ΠΑΝΕΠΙΣΤΗΜΙΟ ΣΤΕΡΕΑΣ ΕΛΛΑΔΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Σχεδιασμός και Ανάπτυξη Κρυπτο-Βιβλιοθήκης για συστήματα πολλαπλών πυρήνων

Κωστόπουλος Μιλτιάδης

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υπεύθυνος
Κακαρούντας Αθανάσιος
Διδάσκων του μαθήματος «Αρχιτεκτονική Υπολογιστών»
Επίκουρος Καθηγητής

Λαμία, 2010

Περίληψη

Η ασφάλεια στο χώρο των πληροφοριακών συστημάτων κατείχε πάντα το μεγαλύτερο ρόλο κατά το σχεδιασμό τους. Η κυριότερη μέριμνα είναι η εξασφάλιση του απόρρητου κατά την μετάδοση, αποθήκευση και ανάγνωση κάποιων δεδομένων.

Για να εξασφαλιστεί αυτό το απόρρητο χρησιμοποιήθηκαν διάφοροι μέθοδοι, ένας από τους οποίους είναι η κρυπτογράφηση. Υπάρχουν πολλά πρότυπα κρυπτογράφησης και χρησιμοποιούνται ανάλογα, του επιπέδου ασφαλείας που επιθυμούμε και του δαπανούμενου κόστους. Η μέχρι τώρα τεχνολογία των επεξεργαστών των ηλεκτρονικών υπολογιστών ήταν εμπόδιο στην υλοποίηση των προτύπων κρυπτογράφησης μέσω λογισμικού (software), αφού είχε μεν, ως αποτέλεσμα, το μικρό κόστος, αλλά από την άλλη μια αρκετά μικρή ταχύτητα κατά την εκτέλεση του κρυπτογραφικού αλγορίθμου. Ως αποτέλεσμα η υλοποίηση έπρεπε να γίνει μέσο υλικού (hardware) για μεγαλύτερη ταχύτητα αλλά αυτό φυσικά είχε και πολύ μεγαλύτερο κόστος .

Σκοπός της Πτυχιακής Εργασίας είναι αφενός να παρουσιαστεί η αρχιτεκτονική των επεξεργαστών πολλαπλών πυρήνων ενός ηλεκτρονικού υπολογιστή και αφετέρου να μελετηθεί η ανάπτυξη κρυπτο-βιβλιοθήκης ώστε να υπάρξει εκμετάλλευση της τεχνολογίας των πολλαπλών πυρήνων στο χώρο της ασφάλειας, με απώτερο σκοπό η υλοποίηση των κρυπτογραφικών αλγορίθμων να γίνεται μέσο λογισμικού χωρίς τα μειονεκτήματα που υπήρχαν μέχρι πρότινος.

Στην παρούσα πτυχιακή θα περιοριστεί στους στους επεξεργαστές οικιακών υπολογιστών γιατί είναι πιο προσβάσιμοι από οικονομικής απόψεως σε σχέση με τους επεξεργαστές διακομιστών και σταθμών εργασίας και επειδή στους επεξεργαστές ειδικής χρήσης ή/και ενσωματωμένων συστημάτων υπάρχουν πολλές διαφοροποιήσεις και παραμετροποιήσεις ανάλογα με τις ανάγκες της κάθε συσκευής, δυσκολεύοντας την μελέτη τους και την ανάπτυξη λογισμικού για αυτούς. Καθώς επίσης και στο πρότυπο κρυπτογράφησης AES λόγο της ευρείας διάδοσης του στο χώρο της ασφάλειας, λόγο της απλής σχετικά δομής του σε σχέση με την ασφάλεια που αποδίδει, αλλά και λόγο της κατά καιρούς ενδελεχούς μελέτης του.

Λέξεις-κλειδιά: κρυπτογράφηση, αλγόριθμοι κρυπτογράφησης, AES, πολλαπλοί επεξεργαστικοί πυρήνες, επεξεργαστές, OpenMP.

Abstract

The security in the area of information systems had always been the biggest role in their design. The main concern is to ensure confidentiality during transmission, storing and reading data.

Have been used different methods, to ensure this safety, one of which is the encryption. There are many encryption standards used depending of the wanted level of security and cost. Past processor technology of computers was a barrier to achieving the standards of encryption software (software), as a result, low cost, but on the other hand a fairly low speed during the execution of the cryptographic algorithm. As a result, the implementation should be a means of material (hardware) for more speed but of course, a much higher cost.

The purpose of this study is, first to present the architecture of a multi-core processor computer and to study the development of crypto-library to exploit the multicore technology in the field of security, leading to the implementation of cryptographic algorithms by software without the drawbacks that existed previously.

In this study, we will be restricted to home computers processors because they are more accessible from an economic standpoint in relation to server processors and workstation processors, and because of specific used processors and / or embedded systems, there are many variations and configurations depending on the needs of each device, makes harder the studying and development of software for them. As well as the standard AES encryption because of the wide dissemination in the area of security, because of the relatively simple structure in relation to security returns, but also because of the occasional in-depth study.

Keywords: encryption, encryption algorithms, AES, multiple processor cores, processors, OpenMP.

Το αφιερώνω στους γονείς μου για όλη την υποστήριξη τους, υλική και πνευματική. Και ένα μεγάλο ευχαριστώ στον καθηγητή μου για την κατανόηση και την βοήθεια του.

Περιεχόμενα

1 Αρχιτεκτονική Επεξεργαστών

1.1 Εισαγωγή

Η ανάγκη του ανθρώπου για περίπλοκους υπολογισμούς, τον οδήγησε στο να σκεφτεί λύσεις ώστε να κάνει αυτούς τους υπολογισμούς στο μικρότερο δυνατό χρόνο και με το λιγότερο δυνατό κόπο. Έτσι κατά τη περίοδο του Β' Παγκόσμιου Πολέμου ο John Mauchly και ο John Presper Eckert πρότειναν τη κατασκευή ενός ηλεκτρονικού υπολογιστή γενικής χρήσης, γνωστού με το όνομα ENIAC[1], για τις ανάγκες του εργαστηρίου βαλλιστικής. Αυτός ο υπολογιστής περιείχε πάνω από 18000 λυχνίες κενού, και ο προγραμματισμός του γινόταν με το χέρι, ανοίγοντας και κλείνοντας διακόπτες ή μετακινώντας συνδετικά καλώδια. Η δυσκολία στην εισαγωγή και μεταβολή προγραμμάτων, σε αυτό τον υπολογιστή, οδήγησε στην κατασκευή ενός νέου ηλεκτρονικού υπολογιστή, από το ινστιτούτο προηγμένης έρευνας από που και πήρε το όνομα του (*Institute for Advanced Study - IAS*), που θα μπορούσε να αποθηκεύει ένα πρόγραμμα, μαζί με τα δεδομένα, σε μία μνήμη έτσι ώστε η δημιουργία ή η μεταβολή ενός προγράμματος να γίνεται καθορίζοντας τις τιμές ενός μέρους της μνήμης.

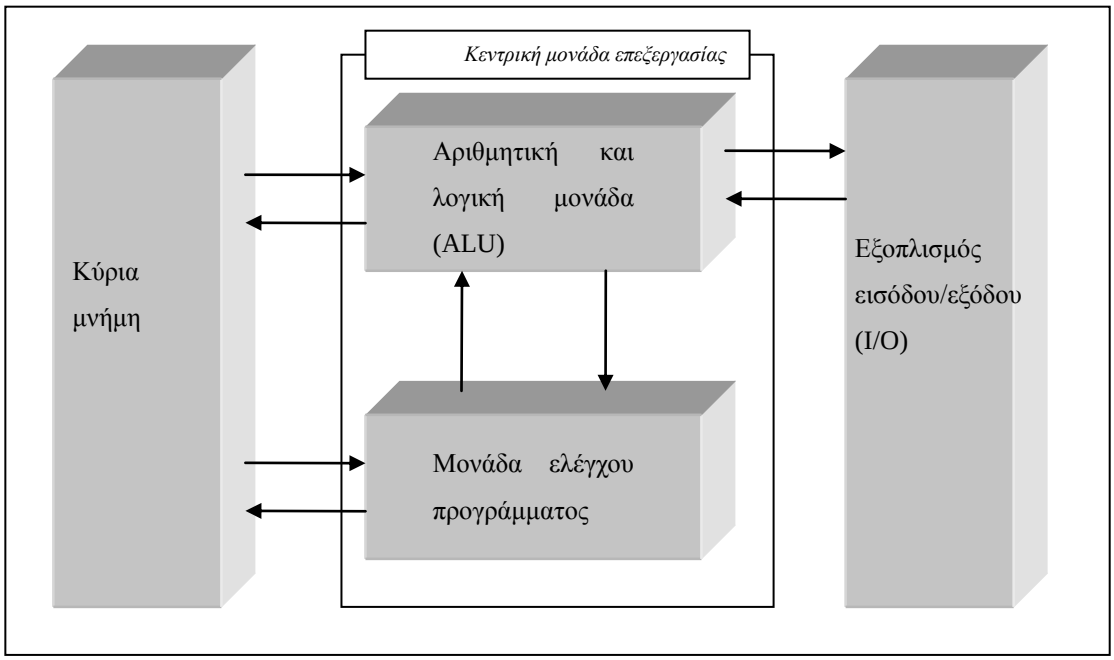
Η δομή αυτού του υπολογιστή είναι και το πρότυπο όλων των επόμενων ηλεκτρονικών υπολογιστών γενικής χρήσης.

Ο IAS αποτελείται από:

- Μια κύρια μνήμη, η οποία αποθηκεύει τόσο δεδομένα, όσο και εντολές.
- Μια αριθμητική και λογική μονάδα (*Arithmetic Logic Unit - ALU*), ικανή να χειρίζεται δυαδικά δεδομένα.
- Μια μονάδα ελέγχου, η οποία ερμηνεύει τις εντολές μέσω της μονάδας ελέγχου.

[1] **Electronic Numerical Integrator and Computer** *(Ηλεκτρονικός αριθμητικός ολοκληρωτής και υπολογιστής)*

- Μονάδες Εισόδου/Εξόδου(*I/O*), οι οποίες λειτουργούν μέσω της μονάδας ελέγχου.



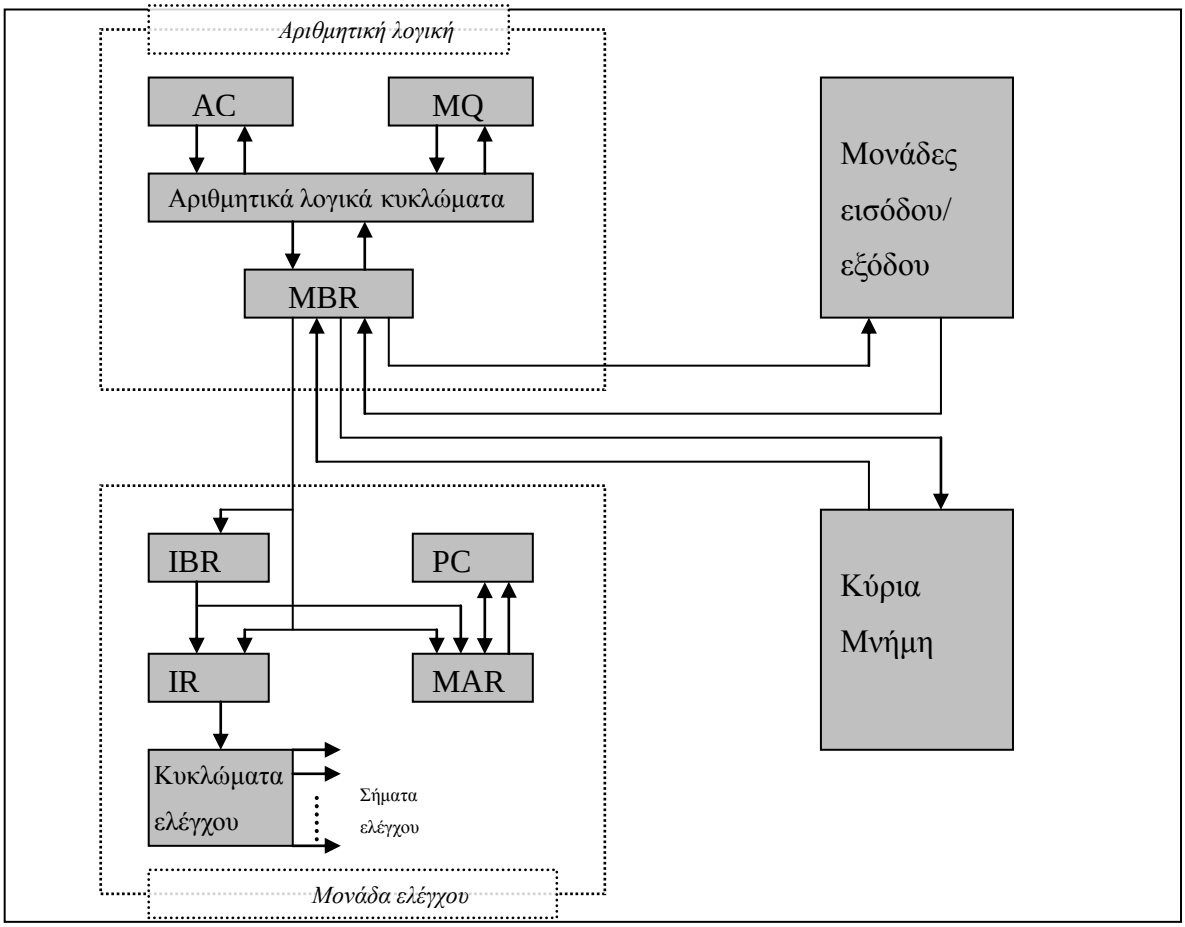
Εικόνα 1.1: Η δομή του IAS

i. Η ALU είναι ένα εξειδικευμένο υποσύστημα για την εκτέλεση των αριθμητικών πράξεων (*πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση*).
ii. Η μονάδα ελέγχου προγράμματος είναι αυτή που μεριμνά για τη σωστή αλληλουχία και την εκτέλεση των εντολών.
iii. Η κύρια μνήμη είναι αυτή που αποθηκεύει τα δεδομένα και τις εντολές που θα προωθηθούν στην ALU για εκτέλεση.
iv. Οι μονάδες εισόδου/εξόδου (*I/O*) είναι αυτές που επιτρέπουν τη μεταφορά πληροφορίας από και προς τις εξειδικευμένες μονάδες (*CPU και μνήμη*).

Με κάποιες εξαιρέσεις, όλοι οι σημερινοί μονού πυρήνα υπολογιστές έχουν την παραπάνω δομή και λειτουργία.

Η μνήμη του υπολογιστή αποτελείται από έναν αριθμό *N* θέσεων αποθήκευσης που ονομάζονται λέξεις, η κάθε μια με *M* δυαδικά ψηφία (*binary digits*, ή πιο απλά, *bits*). Στις θέσεις αυτές αποθηκεύονται τόσο δεδομένα, όσο και εντολές. Έτσι οι αριθμοί πρέπει να αναπαρίστανται σε δυαδική μορφή, και κάθε εντολή πρέπει να είναι ένας δυαδικός κώδικας. Μια λέξη περιέχει έναν αριθμό ή 2 εντολές.

Η μονάδα ελέγχου ελέγχει τη λειτουργία του υπολογιστή προσκομίζοντας εντολές από τη μνήμη και εκτελώντας τις σειριακά (τη μια μετά την άλλη). Για να επεξηγηθεί αυτό χρειάζεται ένα λεπτομερέστατο δομικό διάγραμμα.



Εικόνα 1.2: Αναλυτική δομή του IAS

Στην *Εικόνα 1.2* γίνεται εμφανές ότι τόσο η μονάδα ελέγχου όσο και η ALU περιέχουν θέσεις αποθήκευσης πληροφορίας, οι οποίες ονομάζονται καταχωρητές(*registers*), και ορίζονται ως εξής:

- Ενδιάμεσος καταχωρητής μνήμης (*Memory Buffer Register, MBR*): περιέχει μια λέξη που πρόκειται να αποθηκευτεί στη μνήμη, ή χρησιμοποιείται για να δεχθεί μια λέξη από τη μνήμη.
- Καταχωρητής διεύθυνσης μνήμης (Memory *Address Register, MAR*): καθορίζει τη διεύθυνση στη μνήμη για τη λέξη η οποία πρόκειται να γραφεί ή να αναγνωσθεί από το MBR.

- Καταχωρητής Εντολών *(Instruction Register, IR*): Περιέχει τον οκτάμπιτο κώδικα εντολής που εκτελείται εκείνη τη στιγμή.
- Προσωρινός καταχωρητής εντολής (*Instruction Buffer Register, IBR*): χρησιμοποιείται για την προσωρινή αποθήκευση της πρώτης εντολής από μία λέξη στη μνήμη.
- Απαριθμητής προγράμματος (*Program Counter, PC*): Περιέχει τη διεύθυνση του επόμενου ζεύγους εντολών που πρόκειται να προσκομιστεί από τη μνήμη.
- Συσσωρευτής (*Accumulator, AC*) και δείκτης Πολλαπλασιαστή (*Multiplier Quotient, MQ*): Χρησιμοποιείται για τη προσωρινή αποθήκευση τελεστών και αποτελεσμάτων των πράξεων της ALU. Για παράδειγμα, το αποτέλεσμα του πολλαπλασιασμού δυο αριθμών 40 bits είναι ένας αριθμός 80 bits. Τα σημαντικότερα 40 bits αποθηκεύονται στο καταχωρητή AC, και τα λιγότερο σημαντικά στο καταχωρητή MQ.

Οι εντολές ενός προγράμματος πρέπει να μεταδοθούν από τη κύρια μνήμη στη CPU εκεί οπού και μπορούν να εκτελεστούν. Ένα πρόγραμμα αποτελείται από πολλές εντολές οι οποίες αξιοποιούν τους πόρους της CPU. Η CPU συνήθως χρονίζεται σε μεγαλύτερη ταχύτητα από ότι η κύρια μνήμη, και έτσι συχνά η μνήμη είναι αυτή που αποτελεί τον κύριο περιορισμό στην ταχύτητα εκτέλεσης ενός προγράμματος. Παρακάτω θα εξηγηθούν τα βήματα που ακολουθούνται από τη CPU για να τρέξει μια εντολή.

Πρώτο βήμα είναι η προσκόμιση εντολής *(Instruction Fetch, IF*). Σε αυτό το βήμα η εντολή προσκομίζεται από τη κύρια μνήμη στη CPU, η διεύθυνση της οποίας βρίσκεται στο PC καταχωρητή. Η εντολή αντιγράφεται από τη μνήμη στο IR.

Δεύτερο βήμα είναι η αποκωδικοποίηση της εντολής, στο οποίο η εντολή αποκωδικοποιείται ώστε η CPU να ξέρει από που θα προσκομίσει τα δεδομένα. Ένα παράδειγμα για να γίνει πιο κατανοητό αυτό το βήμα είναι η διαφορά μεταξύ μιας εντολής **add** και μιας εντολής **addi**. Με την εντολή **add** προστίθενται το περιεχόμενο 2 καταχωρητών, οπότε θα πρέπει να προσκομιστούν οι τιμές των καταχωριτών αυτών ώστε να εκτελεστεί η πράξη της πρόσθεσης. Με την εντολή **addi**

προσκομίζεται η τιμή ενός καταχωρητή και μιας sign-extend immediate τιμής για την πράξη της πρόσθεσης.

Τρίτο βήμα είναι η εκτέλεση της εντολής, η οποία γίνεται στην ALU.

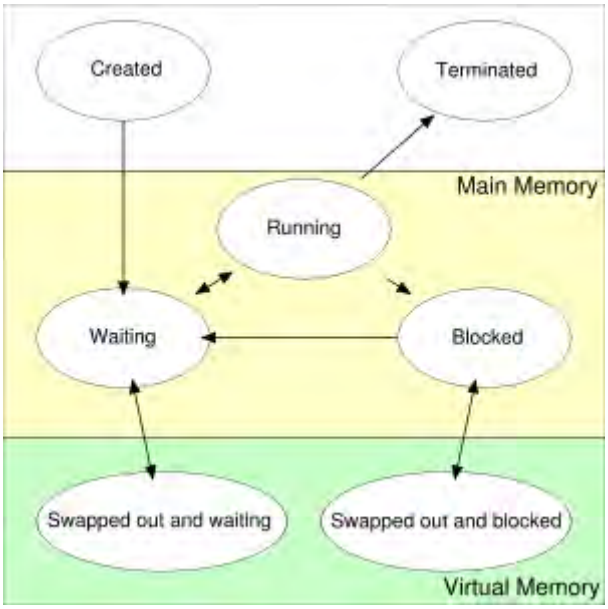
Τέταρτο βήμα είναι η εγγραφή του αποτελέσματος της λειτουργίας που έγινε στο βήμα τρία, στο κατάλληλο καταχωρητή.

Πέμπτο και τελευταίο βήμα είναι η ενημέρωση του PC register. Συνήθως η ενημέρωση που γίνεται είναι PC<-PC+4, αλλά σε μια εντολή διακλάδωσης ή αναπήδησης η τιμή του μπορεί να ενημερωθεί σε άλλη διεύθυνση.

1.2 Εικονικός Παραλληλισμός

Οι πρώτοι Ηλεκτρονικοί Υπολογιστές (*H/Y*), μπορούσαν να εκτελέσουν μόνο ένα πρόγραμμα κάθε φορά. Όσο η ταχύτητα των επεξεργαστών αυξάνονταν τόσο αυτοί παρέμεναν ανενεργοί, κατά την εκτέλεση ενός προγράμματος, περιμένοντας για κάποια είσοδο από κάποιο πόρο. Η λύση στο πρόβλημα αυτό ήταν η δημιουργία των διαδικασιών (*processes*).

Μια διαδικασία είναι ένα στιγμιότυπο ενός προγράμματος το οποίο εκτελείται ακολουθιακά από ένα υπολογιστικό σύστημα το οποίο έχει τη δυνατότητα να τρέξει πολλά προγράμματα ταυτόχρονα. Διάφορες διαδικασίες μπορεί να συνδέονται με το ίδιο πρόγραμμα. Για παράδειγμα, το άνοιγμα πολλών παραθύρων του ίδιου προγράμματος τυπικά σημαίνει και την εκτέλεση παραπάνω της μίας διαδικασίας. Για να είναι δυνατή η εκτέλεση διάφορων προγραμμάτων ταυτόχρονα από ένα υπολογιστικό σύστημα με έναν επεξεργαστή μονού πυρήνα χρησιμοποιείται η μέθοδος διαμοίρασης χρόνου(*time-sharing*). Η διαμοίραση χρόνου επιτρέπει στις διαδικασίες να εναλλάσσονται μεταξύ των καταστάσεων '*εκτελείται'* (*running*) και '*σε αναμονή*' (*waiting*). Αυτή η εναλλαγή γίνεται με τέτοιο ρυθμό οπού προκαλεί την ψευδαίσθηση ότι πολλές διαδικασίες εκτελούνται την ίδια στιγμή. Σε ένα υπολογιστικό σύστημα με επεξεργαστή πολλών πυρήνων είναι δυνατή η πραγματική ταυτόχρονη εκτέλεση περισσοτέρων της μιας διαδικασίας, αναλαμβάνοντας ο κάθε πυρήνας από μια διαδικασία. Το time-sharing εξακολουθεί να ισχύει και σε αυτή την περίπτωση ώστε να είναι δυνατό ο κάθε πυρήνας να αναλαμβάνει περισσότερες από μια διαδικασίες.



Εικόνα 1.3: Οι διάφορες καταστάσεις των διαδικασιών, με τα βέλη να δείχνουν τις δυνατές μεταβάσεις μεταξύ των καταστάσεων

Στην *Εικόνα 1.3* βλέπουμε τις διάφορες καταστάσεις στις οποίες επέρχονται οι διαδικασίες και οι οποίες καθορίζουν το πώς μια διαδικασία χειρίζεται από το kernel[2] του λειτουργικού συστήματος[3]. Όταν μια διαδικασία δημιουργείτε, χρειάζεται να περιμένει για το δρομολογητή διαδικασιών(*process scheduler*) για να τη θέσει σε κατάσταση αναμονής και να τη φορτώσει στη κύρια μνήμη από τις δευτερεύουσες συσκευές αποθήκευσης, όπως είναι ο σκληρός δίσκος. Αφού η διαδικασία ανατεθεί σε έναν επεξεργαστή, η κατάσταση της αλλάζει σε *running* όπου ο επεξεργαστής εκτελεί τις εντολές της. Αν η διαδικασία χρειάζεται να περιμένει για κάποιο πόρο τότε μπαίνει στη κατάσταση *blocked* μέχρι να απελευθερωθεί ο πόρος, οπότε και ξανά επιστρέφει στη κατάσταση *waiting*. Όταν η διαδικασία τελειώσει ή τερματιστεί από το λειτουργικό σύστημα τότε μεταφέρεται στη κατάσταση τερματισμού *(terminated)* όπου και περιμένει να διαγραφεί από τη κύρια μνήμη.

Μία άλλη έννοια παραπλήσια με τη process είναι η task. Ένα task είναι μια ομάδα εντολών προγράμματος φορτωμένες στην κύρια μνήμη. Το task έχει το νόημα

[2] **Kernel:** είναι το κύριο μέρος ενός λειτουργικού συστήματος το οποίο είναι υπεύθυνο για τη διαχείριση των πόρων του συστήματος και κάνει εφικτή την επικοινωνία μεταξύ του hardware και του software.

[3] **Λειτουργικό Σύστημα**: λειτουργεί ως διεπαφή του χρήστη με το σύστημα.

μιας εφαρμογής πραγματικού χρόνου, ενώ η process χρειάζεται χώρο (*memory*) και χρόνο εκτέλεσης. Με τις έννοιες multiprocessing και multitasking δηλώνουμε την δυνατότητα, πραγματικά ή εικονικά, ταυτόχρονης εκτέλεσης διαφόρων processes και tasks. Με το πέρασμα του χρόνου ο όρος time-sharing αντικαταστάθηκε με τον όρο multitasking.

1.3 Επεξεργαστές πολλαπλών πυρήνων

Στην προσπάθεια παράλληλης εκτέλεσης όλο και περισσότερων εντολών ανά κύκλο συχνότητας ενός επεξεργαστή, η λύση των επεξεργαστών με πολλαπλούς Ε.Π (Επεξεργαστικούς Πυρήνες- processor cores) έχει επικρατήσει τα τελευταία χρόνια στο χώρο των επεξεργαστών των οικιακών Η/Υ έναντι των άλλων λύσεων. Έτσι οι επεξεργαστές των 2 δύο κυρίαρχων εταιριών INTEL και AMD σε αυτό τον χώρο, όπως οι σειρές Core 2 Duo, Core I3, Core I5. Core I7 και Athlon II, Phenom, Phenom II, αντίστοιχα, έχουν πολλαπλούς Ε.Π. Η σειρά Core 2 της INTEL έχει επεξεργαστές με 2 ή 4 όμοιους 64bit υπερβαθμωτούς[4] (super scalar) Ε.Π όπου ανά δύο πυρήνες υπάρχει κοινή ΚΜ2 (κρυφή μνήμη 2 - Level 2 Cache). Η σειρά Phenom της AMD έχει επεξεργαστές με 3 ή 4 όμοιους 64bit υπερβαθμωτούς Ε.Π με κοινή ΚΜ3 (Level 3 cache). [1][2][4][3]

Σύμφωνα με τον νόμο του Moore [6] ο αριθμός των αντικειμένων (components) που χωρούν σε ένα κύκλωμα (circuit) αυξάνεται και η τιμή ενός αντικειμένου μειώνεται με τέτοιους ρυθμούς, έτσι ώστε ανά δύο περίπου χρόνια, ο αριθμός των αντικειμένων στα ολοκληρωμένα πυριτίου (silicon chip) διπλασιάζεται ενώ το κόστος παραμένει σταθερό. Η τάση αυτή συνεχίζεται ως σήμερα όπου κατασκευάζονται επεξεργαστές με τεχνολογία 45nm [2] και είναι πιθανόν να συνεχιστεί και στο μέλλον. Αν και η σμίκρυνση ευνοεί την απόδοση των τρανζίστορ καθώς έτσι μειώνεται ο χρόνος εναλλαγής κατάστασης (transistor switching time) , γίνονται εντονότερα προβλήματα όπως οι διαρροές στα τρανζίστορ και η μείωση απόδοσης των διασυνδέσεων. [7] Σε κάθε σμίκρυνση, η υλοποίηση πολύπλοκων ή με μεγάλη συχνότητα ΕΠ γίνεται δυσκολότερη. Η λύση της αύξησης Ε.Π σε ένα επεξεργαστή έχει βρεθεί να είναι πιο απλή στην υλοποίηση και προσφέρει μεγαλύτερη

4 Ένας υπερβαθμωτός επεξεργαστικός πυρήνας προσφέρει παραλληλισμό σε επίπεδο εντολών.

υπολογιστική ισχύς στον ίδιο αριθμό τρανζίστορ από ότι η αύξηση των εντολών που εκτελούνται παράλληλα σε επίπεδο εντολών σε ένα Ε.Π, υπερβαθμωτός, και από ότι η αύξηση των νημάτων που εκτελούνται ταυτόχρονα σε ένα Ε.Π (*ΣΠΝ- Synchronous Multi-Threading, SMT*). [21][22][8]

Από την μεριά του λογισμικού, ένα σύστημα με επεξεργαστή πολλαπλών όμοιων ΕΠ εμφανίζεται σαν ένα σύστημα πολλαπλών όμοιων επεξεργαστών με κοινή μνήμη, συμμετρική πολύ-επεξεργασίας (*ΣΠΕ - Symmetric-Multi-processing, SMP*). Όπως και για την χρήση συστημάτων ΣΠΕ έτσι και για τους πολύ-ΕΠ (επεξεργαστές πολλαπλών ΕΠ- Chip Multi-Processor) απαιτείται υποστήριξη από το ΛΣ (*Λειτουργικό Σύστημα- Operating System, OS*) για την χρήση όλων των ΕΠ. Για την βέλτιστη χρήση των πόρων των πολύ-ΕΠ τα προγράμματα πρέπει να είναι γραμμένα με παράλληλο προγραμματισμό. Δημοφιλή ΛΣ για οικιακούς Η/Υ όπως τα Windows XP/Vista και Linux υποστηρίζουν πολύ- ΕΠ και έχουν την δυνατότητα να κατανέμουν νήματα και διεργασίες στους διαθέσιμους ΕΠ. Ένα πρόγραμμα για να εκμεταλλευτεί τους διαθέσιμους Ε.Π πρέπει να είναι γραμμένο έτσι ώστε να μέρος των εντολών να εκτελούνται σε διαφορετικά νήματα ή διεργασίες. [9][22][10][11][16]

Υπάρχουν δύο μοντέλα παράλληλου προγραμματισμού με βάση το τρόπο πρόσβασης στα δεδομένα. Το μοντέλο της κατανεμημένη μνήμης όπου οι ΕΠ για να μοιραστούν δεδομένα, ο προγραμματιστής πρέπει ρητά να ορίσει την μεταξύ τους επικοινωνία. Βιβλιοθήκες αυτού του μοντέλου είναι οι MPI, PVM και SHMEM. Το άλλο μοντέλο είναι της κοινής μνήμης όπου βασίζεται στον προγραμματισμό βασισμένο στα νήματα, όπως Posix Threads, και σε επεκτάσεις μεταγλωττιστών όπως το OpenMP. Το τελευταίο μοντέλο είναι πιο κατάλληλο για χρήση με επεξεργαστές πολλαπλών επεξεργαστών λόγω της κοινής μνήμης.[12][13][5][14][15]

Σήμερα η πλειοψηφία των επεξεργαστών έχουν από 2 έως 4 υπερβαθμωτούς 64bit ΕΠ με συχνότητες που κυμαίνονται από 1,5GHz έως 4GHz . Η τάση αύξησης της συχνότητας με στόχο την μεγαλύτερη υπολογιστική ισχύς έχει φθαρεί τα τελευταία χρόνια και δίνει την θέση της στην αύξηση παράλληλης εκτέλεσής νημάτων και διεργασιών ανά επεξεργαστή. Αυτό επιτυγχάνεται κυρίως με την αύξηση των ΕΠ σε ένα επεξεργαστή. Παρακάτω υπάρχει ένας πίνακας με κάποια χαρακτηριστικά κάποιων σειρών γενικής χρήσης επεξεργαστών με είτε πολλούς ΕΠ είτε με ΣΠΝ [17][1][2][18]

Χαρακτηριστικά	Intel Pentium 4 HT	Intel Pentium D	Intel Core 2 Duo/Quad	Intel Core I3	Intel Core I5	Intel Core I7	AMD Athlon X2	AMD Phenom X
ΕΠ	1	2	2-4	2	2-4	4-6	2	3-4
Εύρος εντολών ανά ΕΠ	3	3	4	-	-	-	3	3
Νήματα ανά ΕΠ	2	1-2 (Extreme Edition)	1	4	4	8-12	1	1
Συχνότητα	2.8Ghz - 3.8Ghz	2.66Ghz - 3.6Ghz	1.4Ghz - 3.33Ghz	2.93Ghz – 3.33Ghz	2.40Ghz – 3.60Ghz	2.53Ghz – 3.3Ghz	1.9Ghz – 3.2Ghz	1.8Ghz – 2.6Ghz
ΚΜ1 (Δεδομένα)	16kB	16kB/ΕΠ	32kB/ΕΠ	-	-	-	64kB/ΕΠ	64kB/ΕΠ
ΚΜ1 (Εντολές)	12kB	12kB/ΕΠ	32kB/ΕΠ	-	-	-	64kB/ΕΠ	64kB/ΕΠ
ΚΜ2	1MB-2MB	1MB-2MB / ΕΠ	2MB-6MB / ζεύγος ΕΠ	128kb/ΕΠ	256kb/ΕΠ	256kb/ΕΠ	512kB-1MB /ΕΠ	512kB/ΕΠ
ΚΜ3	-	-	-	4MB Smart cache	3-8MB Smart cache	8-12MB Smart cache	-	2MB
Επικοινωνία ΕΠ	-	Εξωτερικά	ΚΜ2/ ζεύγος ΕΠ – Εξωτερικά	Εσωτερικά	Εσωτερικά	Εσωτερικά	Εσωτερικά	ΚΜ3
Στάδια σωληνώσεων	31	31	14	-	-	-	12	12
Έτος διάθεσης	2003	2005	2006	2010	2010	2010	2005	2007

Πίνακας 1-1.1: Χαρακτηριστικά πολύ-ΕΠ

1.4 Αρχιτεκτονική πολλαπλών πυρήνων

Για οικιακούς Η/Υ και όχι μόνο, μέχρι στιγμής έχουν εμφανιστεί διάφορες αρχιτεκτονικές επεξεργαστών πολλαπλών όμοιων ΕΠ που διαφέρουν στην επικοινωνία μεταξύ των ΕΠ ή αλλιώς πως διατηρούν συνοχή μεταξύ των κρυφών μνημών. Δηλαδή αν ***n*** νήματα μιας διεργασίας που εκτελούνται σε ***n*** ΕΠ ενός επεξεργαστή, χρησιμοποιούν τα ίδια δεδομένα και ένα νήμα τροποποιήσει τα κοινά δεδομένα πως θα ενημερωθούν οι υπόλοιποι ***n***-1 ΕΠ. Στην μια περίπτωση η επικοινωνία γίνεται εκτός επεξεργαστή. Στις άλλες περιπτώσεις η επικοινωνία γίνεται μέσα στον επεξεργαστή είτε μέσω μιας κοινής κρυφής μνήμης, είτε μέσω ενός εσωτερικού διαύλου και τέλος είτε με συνδυασμό των προηγούμενων δύο. [19][23][20]

1.4.1 Εξωτερική επικοινωνία μεταξύ των ΕΠ ενός επεξεργαστή.

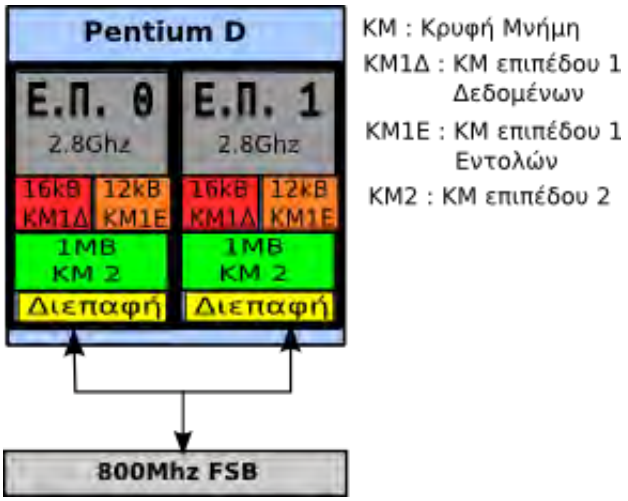
Μια από τις αρχιτεκτονικές είναι αυτή στην οποία η επικοινωνία μεταξύ των ΕΠ γίνεται εξωτερικά σε σχέση με τον επεξεργαστή και είναι συνήθως 2 επεξεργαστές στην ίδια συσκευασία. Σε περίπτωση αλλαγής δεδομένων σε έναν ΕΠ, οι υπόλοιποι που έχουν κοινά τα αλλαγμένα δεδομένα πρέπει να ανατρέξουν έξω από τον επεξεργαστή όπως και σε συστήματα πολλαπλών επεξεργαστών κοινής μνήμης (SMP).

Εικόνα 1.4: Πολύ-ΕΠ εξωτερικής επικοινωνίας

Αυτή η αρχιτεκτονική έχει το μεγάλο μειονέκτημα ότι στη καλύτερη περίπτωση η επικοινωνία μεταξύ των ΕΠ είναι αργή όσο αργή είναι η επικοινωνία μεταξύ ΕΠ από διαφορετικούς επεξεργαστές σε συστήματα πολλαπλών επεξεργαστών κοινής μνήμης (SMP). Τά πλεονέκτημά της είναι στην απλότητα σχεδίασης και υλοποίησης τους. Αφού πρόκειται

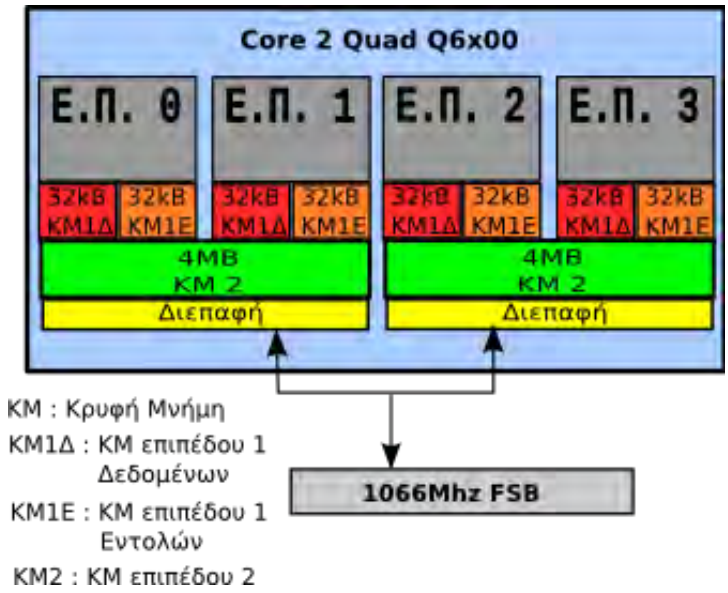
για επεξεργαστές που κατασκευάζονται ήδη και οι αλλαγές στους ΕΠ είναι ελάχιστες [19][20]

Αυτή η αρχιτεκτονική χρησιμοποιήθηκε από την Intel στους πρώτους της επεξεργαστές με διπλό ΕΠ. Συγκεκριμένα στον χώρο των οικιακούς Η/Υ, η σειρά "Pentium D" ήταν βασισμένη σε αυτή την αρχιτεκτονική, ενώ αντίστοιχα στο χώρο τον διακομιστών οι πρώτοι "Xeon" με διπλό πυρήνα. Η επικοινωνία μεταξύ των δύο ΕΠ γίνεται μέσω του εξωτερικού δίαυλου Front Side Bus(FSB) του οποίου συχνότητα είναι 800Mhz ενώ των επεξεργαστών Pentium D αρχίζει από τα 2,8Ghz.[18][5]



Εικόνα 1.5: Αρχιτεκτονική εξωτερικής επικοινωνίας του Pentium D

Επίσης χρησιμοποιήθηκε και στους πρώτους επεξεργαστές της Intel με 4 ΕΠ, "Core 2 Quad" με ΕΠ Kentsfield (Q6600, Q6700). Εδώ δύο επεξεργαστές διπλού ΕΠ με κοινή κρυφή μνήμη ενώθηκαν σε ένα επεξεργαστή. Η επικοινωνία μεταξύ των δύο ζευγών ΕΠ γίνεται μέσου του FSB ενώ μεταξύ των δύο ΕΠ του ίδιου ζεύγους γίνεται μέσω της κοινής επιπέδου 2 κρυφής μνήμης.[24][26]



Εικόνα 1.6: Intel Core 2 Quad

1.4.2 Εσωτερική επικοινωνία - Αποκλειστική κρυφή μνήμη

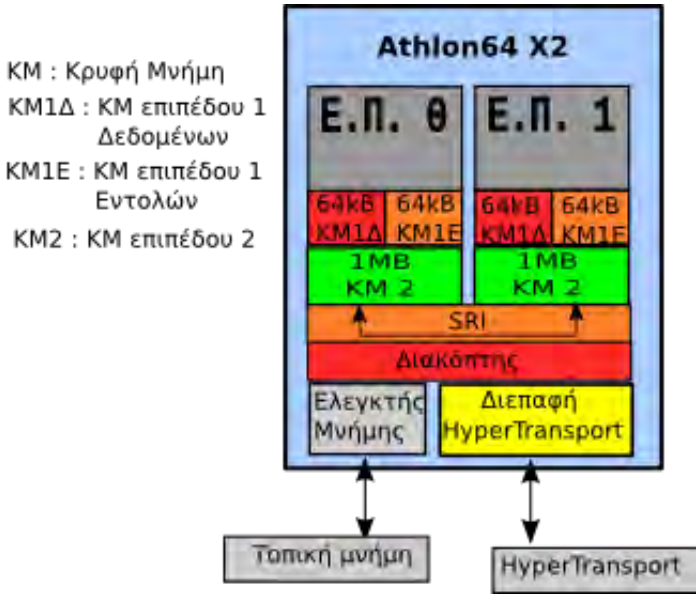
Στην αρχιτεκτονική αυτή, η επικοινωνία μεταξύ των ΕΠ ενός επεξεργαστή γίνεται εσωτερικά στον επεξεργαστή, στην ταχύτητα του επεξεργαστή, μέσω ενός εσωτερικού δίαυλου ή κάποιας διεπαφής ενώ δεν υπάρχει κοινή μνήμη μεταξύ των ΕΠ.

Εικόνα 1.7: Σχεδιάγραμμα εσωτερικής επικοινωνίας

Τα πλεονεκτήματα αυτής της αρχιτεκτονικής έναντι στην αρχιτεκτονική εξωτερικής επικοινωνίας είναι ότι η επικοινωνία γίνεται στην ταχύτητα του επεξεργαστή αντί του εξωτερικού διαύλου και υπάρχει μικρότερη καθυστέρηση στην ενημέρωση των κρυφών μνημών για την διατήρηση της συνοχής. Όμως έχει το μειονέκτημα ότι τα ίδια κοινά δεδομένα επαναλαμβάνονται στις κρυφές μνήμες των ΕΠ που τα χρησιμοποιούν,

σπαταλώντας έτσι πόρους και πρέπει να γίνεται ενημέρωση της κρυφής μνήμης του κάθε εμπλεκόμενου ΕΠ. [19][20]

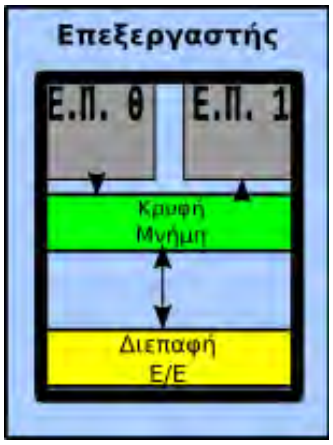
Αυτή η αρχιτεκτονική χρησιμοποιείται σε επεξεργαστές με δύο "*K8*" ΕΠ της AMD, σειρές Athlon64 X2 και Opteron με 2 πυρήνες. Η επικοινωνία αυτών γίνεται μέσω του "System Request Interface" που συνδέεται με την ΚΜ2 κάθε ΕΠ. Επίσης ενσωματωμένο στον επεξεργαστή βρίσκεται και ο ελεγκτής μνήμης και η διεπαφή για την/τις ζεύξεις HyperTranport που χρησιμοποιείται για Ε/Ε..[17][19][29]



Εικόνα 1.8: Αρχιτεκτονική AMD Athlon64 X2

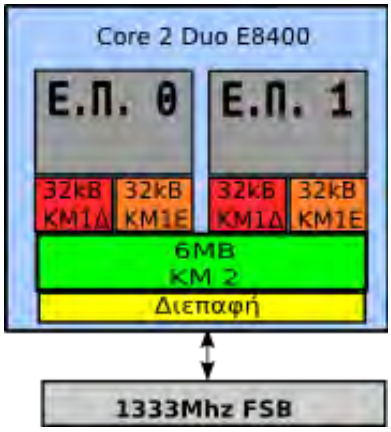
1.4.3 Κοινή κρυφή μνήμη

Στην αρχιτεκτονική κοινής κρυφής μνήμης, υπάρχει πάντα συνοχή της κρυφής μνήμης, στο επίπεδο που είναι κοινή, χωρίς την ανάγκη επιπλέον ενημερώσεων μέσω εσωτερικών ή εξωτερικών διαύλων ή ζεύξεων και άρα χωρίς καθυστερήσεις. Η κάθε αλλαγή που γίνεται στην κοινή κρυφή μνήμη από ένα ΕΠ είναι άμεσα προσβάσιμη από όλους τους ΕΠ που μοιράζονται αυτήν την κρυφή μνήμη. Επίσης άλλο πλεονέκτημα είναι ότι στο επίπεδο που η κρυφή μνήμη είναι κοινή τα κοινά δεδομένα μιας πολύ-νηματικής διεργασίας υπάρχουν μια φορά μόνο. Κοινές κρυφές μνήμες είναι συνήθως αυτές του επιπέδου 2 ή 3 (Level 2 or 3 Cache). [19][20]



Εικόνα 1.9: Άμεση επικοινωνία μέσω κοινής μνήμης

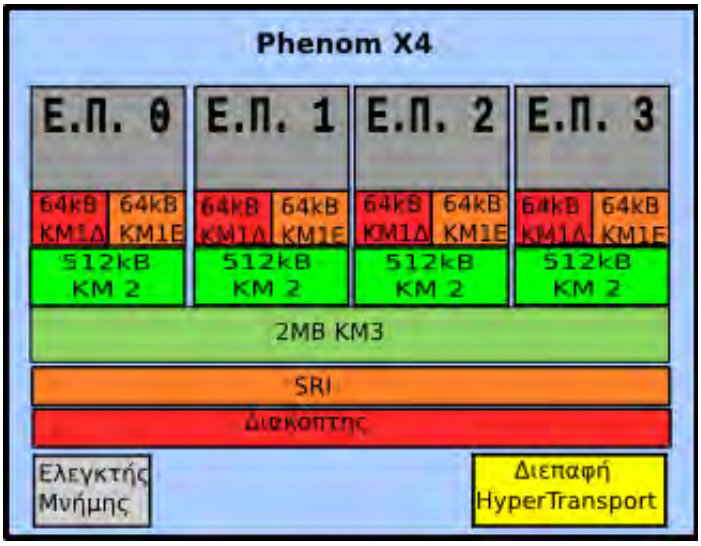
Κοινή κρυφή μνήμη στο δεύτερο επίπεδο έχουν οι επεξεργαστές της σειράς Core 2 Duo της Intel και ανά ζεύγος οι Core 2 Quad. Έχοντας αποκλειστική μόνο μια μικρή πρώτου επιπέδου η επικοινωνία μεταξύ των ΕΠ ενός ζεύγους γίνεται άμεσα με την ελάχιστη καθυστέρηση. Μεταξύ ΕΠ που δεν μοιράζονται ΚΜ υπάρχει πολύ μεγαλύτερη καθυστέρηση αφού η επικοινωνία μεταξύ τους γίνεται εξωτερικά. Για καλύτερη χρήση των πόρων το λειτουργικά σύστημα πρέπει να προτιμάει να μοιράζει τα νήματα μιας διεργασίας σε ΕΠ του ίδιου ζεύγους . Αν όμως γίνει κατανομή των νημάτων σε παραπάνω από δύο ΕΠ τότε θα υπάρχει κάποιο κόστος. [25][26][11][27][28]



Εικόνα 1.9: Αρχιτεκτονική Intel Core 2 Duo

Ενώ κοινή μνήμη τρίτου επιπέδου έχουν οι επεξεργαστές της σειράς Phenom και Opteron με ΕΠ "Barcelona" της AMD, με 3 ή 4 πυρήνες. Εδώ κάθε ΕΠ έχει αποκλειστική

κρυφή μνήμη πρώτου και δεύτερου επιπέδου. Σε σύγκριση με κοινές ΚΜ δευτέρου επιπέδου η χρήση του τρίτου επιπέδου ως κοινή ΚΜ έχει το μειονέκτημα του επιπλέον βήματος ή της επιπλέον καθυστέρησης αφού η ΚΜ2 είναι πιο κοντά στον ΕΠ. Η χρήση όμως μιας κοινής ΚΜ από όλους τους ΕΠ ενός επεξεργαστή βοηθάει στην καλύτερη κατανομή των νημάτων μιας διεργασίας σε όλους τους ΕΠ του επεξεργαστή χωρίς να υπάρχει κόστος.[1][29][11][27]



Εικόνα 1.10: AMD Phenom X4

1.5 Πλεονεκτήματα χρήσης και απόδοση επεξεργαστών πολλαπλών ΕΠ

Η αύξηση των ΕΠ σε επεξεργαστή είναι ο πιο δημοφιλής τρόπος αύξησης εκτέλεσης εντολών ανά κύκλο συχνότητας σε ένα επεξεργαστή σε σχέση με την αύξηση των εντολών που δέχεται ένας ΕΠ, υπερβαθμωτός, ή των νημάτων που εκτελούνται συγχρόνως ανά ΕΠ, ΣΠΝ. Οι κύριοι λόγοι είναι τα πλεονεκτήματα που προσφέρει αυτή η λύση όσο αφορά την κατασκευή, τα φυσικά και οικονομικά μεγέθη και κάτω από κάποιες συνθήκες την απόδοση. Σύμφωνα με άρθρο των L. Hammond, B. Nayfeb και K. Olukotun[27], με προσομοίωση, ένας επεξεργαστής με πολλαπλούς απλούς ΕΠ έχει πολύ καλύτερη απόδοση από ότι ένας επεξεργαστής με ένα υπερβαθμωτό ΕΠ, στις περιπτώσεις που η εφαρμογή είχε διασπαστεί σε νήματα ή σε διεργασίες. Στις ίδιες περιπτώσεις η απόδοση ήταν παρόμοια με ενός επεξεργαστή ενός πολύπλοκου ΣΠΝ ΕΠ. Συγκεκριμένα το μειονέκτημα ενός υπερβαθμωτού ΕΠ είναι ότι ενεργεί σε επίπεδο εντολών. Άρα η απόδοση του εξαρτάται από το πόσο ανεξάρτητες είναι οι εντολές που έχει δεχθεί και πόσο προβλέψιμη είναι η κάθε

διακλάδωση. Όσο μεγαλώνει ο αριθμός εντολών που μπορεί να δεχθεί τόσο μεγαλώνει και το κόστος ανίχνευσης παράλληλων εντολών και πρόβλεψη διακλαδώσεων. Αντίθετα, οι πολύ-ΕΠ ή ένας ΣΠΝ ΕΠ ενεργούν σε επίπεδο νημάτων και διεργασιών όπου τα νήματα ή/και διεργασίες είναι ήδη καθορισμένα.

Σε ένα ΣΠΝ ΕΠ κάποιοι πόροι του, όπως η ΚΜ1, μοιράζονται από όλα τα νήματα που εκτελούνται συγχρόνως και κάποιους από αυτούς τους κοινούς πόρους ίσως να μπορούν να χρησιμοποιηθούν αποκλειστικά από ένα νήμα σε μια χρονική στιγμή. Αυτό έχει ως αποτέλεσμα, στην περίπτωση της ΚΜ1, ότι πολλά νήματα και διεργασίες με διαφορετικά δεδομένα πρέπει να μοιράζονται την μικρή και πιο γρήγορη ΚΜ1 προκαλώντας αρκετές αστοχίες και αναζητήσεις σε άλλα πιο αργά επίπεδα μνήμης μειώνοντας έτσι την απόδοση του ΣΠΜ ΕΠ. Όπως επίσης μειώνεται η απόδοση του όταν ένα νήμα ή διεργασία μονοπωλήσει έναν ή περισσότερους κοινούς αποκλειστικούς πόρους και υπάρχουν άλλα νήματα που περιμένουν να τους χρησιμοποιήσουν. Αντίθετα σε ένα επεξεργαστή πολύ-ΕΠ οι διεργασίες και τα νήματα κατανέμονται στους ΕΠ και κάθε ΕΠ έχει την δικιά του ΚΜ1 και τους δικούς του πόρους. επιτρέποντας τις διεργασίες ή νήματα να εκτελούνται αδιάκοπα ακόμα και αν ένα νήμα ή διεργασία μονοπωλεί τους πόρους ενός ΕΠ. [30][31][32][22][8][27]

Χαρακτηριστική είναι η περίπτωση των επεξεργαστών Pentium 4 με ένα ΣΠΝ ΕΠ, HyperThreading, έναντι των επεξεργαστών Pentium D με δύο σχεδόν ίδους πιο αργούς χωρίς ΣΠΝ ΕΠ. Σύμφωνα με δοκιμές που έγιναν στην AnandTech σε Windows και σε Linux οι επεξεργαστές με διπλούς ΕΠ είχαν πολύ καλύτερη απόδοση σε σχέση με τους ΣΠΝ ΕΠ και ειδικά με τους επεξεργαστές μονού ΕΠ, στην εκτέλεση πολύ-νηματικών ή πολύ-διεργασιακών εφαρμογών και στην εκτέλεση πολλών εφαρμογών ταυτόχρονα. Ακόμα και αν οι επεξεργαστές με έναν ΕΠ ήταν πιο γρήγοροι όσο αφορά την συχνότητα/αρχιτεκτονική και στην εκτέλεση μονού νήματος/διεργασίας συνθετικού τεστ. [4][25]

2 Κρυπτογράφηση

2.1 Κρυπτογραφικοί Αλγόριθμοι

Η επιθυμία προστασίας του περιεχομένου μηνυμάτων οδήγησε στην επινόηση και χρήση κρυπτογραφικών τεχνικών και συστημάτων τα οποία επιτρέπουν το μετασχηματισμό μηνυμάτων ή δεδομένων κατά τέτοιον τρόπο ώστε να είναι αδύνατη η υποκλοπή του περιεχομένου τους κατά τη μετάδοση ή αποθήκευση τους, και βεβαίως, την αντιστροφή του μετασχηματισμού. Η διαδικασία μετασχηματισμού καλείται κρυπτογράφηση και η αντίστροφη της, αποκρυπτογράφηση.

Η συνάρτηση ή το σύνολο των κανόνων, στοιχείων και βημάτων που καθορίζουν την κρυπτογράφηση και την αποκρυπτογράφηση ονομάζεται κρυπτογραφικός αλγόριθμος. Η υλοποίηση του κρυπτογραφικού αλγόριθμου καλείται κρυπτογραφικό σύστημα. Μερικές φορές, ο κρυπτογραφικός αλγόριθμος καλείται και κωδικοποιητής (cipher). Πρωτόκολλα που χρησιμοποιούν κρυπτογραφικούς αλγόριθμους καλούνται κρυπτογραφικά πρωτόκολλα. Επειδή η αποθήκευση μπορεί να θεωρηθεί ως μετάδοση στη διάσταση του χρόνου, με τον όρο μετάδοση θα αναφέρεται και η μετάδοση και η αποθήκευση.

Οι κρυπτογραφικοί αλγόριθμοι χρησιμοποιούν κατά κανόνα (κρυπτογραφικά) κλειδιά, η τιμή των οποίων επηρεάζει την κρυπτογράφηση και την αποκρυπτογράφηση. Το σύνολο των δυνατών τιμών των κλειδιών λέγεται πεδίο τιμών αυτών. Υπάρχουν δυο κατηγορίες κρυπτογραφικών αλγόριθμων που καθορίζουν και τις δύο κατηγορίες κρυπτογραφικών αλγορίθμων. Τους συμμετρικούς και τους ασύμμετρους αλγόριθμους.

Οι συμμετρικοί αλγόριθμοι χρησιμοποιούν το ίδιο κλειδί και για την κρυπτογράφηση και για την αποκρυπτογράφηση, και για το λόγο αυτό καλούνται επίσης, αλγόριθμοι μυστικού κλειδιού ή αλγόριθμοι μονού κλειδιού. Οι ασύμμετροι αλγόριθμοι χρησιμοποιούν ένα ζεύγος κρυπτογραφικών κλειδιών. Το δημόσιο κλειδί για την κρυπτογράφηση και το ιδιωτικό κλειδί για την αποκρυπτογράφηση.

Τα συστήματα υπολογιστών εκτίθενται σε διάφορους κινδύνους , όπως είναι η μη εξουσιοδοτημένη πρόσβαση των δεδομένων που περιέχει. Η λύση ώστε να

διασφαλιστούν αυτά τα δεδομένο όσο το δυνατόν περισσότερο, είναι η κρυπτογράφηση αυτών με κάποιον από τους πολλούς αλγόριθμους κρυπτογράφησης.

Κρυπτογράφηση (*encryption*) ονομάζεται η διαδικασία μετασχηματισμού ενός μηνύματος σε μία ακατανόητη μορφή με την χρήση κάποιου κρυπτογραφικού αλγορίθμου ούτως ώστε να μην μπορεί να διαβαστεί από κανέναν εκτός του νόμιμου παραλήπτη. Η αντίστροφη διαδικασία όπου από το κρυπτογραφημένο κείμενο παράγεται το αρχικό μήνυμα ονομάζεται αποκρυπτογράφηση (*decryption*). Κρυπτογραφικός αλγόριθμος (cipher) είναι η μέθοδος μετασχηματισμού δεδομένων σε μία μορφή που να μην επιτρέπει την αποκάλυψη των περιεχομένων τους από μη εξουσιοδοτημένα μέρη. Κατά κανόνα ο κρυπτογραφικός αλγόριθμος είναι μία πολύπλοκη μαθηματική συνάρτηση, Αρχικό κείμενο (*plaintext*) είναι το μήνυμα το οποίο αποτελεί την είσοδο σε μία διεργασία κρυπτογράφησης. Κλειδί (*key*) είναι ένας αριθμός αρκετών bit που χρησιμοποιείται ως είσοδος στην συνάρτηση κρυπτογράφησης. Κρυπτογραφημένο κείμενο (*ciphertext*) είναι το αποτέλεσμα της εφαρμογής ενός κρυπτογραφικού αλγόριθμου πάνω στο αρχικό κείμενο.

Εικόνα 2.1: Τυπικό σύστημα κρυπτογράφησης-αποκρυπτογράφησης

Η κρυπτογράφηση είναι αναγκαία για την ασφαλή αποθήκευση και μετάδοση δεδομένων όπου η πρόσβαση σε αυτά είναι ανεπιθύμητη από μη εξουσιοδοτημένα πρόσωπα. Για παράδειγμα, στο τομέα της Ιατρικής, είναι αναγκαία για την εξασφάλιση του απορρήτου των προσωπικών πληροφοριών που περιέχονται στους ιατρικούς φακέλους όπως επίσης σε εφαρμογές τηλεματικής όπου ο χειρισμός κάποιου ιατρικού εξαρτήματος, που γίνεται από μακριά, πρέπει να εξασφαλίζεται ότι προέρχεται από εξουσιοδοτημένο πρόσωπο και δεν υπάρχει αλλοίωση των εντολών προς αυτό κατά την μετάδοση τους.

2.2 Κρυπτογράφηση μυστικού κλειδιού

Η κρυπτογράφηση μυστικού κλειδιού (secret key cryptography) βασίζεται στην χρήση μιας μυστικής πληροφορίας που ονομάζεται κλειδί για την κρυπτογράφηση και αποκρυπτογράφηση δεδομένων. Αν κάποιος θέλει να αποστείλει σε έναν τρίτο ένα κωδικοποιημένο μήνυμα, τότε χρησιμοποιεί το μυστικό κλειδί για να κρυπτογραφήσει την πληροφορία (στην ορολογία της κρυπτογραφίας η μη κρυπτογραφημένη πληροφορία αναφέρεται ως plaintext). Το αποτέλεσμα της κρυπτογράφησης είναι να προκύψει μια κωδικοποιημένη πληροφορία (που ονομάζεται ciphertext) και η οποία αποστέλλεται. Για να μπορέσει κάποιος να ανακτήσει την αρχική πληροφορία, θα πρέπει να γνωρίζει το ίδιο μυστικό κλειδί που χρησιμοποιήθηκε για την κωδικοποίηση της. Επειδή το ίδιο ακριβώς κλειδί χρησιμοποιείται τόσο για την κρυπτογράφηση όσο και αποκρυπτογράφηση των δεδομένων, η διαδικασία κρυπτογράφησης μυστικού κλειδιού ονομάζεται και συμμετρική κρυπτογράφηση (symmetric cipher).[33][34][35][36]

Εικόνα 2.2: Απλοποιημένο σχεδιάγραμμα κρυπτογράφησης-αποκρυπτογράφησης

Στην κρυπτογράφηση μυστικού κλειδιού, το ίδιο κλειδί χρησιμοποιείται και για την κρυπτογράφηση και για την αποκρυπτογράφηση των δεδομένων. Οι τρόποι κρυπτογράφησης μυστικού κλειδιού τυπικά χωρίζονται σε 2 κατηγορίες. Η πρώτη περιλαμβάνει διαδικασίες κρυπτογράφησης που εφαρμόζονται πάνω σε ένα μοναδικό bit (ή byte ή word) και υλοποιούν κάποιο μηχανισμό ανατροφοδότησης έτσι ώστε το κλειδί να αλλάζει συνεχώς. Για αυτό και ονομάζονται κρυπτογράφοι ροής (stream ciphers). Η δεύτερη κατηγορία (κρυπτογράφοι μπλοκ - block ciphers) αποτελείται από αλγορίθμους κρυπτογράφησης που λειτουργούν πάνω σε ομάδες δεδομένων κάθε χρονική στιγμή χρησιμοποιώντας το ίδιο κλειδί για κάθε ομάδα. Έτσι στην γενική περίπτωση, όταν το ίδιο μυστικό κλειδί χρησιμοποιείται, η ίδια ομάδα δεδομένων ενός plaintext θα κρυπτογραφηθεί στο ίδιο ciphertext όταν η κρυπτογράφηση γίνεται

με έναν αλγόριθμο κρυπτογράφησης μπλοκ αλλά σε διαφορετικό ciphertext όταν χρησιμοποιηθεί ένας κρυπτογράφος ροής.

Για τους κρυπτογράφους μπλοκ, έχουν επινοηθεί αρκετοί τρόποι λειτουργίας (modes) ώστε να βελτιωθούν κάποια χαρακτηριστικά τους όπως η ασφάλεια που προσφέρουν ή να γίνουν πιο κατάλληλοι για διάφορες εφαρμογές. Τέσσερις είναι οι κυριότεροι τρόποι λειτουργίας :

Electronic Codebook (ECB)

Αυτός ο τρόπος λειτουργίας είναι ο απλούστερος και ο πλέον προφανής. Το μυστικό κλειδί χρησιμοποιείται για την κρυπτογράφηση κάθε μπλοκ δεδομένων του plaintext. Κατά συνέπεια με την χρήση του ίδιου κλειδιού, το ίδιο plaintext μπλοκ θα μετατρέπεται πάντα στο ίδιο ciphertext μπλοκ. Είναι ο πλέον κοινός τρόπος λειτουργίας των κρυπτογράφων μπλοκ γιατί είναι ο απλούστερος και άρα ο πιο εύκολα υλοποιήσιμος και συνάμα ο πιο γρήγορος καθώς δεν χρησιμοποιείται κάποιου είδους ανατροφοδότηση. Μειονέκτημα του είναι ότι είναι ο πιο ευάλωτος τρόπος κρυπτογράφησης σε επιθέσεις τύπου brute-force (ως επίθεση brute-force θεωρείται η προσπάθεια εύρεσης του μυστικού κλειδιού με την εξαντλητική δοκιμή πιθανών κλειδιών).

Cipher Block Chaining (CBC)

Χρησιμοποιώντας την CBC λειτουργία, προστίθεται σε έναν κρυπτογράφο μπλοκ ένας μηχανισμός ανατροφοδότησης. Ο τρόπος αυτός λειτουργίας ορίζει ότι πρωτού να γίνει η κρυπτογράφηση ενός νέου μπλοκ plaintext, γίνεται XOR (αποκλειστικό-Ή) του μπλοκ αυτού και του ciphertext μπλοκ που μόλις πριν έχει παραχθεί. Με τον τρόπο αυτό, 2 ταυτόσημα μπλοκ plaintext δεν κρυπτογραφούνται ποτέ στο ίδιο ciphertext. Σε σχέση με τον ECB προσφέρεται μεγαλύτερη ασφάλεια, με κόστος όμως κυρίως στην ταχύτητα κρυπτογράφησης καθώς για να ξεκινήσει η επεξεργασία ενός μπλοκ plaintext είναι απαραίτητο να έχει ολοκληρωθεί πλήρως η κρυπτογράφηση του προηγούμενου μπλοκ. Αποτρέπεται έτσι η χρήση τεχνικών pipelining (software ή hardware) που μπορούν να επιταχύνουν την διαδικασία.

Cipher Feedback (CFB)

Ο τρόπος αυτός λειτουργίας επιτρέπει σε έναν κρυπτογράφο μπλοκ να συμπεριφερθεί σαν ένας κρυπτογράφος ροής. Αυτό είναι θεμιτό όταν πρέπει να κρυπτογραφούνται δεδομένα που μπορεί να έχουν μέγεθος μικρότερο από ένα μπλοκ. Παράδειγμα τέτοιας εφαρμογής μπορεί να είναι η διαδικασία κρυπτογράφησης ενός terminal session. Περιληπτικά, κατά την CFB λειτουργία χρησιμοποιείται ένας shift καταχωρητής στο μέγεθος του block μέσα στον οποίο τοποθετούνται τα δεδομένα προς κρυπτογράφηση. Όλος ο καταχωρητής κρυπτογραφείται και αυτό που προκύπτει είναι το ciphertext. Η ποσότητα των δεδομένων που μπαίνουν μέσα στον shift καταχωρητή καθορίζεται από την εφαρμογή.

Output Feedback (OFB)

Στόχος και αυτού του τρόπου λειτουργίας των μπλοκ κρυπτογράφων είναι να εξασφαλίσει ότι το ίδιο plaintext μπλοκ δεν μπορεί να παράγει το ίδιο ciphertext μπλοκ. Σε σχέση με το CBC, χρησιμοποιείται και εδώ ένας μηχανισμός ανατροφοδότησης παρόλα αυτά είναι εσωτερικός και ανεξάρτητος από τα plaintext και ciphertext δεδομένα.

Σημαντικοί αλγόριθμοι αυτής της κατηγορίας είναι οι DES (Data Encryption Standard), 3DES, DESX, ο AES (Advanced Encryption Standard), οι RC2, RC4, RC5 και IDEA (International Data Encryption Algorithm). Οι αλγόριθμοι της σειράς DES είναι οι πλέον χρησιμοποιούμενοι σήμερα αλγόριθμοι, αν και πλέον αντικαθιστούνται από τον AES. Επινοήθηκαν από την IBM την δεκαετία του '70 και υιοθετήθηκαν από το National Bureau of Standards (νυν NIST) των ΗΠΑ. Οι DES αλγόριθμοι χρησιμοποιούν κλειδιά μήκους 56 bits (ο 3DES και ο DESX επεκτείνουν κατάλληλα αυτόν τον αριθμό χρησιμοποιώντας περισσότερα κλειδιά) και επεξεργάζονται μπλοκ των 64 bits. Ο AES αλγόριθμος είναι το πρότυπο που καθιερώθηκε από το NIST ως διάδοχος του DES και πλέον αποτελεί τον προτεινόμενο αλγόριθμο κρυπτογράφησης για εφαρμογές υψηλής ασφάλειας. Οι αλγόριθμοι RC είναι αλγόριθμοι μεταβλητού κλειδιού από την RSA Security ενώ ο IDEA χρησιμοποιείται στο πρότυπο PGP (Pretty Good Privacy). [37][38]

2.3 Ο αλγόριθμος κρυπτογράφησης AES

Το πρότυπο κρυπτογράφησης AES (Advanced Encryption Standard) περιγράφει μια διαδικασία κρυπτογράφησης ηλεκτρονικής πληροφορίας βασισμένη στην λογική της κωδικοποίησης ομάδων δεδομένων με κάποιο μυστικό κλειδί. Έχει τυποποιηθεί από το NIST (National Institute of Technology) τον Νοέμβριο του 2001, αντικαθιστώντας το πρότυπο DES (Data Encryption Standard) και πλέον αποτελεί τον προτεινόμενο αλγόριθμο για εφαρμογές κρυπτογράφησης για τους λόγους που αναφέρθηκαν πιο πάνω .

Το πρότυπο AES περιγράφει μια συμμετρική μπλοκ διαδικασία κρυπτογράφησης μυστικού κλειδιού. Το πρότυπο υποστηρίζει την χρήση κλειδιών μήκους 128, 192 και 256 bits. Ανάλογα με το ποιο μήκος κλειδιού χρησιμοποιείται, συνήθως χρησιμοποιείται η συντόμευση AES-128, AES-192 και AES-256 αντίστοιχα. Ανεξάρτητα από το μήκος κλειδιού, ο αλγόριθμος επενεργεί πάνω σε μπλοκ δεδομένων μήκους 128 bits. Η διαδικασία κρυπτογράφησης είναι επαναληπτική. Αυτό σημαίνει ότι σε κάθε μπλοκ δεδομένων γίνεται μια επεξεργασία η οποία επαναλαμβάνεται έναν αριθμό από φορές ανάλογα με το μήκος κλειδιού. Κάθε επανάληψη ονομάζεται γύρος (round). Στον πρώτο γύρο επεξεργασίας ως είσοδος είναι ένα plaintext μπλοκ και το αρχικό κλειδί, ενώ στους γύρους που ακολουθούν ως είσοδος είναι το μπλοκ που έχει προκύψει από τον προηγούμενο γύρο καθώς και ένα κλειδί που έχει παραχθεί από το αρχικό με βάση κάποια διαδικασία που ορίζει ο αλγόριθμος. Το τελικό προϊόν της επεξεργασίας είναι το κρυπτογραφημένο μπλοκ (ciphertext). Το μπλοκ αυτό πρέπει να σημειωθεί ότι έχει ακριβώς το ίδιο μέγεθος (128 bits) με το plaintext μπλοκ.

Ο κρυπτογραφικός αλγόριθμος AES γνωστός και ως Rijndael αναπτύχτηκε από τους Vincent Rijmen και Joan Daemen κατά την πρόσκληση υποβολής προτάσεων το 1997 για νέο Προηγμένο Πρότυπο Κρυπτογράφησης (Advanced Encryption Standard - AES) από το NIST[5] που θα αντικαταστήσει τον DES (Data Encryption Standard) και θα πρέπει να αποτελεί κωδικοποιητή τμημάτων με συμμετρικό σύστημα κρυπτογράφησης, μήκους τμήματος 128bit και να υποστηρίζει κλειδιά μήκους 128bit 192bit και 256bit. Για την συμμετρική κρυπτογράφηση, αλλιώς και συμβατική, πρέπει να ισχύουν τα ακόλουθα:

[5] **NIST:** National Institute of Standards and Technology

- Απαιτείται η ύπαρξη ενός ισχυρού αλγορίθμου τέτοιο ώστε ο επιτιθέμενος να είναι αδύνατο να κρυπταναλύσει το κρυπτογράφημα ή να ανακαλύψει το κλειδί, ακόμη και αν κατέχει κάποια κρυπτογραφήματα μαζί με τα αντίστοιχα αρχικά μηνύματα, από τα οποία παράχθηκε καθένα από αυτά τα κρυπτογραφήματα.
- Ο πομπός και ο δέκτης πρέπει να έχουν παραλάβει τα αντίγραφα του μυστικού κλειδιού με ασφαλή τρόπο και να διαφυλάσσουν αυτό το μυστικό κλειδί σε ασφαλές μέρος.

Στα συστήματα συμμετρικής κρυπτογράφησης το κρυφό στοιχειό είναι το μυστικό κλειδί και όχι ο αλγόριθμος κρυπτογράφησης. Σχεδιαστικά, ο αλγόριθμος Rijndael δεν ακολουθεί την κλασική δομή Feistel, αλλά κάθε κύκλος λειτουργίας περιλαμβάνει τρείς όμοιους μετασχηματισμούς, με όρους ισότιμης αντιμετώπισης κάθε ξεχωριστού bit, γνωστούς ως επίπεδα (layers):

- Το επίπεδο γραμμικής ανάμιξης (linear mixing layer) επιτυγχάνει υψηλή διάχυση σε πολλαπλούς κύκλους.
- Το μη γραμμικό επίπεδο (non-linear layer) αφορά στην παράλληλη εφαρμογή S-boxes τα οποία εμφανίζουν εξαιρετικές μη γραμμικές ιδιότητες για το ενδεχόμενο χειρότερης περίπτωσης (optimum worst-case nonlinearity properties).
- Το επίπεδο πρόσθεσης κλειδιού (key addition layer) αφορά στη συσχέτιση του ενδιάμεσα προκύπτοντος αποτελέσματος με το υποκλειδί του κύκλου, με την πράξη XOR.

Το πρότυπο AES περιγράφει μια συμμετρική μπλοκ διαδικασία κρυπτογράφησης μυστικού κλειδιού. Το πρότυπο υποστηρίζει την χρήση κλειδιών μήκους 128, 192 και 256 bits. Ανάλογα με το ποιο μήκος κλειδιού χρησιμοποιείται, συνήθως χρησιμοποιείται η συντόμευση AES-128, AES-192 και AES-256 αντίστοιχα. Η διαδικασία κρυπτογράφησης είναι επαναληπτική. Αυτό σημαίνει ότι σε κάθε μπλοκ δεδομένων γίνεται μια επεξεργασία η οποία επαναλαμβάνεται 10,12 ή 14 κύκλους, ανάλογα με το μήκος του μυστικού κλειδιού. Κάθε κύκλος περιλαμβάνει τέσσερις μετασχηματισμούς SubBytes step, ShiftRows step, MixColumns step, AddRoundKey step. Ο SubBytes μετασχηματισμός εφαρμόζεται σε όλα τα bytes του τμήματος. Οι μετασχηματισμοί ShiftRows και MixColumns υποστηρίζουν τη γραμμική ανάμειξη των δεδομένων του τμήματος. Ο μετασχηματισμός AddRoundKey συσχετίζει τα

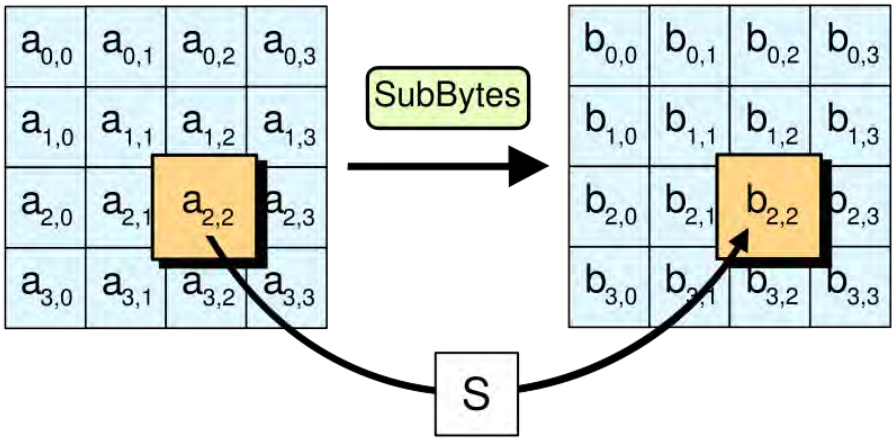
bytes του τμήματος με τα bytes των υποκλειδιών με την πράξη XOR. Επιπλέον, ο μετασχηματισμός AddRoundKey εκτελείται μια ακόμη φορά στη φάση αρχικοποίησης πριν τον πρώτο κύκλο, ενώ στον τελευταίο κύκλο παραλείπεται ο μετασχηματισμός MixColumns.[37][38][39][40]

Η υλοποίηση του πρότυπου AES με γλώσσα προγραμματισμού δίνεται συνοπτικά στο παρακάτω block κώδικα:

```
Cipher(byte in [4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
     byte state[4,Nb]

     state = in

     AddRoundKey(state, w[0, Nb-1])

     for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
     end for

     SubBytes(state)
     ShiftRows(state)
     AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

          out = state
end
```

Κείμενο 2.1: Συνοπτική περιγραφή του πηγαίου κώδικα του αλγορίθμου AES

Στο μετασχηματισμό SubBytes, κάθε byte στον πίνακα αλλάζει χρησιμοποιώντας ένα 8bit κουτί αντικατάστασης (substitution box, S-box). Αυτή η λειτουργία προσφέρει μη γραμμικότητα στο κρυπτογράφημα.



Εικόνα 2.3: Στο μετασχηματισμό SubBytes, κάθε byte της κατάστασης αντικαταστάτε από την ανάλογη εγγραφή σε έναν προκαθορισμένο 8bit πίνακα, *S*; $b_{ij} = S(a_{ij})$.

Το S-box παράγεται από τον υπολογισμό του αντίστροφου πολλαπλασιασμού για έναν δοσμένο αριθμό στο πεπερασμένο πεδίο Rijndael (τα μηδενικά, που δεν αντιστρέφονται, θέτονται ως μηδέν). Έπειτα ο αντίστροφος πολλαπλασιασμός μετασχηματίζεται με ένας γραμμικό μετασχηματισμό ακολοθούμενο από έναν γραμμικό συνδυασμό(affine transformation- $x \mapsto Ax + b$).

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Πίνακας 2.1

Όπου [x_0, ..., x_7] είναι ο αντίστροφος πολλαπλασιασμός ως διάνυσμα.

Ο αντίστροφος πολλαπλασιασμός υπολογίζεται από τον παρακάτω αλγόριθμο:

1. Αποθήκευση του αντίστροφου μετασχηματισμού του εισαγόμενου αριθμού σε 2 8bit προσωρινές μεταβλητές: *s* και *x*.

2. Εκτέλεση της πράξης XOR στην τιμή *x* με την τιμή *s*, αποθηκεύοντας το αποτέλεσμα στην μεταβλητή *x*.
3. Επανάληψη των βημάτων 2 και 3 για άλλες τρεις φορές. Τα βήματα 2 και 3 εκτελούνται συνολικά τέσσερεις φορές.
4. Η τιμή x θα έχει πλέον το αποτέλεσμα του πολλαπλασιασμού.

Εφόσον γίνει ο πολλαπλασιασμός της μήτρας που φαίνεται στον Πίνακα 2.1 εκτελείται η πράξη XOR με τον δεκαδικό αριθμό 99 (με τον δεκαεξαδικό 0x63, με τον δυαδικό αριθμό 1100011, και με την συμβολοσειρά bit 11000110 αντιπροσωπεύοντας τον αριθμό στο LSB[6] πρώτο σύμβολο).

Αυτό θα οδηγήσει στη παραγωγή του παρακάτω S-box που αναπαριστάται στο δεκαεξαδικό σύστημα

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Πίνακας 2.2

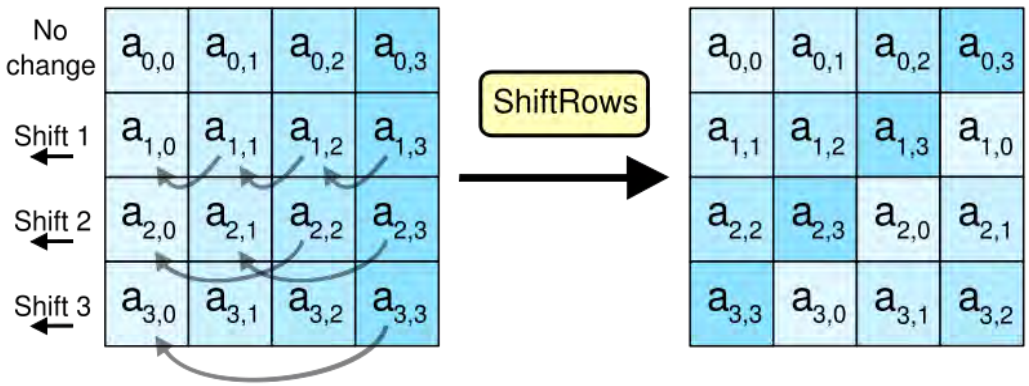
Οι στήλες προσδιορίζονται από τα ελάχιστης σημαντικότητας 4bit, και οι γραμμές προσδιορίζονται από τα μέγιστης σημαντικότητας 4bit.

[6] **LSB**: Less Significant Bit

Το Rijndael S-box σχεδιάστηκε ειδικά για είναι ανθεκτικό στην γραμμική και διαφορική κρυπτανάλυση[7]. Αυτό επιτεύχθηκε με την ελαχιστοποίηση της συσχέτισης(correlation) μεταξύ του γραμμικού μετασχηματισμού των input/output bits, και ταυτόχρονης ελαχιστοποίησης πιθανότητα μετάδοσης λογικής διαφοράς (difference propagation probability[8]).

Επιπροσθέτως, για την ενδυνάμωση του S-box έναντι των αλγεβρικών επιθέσεων, προστέθηκε ο μετασχηματισμός affine[9].

Στο μετασχηματισμό ShiftRows γίνεται κυκλική μετατόπιση των bytes σε κάθε γραμμή με ένα συγκεκριμένο offset. Για τον αλγόριθμο AES η πρώτη γραμμή μένει αμετάβλητη. Κάθε byte στην δεύτερη γραμμή μετατοπίζεται μια θέση προς τα αριστερά. Παρομοίως, στην τρίτη και τέταρτη γραμμή μετατοπίζονται κατά 2 και 3 θέσεις αντίστοιχα.



Εικόνα 2.4: Στον μετασχηματισμό ShiftRows γίνεται κυκλική μετατόπιση των bytes σε κάθε κατάσταση προς τα αριστερά.

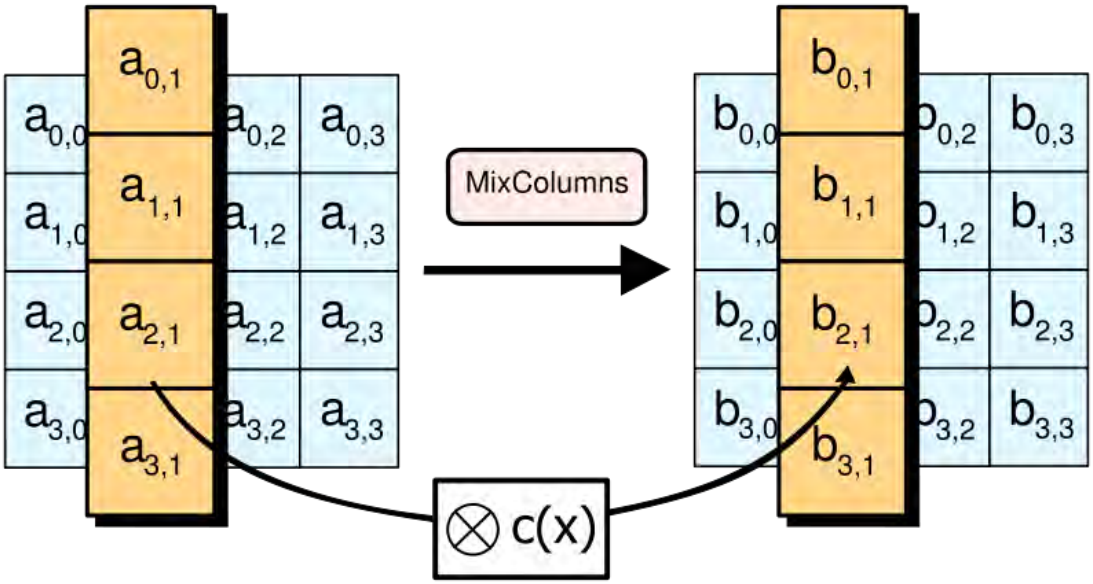
Στο μετασχηματισμό MixColumns, τα 4 byte κάθε στήλης κάθε κατάστασης συνδυάζονται χρησιμοποιώντας έναν αντίστροφο γραμμικό μετασχηματισμό. Η λειτουργία αυτή παίρνει ως είσοδο 4 byte και βγάζει ως έξοδο 4 bytes όπου κάθε byte επηρεάζει όλα τα byte της εξόδου. Ο μετασχηματισμός MixColumns μαζί με των

[7] **Κρυπτανάλυση:** είναι η χρησιμοποίηση τεχνικών ανάλυσης με στόχο την εύρεση του κλειδιού, του μηνύματος ή ενός ισοδύναμου αλγόριθμου που θα βοηθήσει στην ανάγνωση του (κρυφού) μηνύματος.

[8] αναφέρεται στη διαφορά του α και α*, η οποία υπολογίζεται με XOR πύλες (λογική διαφορά των bits) και πως αυτή επηρεάζει τη μετάδοση της διαφοράς και στη συνάρτηση h(a)

[9] **Affine:** είναι ο γραμμικός μετασχηματισμός μεταξύ δύο συνδεδεμένων πεδίων

ShiftRows προσδίδουν διάχυση στο κρυπτογράφημα. Κάθε στήλη ως πολυώνυμο και πολλαπλασιάζεται με το modulo x^4+1 με το $c(x) = 3x^3 + x^2 + x + 2$.



Εικόνα 2.5: Στον μετασχηματισμό MixColumns, κάθε στήλη πολλαπλασιάζεται με ένα δοσμένο πολυώνυμο *c(x)*

Ο πολλαπλασιασμός των στοιχείων γίνεται με τον πίνακα όπως φαίνεται και στην παραπάνω υλοποίηση.

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Πίνακας 2-1.3: Πολλαπλασιασμός των στοιχείων κατά την διαδικασία Mix Columns

Θα πρέπει να σημειωθεί ότι αυτή η υλοποίηση είναι ευπαθής σε επιθέσεις χρόνου[10].

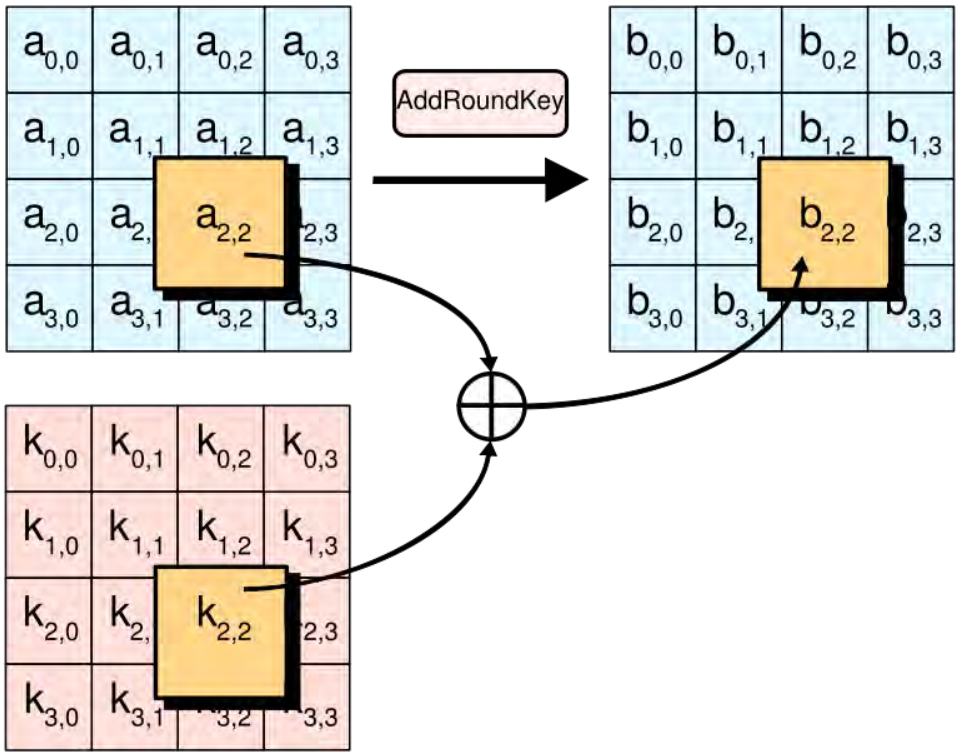
Ένα παράδειγμα για την υλοποίηση του μετασχηματισμού αυτού με γλώσσα προγραμματισμού δίνεται παρακάτω:

```
void gmix_column(unsigned char *r) {
        unsigned char a[4];
        unsigned char b[4];
   unsigned char c;
   unsigned char h;

        /* Το διάνυσμα 'a' είναι απλά μία αντιγραφή του
   διανύσματος     εισόδου 'r'
         * Το διάνυσμα 'b' έχει κάθε στοιχείο του διανύσματος
   'a' πολλαπλασιασμένο με το 2 στο πεδίο τιμών Rijndael's
   Galois
         * a[n] ^ b[n] είναι το στοιχείο n πολλαπλασιασμένο με
   το 3 στο πεδίο τιμών Rijndael's Galois*/

   for(c=0;c<4;c++) {
          a[c] = r[c];
          h = r[c] & 0x80; /* hi bit */
          b[c] = r[c] << 1;
          if(h == 0x80)
                 b[c] ^= 0x1b; /* Rijndael's Galois field */
   }
   r[0]=b[0]^a[3]^a[2]^b[1]^a[1]; /* 2*a0+a3+a2+3*a1 */
   r[1]=b[1]^a[0]^a[3]^b[2]^a[2]; /* 2*a1+a0+a3+3*a2 */
   r[2]=b[2]^a[1]^a[0]^b[3]^a[3]; /* 2*a2+a1+a0+3*a3 */
   r[3]=b[3]^a[2]^a[1]^b[0]^a[0]; /* 2*a3+a2+a1+3*a0 */
```

Κείμενο 2.2: Μετασχηματισμός Mix Coloumns με γλώσσα προγραμματισμού

[10] **Επιθέσεις χρόνου (Timing attacks):** Σε αυτό το είδος επιθέσεων ο επιτιθέμενος προσπαθεί να κρυπταναλύσει το σύστημα με βάση τον χρόνο εκτέλεσης του αλγορίθμου κρυπτογράφησης.

Στο μετασχηματισμό AddRoundKey το κάθε υποκλειδί συνδυάζεται με κάθε αντίστοιχη κατάσταση. Για κάθε κύκλο το υποκλειδί παράγεται από το κύριο κλειδί μέσω της λειτουργίας δρομολόγησης κλειδιού.



Εικόνα 2.5: Στον μετασχηματισμό AddRoundKey κάθε byte της κατάστασης συνδυάζεται με το αντίστοιχο byte του υποκλειδιού με την πράξη XOR.

Η λειτουργία δρομολόγησης κλειδιού χρησιμοποιεί 4 λειτουργίες, περιστροφή, Rcon, S-box και την επέκταση κλειδιού. Αρχικά γίνεται μια περιστροφική διαδικασία η οποία παίρνει μια λέξη 32-bit όπως για παράδειγμα η *2c4f8d6a*, στο δεκαεξαδικό σύστημα, και την περιστρέφει 8 bit προς τα αριστερά, οπότε παράγεται η λέξη *4f8d6a2c*. Ακολουθεί η λειτουργία Rcon η οποία έχει ως πεδίο τιμών το πεδίο τιμών Rijndael[11]

Στην πολυωνυμική μορφή το 2 είναι:

[11] **Πεδίο τιμών Rijndael:** Ανήκει στο πεδίο τιμών Galois. Το πεδίο αυτό αποτελείται από τα στοιχεία 2^k όπου το k παίρνει τις τιμές από 0 μέχρι 7 (συνολικά 8 όρους).

$$2 = 00000010 = 0x^7 + 0x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 1x + 0 = x$$

γίνεται υπολογισμός του rcon(i)=x(254+i) στο πεδίο $\mathbb{F}_{2^8}$ ή ισότιμα

$$\mathrm{rcon}(i) = x^{(254+i)} \mod x^8 + x^4 + x^3 + x + 1$$

Στο πεδίο $\mathbb{F}_2$

Για παράδειγμα, το rcon(1) = 1, το rcon(2) = 2, το rcon(3) = 4, και το rcon(9) = 27 και στο δεκαεξαδικό σύστημα είναι ο αριθμός 0x1b.

```
Rcon[255] = {
0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6,
0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33,
0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02,
0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab,
0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4,
0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d,
0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e,
0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25,
0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb,
0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36,
0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97,
0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72,
0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66,
0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,
0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61,
0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
0x74, 0xe8, 0xcb}
```

Ο πυρήνας της διαδικασίας δρομολόγησης είναι μία εσωτερική επανάληψη στην οποία γίνεται αντιγραφή της εισόδου που είναι μία λέξη 32-bit και ένας αριθμός επανάληψης **i** στην έξοδο όπου είναι και αυτή μία λέξη 32-bit. Γίνεται η περιστροφή τις εξόδου 8 bit προς τα αριστερά και εφαρμόζεται το S-box ξεχωριστά σε κάθε ένα από τα 4 bytes της εξόδου. Στο πρώτο byte της εξόδου εκτελείτε η πράξη XOR με την έξοδο από την λειτουργία Rcon. Η παραπάνω διαδικασία περιγράφεται με τον παρακάτω αλγόριθμο:

1. Τα πρώτα **n** bytes του κλειδιού (16 για 128-bit κλειδί, 24 για 192-bit κλειδί και 32 για 256-bit κλειδί) είναι απλά το κλειδί της κρυπτογράφησης.
2. Η τιμή της μεταβλητής **i** παίρνει την τιμή 1.
3. Για την δημιουργία των επόμενων **n** bytes του αναπτυγμένου κλειδιού γίνονται τα παρακάτω βήματα μέχρι την ύπαρξη 176 για 128-bit κλειδί, 208 για 192-bit κλειδί και 240 για 256-bit κλειδί, bytes του αναπτυγμένου κλειδιού.
    a. Για την δημιουργία 4 bytes του αναπτυγμένου κλειδιού:
        i. Δημιουργείται μια προσωρινή μεταβλητή **t** μεγέθους τεσσάρων bytes.
        ii. Θέτεται η τιμή των προηγούμενων τεσσάρων bytes του αναπτυγμένου κλειδιού στην μεταβλητή **t**.
        iii. Εκτελείτε ο πυρήνας της διαδικασίας δρομολόγησης στη μεταβλητή **t** με την τιμή **i** ως τιμή επανάληψης στη διαδικασία Rcon.
        iv. Αυξάνεται η τιμή του **i** κατά 1.
        v. Εκτελείτε η πράξη XOR ανάμεσα στην τιμή **t** και το τμήμα των 4 bytes **n** bytes πριν το καινούριο αναπτυγμένο κλειδί. Το αποτέλεσμα είναι τα επόμενα 4 bytes του αναπτυγμένου κλειδιού.
    b. Για την δημιουργία των επόμενων 12 bytes του αναπτυγμένου κλειδιού γίνονται τα παρακάτω τρεις φορές:
        i. Θέτεται η τιμή των προηγούμενων τεσσάρων bytes του αναπτυγμένου κλειδιού στην μεταβλητή **t**.

ii. Εκτελείτε η πράξη XOR ανάμεσα στην τιμή **t** και το τμήμα των 4 bytes **n** bytes πριν το καινούριο αναπτυγμένο κλειδί. Το αποτέλεσμα είναι τα επόμενα 4 bytes του αναπτυγμένου κλειδιού.

c. Αν το κλειδί, στο οποίο γίνεται η διαδικασία, είναι μεγέθους 256-bit, εκτελούνται τα παρακάτω για την δημιουργία των επόμενων 4 bytes:
   i. Θέτεται η τιμή των προηγούμενων τεσσάρων bytes του αναπτυγμένου κλειδιού στην μεταβλητή **t**.
   ii. Εκτελείται το S-box ξεχωριστά σε κάθε ένα από τα 4 bytes της μεταβλητής **t**.
   iii. Εκτελείτε η πράξη XOR ανάμεσα στην τιμή **t** και το τμήμα των 4 bytes 32 bytes πριν το καινούριο αναπτυγμένο κλειδί. Το αποτέλεσμα είναι τα επόμενα 4 bytes του αναπτυγμένου κλειδιού.

d. Αν το κλειδί, στο οποίο γίνεται η διαδικασία, είναι μεγέθους 128-bit εκτελούνται τα παρακάτω μια φορά, αν είναι 198-bit, 2 φορές και αν είναι 256-bit, 3 φορές:
   i. Θέτεται η τιμή των προηγούμενων τεσσάρων bytes του αναπτυγμένου κλειδιού στην μεταβλητή **t**.
   ii. Εκτελείτε η πράξη XOR ανάμεσα στην τιμή **t** και το τμήμα των 4 bytes **n** bytes πριν το καινούριο αναπτυγμένο κλειδί. Το αποτέλεσμα είναι τα επόμενα 4 bytes του αναπτυγμένου κλειδιού.

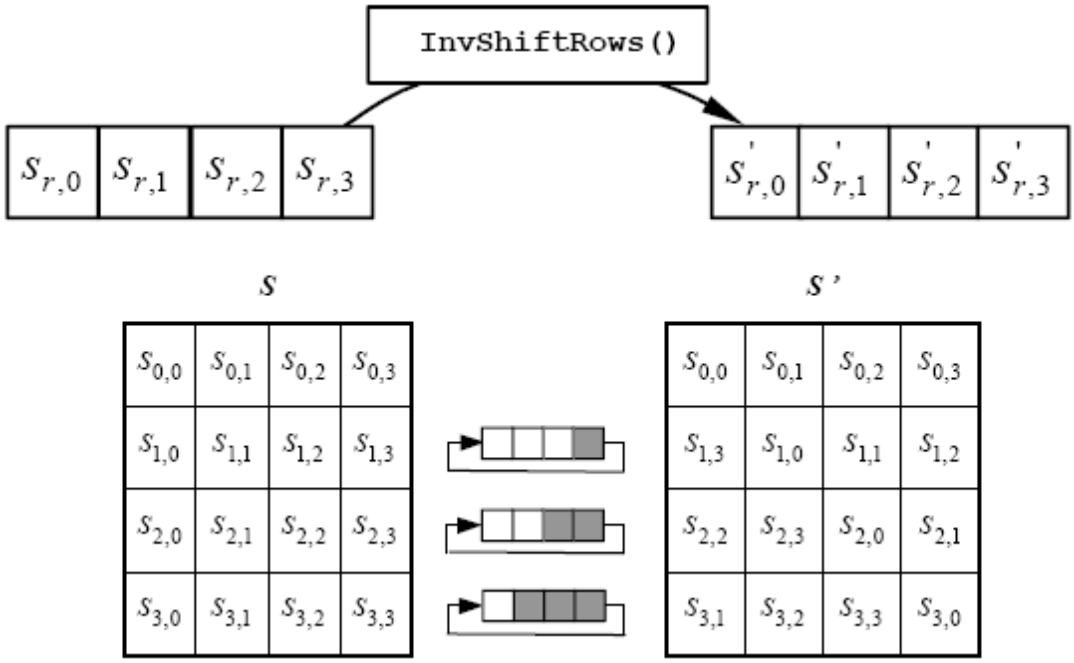
4. Το τρίτο βήμα επαναλαμβάνεται μέχρι να παραχθούν τουλάχιστον 176 bytes για κλειδί μεγέθους 128-bit, 208 bytes για κλειδί μεγέθους 192-bit και 240 bytes για κλειδί μεγέθους 256-bit.

Με την ολοκλήρωση όλων των παραπάνω μετασχηματισμών επιτυγχάνεται η ρωμαλεότητα σε επιθέσεις κρυπτανάλυσης και η αυξημένη ταχύτητα εκτέλεσης του αλγορίθμου κρυπτογράφησης AES.

Για την αποκρυπτογράφηση οι μετασχηματισμοί της διαδικασίας κρυπτογράφησης αντιστρέφονται και τοποθετούνται σε αντίστροφη σειρά ώστε να παραχθεί μια διαδικασία που θα αποκρυπτογραφήσει το κρυπτογράφημα. Έτσι όπως και κατά την κρυπτογράφηση, υπάρχουν τέσσερις διακριτοί μετασχηματισμοί κατά

την αποκρυπτογράφηση, οι InvShiftRows, InvSubBytes, InvMixColumns και AddRoundKey.

Κατά τον μετασχηματισμό InvShiftRows που είναι ο αντίστροφος του ShiftRows τα bytes στις τελευταίες τρεις γραμμές ολισθαίνουν κατά ανάστροφο offset, με αντίθετη φορά από ότι στην ShiftRows διαδικασία



Εικόνα 2.6: Στον αντίστροφο μετασχηματισμό η κύλιση γίνεται και πάλι προς τα αριστερά αλλά αυτή την φορά στην $2^{η}$ γραμμή γίνεται μετατόπιση 3 θέσεις, στην $3^{η}$ δύο και στην $4^{η}$ μια.

Κατά τον μετασχηματισμό InvSubBytes γίνεται η ίδια διαδικασία ακριβώς μόνο που χρησιμοποιείται ο αντίστροφος πίνακας του S-box (inverse S-box).

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Πίνακας 2.3

Στο μετασχηματισμό InvMixColumns ο πολλαπλασιασμός γίνεται με το αντίστροφο πολυώνυμο $c^{-1}(x) = 11x^3 + 13x^2 + 9x + 14$ και ο πολλαπλασιασμός των στοιχείων γίνεται με τον παρακάτω πίνακα.

$$\begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Πίνακας 2.4: Πολλαπλασιασμός των στοιχείων κατά την διαδικασία Inv Mix Columns

ή αλλιώς

$$r_0 = 14a_0 + 9a_3 + 13a_2 + 11a_1$$
$$r_1 = 14a_1 + 9a_0 + 13a_3 + 11a_2$$
$$r_2 = 14a_2 + 9a_1 + 13a_0 + 11a_3$$
$$r_3 = 14a_3 + 9a_2 + 13a_1 + 11a_0$$

Ο μετασχηματισμός AddRoundKey είναι μια απλή XOR πράξη, είναι από μόνος του αντιστρέψιμος και κατά συνέπεια είναι ταυτόσημος με τον αντίστοιχο μετασχηματισμό που γίνεται κατά την διαδικασία της κρυπτογράφησης.[37][38][40]

Στην κρυπτογραφία μέχρι πρότινος, για να εκτελεστεί ο κρυπτογραφικός αλγόριθμος με μεγάλη ταχύτητα έπρεπε να υλοποιηθεί σε υλικό (*hardware*). Αυτό οδήγησε σε εξάρτηση της ταχύτητας του κρυπτογραφικού αλγορίθμου από το εκάστοτε hardware. Η υλοποίηση ενός κρυπτογραφικού αλγορίθμου σε λογισμικό *(software*), λόγω της τεχνολογίας των επεξεργαστών μονού πυρήνα, είναι δυνατή αλλά με επίπτωση στην ταχύτητα της εκτέλεσης του. Έτσι έπρεπε να υπάρχει συμβιβασμός ανάμεσα στο κόστος και την επίδοση, δηλαδή της απόδοσης του. Το πρότυπο AES είναι γρήγορο είτε υλοποιημένο σε hardware είτε σε software, είναι σχετικά εύκολο να υλοποιηθεί, και απαιτεί λίγη μνήμη. Ως καινούριο πρότυπο κρυπτογράφησης αναπτύσσεται σε μεγάλη κλίμακα. Για τους παραπάνω λόγους και για τον λόγο ότι ο αλγόριθμος κρυπτογράφησης AES έχει πολλές επαναληπτικές δομές που τον καθιστούν κατάλληλο για συστήματα παράλληλης επεξεργασίας, επιλέχθηκε σε αυτή τη διπλωματική εργασία.

3 Μοντέλα και βιβλιοθήκες παράλληλου προγραμματισμού

3.1 Γλώσσες προγραμματισμού

Η γλώσσα C++ είναι μια από τις πολλές γλώσσες προγραμματισμού που χρησιμοποιούνται, για την επίλυση προβλημάτων, με τους ηλεκτρονικούς υπολογιστές. Είναι γενικού σκοπού αντικειμενοστραφής γλώσσα προγραμματισμού και θεωρείται μέσου επιπέδου γλώσσα, καθώς περιλαμβάνει έναν συνδυασμό χαρακτηριστικών από γλώσσες υψηλού και χαμηλού επιπέδου. Η μετάφραση της (compilation) δημιουργεί κώδικα μηχανής. Υποστηρίζει δομημένο, αντικειμενοστραφή και γενικό προγραμματισμό. Η γλώσσα C++ αναπτύχθηκε το 1979 από τον Bjarne Stroustrup στα εργαστήρια Bell της AT&T, ως βελτίωση της ήδη υπάρχουσας γλώσσας προγραμματισμού C, και αρχικά ονομάστηκε "C with Classes". Οι βελτιώσεις ξεκίνησαν με την προσθήκη κλάσεων, και ακολούθησαν, μεταξύ άλλων, εικονικές συναρτήσεις, υπερφόρτωση τελεστών, πολλαπλή κληρονομικότητα και πρότυπα. Στον αντικειμενοστραφή προγραμματισμό, η προσπάθεια επικεντρώνεται στη δημιουργία μονάδων, οι οποίες θα περιλαμβάνουν τα δεδομένα, αλλά και τις εντολές, οι οποίες τα διαχειρίζονται. Οι μονάδες αυτές, στις γλώσσες που υποστηρίζουν αντικειμενοστραφή προγραμματισμό, λέγονται συναρτήσεις και στον ίδιο τον αντικειμενοστραφή προγραμματισμό λέγονται αντικείμενα. Ένα αντικείμενο, μπορεί να έχει μία ή περισσότερες συναρτήσεις. Στην C++ οι συναρτήσεις αυτές ονομάζονται συναρτήσεις-μέλη του αντικειμένου. Αυτές δε, ελέγχουν κατά κανόνα, την πρόσβαση στα δεδομένα του αντικειμένου.

3.1.1 Γλώσσα μηχανής

Αρχικά για να μπορέσει ο υπολογιστής να εκτελέσει μία οποιαδήποτε λειτουργία, έπρεπε να δοθούν κατευθείαν οι κατάλληλες ακολουθίες από 0 και 1, δηλαδή εντολές σε μορφή κατανοητή από τον υπολογιστή αλλά πολύ δύσκολα κατανοητές από τον άνθρωπο. Ο τρόπος αυτός ήταν επίπονος και ελάχιστοι μπορούσαν να τον υλοποιήσουν, αφού απαιτούσε βαθιά γνώση του υλικού και της αρχιτεκτονικής του υπολογιστή.

Οι εντολές που δέχεται ένας υπολογιστής μετατρέπονται σε ακολουθίες που αποτελούνται από 0 και 1, γίνονται δηλαδή εντολές σε γλώσσα μηχανής όπως ονομάζονται, έτσι ώστε να γίνουν κατανοητές από τον υπολογιστή και στη συνέχεια να μπορέσει να τις εκτελέσει.

3.1.2 Γλώσσα υψηλού επιπέδου

Οι ανεπάρκειες των συμβολικών γλωσσών και η προσπάθεια για καλύτερη επικοινωνία ανθρώπου μηχανής οδήγησαν στα τέλη της δεκαετίας του 50 στην εμφάνιση των πρώτων γλωσσών προγραμματισμού υψηλού επιπέδου.Το 1957 η IBM ανάπτυξε την πρώτη γλώσσα υψηλού επιπέδου τη **FORTRAN**. Το όνομα FORTRAN προέρχεται από τις λέξεις **FOR**mula **TRAN**slation που σημαίνουν μετάφραση τύπων.

Η FORTRAN αναπτύχθηκε ως γλώσσα κατάλληλη για την επίλυση μαθηματικών και επιστημονικών προβλημάτων. Το 1960 αναπτύχθηκε μία άλλη γλώσσα, σταθμός στον προγραμματισμό η γλώσσα **COBOL**. Η COBOL όπως δηλώνει και το όνομα της (**CO**mmon **B**usiness **O**riented **L**anguage-Κοινή γλώσσα προσανατολισμένη στις επιχειρήσεις) είναι κατάλληλη για ανάπτυξη εμπορικών εφαρμογών, και γενικότερα διαχειριστικών εφαρμογών, τομέας όπου η FORTRAN υστερούσε.

Μια από τις σημαντικότερες γλώσσα προγραμματισμού με ελάχιστη πρακτική εφαρμογή αλλά που επηρέασε ιδιαίτερα τον προγραμματισμό και τις επόμενες γλώσσες είναι η **ALGOL** (**Algo**rithmic **L**anguage-Αλγοριθμική γλώσσα).Στο χώρο της Τεχνητής Νοημοσύνης αναπτύχθηκαν δυο γλώσσες διαφορετικές από τις άλλες.

Στα μέσα του 60 αναπτύχθηκε η **LISP** (**LIS**t **P**rocessor- Επεξεργαστής λίστας), γλώσσα η οποία προσανατολίζεται σε χειρισμό λιστών από τα σύμβολα και η **PROLOG** (**PRO**gramming **LOG**ic- Λογικός Προγραμματισμός) στις αρχές του 70.

Δύο σημαντικότατες γλώσσες γενικού σκοπού, οι οποίες αναπτύχθηκαν τη δεκαετία του 60 αλλά χρησιμοποιούνται πάρα πολύ στις ημέρες μας, είναι η BASIC και η PASCAL.Η γλώσσα προγραμματισμού **BASIC** (**B**eginner's **A**ll **P**urpose **S**ymbolic **C**ode- Συμβολικός Κώδικας Εντολών Γενικής Χρήσης για Αρχάριους) αρχικά αναπτύχθηκε, όπως δηλώνει και το όνομα της, ως γλώσσα για την εκπαίδευση αρχαρίων στον προγραμματισμό. Σχεδιάστηκε για να γράφονται σύντομα προγράμματα, τα οποία εκτελούνται με τη βοήθεια διερμηνευτή (interpreter).

Η γλώσσα **PASCAL** έφερε μεγάλες αλλαγές στον προγραμματισμό. Παρουσιάστηκε το 1970 και στηρίχθηκε πάνω στην ALGOL. Είναι μια γλώσσα

γενικής χρήσης, η οποία είναι κατάλληλη τόσο για την εκπαίδευση όσο και τη δημιουργία ισχυρών προγραμμάτων κάθε τύπου. Χαρακτηριστικό της γλώσσας είναι η καταλληλότητα για τη δημιουργία δομημένων προγραμμάτων. Μία ακόμη γλώσσα που γνώρισε μεγάλη διάδοση είναι η γλώσσα **C.**

Η C χρησιμοποιήθηκε για την ανάπτυξη του λειτουργικού συστήματος Unix γλώσσα με ισχυρά χαρακτηριστικά, μερικά από αυτά κοινά με την Pascal κατάλληλη για την ανάπτυξη δομημένων εφαρμογών αλλά και με πολλές δυνατότητες γλώσσας χαμηλού επιπέδου. Η C εξελίχτηκε στη γλώσσα **C++,** που είναι αντικειμενοστραφής.

Τα τελευταία χρόνια χρησιμοποιείται ιδιαίτερα, ειδικά για προγραμματισμό στο Διαδίκτυο (Internet), η **JAVA**. Η JAVA είναι μια αντικειμενοστραφής γλώσσα που αναπτύχθηκε από την εταιρία SUN με σκοπό την ανάπτυξη εφαρμογών, που θα εκτελούνται σε κατανεμημένα περιβάλλοντα, δηλαδή σε διαφορετικούς υπολογιστές οι οποίοι είναι συνδεδεμένοι με το Διαδίκτυο.

3.1.3 Πλεονεκτήματα των γλωσσών υψηλού επιπέδου

Στα πλεονεκτήματα των γλωσσών προγραμματισμού υψηλού επιπέδου σε σχέση με τις συμβολικές μπορούν να αναφερθούν:

- Ο φυσικότερος και πιο «ανθρώπινος» τρόπος έκφρασης των προβλημάτων. Τα προγράμματα σε γλώσσα υψηλού επιπέδου είναι πιο κοντά στα προβλήματα που επιλύουν.
- Η ανεξαρτησία από τον τύπο του υπολογιστή. Προγράμματα σε μία γλώσσα υψηλού επιπέδου μπορούν να εκτελεστούν σε οποιονδήποτε υπολογιστή με ελάχιστες ή καθόλου μετατροπές. Η δυνατότητα της μεταφερσιμότητας των προγραμμάτων είναι σημαντικό προσόν.
- Η ευκολία της εκμάθησης και εκπαίδευσης ως απόρροια των προηγούμενων.
- Η διόρθωση λαθών και η συντήρηση προγραμμάτων σε γλώσσα υψηλού επιπέδου είναι πολύ ευκολότερο έργο.

Συνολικά οι γλώσσες υψηλού επιπέδου ελάττωσαν σημαντικά το χρόνο και το κόστος παραγωγής νέων προγραμμάτων, αφού λιγότεροι προγραμματιστές μπορούν σε μικρότερο χρόνο να αναπτύξουν προγράμματα που χρησιμοποιούνται σε περισσότερους υπολογιστές.

3.1.4 Γλώσσες 4ης γενιάς

Στις γλώσσες αυτές ο χρήστης ενός υπολογιστή έχει τη δυνατότητα σχετικά εύκολα να υποβάλει ερωτήσεις στο σύστημα ή να αναπτύσσει εφαρμογές που ανακτούν πληροφορίες από βάσεις δεδομένων και να καθορίζει τον ακριβή τρόπο εμφάνισης αυτών των πληροφοριών.

Τα χαρακτηριστικά της C++ είναι, όπως αναφέρθηκε και πιο πάνω, οι κλάσεις, οι εικονικές συναρτήσεις, η υπερφόρτωση τελεστών, η πολλαπλή κληρονομικότητα και τα πρότυπα. Όταν έχουμε όμοια αντικείμενα, λέμε ότι, τα αντικείμενα αποτελούν ή ανήκουν σε μια κλάση. Μια κλάση μπορεί να έχει ένα ή περισσότερα του ενός αντικείμενα. Μετά τον ορισμό της κλάσης, μπορούμε να δηλώσουμε ένα ή περισσότερα αντικείμενα σε αυτή τη κλάση. Για παράδειγμα η δήλωση πραγματικών μεταβλητών σε μια γλώσσα προγραμματισμού με προκαθορισμένο τρόπο, όπως float, αποτελεί μια κλάση. Στην κλάση αυτή μπορούμε να δηλώσουμε, όσες μεταβλητές(αντικείμενα), θέλουμε.

```
float a, b, utr1, mbg6, aw_8;
```

Κάθε αντικείμενο θα έχει τις ίδιες ιδιότητες με αυτές της κλάσης, δηλαδή κάθε μεταβλητή θα δέχεται έναν και μόνον έναν αριθμό και αυτός θα είναι πραγματικός.

Υπερφόρτωση τελεστών ονομάζεται η χρησιμοποίηση του ίδιου συμβόλου για πολλαπλές λειτουργίες. Γενικότερα η ικανότητα μιας γλώσσας να χρησιμοποιεί τα ίδια σύμβολα για πολλαπλές δουλειές ονομάζεται πολυμορφισμός.

Η πολλαπλή κληρονομικότητα υφίσταται όταν μια κλάση είναι υποκλάση περισσοτέρων της μιας κλάσης οπότε και κληρονομεί τις ιδιότητες αυτών.

Τα πρότυπα (templates) παρέχουν έναν απλό τρόπο για την αναπαράσταση ενός ευρύ φάσματος γενικών εννοιών, και απλούς τρόπους για τον συνδυασμό αυτών των εννοιών. Οι τάξεις και οι συναρτήσεις που προκύπτουν μπορούν να φτάσουν τον πιο εξειδικευμένο χειροποίητο κώδικα σε αποδοτικότητα χρόνου εκτέλεσης και χώρου μνήμης. Τα πρότυπα παρέχουν έμμεση υποστήριξη για το γενικευμένο προγραμματισμό, δηλαδή τον προγραμματισμό όπου χρησιμοποιούνται τύποι ως πρότυπα. Ο μηχανισμός προτύπων της C++ επιτρέπει να είναι ένας τύπος παράμετρος στον ορισμό μιας τάξης ή μιας συνάρτησης. Ένα πρότυπο βασίζεται μόνο

στις ιδιότητες που χρησιμοποιεί πραγματικά από τους τύπους των παραμέτρων του, και δεν απαιτεί να έχουν ρητή σχέση οι διάφοροι τύποι που χρησιμοποιούνται ως ορίσματα. Ειδικότερα, οι τύποι των ορισμάτων που χρησιμοποιούνται σε ένα πρότυπο δε χρειάζεται να ανήκουν σε μια ενιαία ιεραρχία κληρονομικότητας.

3.2 OpenMP

Λόγω της ανάγκης που δημιουργήθηκε για την ταυτόχρονη εκτέλεση προγραμμάτων ο τρόπος που γράφονταν οι πηγαίοι κώδικες έπρεπε να αλλάξει με κατάλληλο τρόπο ώστε ο επεξεργαστής να μπορεί να εκτελεί κομμάτια διαφορετικών προγραμμάτων το ένα μετά το άλλο δημιουργώντας την ψευδαίσθηση ότι πολλά προγράμματα τρέχουν ταυτόχρονα. Λόγω αυτής της ανάγκης ο αντικειμενοστραφής προγραμματισμός από μόνος του δεν επαρκούσε. Αυτό είχε ως αποτέλεσμα την ανάπτυξη βιβλιοθηκών, υποστηριζόμενες από τις υπάρχουσες γλώσσες προγραμματισμό, με εξειδίκευση στην τεχνολογία πολλαπλών νημάτων (multithreading) όπου το πρόγραμμα σπάει σε νήματα τα οποία μπορούν να εκτελεστούν παράλληλα από δύο οι περισσότερους επεξεργαστικούς πυρήνες του επεξεργαστή ή νήματα από διαφορετικά προγράμματα να εκτελεστούν σε έναν επεξεργαστικό πυρήνα, όπου λόγω της μεγάλης ταχύτητας επεξεργασίας, δημιουργείται η ψευδαίσθηση της ταυτόχρονης εκτέλεσης πολλών προγραμμάτων. Ένα τέτοιο πακέτο βιβλιοθηκών είναι η OpenMP που χρησιμοποιείται πολύ στην ανάπτυξη παιχνιδιών.

Η αρχιτεκτονική των πολύ-ΕΠ έχει αρκετές ομοιότητες με την αρχιτεκτονική πολλών επεξεργαστών κοινής μνήμης όπως επίσης σε διαφορετικό βαθμό με την αρχιτεκτονική ΣΠΝ ΕΠ . Έτσι μοντέλα – βιβλιοθήκες γραμμένα για αυτές τις αρχιτεκτονικές ή γενικώς για παράλληλο προγραμματισμό μπορούν να χρησιμοποιηθούν για προγραμματισμό και ανάπτυξη εφαρμογών για πολύ-ΕΠ. Αυτά μπορούν να χωριστούν σε δύο κατηγορίες ανάλογα με το τρόπο πρόσβασης των νημάτων ή διεργασιών στα δεδομένα. Αλλά υπάρχουν και υβριδικές λύσεις που συνδυάζονται μοντέλα-βιβλιοθήκες και από τις δύο κατηγορίες Τα πιο διακεκριμένα είναι το OpenMP και το MPI.[5][13][14][15]

Το OpenMP είναι μια ανοιχτή διεπαφή προγραμματισμού εφαρμογών σε C/C++ και Fortran σε πολλές πλατφόρμες για την ανάπτυξη πολύ-νηματικών εφαρμογών

κοινής μνήμης. Αποτελείται από ένα σύνολο επεκτάσεων μεταγλωττιστών και βιβλιοθηκών. Ορίζεται από κοινού από μια ομάδα με τις σημαντικότερες εταιρείες στον χώρο της πληροφορικής και είναι αναγνωρισμένο πρότυπο με ευρεία αποδοχή. Υποστήριξη για το OpenMP έχουν μεταγλωττιστές των GNU, IBM, Sun, Intel, Microsoft και άλλους.. Σκοπός είναι να δώσει στους προγραμματιστές μια απλή και ευέλικτη διεπαφή για ανάπτυξη παράλληλων εφαρμογών με τον λιγότερο δυνατό κώδικα χωρίς να είναι απαραίτητη η γνώση των γηγενών διεπαφών για νήματα. Οι εντολές όπως αυτές του κατακερματισμού, κλειδώματος, συγχρονισμού, ορισμού τμημάτων υλοποιούνται με την χρήση των εντολών προ-επεξεργαστή "**pragma**". Ενώ η διαχείριση των παραμέτρων του συστήματος OpenMP γίνεται με συναρτήσεις αλλά και μεταβλητές περιβάλλοντος. Μέσω των επεκτάσεων του μεταγλωττιστή ο προγραμματιστής μπορεί να διασπάσει σε νήματα επαναλήψεις ή μερικές γραμμές κώδικα με την προσθήκη ακόμα και μίας γραμμής. Τα νήματα αυτά μπορούν να διαρκέσουν όσο η επανάληψη ή όσο θέλει ο προγραμματιστής. Αυτό δίνει την δυνατότητα μετατροπής σειριακών προγραμμάτων σε παράλληλα χωρίς την αλλαγή της δομής του. Ένα άλλο πλεονέκτημα είναι η διάδοση αυτής της διεπαφής και η ευρεία υποστήριξη της. Μοναδικό ίσως μειονέκτημα του OpenMP είναι ότι χρειάζεται υποστήριξη μεταγλωττιστή.[5]

3.2.1 Εντολές pragma και συναρτήσεις OpenMP

Οι λειτουργίες κατακερματισμού στο OpenMP υλοποιούνται με μια σειρά από εντολές pragma, οι οποίες μπορούν δεχθούν και παραμέτρους οι οποίες μπορεί να έχουν καμία, μία και παραπάνω τιμές. Η σύνταξη των εντολών έχει ως εξής:

```
    #pragma omp εντολή [παράμετρος1[(τιμή1[,τιμή2,...])]] ...
νέα γραμμή
      τμήμα δομημένου κώδικα
    πχ.
    #pragma omp parallel default(shared)
private(threadid,num_thread)
     {
          threadid=omp_get_thread_num()+1;
          num_thread=omp_get_num_threads();
          printf("Νήμα %d από %d\n",threadid,num_thread);
     }
```

Εντολή pragma omp	Περιγραφή
Parallel	Βασική εντολή όπου δηλώνει ότι αρχίζει το παράλληλο τμήμα. Το παράλληλο τμήμα εκτελείται ξεχωριστά από n νήματα, όπου n ο μέγιστος αριθμός των νημάτων ή αυτός που δηλώθηκε ως παράμετρος στην εντολή αυτή. Σχεδόν όλες η άλλες εντολές πρέπει να βρίσκονται μέσα στο τμήμα της parallel.
For	Εντολή που κατακερματίζει το βρόγχο for που υποχρεωτικά βρίσκεται ακριβώς μετά την εντολή αυτή.
Sections	Εντολή που κατακερματίζει ένα τμήμα του κώδικα σε *x* section. Πρέπει να περιέχει εντολές section.
Section	Εντολή που δηλώνει ότι το τμήμα που ακολουθεί είναι ένα κατακερματισμένο τμήμα του γονικού sections. Βρίσκεται ακριβώς κάτω από την εντολή sections.
Single	Δηλώνει ότι το τμήμα δομημένου κώδικα που ακολουθεί θα εκτελεστεί μόνο από ένα νήμα.
Critical	Δηλώνει ότι το τμήμα δομημένου κώδικα που ακολουθεί θα εκτελεστεί μόνο από ένα νήμα τη φορά.
Barrier	Δηλώνει ότι τα νήματα που φτάνουν σε αυτό το σημείο πρέπει να περιμένουν τα υπόλοιπα νήματα. Τα νήματα προχωρούν μόνο όταν φτάσουν όλα τα νήματα σε αυτό το σημείο
Parallel for	Μια ευκολία που συνδυάζει τις εντολές parallel και for σε μία γραμμή.
Parallel sections	Μια ευκολία που συνδυάζει τις εντολές parallel και sections σε μία γραμμή.

Πίνακας 3.1: Μερικές βασικές εντολές pragma omp.

Παράμετρος	Ισχύει για εντολές	Περιγραφή
Private	parallel, for, sections, single	Δέχεται ως τιμές τις μεταβλητές που επιθυμούμε να είναι ιδιωτικές για κάθε νήμα Δηλαδή σε κάθε νήμα θα μπορεί να έχει διαφορετική τιμή. Κυρίως για μεταβλητές που μεταβάλλονται από ένα η παραπάνω νήματα.
Shared	parallel, for	Δέχεται ως τιμές τις μεταβλητές που επιθυμούμε να είναι κοινές για κάθε νήμα Δηλαδή σε κάθε νήμα θα έχει πάντα την ίδια τιμή. Κυρίως για μεταβλητές που δεν μεταβάλλονται.
Default	Parallel	Δέχεται ως τιμή είτε “shared” είτε “none”. Στην πρώτη περίπτωση προεπιλογή είναι ότι η μεταβλητές είναι κοινές. Στην δεύτερη περίπτωση ότι δεν υπάρχει προεπιλογή και πρέπει να οριστεί για κάθε μεταβλητή που θα χρησιμοποιηθεί.
schedule	For	Ορίζει τον τύπο του κατακερματισμού ενός βρόγχου for. Δέχεται ως τιμές τον τύπο του κατακερματισμού και ανάλογα με τον τύπο μπορεί να δεχθεί μια ακόμα τιμή, το μέγεθος κάθε κατακερματισμένου τμήματος.
Nowait	for, sections, single	Δηλώνει ότι τα νήματα που τελειώνουν δεν χρειάζονται να περιμένουν τα άλλα νήματα να τελειώσουν.

Πίνακας 3.2: Μερικές από τις βασικές παραμέτρους των εντολών pragma omp

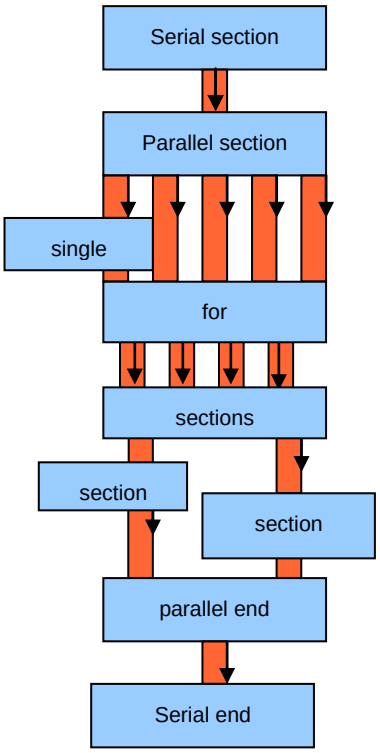
Συνάρτηση	Περιγραφή
omp_set_num_threads(int t)	Ορίζει τον αριθμό των νημάτων που θα χρησιμοποιηθούν στο επόμενο παράλληλο τμήμα.
int omp_get_num_threads(void)	Επιστρέφει τον συνολικό αριθμό των νημάτων μέσα σε ένα παράλληλο τμήμα.
int omp_get_thread_num(void)	Επιστρέφει τον αριθμό του τρέχων νήματος.
int omp_get_num_procs(void)	Επιστρέφει τον αριθμό των ΕΠ του συστήματος.

Πίνακας 3.3: Μερικές από τις βασικές συναρτήσεις OpenMP

3.2.2 Παράδειγμα OpenMP με γραφική απεικόνιση κατακερματισμού

```
#include <omp.h>
#include <stdio.h>

int main(int argc,char *argv[]) {
  // αριθμός επεξεργαστών πυρήνων
  int num_proc = omp_get_num_procs();
  // μέγιστος αριθμός νημάτων
  int max_thread= omp_get_max_threads();
  int threadid,num_thread,i;
  printf("Αριθμός ΕΠ = %d\n",num_proc);
  printf("Μέγιστος αριθμός νημάτων = %d\n",max_thread);
  puts("------- Αρχή παράλληλου τμήματος ------");
#pragma omp parallel default(shared)
private(threadid,num_thread)
  {
    threadid=omp_get_thread_num()+1;
    num_thread=omp_get_num_threads();
    printf("Νήμα %d από %d\n",threadid,num_thread);
#pragma omp single nowait
      printf("Εκτελέστηκε μόνο από το νήμα
%d\n",threadid);
#pragma omp for private(i)
      for(i=0;i<10;i++) {
        printf("--- Βρόγχος for νήμα=%d,
i=%d\n",omp_get_thread_num()+1,i);
      }
#pragma omp sections
    {

#pragma omp section
      {
 printf("--- section 1 νήμα=%d\n",omp_get_thread_num()+1);
      }
#pragma omp section
```

```
            {
              printf("---                    section                    2
νήμα=%d\n",omp_get_thread_num()+1);
              printf("---          section          2          (ξανά)
νήμα=%d\n",omp_get_thread_num()+1);
            }
          }
        }
        puts("------- Τέλος παράλληλου τμήματος ------");
        return 0;
      }
```



Εικόνα 3.1: Γραφική απεικόνιση του κατακερματισμού του παραπάνω παραδείγματος.

4 Συγκριτικές δοκιμές

4.1 Περιβάλλον εργασίας

Η ενότητα αυτή περιγράφει το περιβάλλον εργασίας που θα χρησιμοποιηθεί για την ανάπτυξη και εκτέλεσης των συγκριτικών δοκιμών και περιλαμβάνει το Λειτουργικό Σύστημα (ΛΣ), το περιβάλλον ανάπτυξης εφαρμογών (Integrated development environment – IDE).

4.2 Υλικό Η/Υ

Ο Η/Υ που θα χρησιμοποιηθεί για την εκτέλεση των δοκιμών περιέχει τον επεξεργαστή Intel E8400. Συγκεκριμένα :

Χαρακτηριστικά	Η/Υ
Επεξεργαστής - μοντέλο	Intel E8600
Επεξεργαστής – ΕΠ	2
Επεξεργαστής – συχνότητα	2.40Ghz
Επεξεργαστής – KM2 (cache L2)	6MB
Επεξεργαστής – τρόπος επικοινωνίας	KM2
Αριθμός επεξεργαστών	1
Κεντρική μνήμη (RAM)	4GB
Κεντρική μνήμη – διάταξη	2 x 2GB
Κεντρική μνήμη – τεχνολογία	DDR2
Κεντρική μνήμη – Συχνότητα	667Mhz

Πίνακας 4.1: Υλικό Η/Υ

4.3 Λειτουργικό σύστημα

Το λειτουργικό σύστημα που θα χρησιμοποιηθεί είναι η διανομή Linux Ubuntu 10. 64bit στο οποίο έχουν εφαρμοσθεί όλες οι αναβαθμίσεις (updates) .

Στοιχείο	Έκδοση (rpmquery)
Πυρήνας (Kernel)	kernel-2.6.32-25-generic
Βιβλιοθήκη C (glibc)	glibc-2.9-3.x86_64
Βιβλιοθήκη C++	libstdc++-4.3.2-7.x86_64
OpenMP	libgomp-4.3.2-7.x86_64
Profiler – Oprofile	oprofile-0.9.4-3.fc10.x86_64

Πίνακας 4.2: Εκδόσεις βασικών στοιχείων ΛΣ

Οι λόγοι οι οποίοι έχει επιλεχθεί αυτό το λειτουργικό σύστημα είναι οι εξής :

- Μηδενικό κόστος απόκτησης της διανομής χωρίς περιορισμούς.
- Πληθώρα εργαλείων ανάπτυξης περιλαμβάνονται στην διανομή
- Περιλαμβάνει πληθώρα βιβλιοθηκών παράλληλου προγραμματισμού όπως *Posix Threads, Intel Threading Building Blocks, Java* και μεταγλωττιστή με υποστήριξη *OpenMP*.

Άλλες λύσεις όπως Windows έχουν ένα κόστος και τα εργαλεία ανάπτυξης πρέπει είτε να αγοραστούν είτε να χρησιμοποιηθούν μη σταθερές εκδόσεις για Windows των εργαλείων που περιλαμβάνονται στο Linux.

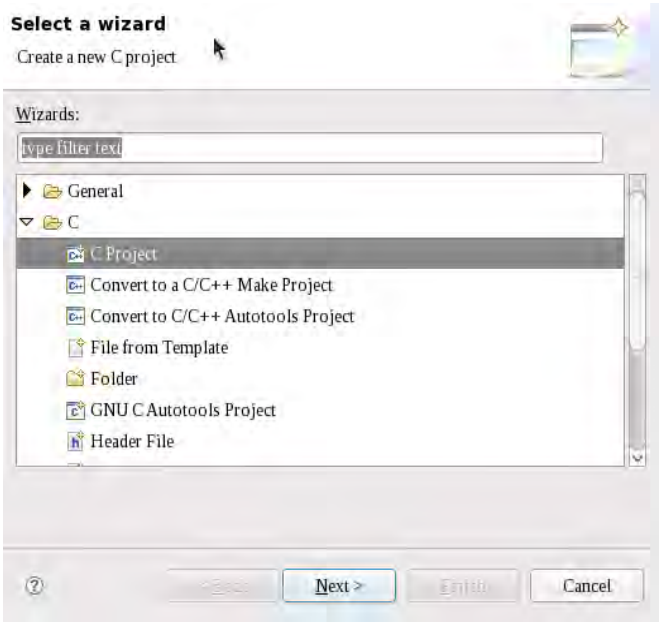
Επίσης θα χρησιμοποιηθεί το περιβάλλον ανάπτυξης εφαρμογών Eclipse το οποίο περιλαμβάνεται στην διανομή Ubuntu. Ο λόγος που επιλέχθηκε είναι γιατί είναι ένα ώριμο δημοφιλής περιβάλλον ανάπτυξης εφαρμογών και περιλαμβάνεται στην διανομή. Η έκδοση που θα χρησιμοποιηθεί είναι η Ubuntu Eclipse 3.4.1

Ο μεταγλωττιστής που θα χρησιμοποιηθεί είναι ο GCC της GNU καθώς υποστηρίζει το OpenMP 2.5, είναι αξιόπιστος και είναι διαθέσιμος από την διανομή. Η έκδοση που θα χρησιμοποιηθεί είναι η 4.3.2.

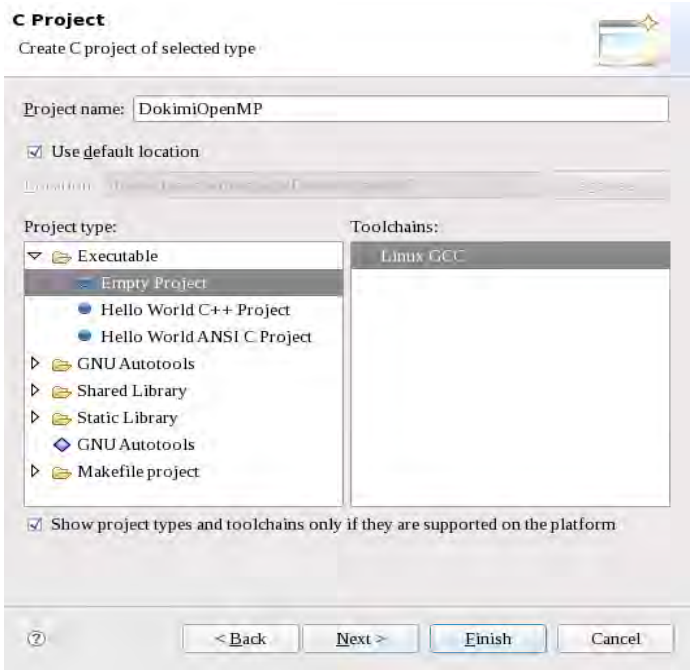
Τέλος θα χρησιμοποιηθεί η βιβλιοθήκη OpenMP διότι είναι κατάλληλη για την χρήση με πολύ-ΕΠ. Είναι αναγνωρισμένο ανοιχτό πρότυπο με ευρεία αποδοχή και χρήση που προσφέρει ένα σχετικά εύκολο περιβάλλον χρήσης.

4.4 Περιβάλλον ανάπτυξης εφαρμογών

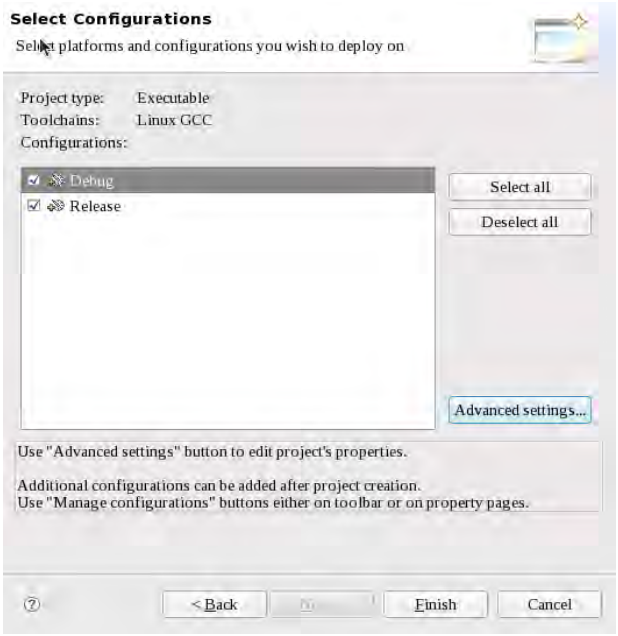
Το περιβάλλον ανάπτυξης του Eclipse είναι πάρα πολύ απλό, εύχρηστο και κατανοητό καθώς είναι παραθυρικό περιβάλλον. Ανοίγοντας το Eclipse ανοίγει το παράθυρο που καθοδηγεί για την δημιουργία του καινούργιου περιβάλλοντος για την ανάπτυξη του καινούριου προγράμματος.



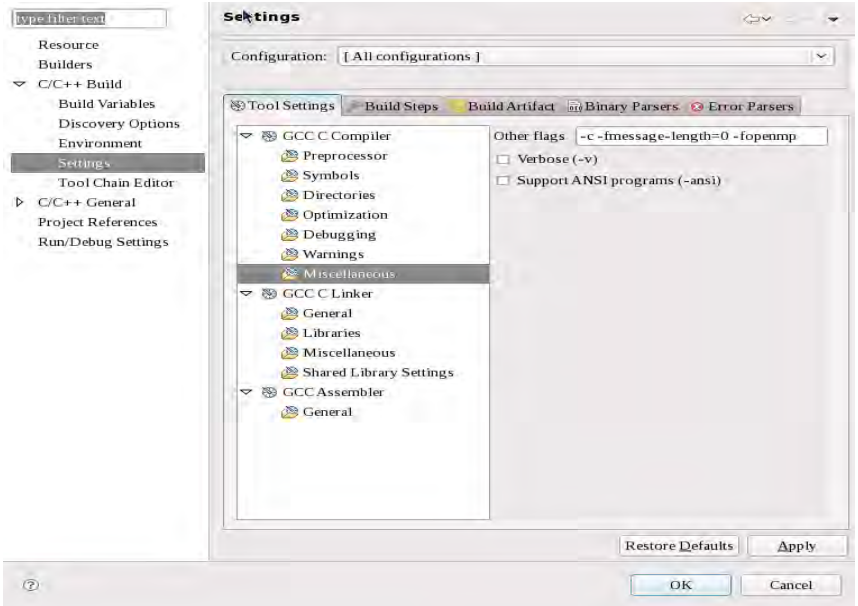
Δημιουργία νέου “Project” στο Eclipse και επιλογή “C project”.



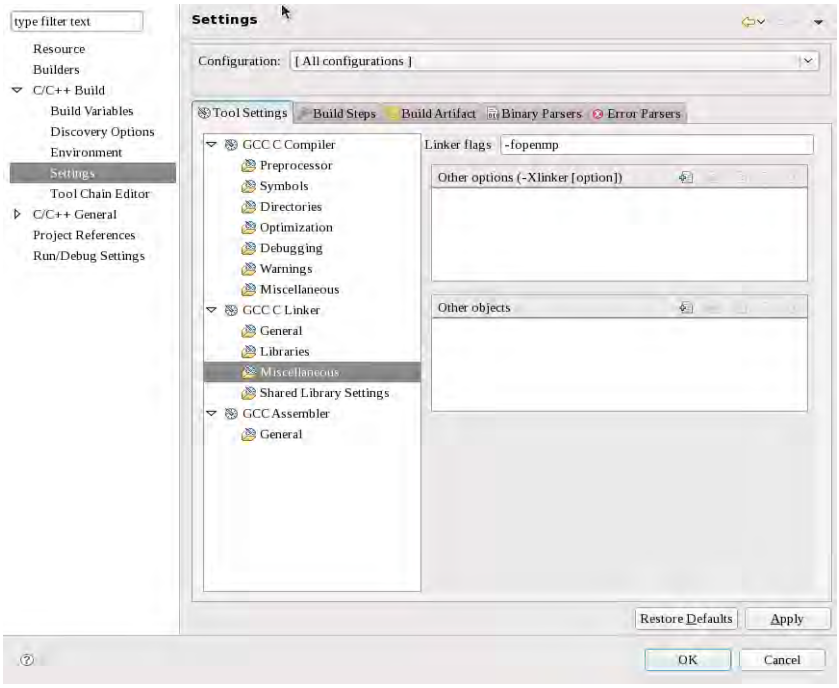
Ορισμός του ονόματος του έργου (project), στην περίπτωση αυτή DokimiOpenMP. Στο τύπο του έργου (Project type) γίνεται επιλογή “Executable->Empty Project” καθώς είναι απλούστερη λύση.

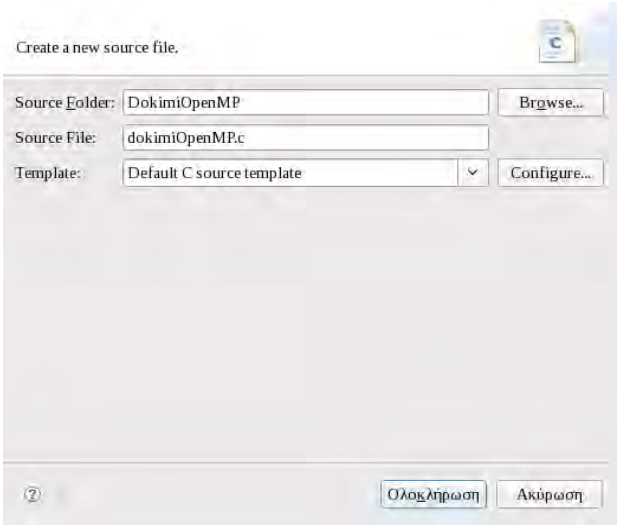


Στην συνέχεια κάνοντας κλικ στα “Advanced Settings” εμφανίζεται το παρακάτω παράθυρο.



Εδώ για να ενεργοποιηθεί το OpenMP πρέπει να προστεθεί η παράμετρος "-fopenmp" στις παραμέτρους του μεταγλωττιστή (GCC C Compiler, Miscellaneous, Other Flags) άλλα και στον "GCC C Linker", Miscellaneous, Linker Flags :





Για τη δημιουργία ενός C αρχείου πηγαίνουμε στο Αρχείο -> Δημιουργία, Source File όπου εμφανίζεται ο παραπάνω διάλογος. Στο ίδιο το αρχείο που θα περιλαμβάνει pragma ή συναρτήσεις OpenMP θα πρέπει να εισάγει την κεφαλίδα του OpenMP, *omp.h.*:

```
#include <omp.h>
```

4.5 Εργαλεία μετρήσεων

Για την μέτρηση της χρήσης των ΕΠ χρησιμοποιήθηκε το oprofile[41] και την συνάρτηση clock_gettime(). Με το oprofile μετρούνται δύο μετρητές των επεξεργαστών τύπου Intel Core 2 CPU_CLK_UNHALTED(Unhalted core cycles)[12] και BUS_TRAN_MEM(All Cores)[13] . Οι παράμετροι του oprofile είναι η εξής :

```
SESSION_DIR=/var/lib/oprofile
CHOSEN_EVENTS_0=CPU_CLK_UNHALTED:100000:0:0:1
CHOSEN_EVENTS_1=BUS_TRAN_MEM:10000:0xc0:0:1
NR_CHOSEN=2
SEPARATE_LIB=0
SEPARATE_KERNEL=0
SEPARATE_THREAD=0
SEPARATE_CPU=1
VMLINUX=none
IMAGE_FILTER=<Η διαδρομή του εκτελέσιμου αρχείου.>
BUF_SIZE=65536
CPU_BUF_SIZE=0
CALLGRAPH=0
XENIMAGE=none
```

Επίσης για την μέτρηση απόδοσης των παράλληλων και σειριακών αλγορίθμων υλοποιημένων σε C/C++, θα χρησιμοποιηθεί και η συνάρτηση clock_gettime() της βιβλιοθήκης C για την μέτρηση του χρόνου εκτέλεσης των αλγορίθμων. Αντίστοιχα στην υλοποίηση Java θα χρησιμοποιηθεί η συνάρτηση System.nanoTime().

12 Το CPU_CLK_UNHALTED(Unhalted core Cycles) μετράει τον αριθμό των κύκλων ενός ΕΠ όταν δεν είναι σε κατάσταση HALT, σταματημένο.

13 Το BUS_TRAN_MEM(All Cores) μετράει τον αριθμό των ολοκληρωμένων συναλλαγών μνήμης κάθε ΕΠ.

Ένας σειριακός αλγόριθμος έχει πλεονέκτημα στους μικρούς χρόνους αφού κατά την εκτέλεση του υπάρχει ένας διαθέσιμος ΕΠ για χρήση από το σύστημα. Έτσι η παραμικρή χρήση ΕΠ από τρίτη διεργασία επηρεάζει τον πραγματικό χρόνο εκτέλεσης του παράλληλου αλγόρθμου περισσότερο από ότι έναν σειριακό. Για αυτό και οι σειριακοί και οι παράλληλοι αλγόριθμοι θα εκτελούνται πολλές φορές και θα χρησιμοποιήται η μικρότερη τιμή.

Επίσης οι μετρήσεις εξαρτιούνται από τον φόρτο των ΕΠ και την συχνότητά τους. Γιαυτό κατά την διάρκεια της εκτέλεσης θα απενεργοποιηθεί η λειτουργία κλιμάκωσης των συχνοτήτων των ΕΠ και το ΛΣ θα τρέχει σε κατάσταση ατομικού χρήστη 1 (runlevel 1 – Single user).

4.6 Περιορισμοί παράλληλων αλγορίθμων και πιθανές λύσεις

Σύμφωνα με τον Amdahl[42] κάποια τμήματα ενός αλγορίθμου είναι δύσκολο να κατακερματιστούν καθώς είτε γιατί υπάρχει κάποια εξάρτηση που το απαγορεύει είτε γιατί οι εργασίες μέσα σε αυτά πρέπει να εκτελεστούν με κάποια συγκεκριμένη σειρά. Αυτό έχει ως αποτέλεσμα ανάλογα με το ποσοστό σειριακών τμημάτων σε ένα αλγόριθμο και την είσοδό του, μετά από έναν αριθμό ΕΠ η απόδοση παραμένει σταθερή. Αυτό γιατί στον συγκεκριμένο αριθμό ΕΠ έχει επιτευχθεί ήδη μέγιστος κατακερματισμός και οι υπόλοιποι ΕΠ δεν χρησιμοποιούνται πλήρως ή καθόλου. Σε μερικές περιπτώσεις η επανασχεδίαση των αλγορίθμων μπορεί να μειώσει το ποσοστό του κώδικα που εκτελείται σειριακά. Αλλά αυτό ίσως να μην είναι αποτελεσματικό ή δυνατό.

Επίσης ο αριθμός των νημάτων που εκτελούνται ταυτόχρονα σε ένα κατακερματισμό σε συνδυασμό με ότι όλες οι εργασίες δεν έχουν την ίδια διάρκεια είναι ένα πρόβλημα που επηρεάζει την απόδοση του αλγορίθμου. Ένας κακός κατακερματισμός μπορεί να έχει ως αποτέλεσμα να εκτελείται μόνο ένα νήμα για ένα μεγάλο χρονικό διάστημα χάνοντας το πλεονέκτημα της χρήσης πολύ-ΕΠ. Όπως και η χρήση υπερβολικού αριθμού νημάτων έχει αρνητικά αποτελέσματα όσο αφορά την απόδοση ενός αλγόριθμου.

Με την χρήση τουλάχιστον κάποιον μοντέλων ανάπτυξης παράλληλων αλγορίθμων όπως το OpenMP μπορεί ο αλγόριθμος να θέτει κατά την ώρα της εκτέλεσης του αλγορίθμου τον αριθμό των νημάτων που θα εκτελούνται σε όλους ή

κάποιους κατακερματισμούς. Θα μπορούσε για παράδειγμα να ρωτήσει το ΛΣ πόσους ΕΠ είναι διαθέσιμοι και ανάλογα να θέσει τον αριθμό των νημάτων. Σε περίπτωση κακού κατακερματισμού των εργασιών μπορεί να είναι δυνατόν να γίνει ανάλυση της εκτέλεσης του αλγορίθμου με σκοπό να γίνουν αλλαγές στον αλγόριθμο έτσι ώστε να είναι δυνατός ένας καλύτερος κατακερματισμός.

4.7 Υπάρχουσα μελέτη παραλληλισμού του αλγορίθμου AES

Κατά καιρούς έχουν γίνει πολλές μελέτες για την βελτιστοποίηση της απόδοσης αλγορίθμων σε σχέση με την τεχνολογία των επεξεργαστών που ήταν διαθέσιμη εκείνη την στιγμή. Έχει γίνει ήδη μελέτη και ανάπτυξη μεθόδων υλοποίησης για την βελτίωση της απόδοσης της ταχύτητας εκτέλεσης των αλγορίθμων, RC4, SEAL, RC5, Blowfish και Khufu/Khafre, στον επεξεργαστή Intel Pentium.[47] Επίσης έχει γίνει προσπάθεια για την χρησιμοποίηση των δυνατοτήτων των 64bit επεξεργαστών, που προηγήθηκαν των επεξεργαστών με τεχνολογία πολλών πυρήνων, σε επιμέρους διεργασίες που λαμβάνουν χώρα μέσα σε έναν κρυπτογραφικό αλγόριθμο.[48]

Με σκοπό την παράλληλη υλοποίηση του αλγορίθμου AES, στην είδη υπάρχουσα μελέτη, έχει χρησιμοποιηθεί η προσέγγιση του Rafael R. Sevilla που έχει γράψει με την βοήθεια της γλώσσας προγραμματισμού C τον κρυπτογράφο του Rijndael ως module της Perl. Αυτή η επιλογή καθιστά δυνατό της αποδοτικότερης υλοποίησης παραλληλισμού βάση κάποιον πλεονεκτημάτων που παρουσιάζει ο συγκεκριμένος πηγαίος κώδικας (υψηλή σαφήνεια, υλοποιεί τους περισσότερους υπολογισμούς σε επαναληπτικούς βρόγχους, χρήση μικρού αριθμού συναρτήσεων). Προκείμενου να καταστεί δυνατή η κρυπτογράφηση και η αποκρυπτογράφηση του όποιου αριθμού μπλοκ δεδομένων, έχουν δημιουργηθεί δυο καινούριες συναρτήσεις, η Rijndael_enc() για την διαδικασία την κρυπτογράφησης και η Rijndael_dec() για την διαδικασία της αποκρυπτογράφησης , σε αναλογία με παρόμοιες συναρτήσεις που περιλαμβάνονται σε πηγαίους κώδικες της C κρυπτογραφικών αλγορίθμων. Παρακάτω θα γίνει μία εισαγωγή για τους τύπους εξαρτήσεων μεταξύ των δεδομένων πριν συζητηθεί η παραλληλοποίηση του αλγορίθμου AES.

Υπάρχουν τα ακόλουθα είδη εξαρτήσεων μεταξύ των δεδομένων που συμβαίνουν σε έναν επαναληπτικό βρόγχο:

- Η εξάρτηση ροής των δεδομένων (Data flow dependence): θα πρέπει δηλαδή να εξασφαλιστεί ότι έχει γένει η εγγραφή στη μεταβλητή πριν πάει να διαβαστεί. Ένα παράδειγμα αυτής της εξάρτησης είναι :

```
for (int i=0; i<n; i  )
    a[i] = a[i-1];
```

- Data Antidependence: θα πρέπει δηλαδή να εξασφαλιστεί ότι δεν θα διαβαστεί η μεταβλητή από έναν επεξεργαστικό πυρήνα πριν τελειώσει κάποιος παράλληλος υπολογισμός της μεταβλητής αυτής από κάποιον άλλον επεξεργαστικό πυρήνα.

```
for (int i=0; i<n; i  )
    a[i] = a[i+1];
```

- Output Dependence: είναι μία εξάρτηση πού ορίζει την σειρά εγγραφής στην ίδια μεταβλητή

```
for (int i=0; i<n; i  )
     a[0] = a[i];
```

Όλοι οι παραπάνω βρόγχοι δεν μπορούν να εκτελεστούν παράλληλα με αυτή την μορφή, επειδή το παραγόμενο αποτέλεσμα του παράλληλα εκτελούμενου βρόγχου είναι διαφορετικό από το αποτέλεσμα του σειριακού.

Ως εκ τούτου, είναι αναγκαίο να μετατραπεί ο τρόπος γραφής αυτών των βρόγχων ώστε να εξαλειφθούν αυτές οι εξαρτήσεις. Η εξάρτηση ροής δεδομένων δεν μπορεί να αποφευχθεί και έτσι περιορίζει τον παραλληλισμό του προγράμματος.

Η στρατηγική του παραλληλισμού του αλγορίθμου AES μπορεί να διαχωριστεί στα παρακάτω επιμέρους στάδια:

1. Την εξεύρεση της πλέον χρονοβόρας διαδικασίας του αλγορίθμου.
2. Τον προκαταρκτικό μετασχηματισμό των πολύ χρονοβόρων βρόγχων.
3. Την ανάλυση των εξαρτήσεων των δεδομένων των παραπάνω βρόγχων.
4. Εξάλειψη των εξαρτήσεων αυτών (όπου είναι δυνατόν).
5. Κατασκευή παράλληλων βρόγχων σύμφωνα με την βιβλιοθήκη που θα χρησιμοποιηθεί για την μετατροπή του πηγαίου κώδικα του αλγορίθμου σε παράλληλο.
6. Την επαλήθευση του μετατρεπομένου κώδικα.

Το αποτέλεσμα της παραπάνω διαδικασίας θα είναι ο παραλληλοποιημένος αλγόριθμος AES.[49]

Μετά από πολλά πειράματα στον σειριακό αλγόριθμο AES κρυπτογραφώντας και αποκρυπτογραφώντας 10 megabytes κειμένου διαπιστώθηκε ότι οι πιο χρονοβόρες συναρτήσεις είναι η Rijndael_enc() και η Rijndael_dec()

```
   void rijndael_enc(RIJNDAEL_context *ctx,
     UINT8 *input, int inputlen, UINT8 *output)
{
      int i, nblocks;
      nblocks = inputlen / RIJNDAEL_BLOCKSIZE;
        for (i = 0; i<nblocks; i++) {
            rijndael_encrypt(ctx, input, output);
            input+= RIJNDAEL_BLOCKSIZE;
            output+= RIJNDAEL_BLOCKSIZE;
      }
}
```

```
   void rijndael_dec(RIJNDAEL_context *ctx,
     UINT8 *input, int inputlen, UINT8 *output)
{
      int i, nblocks;
      nblocks = inputlen / RIJNDAEL_BLOCKSIZE;
        for (i = 0; i<nblocks; i++) {
             rijndael_decrypt(ctx, input, output);
             input+= RIJNDAEL_BLOCKSIZE;
             output+= RIJNDAEL_BLOCKSIZE;
      }
}
```

Οι πιο χρονοβόροι βρόγχοι περιλαμβάνονται στις συναρτήσεις Rijndael_enc() Rijndael_dec(), με αποτέλεσμα ο παραλληλισμός τους να είναι καίριας σημασίας για

την μείωση του συνολικού χρόνου εκτέλεσης του αλγορίθμου. Λαμβάνοντας υπόψη την μεγάλη ομοιότητα αυτών των βρόγχων θα μελετηθεί μόνο η μετατροπή του πρώτου βρόγχου. Προκειμένου να γίνει η ανάλυση των εξαρτήσεων μεταξύ των δεδομένων θα πρέπει το σώμα της συνάρτησης Rijndael_encrypt() να τοποθετηθεί μέσα στο βρόγχο. Στην συνέχεια θα εξαλειφθούν οι εξαρτίσεις με την εκτέλεση των ακόλουθων μεταβολών:

a) Εισαγωγή στην αρχή του βρόγχου τις δυο ακόλουθες καταχωρίσεις:
   "plaintext = &input[RIJNDAEL_BLOCKSIZE*i];"
   "ciphertext=&output[RIJNDAEL_BLOCKSIZE*i];"
b) Διαγραφή από το τέλος του βρόγχου τις παρακάτω δυο καταχωρίσεις:
   "input+= RIJNDAEL_BLOCKSIZE;"
   "output+= RIJNDAEL_BLOCKSIZE;"
c) Οι παρακάτω 8 μεταβλητές να καταστούν ιδιωτικές:
   "i", "plaintext", "ciphertext", "r", "j", "t", "e",
   "wtxt".

Ο πηγαίος κώδικας του βρόγχου μετατρέπεται σύμφωνα με τις παραπάνω ενδείξεις και είναι κατάλληλος για να εφαρμοστή η παρακάτω δομή σύμφωνα με το OpenMP API.

```
Void rijndael_enc(RIJNDAEL_context *ctx,
     UINT8 *input, int inputlen, UINT8 *output)
{
    int i, nblocks;
    const UINT8 *plaintext;
    UINT8 *ciphertext;
    int r, j;
    UINT32 wtxt[4], t[4], e;
    nblocks = inputlen / RIJNDAEL_BLOCKSIZE;
     #pragma omp parallel private (i, plaintext, ciphertext ,
     r,j, t, e, wtxt)
          #pragma omp for
          for (i = 0; i<nblocks; i++) {
              plaintext=&input[RIJNDAEL_BLOCKSIZE*i];
```

```
            ciphertext = &output[RIJNDAEL_BLOCKSIZE*i];
            key_addition_8to32(plaintext,   &(ctx->keys[0]),
        wtxt);
              for (r=1; r<ctx->nrounds; r++) {
                    for (j=0; j<4; j++) {
                      t[j] = dtbl[wtxt[j] & 0xff] ^
                        ROTRBYTE(dtbl[(wtxt[idx[1][j]] >> 8)
                                    & 0xff]^
                       ROTRBYTE(dtbl[(wtxt[idx[2][j]] >> 16) &
                                    0xff] ^
                       ROTRBYTE(dtbl[(wtxt[idx[3][j]] >> 24) &
                                    0xff])));
                    }
               key_addition32(t, &(ctx->keys[r*4]), wtxt);
             }
             for (j=0; j<4; j++) {
                   e = wtxt[j] & 0xff;
                   e |= (wtxt[idx[1][j]]) & (0xff << 8);
                   e |= (wtxt[idx[2][j]]) & (0xff << 16);
                  e |= (wtxt[idx[3][j]]) & (0xff << 24);
                  t[j] = e;
             }
        for (j=0; j<4; j++)
                 t[j] = SUBBYTE(t[j], sbox);
        key_addition32to8(t,&(ctx->keys[4*ctx->nrounds]),
                        ciphertext);
        }
    }
```

Όπου **“#pragma omp parallel”** ορίζει την αρχή της παράλληλα εκτελούμενης περιοχής, και **“#pragma omp for”** δηλώνει ότι όλες οι πράξεις της συγκεκριμένης επανάληψης μπορούν να εκτελεστούν παράλληλα. [49]

5 Μετρήσεις

5.1 Μετατροπή πηγαίου κώδικα

Όπως προαναφέρθηκε και στο προηγούμενο κεφάλαιο, δεν είναι δυνατό όλες οι επιμέρους εργασίες μέσα σε έναν αλγόριθμο να γίνουν παράλληλα. Αυτό συμβαίνει διότι οι επεξεργασία των δεδομένων πρέπει να ακολουθήσει κάποια συγκεκριμένη σειρά, όπως για παράδειγμα πρώτα να καταχωρηθεί μια συγκεκριμένη τιμή σε μία μεταβλητή και ύστερα να διαβαστεί. Αυτό πρέπει να γίνει σειριακά, γιατί αν προσπελαστεί η τιμή της μεταβλητής ταυτόχρονα με την εγγραφή της, το αποτέλεσμα που θα παραχθεί, δεν θα είναι το σωστό, αφού ως αποτέλεσμα θα δοθεί μία προηγούμενη τιμή που είχε η μεταβλητή και όχι αυτή που προσδοκείτε. Για αυτό τον λόγο, για να μετατραπεί ένας σειριακός κώδικας σε παράλληλο, πρέπει πρώτα να μελετηθούν οι εξαρτίσεις μεταξύ των δεδομένων (Data Dependences).

Κατά την διαδικασία μετατροπής του κώδικα θα πρέπει να ληφθεί υπόψη ότι στόχος είναι, η παράλληλη εκτέλεση των πιο χρονοβόρων(time-consuming) διαδικασιών, και όχι η μετατροπή όλου του κώδικα σε παράλληλο γιατί αυτό μπορεί να οδηγήσει σε αντίθετα αποτελέσματα από τα επιθυμητά. Αυτό θα γίνει διότι θα υπάρχουν πάρα πολλά νήματα προς εκτέλεση με κοινές μεταβλητές και αυτό θα οδηγήσει στην αναμονή αρκετών νημάτων μέχρι να τελειώσουν προηγούμενα που χρησιμοποιούν αυτές τις κοινές μεταβλητές. Ωστόσο, το λειτουργικό σύστημα θα χρησιμοποιήσει τους αδρανείς πόρους για την εκτέλεση άλλων διεργασιών. Ως συνέπεια αυτού θα έχει δημιουργηθεί ένας κώδικας που συνολικά στο χρόνο εκτέλεσης του θα είναι πιο χρονοβόρος και από τον σειριακό.

Στη παρούσα πτυχιακή εργασία χρησιμοποιήθηκε σειριακός πηγαίος κώδικας για τον αλγόριθμο κρυπτογράφησης AES, όπου παρουσιάζεται κάθε βήμα του κρυπτογραφικού αλγόριθμου ως συνάρτηση που καλείται στη main(). Παρακάτω θα παρατεθούν τα σειριακά κομμάτια του κώδικα από αυτές τις συναρτήσεις και η μετατροπή τους, καθώς και επεξηγήσεις αυτών και των εξαρτήσεων μεταξύ των δεδομένων.

```
    void KeyExpansion()
    {
      int i,j;
      unsigned char temp[4],k;

      // The first round key is the key itself.
      for(i=0;i<Nk;i++)
      {
            RoundKey[i*4]=Key[i*4];
            RoundKey[i*4+1]=Key[i*4+1];
            RoundKey[i*4+2]=Key[i*4+2];
            RoundKey[i*4+3]=Key[i*4+3];
      }

     // All other round keys are found from the previous round
keys.
      while (i < (Nb * (Nr+1)))
      {
            for(j=0;j<4;j++)
            {
            temp[j]=RoundKey[(i-1) * 4 + j];
            }
            if (i % Nk == 0)
            {
   // This function rotates the 4 bytes in a word to the left
once.
   // [a0,a1,a2,a3] becomes [a1,a2,a3,a0]
   // Function RotWord()
                  {
                  k = temp[0];
                  temp[0] = temp[1];
                  temp[1] = temp[2];
                  temp[2] = temp[3];
                  temp[3] = k;
                  }
```

```
   // SubWord() is a function that takes a four-byte input
word and
   // applies the S-box to each of the four bytes to produce
an output word.
                // Function Subword()
                {
                temp[0]=getSBoxValue(temp[0]);
                temp[1]=getSBoxValue(temp[1]);
                temp[2]=getSBoxValue(temp[2]);
                temp[3]=getSBoxValue(temp[3]);
                }

                temp[0] =  temp[0] ^ Rcon[i/Nk];
           }
           else if (Nk > 6 && i % Nk == 4)
           {
                // Function Subword()
                {
                temp[0]=getSBoxValue(temp[0]);
                temp[1]=getSBoxValue(temp[1]);
                temp[2]=getSBoxValue(temp[2]);
                temp[3]=getSBoxValue(temp[3]);
                }
           }
           RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];
           RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];
           RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];
           RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];
           i++;
      }
   }
```

Στο παραπάνω μπλοκ κώδικα, βλέπουμε 2 επαναληπτικούς βρόγχους **for** και μια **while**. Στον πρώτο επαναληπτικό βρόγχο **for** έχουμε 4 εντολές:

```
RoundKey[i*4]=Key[i*4];
RoundKey[i*4+1]=Key[i*4+1];
RoundKey[i*4+2]=Key[i*4+2];
RoundKey[i*4+3]=Key[i*4+3];
```

Όπως βλέπουμε κάθε εντολή ξεχωριστά απασχολεί διαφορετική μεταβλητή `RoundKey[]` και `Key[]` και δεν υπάρχει εξάρτηση δεδομένων μεταξύ των εντολών. Για αυτό τον λόγο μπορεί να χρησιμοποιηθεί η εντολή **pragma omp parallel for** που κατακερματίζει τον βρόγχο σε νήματα δηλώνοντας την μεταβλητή **i** ως ιδιωτική εφόσον για κάθε νήμα θα μπορεί να έχει διαφορετική τιμή. Αυτό το χρησιμοποιούμε κυρίως για μεταβλητές που μεταβάλλονται από ένα η παραπάνω νήματα.

```
#pragma omp parallel for default(shared) private(i)
for(i=0;i<Nk;i++)
{
  RoundKey[i*4]=Key[i*4];
  RoundKey[i*4+1]=Key[i*4+1];
  RoundKey[i*4+2]=Key[i*4+2];
  RoundKey[i*4+3]=Key[i*4+3];
}
```

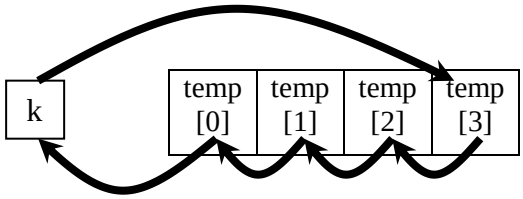
Ο βρόγχος **while** δεν μπορεί να εκτελεστεί παράλληλα για κάθε **i** διότι στο εσωτερικό του βρόγχου υπάρχουν αναδρομικές εντολές που χρησιμοποιούν τιμές μεταβλητών που υπολογίστηκαν στο βήμα **i-1.** Για αυτό το λόγο ο κατακερματισμός θα γίνει εσωτερικά του βρόγχου.

```
while (i < (Nb * (Nr+1)))
{
 #pragma omp parallel default(shared) private(i,Nk)
 #pragma omp for private(j)
 for(j=0;j<4;j++)
 {
       temp[j]=RoundKey[(i-1) * 4 + j];
 }
 …
```

5.2 Μέσα στο βρόγχο while υπάρχουν κάποιες εντολές που επεμβαίνουν σε δεδομένα στα οποία υπάρχει εξάρτηση ροής δεδομένων.

```
k = temp[0];
temp[0] = temp[1];
temp[1] = temp[2];
temp[2] = temp[3];
temp[3] = k;

temp[0]=getSBoxValue(temp[0]);
temp[1]=getSBoxValue(temp[1]);
temp[2]=getSBoxValue(temp[2]);
temp[3]=getSBoxValue(temp[3]);

temp[0] =  temp[0] ^ Rcon[i/Nk];
```

Εκτελείτε μια κύλιση των στοιχείων του πίνακα temp μια θέση δεξιά. Αυτό δεν μπορεί να γίνει παράλληλα διότι η εγγραφή των δεδομένων γίνεται πάνω στις θέσεις του πίνακα.



Η εκτέλεση αυτής τις διαδικασίας μπορεί να γίνει και παράλληλα. Αυτό προϋποθέτει όμως μετατροπή του τρόπου προσέγγισης και διαφοροποίηση όλων των εντολών. Θα πρέπει δηλαδή να δημιουργηθεί ένας δεύτερος πίνακας (π.χ shift) ίδιου μεγέθους με τον πίνακα temp, να αντιγραφούν τα στοιχεία του πίνακα temp στον πίνακα shift και να δηλωθούν ποια συγκεκριμένα στοιχεία του πίνακα shift θα αντικαταστήσουν συγκεκριμένα στοιχεία του πίνακα temp. Ο συγκεκριμένος τρόπος έχει το μειονέκτημα ότι χρησιμοποιεί περισσότερους πόρους και για μεγάλα μεγέθη πινάκων, καθιστάτε πολύ πιο χρονοβόρος σε σχέση με τον σειριακό τρόπο εκτέλεσης.

Για τους παραπάνω λόγους, οι συγκεκριμένες εντολές θα πρέπει να εκτελεστούν σειριακά. Αυτό μπορεί να επιτευχθεί δηλώνοντας αυτό το μπλοκ των εντολών να εκτελεσθεί από ένα και μόνο νήμα.

```
            #pragma omp single nowait
                  {
                  k = temp[0];
                  temp[0] = temp[1];
                  temp[1] = temp[2];
                  temp[2] = temp[3];
                  temp[3] = k;

                  temp[0]=getSBoxValue(temp[0]);
                  temp[1]=getSBoxValue(temp[1]);
                  temp[2]=getSBoxValue(temp[2]);
                  temp[3]=getSBoxValue(temp[3]);

                  temp[0] =  temp[0] ^ Rcon[i/Nk];
                  }
```

Για το βήμα SubBytes, στο οποίο τα στοιχεία του πίνακα, στον οποίο έχουν αποθηκευτεί τα δεδομένα προς κρυπτογράφηση, ενημερώνονται μέσω του S-box, χρησιμοποιείται μια επανάληψη for για δισδιάστατο πίνακα.

```
void SubBytes()
{
      int i,j;
      for(i=0;i<4;i++)
      {
            for(j=0;j<4;j++)
            {
                  state[i][j] = getSBoxValue(state[i][j]);

            }
      }
}
```

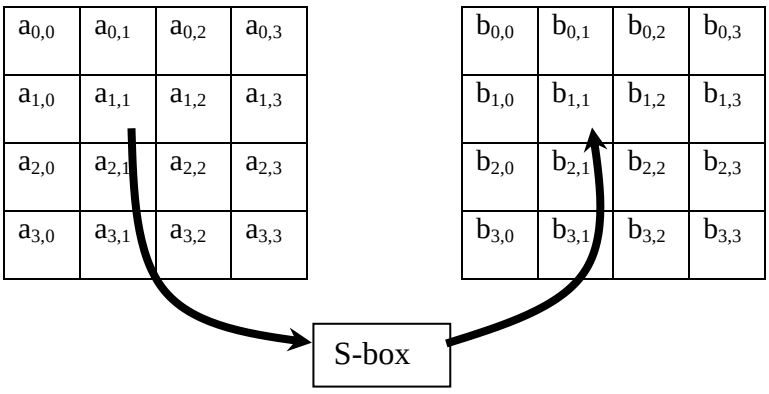
Δεν υπάρχει καμία εξάρτηση δεδομένων στην παραπάνω διαδικασία για να εμποδίζει την παράλληλη εκτέλεση της. Το μόνο που χρειάζεται είναι η δήλωση της επανάληψης αυτής ως παράλληλα εκτελούμενη. Δηλαδή με την OpenMP το μόνο που χρειάζεται είναι η προσθήκη της εντολής parallel for.

```
void SubBytes()
{
      int i,j;
```

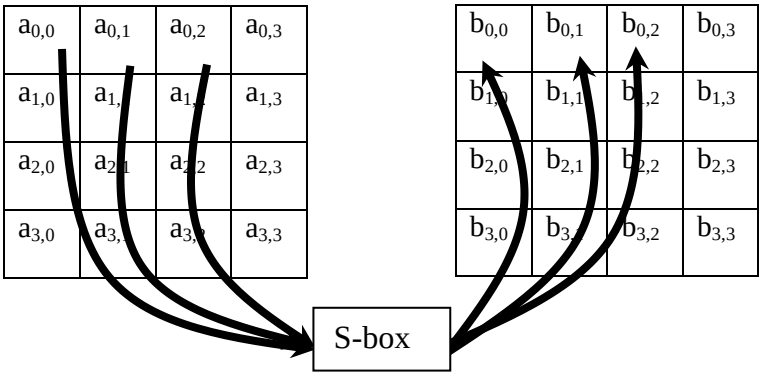
```
	#pragma omp parallel for default(shared) private (i,j)
	for(i=0;i<4;i++)
	{
		for(j=0;j<4;j++)
		{
			state[i][j] = getSBoxValue(state[i][j]);

		}
	}
}
```

Με την μετατροπή αυτή, η διαδικασία που κατά την συριακή εκτέλεση, κάθε κελί του πίνακα ενημερώνεται μέσω του S-box και αποθηκεύεται και πάλι στο αντίστοιχο κελί, εκτελείται παράλληλα, ανάλογα με τον αριθμό των επεξεργαστικών πυρήνων, ενημερώνοντας ταυτόχρονα τόσα κελιά όσοι και οι επεξεργαστικοί πυρήνες.

Σειριακή Εκτέλεση



Παράλληλη Εκτέλεση



Στο βήμα ShiftRows, δεν μπορεί να γίνει καμία αλλαγή και αυτό διότι υπάρχει και πάλι εξάρτηση ροής δεδομένων. Όπως και στην προηγούμενη διαδικασία με την ίδια εξάρτηση, υπάρχει τρόπος να εκτελεστεί παράλληλα αλλά είναι πολύ πιο χρονοβόρος και χρησιμοποίει περισσότερους πόρους.

```
void ShiftRows()
{
	unsigned char temp;
	// Rotate first row 1 columns to left
	temp=state[1][0];
	state[1][0]=state[1][1];
	state[1][1]=state[1][2];
	state[1][2]=state[1][3];
	state[1][3]=temp;

	// Rotate second row 2 columns to left
	temp=state[2][0];
	state[2][0]=state[2][2];
	state[2][2]=temp;

	temp=state[2][1];
	state[2][1]=state[2][3];
	state[2][3]=temp;

	// Rotate third row 3 columns to left
	temp=state[3][0];
	state[3][0]=state[3][3];
	state[3][3]=state[3][2];
	state[3][2]=state[3][1];
	state[3][1]=temp;
}
```

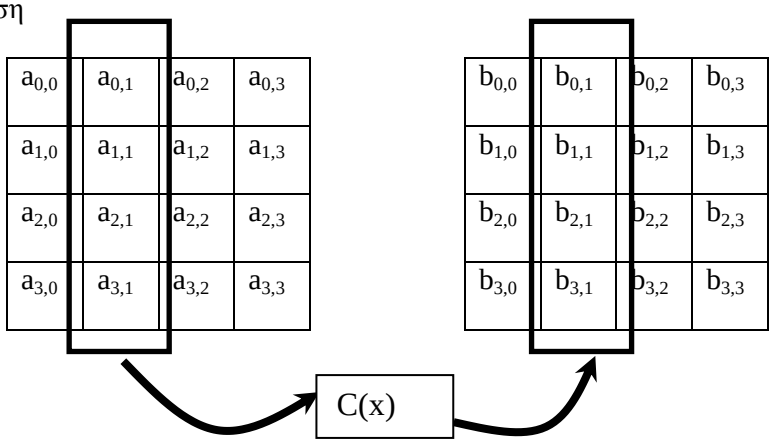
Στο βήμα MixColumns υπάρχει μόνο μια επαναληπτική δομή for η οποία δεν κρύβει καμία εξάρτηση δεδομένων μεταξύ των μεταβλητών που περιέχει. Οπότε με μια απλή δήλωση parallel for, ο επαναληπτικός βρόγχος μπορεί να εκτελεστεί παράλληλα.

```
void MixColumns()
{
	int i;
	unsigned char Tmp,Tm,t;
	#pragma omp parallel for default(shared) private(i)
	for(i=0;i<4;i++)
	{
```

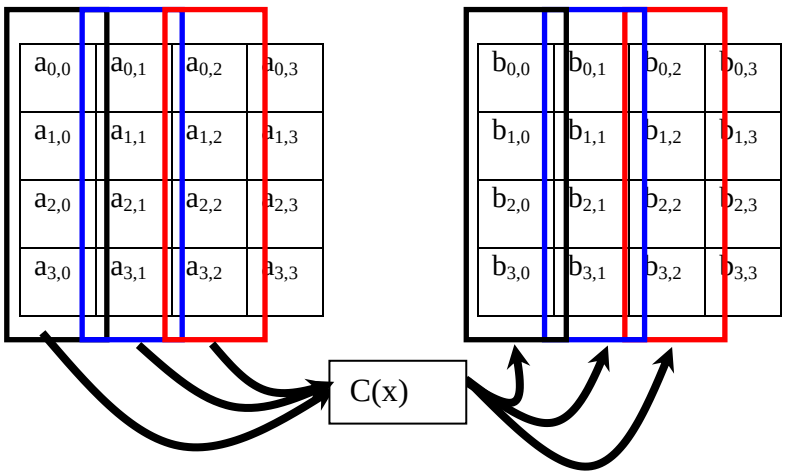
```
            t=state[0][i];
            Tmp  =  state[0][i]  ^  state[1][i]  ^  state[2][i]  ^
state[3][i] ;
            Tm  =  state[0][i]  ^  state[1][i]  ;  Tm  =  xtime(Tm);
state[0][i] ^= Tm ^ Tmp ;
            Tm  =  state[1][i]  ^  state[2][i]  ;  Tm  =  xtime(Tm);
state[1][i] ^= Tm ^ Tmp ;
            Tm  =  state[2][i]  ^  state[3][i]  ;  Tm  =  xtime(Tm);
state[2][i] ^= Tm ^ Tmp ;
            Tm  =  state[3][i]  ^  t  ;  Tm  =  xtime(Tm);  state[3][i]
^= Tm ^ Tmp ;
      }
}
```

Η παραπάνω μετατροπή έχει ως αποτέλεσμα την παρακάτω διαφοροποίηση κατά την εκτέλεση του κώδικα.

Σειριακή Εκτέλεση



Παράλληλη Εκτέλεση

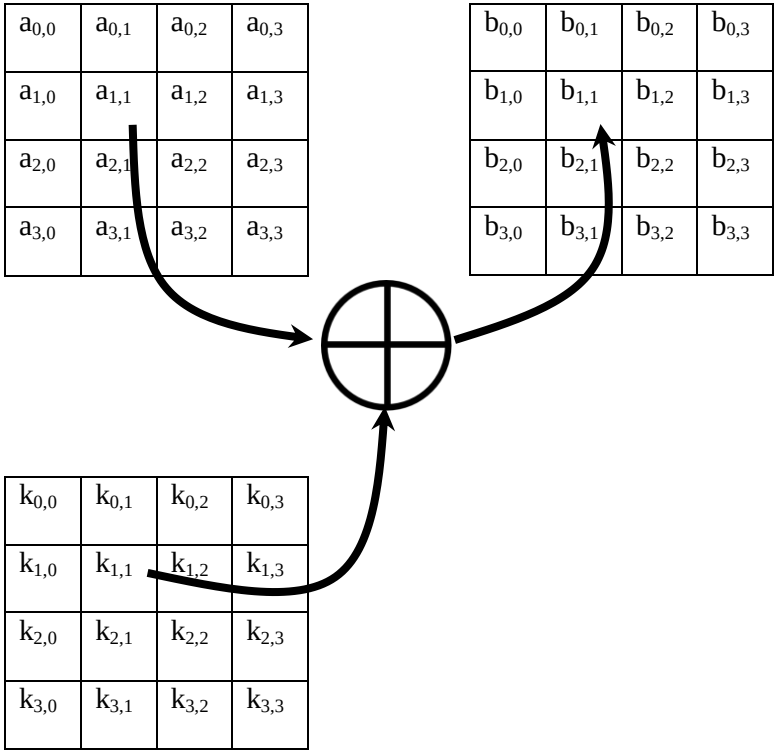


Στο τελευταίο βήμα AddRoundKey για την υλοποίηση του χρησιμοποιείται και πάλι ένας επαναληπτικός βρόγχος for για δυσδιάστατο πίνακα και δεν υπάρχουν εξαρτήσεις μεταξύ των δεδομένων. Οπότε η μετατροπή του θα είναι ακριβώς ίδια όπως και στο βήμα SubBytes.

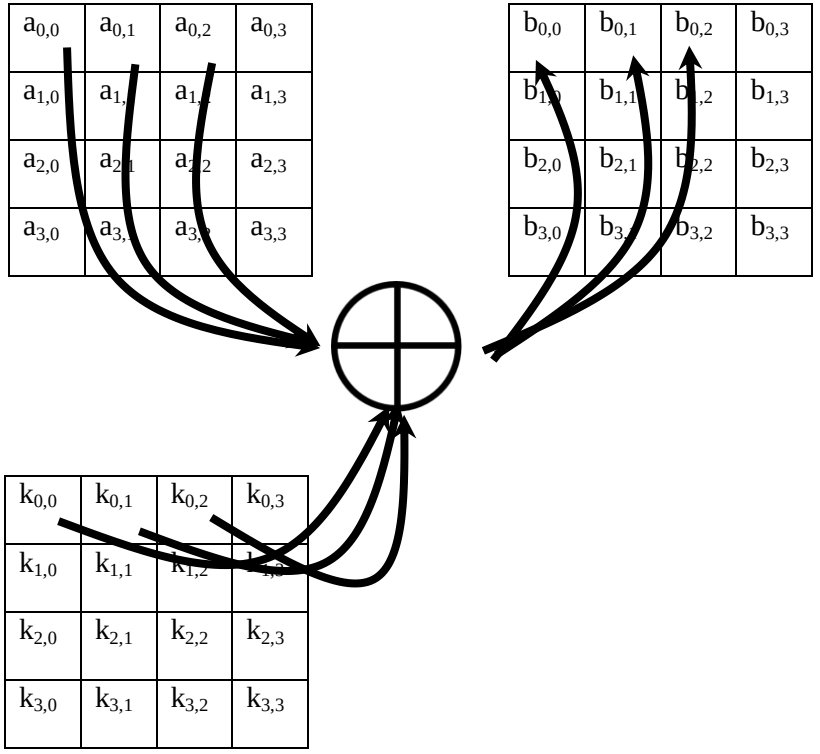
```
void AddRoundKey(int round)
{
	int i,j;
	for(i=0;i<4;i++)
	{
		for(j=0;j<4;j++)
		{
		state[j][i] ^= RoundKey[round * Nb * 4 + i * Nb+j];
		}
	}
}
```

```
void AddRoundKey(int round)
{
	int i,j;
	#pragma omp parallel for default(shared) private(i,j)
	for(i=0;i<4;i++)
	{
		for(j=0;j<4;j++)
		{
		state[j][i] ^= RoundKey[round * Nb * 4 + i * Nb+j];
		}
	}
}
```

Σειριακή Εκτέλεση



Παράλληλη Εκτέλεση



Κατά την διαδικασία της αποκρυπτογράφησης οι μετασχηματισμοί της διαδικασίας κρυπτογράφησης αντιστρέφονται και τοποθετούνται σε αντίστροφη σειρά. Έτσι όπως και κατά την κρυπτογράφηση, υπάρχουν τέσσερις διακριτοί μετασχηματισμοί, οι InvShiftRows, InvSubBytes, InvMixColumns και AddRoundKey. Οι τέσσερις αυτές διαδικασίες έχουν την ίδια δομή με αυτές της κρυπτογράφησης και οι μετατροπές που γίνονται στο πηγαίο κώδικα είναι οι ίδιες. Οι διαφορές κατά τον τρόπο εκτέλεσης τους, σειριακά και παράλληλα, αποτυπώνονται ακριβώς με τον ίδιο τρόπο όπως και στην διαδικασία κρυπτογράφησης.

5.2 Μέτρηση γεγονότων επεξεργαστή και μετρήσεις χρόνων εκτέλεσης.

Παρακάτω αναρτώνται τα αποτελέσματα για τους χρόνους εκτέλεσης ανάλογα με το μέγεθος του κλειδιού, το μέγεθος του plaintext και του κώδικα , σειριακού και παράλληλου.

Encode AES

		Μέγεθος plaintext σε bytes									
Serial	**key**		**16**	**32**	**64**	**128**	**1024**	**2048**	**10240**	**16384**	**100K**	**1M**
		128	0,04ms	0,08ms	0,12ms	0,21ms	1,76ms	3,96ms	14,43ms	19,33ms	166,69ms	2164,83ms
		192	0,04ms	0,08ms	0,12ms	0,25ms	1,87ms	4,16ms	15,61ms	21,61ms	184,39ms	2534,25ms
		256	0,04ms	0,08ms	0,11ms	0,19ms	1,65ms	3,54ms	13,67ms	18,12ms	158,03ms	1836,37ms

Πίνακας 5-1: Χρόνοι εκτέλεσης του κώδικα κρυπτογράφησης κατά την σειριακή εκτέλεση του σε σχέση με το μέγεθος του plaintext

		Μέγεθος plaintext σε bytes									
OpenMP	**key**		**16**	**32**	**64**	**128**	**1024**	**2048**	**10240**	**16384**	**100K**	**1M**
		128	0,08ms	0,16ms	0,35ms	0,67ms	3,77ms	6,62ms	15,84ms	21,33ms	136,48ms	1287,83ms
		192	0,08ms	0,16ms	0,37ms	0,73ms	4,08ms	7,21ms	17,32ms	22,47ms	152,62ms	1654,43ms
		256	0,08ms	0,17ms	0,41ms	0,79ms	4,16ms	7,98ms	18,65ms	24,75ms	171,84ms	1838,94ms

Πίνακας 5-2: Χρόνοι εκτέλεσης του κώδικα κρυπτογράφησης κατά την παράλληλη εκτέλεση του σε σχέση με το μέγεθος του plaintext

Decode AES

		Μέγεθος plaintext σε bytes									
Serial	**key**		**16**	**32**	**64**	**128**	**1024**	**2048**	**10240**	**16384**	**100K**	**1M**
		128	0,04ms	0,08ms	0,11ms	0,22ms	1,71ms	3,66ms	13,87ms	19,54ms	169,47ms	1929,42ms
		192	0,04ms	0,08ms	0,13ms	0,23ms	1,79ms	4,17ms	14,77ms	20,43ms	177,84ms	2284,76ms
		256	0,04ms	0,08ms	0,13ms	0,28ms	1,94ms	4,33ms	16,37ms	23,15ms	199,39ms	2783,29ms

Πίνακας 5-3: Χρόνοι εκτέλεσης του κώδικα αποκρυπτογράφησης κατά την σειριακή εκτέλεση του σε σχέση με το μέγεθος του plaintext

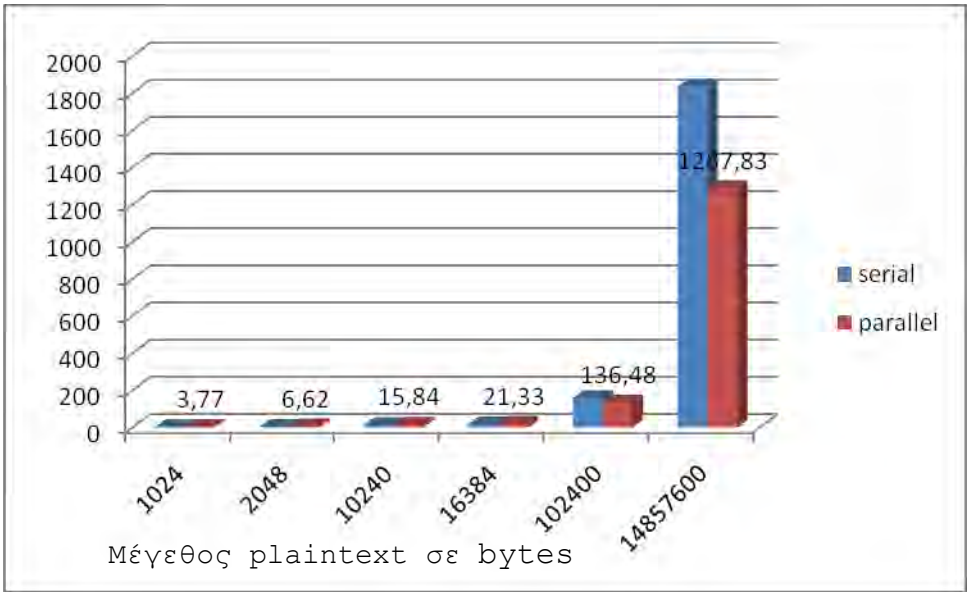
		Μέγεθος plaintext σε bytes									
OpenMP	**key**		**16**	**32**	**64**	**128**	**1024**	**2048**	**10240**	**16384**	**100K**	**1M**
		128	0,04ms	0,16ms	0,41ms	0,77ms	3,98ms	6,84ms	16,04ms	21,67ms	138,85ms	1347,48ms
		192	0,04ms	0,16ms	0,44ms	0,84ms	4,27ms	7,39ms	17,85ms	22,59ms	155,94ms	1793,28ms
		256	0,04ms	0,17ms	0,49ms	0,92ms	4,32ms	8,14ms	18,94ms	25,08ms	177,47ms	1985,46ms

Πίνακας 5-4: Χρόνοι εκτέλεσης του κώδικα αποκρυπτογράφησης κατά την παράλληλη εκτέλεση του σε σχέση με το μέγεθος του plaintext

Όπως ήταν αναμενόμενο για μικρά μεγέθη αρχικού κειμένου κατά την παράλληλη εκτέλεση τους ο χρόνο εκτέλεσης σε σχέση με τον σειριακό είναι πολύ μεγαλύτερος

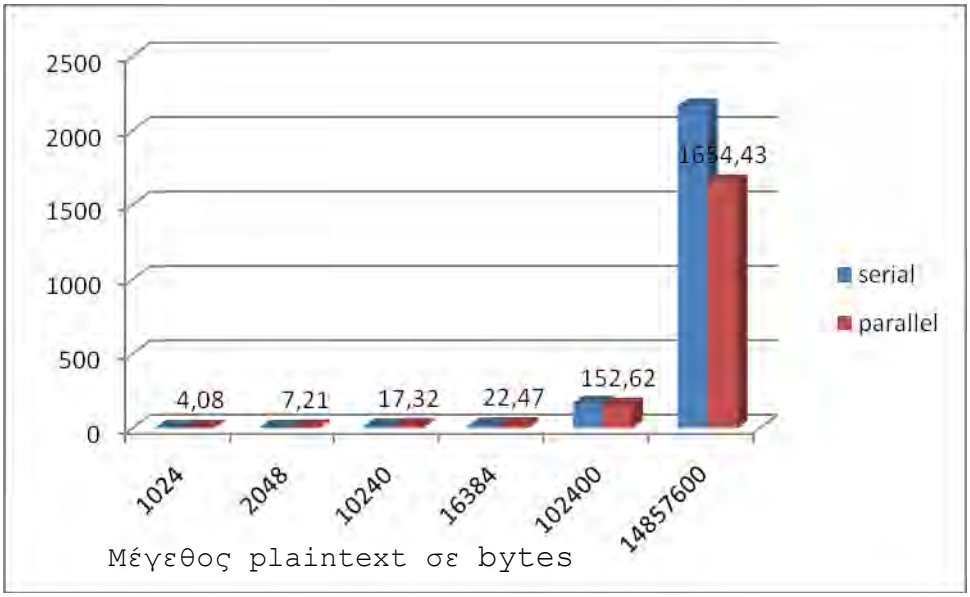
.Παρακάτω παρουσιάζονται τα γραφήματα που δείχνουν την διαφορά του χρόνου εκτέλεσης της κρυπτογράφησης, σειριακά και παράλληλα , σε σχέση με το μέγεθος του αρχικού κειμένου, για κάθε μέγεθος κλειδιού ξεχωριστά.

AES Encode – 128bit key

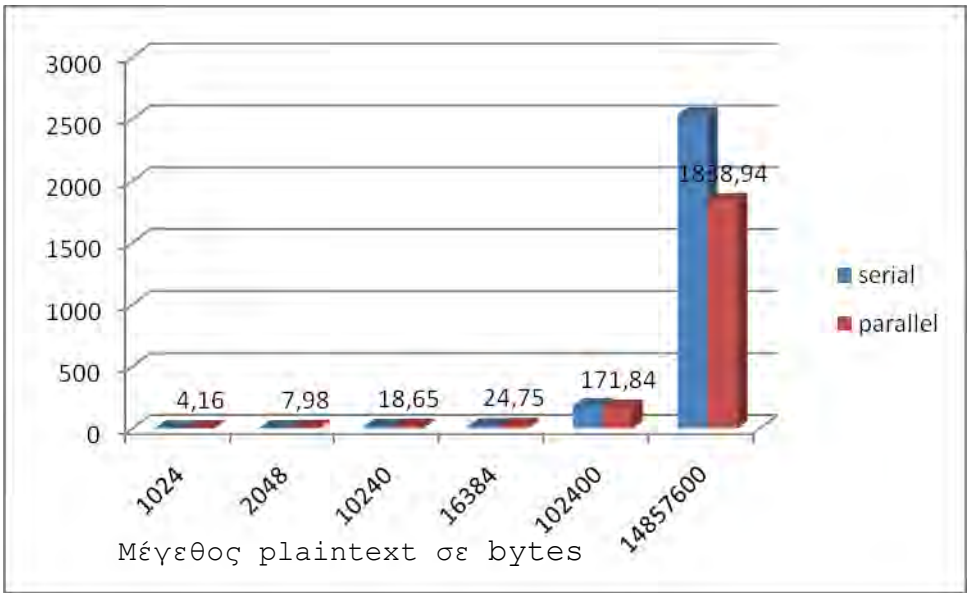


Γράφημα 5-1: Παρουσίαση των χρόνων εκτέλεσης σε msec για σειριακή και παράλληλη εκτέλεση κρυπτογράφησης για κλειδί μεγέθους 128bit

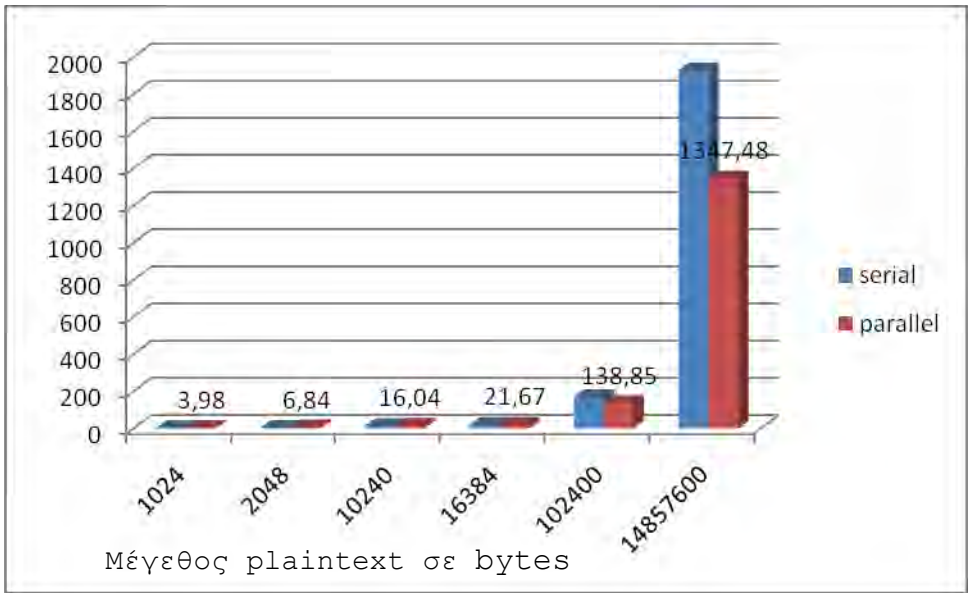
AES Encode – 192bit key



Γράφημα 5-2: Παρουσίαση των χρόνων εκτέλεσης σε msec για σειριακή και παράλληλη εκτέλεση κρυπτογράφησης για κλειδί μεγέθους 192bit

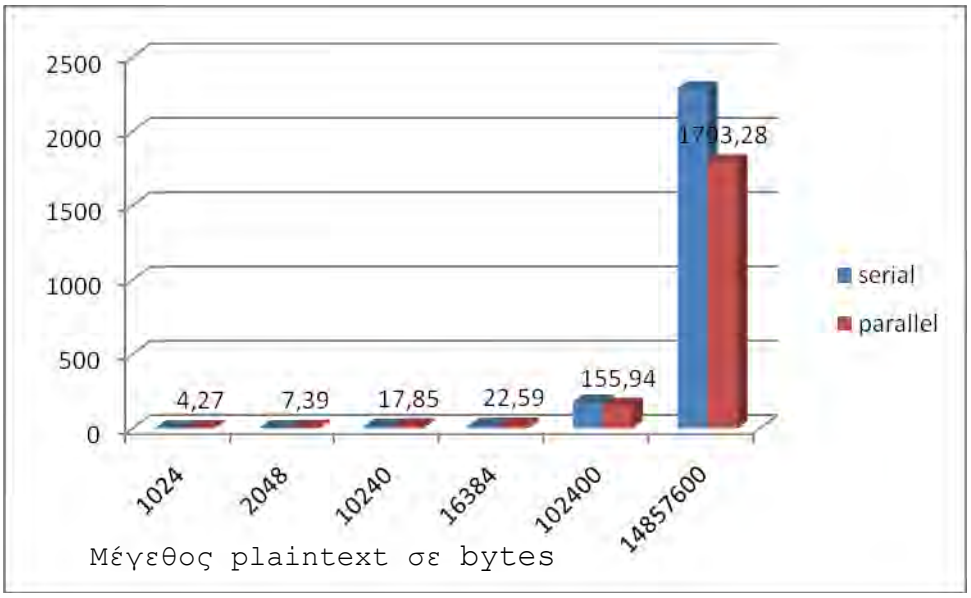


Γράφημα 5-3: Παρουσίαση των χρόνων εκτέλεσης σε msec για σειριακή και παράλληλη εκτέλεση κρυπτογράφησης για κλειδί μεγέθους 256bit



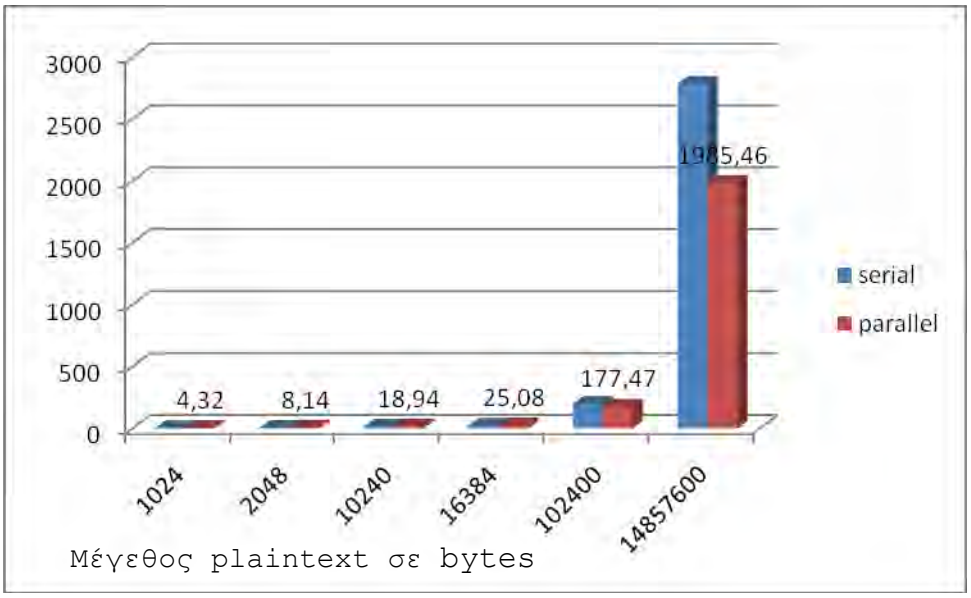
Γράφημα 5-4: Παρουσίαση των χρόνων εκτέλεσης σε msec για σειριακή και παράλληλη εκτέλεση αποκρυπτογράφησης για κλειδί μεγέθους 128bit

AES Decode – 192bit key



Γράφημα 5-5: Παρουσίαση των χρόνων εκτέλεσης σε msec για σειριακή και παράλληλη εκτέλεση αποκρυπτογράφησης για κλειδί μεγέθους 192bit

AES Decode – 256bit key



Γράφημα 5-6: Παρουσίαση των χρόνων εκτέλεσης σε msec για σειριακή και παράλληλη εκτέλεση αποκρυπτογράφησης για κλειδί μεγέθους 256bit

Τα οφέλη της μετατροπής με την βιβλιοθήκη OpenMp φαίνονται σε επεξεργασία μεγάλου όγκου δεδομένων ενώ για μικρό όγκο έχουμε αντίθετα αποτελέσματα. Αυτό φαίνεται και από το παρακάτω πίνακα.

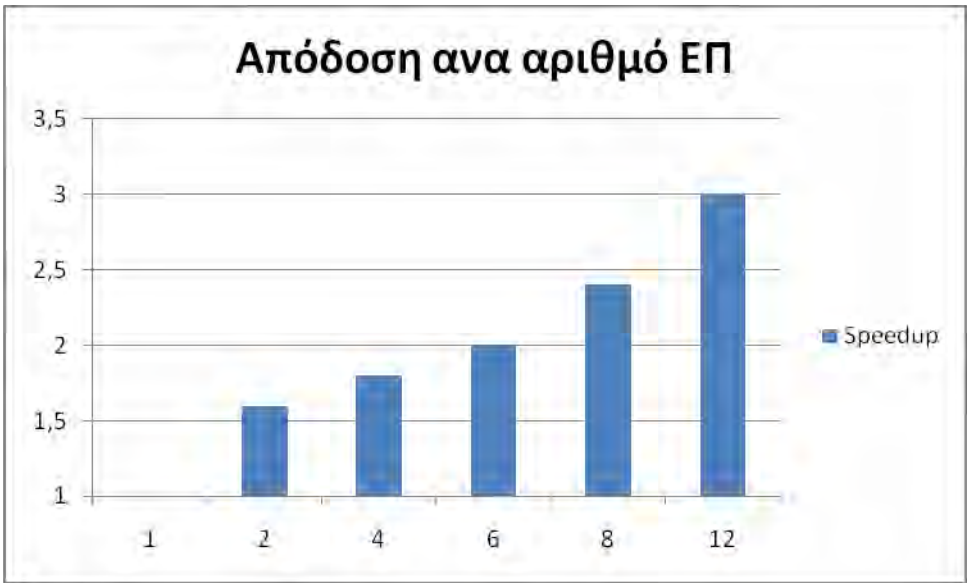
Serial	Encode AES			Decode AES			
	Key			key			
	128	192	256	128	192	256	
	1	1	1	1	1	1	Cycles * 100000
	0,000750%	0,000390%	0,000340%	0,000390%	0,013000%	0,010000%	CPU Usage

OpenMP	Encode AES			Decode AES			
	Key			key			
	128	192	256	128	192	256	
	2	2	2	1	1	1	Cycles * 100000
	0,004300%	0,007700%	0,010000%	0,013000%	0,013000%	0,010000%	CPU Usage

Πίνακας 5-5: Παρουσίαση της χρήσης του επεξεργαστή κατά την εκτέλεση της κρυπτογράφησης και αποκρυπτογράφησης, σειριακά και παράλληλα με την χρήση της OpenMP, καθώς και το αριθμό των κύκλων επεξεργασίας.

Παρατηρείται ότι κατά την εκτέλεση και του encode και του decode, μετά την μετατροπή με την βιβλιοθήκη OpenMP, για μέγεθος αρχικού κειμένου 16 bytes χρησιμοποιούνται και περισσότεροι κύκλοι επεξεργασία και υπάρχει μεγαλύτερη απασχόληση του επεξεργαστή, δηλαδή χρήση περισσότερων πόρων, σε σχέση με την σειριακή εκτέλεση για το συγκεκριμένο μέγεθος του αρχικού κειμένου.

Εν κατακλείδι, για μεγάλο όγκο δεδομένων ο δείκτης επιτάχυνσης της παράλληλης εκτέλεσης, είναι ανάλογος με τον αριθμό των επεξεργαστικών πυρήνων.



6 Συμπεράσματα

Για την αξιοποίηση των πόρων των διαφόρων πολύ-ΕΠ πρέπει, όπως και στις περιπτώσεις των συστημάτων με πολλαπλούς επεξεργαστές και επεξεργαστών πολλαπλών νημάτων, να γραφτεί η εφαρμογή με παράλληλο προγραμματισμό χρησιμοποιώντας μία βιβλιοθήκη – μοντέλο κοινής μνήμης. Από τις βιβλιοθήκες-μοντέλα κοινής μνήμης που εξετάσθηκαν υπήρχαν κάποιες που άφηναν το βάρος της διαχείρισης του κατακερματισμού στον προγραμματιστή, ενώ υπήρχαν κάποιες που μείωναν κατά πολύ αυτό το βάρος. Από τις τελευταίες ξεχωρίζει το μοντέλο OpenMP που δίνει στο προγραμματιστή την δυνατότητα να μετατρέπει σειριακούς αλγορίθμους σε παράλληλους με την προσθήκη μερικών γραμμών έχοντας ταυτόχρονα πολύ καλή απόδοση. Το μοντέλο OpenMP μπορεί να αναλάβει το πως θα γίνει ο κατακερματισμός αφήνοντας στον προγραμματιστή να αποφασίσει μόνο που θα γίνει ο κατακερματισμός. Αυτό χωρίς να εμποδίζει τον προγραμματιστή να παραμετροποίηση το πώς θα γίνει ο κατακερματισμός.

Η βελτίωση της απόδοσης από την μετατροπή ενός σειριακού αλγορίθμου σε παράλληλο, εξαρτάται κατά κύριο λόγο από τον ίδιο τον αλγόριθμο. Επίσης μεγάλο ρόλο παίζουν και τα μεγέθη των δεδομένων που θα επεξεργάζεται ο αλγόριθμος ανάλογα με τις ανάγκες του καθενός αλλά και η βιβλιοθήκη-μοντέλο που θα χρησιμοποιηθεί. Υπάρχουν αλγόριθμοι που μπορούν να κατακερματιστούν σχεδόν πλήρως και παρουσιάζουν βελτίωση απόδοσης πολλαπλάσια των αριθμό των ΕΠ σε πολύ-ΕΠ. Κάποιοι άλλοι που μπορούν να κατακερματιστούν άλλα ένα τμήμα του αλγορίθμου εκτελείται σειριακά και επομένως δεν έχουν τόσο μεγάλη βελτίωση της απόδοσης. Τέλος κάποιοι αλγόριθμοι δεν μπορούν να κατακερματιστούν και επομένως δεν παρουσιάζουν βελτίωση στην απόδοση τους. Στην περίπτωση των block ciphers το κέρδος από μια τέτοια μετατροπή είναι αρκετά μικρότερο σε σχέση από παρόμοια μετατροπή ενός stream cipher, διότι δεν υπάρχουν χρονικές εξαρτίσεις οπότε κάποιες διαδικασίες μπορούν να ανατεθούν σε άλλον επεξεργαστικό πυρήνα ή σε ξεχωριστά threads. Και στις δυο περίπτωσης όμως, δηλαδή και των block ciphers αλλά και των stream ciphers, τα οφέλη της μετατροπή με την βιβλιοθήκη OpenMP φαίνονται όσο μεγαλώνει το μέγεθος του plaintext.

7 Αναφορές –Βιβλιογραφία

[1] AMD, http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_15331,00.html (ημερομηνία προσπέλασης: 18/9/2008)

[2] INTEL, http://www.intel.com/products/desktop/processors/index.htm?iid=processors_body+dt_core (ημερομηνία προσπέλασης: 18/9/2008)

[3] Shane Rau, Michael Shirer, http://www.idc.com/getdoc.jsp?containerId=prUS21263208 (ημερομηνία προσπέλασης: 7/9/2008)

[4] Ofri Wechsler, ftp://download.intel.com/technology/architecture/new_architecture_06.pdf (ημερομηνία προσπέλασης: 27/9/2008)

[5] Richard Friedman, http://openmp.org/wp/about-openmp/ (ημερομηνία προσπέλασης: 26/9/2008)

[6] Gordon E. Moore, Cramming more componentsonto integrated circuits, Electronics, 38, 1965, σελ.

[7] JEFFREY A. DAVIS, RAGURAMAN VENKATESAN, ALAIN KALOYEROS,MICHAEL BEYLANSKY, SHUKRI J. SOURI, KAUSTAV BANERJEE, KRISHNA C. SARASWAT, , ARIFUR RAHMAN, RAFAEL REIF, JAMES D. MEINDL, Interconnect limits on gigascale integration, Proceedings of the IEEE, 89, 2001, σελ. 305-324

[8] Matthew Curtis-Maury, Xiaoning Ding, Christos D. Antonopoulos, Dimitrios S. Nikolopoulos, An Evaluation of OpenMP on Current and Emerging Multithreaded/Multicore Processors , Lecture Notes in Computer Science, 4315, 2008, σελ. 133-144

[9] Blaise Barney, https://computing.llnl.gov/tutorials/parallel_comp (ημερομηνία προσπέλασης: 24/9/2008)

[10] Microsoft, http://www.microsoft.com/licensing/highlights/multicore.mspx (ημερομηνία προσπέλασης: 21/9/2008)

[11] Suresh Siddha, http://oss.intel.com/pdf/mclinux.pdf (ημερομηνία προσπέλασης: 21/9/2008)

[12] Christian Terboven, Dieter an Mey and Samuel Sarholz, A Practical Programming Model for the Multi-Core Era / OpenMP on Multicore Architectures (Springer Berlin / Heidelberg,2008)

[13] G. A. Geist, J.A. Kohl, P.M. Papadopoulos, PVM and MPI: a comparison of features, Calculateurs Paralleles, 8, 1996, σελ.

[14] various, http://www-unix.mcs.anl.gov/mpi/ (ημερομηνία προσπέλασης: 26/9/2008)

[15] various, http://www.csm.ornl.gov/pvm/ (ημερομηνία προσπέλασης: 26/9/2008)

[16] various, http://en.wikipedia.org/wiki/Linux_kernel#Technical_features (ημερομηνία προσπέλασης: 21/9/2008)

[17] AMD, http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/33425.pdf (ημερομηνία προσπέλασης: 1/10/2008)

[18] INTEL, http://www.intel.com/Products/Desktop/Processors/PentiumD/PentiumD-technicaldocuments.htm (ημερομηνία προσπέλασης: 28/9/2008)

[19] David Kanter, http://www.realworldtech.com/page.cfm?ArticleID=RWT101405234615&p=1 (ημερομηνία προσπέλασης: 28/9/2008)

[20] Lu Peng, Jih-Kwon Peir ,Tribuvan K. Prakash ,Carl Staelin , Yen-Kuang Chen, David Koppelman, Memory hierarchy performance measurement of commercial dual-core desktop processors, Journal of Systems Architecture, 54, 2008, σελ. 816–828

[21] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, Kunyung Chang, The Case for a Single-Chip Multiprocessor (ACM,1996)

[22] L. Hammond, B.Nayfeb, K.Olukotun, A Single-ChipMultiprocessor, IEEE Computer, 30, 1997, σελ. 79-85

[23] Jie Tao, Marcel Kunze, Fabian Nowak,Rainer Buchty, Wolfgang Karl, Performance Advantage of Reconfigurable CacheDesign on Multicore Processor Systems, Int J Parallel Programming, 36, 2008, σελ. 347-360

[24] INTEL, http://www.intel.com/products/processor/core2quad/index.htm (ημερομηνία προσπέλασης: 28/9/2008)

[25] Ilya Gavrichenkov, http://www.xbitlabs.com/articles/cpu/display/core2duo-preview.html (ημερομηνία προσπέλασης: 2/10/2008)

[26] Scott Wasson, http://techreport.com/articles.x/11160/1 (ημερομηνία προσπέλασης: 28/9/2008)

[27] Suresh Siddha, Venkatesh Pallipadi, Asit Mallick, Process Scheduling Challenges in the Era of Multi-core Processors, , 11, 2007, σελ. 361-370

[28] Tim Smalley, http://www.bit-tech.net/hardware/2006/07/14/intel_core_2_duo_processors/3 (ημερομηνία προσπέλασης: 2/10/2008)

[29] David Kanter, http://www.realworldtech.com/page.cfm?ArticleID=RWT051607033728&p=2 (ημερομηνία προσπέλασης: 1/10/2008)

[30] James Burns, Jean-Luc Gaudiot: Area and System Clock Effects on SMT/CMP Processors., Proceedings of the 2001 international Conference on Parallel Architectures and Compilation Techniques, Σεπτέμβριος, 2001, PACT. IEEE Computer Society, Washington, DC, 211., , σελ. 211-

[31] John Goodacre, The Design Dilemma:Multiprocessing Using Multiprocessors and Multithreading, ARM Information Quarterly, 5, 2006, σελ. 20-23

[32] Jon Stokes, http://arstechnica.com/articles/paedia/cpu/hyperthreading.ars/1 (ημερομηνία προσπέλασης: 5/10/2008)

[33] S. Goldwasser, S. Micali, C. Rackoff, «The Knowledge Complexity of Interactive Proof Systems», *Proc. of the 17th ACM Symposium on the Theory of the Computing*, 1985, pp. 291 – 304.

[34] V.Miller, «Uses of Elliptic Curves in Cryptography», Proc. inAdvances in Cryptography – Crypto ’85, *Lecture Notes in Computer Science*,218, Springer – Verlag, 1986, pp. 17 – 26.

[35] R. Rivest, A. Shamir, L. Adleman, «A Method of Obtaining DigitalSignatures and Public – Key Cryptosystems», *J. Communications of the ACM*,vol. 21, 1978, pp. 120 – 126.

[36] C. E. Shannon, «Communication Theory of Secrecy Systems»,*Bell System Technical Journal*, vol. 28, no. 4, 1999, pp. 656 – 715

[37] FIPS. Advanced Encryption Standard (AES). FIPS PUB-197, *April 2, 2001*

[38] AES Lounge, http://www.iaik.tugraz.at/content/research/krypto/AES/

[39] An Overview of Cryptography, http://www.garykessler.net/library/crypto.html

[40] A. Menezes, P. van Oorschot and S. Vanstone. Handbook of Applied Cryptography, *CRC Press, 1996*

[41] NA, http://oprofile.sourceforge.net/news/ (ημερομηνία προσπέλασης: 12/01/2009)

[42] Gene Amdahl, Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities, AFIPS Conference Proceedings, 30, 1967, σελ. 483-485

[43] Moldovan, D.I., *Parallel Processing. From Applications to Systems*, Morgan Kaufmann Publishers, Inc., 1993.

[44] Schneier, B., Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition, John Wiley & Sons, 1995.

[45] Various http://www.cs.umd.edu/projects/omega/

[46] Dworkin, M., Recommendation for Block Cipher Modes of Operation: Methods and Techniques, *NIST Special Publication 800-38A*,December 2001

[47] Bruce Schneier and Doug Whiting. Fast Software Encryption: Designing Encryption Algorithms for Optimal Software Speed on the Intel Pentium Processor

[48] Zhijie Shi, Ruby B. Lee. Department of Electrical Engineering, Princeton University, "*Bit Permutation Instructions for Accelerating Software Cryptography*" Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors, July 10-12, 2000, Boston, Massachusetts, USA, pp. 138-148

[49] WŁODZIMIERZ BIELECKI, DARIUSZ BURAK, Faculty of Computer Science and Information Systems Szczecin University of Technology *"Parallelization of the AES Algorithm.* Proceedings of the 4th WSEAS Int. Conf. on Information Security, Communications and Computers, Tenerife, Spain, December 16-18, 2005 (pp224-228)