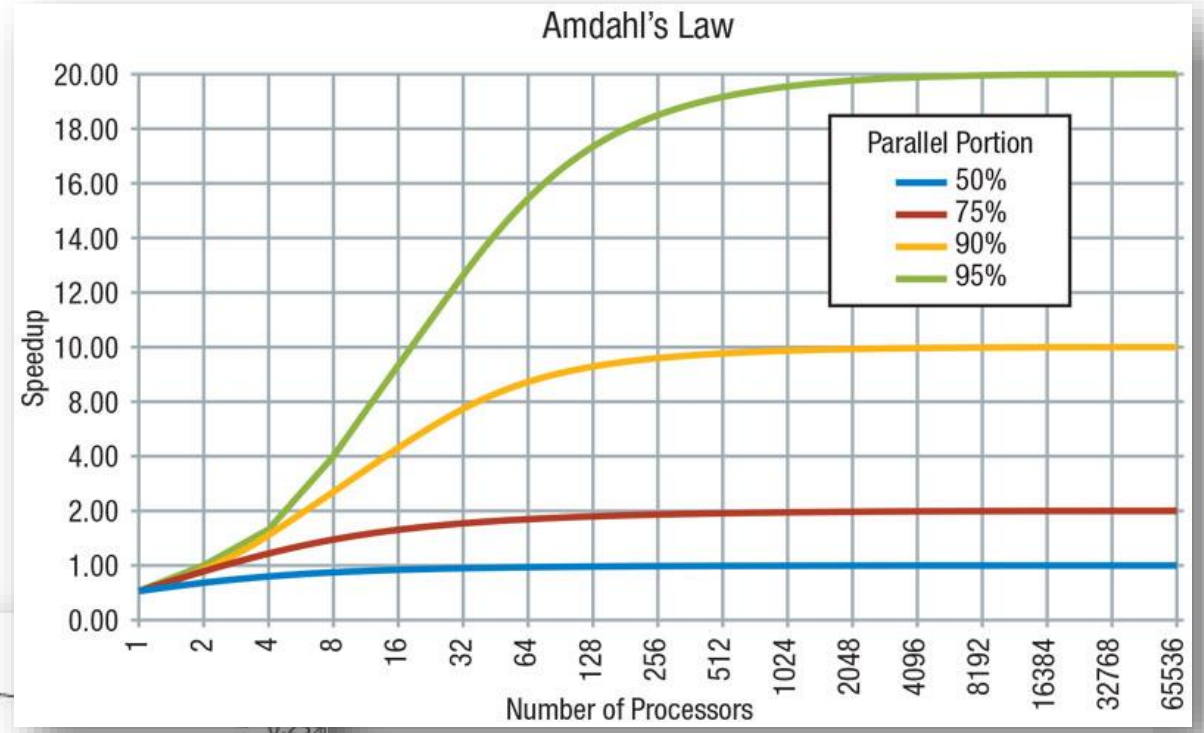
- Τότε:

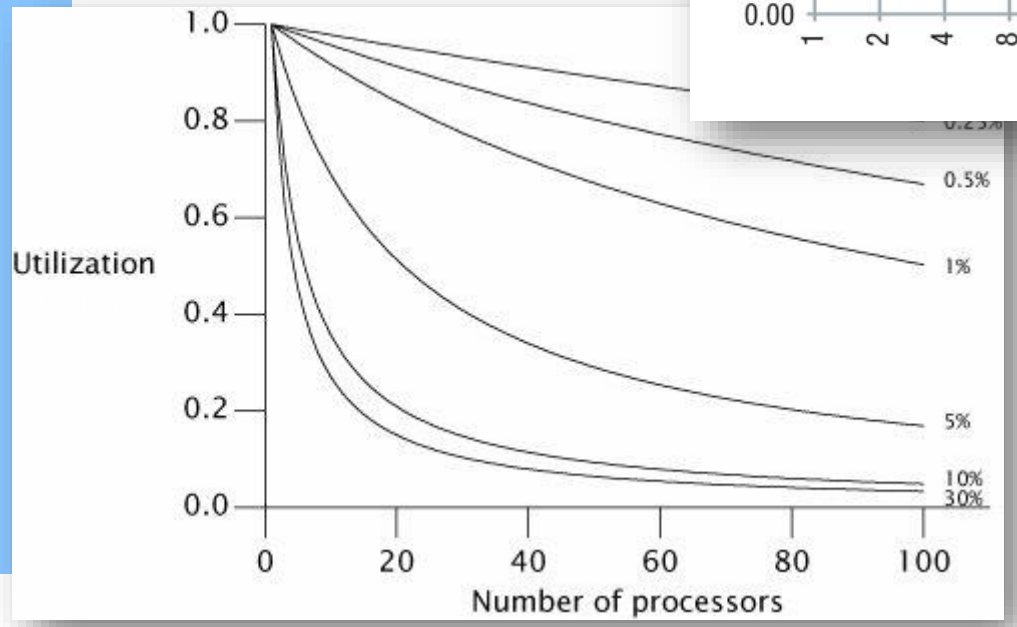
$$T_n = fT_1 + (1-f)\frac{T_1}{n}$$

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

Ο νόμος του Amdahl γραφικά

- 1967
- Αρνητικός «νόμος»!



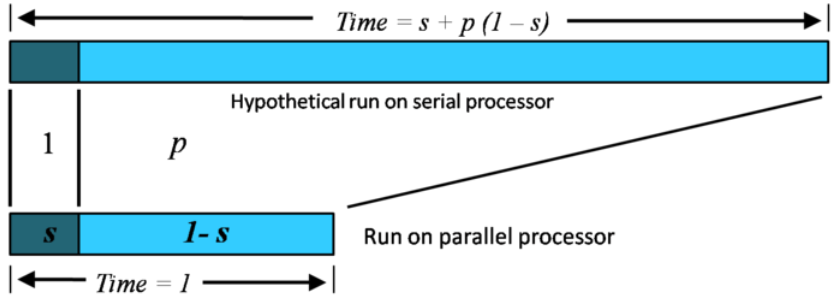


Ο νόμος του Gustafson

- 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:

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

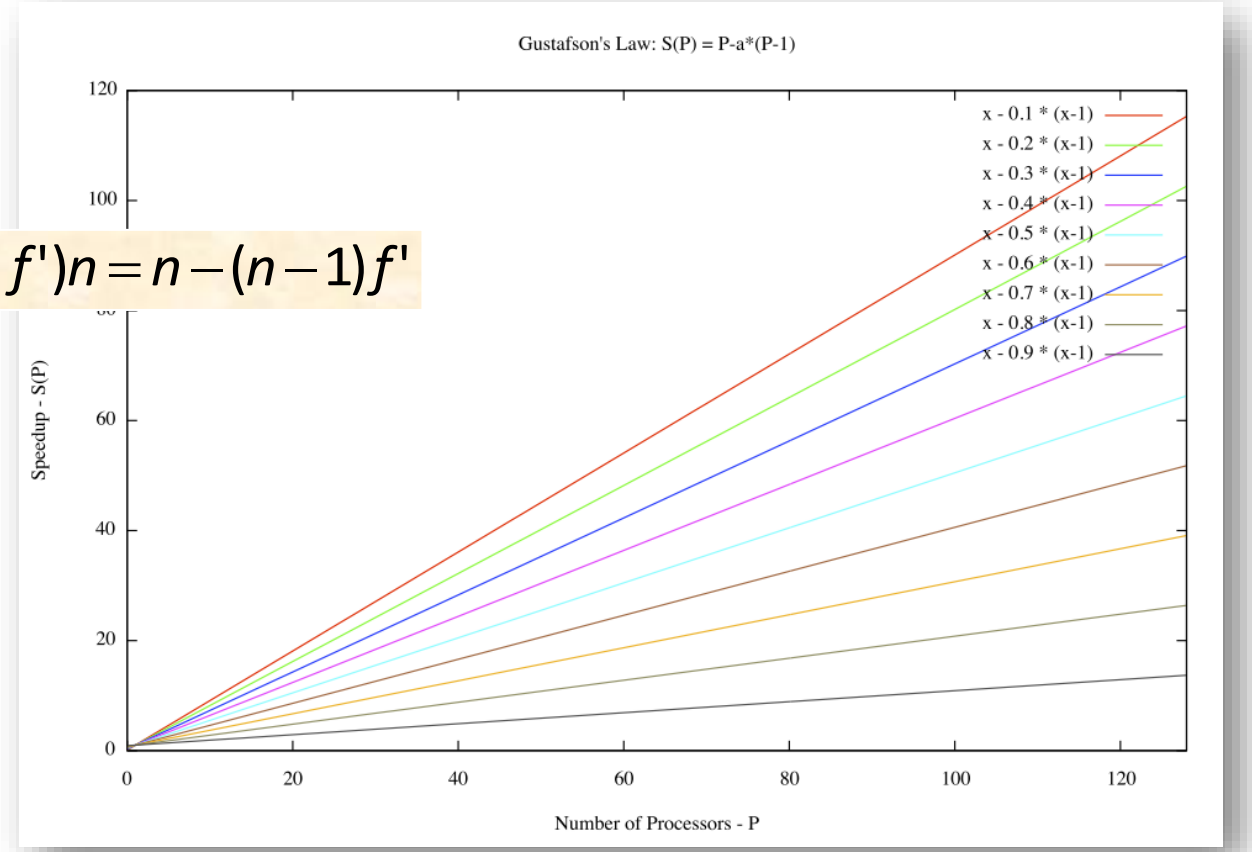


- Επομένως:

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

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

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

- 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*).

Final projects!

Ανάθεση

Αγγελική	Merge sort ή Mini proxy apps
Άρης	SUMMA – πολλαπλασιασμός πινάκων με επιδόσεις
Κωνσταντίνα	NPB – NAS parallel benchmarks (ενημέρωση της έκδοσης C)
Μαρία	Intel samples
Παναγιώτης	SUDOKU – δημιουργία και λύση με παραλληλισμό

Milestones & progress

- Εως 15/6, περιμένω επιγραμματικά & με links:
  - μάζεμα σχετικού υλικού (papers, documents κλπ)
  - ξεκαθάρισμα υλοποιήσεων στις οποίες θα βασιστούμε
- Έως 22/6 [ατομικές συναντήσεις]
  - πλήρης γνώση και καταγραφή του τι έχει γίνει μέχρι τώρα
  - απαιτήσεις υλοποίησης, καθορισμός μηχανημάτων κλπ
  - αρχικές δοκιμές με υπάρχοντες κώδικες ή/και δικούς μας
- Έως 29/6
  - υλοποιήσεις
  - πρώτα αποτελέσματα
- Έως αρχές Ιουλίου
  - τελειώματα υλοποιήσεων και μετρήσεων
  - πλήρες report μαζί με οδηγό χρήσης του κώδικα