Parallelizing applications for the GRID

Γιώργος Γκούμας
goumas@cslab.ece.ntua.gr

Σύνοψη Παρουσίασης

- Εφαρμογές με ανάγκες για υψηλές επιδόσεις
- Αξιολόγηση επίδοσης
- Παραλληλοποίηση εφαρμογών
  - Γενικές αρχές
  - Παραδείγματα
    - Embarassingly parallel
    - Μερικές Διαφορικές Εξισώσεις
    - Εύρεση ελάχιστου μονοπατιού
    - Πράξεις με αραιούς πίνακες
  - Αποτελέσματα

“Απαιτητικές” εφαρμογές

- Εξομοιώσεις φυσικών διεργασίων
- Γραμμική άλγεβρα
- Αλγόριθμοι γράφων
- Επεξεργασία σήματος
- Βελτιστοποίηση
- Ταξινόμηση
- Μέθοδοι Monte Carlo
- Data search
- ……

Οι 7 νάνοι

- ***The Landscape of Parallel Computing Research: A View from Berkeley***
- **K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams and K. A. Yelick**
- **http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf**

1. Dense linear algebra
2. Sparse linear algebra
3. Spectral methods (frequency domain, FFT)
4. N-body methods
5. Structured grids (e.g. Red-black)
6. Unstructured grids
7. Monte Carlo

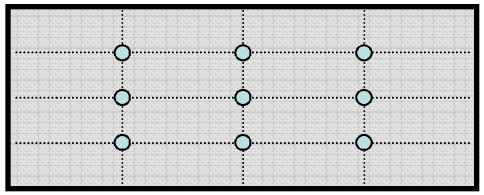
Εξομοιώσεις φυσικών διεργασιών

- Ρευστοδυναμική (Computational Fluid Dynamics - CFD)
- Φυσική της ατμόσφαιρας - Μετεωρολογία
- Διάχυση (θερμότητας, υλικών, κλπ)
- Μετάδοση Η/Μ ακτινοβολίας
- Δομική μηχανική
- Φυσική υψηλών ενεργειών
- Αστρονομία
- Βιολογία (αναδίπλωση πρωτεϊνών – protein folding)
- …

Αριθμητική επίλυση ΜΔΕ

- **Διακριτοποίηση του υπό εξέταση χωρίου (δημιουργία πλέγματος)**

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial^2 x} + \frac{\partial^2 u}{\partial^2 y}$$



- **Διακριτοποίηση του συνεχούς μοντέλου περιγραφής του προβλήματος, π.χ.:**

$$\frac{\partial u}{\partial t} = \frac{u_{xy}^{n+1} - u_{xy}^{n}}{dt} \qquad \frac{\partial^2 u}{\partial^2 x} = \frac{u_{x-1y}^{n} - 2u_{xy}^{n} + u_{x+1y}^{n}}{dx^2}$$

- **Υλοποίηση επαναληπτικής μεθόδου επίλυσης**

$$u_{xy}^{n+1} = \left(1 - 4\frac{dt}{dx^2}\right) u_{xy}^{n} + \frac{dt}{dx^2}\left(u_{x-1y}^{n} + u_{x+1y}^{n} + u_{xy-1}^{n} + u_{xy+1}^{n}\right)$$

Αριθμητική επίλυση ΜΔΕ (2)

- **Μέθοδοι διακριτοποίησης ΜΔΕ:**
  - Πεπερασμένες διαφορές (finite differences)
  - Πεπερασμένα στοιχεία (finite elements)
  - Πεπερασμένοι όγκοι (finite volumes)
- **Εν γένει οι παραπάνω διακριτοποιήσεις οδηγούν στην κατασκευή ενός μεγάλου αραιού γραμμικού συστήματος της γνωστής μορφής**

$$Au=b$$

όπου:

- **A** NxN αραιός πίνακας που εξαρτάται από τη διακριτοποίηση του χώρου
- **u** N-διάστατο διάνυσμα με τις προς υπολογισμό τιμές τις συνάρτησης u στα N σημεία του πλέγματος
- **b** N-διάστατο διάνυσμα που προκύπτει από τις αρχικές/συνοριακές τιμές του προβλήματος

Αριθμητική επίλυση ΜΔΕ (3)

- Μέθοδοι επίλυσης συστήματος Au=b:
  - Direct μέθοδοι, π.χ. LU Decomposition (A=LU).
  - Επαναληπτικές μέθοδοι: Conjugate Gradient (CG), Generalized Minimal Residual Method (GMRES).
  - Multigrid μέθοδοι (γεωμετρικές, αλγεβρικές)

Αξιολόγηση επίδοσης (παράλληλου) προγράμματος: Μετρικές

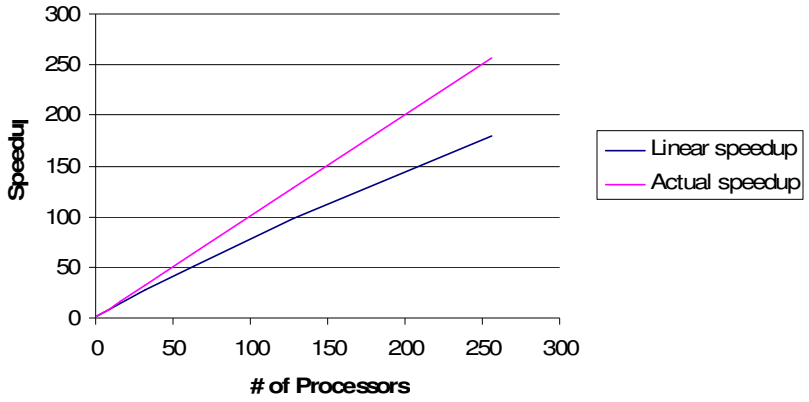
- Χρόνος εκτέλεσης σειριακού (T_S), χρόνος εκτέλεσης παράλληλου (T_P)
- Αριθμός πράξεων κινητής υποδιαστολής / δευτερόλεπτο (MFLOP/s)
- Επιτάχυνση (speedup)
- Αποδοτικότητα (parallel efficiency)

MFLOP/s

- Ο αριθμός των πράξεων κινητής υποδιαστολής είναι γενικά χαρακτηριστικό του αλγόριθμου, π.χ.:
  - Πολλαπλασιασμός διανυσμάτων: 2N-1
  - Πολλαπλασιασμός πίνακα-διάνυσμα: $2N^2$-N
  - LU decomposition: N^3
    - Επίδοση $_{(MFLOP/s)}$=MFLOP/T
- Χαρακτηρίζει την επίδοση του αλγόριθμου σε μία πλατφόρμα

Επιτάχυνση (Speedup)

- Speedup $(S)=T_S/T_P$

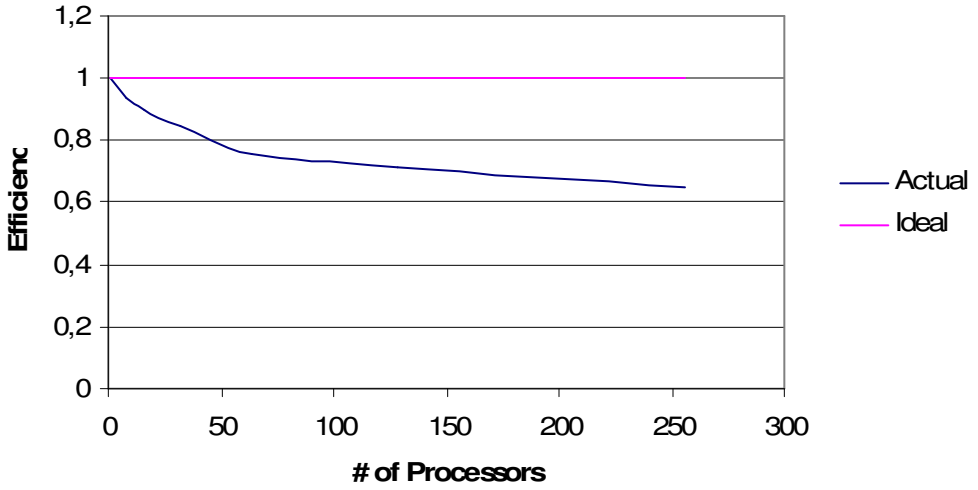


- Χαρακτηρίζει την επιτυχία παραλληλοποίησης ενός αλγόριθμου σε P επεξεργαστές
- Ο νόμος του Amdahl:

$$Speedup_{(overall)} = \frac{T_S}{T_P} = \frac{T_S}{fT_P + (1-f)\frac{T_P}{P}}$$

Efficiency

- Efficiency (E)=S/P



- Εκφράζει το ποσοστό του χρόνου που κάθε επεξεργαστής καταναλώνει σε χρήσιμη εργασία

Παραλληλοποίηση εφαρμογών

- **Στόχοι:**
  - **Μείωση χρόνου εκτέλεσης**
  - **Βελτίωση "ποιότητας" των αποτελεσμάτων (π.χ. ακρίβεια)**
  - **Επίλυση «μεγαλύτερου» προβλήματος**
- **Απαιτήσεις:**
  - **Καλή γνώση του προς επίλυση προβλήματος**
  - **Γνώση της υφιστάμενης αρχιτεκτονικής (?)**
  - **Φαντασία! ☺**

1ος (και 2ος) κανόνας για τη βελτιστοποίηση του λογισμικού

Μην το κάνεις!!!!

"premature optimization is the root of all evil" Donald Knuth

Γενική μεθοδολογία παραλληλοποίησης

- **Βήμα 1:** Κατανομή υπολογισμών
- **Βήμα 2:** Κατανομή δεδομένων
- **Βήμα 3:** Σχεδιασμός επικοινωνίας/συγχρονισμού
- **Βήμα 4:** Βελτιστοποιήσεις (επιστροφή στο Βήμα 1?)
- **Βήμα 5:** Ανάπτυξη παράλληλου κώδικα
- **Βήμα 6:** Ανάθεση διεργασιών σε επεξεργαστές

Γενική μεθοδολογία παραλληλοποίησης

Ιδιαίτερη προσοχή σε:

- Ανάδειξη όλου του εγγενούς παραλληλισμού
- Επιλογή μεταξύ “coarse-grain” και “fine-grain” παραλληλισμού
- Ομοιόμορφη κατανομή του φορτίου (load balancing)
- Ελαχιστοποίηση του κόστους επικοινωνίας/συγχρονισμού
  - Αριθμός δεδομένων επικοινωνίας
  - Αριθμός μηνυμάτων
  - Συχνότητα συγχρονισμού
- Bottlenecks και ιδιομορφίες της πλατφόρμας εκτέλεσης

Αρχιτεκτονικές και Προγραμματιστικά Μοντέλα από την οπτική του προγραμματιστή

		Αρχιτεκτονική	
		Κοινής μνήμης (shared memory)	Κατανεμημένης μνήμης (distributed memory)
Προγραμματιστικό μοντέλο	Κοινός χώρος διευθύνσεων (shared address space)	+ Ευκολία υλοποίησης + Προγραμματιστική ευκολία + Υψηλή επίδοση	+ Προγραμματιστική ευκολία - Δυσκολία υλοποίησης - Χαμηλή επίδοση
	Ανταλλαγή μηνυμάτων (message-passing)	+ Ευκολία υλοποίησης + Υψηλή επίδοση - Προγραμματιστική δυσκολία	+ Ευκολία υλοποίησης + Υψηλή επίδοση - Προγραμματιστική δυσκολία

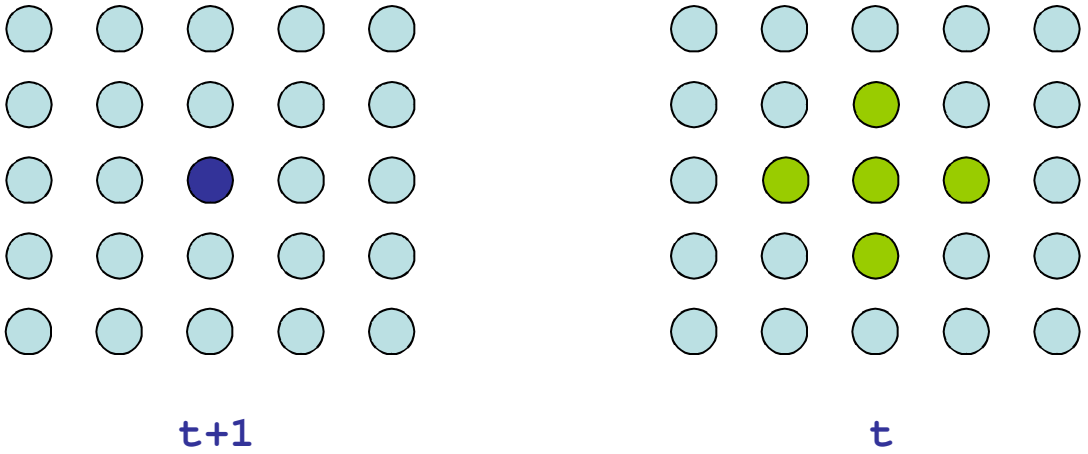
Εφαρμογές “Embarassingly Parallel”

- Μπορούν να χωριστούν σε τελείως ανεξάρτητα tasks
- Επεξεργάζονται διαφορετικά δεδομένα
- Data-parallel εφαρμογές
- Παραδείγματα:
  - Mp3 encoding N αρχείων σε P επεξεργαστές με N >> P
  - Αναζήτηση
  - Μέθοδοι Monte Carlo
- Monte carlo methods
  1. Ορισμός χώρου από εισόδους
  2. Επιλογή τυχαίας εισόδου
  3. Εκτέλεση «πειράματος»
  4. Συλλογή αποτελεσμάτων

  - Το βήμα 3 που καταναλώνει τον περισσότερο χρόνο είναι τελείως ανεξάρτητο
  - Προσοχή όμως στη γεννήτρια τυχαίων αριθμών σε συστήματα κοινής μνήμης!!!!

Εξίσωση διάχυσης στις 2 διαστάσεις

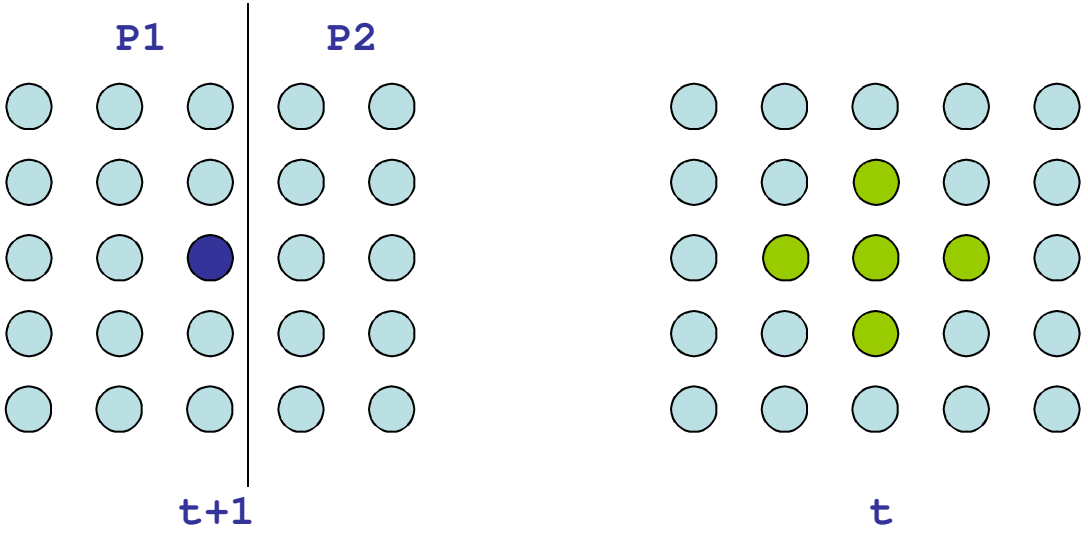
```
1.  while (!converged){
2.    for x=1 to X-1 do
3.      for y=1 to Y-1 do
4.        A[t+1][x][y] = f(A[t][x][y], A[t][x-1][y],
                           A[t][x+1][y], A[t][x][y-1], A[t][x][y]+1);
5.    t++;}
```



Nearest neighbor update (stencil computation) dwarf#5

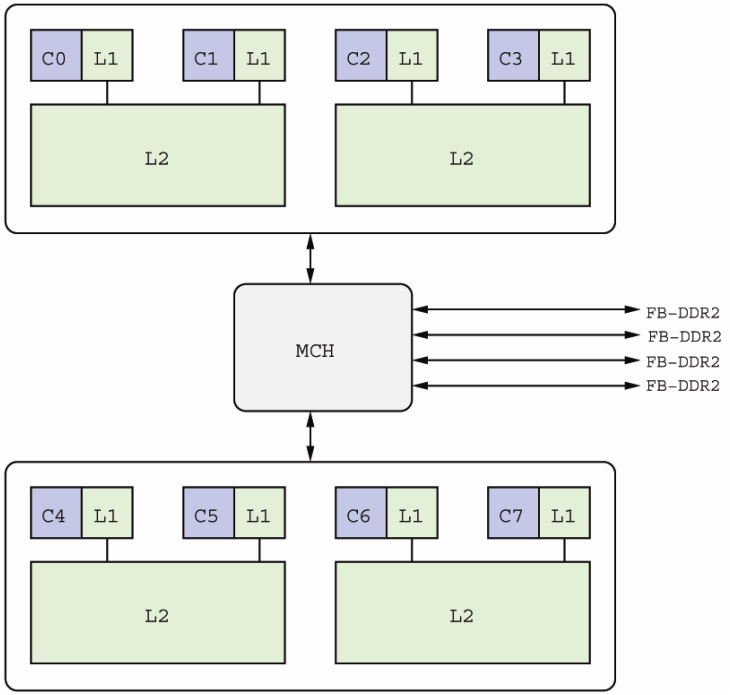
Παραλληλοποίηση σε κοινό χώρο διευθύνσεων

```
1.  while (!converged){    //NOT PARALLEL
2.  #pragma omp parallel for private (x,y)
3.    for x=1 to X-1 do    //PARALLEL AND PARALLELIZED
4.      for y=1 to Y-1 do //PARALLEL BUT NOT PARALLELIZED
5.        A[t+1][x][y] = f(A[t][x][y], A[t][x-1][y],
                           A[t][x+1][y], A[t][x][y-1], A[t][x][y]+1);
6.    t++;}
```

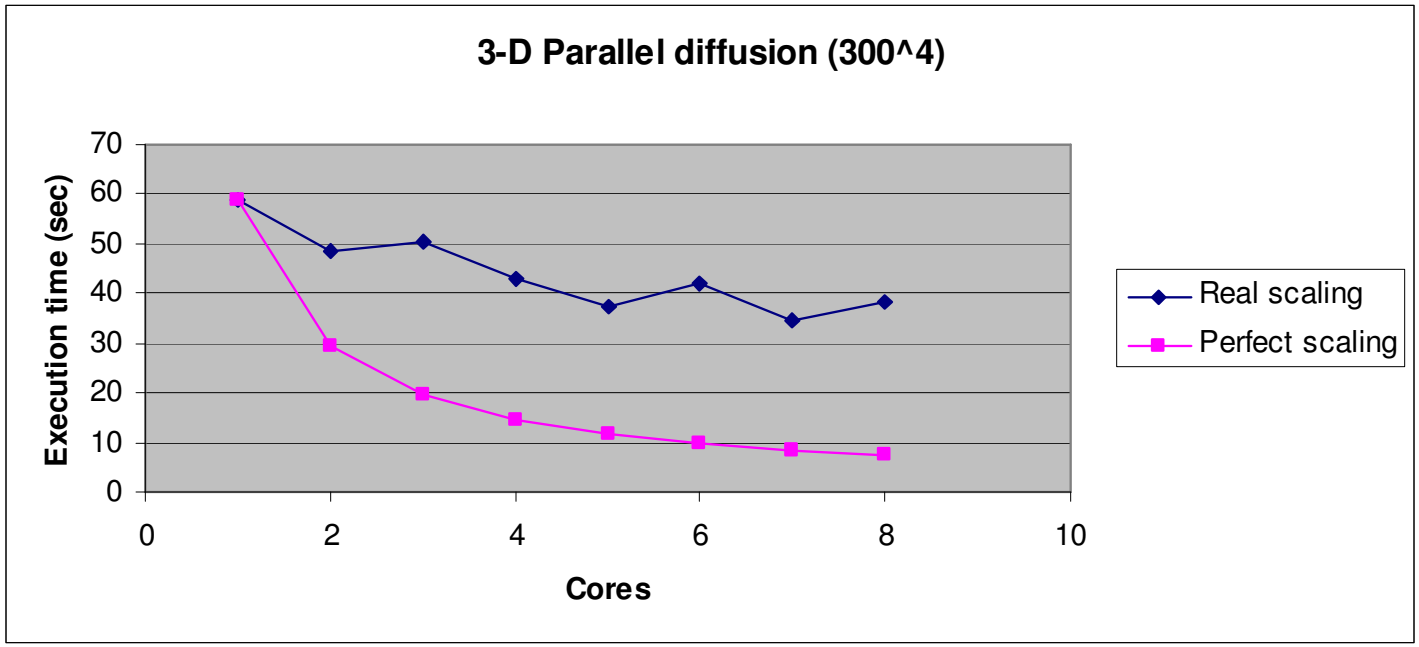


ΑΠΟΤΕΛΕΣΜΑΤΑ

- 8 core Intel(R) Xeon(R) CPU E5335 @ 2.00GHz
- 4MB L2 cache (shared between 2 cores)

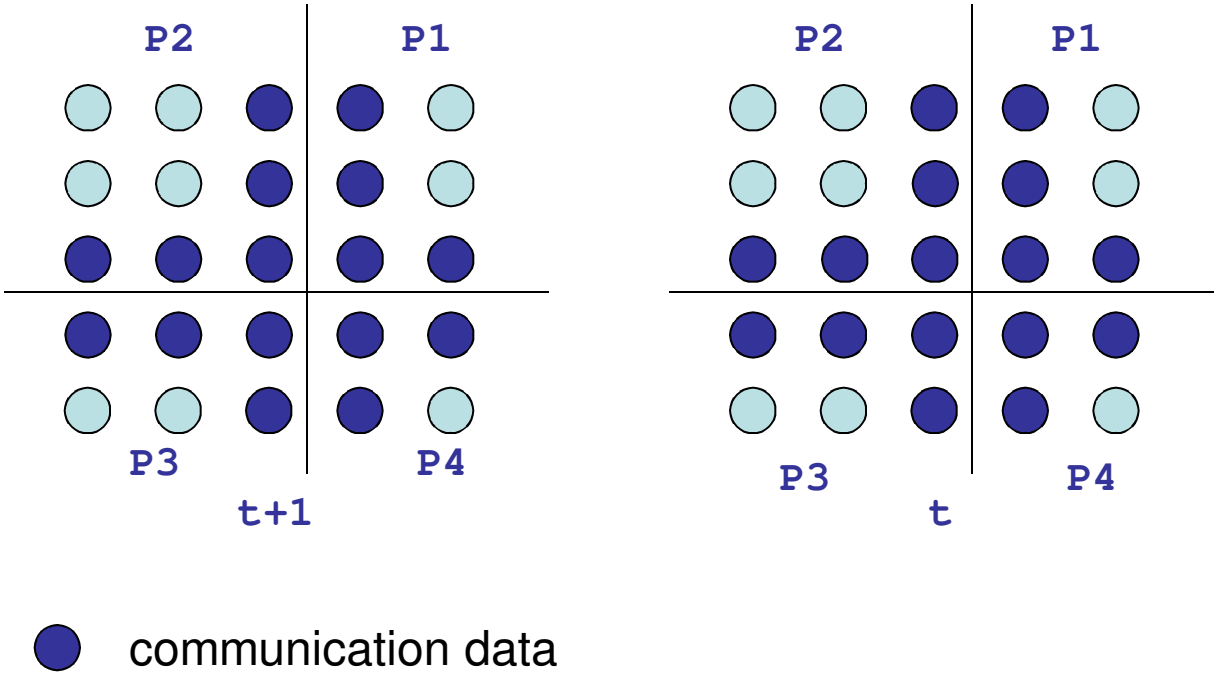


ΑΠΟΤΕΛΕΣΜΑΤΑ



Παραλληλοποίηση με ανταλλαγή μηνυμάτων

- Κατανομή υπολογιστικού πλέγματος σε διεργασίες

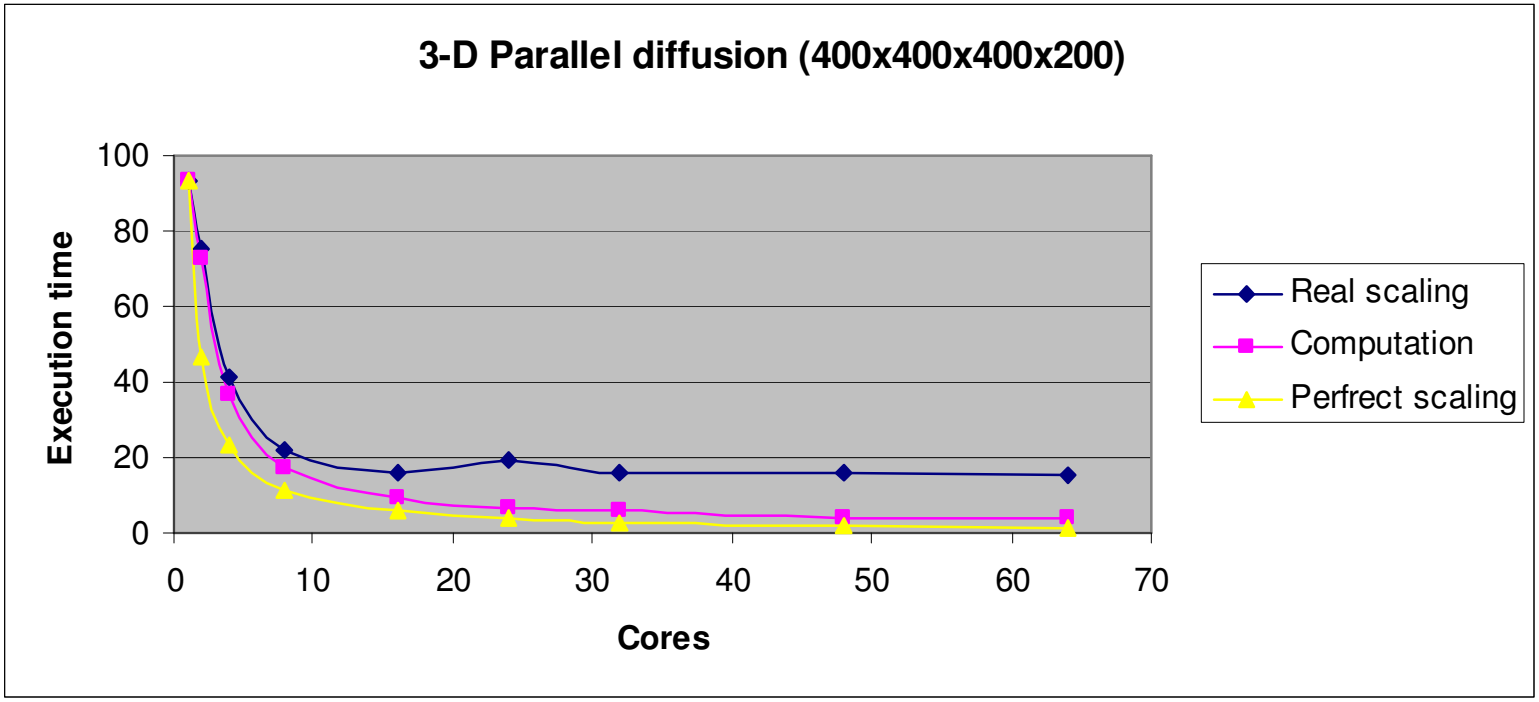


Παραλληλοποίηση με ανταλλαγή μηνυμάτων (pseudocode)

```
1.  while (!converged){
2.      Send_data_to_left(…);
3.      Receive_data_from_left(…);
4.      Send_data_to_right(…);
5.      Receive_data_from_right(…);
6.      Send_data_to_up(…);
7.      Receive_data_from_up(…);
8.      Send_data_to_down(…);
9.      Receive_data_from_down(…);
10.     Compute();
11. }
```

ΑΠΟΤΕΛΕΣΜΑΤΑ

- MPI implementation
- 8-node cluster (total 64 cores)



Αλγόριθμοι γράφων

Παράδειγμα: All to All Shortest Path (Αλγόριθμος Floyd-Warshall)

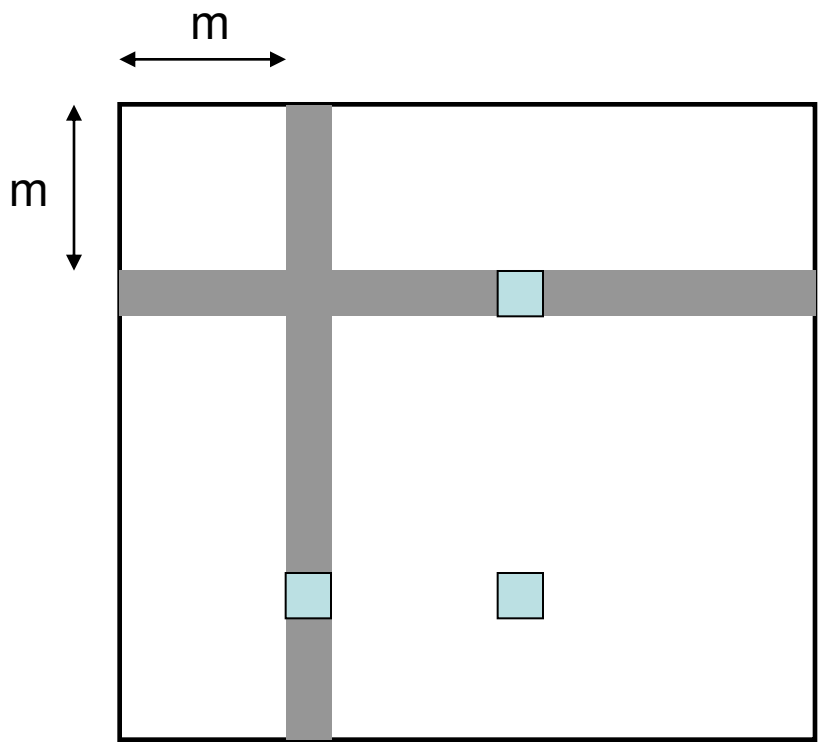
- **Σειριακός αλγόριθμος:**

1. **Procedure FLOYD_ALL_PAIRS_SP(A)**
2. **Begin**
3. $D^{(0)}$**:=**A;
4. **for** k=1 to n **do**
5. **for** i=1 to n **do**
6. **for** j=1 to n **do**
7. $d_{i,j}^{(k)} = \min(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)})$
8. **End FLOYD_ALL_PAIRS_SP**

Αλγόριθμοι γράφων

Παράδειγμα: All to All Shortest Path (Αλγόριθμος Floyd-Warshall)

k=m

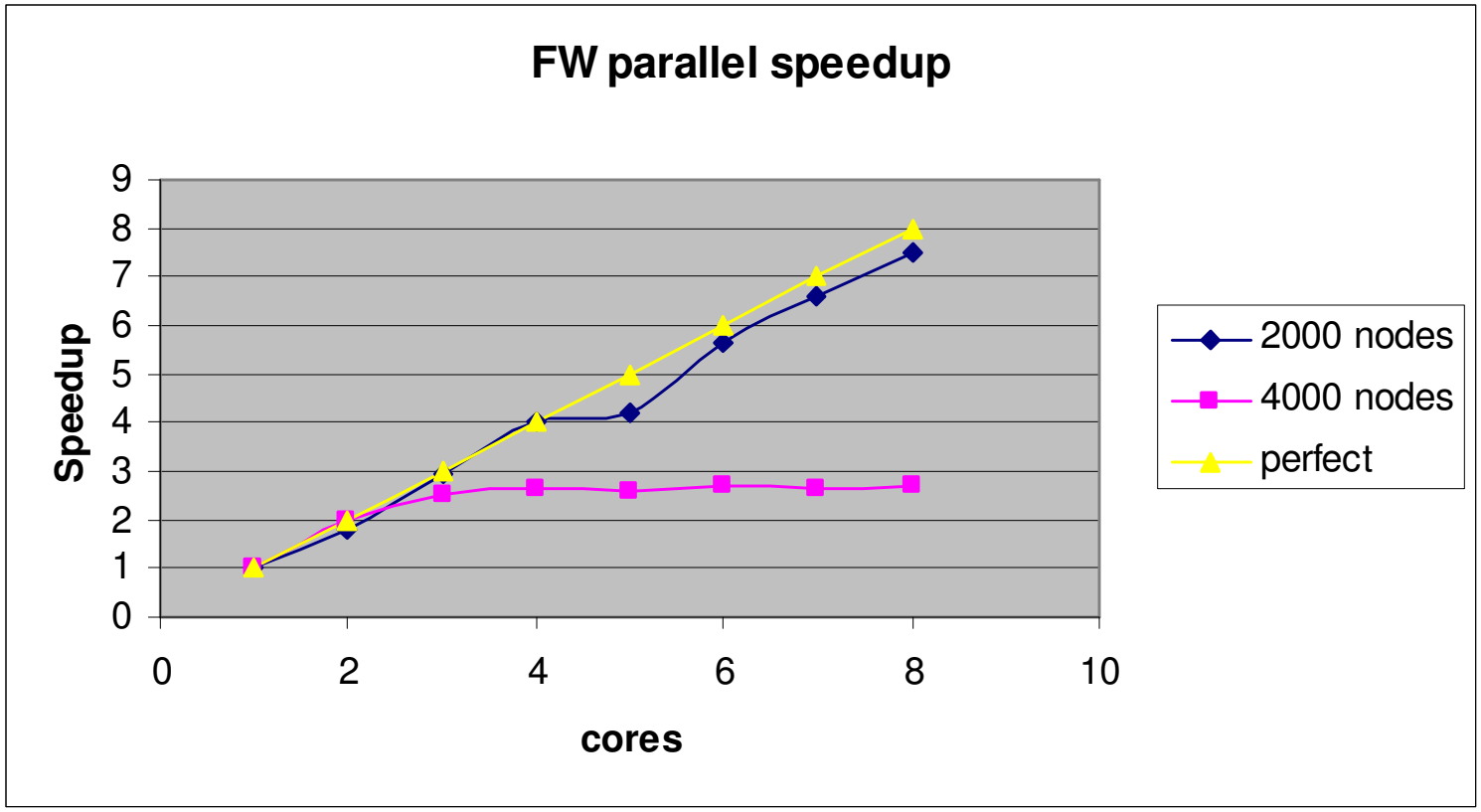


Floyd-Warshall: Παραλληλοποίηση σε κοινό χώρο διευθύνσεων

- **Procedure FLOYD_ALL_PAIRS_SP(A)**

1. **Begin**
2. $D^{(0)}$**:=**A;
3. **for** k=1 to n **do**
4. **#pragma omp parallel for (private i,j) shared D**
5. **for** i=1 to n **do**
6. **for** j=1 to n **do**
7. $d_{i,j}^{(k)} = \min(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)})$
8. **End FLOYD_ALL_PAIRS_SP**

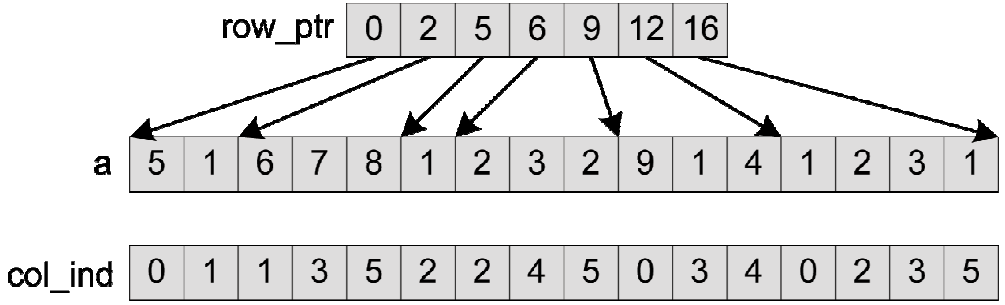
ΑΠΟΤΕΛΕΣΜΑΤΑ



Υπολογιστικός Πυρήνας SpMV

- Αποθήκευσης Συμπιεσμένης Αραιής Γραμμής (CSR)

$$A = \begin{bmatrix} 5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 7 & 0 & 8 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 3 & 2 \\ 9 & 0 & 0 & 1 & 4 & 0 \\ 1 & 0 & 2 & 3 & 0 & 1 \end{bmatrix}$$



- Υπολογιστικός πυρήνας

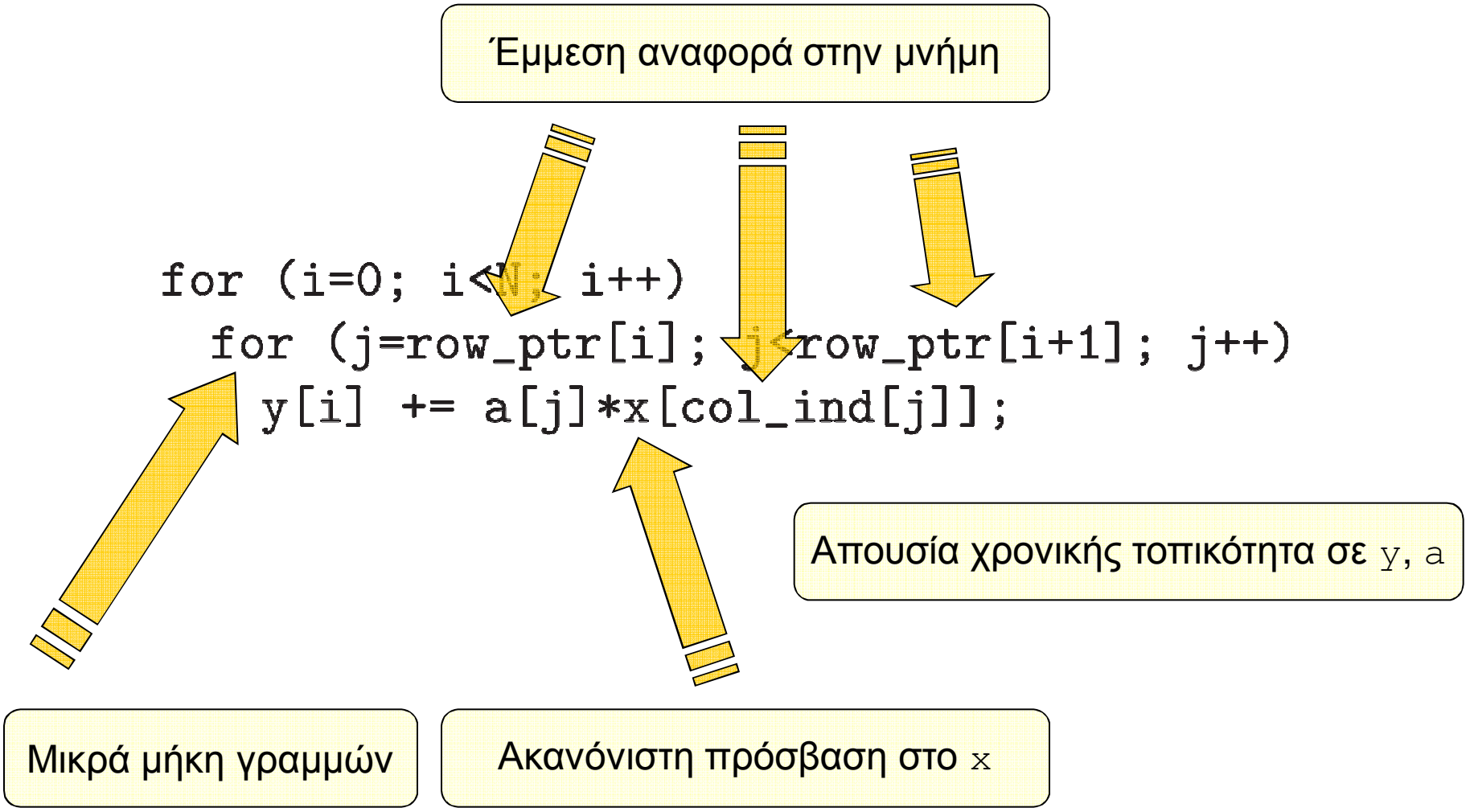
```
for (i=0; i<N; i++)
  for (j=0, l=i*M; j<M; j++)
    y[i] += a[l+j]*x[j];
```

DMV (πυκνός πίνακας)

```
for (i=0; i<N; i++)
  for (j=row_ptr[i]; j<row_ptr[i+1]; j++)
    y[i] += a[j]*x[col_ind[j]];
```

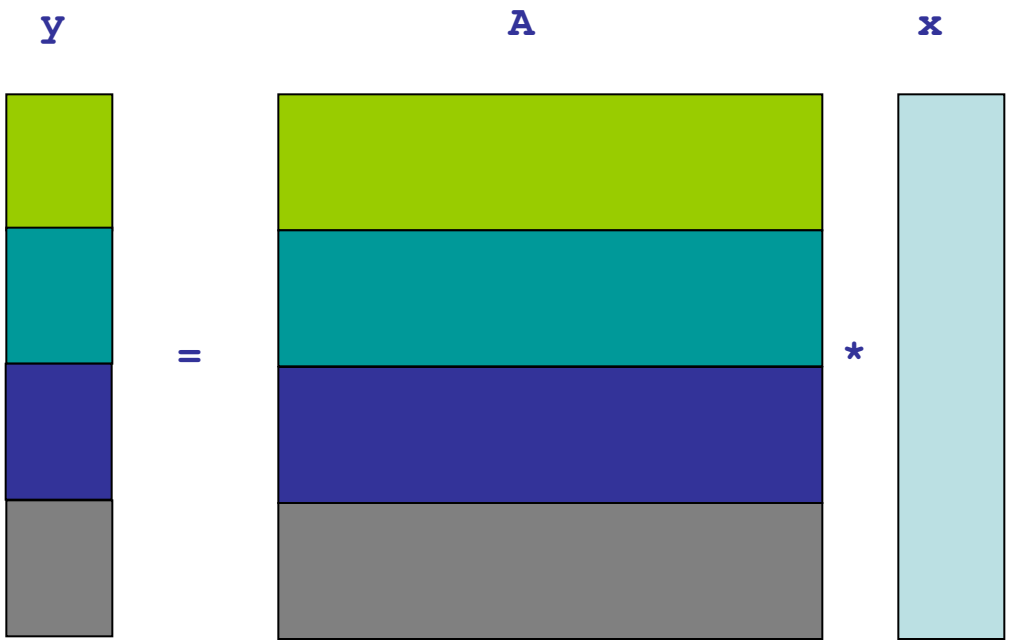
SpMV (αραιός πίνακας)

Θέματα επίδοσης SpMV

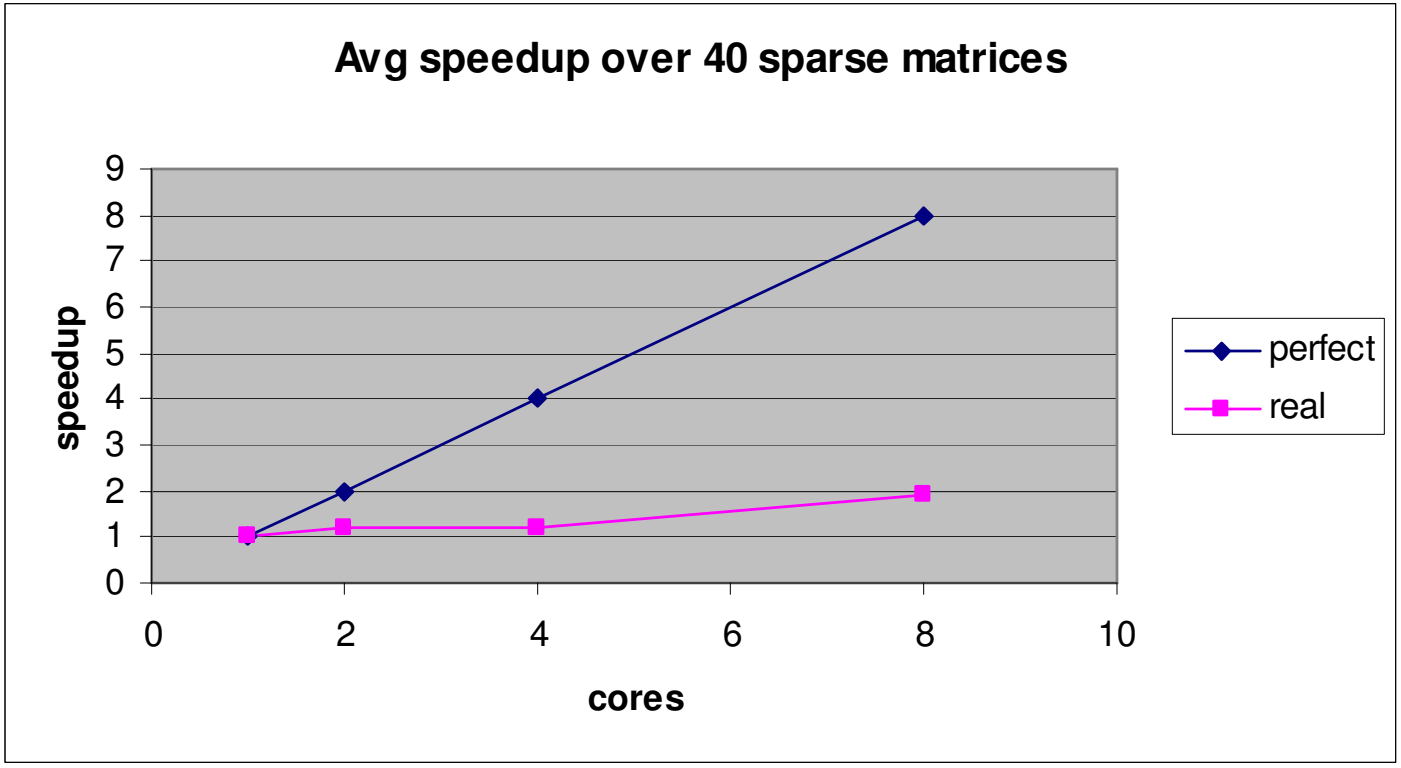


Παραλληλοποίηση σε κοινό χώρο διευθύνσεων

```
1.  #pragma omp parallel for private(i,j) shared (y,i,row_ptr,col_ind)
2.  for (i=0; i<N; i++)
3.     for (j=row_ptr[i]; j<row_ptr[i+1]; j++)
4.        y[i] += a[j]*x[col_ind[j]];
```

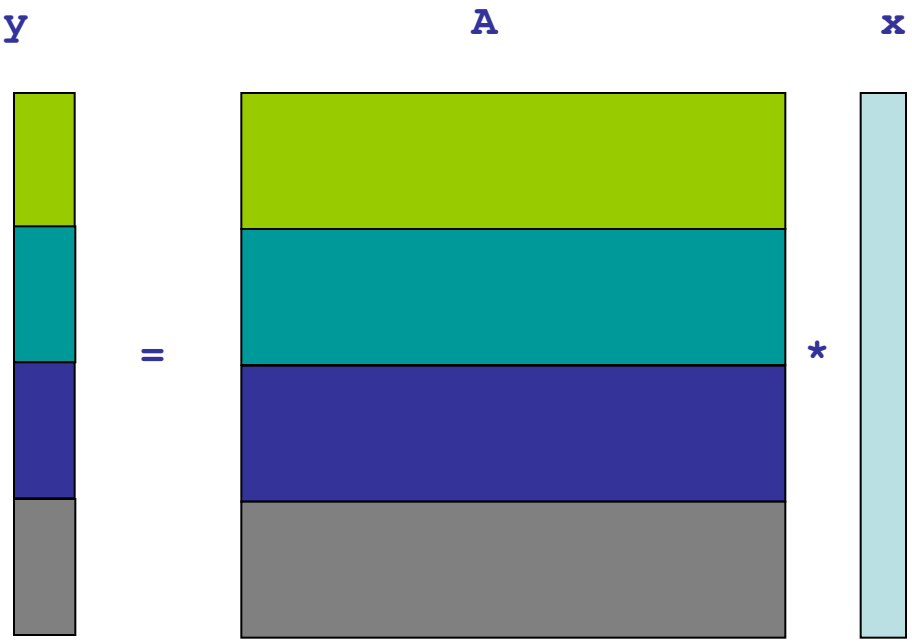


ΑΠΟΤΕΛΕΣΜΑΤΑ



Ζητήματα παραλληλοποίησης με ανταλλαγή μηνυμάτων

- Συνήθως η πολ/μός καλείται επαναληπτικά μέσα από κάποια μέθοδο επίλυση γραμμικών συστημάτων (CG, GMRES)
- y → x
- Κατανομή υπολογισμών και δεδομένων όπως πριν:



Ζητήματα παραλληλοποίησης με ανταλλαγή μηνυμάτων

- Απαιτείται επικοινωνία για το διάνυσμα x
- 2 προσεγγίσεις:
  - Ανταλλαγή ολόκληρου του διανύσματος μετά από κάθε επανάληψη
  - Ανταλλαγή μόνο των στοιχείων που χρειάζονται (ανάλογα με τη δομή του αραιού πίνακα)
    - Πρόβλημα graph partitioning: «Κατανομή των γραμμών του πίνακα A σε επεξεργαστές με στόχο την ισοκατανομή του φορτίου και της επικοινωνίας»
    - NP-complete
    - Πληθώρα αλγορίθμων και εργαλείων για την επίλυση (π.χ. METIS)

ΣΥΜΠΕΡΑΣΜΑΤΑ

- Η βελτιστοποίηση / παραλληλοποίηση κώδικα είναι μία επίπονη διαδικασία
- Απαιτεί κατανόηση της συμπεριφοράς εκτέλεσης της εφαρμογής (profiling)
- Βοηθά ιδιαίτερα η κατανόηση των αρχιτεκτονικών χαρακτηριστικών της πλατφόρμας εκτέλεσης (π.χ. memory hierarchy, interconnection network)
- Οι πολυπύρηνοι κόμβοι παρουσιάζουν συχνά προβλήματα στην κλιμάκωση των εφαρμογών κυρίως λόγω memory bottleneck