ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ
ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΠΟΛΙΤΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΔΟΜΟΣΤΑΤΙΚΗΣ

Ακαδ. Έτος: 2016-2017

Μάθημα: Εφαρμογές Η/Υ

Δευτέρα, 5/12/2016

Διδάσκοντες:
Ν.Δ. Λαγαρός (Επικ. Καθηγητής), Αθ. Στάμος (ΕΔΙΠ), Χ. Φραγκουδάκης (ΕΔΙΠ)

Παραδείγματα για την 10η παράδοση – Εισαγωγή στους παράλληλους υπολογισμούς

1. Παράλληλος υπολογισμός ολικού μητρώου δυσκαμψίας επιπέδου δικτυώματος

Το μητρώο δυσκαμψίας του ράβδου ij επιπέδου δικτυώματος που σχηματίζει γωνία θ με το οριζόντιο άξονα και συνδέει τον κόμβο i με τον κόμβο j δίνεται:

$$[K_{ij}]=\frac{A_{ij}E_{ij}}{L_{ij}}\begin{bmatrix} c^2 & & \text{συμ} & \\ cs & s^2 & & \\ -c^2 & -cs & c^2 & \\ -cs & -s^2 & cs & s^2 \end{bmatrix}$$

όπου L, A, E το μήκος, το εμβαδόν διατομής και το μέτρο ελαστικότητας αντίστοιχα και c=cos(θ), s=sin(θ).
Να συνταχθεί πρόγραμμα σε Python το οποίο:
1. Να διαβάζει από το αρχείο k.txt τα στοιχεία L, A, E, θ (μοίρες), i, j των ράβδων (μία ράβδος ανά σειρά αρχείου).
2. Να δημιουργεί και να τυπώνει με εκθετική μορφή και στοιχισμένα στο αρχείο k.txt το ολικό μητρώο δυσκαμψίας χρησιμοποιώντας όλους τους διαθέσιμους επεξεργαστές.
Δοκιμάστε το πρόγραμμά σας με το αρχείο k.txt (1000 ράβδοι):

```
3 0.09 200e9 0 0 1
4 0.16 200e9 90 0 2
5 0.12 200e9 143.13 1 2
...
```

3. Να γράφει στο τέλος του αρχείου time.csv το πλήθος των διεργασιών και το χρόνο υπολογισμού (χωρίς το χρόνο ανάγνωσης και εγγραφής σε αρχεία).

Λύση

Οι υπολογισμοί που μπορούν να γίνουν παράλληλα είναι η δημιουργία των επί μέρους μητρώων. Κάθε διεργασία (που εκτελείται σε έναν επεξεργαστή ή πυρήνα) υπολογίζει τα επί μέρους μητρώα δυσκαμψίας μερικών ράβδων και δημιουργεί ελλιπές “ολικό” μητρώο με αυτά. Στη συνέχεια τα ελλιπή “ολικά” μητρώα κάθε διεργασίας προστίθενται με την μέθοδο Reduce() για να προκύψει το ολικό μητρο του δικρυώματος.
Ο χωρισμός των ράβδων γίνεται ισόποσα σε n διεργασίες, όπου n το διαθέσιμο πλήθος επεξεργαστών ή πυρήνων. Αν η διαίρεση δίνει υπόλοιπο m, τότε στις πρώτες m διεργασίες προσθέτουμε μία ράβδο ακόμα.
Για να μοιράσουμε τις ράβδους ανομοιόμορφα χρησιμοποιούμε τη μέθοδο Scatterv() η οποία απαιτεί το πλήθος των ράβδων για κάθε διεργασία και τον αύξοντα αριθμό της πρώτης ράβδου της διεργασίας.

```
import time
import numpy as np
from mpi4py import MPI

def barMat(L, A, E, theta):
    "Computes the axial stiffness matrix of a bar in 2D."
    theta = np.deg2rad(theta)
    c = np.cos(theta)
    s = np.sin(theta)
    c2 = c*c
```

```
    s2 = s*s
    cs = c*s
    k = E*A/L
    K = np.zeros((4, 4))

    K[0, 0] = k * c2

    K[1, 0] = k * cs
    K[1, 1] = k * s2

    K[2, 0] = -K[0, 0]
    K[2, 1] = -K[1, 0]
    K[2, 2] =  K[0, 0]

    K[3, 0] = -K[1, 0]
    K[3, 1] = -K[1, 1]
    K[3, 2] =  K[1, 0]
    K[3, 3] =  K[1, 1]

    for i in range(3):
        K[i, i+1:] = K[i+1:, i]
    return K

def ypolKol(N, slocal):
    "Υπολόγισε ολικό μητρώο δυσκαμψίας."
    Kol = np.zeros((2*N, 2*N))
    for L, A, E, theta, i, j in slocal:
        K = barMat(L, A, E, theta)
        i = int(i*2)
        j = int(j*2)
        Kol[i:i+2, i:i+2] += K[0:2, 0:2]
        Kol[i:i+2, j:j+2] += K[0:2, 2:4]
        Kol[j:j+2, i:i+2] += K[2:4, 0:2]
        Kol[j:j+2, j:j+2] += K[2:4, 2:4]
    return Kol

def scatterargs(ns):
    "Create the arguments for scatter."
    n1 = ns // comm.size             #Γραμμές
    sendcounts = [n1*6] * comm.size  #Στοιχεία
    for i in range(ns-n1*comm.size): #Γραμμές
        sendcounts[i] += 1*6         #Στοιχεία
    displacements = [0]              #Στοιχεία
    for i in range(1, comm.size):
        displacements.append(displacements[i-1] + sendcounts[i-1])
    return sendcounts, displacements

def pyMain():
    "Start here."
    global comm
    comm = MPI.COMM_WORLD
    ns = 0
    if comm.rank == 0:
        s = np.loadtxt("k.txt")
        ns = len(s)
        N = int(np.max(s[:, 4:6])) + 1
    else:
        s = None
        ns = None
        N = None
    ns = comm.bcast(ns, root=0)
    N = comm.bcast(N, root=0)
    sendcounts, displacements = scatterargs(ns)
    slocal = np.zeros((sendcounts[comm.rank]//6, 6))  #Στοιχεία->γραμμές των 6
    comm.Scatterv([s, sendcounts, displacements, MPI.DOUBLE], slocal, root=0)
    t1 = time.time()
    Kol_local = ypolKol(N, slocal)
    if comm.rank == 0:
        Kol = np.zeros(Kol_local.shape)
    else:
        Kol = None
```

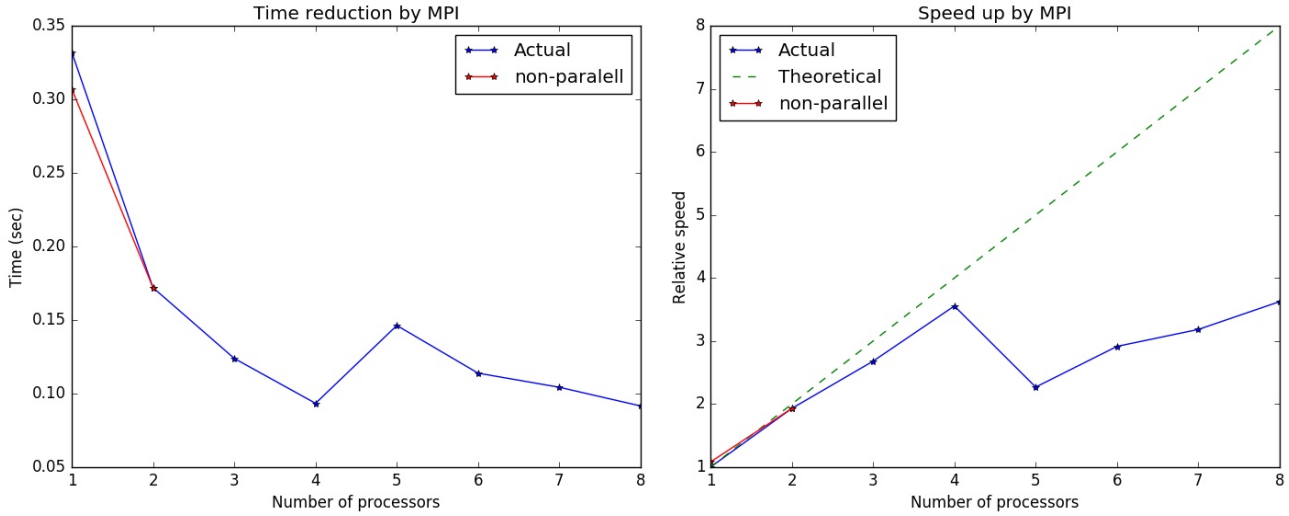
```
    comm.Reduce(Kol_local, Kol, op=MPI.SUM, root=0)
    t2 = time.time()
    if comm.rank == 0:
        np.savetxt("kol.txt", Kol, fmt="%14.6e")
        print(t2-t1, "sec")
        fw = open("time.csv", "a")
        fw.write("{:10d}  {:10.4f}\n".format(comm.size, t2-t1))
        fw.close()

pyMain()
```

Τρέξιμο του προγράμματος σε Intel Core I7, 2.96GHz, 4 πυρήνες/8 νήματα, 4GB RAM, SuSE Linux 42.1, με δικτύωμα αποτελούμενο από 1000 ράβδους, έδωσε τα εξής αποτελέσματα:

Processors	Time	Relative speed Theoretical	Actual	Utilization
nonpar. 1	0.3071	1.00	1.08	108 %
1	0.3321	1.00	1.00	100 %
2	0.1720	2.00	1.93	96 %
3	0.1239	3.00	2.68	89 %
4	0.0934	4.00	3.56	88 %
5	0.1463	5.00	2.27	45 %
6	0.1139	6.00	2.92	48 %
7	0.1043	7.00	3.18	45 %
8	0.0916	8.00	3.63	45 %

Η πρώτη σειρά δείχνει το πρόγραμμα χωρίς τη χρήση MPI καθόλου, που φυσικά τρέχει μόνο σε ένα επεξεργαστή. Η δεύτερη σειρά δείχνει το πρόγραμμα με MPI αλλά σε 1 επεξεργαστή, όπου φαίνεται ότι το MPI καθυστερεί λίγο σε σχέση με το καθαρά σειριακό πρόγραμμα (χωρίς MPI). Για μέχρι 4 επεξεργαστές (πυρήνες) θεωρητικά η ταχύτητα αυξάνει γραμμικά, ενώ οι επόμενοι 4 επεξεργαστές δεν είναι πραγματικοί επεξεργαστές (είναι “λογικοί” επεξεργαστές) αλλά δεύτερο νήμα στους πρώτους 4 πυρήνες.



Από τα διαγράμματα βλέπουμε ότι υπάρχει καλή απόδοση (σχεδόν γραμμική αύξηση της ταχύτητας) για τους 4 πρώτους πυρήνες. Στους επόμενους 4 (λογικούς) πυρήνες η απόδοση δεν είναι καθόλου καλή και δεν προσφέρει τίποτα στην αύξηση της ταχύτητας.

2. Παράλληλος υπολογισμός μετατόπισης ψηφιακού μοντέλου εδάφους (DEM)

Ψηφιακό μοντέλο εδάφους (DEM) είναι αντικείμενο που σε συγκεκριμένη περιοχή της γήινης επιφάνειας δίνει το υψόμετρο z του εδάφους για οποιοδήποτε σημείο x, y. Μπορεί να αναπαρασταθεί ως μία συνάρτηση z=f(x, y). Μερικές φορές λόγω διάφορων λαθών το DEM έχει μετατοπιστεί κατά Δx, Δy ως στερεό σώμα δηλαδή για να υπολογιστεί το υψόμετρο σημείου x, y πρέπει να προστεθεί Δx, Δy: z=f(x+Δx, y+Δy). Η μετατόπιση δημιουργεί σφάλμα αν δεν είναι γνωστή. Θεωρώντας N σημεία Xk, Yk με γνωστά υψόμετρα Zk το υψομετρικό σφάλμα υπολογίζεται:

$$e_k = Z_k - f(X_k, Y_k)$$

και το μέσο τετραγωνικό σφάλμα όλων των γνωστών σημείων:

$$e_{RMS} = \Sigma e_k / N$$

Για να υπολογιστεί η μετατόπιση δοκιμάζονται όλα τα δυνατά ζεύγη Δx, Δy και για κάθε ζεύγος υπολογίζεται το το μέσο τετραγωνικό σφάλμα:

$$e_k = Z_k - f(X_k+\Delta x, Y_k+\Delta y) \quad , \quad e_{RMS} = \Sigma e_k / N$$

Το ζεύγος Δx, Δy που δίνει το ελάχιστο μέσο τετραγωνικό σφάλμα είναι η ζητούμενη μετατόπιση.

α. Να συνταχθεί πρόγραμμα σε Python το οποίο υπολογίζει τη μετατόπιση DEM. Δίνονται:
i. Τα πρόγραμματα demasc.py, gen.py, xymm.py, jorpath.py τα οποία υλοποιούν κλάση DEM:

```
from demasc import ThanDEMasc
dem = ThanDEMasc()         #Δημιουργία αντικειμένου
dem.importText(fn)         #Ανάγνωση από αρχείο με όνομα fn
z = dem.thanPointZ([x, y]) #Υπολογισμός υψομέτρου σημείου
```

ii. Το αρχείο 17syk_simpl_b.demt το οποίο περιέχει το DEM.
iii. Το αρχείο trig.syn το οποίο περιέχει τα σημ,εία με γνωστό υψόμετρο: α/α x y z σε κάθε σειρά.
iv. Η διερεύνηση να γίνει για Δx, Δy από -100 έως 100 ανά 10.
Τα αρχεία υπάρχουν στη διεύθυνση: http://users.ntua.gr/stamthan/efarmogeshy/paradeigmata10/
Το πρόγραμμα να γράφει στο τέλος του αρχείου time.csv τον κωδικό 0 και το χρόνο υπολογισμού (χωρίς το χρόνο ανάγνωσης και εγγραφής σε αρχεία).

β. Το πρόγραμμα του ερωτήματος α. να γίνει παράλληλο.

Λύση α

Χρειάζεται διπλός βρόχος για τα Δx και Δy αντίστοιχα.

```
import time
import numpy as np
from demasc import ThanDEMasc
from math import sqrt

def readData():
    "Read the dem and the control points."
    dem = ThanDEMasc()
    dem.importText("17syk_simpl_b.demt")
    points = np.loadtxt("trig.syn", usecols=(1,2,3))
    return dem, points

def computeBest(dem, points):
    "Compute the best translation dx, dy."
    ermin = np.array((1.0e30, 0.0, 0.0))
    for dx in range(-100, 110, 10):
        for dy in range(-100, 110, 10):
            er = 0.0
            n = 0
            for p in points:
                z = dem.thanPointZ((p[0]+dx, p[1]+dy))
                if z is None: continue
                er += (p[2]-z)**2
                n += 1
            er = sqrt(er/n)
            if er < ermin[0]:
                ermin[:] = er, dx, dy
        print(dx)
    return ermin

def pyMain():
    "Start here."
    dem, points = readData()
    t1 = time.time()
    er, dx, dy = computeBest(dem, points)
    res = np.array([er, dx, dy])
    t2 = time.time()
    print(res)
    np.savetxt("best.txt", res, fmt="%15.3f")
```

```
    print(t2-t1, "sec")
    fw = open("time.csv", "a")
    fw.write("Processors  time (sec)\n")
    fw.write("{:10d}  {:10.4f}\n".format(0, t2-t1))
    fw.close()

pyMain()
```

Λύση β

Για να γίνει παράλληλο τα Δx θα μοιραστούν στις διεργασίες με τη μέθοδο Scatterv(). Αφού κάθε διεργασία υπολογίζει το ελάχιστο σφάλμα για τα Δ που της αντιστοιχούν, με τη μέθοδο Reduce(..., op=MPI.MIN) θα υπολογιστεί το ελάχιστο από αυτά. Επίσης χρησιμοποιούμε την έκδοση Bcast() της μεθόδου bcast() με κεφαλαίο το πρώτο γράμμα, τα ορίσματα της οποίας είναι μητρώα της βιβλιοθήκης numpy, και η οποία είναι γρηγορότερη.

```
import time
import numpy as np
from mpi4py import MPI
from demasc import ThanDEMasc
from math import sqrt

def readData():
    "Read the «dem and the control points."
    dem = ThanDEMasc()
    dem.importText("17syk_simpl_b.demt")
    points = np.loadtxt("trig.syn", usecols=(1,2,3))
    return dem, points

def computeBest(dem, points, dxslocal):
    "Compute the best translation dx, dy."
    ermin = np.array((1.0e30, 0.0, 0.0))

    for dx in dxslocal:
        for dy in range(-100, 110, 10):
            er = 0.0
            n = 0
            for p in points:
                z = dem.thanPointZ((p[0]+dx, p[1]+dy))
                if z is None: continue
                er += (p[2]-z)**2
                n += 1
            er = sqrt(er/n)
            #print(dx, dy, er)
            if er < ermin[0]:
                ermin[:] = er, dx, dy
    return ermin

def scatterargs(ns):
    "Create the arhuments for scatter."
    n1 = ns // comm.size
    sendcounts = [n1*1] * comm.size
    for i in range(ns-n1*comm.size):
        sendcounts[i] += 1*1
    displacements = [0]
    for i in range(1, comm.size):
        displacements.append(displacements[i-1] + sendcounts[i-1])
    return sendcounts, displacements

def pyMain():
    "Start here."
    global comm
    comm = MPI.COMM_WORLD

    if comm.rank == 0:
        dem, points = readData()
        npoints = len(points)
        dxs = np.array(range(-100, 110, 10), np.float)
        print("dxs=", dxs)
```

```
        ns = len(dxs)
    else:
        dem = points = npoints = dxs = ns = None
    dem = comm.bcast(dem, root=0)
    npoints  = comm.bcast(npoints, root=0)
    if comm.rank != 0:
        points = np.zeros((npoints, 3), np.float)
    comm.Bcast(points, root=0)
    ns = comm.bcast(ns, root=0)

    sendcounts, displacements = scatterargs(ns)
    print("sendcount=", sendcounts)
    print("displacements=", displacements)
    dxslocal = np.zeros((sendcounts[comm.rank]//1,))
    comm.Scatterv([dxs, sendcounts, displacements, MPI.DOUBLE], dxslocal, root=0)
    print("rank=", comm.rank, "dxslocal size=", len(dxslocal))
    print(MPI.Get_processor_name(), comm.rank, 'dxslocal=', dxslocal)

    t1 = time.time()
    er, dx, dy = computeBest(dem, points, dxslocal)
    res_local = np.array([er, dx, dy])
    print("rank=", comm.rank, "result=", res_local)
    if comm.rank == 0:
        res = np.zeros(res_local.shape)
    else:
        res = None
    comm.Reduce(res_local, res, op=MPI.MIN, root=0)
    t2 = time.time()
    if comm.rank == 0:
        np.savetxt("best.txt", res, fmt="%15.3f")
        print(t2-t1, "sec")
        fw = open("time.csv", "a")
        fw.write("{:10d}  {:10.4f}\n".format(comm.size, t2-t1))
        fw.close()

pyMain()
```

Στα μητρώα res_local της κάθε διεργασίας αποθηκεύονται τα Δx, Δy και σφάλμα e που βρήκε η κάθε διεργασία, και με τη μέθοδο Reduce() υπολογίζεται το ελάχιστο res όλων των μητρώων res_local. Δυστυχώς αυτό δεν λειτουργεί διότι το res_local έχει το ελάχιστο των e, αλλά δεν έχει τα Δx, Δy που αντιστοιχούν σε αυτό αλλά το ελάχιστο Δx και το ελάχιστο Δy από όλες τις διεργασίες. Για να διορθωθεί, θα συνταχθεί συνάρτηση που ελέγχει μόνο το πρώτο στοιχείο δύο μητρώων a και b και επιστρέφει αυτό με το μικρότερο πρώτο στοιχείο.

```
def minLexical(a, b, mpitype):
    "Given 2 1dimensional numpy arrays return the smallest lexicographically."
    if a[0] <= b[0]: return a
    return b
```

Το όρισμα mpitype δεν χρησιμοποιείται. Στη συνέχεια δημιουργείται νέος τελεστής (operator) του MPI και χρησιμοποιείται αυτός στη μέθοδο Reduce() αντί για τον τελεστή MPI.MIN:

```
def pyMain():
    ...
    if comm.rank == 0:
        res = np.zeros(res_local.shape)
    else:
        res = None
    MINLEXICAL = MPI.Op.Create(minLexical, commute=True)
    comm.Reduce(res_local, res, op=MINLEXICAL, root=0)
    ...
```

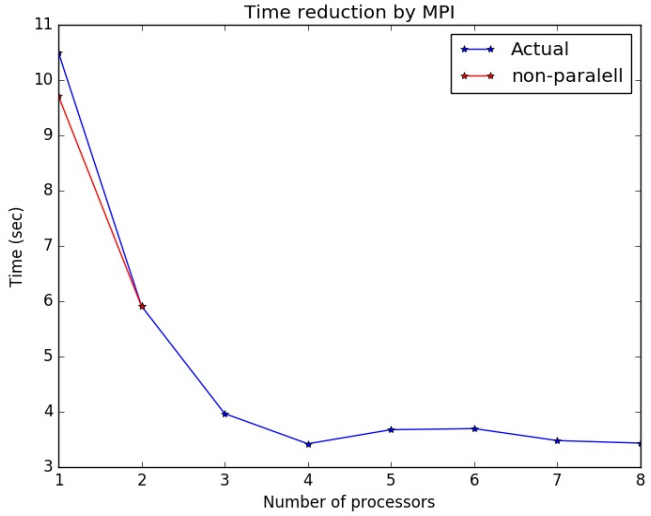
Το όρισμα commute=True σημαίνει ότι ο τελεστής είναι αντιμεταθετικός.

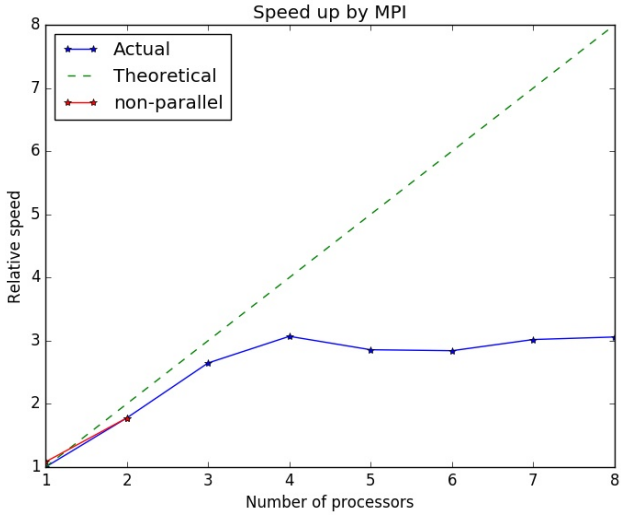
Τρέξιμο του προγράμματος σε Intel Core I7, 2.96GHz, 4 πυρήνες/8 νήματα, 4GB RAM, SuSE Linux 42.1, με δικτύωμα αποτελούμενο από 1000 ράβδους, έδωσε τα εξής αποτελέσματα:

```
                        Relative speed
```

Processors	Time	Theoretical	Actual	Utilization
nonpar. 1	9.7164	1.00	1.08	108 %
1	10.5051	1.00	1.00	100 %
2	5.9115	2.00	1.78	88 %
3	3.9680	3.00	2.65	88 %
4	3.4224	4.00	3.07	76 %
5	3.6765	5.00	2.86	57 %
6	3.6963	6.00	2.84	47 %
7	3.4797	7.00	3.02	43 %
8	3.4337	8.00	3.06	38 %

Η πρώτη σειρά δείχνει το πρόγραμμα χωρίς τη χρήση MPI καθόλου, που φυσικά τρέχει μόνο σε ένα επεξεργαστή. Η δεύτερη σειρά δείχνει το πρόγραμμα με MPI αλλά σε 1 επεξεργαστή, όπου φαίνεται ότι το MPI καθυστερεί λίγο σε σχέση με το καθαρά σειριακό πρόγραμμα (χωρίς MPI). Για μέχρι 4 επεξεργαστές (πυρήνες) θεωρητικά η ταχύτητα αυξάνει γραμμικά, ενώ οι επόμενοι 4 επεξεργαστές δεν είναι πραγματικοί επεξεργαστές (είναι “λογικοί” επεξεργαστές) αλλά δεύτερο νήμα στους πρώτους 4 πυρήνες.





Από τα διαγράμματα βλέπουμε ότι υπάρχει αρκετά καλή απόδοση (σχεδόν γραμμική αύξηση της ταχύτητας) για τους 4 πρώτους πυρήνες. Στους επόμενους 4 (λογικούς) πυρήνες η απόδοση δεν είναι καθόλου καλή και δεν προσφέρει τίποτα στην αύξηση της ταχύτητας.