Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχ. και Μηχανικών Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Παράλληλος προγραμματισμός:
Σχεδίαση και υλοποίηση παράλληλων προγραμμάτων

Συστήματα Παράλληλης Επεξεργασίας

9ο Εξάμηνο

National Technical University of Athens
CSLab

Σύνοψη παρουσίασης

- **Παράλληλες υπολογιστικές πλατφόρμες**
  - **PRAM: Η ιδανική παράλληλη πλατφόρμα**
  - **Η ταξινόμηση του Flynn**
  - **Συστήματα κοινής μνήμης**
  - **Συστήματα κατανεμημένης μνήμης**
- **Ανάλυση παράλληλων προγραμμάτων**
  - **Μετρικές αξιολόγησης επίδοσης**
  - **Ο νόμος του Amdahl**
  - **Μοντελοποίηση παράλληλων προγραμμάτων**
- Οργάνωση πρόσβασης στα δεδομένα
- Παράλληλα προγραμματιστικά μοντέλα
  - Κοινού χώρου διευθύνσεων
  - Ανταλλαγής μηνυμάτων

Σύνοψη παρουσίασης

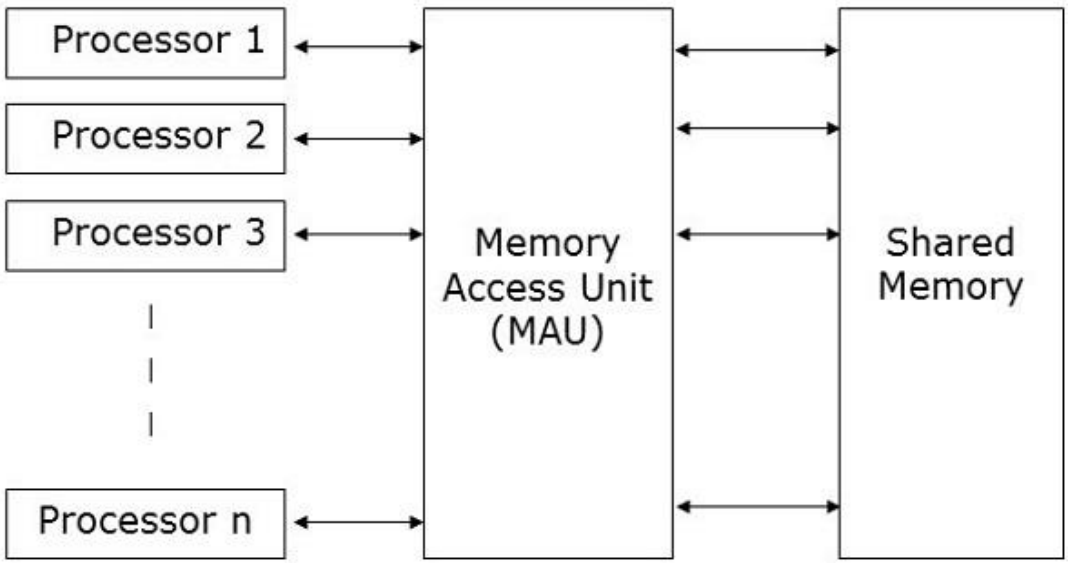
- Σχεδίαση και υλοποίηση παράλληλων προγραμμάτων
  - Παραλληλισμός σε επίπεδο εργασίας (task parallelism)
  - Παραλληλοποίηση σε επίπεδο δεδομένων (data parallelism)
  - Παραλληλοποίηση σε επίπεδο βρόχου (loop parallelism)
  - Παραλληλοποίηση σε επίπεδο συνάρτησης (function parallelism)
- Γλώσσες και εργαλεία
  - POSIX threads, MPI, OpenMP, Cilk, Cuda, Γλώσσες PGAS
- Αλληλεπίδραση με το υλικό
  - Συστήματα κοινής μνήμης
  - Συστήματα κατανεμημένης μνήμης και υβριδικά

PRAM: Η ιδανική παράλληλη μηχανή

- Η θεωρητική ανάλυση και αξιολόγηση σειριακών αλγορίθμων βασίζεται στη χρήση ενός υπολογιστικού μοντέλου:
  - Turing machine
  - Random Access Machine (RAM)
- Γιατί να μην κάνουμε το ίδιο και για τους παράλληλους αλγόριθμους;
- Ας ορίσουμε μια παράλληλη υπολογιστική μηχανή που θα μας βοηθάει να αναλύουμε και να συγκρίνουμε παράλληλους αλγορίθμους:
  - PRAM: Parallel Random Access Machine

PRAM: Η ιδανική παράλληλη μηχανή

- *n* υπολογιστικές μονάδες (*n* όσο μεγάλο απαιτεί το προς επίλυση πρόβλημα)
- Κοινή μνήμη μεγέθους m (τυπικά m ≥ n), γενικά οσοδήποτε μεγάλη
- Ομοιόμορφη πρόσβαση στη μνήμη από όλους τους επεξεργαστές σε χρόνο *Θ(1)*
- Συγχρονισμένη λειτουργία (καθολικό ρολόι)
- Δεν υπάρχει συμφόρηση



PRAM: Λειτουργία

- Οι επεξεργαστές περνούν από τις εξής φάσεις συγχρονισμένα:
  - **Φάση ανάγνωσης:** Οι επεξεργαστές διαβάζουν δεδομένα από τη μνήμη στους τοπικούς καταχωρητές. Υποστηρίζονται ταυτόχρονες αναγνώσεις. Μπορεί να εκτελούν την ίδια ή διαφορετικές εντολές (βλ. αργότερα SIMD / MIMD και μοντέλα πρόσβασης στη μνήμη)
  - **Φάση εκτέλεσης:** Οι επεξεργαστές εκτελούν μια στοιχειώδη πράξη.
  - **Φάση εγγραφής:** Οι επεξεργαστές γράφουν στη μνήμη. Για ταυτόχρονες εγγραφές βλέπε συνέχεια.

PRAM: Ταυτόχρονη πρόσβαση σε δεδομένα

- Τι συμβαίνει όταν δύο επεξεργαστές αποπειρώνται να προσπελάσουν την ίδια θέση μνήμης ταυτόχρονα;
- 4 διαφορετικές προσεγγίσεις (διαφορετικά μοντέλα):
  - Exclusive Read – Exclusive Write (EREW):
    - Κάθε θέση μνήμης μπορεί να αναγνωστεί/εγγραφεί από μόνο έναν επεξεργαστή (σε μια δεδομένη χρονική στιγμή)
    - Αν συμβεί ταυτόχρονη πρόσβαση το πρόγραμμα τερματίζει
    - Πρόκειται για πολύ περιοριστικό μοντέλο
  - Concurrent Read – Exclusive Write (CREW):
    - Επιτρέπεται ταυτόχρονη ανάγνωση αλλά όχι εγγραφή
  - Exclusive Read – Concurrent Write (ERCW):
    - Δεν παρουσιάζει θεωρητικό ή πρακτικό ενδιαφέρον
  - Concurrent Read – Concurrent Write (CRCW)
    - Ισχυρή μηχανή που επιτρέπει ταυτόχρονες αναγνώσεις / εγγραφές
    - Ταυτόχρονες εγγραφές: (common, arbitrary, priority, reduction)

PRAM: μετρικές πολυπλοκότητας

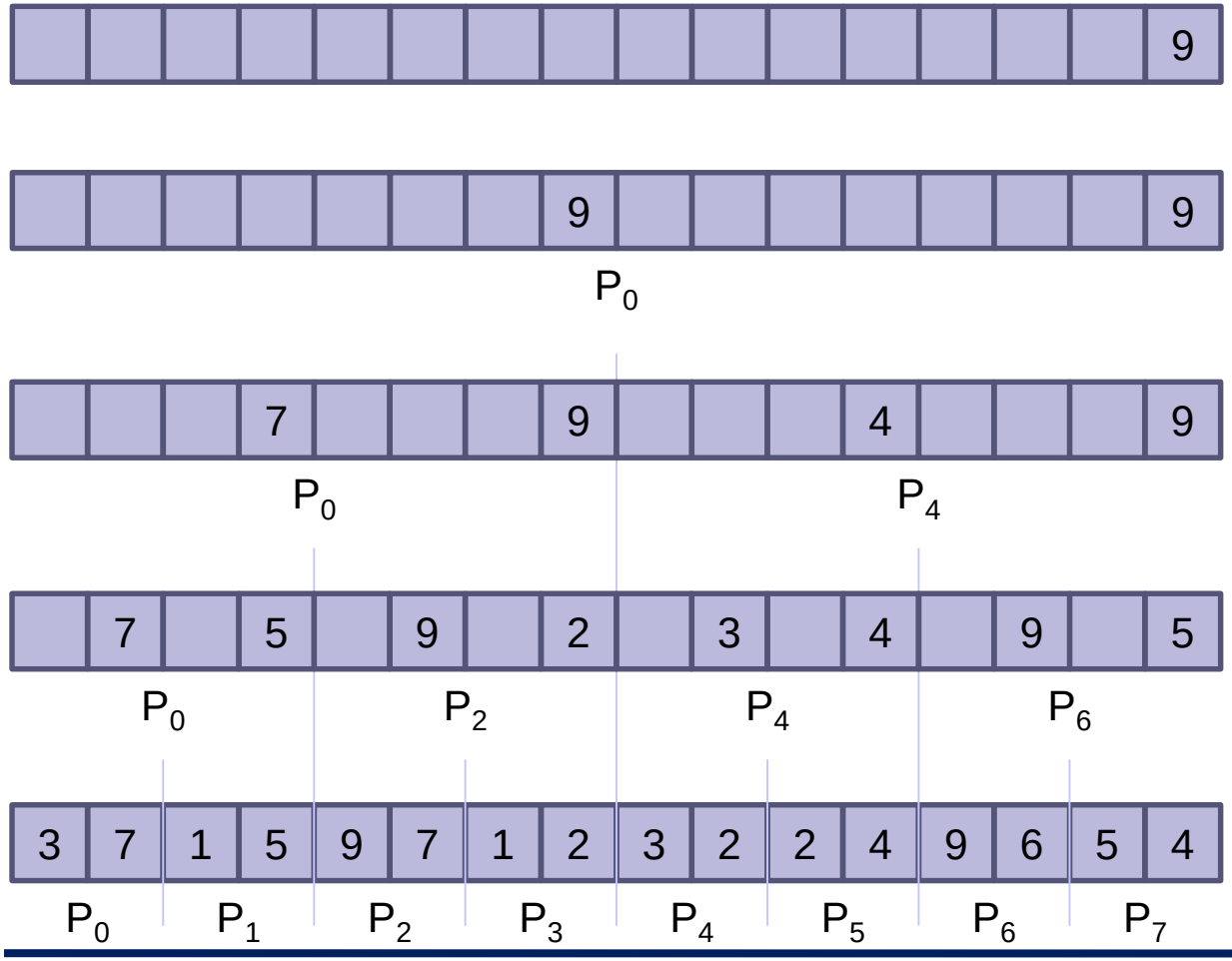
- Χρονικά βήματα (**time** cost) – όπως στη σειριακή περίπτωση
- Χώρος μνήμης (**space** cost) – όπως στη σειριακή περίπτωση
- **Αριθμός επεξεργαστών** που χρησιμοποιούνται (hardware cost)
- **Τύπος PRAM** μηχανής που απαιτείται (EREW, CREW, CRCW – hardware cost)
- Συνολικά βήματα (**work** – energy cost)

Τα tradeoffs είναι πιο σύνθετα!

Γενικά επιθυμούμε να ελαχιστοποιήσουμε το χρόνο εκτέλεσης, κρατώντας τον αριθμό των επεξεργαστών μικρό και ιδανικά να έχουμε τον ίδιο αριθμό συνολικών βημάτων με τον καλύτερο σειριακό αλγόριθμο

PRAM Παράδειγμα: Εύρεση μέγιστου

- Είσοδος 2n, μοντέλο **EREW**
- Τυπικό δενδρικό μοτίβο (π.χ. χρησιμοποιείται και για γενικευμένη πράξη reduction)



n επεξεργαστές (P_i)

Time complexity: O(log n)

Work: n + n/2 + n/4 +… = 2n-1 = O(n)

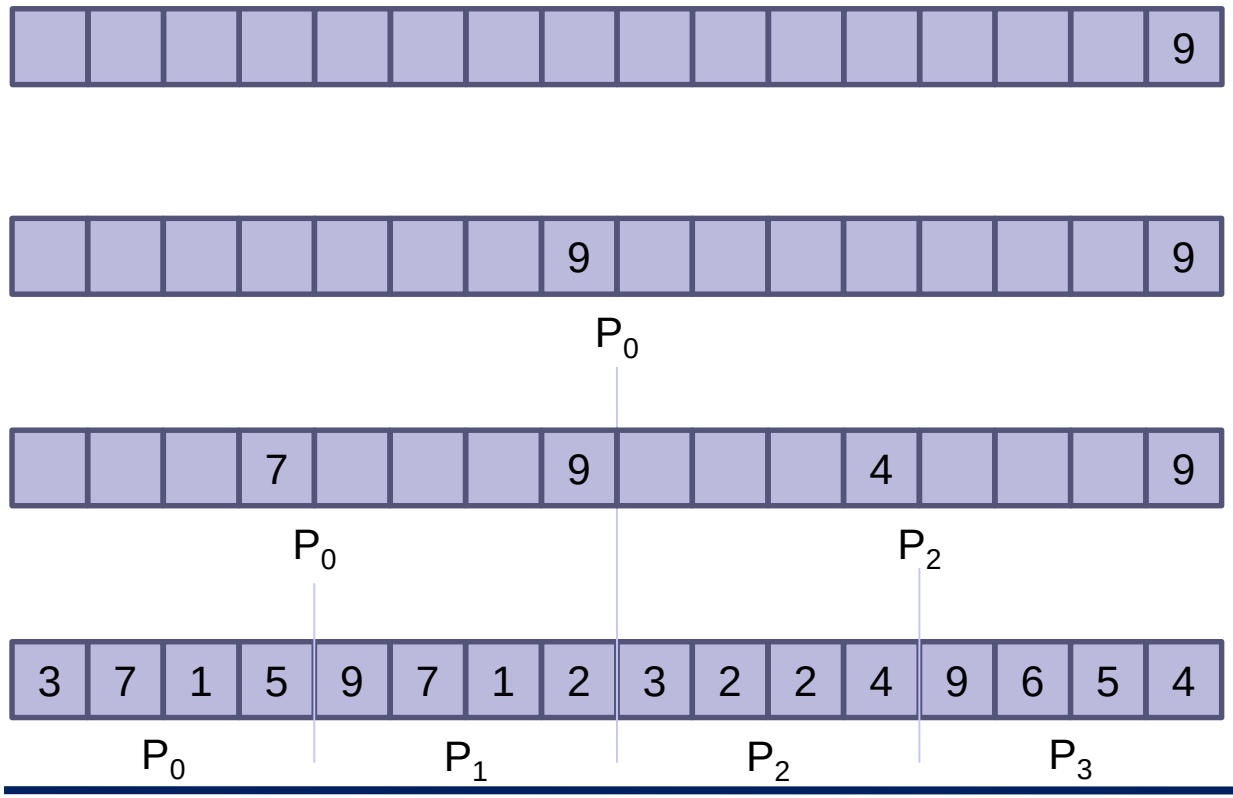
PRAM Παράδειγμα: Εύρεση μέγιστου

- Είσοδος 2n, μοντέλο **EREW**
- Τυπικό δενδρικό μοτίβο (π.χ. χρησιμοποιείται και για γενικευμένη πράξη reduction)

n/log(n) επεξεργαστές (P_i)

Time complexity:
O(n/(n/logn) + O(log n) =
O(log n)

Work: = O(n)



PRAM Παράδειγμα: Εύρεση μέγιστου

- Είσοδος n A[0, n – 1], μοντέλο **CRCW**
- n^2 επεξεργαστές, P_{ij}, $0 \le i, j, \le n - 1$

n^2 επεξεργαστές (P_i)

Time complexity: O(1)

Work: = $O(n^2)$

```
FAST-MAX(A)
   for i = 0 to n-1 in parallel
      do m[i] = true
   for i = 0 to n-1 and j = 0 to n-1 in parallel
      do if A[i] < A[j] m[i] = false
   for i = 0 to n-1 in parallel do
     if m[i] = true then max = A[i]
```

		A[j]					
		5	6	9	2	9	m
A[i]	5	F	T	T	F	T	F
	6	F	F	T	F	T	F
	9	F	F	F	F	F	T
	2	T	T	T	F	T	F
	9	F	F	F	F	F	T

max=9

PRAM: Ζητήματα υλοποίησης και χρήσης

- Η πλατφόρμα PRAM κάνει κάποιες σοβαρές απλουστεύσεις που την απομακρύνουν από τις πραγματικές μηχανές:
  - Απεριόριστος αριθμός επεξεργαστών (όσους έχει ανάγκη το πρόγραμμα)
  - Καθολικό ρολόι
  - Ταυτόχρονη πρόσβαση στη μνήμη για όλους τους επεξεργαστές
  - Ομοιόμορφος χρόνος πρόσβασης στη μνήμη
- Για να συνδυαστούν τα παραπάνω απαιτείται ένα τεράστιας πολυπλοκότητας και κόστους δίκτυο διασύνδεσης με διακόπτες και μνήμες πολλών θυρών (multi-port). Τέτοια δίκτυα διασύνδεσης με την παρούσα τεχνολογία είναι ρεαλιστικά για λίγες δεκάδες επεξεργαστές

PRAM: Ζητήματα υλοποίησης και χρήσης

- Η πλατφόρμα PRAM κάνει κάποιες σοβαρές απλουστεύσεις που την απομακρύνουν από τις πραγματικές μηχανές:
  - ~~Απεριόριστος αριθμός επεξεργαστών (όσους έχει ανάγκη το πρόγραμμα)~~
  - **Σε πραγματικά συστήματα:** Πεπερασμένος και μικρός (για κοινή μνήμη)
    - Δεν λειτουργούν αλγόριθμοι που υποθέτουν ταυτόχρονη πρόσβαση από n επεξεργαστές (π.χ. CRCW – min/max reduction)
  - ~~Καθολικό ρολόι~~
  - **Σε πραγματικά συστήματα:** Κάθε επεξεργαστής έχει το δικό του ρολόι
    - Κίνδυνος race conditions που αντιμετωπίζεται με σχήματα συγχρονισμού με σημαντικό κόστος
  - ~~Ταυτόχρονη πρόσβαση στη μνήμη για όλους τους επεξεργαστές~~
  - **Σε πραγματικά συστήματα:** Οι προσβάσεις σειριοποιούνται
    - Υπάρχει κόστος από τη συμφόρηση ακόμα και για λίγους επεξεργαστές
  - ~~Ομοιόμορφος χρόνος πρόσβασης στη μνήμη~~
  - **Σε πραγματικά συστήματα:** Υπάρχουν κρυφές μνήμες
    - Δύο προσβάσεις στη μνήμη μπορεί να διαφέρουν κατά δύο τάξεις μεγέθους στο κόστος τους

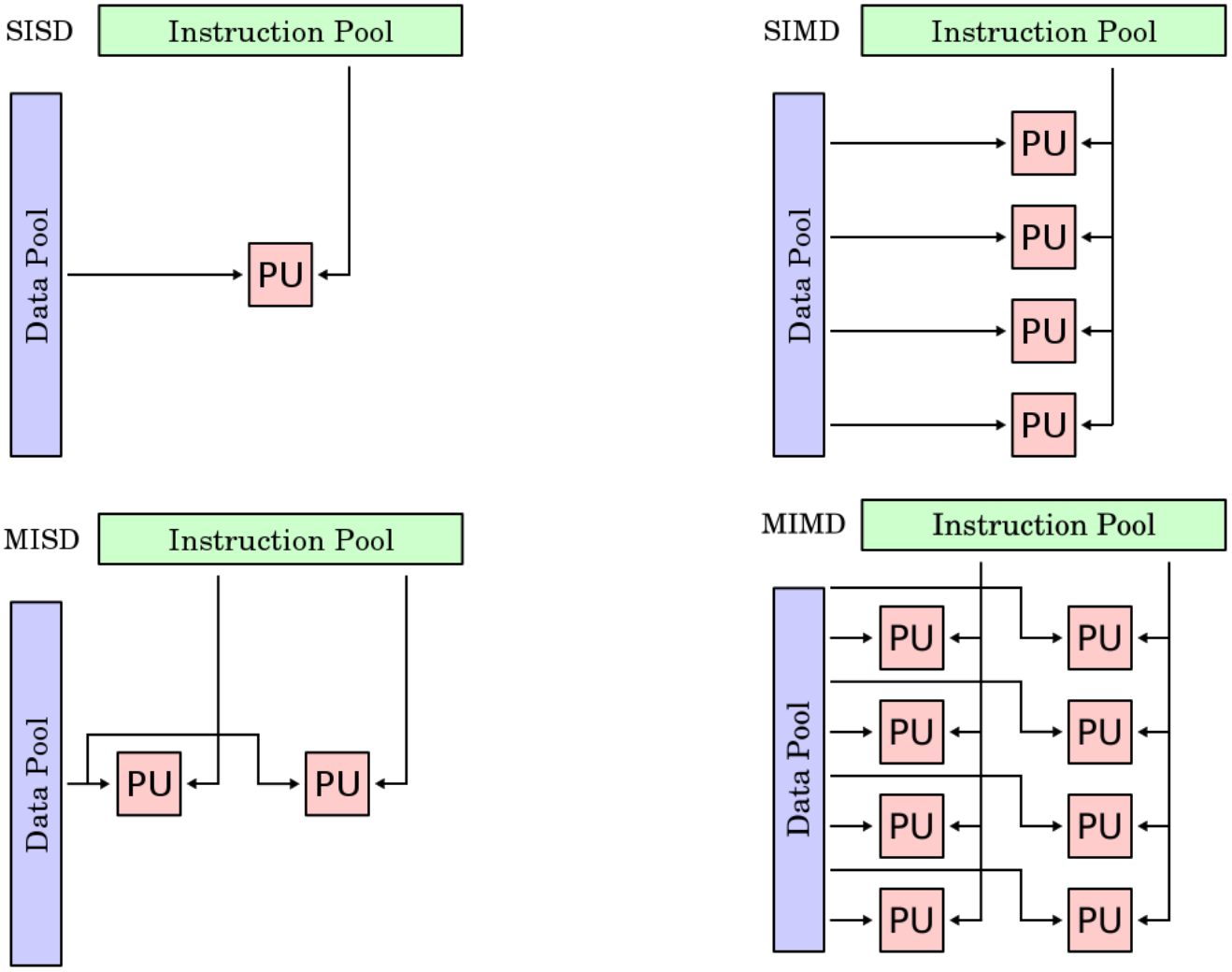
PRAM: Ζητήματα υλοποίησης και χρήσης

Συμπέρασμα: Το μοντέλο PRAM έχει θεωρητικό ενδιαφέρον, χρησιμοποιείται για την **ανάλυση και κατανόηση του εγγενούς παραλληλισμού των προβλημάτων** αλλά **δεν μπορεί να χρησιμοποιηθεί για να περιγράψει με ακρίβεια τη συμπεριφορά των προγραμμάτων σε πραγματικά συστήματα**

Flynn's taxonomy

- SISD
  - Single Instruction Single Data
- SIMD
  - Single Instruction Multiple Data
- MISD
  - Multiple Instruction Single Data
- MIMD
  - Multiple Instruction Multiple Data

Flynn's taxonomy

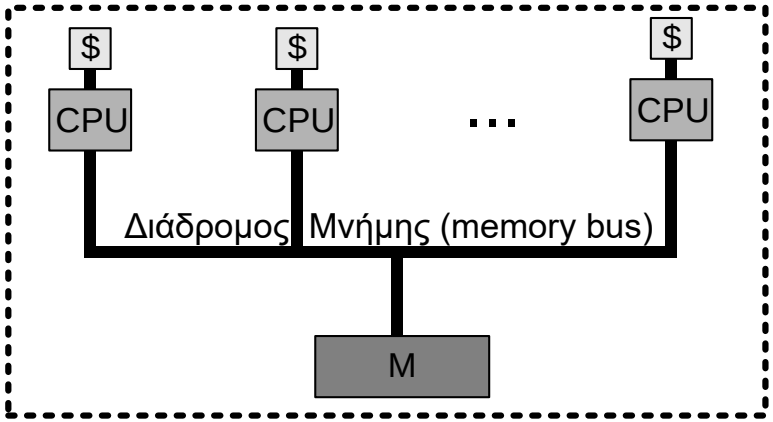


Παράλληλες αρχιτεκτονικές: Οργάνωση μνήμης

- Κοινής μνήμης (shared memory)
  - UMA (Uniform memory access): Χρόνος προσπέλασης ανεξάρτητος του επεξεργαστή και της θέσης μνήμης
  - NUMA (Non-uniform memory access): Χρόνος προσπέλασης εξαρτάται από τον επεξεργαστή και τη θέση μνήμης
  - cc-NUMA (cache-coherent NUMA): NUMA με συνάφεια κρυφής μνήμης
- Κατανεμημένης μνήμης (distributed memory)
- Υβριδική

Αρχιτεκτονική κοινής μνήμης

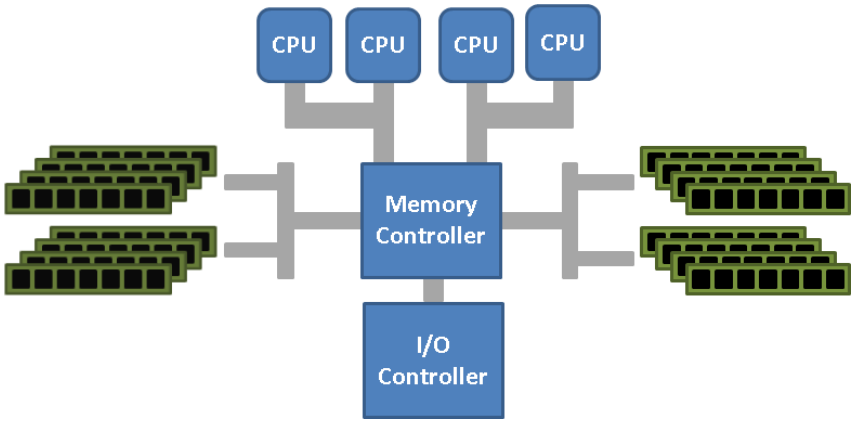
- Οι επεξεργαστές έχουν κοινή μνήμη
- Η πρόσβαση σε όλα τα δεδομένα γίνονται με εντολές ανάγνωσης και εγγραφής στη μνήμη (load / store)
- Τυπικά τα συστήματα διαθέτουν ατομικές εντολές που διευκολύνουν το συγχρονισμό (TAS, CAS)
- Κάθε επεξεργαστής διαθέτει τοπική ιεραρχία κρυφών μνημών
- Απαιτείται υλοποίηση πρωτοκόλλου συνάφειας μνήμης για να διατηρηθεί η συνάφεια των δεδομένων στην κρυφή μνήμη.
- Συνήθως η διασύνδεση γίνεται μέσω διαδρόμου μνήμης (memory bus)
  - Αλλά και πιο εξελιγμένα δίκτυα διασύνδεσης
- Ομοιόμορφη ή μη-ομοιόμορφη προσπέλαση στη μνήμη (Uniform Memory Access – UMA, Non-uniform Memory Access – NUMA)
- Η κοινή μνήμη διευκολύνει τον παράλληλο προγραμματισμό (αλλά μπορεί να δημιουργήσει σοβαρά προβλήματα λόγω race conditions!)
- Δύσκολα κλιμακώσιμη αρχιτεκτονική – τυπικά μέχρι λίγες δεκάδες κόμβους (δεν κλιμακώνει το δίκτυο διασύνδεσης)



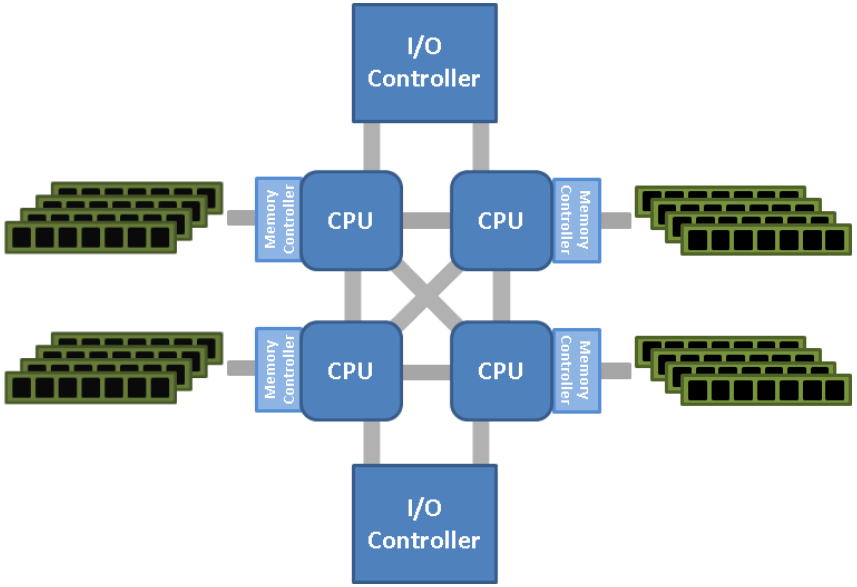
SMP vs NUMA

Symmetric Multiprocessing (SMP)

Uniform Memory Access (UMA)



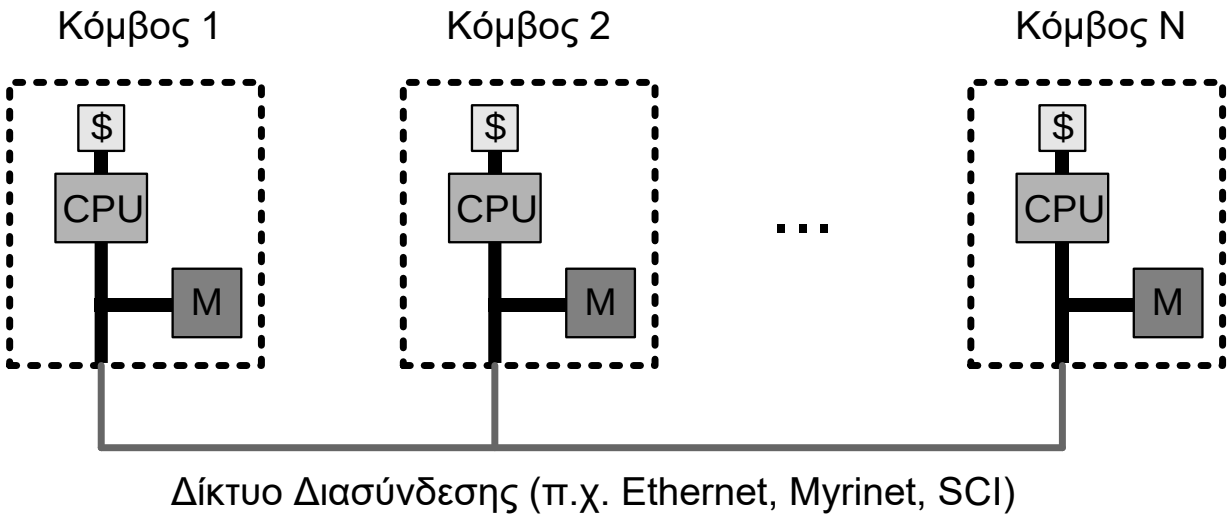
Non-Uniform Memory Access (NUMA)



Images taken from:
http://www.sqlskills.com/blogs/jonathan/understanding-non-uniform-memory-accessarchitectures-numa/

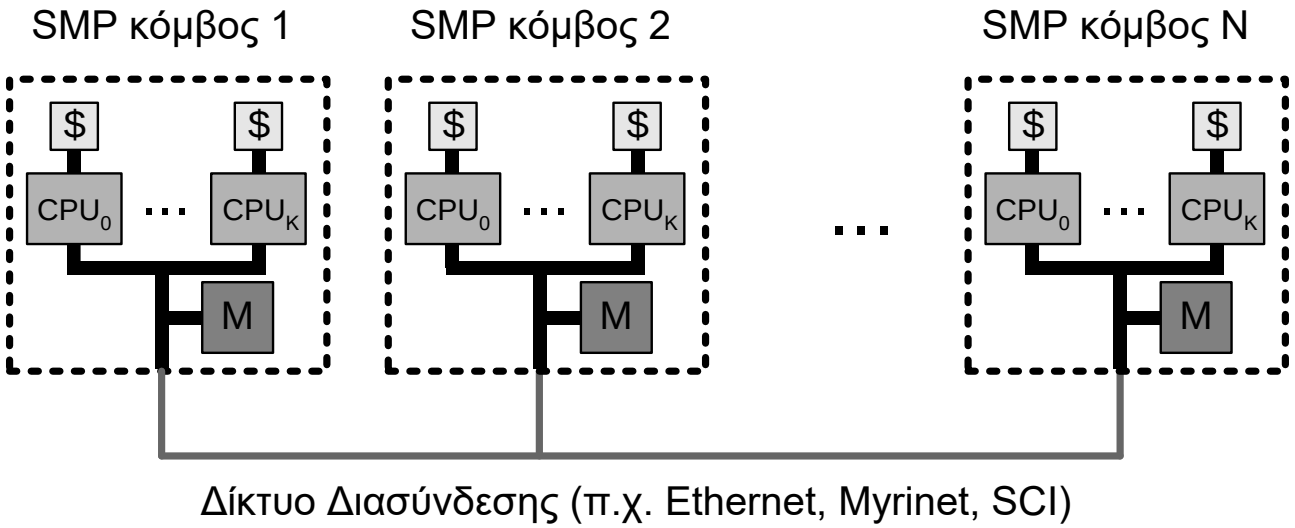
Αρχιτεκτονική κατανεμημένης μνήμης

- Κάθε επεξεργαστής έχει τη δική του τοπική μνήμη και ιεραρχία τοπικών μνημών
- Διασυνδέεται με τους υπόλοιπους επεξεργαστές μέσω δικτύου διασύνδεσης
- Η πρόσβαση σε δεδομένα που βρίσκονται σε απομακρυσμένους κόμβους γίνεται ρητά μέσω κλήσεων επικοινωνίας, ανταλλαγής μηνυμάτων (send / receive) ή μέσω συνεννόησης και των δύο πλευρών για πρόσβαση στην απομακρυσμένη μνήμη (παρεμβαίνει το ΛΣ)
- Η κατανεμημένη μνήμη δυσκολεύει τον προγραμματισμό γιατί ο προγραμματιστής απαιτείται να σχεδιάσει και να υλοποιήσει την πρόσβαση σε διακριτές μνήμες (κατακερματισμένος – fragmented προγραμματισμός)
- Η αρχιτεκτονική κλιμακώνει σε χιλιάδες υπολογιστικούς κόμβους



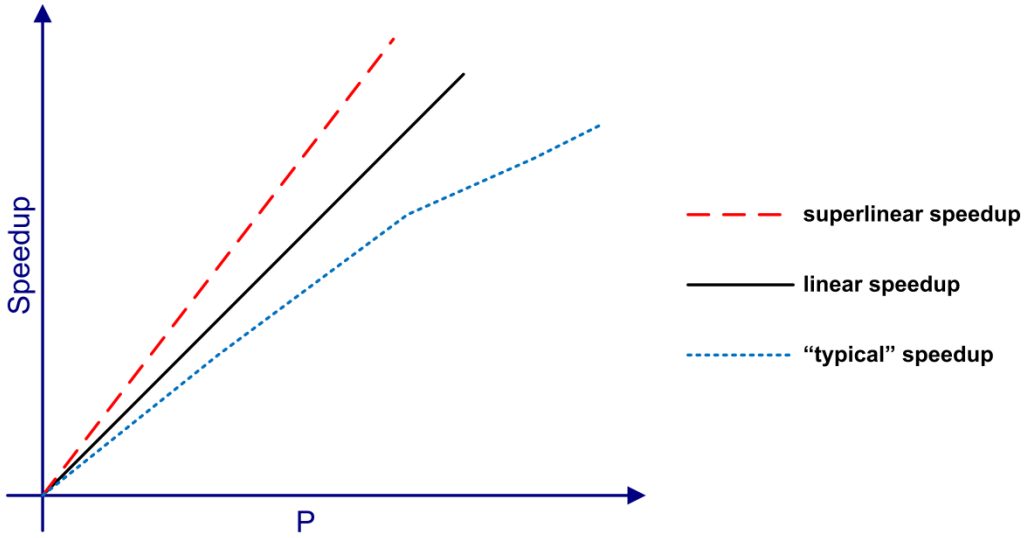
Υβριδική αρχιτεκτονική

- Συνδυάζει τις δύο παραπάνω αρχιτεκτονικές: κόμβοι με αρχιτεκτονική κοινής μνήμης διασυνδέονται με ένα δίκτυο διασύνδεσης σε αρχιτεκτονική κατανεμημένης μνήμης
- Τυπική αρχιτεκτονική των σύγχρονων συστοιχιών-υπερυπολογιστών, data centers και υποδομών cloud



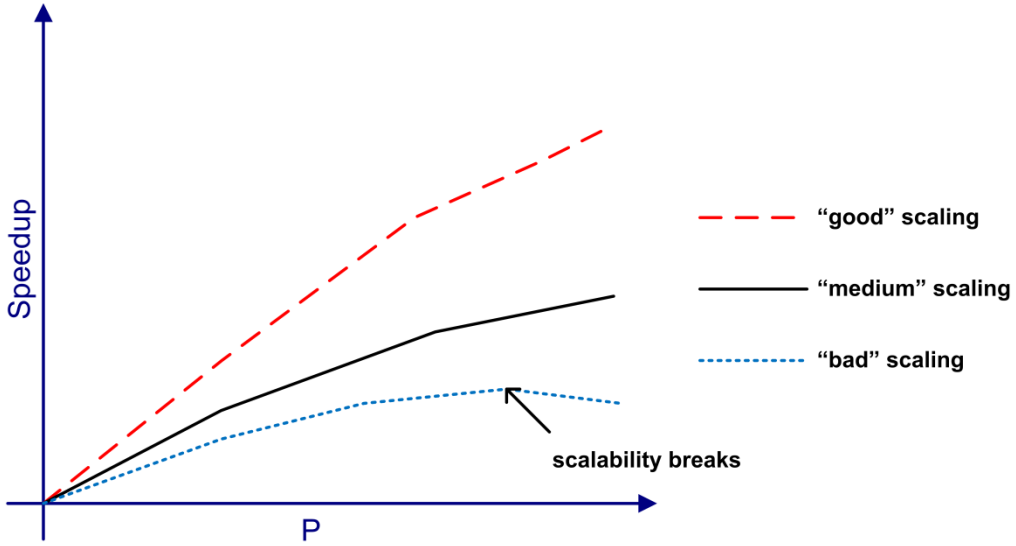
Μετρικές αξιολόγησης επίδοσης: Επιτάχυνση (Speedup)

- Χρόνος καλύτερου σειριακού αλγορίθμου: $\mathbf{T_s}$
- Χρόνος παράλληλου αλγορίθμου: $\mathbf{T_p}$
- **Επιτάχυνση** (speedup) $\mathbf{S = T_s / T_p}$
  - Δείχνει πόσες φορές πιο γρήγορο είναι το παράλληλο πρόγραμμα από το σειριακό
  - **linear** (perfect) **speedup** σε p επεξεργαστές: p
  - Τυπικά: $\mathbf{S \leq p}$
  - Αν $\mathbf{S > p}$ → **superlinear speedup** (πρέπει να το ερμηνεύσουμε προσεκτικά αν προκύψει στις μετρήσεις μας)



Αποδοτικότητα (efficiency) και κλιμακωσιμότητα (scalability)

- **Αποδοτικότητα** (efficiency) **E = S / p**
  - Δείχνει πόσο επιτυχημένη είναι η παραλληλοποίηση – τι ποσοστό του χρόνου κάθε επεξεργαστής κάνει χρήσιμη δουλειά
  - Τυπικά **E ≤ 1**
- Η **κλιμακωσιμότητα (scalability)** εκφράζει ποιοτικά την ικανότητα ενός προγράμματος (συστήματος) να βελτιώνει την επίδοσή του με την προσθήκη επιπλέον επεξεργαστών (πόρων)
  - Strong scaling: Κρατάμε το συνολικό μέγεθος του προβλήματος σταθερό
  - Weak scaling: Κρατάμε το μέγεθος ανά επεξεργαστή σταθερό



Ο (αμείλικτος) νόμος του Amdahl

- Χρόνος **καλύτερου** σειριακού αλγορίθμου: $\mathbf{T_s}$
- **f** το κλάσμα του χρόνου ενός σειριακού προγράμματος που δεν παραλληλοποιείται
- Νόμος του Amdahl:

$$T_p = fT_s + \frac{(1-f)T_s}{p}$$

$$S = \frac{T_s}{T_p} = \frac{1}{f + \frac{(1-f)}{p}}$$

- Έστω ότι ένα πρόγραμμα μπορεί να μοιραστεί σε 5 παράλληλες εργασίες εκ των οποίων η μία απαιτεί διπλάσιο χρόνο από τις υπόλοιπες 4
- f = 1/6
- Αν p = 5 (αναθέτουμε κάθε μία από τις παράλληλες εργασίες σε έναν επεξεργαστή) → **S = 3, E=0,6**
- Συνέπειες του νόμου του Amdahl:
  - Παραλληλοποιούμε (και γενικά βελτιστοποιούμε) τμήματα του κώδικα που καταλαμβάνουν το μεγαλύτερο ποσοστό του χρόνου εκτέλεσης
  - Αναζητούμε παραλληλία **παντού**! (**π.χ. 1% σειριακός κώδικας → μέγιστο speedup 100!**)

Γιατί δεν κλιμακώνει το πρόγραμμά μου;

Ο Νόμος του Amdahl δίνει ένα θεωρητικό άνω όριο της κλιμάκωσης σε ιδανικές συνθήκες παραλληλοποίησης (αγνοεί διάφορα επιπρόσθετα κόστη που προκύπτουν από την υλοποίηση και την πλατφόρμα). Γενικά τα αίτια της μη κλιμάκωσης:

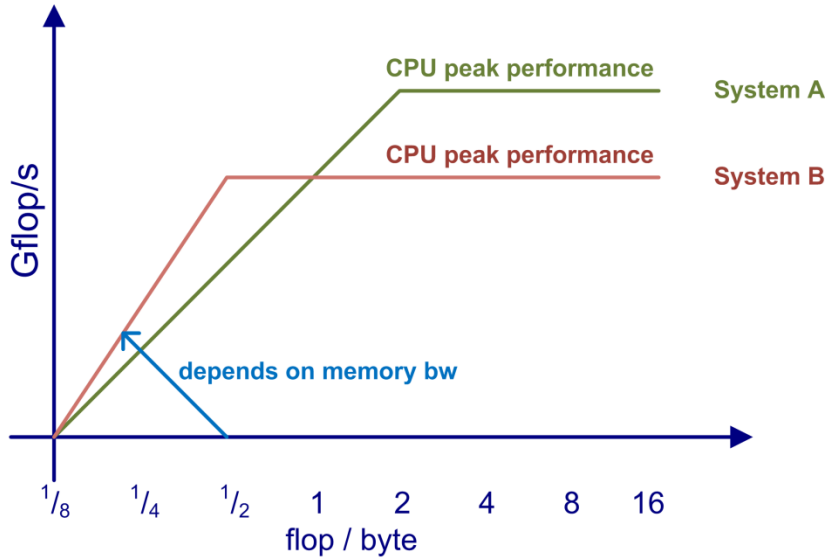
1. Δεν έχει παραλληλοποιηθεί το κατάλληλο τμήμα κώδικα (υπάρχει μεγάλο σειριακό μέρος) – Amdahl's law
2. Υπάρχει ανισοκατανομή φορτίου (load imbalance) – Amdahl's law!
3. Κόστος συγχρονισμού / επικοινωνίας
   - Μπορεί να οδηγήσει ακόμα και σε αύξηση του χρόνου εκτέλεσης
4. Κόστος από την παραλληλοποίηση (επιπλέον εργασία στον παράλληλο αλγόριθμο)
   - Επιπλέον εργασία λόγω παράλληλου αλγορίθμου (βλ. work στην PRAM)
   - Επιπλέον λειτουργίες κατανομής εργασίας (στατικά ή δυναμικά)
   - Διαχείριση οντοτήτων εκτέλεσης (νήματα, διεργασίες, κλπ)
5. Συμφόρηση στο διάδρομο μνήμης (για αρχιτεκτονικές κοινής μνήμης) – Amdahl's law!!

Μοντελοποίηση επίδοσης

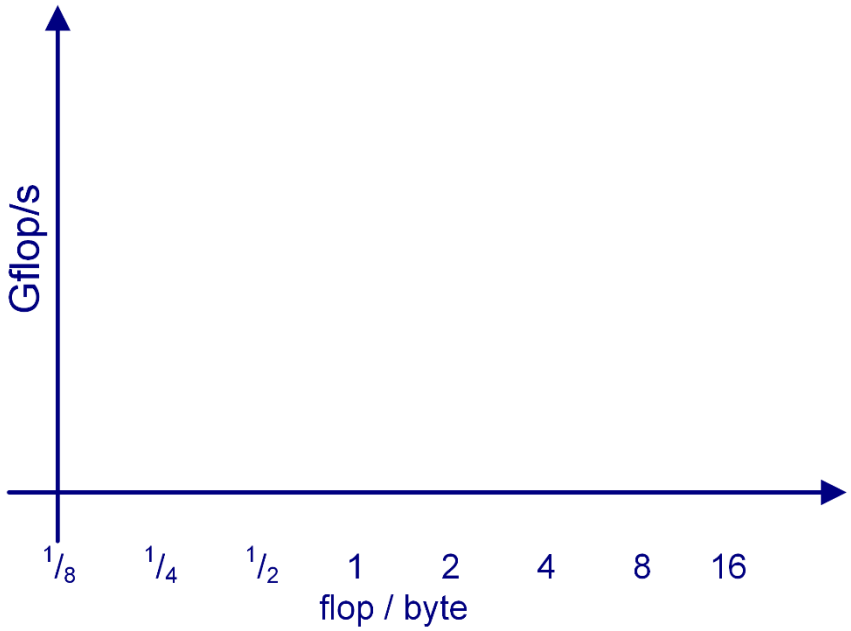
- Η μοντελοποίηση της επίδοσης είναι ιδιαίτερα δύσκολη σε ένα πολύπλοκο σύστημα όπως είναι η εκτέλεση μιας εφαρμογής σε μία παράλληλη πλατφόρμα
- Θέλουμε το μοντέλο να είναι **απλό** και **ακριβές**
- Συνήθως καθοδηγεί σχεδιαστικές επιλογές (π.χ. συγκρίνει διαφορετικές στρατηγικές)
- Συχνά αρκεί να έχουμε άνω και κάτω όρια στην επίδοση
- Ο παράλληλος χρόνος εκτέλεσης καταναλώνεται σε:
  - Χρόνο υπολογισμού (T_{comp})
  - Χρόνο επικοινωνίας (T_{comm})
  - Άεργο χρόνο (T_{idle})
- Παράλληλος χρόνος εκτέλεσης:
  - Ο χρόνος από την έναρξη της πρώτης διεργασίας μέχρι τη λήξη της τελευταίας
  - Για κάθε διεργασία *i*, $T_i = T_{comp} + T_{comm} + T_{idle}$
  - Για το συνολικό, παράλληλο χρόνο;

Χρόνος υπολογισμών

- Κλασικό μοντέλο πρόβλεψης:
  - **T_{comp} = ops * CPU speed (sec / op)** [Η CPU μπορεί να έχει πολλούς υπολογιστικούς πυρήνες)
  - Θεωρεί ότι η CPU μπορεί να τροφοδοτείται με δεδομένα από το υποσύστημα μνήμης
- Μοντέλο roofline
  - Λαμβάνει υπόψη:
    - το bandwidth (bw σε byte/sec) του υποσυστήματος μνήμης
    - Το Operational Intensity (OI σε ops/byte) του υπολογιστικού πυρήνα
  - **T_{comp} = ops * max (CPU speed, 1/(OI * bw))**
  - Συσχετίζει την εφαρμογή με την αρχιτεκτονική

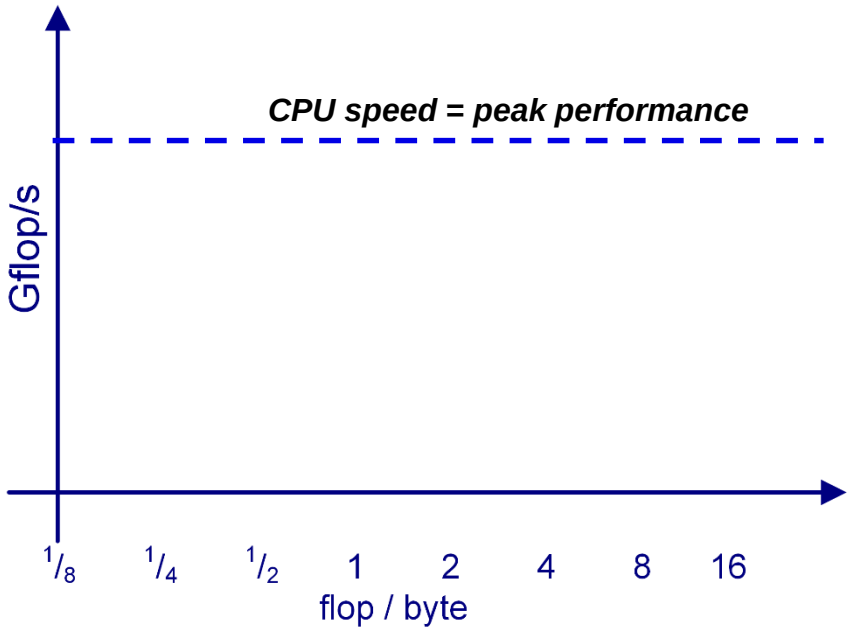


Καταστρώνοντας το μοντέλο roofline



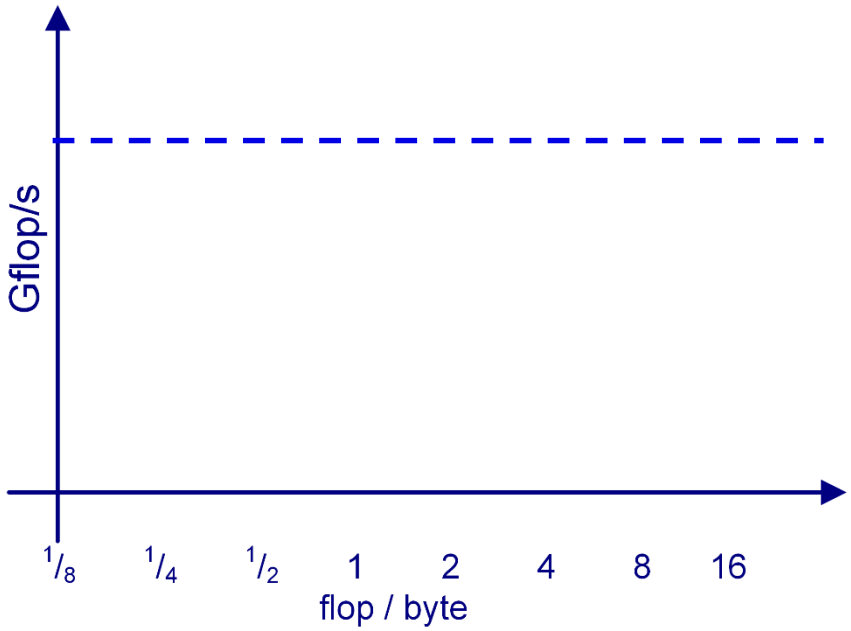
Καταστρώνοντας το μοντέλο roofline

Βήμα 1: Μέτρηση της μέγιστης υπολογιστικής ισχύος (**CPU speed**) του επεξεργαστή (Gflops/sec) με κατάλληλο μετροπρόγραμμα ή αναλυτικά



Καταστρώνοντας το μοντέλο roofline

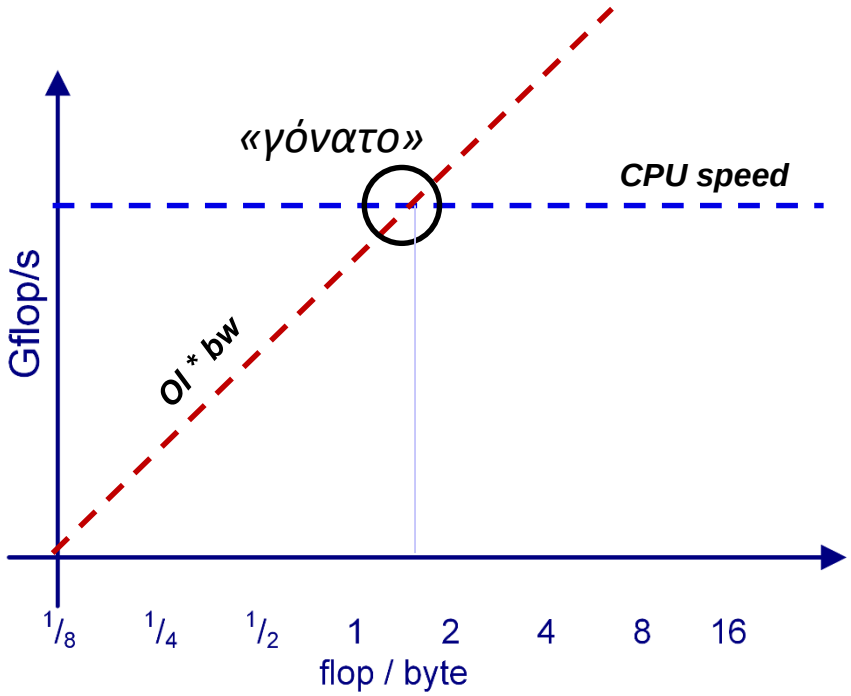
Βήμα 2: Προσδιορισμός bandwidth (**bw**) διαύλου μνήμης με κατάλληλο μετροπρόγραμμα



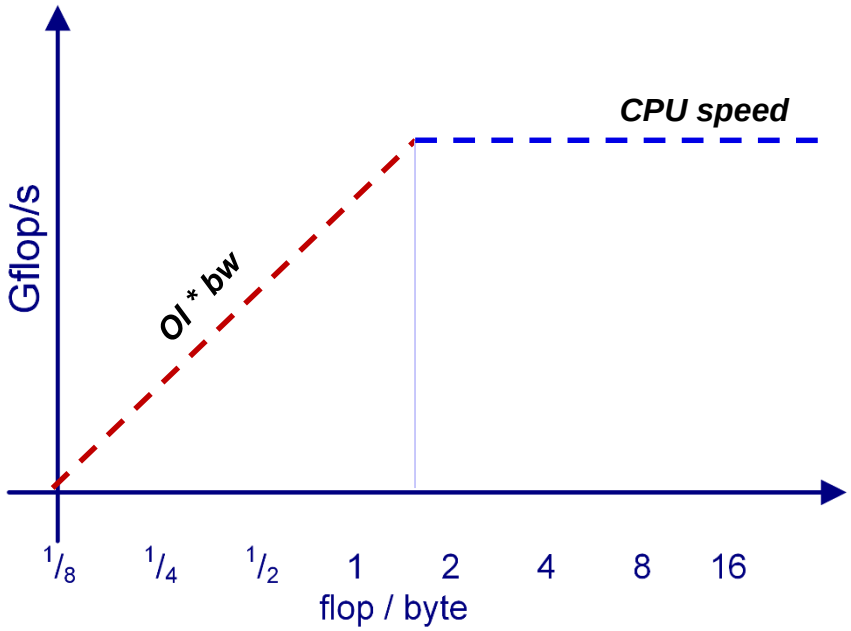
Καταστρώνοντας το μοντέλο roofline

Βήμα 3: Προσδιορισμός «γονάτου» **OI** στο διάγραμμα (σημείο τομής των δύο ευθειών του roofline) από τη σχέση:

OI * bw = CPU speed => OI = CPU speed / bw



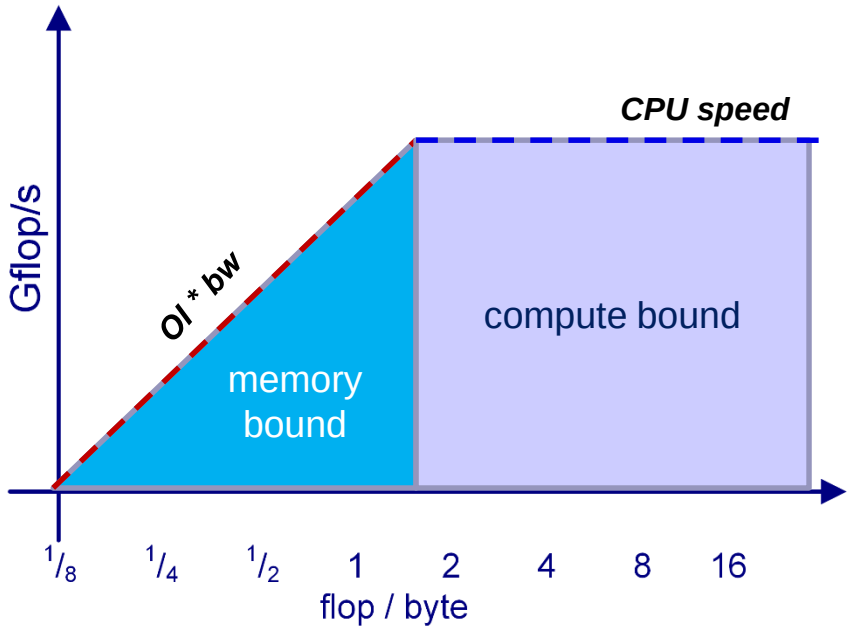
Καταστρώνοντας το μοντέλο roofline



Χρήση του μοντέλου roofline

- Μπορεί να μας ενημερώσει αν ένας αλγόριθμος (μαζί με το μέγεθος των δεδομένων του) σε μία αρχιτεκτονική είναι περιορισμένος από τη μνήμη (memory bound) ή από την υπολογιστική ισχύ του επεξεργαστή (cpu bound), και έτσι να οδηγήσει τις σχετικές βελτιστοποιήσεις
- Μπορεί να μας δώσει αδρές εκτιμήσεις για το χρόνο υπολογισμών (βλ. παράδειγμα στη συνέχεια)
- Μπορεί να μας βοηθήσει να εκτιμήσουμε αν μία υλοποίηση έχει ακόμα περιθώρια βελτιστοποίησης ή όχι (βλ. παράδειγμα στη συνέχεια)

Μοντέλο roofline: cpu bound vs memory bound



Μοντέλο roofline: cpu bound vs memory bound

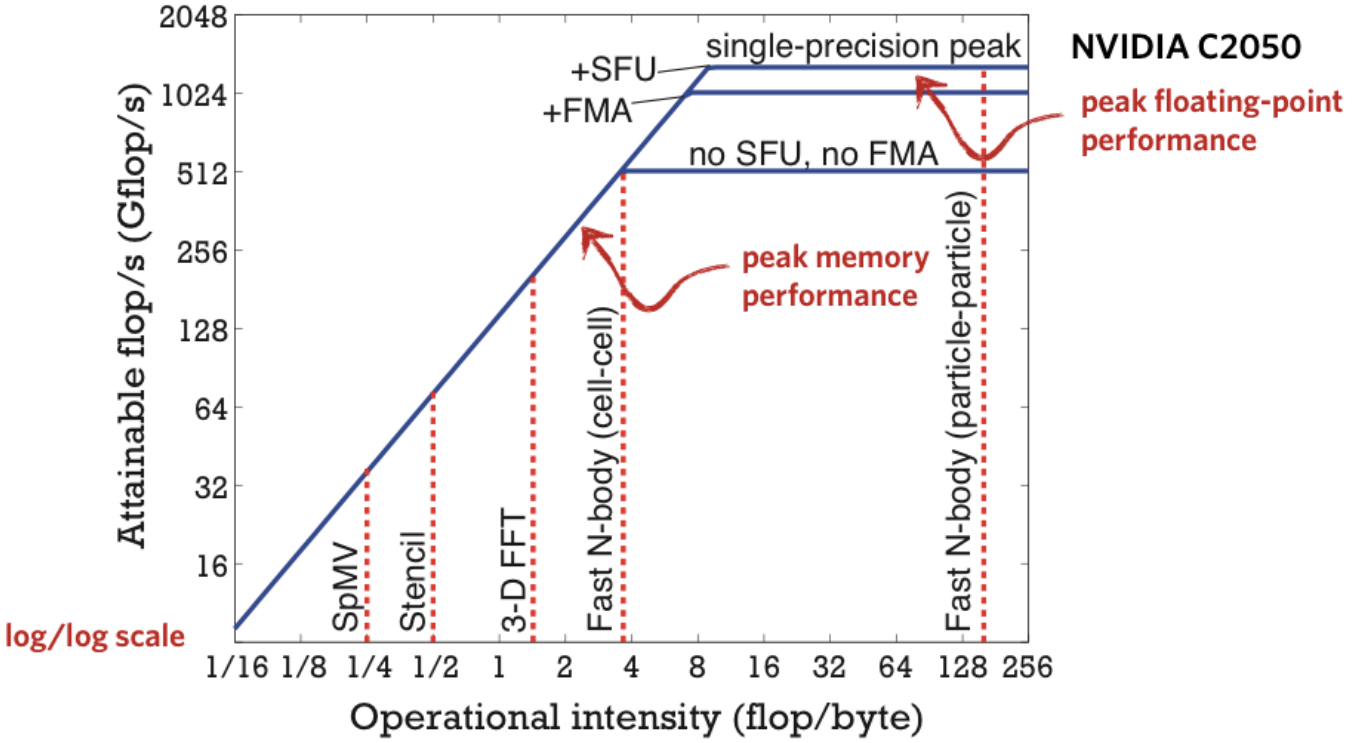


Image taken from http://lorenabarba.com/news/new-paper-published-cise-journal

Μοντέλο roofline: cpu bound vs memory bound

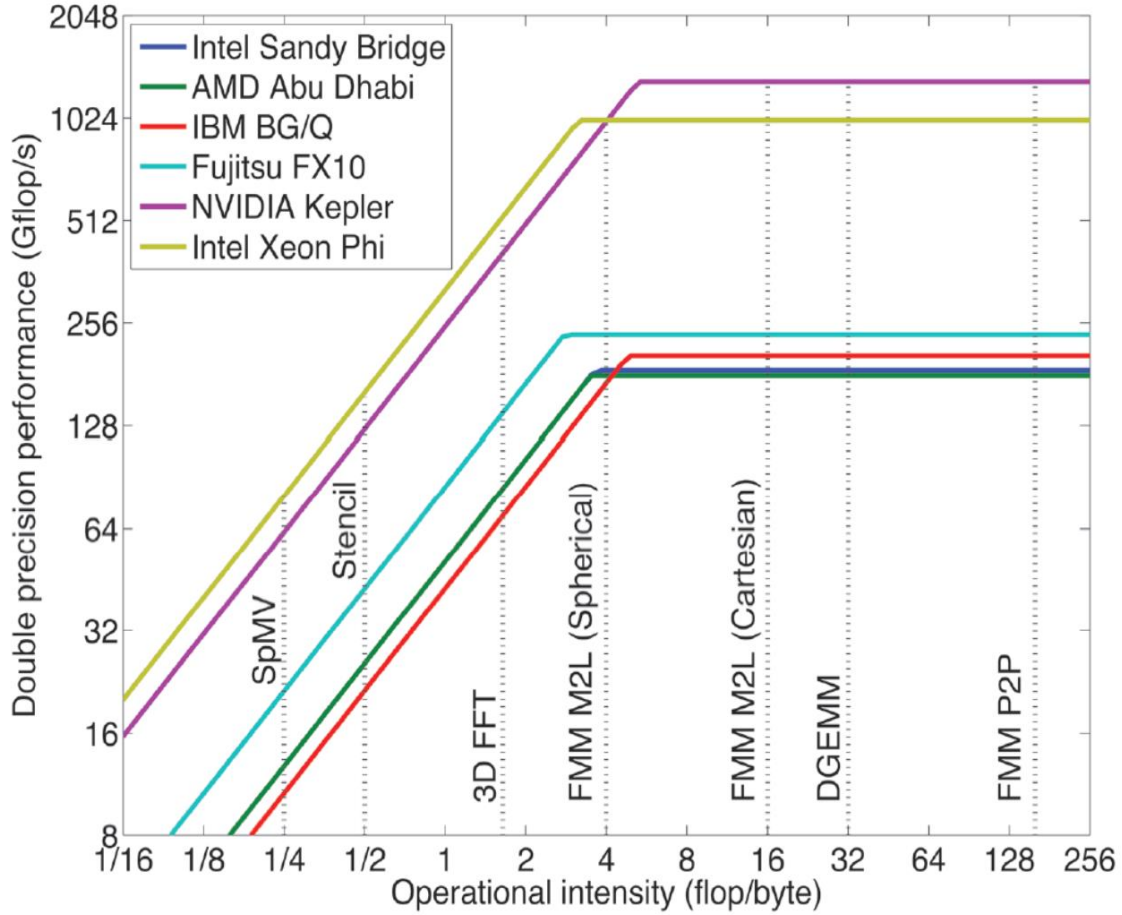


Image taken from: Leopold Grinberg,
https://www.dam.brown.edu/people/lgrinb/APMA2821/Lectures_2015/APMA2821H-L_roof_line_model.pdf

Εκτίμηση χρόνου υπολογισμών
Παράδειγμα: LU decomposition

```
for(k = 0; k < N-1; k++)
   for(i = k+1; i < N; i++){
      L[i] = A[i][k] / A[k][k];
      for(j = k+1; j < N; j++)
         A[i][j] = A[i][j] - L[i]*A[k][j];
   }
```

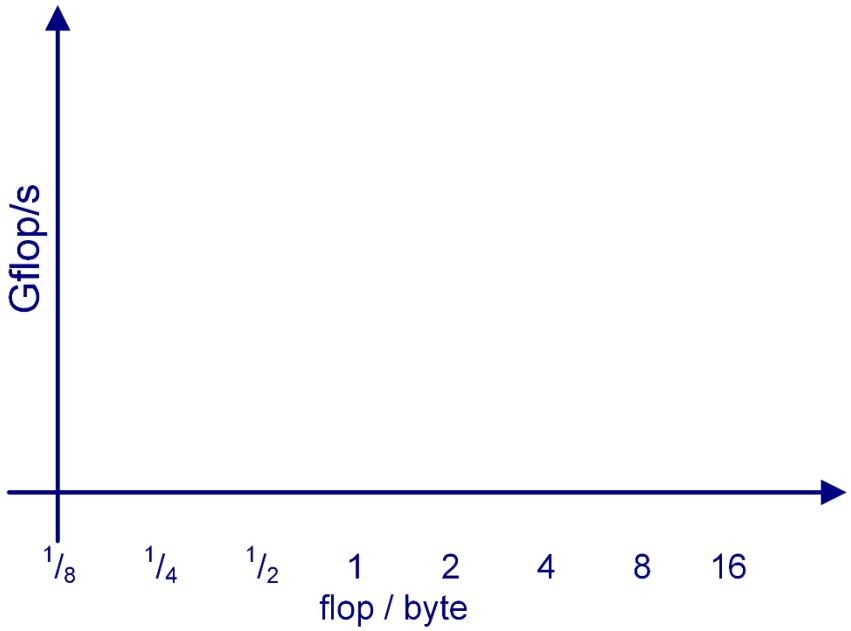
- Operations = N^3 / 3 πολλαπλασιασμοί, N^3/3 προσθέσεις
  - Μπορούμε να αγνοήσουμε τις προσθέσεις
- Operational intensity (ops/byte):
  - Μας αφορούν δεδομένα που έρχονται από την κύρια μνήμη (όχι από την cache)
  - Μπορούμε να αγνοήσουμε το διάνυσμα L
  - Μέγεθος A = $8*N^2$ (θεωρούμε διπλή ακρίβεια, 8 bytes για κάθε στοιχείο του A)
  - Αν ο A χωράει στην cache: **OI =** (N^3/ 3) / ($8*N^2$) = **N/24 muls/byte**
  - Αν ο A >> cache size: Ας θεωρήσουμε ότι σε κάθε βήμα k ο A φορτώνεται από τη μνήμη (γιατί δεν είναι τυπικά σωστό αυτό?). **OI = 1/24 muls/byte**

Εκτίμηση χρόνου υπολογισμών Παράδειγμα: LU decomposition

- Έστω επεξεργαστής με:
  - CPU speed $5*10^8$ muls/sec ή $2*10^{-9}$ sec/mul (500MFlops)
  - Bandwidth διαδρόμου μνήμης: 4GB/sec = $4*10^9$ bytes /sec
  - 4MB L2 cache
- N = 100 (ο Α χωράει στην cache, OI = 4,16 muls/byte)
  - **Παραδοσιακό μοντέλο:**
    - $\mathbf{T_{comp}}$ = ops * CPU speed = $0.33*10^6$ muls * $2*10^{-9}$ sec/mul = **0.66msec**
  - **Μοντέλο roofline:**
    - $\mathbf{T_{comp}}$ = ops * min (CPU speed, 1/(OI * memory bandwidth)) =
      $0.33*10^6$ muls * max ($2*10^{-9}$ sec/mul , (4,16 muls/byte * $4*10^9$ bytes/sec)$^{-1}$)=
      $0.33*10^6$ muls * max ($2*10^{-9}$ sec/mul , $6*10^{-11}$ sec/mul) = **0.66msec**
- N = 2000 (ο Α δεν χωράει στην cache, OI = 0,042 muls/byte)
  - **Παραδοσιακό μοντέλο:**
    - $\mathbf{T_{comp}}$ = ops * CPU speed = $2.66*10^9$ muls * $2*10^{-9}$ sec/mul = **5.33sec**
  - **Μοντέλο roofline:**
    - $\mathbf{T_{comp}}$ = ops * max (CPU speed, 1/(OI * memory bandwidth)) =
      $2.66*10^9$ muls * max ($2*10^{-9}$ sec/mul , (0,042 muls/byte* $4*10^9$ bytes /sec)$^{-1}$) =
      $2.66*10^9$ muls * max ($2*10^{-9}$ sec/mul , $5,95*10^{-9}$) = **15.9sec**

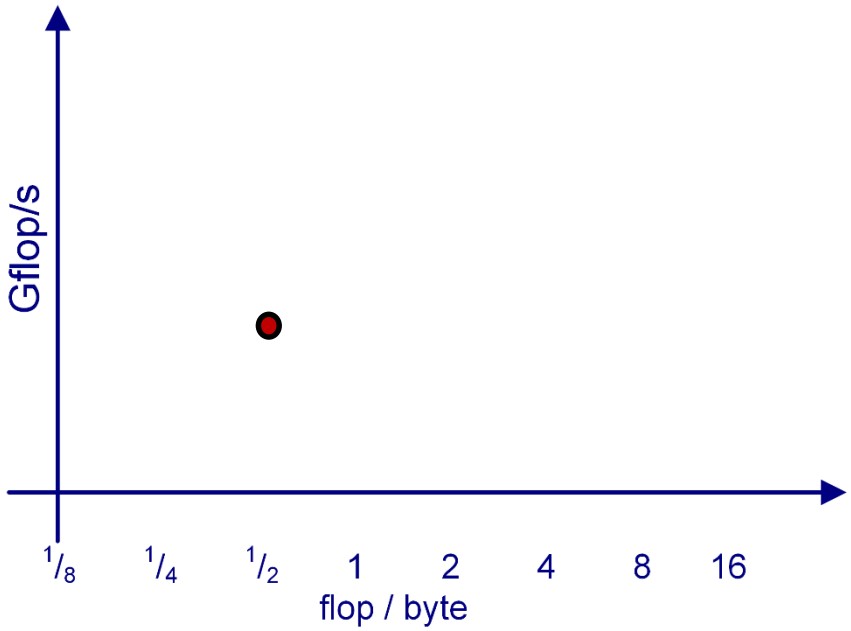
Είναι βελτιστοποιημένος ο κώδικάς μου;

- Υπολόγισε πόσα operations (FLOP) κάνει ο αλγόριθμός σου
- Μέτρα πόσο χρόνο διαρκεί η εκτέλεσή του και πόσα bytes μεταφέρονται από τη μνήμη (οι σύγχρονοι επεξεργαστές περιλαμβάνουν “performance counters” που παρέχουν αυτές τις μετρήσεις)
- Υπολόγισε το operational intensity και τα FLOPS (ή FLOP/s)



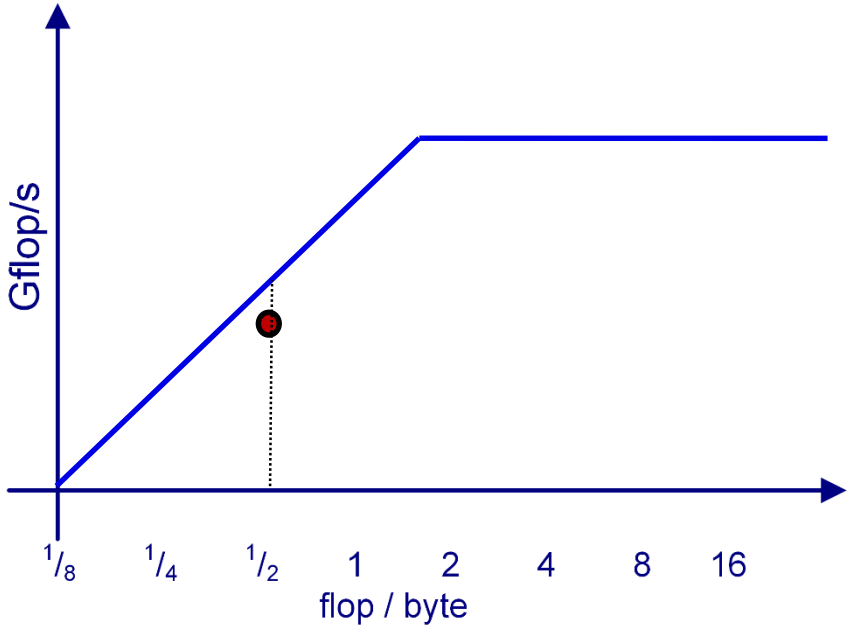
Είναι βελτιστοποιημένος ο κώδικάς μου;

- Υπολόγισε πόσα operations (FLOP) κάνει ο αλγόριθμός σου
- Μέτρα πόσο χρόνο διαρκεί η εκτέλεσή του και πόσα bytes μεταφέρονται από τη μνήμη (οι σύγχρονοι επεξεργαστές περιλαμβάνουν “performance counters” που παρέχουν αυτές τις μετρήσεις)
- Υπολόγισε το operational intensity και τα FLOPS (ή FLOP/s)



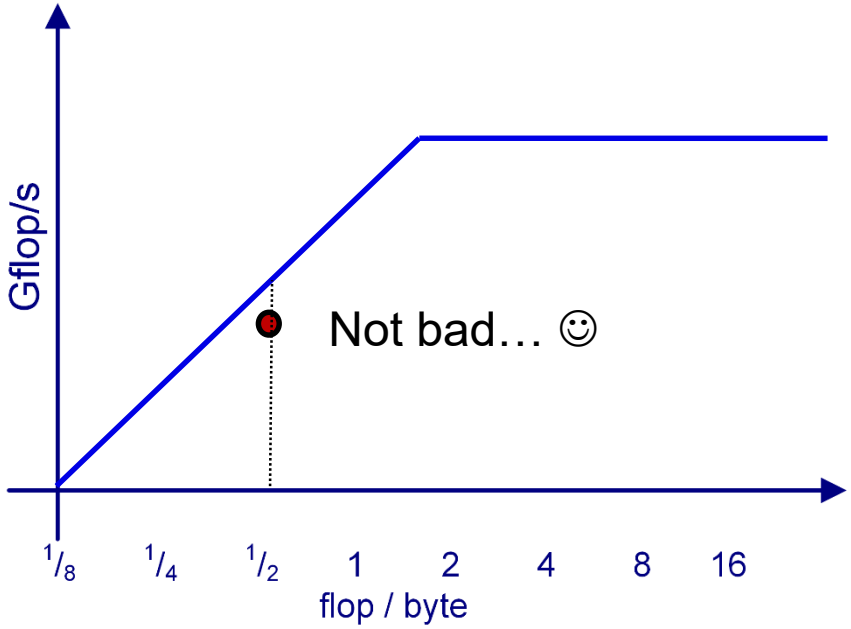
Είναι βελτιστοποιημένος ο κώδικάς μου;

- Υπολόγισε πόσα operations (FLOP) κάνει ο αλγόριθμός σου
- Μέτρα πόσο χρόνο διαρκεί η εκτέλεσή του και πόσα bytes μεταφέρονται από τη μνήμη (οι σύγχρονοι επεξεργαστές περιλαμβάνουν “performance counters” που παρέχουν αυτές τις μετρήσεις
- Υπολόγισε το operational intensity και τα FLOPS (ή FLOP/s)
- Πήγαινε στο roofline



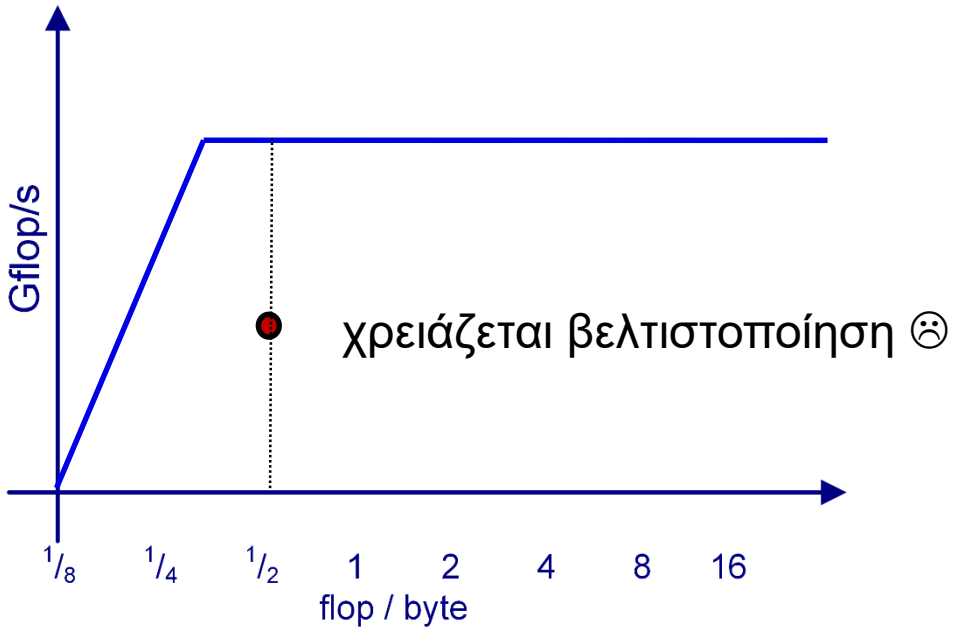
Είναι βελτιστοποιημένος ο κώδικάς μου;

- Υπολόγισε πόσα operations (FLOP) κάνει ο αλγόριθμός σου
- Μέτρα πόσο χρόνο διαρκεί η εκτέλεσή του και πόσα bytes μεταφέρονται από τη μνήμη (οι σύγχρονοι επεξεργαστές περιλαμβάνουν “performance counters” που παρέχουν αυτές τις μετρήσεις
- Υπολόγισε το operational intensity και τα FLOPS (ή FLOP/s)
- Πήγαινε στο roofline



Είναι βελτιστοποιημένος ο κώδικάς μου;

- Υπολόγισε πόσα operations (FLOP) κάνει ο αλγόριθμός σου
- Μέτρα πόσο χρόνο διαρκεί η εκτέλεσή του και πόσα bytes μεταφέρονται από τη μνήμη (οι σύγχρονοι επεξεργαστές περιλαμβάνουν “performance counters” που παρέχουν αυτές τις μετρήσεις
- Υπολόγισε το operational intensity και τα FLOPS (ή FLOP/s)
- Πήγαινε στο roofline

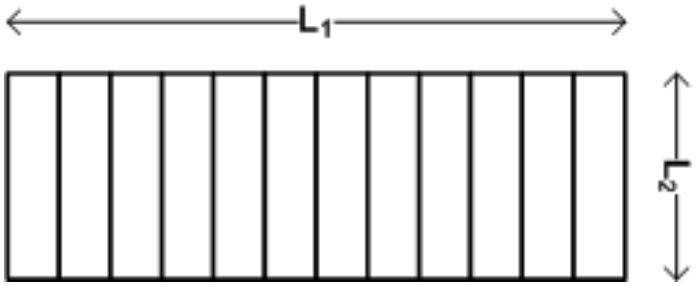


Χρόνος επικοινωνίας

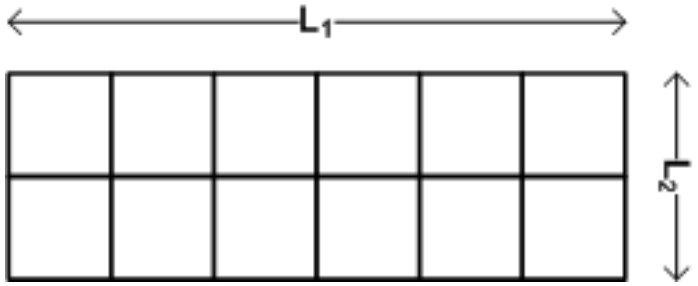
- Χρόνος επικοινωνίας (point to point):
  - Startup time (t_s)
  - Per-hop time (t_h)
  - Per word transfer time (t_w)
  - $T_{comm} = t_s + lt_h + mt_w$ (l = number of hops, m = message size)
  - Το lt_h είναι συνήθως μικρό και μπορεί να ενσωματωθεί στο t_s, άρα:
  - **$T_{comm} = t_s + mt_w$**
- Χρόνος επικοινωνίας (broadcast σε P κόμβους)
  - P διαφορετικά μηνύματα
    - $T_{comm} = P(t_s + mt_w)$
  - Δενδρική υλοποίηση:
    - $T_{comm} = \log_2 P(t_s + mt_w)$

Παράδειγμα: Nearest-neighbor communication

- 2 διάστατο υπολογιστικό χωρίο $L_1 * L_2$
- Κάθε επεξεργαστής επικοινωνεί με τους γειτονικούς του και ανταλλάζει τις οριακές επιφάνειες
- Ποια τοπολογία επεξεργαστών ελαχιστοποιεί το κόστος επικοινωνίας;



1D



2D

Παράδειγμα: Nearest-neighbor communication

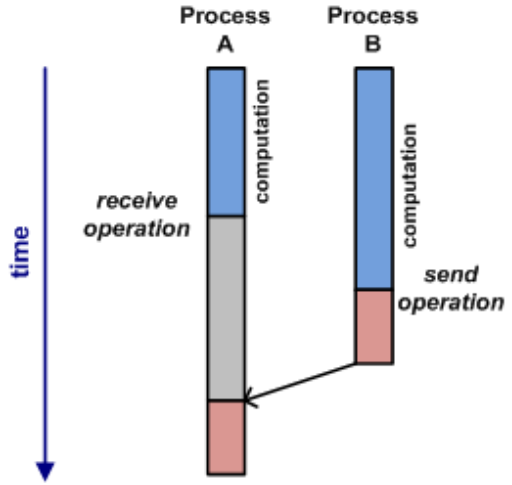
- Αριθμός επεξεργαστών: P = 100
  - 1D = 100*1
  - 2D = 10*10
- διπλή ακρίβεια (8 bytes)
- $L_1 = 10^4$, $L_2 = 10^3$
- $\mathbf{t_s} = 6*10^{-6}$ sec
- bandwidth = 500MB/sec → $\mathbf{t_w} = 1.9*10^{-9}$ sec/byte
- 1D
  - 2 γείτονες, 1 μήνυμα ανά γείτονα, 2 μηνύματα ανά διεργασία
  - Μέγεθος μηνύματος: $m = 8L_2 = 8*10^3$ bytes
  - $T_{comm} = t_s + mt_w = 2\ (6*10^{-6}$ sec + $8*10^3$ bytes * $1.9*10^{-9}$ sec/byte**) = 42,4usec**
- 2D
  - 4 μηνύματα ανά διεργασία
  - $m_1 = 8L_1/10 = 8*10^3$ bytes, $m_2 = 8L_2/10 = 8*10^2$ bytes
  - $T_{comm} = t_s + mt_w = 4\ (6*10^{-6}$ sec) + $2(8*10^3$ bytes * $1.9*10^{-9}$ sec/byte) + $2(8*10^2$ bytes * $1.9*10^{-9}$ sec/byte) = 24us + 30,4us + 3,04us = **57,4us**

Χρόνος επικοινωνίας (στον πραγματικό κόσμο)

- Συστήματα μεγάλης κλίμακας με χιλιάδες κόμβους, δεκάδες πυρήνες ανά κόμβο
- Παράμετροι που επηρεάζουν την επικοινωνία:
  - Μέγεθος μηνύματος
    - Διαφορετικό bandwidth για μικρά/μεσαία/μεγάλα μηνύματα λόγω χρήσης διαφορετικού πρωτοκόλλου δικτύου
  - Τοπικός ανταγωνισμός στο εσωτερικό του κόμβου για πρόσβαση στο δίκτυο
  - Ανταγωνισμός στο δίκτυο κορμού
    - Μπορεί να οφείλεται και σε άλλες εφαρμογές!
  - Απόσταση μεταξύ των κόμβων που επικοινωνούν
    - Εξαρτάται από τη διάθεση κόμβων από τον scheduler του συστήματος
  - Αριθμός μηνυμάτων
- Άλλα ζητήματα:
  - Επικάλυψη υπολογισμών-επικοινωνίας
  - Ανισοκατανομή φορτίου και άεργος χρόνος
  - Πώς ορίζεται ο χρόνος επικοινωνίας σε ένα παράλληλο πρόγραμμα;

Άεργος χρόνος

- Ο άεργος χρόνος μπορεί να μοντελοποιηθεί δύσκολα
- Προκύπτει συνήθως από ανισοκατανομή του φόρτου εργασίας (load imbalance)
- Μπορεί να οφείλεται σε αναμονή για δεδομένα
  - Σε αυτή την περίπτωση μπορεί να μοντελοποιηθεί σας μέρος του t_s στο χρόνο επικοινωνίας
- Συχνά δεν είναι απλό να διαχωριστεί από το χρόνο επικοινωνίας
  - Π.χ. στην παρακάτω περίπτωση 2 διεργασιών που επικοινωνούν, δεν είναι ξεκάθαρο ποιο κομμάτι του χρόνου θα πρέπει να χαρακτηριστεί ως άεργος χρόνος και ποιο ως χρόνος επικοινωνίας



Ερωτήσεις;