ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Προγραμματιστικό Πλαίσιο Διασύνδεσης Δομικών Στοιχείων Ετερογενούς Επεξεργασίας Πολυμεσικού Περιεχομένου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Διονύσιος Γ. Κακολύρης

Επιβλέπων : Βασίλειος Λούμος
Καθηγητής

Αθήνα, Ιούνιος 2013

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Προγραμματιστικό Πλαίσιο Διασύνδεσης Δομικών Στοιχείων Ετερογενούς Επεξεργασίας Πολυμεσικού Περιεχομένου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Διονύσιος Γ. Κακολύρης

Επιβλέπων : Βασίλειος Λούμος
Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30η Ιουνίου 2013.

.....	.....	.....
Λούμος Βασίλειος Καθηγητής	Καγιάφας Ελευθέριος Καθηγητής	Αναγνωστόπουλος Χρήστος-Νικόλαος Επίκουρος Καθηγητής

Αθήνα, Ιούνιος 2013

.....
Διονύσιος Γ. Κακολύρης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Στον αδερφό μου Νίκο, χωρίς τη βοήθεια του οποίου
δε θα μπορούσα να συνεχίσω τις σπουδές μου

Περίληψη

Η παρούσα διπλωματική εργασία αναλύει ένα προτεινόμενο ετερογενές προγραμματιστικό πλαίσιο για επεξεργασία πολυμεσικού σήματος. Το σύστημα αναπτύχθηκε στα πλαίσια των δραστηριοτήτων του Εργαστηρίου Πολυμέσων του Ε.Μ.Π. κατά το έτος 2011.

Η ολοένα και αυξανόμενη διάδοση των πολυμέσων δημιουργεί περισσότερες απαιτήσεις στην αποδοτική επεξεργασία και μετάδοσή τους. Η Τεχνολογία Πολυμέσων ως επιστήμη συνεχώς παρουσιάζει καινοτομίες που έχουν άμεσο αντίκτυπο στην καθημερινότητά μας. Ο Ετερογενής Παραλληλισμός αποτελεί λύση για αρκετά προβλήματα που εγείρονται λόγω της αυξανόμενης πολυπλοκότητας και μεγέθους του περιεχομένου. Το σύστημα που αναλύεται στην παρούσα εργασία αναλαμβάνει να εκθέσει τον Ετερογενή Παραλληλισμό στο χρήστη παρέχοντας ως βάση μια λύση που αναλαμβάνει τη διευθέτηση των διαφόρων θεμάτων που ο ταυτοχρονισμός συνεπάγεται.

Ξεκινώντας από θεμελιώδεις έννοιες για την Πληροφορία και στοιχεία επεξεργασίας σημάτων, γίνεται μια επισκόπηση του πολυμεσικού σήματος, εστιάζοντας στο σήμα εικόνας. Ενδεικτικά παραθέτονται διάφορες τεχνικές επεξεργασίας εικόνας και τρόποι ψηφιακής αναπαράστασής της.

Στη συνέχεια αναλύεται ο Ετερογενής Παραλληλισμός. Στόχος είναι ο αναγνώστης από τη μία να έχει μια σφαιρική εικόνα των διαφορετικών υπο-συστημάτων που συνεργάζονται για την επίτευξή του, αλλά και από την άλλη, να δει πρακτικά διάφορα μοτίβα, μοντέλα, πρότυπα και πλαίσια με τα οποία μπορεί να εργασθεί ώστε να υλοποιήσει πιο αποδοτικά, σε σχέση με το συμβατικό τρόπο, τις ιδέες των προηγούμενων κεφαλαίων.

Το δεύτερο μέρος της εργασίας αποτελείται από την ανάλυση της υλοποίησης του συστήματος, διάφορες περιπτώσεις χρήσης, αλλά και μελλοντικές βελτιώσεις - κατευθύνσεις που κανείς μπορεί να ακολουθήσει. Όπως γίνεται αντιληπτό, το σύστημα δεν είναι περιορισμένο στην επεξεργασία μόνο πολυμεσικού σήματος, αλλά μπορεί να αποτελέσει και εργαλείο για διάφορες άλλες χρήσεις γενικότερου σκοπού.

Λέξεις κλειδιά: Πολυμέσα, Προγραμματιστικό Πλαίσιο, Ετερογενής Παραλληλισμός, Ταυτοχρονισμός, Αλγοριθμικοί Σκελετοί, Ταυτόχρονος Πίνακας Κατακερματισμού, CG-Stone

Abstract

The present Diploma Thesis analyzes a proposed framework for multimedia processing. The system was developed in the context of the N.T.U.A. Medialab's activities during the year 2011.

The ever-increasing proliferation of multimedia demands more efficient processing and broadcasting. Multimedia Technology as a science discipline continuously presents innovations that affect our every-day lives. Heterogeneous Parallelism constitutes a solution for many problems arising from the increasing complexity and size of multimedia content. The system analyzed in the present thesis undertakes the task of exposing Heterogeneous Parallelism to the user, providing as a foundation, a solution that manages issues emanating from the nature of concurrency.

Starting from basic concepts regarding Information and fragments of signal processing, a review of multimedia signals is done, focussing on image processing. Some techniques regarding image processing and digital image representation are also presented.

Next, we focus on Heterogeneous Parallelism. The target is for the reader to have a spherical view in terms of the various sub-systems that work together for this to happen. On the other hand, the reader should be able to see the applications of various patterns, models, standards and frameworks with the aid of which she can work in order to implement the previous chapters' procedures in a more efficient way.

The second part of this thesis consists of the system's implementation analysis, various use cases and future improvements - directions one can pursue. It becomes evident that the system is not limited to multimedia processing - it can be used for many other general purpose applications.

Keywords: Multimedia, Framework, Heterogeneous Parallelism, Concurrency, Algorithmic Skeletons, Concurrent Hash Table, CG-Stone

Πίνακας Περιεχομένων

1. ΠΛΗΡΟΦΟΡΙΑ ΚΑΙ ΣΤΟΙΧΕΙΑ ΕΠΕΞΕΡΓΑΣΙΑΣ ΣΗΜΑΤΟΣ

1.1 Πληροφορία

Με τον όρο πληροφορία θεωρούμε μια διέγερση που μπορεί να αντιληφθεί ένας παρατηρητής. Αυτή μπορεί να περιγραφεί από ένα σύνολο φυσικών μεταβλητών, η τιμή των οποίων είναι μια συνάρτηση του χρόνου ή / και του χώρου. Κατά τη θεώρηση αυτή δεν εξετάζουμε το σημασιολογικό περιεχόμενο της πληροφορίας (το πώς δηλαδή ο παρατηρητής αντιλαμβάνεται την πληροφορία αυτή καθεαυτή και τι εννοιολογική σημασία της προσδίδει). Για παράδειγμα, αν μια οθόνη υπολογιστή αποδίδει μια στατική εικόνα που απεικονίζει έναν άνθρωπο μπροστά από ένα κτίριο, η πληροφορία είναι οι τιμές των εικονοστοιχείων που αποτελούν την εικόνα και όχι η σημασιολογία που αποδίδει ένας ανθρώπινος παρατηρητής, του οποίου ο εγκέφαλος, επεξεργαζόμενος την πληροφορία, τον οδηγεί στο επαγόμενο συμπέρασμα ότι η εικόνα απεικονίζει έναν άνθρωπο μπροστά από ένα κτίριο. Οι φυσικές ποσότητες που περιγράφουν την πληροφορία αποτελούν λοιπόν ένα σήμα. Από τη θεωρία Σημάτων και Συστημάτων είναι γνωστό ότι το σήμα αντιπροσωπεύει μια διαδικασία που εξελίσσεται στο χρόνο, ενώ το σύστημα μια οντότητα με χωρική υπόσταση. Τα σήματα και το πώς αυτά ανακτούνται, υπόκεινται σε επεξεργασία, μεταδίδονται και αποθηκεύονται είναι τα κύρια θέματα που πραγματεύεται το παρόν κεφάλαιο.

Κύριο διαχωρισμό των σημάτων αποτελούν οι παρακάτω δύο κατηγορίες:

- *Αναλογικό* είναι το σήμα το οποίο είναι συνεχής συνάρτηση του χρόνου ή / και του χώρου.

- *Ψηφιακό* είναι το σήμα το οποίο αποτελείται από μια ακολουθία διακριτών τιμών που είναι κωδικοποιημένες στο δυαδικό σύστημα και εξαρτώνται από το χρόνο ή το χώρο.[16]

1.2 Μέτρο πληροφορίας

Έστω μια πηγή πληροφορίας διακριτού χρόνου με διακριτό αλφάβητο και ένας παρατηρητής. Το πληροφορικό περιεχόμενο της πηγής πρέπει να ικανοποιεί κάποιες διαισθητικές ιδιότητες. Πρώτη διαισθητική ιδιότητα είναι ότι ένα λογικό μέτρο της πληροφορίας μιας εξόδου της πηγής είναι φθίνουσα συνάρτηση της πιθανότητας της εξόδου αυτής. Ένα απλό παράδειγμα αυτού είναι το εξής: αν έχουμε μια λευκή σελίδα και θεωρήσουμε ως έξοδο το περιεχόμενο σε ένα συγκεκριμένο σημείο της (π.χ. ένα μικρό τετράγωνο), περισσότερη πληροφορία πληροφορία μεταφέρει η έξοδος "μαύρη κουκίδα στο σημείο (x, y)" παρά η έξοδος "λευκό στο σημείο (x, y)". Η δεύτερη έξοδος έχει μεγαλύτερη πιθανότητα να εμφανιστεί από την πρώτη. Δεύτερη διαισθητική ιδιότητα είναι ότι μικρή αλλαγή στην πιθανότητα μιας δεδομένης εξόδου δε μεταβάλλει πολύ την πληροφορία που παρέχεται από την έξοδο αυτή. Στο παραπάνω παράδειγμα, μια μικρή αλλαγή στο πάχος της κουκίδας δεν πρέπει να έχει σοβαρές επιπτώσεις στην πληροφορία "μαύρη κουκίδα στο σημείο (x, y)". Επίσης, εφόσον οι συνιστώσες που αποτελούν την πληροφορία είναι ανεξάρτητες μεταξύ τους, θεωρούμε προφανώς, ότι η πληροφορία αποτελείται από την ένωσή τους και διαισθητικά είναι το άθροισμα των πληροφοριών αυτών των συνιστωσών.

Συμπέρασμα των παραπάνω είναι ότι η ποσότητα της πληροφορίας που παρέχει μια έξοδος a_j με πιθανότητα p_j ικανοποιεί τις επόμενες τέσσερις συνθήκες:[12]

1. Το περιεχόμενο της πληροφορίας της εξόδου a_j εξαρτάται μόνο από την πιθανότητα της a_j και όχι από την τιμή της a_j . Η συνάρτηση αυτή $I(p_j)$ ονομάζεται ιδία-πληροφορία (self-information).

2. Η ιδία-πληροφορία είναι μια συνεχής συνάρτηση της p_j , δηλαδή η I είναι μια συνεχής συνάρτηση.

3. Η ιδία-πληροφορία είναι μια φθίνουσα συνάρτηση του ορίσματός

της.

4. Αν $p_j=(p_{j1}, p_{j2})$, τότε $I(p_j)=I(p_{j1})+I(p_{j2})$.

Η μόνη συνάρτηση που ικανοποιεί τα παραπάνω είναι η λογαριθμική συνάρτηση και είναι

$I(x)=-\log_b x$

Αν η βάση b είναι το 2, τότε η πληροφορία μετριέται σε bits. Το πληροφορικό περιεχόμενο της πηγής (εντροπία της πηγής) ορίζεται ως η σταθμισμένη μέση τιμή της ιδίας-πληροφορίας των εξόδων της πηγής. Είναι

$\sum_{i=1}^{N} p_i \cdot I(p_i)=-\sum_{i=1}^{N} p_i \cdot \log(p_i)$, με $0 \cdot log0=0$.

Η εντροπία μιας πηγής πληροφορίας είναι ένα μέτρο της αβεβαιότητας ή ισοδύναμα του πληροφορικού περιεχομένου της πηγής. Κατά μέσο όρο, κάθε έξοδος της πηγής απαιτεί $H(X)$ bits για μια ουσιαστικά, χωρίς σφάλματα αναπαράσταση. Όλα τα παραπάνω ισχύουν για πηγές χωρίς μνήμη, δηλαδή για πηγές στις οποίες η τιμή μιας εξόδου δεν εξαρτάται από τις προηγούμενες εξόδους.

Για πηγή διακριτού χρόνου με συνεχές αλφάβητο (οι έξοδοι είναι πραγματικοί αριθμοί) δεν υπάρχει κάτι που να έχει το διαισθητικό νόημα της εντροπίας. Έτσι, ορίζεται μια άλλη ποσότητα, η διαφορική εντροπία. Η διαφορική εντροπία $h(X)$ μιας συνεχούς τυχαίας μεταβλητής X με f_X είναι

$h(X)=-\int_{-\infty}^{\infty} f_X \cdot \log(f_X) dx$,όπου f_X η συνάρτηση πυκνότητας πιθανότητας (probability density function) της X . Αυτή ορίζεται ως

$f_x=\frac{dF_X(x)}{dx}$,όπου F_X η αθροιστική συνάρτηση κατανομής (cumulative distribution function) που ισούται με

$F_X(x)=P(\omega \in \Omega : X(\omega) \leqslant x)$

Για να ανακτήσουμε αξιόπιστα την έξοδο μιας συνεχούς πηγής, για

κάθε έξοδο της πηγής είναι απαραίτητος ένας άπειρος αριθμός δυαδικών ψηφίων λόγω αναπαράστασης.

1.3 Απόκτηση σήματος και ψηφιοποίηση

Κάποιος αισθητήρας θα πρέπει να είναι σε θέση να μετατρέπει τη φυσική ποσότητα που μετρά σε μια άλλη ποσότητα (π.χ. ηλεκτρική τάση). Ένα αναλογικό σήμα μπορεί να πάρει οποιαδήποτε τιμή στο πεδίο τιμών του, δηλαδή άπειρες τιμές (συνεχές πεδίο ισοδυναμεί με άπειρες δυνατές τιμές). Έτσι, συνήθως η ανάκτηση ενός σήματος από ένα σύστημα περιλαμβάνει κάποιου είδους αναλογική σε ψηφιακή μετατροπή. Κατά τη μετάδοση του σήματος από ένα σύστημα σε ένα άλλο, έχουμε συνήθως ψηφιακή σε ψηφιακή μετατροπή. Τέλος, κατά την αναπαράσταση ενός σήματος ενδέχεται να υπάρχει και ψηφιακή σε αναλογική μετατροπή. Τα παραπάνω αφορούν κυρίως πληροφορίες που επεξεργάζονται υπολογιστικά συστήματα και η αναπαράστασή τους αφορά ανθρώπινους παρατηρητές.

1.4 Δειγματοληψία

Κατά τη δειγματοληψία (sampling), από το άπειρο πλήθος τιμών του συνεχούς σήματος, κρατάμε μόνο ένα σύνολο διακριτών τιμών που συνήθως διαφέρουν κατά κάποιο σταθερό χρονικό διάστημα. Για ακριβή αναπαράσταση του αρχικού σήματος (όσο αυτό είναι δυνατό για αναλογικό σήμα, καθώς για εντελώς πιστή αναπαράσταση, το αρχικό σήμα πρέπει να είναι ζωνοπεριορισμένο), στο στάδιο αυτό πρέπει να λάβουμε υπόψιν το Θεώρημα της Δειγματοληψίας σύμφωνα με το οποίο η συχνότητα δειγματοληψίας πρέπει να είναι τουλάχιστο διπλάσια (δειγματοληψία με ρυθμό Nyquist) από την υψηλότερη συστατική συχνότητα του αρχικού σήματος.

1.5 Θεώρημα Δειγματοληψίας

Έστω $x(t)$ ένα σήμα με περιορισμένο εύρος ζώνης χαμηλών συχνοτήτων W , δηλαδή έστω $X(f)=0$ για $|f|\geqslant W$. Έστω ότι το $x(t)$ δειγματοληπτείται σε πολλαπλάσια ενός βασικού διαστήματος δειγματοληψίας T_s , όπου $T_s\leqslant\frac{1}{2\cdot W}$, και ότι λαμβάνεται η ακολουθία $\{x(n\cdot T_s)\}_{n=-\infty}^{\infty}$. Τότε είναι δυνατό να ανακατασκευασθεί το αρχικό σήμα $x(t)$ από τις τιμές των δειγμάτων του, με τον τρόπο που περιγράφει η εξίσωση[12]

$x(t)=\sum_{n=-\infty}^{\infty}2\cdot W'\cdot T_s\cdot x(n\cdot T_s)\cdot sinc(2\cdot W'\cdot(t-n\cdot T_s))$,όπου W' μια αυθαίρετη παράμετρος που ικανοποιεί τη $W\leq W'\leq\frac{1}{T_s}-W$. Στην ειδική περίπτωση όπου $T_s=\frac{1}{2\cdot W}$, η σχέση ανακατασκευής απλοποιείται στην

$x(t)=\sum_{n=-\infty}^{\infty}x(n\cdot T_s)\cdot sinc(\frac{t}{T_s-n})=\sum_{n=-\infty}^{\infty}x(\frac{n}{2\cdot W})\cdot sinc(2\cdot W\cdot(t-\frac{n}{2\cdot W}))$. Η συχνότητα δειγματοληψίας (sampling rate) προφανώς είναι $f_s=\frac{1}{T_s}$.

Στην πράξη, πριν την εισαγωγή του σήματος στο δειγματολήπτη, χρησιμοποιείται ένα βαθυπερατό φίλτρο έτσι, ώστε να απορρίπτονται συχνότητες μεγαλύτερες από την W . Επίσης, η δειγματοληψία γίνεται σε ρυθμούς μεγαλύτερους από το ρυθμό Nyquist.

1.6 Κβαντισμός

Μια ακριβής περιγραφή μιας αναλογικής πηγής απαιτεί έναν άπειρο

αριθμό δυαδικών ψηφίων ανά έξοδο. Έτσι, η κβάντιση των τιμών της εξόδου είναι κάτι το αναγκαίο. Στόχος είναι η αναπαράσταση των εξόδων της πηγής με χαμηλούς ρυθμούς χωρίς την εισαγωγή μεγάλης παραμόρφωσης. Προφανώς, αν επιλέξουμε διακριτές στάθμες πλάτους με αρκετά μικρό βήμα μεταξύ τους, μπορούμε να κάνουμε το προσεγγιζόμενο σήμα να μην ξεχωρίζει πρακτικά από το αρχικό συνεχές σήμα.

Η διαφορά μεταξύ δύο γειτονικών διακριτών τιμών ονομάζεται κβάντο (quantum). Τα σήματα που εφαρμόζονται σε έναν κβαντιστή (quantizer) ταξινομούνται σε στάθμες πλάτους (τα βήματα) και όλα τα σήματα εισόδου μέσα στο συν ή πλην μισό ενός κβάντου της μεσαίας τιμής μιας στάθμης αντικαθίστανται στην έξοδο από την υπόψη μεσαία τιμή. Το σφάλμα κβαντισμού αποτελείται από τη διαφορά μεταξύ των σημάτων εισόδου και εξόδου του κβαντιστή. Ο κβαντιστής που χρησιμοποιεί ομοιόμορφη απόσταση μεταξύ των επιπέδων κβαντισμού καλείται ομοιόμορφος κβαντιστής. Σε κάποιες εφαρμογές ωστόσο, είναι προτιμητέο να χρησιμοποιηθεί μεταβλητή απόσταση μεταξύ των επιπέδων κβαντισμού, ώστε να έχουμε μεγαλύτερη "ευαισθησία" του κβαντιστή σε ένα φάσμα τιμών που μας ενδιαφέρει περισσότερο, οπότε και χρησιμοποιούμε μη ομοιόμορφο κβαντιστή, του οποίου η χρήση ισοδυναμεί με τη διέλευση του σήματος βασικής ζώνης μέσω ενός συμπιεστή (compressor) και στη συνέχεια την εφαρμογή του συμπιεσμένου σήματος σε ομοιόμορφο κβαντιστή.[6]

1.7 Θεώρημα Κωδικοποίησης Πηγής

Μια πηγή εντροπίας (ή ρυθμού εντροπίας) Η μπορεί να κωδικοποιηθεί με αυθαίρετα μικρή πιθανότητα σφάλματος σε οποιοδήποτε ρυθμό R (bits/έξοδο πηγής) εφόσον $R>H$. Αντίστροφα, αν $R<H$, η πιθανότητα σφάλματος θα παραμείνει μακριά από το μηδέν, ανεξάρτητα από την πολυπλοκότητα του κωδικοποιητή και του αποκωδικοποιητή που χρησιμοποιούνται.[12]

Το θεώρημα κωδικοποίησης πηγής είναι ένα από τα θεμελιώδη θεωρήματα της Θεωρίας Πληροφορίας και πρωτοδιατυπώθηκε από τον Shannon. Δίνει μόνο τις αναγκαίες και ικανές συνθήκες για την ύπαρξη κωδίκων πηγής.

1.8 Παραμόρφωση

Μια πηγή με υψηλό αριθμό διαμεταγωγής δεδομένων και μεγάλη εντροπία κάνει δύσκολο έως ανέφικτο το έργο αποθήκευσης ή αναμετάδοσης της πληροφορίας της, αφού ο ρυθμός δυαδικών ψηφίων ανά σύμβολο ($H(X)$ / σύμβολο) είναι μεγάλος. Άρα, για δεδομένο αριθμό δυαδικών ψηφίων ανά σύμβολο θα θέλαμε να ξέρουμε τον ελάχιστο ρυθμό σφαλμάτων που μπορεί να επιτευχθεί (και πώς) ή αντίστροφα, για καθορισμένο επίπεδο παραμόρφωσης ποιος είναι ο ελάχιστος αριθμός δυαδικών ψηφίων ανά σύμβολο. Σε αυτό μας βοηθά ο ορισμός της παραμόρφωσης. Παραμόρφωση στην αναπαραγωγή μιας πηγής είναι ένα μέτρο της πιστότητας ή εγγύτητας της αναπαραγόμενης προς την αρχική έξοδο της πηγής. Αποτελεί δηλαδή μέτρο της πιστότητας αναπαραγωγής. Ένα μέτρο παραμόρφωσης είναι η απόσταση μεταξύ του x (αρχική έξοδος) και του $\hat{x}$ (αναπαραγωγή του x). Στη διακριτή περίπτωση, ένα τέτοιο μέτρο είναι η απόσταση ή παραμόρφωση Hamming (Hamming distance) και ορίζεται ως:[12]

$$d_H = \begin{cases} 1, & x \neq \hat{x} \\ 0, & \alpha\lambda\lambda\iota\acute{\omega}\varsigma \end{cases}$$

Στη συνεχή περίπτωση χρησιμοποιείται συχνά η παραμόρφωση του τετραγωνικού σφάλματος που ορίζεται ως

$$d(x, \hat{x}) = (x - \hat{x})^2$$

Ως μέτρο παραμόρφωσης μιας ακολουθίας ορίζεται το μέτρο παραμόρφωσης ανά γράμμα, δηλαδή το μέσο όρο των παραμορφώσεων των στοιχείων της ακολουθίας. Έτσι

$$d(x^n, \hat{x}^n) = \frac{1}{n} \cdot \sum_{i=1}^{n} d(x_i, \hat{x}_i)$$

Αυτό σημαίνει ότι η θέση του σφάλματος στην αναπαραγωγή δεν είναι σημαντική και η παραμόρφωση δεν εξαρτάται από τα "συμφραζόμενα". Επίσης, ως παραμόρφωση για την πηγή ορίζεται η αναμενόμενη τιμή της τυχαίας μεταβλητής $d(\hat{X}, \hat{X}^n)$ (η έξοδος της πηγής είναι μια τυχαία διαδικασία).

$D = E[d(\hat{X}, \hat{X}^n)] = \frac{1}{n} \cdot \sum_{i=1}^{n} E[d(X_i, \hat{X}_i)] = E[d(X, \hat{X})]$,όπου στο τελευταίο βήμα έχουμε χρησιμοποιήσει την παραδοχή της στατικότητας της πηγής (ανεξαρτησία των κατανομών από το δείκτη i).

1.9 Θεώρημα Ρυθμού - Παραμόρφωσης

Ο ελάχιστος αριθμός δυαδικών ψηφίων ανά έξοδο πηγής που απαιτείται για να αναπαραχθεί μια πηγή χωρίς μνήμη με παραμόρφωση μικρότερη ή ίση του D ονομάζεται συνάρτηση ρυθμού - παραμόρφωσης, εκφράζεται με $R(D)$ και ισούται με [12]

$$R(D) = \min_{p(\hat{x}|x): E[d(X, \hat{X})] \leq D} \{I(X; \hat{X})\}$$

Από το παραπάνω θεώρημα έχουμε ότι αν για μια πηγή δίνεται η συνάρτηση ρυθμού - παραμόρφωσης και ένα μέτρο παραμόρφωσης, τότε γνωρίζουμε τον ελάχιστο αριθμό δυαδικών ψηφίων ανά σύμβολο πηγής που απαιτείται για να ανακατασκευάσουμε την πηγή με οποιαδήποτε μέτρο παραμόρφωσης. Αντίστροφα, για κάθε ρυθμό R μπορούμε να προσδιορίσουμε την ελάχιστη επιτεύξιμη παραμόρφωση αν χρησιμοποιηθεί ένας κώδικας με το ρυθμό αυτό. Θα πρέπει να τονισθεί ότι, όπως και στην περίπτωση του Θεωρήματος Κωδικοποίησης της Πηγής, τα αποτελέσματα που υποδεικνύονται από τη συνάρτηση ρυθμού - παραμόρφωσης αποτελούν θεμελιώδη όρια υπό την έννοια ότι μπορούν να επιτευχθούν μόνο

ασυμπτωτικά με αυξανόμενη πολυπλοκότητα των σχημάτων κωδικοποίησης και αποκωδικοποίησης.

1.10 Παλμοκωδική Διαμόρφωση (Pulse-Code Modulation - PCM)

Η μετάδοση σημάτων πληροφορίας, τα οποία είναι από τη φύση τους αναλογικά (όπως τα σήματα φωνής και εικόνας), απαιτεί τα σήματα αυτά να μετατραπούν σε ψηφιακά. Η χρήση της ψηφιακής αναπαράστασης των αναλογικών σημάτων προσφέρει τα παρακάτω πλεονεκτήματα:

- Αντοχή στο θόρυβο μετάδοσης και στην παρεμβολή.
- Αποτελεσματική αναγέννηση του κωδικοποιημένου σήματος κατά μήκος της διαδρομής μετάδοσης.
- Δυνατότητα ομοιόμορφου σχήματος μετάδοσης για διαφορετικά είδη σημάτων βασικής ζώνης.

Στην παλμοκωδική διαμόρφωση χρησιμοποιείται μια περιοδική ακολουθία παλμών ως φέρον και κάποιο χαρακτηριστικό του κάθε παλμού (π.χ. πλάτος, διάρκεια ή θέση) μεταβάλλεται με διακριτό τρόπο σύμφωνα με την τιμή του δείγματος της πληροφορίας. Πρόκειται για μια διακριτού χρόνου και διακριτού πλάτους παράσταση για το σήμα. Οι ουσιώδεις λειτουργίες του πομπού ενός συστήματος PCM είναι η δειγματοληψία, η κβαντιση και η κωδικοποίηση του σήματος που εκπέμπει. Αντίστοιχα, οι ουσιώδεις λειτουργίες του δέκτη είναι η αναγέννηση των εξασθενημένων σημάτων, η αποκωδικοποίηση και η αποδιαμόρφωση της ακολουθίας των κβαντισμένων σημάτων.[16]

2. ΠΟΛΥΜΕΣΑ (MULTIMEDIA)

2.1 Πολυμεσικό σήμα

Το πολυμεσικό σήμα είναι σύνθετο σήμα που αποτελείται από σήμα κειμένου, ήχου, εικόνας, κινούμενης εικόνας, κ.ά. Μπορεί να έχουμε ένα εξ αυτών ή συνδυασμό τους. Η Τεχνολογία Πολυμέσων ασχολείται με τη διαχείριση, επεξεργασία, αποθήκευση, μετάδοση και εκτίμηση τέτοιων σημάτων. Πολλές ιδέες και αρχές όμως απορρέουν από την Επεξεργασία σήματος (και ειδικά την Ψηφιακή Επεξεργασία Σήματος) καθώς και τη Θεωρία Πληροφορίας (όπως και άλλων κλάδων της Επιστήμης γενικότερα). Η Τεχνολογία Πολυμέσων είναι πολυδιάστατη επιστήμη και στη σύγχρονη ψηφιακή εποχή βρίσκει ολοένα και περισσότερες εφαρμογές που έχουν μεγάλες επιπτώσεις στην καθημερινή μας ζωή. Στο παρόν κεφάλαιο θα επικεντρωθούμε στην αναπαράσταση διαφόρων τύπων πολυμεσικού σήματος, στην επεξεργασία σήματος εικόνας, καθώς και στη μετάδοση πολυμεσικού σήματος μέσω δικτύου.

2.1.1 Σήμα κειμένου

Μια απλή μορφή πολυμεσικού σήματος είναι το κείμενο. Γραπτά κείμενα αποτυπώνονται και αποθηκεύονται κατά τη διάρκεια όλης της ανθρώπινης ιστορίας (μιας και με τον όρο προϊστορία εννοούμε το χρονικό διάστημα πριν τη γραφή). Ένα κείμενο αποτελείται από μια ακολουθία συμβόλων. Σύγχρονες συσκευές επεξεργασίας και διαχείρισης κειμένου χρησιμοποιούν προτυποποιημένες κωδικοποιήσεις. Για λατινικούς χαρακτήρες ευρέως διαδεδομένη είναι η κωδικοποίηση ASCII (American Standard Code for Information Interchange) όπου για την αναπαράσταση κάθε χαρακτήρα χρησιμοποιούνται 7 δυαδικά ψηφία, δηλαδή μπορούν να αναπαρασταθούν 128 διαφορετικοί χαρακτήρες. Σε υλοποιήσεις που

συναντούμε σε Η/Υ συνήθως χρησιμοποιούνται 8 δυαδικά ψηφία (1 byte) για την αποθήκευση ενός χαρακτήρα, οπότε έχουμε παράσταση 256 χαρακτήρων. Οι πρώτοι 128 χαρακτήρες είναι δεσμευμένοι για λατινικούς χαρακτήρες, σύμβολα στίξης, διάφορα άλλα σύμβολα καθώς και χαρακτήρες ελέγχου (π.χ. ήχος ειδοποίησης). Οι τελευταίοι 128 αποτελούνται από επιπρόσθετα σύμβολα και χαρακτήρες τοπικού αλφαβήτου για συγκεκριμένη χρήση (οπότε πρακτικά διαφέρουν από χώρα σε χώρα) και δεν αποτελούν μέρος της ASCII κωδικοποίησης. Για να μπορεί όμως να γίνει εφικτή η επικοινωνία σε παγκόσμιο επίπεδο και να αναπτυχθούν πολυγλωσσικές εφαρμογές, έπρεπε να υπάρξει προτυποποίηση που να λαμβάνει υπόψιν της όσο το δυνατόν περισσότερες γλώσσες και αλφάβητα. Έτσι υιοθετήθηκαν πρότυπα όπως κωδικοσελίδες ISO, Unicode, UTF-8, UTF-16, όπου για την αναπαράσταση ενός χαρακτήρα χρησιμοποιούμε παραπάνω του 1 byte. Στον αντίποδα όμως, υπάρχει ενιαίος τρόπος αναπαράστασης χαρακτήρων διαφορετικών αλφαβήτων. Η ASCII κωδικοποίηση παραμένει το πρότυπο για κείμενα τα οποία χρησιμοποιούν μόνο λατινικούς χαρακτήρες, όπως για παράδειγμα ο πηγαίος κώδικας (source code) ενός προγράμματος Η/Υ.

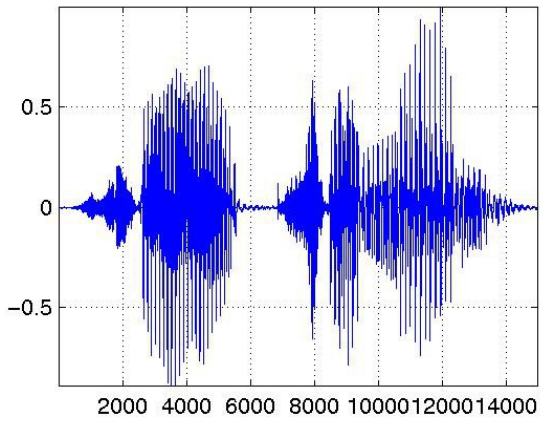
```
The quick brown fox jumps over the lazy dog.
```

Σήμα κειμένου με λατινικούς χαρακτήρες.

2.1.2 Σήμα ήχου

Το ψηφιακό ηχητικό μονοφωνικό σήμα αναπαρίσταται από ένα συρμό στοιχείων τα οποία συνήθως προκύπτουν από δειγματοληψία αναλογικού μονοφωνικού σήματος. Πρόκειται για μονοδιάστατο σήμα (συνάρτηση του χρόνου). Τα στοιχεία που αποτελούν το σήμα είναι τα δείγματα. Συνηθέστερες κωδικοποιήσεις είναι με δείγματα 16 ή 24 δυαδικών ψηφίων, καθώς και κανονικοποιημένων πραγματικών αριθμών με χρήση αναπαράστασης κινητής υποδιαστολής κατά IEEE. Η κωδικοποίηση αυτή συνιστά το βάθος των στοιχείων που αποτελούν το σήμα. Άρα, τον καθορισμό του σήματος από τα δείγματά του, χρειάζεται να ξέρουμε τη

συχνότητα δειγματοληψίας και το βάθος των στοιχείων. Η παραπάνω διαδικασία αποτελεί Παλμοκωδική Διαμόρφωση. Αντίστοιχα, για την αναπαράσταση πολυφωνικού (στερεοφωνικού) ηχητικού σήματος έχουμε υπέρθεση μονοφωνικών συνιστωσών (κανάλια), τα οποία προκύπτουν από τις μονοφωνικές συνιστώσες.



Κυματομορφή ηχητικού σήματος ανθρώπινης ομιλίας.

2.1.3 Σήμα εικόνας

Το ψηφιακό σήμα εικόνας έχει αρκετές ομοιότητες με το αντίστοιχο του ήχου. Η συνήθης αναπαράστασή του αφορά τα δείγματα που αποτελούν τη στατική εικόνα. Τα δείγματα αυτά καλούνται εικονοστοιχεία (picture elements - pixels).

Η πασίγνωστη εικόνα "Lenna" που χρησιμοποιείται ως υπόδειγμα σε πολλές εφαρμογές επεξεργασίας εικόνας.

2.1.3.1 Χρωματικά Μοντέλα Εικόνων

Το είδος αναπαράστασης (χρωματικό μοντέλο) καθορίζει τα δεδομένα που αποθηκεύονται για το κάθε εικονοστοιχείο. Ευρέως χρησιμοποιούμενα χρωματικά μοντέλα για έγχρωμες εικόνες είναι τα RGB, CMYK, HSV, HSL και τα YUV, YCbCr. Εξάλλου, για μονόχρωμες εικόνες υπάρχει το μοντέλο με αναπαράσταση σε τόνους του γκρίζου (grayscale) και με δυαδική αναπαράσταση (bitmap).[15]

Στη δυαδική αναπαράσταση κάθε εικονοστοιχείο αναπαριστάται με ένα δυαδικό ψηφίο (bit), ώστε να γίνεται ο διαχωρισμός ανάμεσα σε έντονο και σκούρο (πληροφορία εικόνας όσον αφορά την ένταση). Είναι η πιο απλή μορφή αναπαράστασης, όπου συνήθως απαντάται η αντιστοίχιση της στάθμης 0 για το μαύρο και της στάθμης 1 για το λευκό. Η ένταση των εικονοστοιχείων δεν υφίσταται μικρές διακυμάνσεις, αφού έχουμε δύο τιμές. Επομένως, η αναπαράσταση αυτή αποτελεί την ιδανική επιλογή για τις περισσότερες τεχνικές επεξεργασίας εικόνας. Σε πραγματικές συνθήκες, η διακύμανση των τιμών των εικονοστοιχείων μίας εικόνας είναι τυχαία και εξαρτάται από την ένταση του φωτός της σκηνής. Όμως, με τις δυαδικές

εικόνες, είναι εφικτή με ενιαίο τρόπο, η εξαγωγή ομοιόμορφων αποτελεσμάτων.

Δυαδική αναπαράσταση εικόνας.

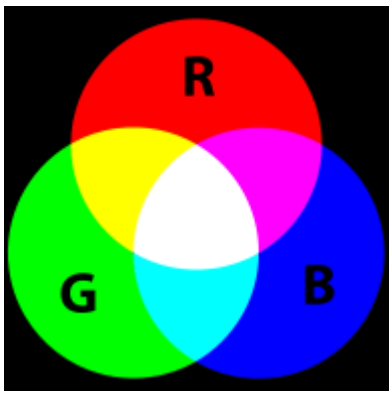
Η αναπαράσταση με τόνους του γκρίζου αφορά στην αναπαράσταση της έντασης κάθε εικονοστοιχείου της εικόνας. Πρόκειται λοιπόν για μονοχρωματικές εικόνες. Οι πιο συνήθεις μορφές ψηφιακής αναπαράστασης για κάθε εικονοστοιχείο είναι με 8, 12 ή 16 δυαδικά ψηφία για ακέραιες τιμές ή αναπαράσταση κινητής υποδιαστολής (κατά IEEE) για πραγματικές τιμές. Στα παραπάνω, η ελάχιστη τιμή (μηδέν) χρησιμοποιείται για το σκούρο (συνήθως μαύρο) και η μέγιστη για το έντονο (συνήθως λευκό). Η επιλογή εξαρτάται από την ευαισθησία του αισθητήριου οργάνου που θα χρησιμοποιηθεί για την περαιτέρω ανάλυση (π.χ. ανθρώπινο μάτι, όπου συνήθως έχουμε αναπαράσταση με 8 δυαδικά ψηφία). Βασικό πλεονέκτημα της αναπαράστασης αυτής αποτελεί η μείωση της πολυπλοκότητας των αλγορίθμων και ο περιορισμός του μεγέθους των προς επεξεργασία εικόνων, μιας και έχουμε απουσία χρωματικής πληροφορίας. Με απλή κβαντοποίηση (χρησιμοποιώντας μια αξία κατωφλίου) μπορούμε εύκολα να περάσουμε στη δυαδική αναπαράσταση με ένα ψηφίο.

Αναπαράσταση εικόνας με τόνους του γκρίζου.

Για αναπαράσταση έγχρωμης εικόνας συνήθως χρησιμοποιούμε υπέρθεση καναλιών, το καθένα από τα οποία αναπαριστάται συνήθως με τόνους του γκρίζου (δηλαδή μονόχρωμη πληροφορία). Η ακρίβεια κάθε καναλιού (αριθμός δυαδικών ψηφίων) προφανώς καθορίζει το χρωματικό βάθος της εικόνας. Συναντώνται αναπαραστάσεις με ακέραια τιμή για κάθε εικονοστοιχείο, όπως και με κινητή υποδιαστολή (που είναι κανονικοποιημένες στην κλίμακα 0,0 έως και 1,0). Σε όσα από τα παρακάτω μοντέλα υπάρχει αναπαράσταση της φωτεινότητας (luminance) έχουμε το πλεονέκτημα της συμβατότητας όσον αφορά την αναπαραγωγή σε εξόδους μονόχρωμου σήματος, αναπαράγοντας μόνο το πρώτο κανάλι (αυτό της φωτεινότητας δηλαδή).

Το RGB μοντέλο (Red, Green, Blue) αναπαριστά κάθε εικονοστοιχείο με 3 μονοχρωματικές συνιστώσες (κανάλια): κόκκινο, μπλε και πράσινο. Αναλόγως την επιθυμητή ακρίβεια, οι αναπαραστάσεις είναι με 16 δυαδικά ψηφία (5 για κόκκινο, 6 για μπλε, 5 για πράσινο, όπου μπορούμε να αναπαραστήσουμε 65.536 διαφορετικά χρώματα) ή με 24 δυαδικά ψηφία (truecolor - 8 για κάθε ένα από τα 3 χρώματα, όπου μπορούμε να αναπαραστήσουμε 16.777.216 διαφορετικά χρώματα). Για κάθε κανάλι η τιμή 0 συνήθως δηλώνει απουσία της αντίστοιχης συνιστώσας, ενώ η μέγιστη τιμή δηλώνει έντονο κορεσμό για αυτήν. Η σύνθεση της χρωματικής απόδοσης του εικονοστοιχείου γίνεται με προσθετικό τρόπο για τα 3

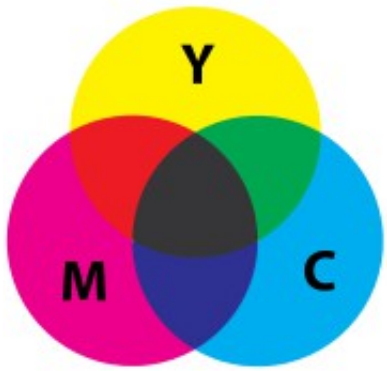
κανάλια. Πλεονέκτημα αυτής της αναπαράστασης αποτελεί το γεγονός ότι προσομοιώνει το μοντέλο της ανθρώπινης όρασης. Επίσης είναι το μοντέλο που απαντάται στην πλειοψηφία των εφαρμογών γραφικής με υπολογιστή (computer graphics), αλλά και των υλοποιήσεων κυκλωμάτων γραφικών φηφιακού υπολογιστή. Επί πλέον είναι χρήσιμο για απόδοση σε μαύρο καμβά (π.χ. εκτύπωση σε μαύρη επιφάνεια ή απόδοση σκηνής σε οθόνη υπολογιστή), λόγω του ότι απουσία των 3 συνιστωσών (μηδέν κόκκινο, μηδέν πράσινο, μηδέν μπλε) είναι το μαύρο χρώμα. Ένας τρόπος για να περάσουμε από το RGB μοντέλο σε μονοχρωματικό με τόνους του γκρίζου είναι μέσω της φωτεινότητας, όπου $V=\frac{R+G+B}{3}$ με την κατάλληλη κανονικοποίηση (βλ. παρακάτω μοντέλο HSV).



Ο χρωματικός χώρος RGB.

Συμπληρωματικό του RGB μοντέλου είναι το CMYK (τετραχρωμία Cyan, Magenta, Yellow, Key (blacK)) όπου η σύνθεση γίνεται με αφαιρετικό τρόπο (σε αντίθεση με τον προσθετικό του RGB). Με άλλα λόγια, κατανοούμε τη σχέση των δύο παραπάνω αναπαραστάσεων (και τη μετάβαση από το RGB στο CMYK) από το προφανές γεγονός ότι απουσία του κόκκινου με μέγιστη τιμή πράσινου και μπλε είναι το κυανό, απουσία του πράσινου με μέγιστη τιμή κόκκινου και μπλε είναι το ματζέντα, απουσία του μπλε με μέγιστη τιμή κόκκινου και πράσινου είναι το κίτρινο και τέλος, απουσία όλων συγχρόνως είναι το μαύρο. Είναι προφανές ότι σε αυτήν την αναπαράσταση έχουμε 4 συνιστώσες. Το μοντέλο αυτό συναντάται κυρίως σε

συστήματα εκτύπωσης, αφού η τιμή μηδέν για όλες τις συνιστώσες ισοδυναμεί με το λευκό χρώμα (η εκτύπωση γίνεται σε λευκή επιφάνεια). Επίσης, για έναν εκτυπωτή που διαθέτει ξεχωριστά μελάνια για κάθε συνιστώσα, η εκτύπωση ασπρόμαυρης εικόνας καθίσταται οικονομική μιας και θα χρησιμοποιηθεί μόνο το μαύρο μελάνι.



Ο χρωματικός χώρος CMYK.

Στο μοντέλο HSV (Hue, Saturation, Value ή HSB: Hue, Saturation, Brightness) έχουμε αναπαράσταση του RGB μοντέλου σε κυλινδρικές συντεταγμένες. Αποτελεί μη γραμμικό μετασχηματισμό του προαναφερθέντος μοντέλου. Η απόχρωση (hue) δηλώνει τον τύπο του χρώματος (π.χ. κόκκινο), ο κορεσμός (saturation) δηλώνει πόσο "μουντό" είναι το χρώμα (δηλαδή πόσο κοντά είναι στο γκρίζο) και η φωτεινότητα (value / brightness) δηλώνει την ποσότητα φωτός στο χρώμα. Ο χρωματικός χώρος HSV είναι κυλινδρικής γεωμετρίας. Η απόχρωση, που είναι η γωνιακή διάσταση, ξεκινά από 0^o (κόκκινο), περνά από τις 120^o (πράσινο) και τις 240^o (μπλε) για να ξαναφτάσει στο κόκκινο (360^o). Στον κατακόρυφο άξονα αναπαριστάται η φωτεινότητα κανονικοποιημένη στο διάστημα $[0, 1]$. Ο κορεσμός είναι στον οριζόντιο άξονα στο διάστημα $[0, 1]$. Η χρωματική αναπαράσταση αυτή χρησιμοποιείται συνήθως σε εφαρμογές όπου ο χρήστης διαδραστικά επιλέγει χρώμα (colour selection - colour picker), λόγω της ομαλής κλίσης (gradient) στα χρώματα. Αυτό γίνεται ως επί το πλείστον σε εφαρμογές προβολής και επεξεργασίας εικόνας και κινούμενης εικόνας (video). Από το χώρο RGB στο χώρο HSV μεταβαίνουμε μέσω των

εξής σχέσεων, για κάθε εικονοστοιχείο του οποίου η κάθε RGB συνιστώσα είναι κανονικοποιημένη στο διάστημα $[0, 1]$:

Για την απόχρωση H έχουμε

$$M = max\{R, G, B\}$$
$$m = min\{R, G, B\}$$
$$C = M - m$$
$$H' = \begin{cases} απροσδιόριστη, & C = 0 \\ \frac{G-B}{C} mod\ 6, & M = R \\ \frac{B-R}{C} + 2, & M = G \\ \frac{R-G}{C} + 4, & M = B \end{cases}$$
$$H = \begin{cases} 0, & C = 0 \\ 60^o \cdot H', & αλλιώς \end{cases}$$

Για τη φωτεινότητα V έχουμε

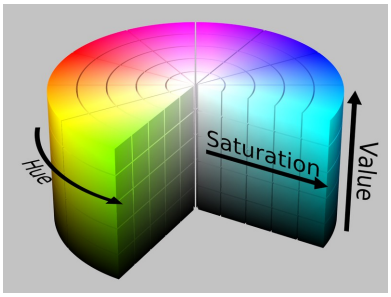
$$V = \frac{R+G+B}{3}$$

Για τον κορεσμό S έχουμε

$$S = \begin{cases} 0, & C = 0 \\ \frac{C}{V}, & αλλιώς \end{cases}$$

Αντίστροφα, για τη μετατροπή από HSV μοντέλο σε RGB:

$$C = V \cdot S$$
$$H' = \frac{H}{60^o}$$
$$X = C \cdot (1 - |(H' \bmod 2) - 1|)$$
$$(R_1, G_1, B_1) = \begin{cases} (0,0,0), & C = 0 \\ (C, X, 0), & 0 \leqslant H' < 1 \\ (X, C, 0), & 1 \leqslant H' < 2 \\ (0, C, X), & 2 \leqslant H' < 3 \\ (0, X, C), & 3 \leqslant H' < 4 \\ (X, 0, C), & 4 \leqslant H' < 5 \\ (C, 0, X), & 5 \leqslant H' < 6 \end{cases}$$
$$m = V - C$$
$$(R, G, B) = (R_1 + m, G_1 + m, B_1 + m)$$



Ο χρωματικός χώρος HSV σε κυλινδρική γεωμετρία.

Αν αντί του κορεσμού S χρησιμοποιήσουμε τη χρωματική συνιστώσα C (chroma), τότε αντί της κυλινδρικής γεωμετρίας, η αναπαράσταση γίνεται με κωνική γεωμετρία. Παρόμοιο μοντέλο με το HSV είναι το HSL, όπου αντί της φωτεινότητας V έχουμε την ένταση L (Luminance / Luminosity) που αντιπροσωπεύει την ικανότητα ανάκλασης του φωτός ενός χρώματος. Ο όρος ένταση χαρακτηρίζει ετερόφωτα αντικείμενα, σε αντίθεση με τη φωτεινότητα που αντιστοιχεί σε αυτόφωτα αντικείμενα. Το μοντέλο HSL μετασχηματίζει μη γραμμικά τον κύβο του RGB σε διπλό κώνο (σε αντίθεση με το HSV). Οι δύο κορυφές του κώνου αντιστοιχούν στο λευκό και το μαύρο χρώμα, η γωνία ως προς τον κατακόρυφο άξονα αντιπροσωπεύει την απόχρωση και η απόσταση από τον άξονα προσδιορίζει την ένταση του χρώματος. Το μοντέλο HSL πλεονεκτεί σε σχέση με τα υπόλοιπα, διότι αναπαριστά τόσο τον κορεσμό όσο και την ένταση. Όμως, ο τρόπος με τον οποίο ορίζει τον κορεσμό έχει επανειλημμένα αποτελέσει

αντικείμενο κριτικής, αφού η υψηλή τιμή του συνεπάγεται το λευκό χρώμα, γι' αυτό και προτιμάται η δι-κωνική αναπαράσταση αντί της κυλινδρικής. Τα μοντέλα HSV και HSL χρησιμοποιούνται ευρέως σε αλγόριθμους όρασης υπολογιστών (computer vision) και ανάλυσης εικόνων (image analysis) για ανίχνευση χαρακτηριστικών (feature detection) και κατάτμηση εικόνας (image segmentation). Αυτό γίνεται γιατί ενώ στο RGB μοντέλο οι χρωματικές συνιστώσες είναι συσχετισμένες (ποσότητα φωτός στο οποίο εκτίθεται το αντικείμενο), στα HSV / HSL δε συμβαίνει κάτι τέτοιο. Οπότε οι περισσότεροι αλγόριθμοι που εφαρμόζονται σε 1 μονοχρωματική συνιστώσα μπορούν να εφαρμοστούν για τη φωτεινότητα ή ένταση ή παράλληλα να εφαρμοστούν για κάθε μια από τις H, S, V ή L . Αντίστοιχα με το HSV μοντέλο, κατά τη μετάβαση από το RGB μοντέλο έχουμε

$$M = max\{R, G, B\}$$
$$m = min\{R, G, B\}$$
$$C = M - m$$
$$H' = \begin{cases} απροσδιόριστη, & C = 0 \\ \frac{G-B}{C} mod\, 6, & M = R \\ \frac{B-R}{C} + 2, & M = G \\ \frac{R-G}{C} + 4, & M = B \end{cases}$$
$$H = \begin{cases} 0, & C = 0 \\ 60^o \cdot H', & αλλιώς \end{cases}$$
$$L = \frac{M+m}{2}$$
$$S = \begin{cases} 0, & C = 0 \\ \frac{C}{1-|2 \cdot L - 1|}, & αλλιώς \end{cases}$$

Αντίστροφα, για τη μετάβαση στο μοντέλο RGB, πάλι παρόμοια με το HSV:

$$C=(1-|2\cdot L-1|)\cdot S$$

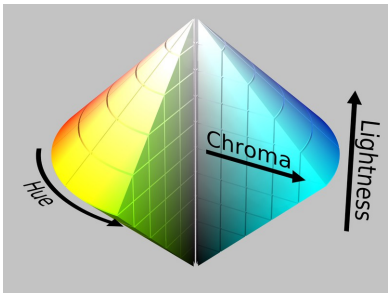
$$H'=\frac{H}{60^o}$$

$$X=C\cdot(1-|(H'\ mod\ 2)-1|)$$

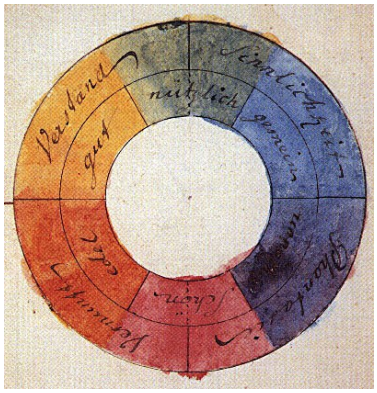
$$(R_1,G_1,B_1)=\begin{cases}(0,0,0), & C=0\\(C,X,0), & 0\leqslant H'<1\\(X,C,0), & 1\leqslant H'<2\\(0,C,X), & 2\leqslant H'<3\\(0,X,C), & 3\leqslant H'<4\\(X,0,C), & 4\leqslant H'<5\\(C,0,X), & 5\leqslant H'<6\end{cases}$$

$$m=L-\frac{C}{2}$$

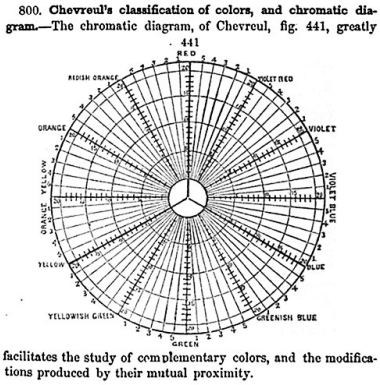
$$(R,G,B)=(R_1+m,G_1+m,B_1+m)$$



Ο χρωματικός χώρος HSL (κωνική γεωμετρία).



Ο "Τροχός Χρωμάτων" του Goethe από τη "Θεωρία Χρωμάτων" (1810).



Το "Χρωματικό Διάγραμμα" του Chevreul (1855).

2.1.3.2 Στοιχεία ψηφιακής επεξεργασίας εικόνας

Η επεξεργασία εικόνας είναι ένας βασικός τομέας της Τεχνολογίας Πολυμέσων. Οι τεχνικές που χρησιμοποιούνται μπορούν να εφαρμοστούν και σε άλλους τύπους πολυμεσικού σήματος. Αντίστροφα, ισοδύναμος μετασχηματισμός στην αναπαράσταση άλλου τύπου πολυμεσικού σήματος, ώστε αυτός να είναι ισοδύναμος με σήμα εικόνας οδηγεί στην άμεση χρήση τεχνικών της επεξεργασίας σήματος εικόνας. Για παράδειγμα, η αναπαράσταση ενός γεωγραφικού χώρου με πολύγωνα στις τρεις διαστάσεις

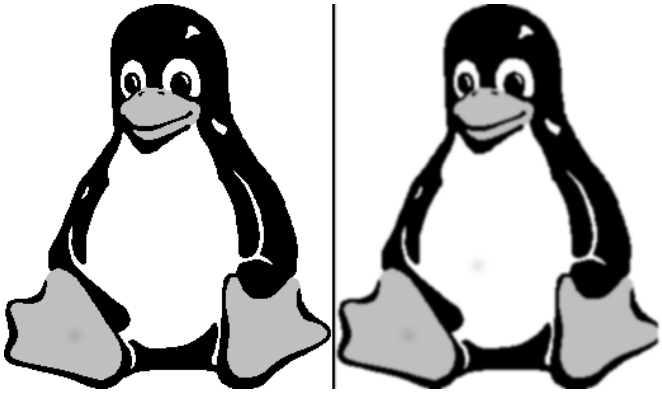
μπορεί να μετασχηματισθεί σε έναν υψομετρικό χάρτη (height map) σε δύο διαστάσεις, μια εικόνα δηλαδή της οποίας τα εικονοστοιχεία αντιστοιχούν στο υψόμετρο του εκάστοτε μέρους του χώρου. Αυτή η εικόνα δύναται να υποστεί άμεσα επεξεργασία με τεχνικές επεξεργασίας εικόνων, τα αποτελέσματα των οποίων αντανακλώνται στην αρχική τοπολογία των πολυγώνων στις τρεις διαστάσεις.

2.1.3.2.1 Εξομάλυνση εικόνας

Η εξομάλυνση της εικόνας (image smoothing) απομακρύνει ανεπιθύμητο θόρυβο (image noise) από αυτήν, ενώ συγχρόνως προστατεύει όλες τις λεπτομέρειες που θέλει ο παρατηρητής να δει στην αυθεντική εικόνα. Συνήθως ο θόρυβος παρουσιάζεται με την μορφή ταλαντώσεων υψηλών συχνοτήτων. Σ' αυτή την περίπτωση, είναι απαραίτητη η εφαρμογή ενός κατωδιαβατού φίλτρου, ώστε να απομακρυνθούν οι υψηλές συχνότητες της εικόνας. Συγκεκριμένα, με τη χρήση του φίλτρου, η εικόνα συνελίσσεται με ένα πυρήνα διαστάσεων συνήθως 3×3 ή 5×5 .[15]

Ο πυρήνας κυλίεται στην εικόνα από πάνω προς τα κάτω και από αριστερά προς τα δεξιά και επεξεργάζεται ένα-ένα τα εικονοστοιχεία της. Για πυρήνα A με στοιχεία a_{ij} διαστάσεων 3×3 $(1\leqslant i,j\leqslant3)$, το εικονοστοιχείο I_{ij} στη θέση i,j θα ισούται με

$$\begin{aligned} A * I_{ij} = {} & a_{11}\cdot I_{i-1,j-1} + a_{12}\cdot I_{i-1,j} + a_{13}\cdot I_{i-1,j+1} \\ & + a_{21}\cdot I_{i,j-1} + a_{22}\cdot I_{i,j} + a_{23}\cdot I_{i,j+1} \\ & + a_{31}\cdot I_{i+1,j-1} + a_{32}\cdot I_{i+1,j} + a_{33}\cdot I_{i+1,j+1} \end{aligned}$$

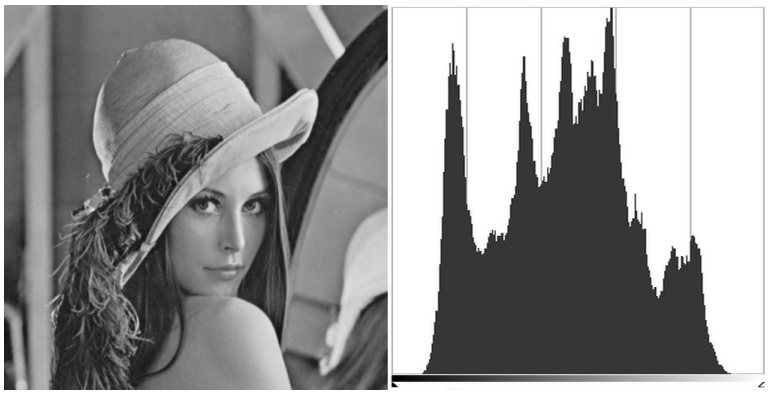


Παράδειγμα εφαρμογής φίλτρου εξομάλυνσης εικόνας.

Διάφορα φίλτρα εξομάλυνσης εικόνας είναι το φίλτρο μέσου όρου, το φίλτρο Gauss, το φίλτρο median, το φίλτρο k-nearest neighbour.

2.1.3.2.2 Ισοστάθμιση Ιστογράμματος

Το ιστόγραμμα μιας εικόνας απεικονίζει την κατανομή των εντάσεων του γκρίζου των εικονοστοιχείων της. Έτσι, πληροφορείται κανείς για το επίπεδο φωτεινότητας της εικόνας και επιπλέον για το εύρος των τιμών των εικονοστοιχείων της. Με την εξισορρόπηση του ιστογράμματος επιτυγχάνεται ένα ικανοποιητικό επίπεδο φωτεινότητας και αντίθεσης της εικόνας, ώστε να μεγιστοποιηθεί η πιθανότητα επιτυχίας του συστήματος.[15]



Εικόνα με το ιστόγραμμά της.

2.1.3.2.3 Γεωμετρικοί μετασχηματισμοί εικόνας

Οι γεωμετρικοί μετασχηματισμοί εφαρμόζονται σε ένα-προς-ένα τα εικονοστοιχεία μιας εικόνας και τα μεταφέρουν από ένα σημείο στην εικόνα σε κάποιο άλλο. Πρόκειται για μετασχηματισμούς σε δύο διαστάσεις. Παρακάτω θα αναφερθούμε στους γεωμετρικούς μετασχηματισμούς που μπορούν να εφαρμοστούν σε μία εικόνα.[15]

Ο απλούστερος μετασχηματισμός προκύπτει με τη χρήση μεταφοράς (translation) και περιστροφής (rotation) μιας εικόνας. Εάν $I(x,y)$ είναι η αρχική εικόνα, η μετασχηματισμένη $I'(x',y')$ θα ακολουθεί την εξής σχέση

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\varphi & \sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} ,$$

όπου φ η γωνία περιστροφής και t_x, t_y η μεταφορά στον x και y άξονα αντίστοιχα.

Ο μετασχηματισμός ομοιότητας προκύπτει από το μετασχηματισμό μεταφοράς και περιστροφής και έχει επιπλέον τη δυνατότητα αλλαγής της κλίμακας (scaling) μιας εικόνας. Είναι

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = k \cdot \begin{bmatrix} cos\varphi & -sin\varphi \\ sin\varphi & cos\varphi \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} ,$$

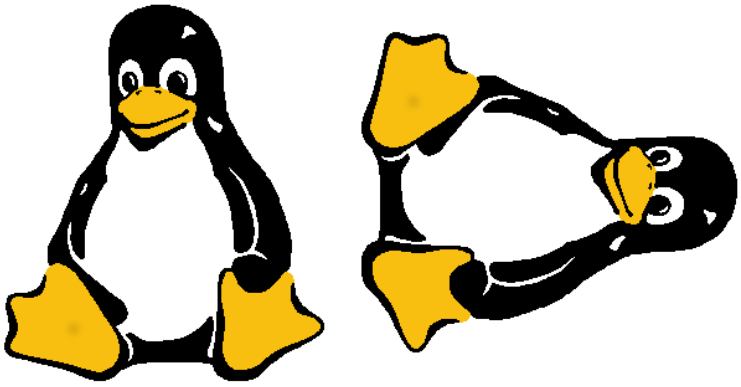
όπου k ο παράγων αλλαγής της κλίμακας.

Τέλος, ο γενικευμένος γραμμικός μετασχηματισμός (affine transform) έχει όλες τις δυνατότητες του μετασχηματισμού ομοιότητας ενώ εισάγει και την έννοια της στρέβλωσης (skewing) μιας εικόνας. Είναι

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Η τεχνική της παρεμβολής (interpolation) επιχειρεί να εισάγει ή να συμπτύξει εικονοστοιχεία μίας εικόνας ώστε τη μεγεθύνει ή να τη σμικρύνει,

χρησιμοποιώντας πληροφορία από τα γειτονικά εικονοστοιχεία του παρεμβαλλόμενου εικονοστοιχείου.



Περιστροφή εικόνας κατά 90 μοίρες με τη φορά των δεικτών του ρολογιού.

2.1.3.2.4 Δυαδικοποίηση εικόνας

Η δυαδικοποίηση είναι η διαδικασία κατά τη οποία μία εικόνα διαφορετικής μορφής μετατρέπεται σε δυαδική. Για το σκοπό αυτό, χρησιμοποιείται ένα κατώφλι βάσει του οποίου διαχωρίζονται οι «λευκές» από τις "μαύρες" τιμές. Μια κατηγοριοποίηση των τεχνικών αυτών είναι η ακόλουθη[15]

- Παγκόσμιες μέθοδοι (global thresholding), στις οποίες χρησιμοποιείται ένα κατώφλι για όλη την εικόνα, επιλεγμένο με στατιστικές μεθόδους, όπως η μέση τιμή των εικονοστοιχείων της εικόνας

- Επιλογή βάσει επεξεργασίας ιστογράμματος, δηλαδή ανάλυσης των κορυφών, των κοιλάδων και της καμπυλότητάς του

- Μέθοδοι που βασίζονται στη συσταδοποίηση (clustering), με τις οποίες τα εικονοστοιχεία ομαδοποιούνται, αρχικά, σε δύο κατηγορίες: του προσκηνίου και του παρασκηνίου

- Χρήση της εντροπίας των περιοχών των εικόνων

- Μέθοδοι που χρησιμοποιούν χαρακτηριστικά των αντικειμένων της εικόνας, όπως οι περιγραφείς που αναφέρθηκαν στην προηγούμενη ενότητα

- Τοπικές μέθοδοι, όπου το κατώφλι για κάθε εικονοστοιχείο προσαρμόζεται ανάλογα με τις εντάσεις των γειτονικών του.

2.1.3.2.5 Μαθηματική μορφολογία

Οι μορφολογικές λειτουργίες είναι μέθοδοι επεξεργασίας δυαδικών εικόνων με βάση τη γεωμετρία. Δέχονται δυαδικές εικόνες ως στοιχεία εισόδου, τις μετασχηματίζουν και τις εξάγουν, επίσης, σε μορφή δυαδικών εικόνων. Αφού επιλεγεί μια κατάλληλη περιοχή για "γειτονιά" (pixel neighbourhood), κατασκευάζεται ένα μορφολογικός πυρήνας (kernel) για να διαμορφωθεί η ευαισθησία του πυρήνα σε συγκεκριμένες μορφές της εικόνας. Η μαθηματική μορφολογία έχει τις ρίζες της στα μαθηματικά και αφορά στην ανάλυση της μορφής και της δομής του αντικειμένου. Χρησιμοποιεί τις λογικές πράξεις όπως $'H, KAI, OXI$. Η διαδικασία συνίσταται στην κύλιση του πυρήνα στην εικόνα, ώστε να επεξεργαστεί τις περιοχές της εικόνας. Οι λειτουργίες που χαρακτηρίζουν τη διαδικασία εφαρμογής των μορφολογικών τελεστών είναι οι κάτωθι[15]

- <u>*Ανήκει*</u>. Συμβολίζεται με το γράμμα ε και έχει ως αποτέλεσμα μια τιμή αληθείας (ΝΑΙ ή ΟΧΙ). Για παράδειγμα, έστω ότι σε μια εικόνα N εικονοστοιχείων, υπάρχει ένα αντικείμενο προς κατάτμηση σε μία περιοχή που απαρτίζεται από A εικονοστοιχεία. Τότε, αν ένα εικονοστοιχείο p ανήκει στο σύνολο A , τότε θα ανήκει και στο N .

- <u>*Συμπλήρωμα*</u>. Έστω ένα υποσύνολο εικονοστοιχείων A μιας εικόνας. Τότε το συμπλήρωμά του $\bar{A}$, θα αποτελείται από τα υπόλοιπα εικονοστοιχεία της εικόνας.

- <u>*Μετάθεση*</u>. Η μετάθεση μιας ομάδας εικονοστοιχείων A είναι ένα σύνολο όπου η θέση κάθε εικονοστοιχείου προκύπτει από την αντανάκλασή του σε σχέση με κάποιο όρισμα. Η μετάθεση του A συμβολίζεται ως A^t .

- *Ένωση*. Η ένωση δύο συνόλων A και B είναι ένα νέο σύνολο που συμβολίζεται με $A \cup B$ (αντιστοιχία με τη λογική πράξη $'H$). Το νέο σύνολο αποτελείται από τα εικονοστοιχεία και των δύο υποσυνόλων.

- *Τομή*. Η τομή δύο συνόλων A και B είναι ένα νέο σύνολο που συμβολίζεται με $A \cap B$ (αντιστοιχία με τη λογική πράξη KAI). Το νέο σύνολο αποτελείται από τα κοινά εικονοστοιχεία των δύο συνόλων.

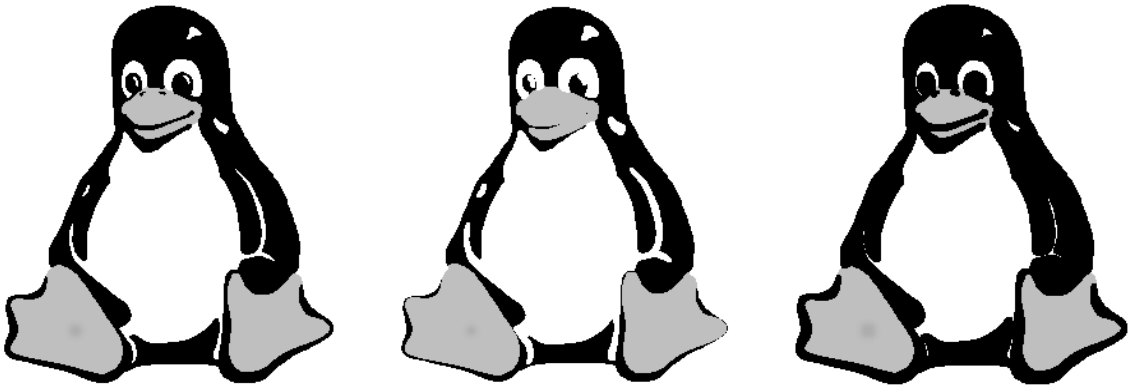
- *Διαφορά*. Η διαφορά δύο συνόλων A και B είναι ένα νέο σύνολο που συμβολίζεται με $A \setminus B$. Είναι το σύνολο στο οποίο κάθε εικονοστοιχείο είναι είτε στοιχείο του A είτε του B , αλλά δεν είναι κοινό τους.

- *Μετακίνηση*. Η μετακίνηση ενός συνόλου A κατά ένα διάνυσμα $\vec{p}$ είναι ένα νέο σύνολο που συμβολίζεται με A_p του οποίου τα στοιχεία προκύπτουν από την πρόσθεση του $\vec{p}$ στα αντίστοιχα στοιχεία του A .

Οι κύριες μορφολογικές λειτουργίες είναι η διαστολή (dilation) και η διάβρωση (erosion). Σχετίζονται μεταξύ τους, καθώς αποτελούν δυαδικά αντίθετες λογικές πράξεις. Η διαστολή επεκτείνει τις λευκές περιοχές ενώ η διάβρωση τις σμικρύνει. Κάθε τελεστής κυλίεται σε όλον το χώρο της εικόνας και επεξεργάζεται μια γειτονιά εικονοστοιχείων σύμφωνα με ένα πυρήνα. Ο πυρήνας είναι μια δυαδική μάσκα η οποία καθορίζει ποια γειτονικά εικονοστοιχεία συμμετέχουν στην επεξεργασία της περιοχής. Ο πυρήνας μπορεί να έχει διάφορα σχήματα και μεγέθη, ανάλογα με τις απαιτήσεις της λειτουργίας. Το κεντρικό εικονοστοιχείο του πυρήνα συνήθως καθορίζει και το ποιο εικονοστοιχείο της εικόνας εισόδου επεξεργάζεται κάθε στιγμή.

Με τη διαστολή συντελείται η λογική πράξη $'H$ μεταξύ των λευκών εικονοστοιχείων της μάσκας και των λευκών εικονοστοιχείων της περιοχής υπό επεξεργασία. Αν οποιαδήποτε πράξη $'H$ μεταξύ των εικονοστοιχείων της μάσκας και εκείνων της εικόνας εξάγει την τιμή 1 , τότε το εικονοστοιχείο της εικόνας εξόδου, που αντιστοιχεί στο κεντρικό στοιχείο του πυρήνα, θα σημειωθεί ως λευκό. Για μια δυαδική εικόνα A θα είναι $A \ (+) \ B = \cup_{b \in B} A_b$, όπου $B = \{b_1, b_2, \ldots, b_n\}$ ο πυρήνας.

Αντίθετα, η διάβρωση επεξεργάζεται την εικόνα χρησιμοποιώντας τη λογική πράξη *ΚΑΙ* . Επομένως, για να σημειωθεί λευκό εικονοστοιχείο στην εικόνα εξόδου, θα πρέπει όλα τα εικονοστοιχεία της εικόνας εισόδου, που αντιπαραβάλλονται με τα λευκά του πυρήνα, να έχουν λευκό χρώμα. Αλλιώς, το εικονοστοιχείο υπό επεξεργασία στην εικόνα εξόδου λαμβάνει την τιμή 0 και σημειώνεται ως μαύρο. Για μια δυαδική εικόνα A θα είναι $A \ominus B = \cap_{b \in B} A_{-b}$, όπου $B = \{b_1, b_2, \ldots, b_n\}$ ο πυρήνας.



Αρχική εικόνα, εικόνα με διάβρωση, εικόνα με διαστολή.

Οι υπόλοιποι μορφολογικοί τελεστές προκύπτουν από το συνδυασμό των δύο βασικών, δηλαδή της διάβρωση και της διαστολής. Ενδεικτικά αναφέρονται το κλείσιμο (closure) της εικόνας (που περιλαμβάνει μια διαδικασία διαστολής ακολουθούμενη από μια λειτουργία διάβρωσης με τον ίδιο πυρήνα) και το άνοιγμα (opening) της εικόνας (όπου η διαδικασία διάβρωσης προηγείται της λειτουργίας διαστολής με τον ίδιο πυρήνα).

2.1.3.2.6 Άλγεβρα εικόνων

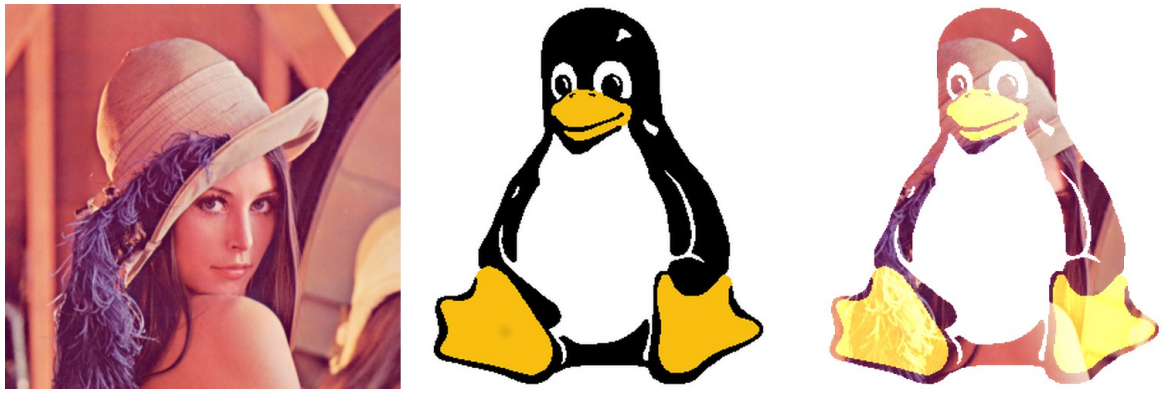
Υπάρχουν δύο κύριοι τύποι αλγεβρικών λειτουργιών που εφαρμόζονται στις εικόνες. Οι αριθμητικές και οι λογικές. Στις πρώτες κατηγοριοποιούνται οι πράξεις της πρόσθεσης, της αφαίρεσης, του

πολλαπλασιασμού και της διαίρεσης. Στις δεύτερες υπολογίζονται οι λογικές πράξεις *'Η, ΚΑΙ, ΟΧΙ* . Όλοι οι τελεστές των παραπάνω πράξεων είναι δυαδικοί (binary operators), οπότε εφαρμόζονται μεταξύ δύο ή περισσότερων εικόνων, εκτός από τον τελεστή *ΟΧΙ* που είναι εναδικός (unary operator).

2.1.3.2.7 Αριθμητικές πράξεις

Οι αριθμητικές πράξεις ανάμεσα σε δύο εικόνες εφαρμόζονται απευθείας στις αντίστοιχες τιμές των εικονοστοιχείων τους. Για την άθροιση ή την αφαίρεση δύο ή περισσότερων εικόνων, πρέπει αυτές να έχουν τις ίδιες διαστάσεις. Η άθροιση χρησιμοποιείται για το συνδυασμό των πληροφοριών δύο ή περισσότερων εικόνων. Οι εφαρμογές της λειτουργίας αυτής περιλαμβάνουν αλγόριθμους αποκατάστασης για τη μοντελοποίηση αθροιστικού θορύβου και την εισαγωγή ειδικών εφέ. Η αφαίρεση χρησιμοποιείται ευρύτατα για τον εντοπισμό κίνησης. Με την αφαίρεση, για παράδειγμα, δύο συνεχόμενων σκηνών σε μια εικονοσειρά, αποκαλύπτονται τα μέρη της εικόνας όπου υπάρχουν διαφορές. Στα σημεία που δε συντελείται αλλαγή μεταξύ των δύο σκηνών, οι τιμές των εικονοστοιχείων των αντίστοιχων καρέ δεν αλλάζουν, οπότε η διαφορά τους θα είναι μηδέν ή τουλάχιστον κοντά στο μηδέν.

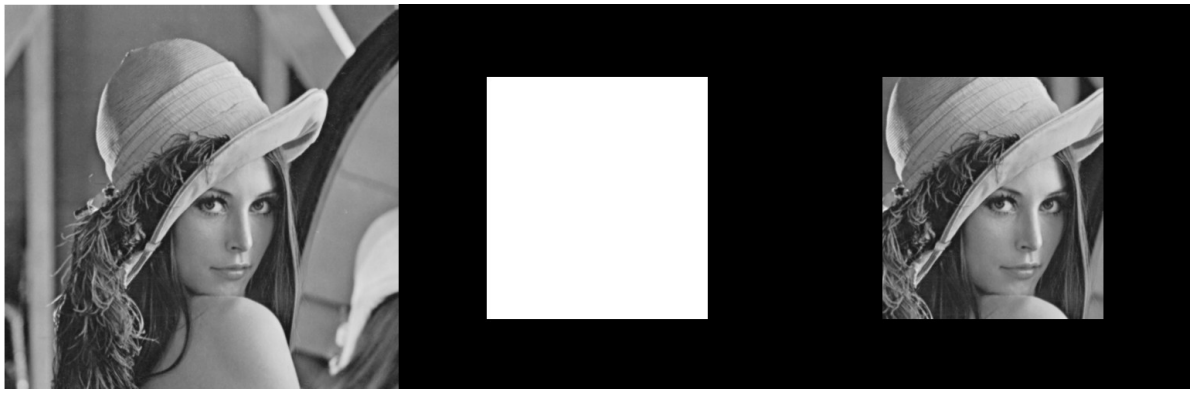
Ο πολλαπλασιασμός και η διαίρεση χρησιμοποιούνται για την προσαρμογή της φωτεινότητας στην εικόνα. Σε αυτές τις πράξεις χρειάζεται μόνο μια εικόνα και ο τελεστής που συμμετέχει στην πράξη. Ο πολλαπλασιασμός της εικόνας με μια τιμή μεγαλύτερη του 1 προκαλεί αύξηση της φωτεινότητας στην εικόνα, ενώ ο πολλαπλασιασμός της με στοιχείο μικρότερο του 1 , μειώνει τη φωτεινότητα (αποτελεί στην ουσία την πράξη της διαίρεσης).[15]



Παράδειγμα πρόσθεσης εικόνων.

2.1.3.2.8 Λογικές πράξεις

Οι λογικές πράξεις *ΚΑΙ* και *Ή* χρησιμοποιούνται προκειμένου να συνδυαστούν οι πληροφορίες δύο εικόνων. Βρίσκουν εφαρμογή στο συνδυασμό δυαδικών μασκών και εικόνων υπό κατάτμηση, ώστε να εντοπιστούν οι περιοχές ενδιαφέροντος.



Παράδειγμα λογικής πράξης ΚΑΙ.

2.1.4 Κινούμενη εικόνα

Στην απλούστερη θεώρησή του, το σήμα κινούμενης εικόνας (video) αποτελείται από μια ακολουθία εικόνων. Είναι δηλαδή συνάρτηση τριών μεταβλητών (2 μεταβλητών για το χώρο και 1 για το χρόνο). Η βασική αρχή

είναι στην κωδικοποίηση της αλληλουχίας των εικόνων που αποτελούν το σήμα. Άρα, για την πιστή αναπαραγωγή πρέπει να είναι γνωστή η συχνότητα δειγματοληψίας (εικόνες ή καρέ ανά δευτερόλεπτο - frames per second) και το βάθος αναπαράστασης των εικονοστοιχείων. Έτσι, μια απλουστευμένη θεώρηση έχει ως υλοποίηση ένα διατεταγμένο σύνολο ομάδων εικονοστοιχείων τα οποία αναπαριστώνται με έναν από τους προαναφερθέντες τρόπους (π.χ. χρωματικός χώρος RGB). Στην πράξη όμως, επειδή μια τέτοια υλοποίηση απαιτεί αρκετό χώρο για αποθήκευση και κανάλι μεγάλου εύρους για μετάδοση, είναι αναγκαία η αναπαράσταση των εικόνων που αποτελούν το σήμα με τρόπους που λαμβάνουν υπόψιν διάφορες παραμέτρους της ανθρώπινης όρασης (π.χ. ευαισθησία ανθρώπινου ματιού), καθώς και συμπίεσης του σήματος, ευκολία αναπαραγωγής, γρήγορη μετατροπή σε άλλους χρωματικούς χώρους (ώστε να μπορούν να οδηγηθούν διαφορετικές συσκευές αναπαραγωγής) και συμβατότητα με παλαιότερες συσκευές αναπαραγωγής (για παράδειγμα ασπρόμαυρες τηλεοράσεις). Ένα ευρέως διαδεδομένο μοντέλο αναπαράστασης είναι το μοντέλο YUV.

2.1.4.1 Το μοντέλο YUV

Το YUV μοντέλο χρησιμοποιείται κυρίως σε εφαρμογές αναπαραγωγής κινούμενης εικόνας. Για ψηφιακή αναπαράσταση χρησιμοποιείται το μοντέλο YCbCr. Η Y συνιστώσα χαρακτηρίζει τη φωτεινότητα του χρώματος, ενώ οι C_b και C_r τη χρωματικότητα του μπλε και κόκκινου αντίστοιχα. Πλεονεκτεί ως προς την καλύτερη συμπίεση της εικόνας, διότι λαμβάνει υπόψιν την αυξημένη ευαισθησία του ανθρώπινου ματιού στη φωτεινότητα, σε σχέση με το χρώμα. Επομένως, απαιτείται κωδικοποίηση μικρότερης ανάλυσης για τα C_b και C_r , με απώτερη συνέπεια, τη σημαντική μείωση του όγκου των δεδομένων προς αποθήκευση ή διακίνηση. Επί πλέον, μιας και οι 3 συνιστώσες του δεν είναι πεπλεγμένες, είναι ιδανικό για αναπαραγωγή σε παλαιότερες συσκευές. Πρακτικά, σε μονοχρωματικές συσκευές αναπαραγωγής (όπως ασπρόμαυρες τηλεοράσεις),

αρκεί μόνο η Y συνιστώσα (φωτεινότητα). Αντίθετα σε ένα μοντέλο όπως το RGB όπου οι συνιστώσες είναι πεπλεγμένες, κάτι τέτοιο (λαμβάνοντας δηλαδή υπόψη μία μόνο χρωματική συνιστώσα) δεν οδηγεί σε πιστή αναπαράσταση του σήματος. Στην πράξη και, λαμβάνοντας υπόψιν την ευαισθησία του ανθρώπινου ματιού, οι τρεις συνιστώσες αποθηκεύονται σε τόνους του γκρι. Κατά την πλειονότητά τους, τα συστήματα κωδικοποίησης και αναπαραγωγής κινούμενης εικόνας ακολουθούν τη διαδικασία μετασχηματισμού RGB κωδικοποίησης σε αντίστοιχη YCbCr (για κωδικοποίηση, αποθήκευση και μετάδοση) και την αντίστροφη διαδικασία (μετασχηματισμός YCbCr σε RGB) για αναπαραγωγή. Αυτό ισχύει για σύγχρονες συσκευές αναπαραγωγής (π.χ. οθόνες υπολογιστών, σύγχρονες τηλεοράσεις, φορητές συσκευές) σε αντίθεση με συσκευές παλαιότερης τεχνολογίας (όπως για παράδειγμα αναλογικές τηλεοράσεις). Οι μετασχηματισμοί από και προς το μοντέλο RGB ορίζονται από διάφορα πρότυπα της Διεθνούς Ένωσης Τηλεπικοινωνιών (ITU, τμήμα ITU-R). Για παράδειγμα, σύμφωνα με το πρότυπο ITU-R BT.601, ο μετασχηματισμός για ψηφιακό σήμα κινούμενης εικόνας από ψηφιακό σήμα RGB (συνιστώσες R'_D, G'_D, B'_D) είναι:

$$Y' = 16 + \frac{65.738 \cdot R'_D}{256} + \frac{129.057 \cdot G'_D}{256} + \frac{25.064 \cdot B'_D}{256}$$
$$C_b = 128 - \frac{37.945 \cdot R'_D}{256} - \frac{74.494 \cdot G'_D}{256} + \frac{112.439 \cdot B'_D}{256}$$
$$C_r = 128 + \frac{112.439 \cdot R'_D}{256} - \frac{94.154 \cdot G'_D}{256} - \frac{18.285 \cdot B'_D}{256}$$

Ο αντίστροφος είναι:

$$R'_D = \frac{298.082 \cdot Y'}{256} + \frac{408.583 \cdot C_r}{256} - 222.921$$
$$G'_D = \frac{298.082 \cdot Y'}{256} - \frac{100.291 \cdot C_b}{256} - \frac{208.120 \cdot C_r}{256} + 135.576$$
$$B'_D = \frac{298.082 \cdot Y'}{256} + \frac{516.412 \cdot C_b}{256} - 276.836$$

Στις παραπάνω εξισώσεις οι όροι Y', R'_D, G'_D, B'_D εμπεριέχουν γ διόρθωση φωτεινότητας (gamma correction), η οποία χρησιμοποιείται για καλύτερη χρήση του εύρους τιμών κάθε μεγέθους σε συνάρτηση με την ανθρώπινη όραση όσον αφορά την αντίληψη της φωτεινότητας. Πρόκειται για μη-γραμμικό μετασχηματισμό. Για ένα μέγεθος X , η γ διόρθωση στην πιο απλή της μορφή ακολουθεί το νόμο $X_o = A \cdot X_i^{\gamma}$, όπου X_o η τιμή εξόδου, X_i η τιμή εισόδου (X_o , X_i μη-αρνητικές τιμές), A μια σταθερά και γ η διόρθωση.

3. ΕΤΕΡΟΓΕΝΗΣ ΠΑΡΑΛΛΗΛΙΣΜΟΣ (HETEROGENEOUS PARALLELISM)

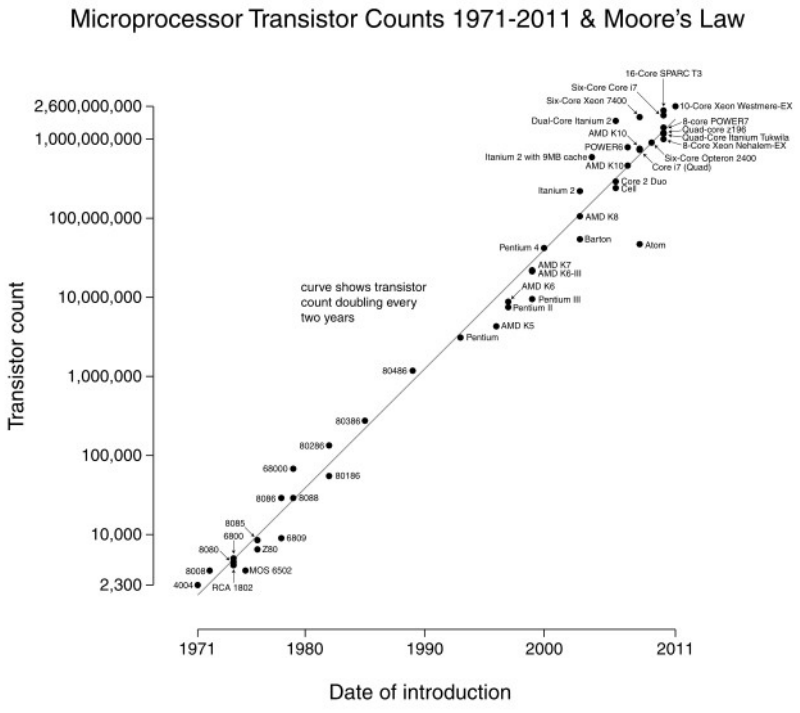
3.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο είδαμε διάφορους τύπους πολυμεσικού σήματος, τρόπους αναπαράστασής τους, τεχνικές επεξεργασίας εικόνων και μια σύντομη επισκόπηση όσον αφορά τη μετάδοση. Στο παρόν κεφάλαιο θα δούμε τους θεμέλιους λίθους για την παράλληλη επεξεργασία πληροφορίας (και κατ' επέκταση του πολυμεσικού σήματος), καθώς και τις δυσκολίες - επιπλοκές που παρουσιάζονται.

3.2 Νόμος του Moore

Ο Νόμος του Moore αποτυπώνει την παρατήρηση ότι ο αριθμός των τρανζίστορ στα ολοκληρωμένα κυκλώματα υπολογιστικών συστημάτων διπλασιάζεται κάθε δύο χρόνια περίπου. Είναι ένα εμπειρικό συμπέρασμα που οφείλεται στην εξέλιξη των μεθόδων κατασκευής ολοκληρωμένων κυκλωμάτων μαζικής παραγωγής και οφείλεται στον συνιδρυτή της εταιρείας ημιαγωγών Intel, Gordon Moore. Στηρίζεται σε προβλέψεις που είχαν διατυπωθεί πρότερα από επιφανείς επιστήμονες όπως ο Alan Turing και ο Douglas Engelbart. Το φαινόμενο της περιοδικής αύξησης των τρανζίστορ έχει πρακτικές επιπτώσεις στην αύξηση παραμέτρων όπως η επεξεργαστική ισχύς, η χωρητικότητα και η πολυπλοκότητα των υπολογιστικών συστημάτων. Βέβαια, χωρίς την υιοθέτηση διαφορετικών τεχνικών κατασκευής και υλικών, αυτή η αύξηση (και κατά συνέπεια η ολοκλήρωση σε όλο και μικρότερη κλίμακα) δεν μπορεί να συνεχίσει επ' αόριστον. Υπάρχουν αρκετοί περιορισμοί που σχετίζονται κυρίως με τα υλικά κατασκευής των ολοκληρωμένων κυκλωμάτων και τα φυσικά φαινόμενα που συνεπάγεται η ολοκλήρωση σε τόσο μικρή κλίμακα (όπως απαγωγή

θερμότητας, διάδοση του σήματος), τα οποία δε μας επιτρέπουν να στηριζόμαστε στην υπόθεση ότι ένα σειριακό πρόγραμμα σε μελλοντικά υπολογιστικά συστήματα θα συνεχίσει να εκτελείται όλο και πιο γρήγορα (με σταθερό παράγοντα βελτίωσης ή επιτάχυνσης). Αυτό συμβαίνει γιατί οι σχεδιαστές των σύγχρονων μονάδων επεξεργασίας (είτε πρόκειται για Κεντρικές Μονάδες Επεξεργασίας (CPUs) είτε όχι) προτιμούν να εκμεταλλευτούν το μεγαλύτερο αριθμό των τρανζίστορ για να υλοποιήσουν περισσότερες μονάδες επεξεργασίας στο ίδιο ολοκληρωμένο. Έτσι, μια εφαρμογή για να μπορεί να εκμεταλλεύεται τα μελλοντικά συστήματα όπου θα μπορεί να εκτελεστεί, θα πρέπει να λάβει υπόψη στη σχεδίαση και υλοποίησή της την έννοια του παραλληλισμού. Επίσης, μια παράμετρος που πρέπει να ληφθεί σοβαρά υπόψη είναι ότι ενώ οι Κεντρικές Μονάδες Επεξεργασίας συνεχώς βελτιώνονται με ταχείς ρυθμούς, ο ρυθμός μεταφοράς δεδομένων από και προς την κύρια μνήμη βελτιώνεται πρακτικά με πιο βραδύ ρυθμό. Αυτό έχει άμεσες συνέπειες στην επίδοση εφαρμογών όσον αφορά τη μη βέλτιστη χρήση του υλικού.



Ο Νόμος του Moore.

Έστω μια διαδικασία που τρέχει σειριακά σε μια μονάδα επεξεργασίας (επεξεργαστή). Είναι προφανές πως ασχέτως του πλήθους επεξεργαστών που μπορεί να διαθέτει ένα υπολογιστικό σύστημα, το κέρδος το οποίο εκφράζει την επιτάχυνση στην εκτέλεση σε σχέση με τον αριθμό των επεξεργαστών (speedup) θα είναι μηδενικό. Επίσης, μια διαδικασία που παραλληλοποιείται εξ ολοκλήρου (δηλαδή όλα τα μέρη που την αποτελούν μπορούν να παραλληλοποιηθούν) αναμένεται να έχει γραμμική επιτάχυνση στην εκτέλεση σε σχέση με τον αριθμό των επεξεργαστών που θα εκτελεστεί. Η ενδιάμεση περίπτωση αναμένουμε να έχει κάποιο κέρδος (όχι μέγιστο). Τα παραπάνω συνοψίζονται στο Νόμο του Amdahl, όπου όπως θα δούμε, τα πράγματα δεν είναι ακριβώς όπως τα περιμένουμε.

Έστω πρόγραμμα p το οποίο αποτελείται από μέρη που μπορούν να παραλληλοποιηθούν (λαμβάνοντας υπόψιν την καλύτερη δυνατή περίπτωση που αφορά γραμμική σχέση μεταξύ επιτάχυνσης και αριθμό επεξεργαστών) και μέρη που υποχρεωτικά (λόγω της φύσης τους) εκτελούνται σειριακά. Έστωσαν P το ποσοστό που παραλληλοποιείται και $1-P$ το ποσοστό σειριακής εκτέλεσης σύμφωνα με την παραπάνω υπόθεση. Τότε, η επιτάχυνση S συναρτήσει των μονάδων επεξεργασίας N θα είναι[1]

$$S(N)=\frac{1}{1-P+\frac{P}{N}}$$

Είναι προφανές ότι στη γενική περίπτωση (που πρακτικά ισχύει), όπου όλα τα μέρη μιας εφαρμογής δεν παραλληλοποιούνται, περισσότερες μονάδες επεξεργασίας δε συνεπάγονται γραμμική επιτάχυνση (η γραμμική επιτάχυνση θα ήταν το επιθυμητό). Μηδενικό κέρδος έχουμε για $S(N)=1$ και μέγιστο για $S(N)=N$. Μάλιστα, το κέρδος στην επιτάχυνση, στην πλειοψηφία των περιπτώσεων είναι αρκετά κατώτερο του διαισθητικώς αναμενόμενου. Αυτό είναι εμφανές με μια απλή αντικατάσταση στην παραπάνω σχέση. Για παράδειγμα, για ένα πρόγραμμα που παραλληλοποιείται στο 99% της υλοποίησής του, το κέρδος στην επιτάχυνση θα είναι περίπου 50 αν εκτελεστεί σε 100 επεξεργαστές. Ο Νόμος του

Amdahl υποδεικνύει το έσχατο αναμενόμενο κέρδος σε σχέση με τη σχεδίαση και υλοποίηση μιας εφαρμογής και την εκτέλεσή της σε ορισμένο αριθμό επεξεργαστών.

3.4 Υπολογιστικά μοντέλα και μηχανές

Η Μηχανή Turing (Turing Machine) είναι μια υποθετική μηχανή που μπορεί να εκτελέσει κάποιον αλγόριθμο. Επεξεργάζεται διαδοχικά (σειριακά) μια αέναη ταινία πάνω στην οποία είναι γραμμένα σε σειρά τα σύμβολα της γλώσσας που η μηχανή αναγνωρίζει. Η κεφαλή της μηχανής καταναλώνει ένα σύμβολο κάθε φορά και εκτελεί ένα πρόγραμμα (τον αλγόριθμο) το οποίο αποτελείται από μια κατάσταση (την τρέχουσα κατάσταση) και μια λίστα δυνατών καταστάσεων, καθώς και τους κανόνες μετάβασης από μια κατάσταση σε μια άλλη. Αυτό είναι ένα αφαιρετικό μοντέλο που χρησιμοποιείται στη Θεωρία Υπολογισμού για την (εκτός των άλλων) ανάλυση της πολυπλοκότητας ενός αλγόριθμου. Έχει αποδειχτεί ότι οποιοσδήποτε αλγόριθμος μπορεί επαρκώς να διατυπωθεί μπορεί να εκτελεστεί από μια τέτοια υποθετική μηχανή. Επέκταση αυτής της ιδέας αποτελεί η Καθολική Μηχανή Turing (Universal Turing Machine) η οποία είναι και αυτή μια Μηχανή Turing και αναλόγως την είσοδο μπορεί να εξομοιώσει μια άλλη "αυθαίρετη" Μηχανή Turing. Αυτό το μοντέλο είναι πλήρως συμβατό με τη θεώρηση της Αρχιτεκτονικής von Neumann κατά την οποία οι εντολές ενός προγράμματος και τα δεδομένα του προγράμματος υπάρχουν στον ίδιο χώρο στη μνήμη. Αυτό έχει ως συνέπεια, κατά την εκτέλεση του προγράμματος, η εντολή φόρτωσης ενός δεδομένου (σε κάποιον καταχωρητή) και η εντολή επεξεργασίας του δεδομένου να μην μπορούν να εκτελεστούν παράλληλα, αφού χρησιμοποιούν τον ίδιο δίαυλο μεταφοράς (bus).

Ισοδύναμο αφαιρετικό μοντέλο με τη Μηχανή Turing στη Θεωρία Υπολογισμού είναι ο λ-λογισμός (lambda calculus) που πρωτοδιατυπώθηκε

από τον Alonzo Church. Στηρίζεται στο μαθηματικό φορμαλισμό και στη μοντελοποίηση και έκφραση διαδικασιών με μαθηματικές συναρτήσεις οι οποίες αποτελούν "πολίτες πρώτης τάξης" (first-class citizens) - μπορούν δηλαδή να είναι παράμετροι για κάποια άλλη συνάρτηση, μπορούν να λειτουργούν ως "μεταβλητές" και ως "ανώνυμες παραστάσεις" - σταθερές (όπως δηλαδή ο αριθμός 42 θεωρείται μια βαθμωτή ακέραιη σταθερή ποσότητα). Η αναδρομή αποτελεί βασική φόρμα έκφρασης διαδικασιών. Με το μοντέλο αυτό έχουμε μια εξαιρετικά κομψή αναπαράσταση του υπολογισμού που θέλουμε να προβούμε και είναι ο θεμέλιος λίθος για τη δημιουργία γλωσσών συναρτησιακού προγραμματισμού. Ως αντιπαραβολή μπορούμε να δούμε τη Μηχανή Turing (στην πρωτόλεια θεώρησή της) ως το θεμέλιο λίθο για τον προστακτικό προγραμματισμό. Όπως όμως αναφέρουμε και παραπάνω, τα δύο μοντέλα είναι ισοδύναμα.

Μια άλλη αφαιρετική μηχανή, η οποία στηρίζεται στη Μηχανή Turing, είναι η Μηχανή Τυχαίας Προσπέλασης (Random-Access Machine - RAM). Ανήκει στην κλάση μηχανών με καταχωρητές (register machines). Αποτελεί μηχανή ακολουθιακού ή σειριακού υπολογισμού και μπορεί να χρησιμοποιεί τους καταχωρητές της και ως καταχωρητές έμμεσης διευθυνσιοδότησης (indirect addressing). Επί πλέον, υπάρχει καταχωρητής με το ρόλο του συσσωρευτή (accumulator). Στη γενική περίπτωση (χωρίς διάφορους περιορισμούς), μια τέτοια μηχανή είναι ισοδύναμη με μια Μηχανή Turing, οπότε και μπορούμε να προβούμε σε θεωρητική ανάλυση και αξιολόγηση σειριακών αλγόριθμων. Αξίζει να σημειωθεί πως, στη μοντελοποίηση με αυτήν τη μηχανή, δε λαμβάνουμε υπόψιν πρακτικά θέματα υλοποίησης όπως ο χρόνος προσπέλασης της μνήμης. Πάνω σε αυτό το σχεδιαστικό μοντέλο στηρίζεται η πλειοψηφία των υλοποιήσεων υπολογιστικών συστημάτων και η συμβολική γλώσσα μηχανής (assembly language) που εκθέτουν για τον προγραμματισμό τους. Αυτός είναι και ο κύριος λόγος που αναφέρεται εδώ η συγκεκριμένη μηχανή, χωρίς να εισέλθουμε σε περαιτέρω λεπτομέρειες.

Στηριζόμενοι στην παραπάνω μηχανή, για την ανάλυση παράλληλων

αλγόριθμων, ορίζουμε την Παράλληλη Μηχανή Τυχαίας Προσπέλασης (Parallel Random-Access Machine - PRAM). Σύμφωνα με το μοντέλο αυτό, έχουμε στη διάθεσή μας απεριόριστο αριθμό υπολογιστικών μονάδων, απεριόριστου μεγέθους κοινή μνήμη και ομοιόμορφη πρόσβαση σε αυτή από όλους τους επεξεργαστές.[10] Και εδώ υποθέτουμε ότι πρακτικά ζητήματα υλοποίησης δεν τα λαμβάνουμε υπόψιν, μιας και μας ενδιαφέρει η θεωρητική ανάλυση. Βέβαια, στον προγραμματισμό παράλληλων αλγόριθμων έχουμε περισσότερα τέτοια ζητήματα που αφορούν κυρίως το κόστος επικοινωνίας και συγχρονισμού κατά τον ταυτοχρονισμό. Ενδιαφέρον παρουσιάζει η περίπτωση απόπειρας δύο ή περισσοτέρων επεξεργαστών για ταυτόχρονη προσπέλαση της ίδιας θέσης μνήμης, όπου και έχουμε 4 διαφορετικές προσεγγίσεις στη στρατηγική επίλυσης του ανταγωνισμού για αυτόν τον πόρο.[18],[14]

1. *Αποκλειστική Ανάγνωση, Αποκλειστική Εγγραφή (Exclusive Read, Exclusive Write - EREW)*. Όλες οι ταυτόχρονες αιτήσεις σειριοποιούνται με αποτέλεσμα για κάθε δεδομένη χρονική στιγμή, μόνο ένας επεξεργαστής να έχει πρόσβαση σε κάθε θέση μνήμης. Αυτό αποτελεί ένα αρκετά περιοριστικό μοντέλο.

2. *Ταυτόχρονη Ανάγνωση, Αποκλειστική Εγγραφή (Concurrent Read, Exclusive Write - CREW)*. Κάθε θέση μνήμης μπορεί να αναγνωστεί από πολλούς επεξεργαστές ταυτόχρονα, αλλά στην περίπτωση που τα περιεχόμενα αλλάξουν (εγγραφή), πρόσβαση έχει μόνο ένας επεξεργαστής. Διευκολύνεται η υλοποίηση του ταυτοχρονισμού και είναι ένα λιγότερο περιοριστικό μοντέλο σε σχέση με το παραπάνω.

3. *Αποκλειστική Ανάγνωση, Ταυτόχρονη Εγγραφή (Exclusive Read, Concurrent Write - ERCW)*. Δεν έχει πρακτική εφαρμογή.

4. *Ταυτόχρονη Ανάγνωση, Ταυτόχρονη Εγγραφή (Concurrent Read, Concurrent Write - CRCW)*. Είναι μοντέλο χωρίς περιορισμούς, αφού επιτρέπει ταυτόχρονες αναγνώσεις και εγγραφές. Οι ταυτόχρονες αναγνώσεις δεν εισάγουν κινδύνους ασυμφωνίας στο περιεχόμενο της θέσης μνήμης, μιας και δεν αποτελούν λειτουργία που μεταλλάσει το περιεχόμενο

(mutation). Οι ταυτόχρονες εγγραφές όμως εισάγουν εξαρτήσεις. Έτσι, έχουμε διαφορετικές περιπτώσεις για το πώς γίνεται μια ταυτόχρονη εγγραφή. Αυτές είναι:

α) *Κοινή (Common)*. Όλοι οι επεξεργαστές γράφουν την ίδια τιμή, αλλιώς προκαλείται εξαίρεση (exception) κατά την εκτέλεση.

β) *Αυθαίρετη (Arbitrary)*. Εκλέγεται αυθαίρετα η αίτηση ενός επεξεργαστή για εγγραφή και οι υπόλοιπες αποσύρονται.

γ) *Με προτεραιότητα (Priority)*. Στους επεξεργαστές έχει δοθεί βαθμός προτεραιότητας και η αίτηση του επεξεργαστή με τη μεγαλύτερη προτεραιότητα επιτυγχάνει. Η εκλογή δηλαδή γίνεται με ντετερμινιστικό κριτήριο.

δ) *Με αναγωγή (Reduction)*. Πράξεις που γίνονται με τελεστές που το αποτέλεσμά τους δεν εξαρτάται από τη σειρά τους μπορούν να γίνουν ταυτόχρονα. Παραδείγματα αποτελούν το άθροισμα, το γινόμενο, το μέγιστο / ελάχιστο στοιχείο. Αυτό είναι μια πολύ ισχυρή θεώρηση για τον ταυτοχρονισμό και είναι κάτι που η παρούσα εργασία λαμβάνει σοβαρά υπόψιν.

3.5 Ταξινόμηση κατά Flynn

Η ταξινόμηση κατά Flynn είναι ένας τρόπος κατηγοριοποίησης αρχιτεκτονικών υπολογιστικών συστημάτων βάσει του αριθμού των ταυτόχρονων λειτουργιών σε ρεύματα δεδομένων που είναι ικανή να διεκπεραιώσει η εκάστοτε αρχιτεκτονική. Διακρίνουμε 4 κατηγορίες.[4]

1. *Μονής Εντολής, Μονού Δεδομένου (Single Instruction, Single Data stream - SISD)*. Με την αρχιτεκτονική αυτή έχουμε σειριακή εκτέλεση των λειτουργιών σε ρεύμα δεδομένων. Κάθε "κομμάτι" (δεδομένο), μέσω ενός καναλιού μεταφοράς, έρχεται στη μονάδα ελέγχου και κατόπιν υπόκειται σε επεξεργασία. Πρόκειται για αρχιτεκτονική σειριακής επεξεργασίας δεδομένων και ήταν η αρχιτεκτονική που μέχρι πρότινος βασιζόταν η πλειονότητα των υπολογιστικών συστημάτων μαζικής παραγωγής που

απευθύνονταν στο μέσο καταναλωτή. Συνήθως τέτοιου είδους αρχιτεκτονικές καλούνται Επεξεργαστές Βαθμωτών Μεγεθών (scalar processors).

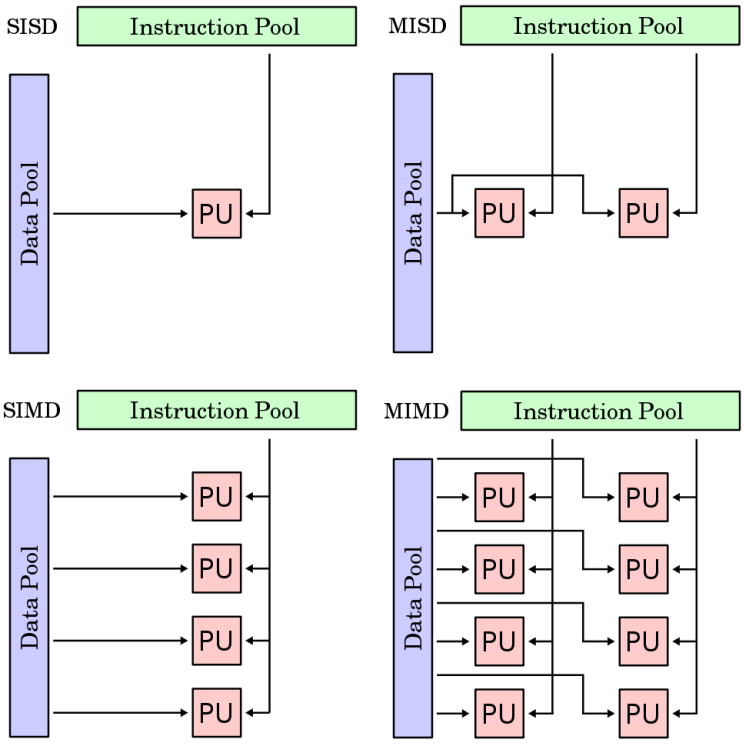
2. *Μονής Εντολής, Πολλαπλών Δεδομένων (Single Instruction, Multiple Data streams - SIMD)*. Με την ίδια εντολή επεξεργαζόμαστε πολλά δεδομένα. Αποτελεί έκφραση του "φυσικού παραλληλισμού" και έχει πληθώρα πρακτικών εφαρμογών. Ένα απλό παράδειγμα τέτοιου είδους παραλληλισμού είναι ο τελεστής πρόσθεσης ν-διάστατων διανυσμάτων. Αν δούμε τα βαθμωτά μεγέθη που αποτελούν ένα τέτοιο διάνυσμα ως δεδομένα και τον τελεστή ως εντολή, η λειτουργία "πρόσθεση" αποτελείται από προσθέσεις που γίνονται παράλληλα στα βαθμωτά αυτά μεγέθη. Π.χ. $\begin{bmatrix}42 & 69105 & 69\end{bmatrix}^T+\begin{bmatrix}5 & 1 & 3\end{bmatrix}^T=\begin{bmatrix}42+5 & 69105+1 & 69+3\end{bmatrix}^T$.Μοντέρνα κυκλώματα επεξεργασίας γραφικών που έχουν εξελιχθεί σε διανυσματικούς πολυ-επεξεργαστές είναι ένας τομέας πρακτικής εφαρμογής αυτής της αρχιτεκτονικής. Υλοποιήσεις SIMD συνήθως καλούνται Επεξεργαστές Διανυσματικών Μεγεθών (vector processors) ή Επεξεργαστές Ρευμάτων Δεδομένων (stream processors).

3. *Πολλαπλών Εντολών, Μονού Δεδομένου (Multiple Instruction, Single Data stream - MISD)*. Στο ίδιο δεδομένο επενεργούν διαφορετικές εντολές ταυτόχρονα. Αυτό μπορεί να υλοποιηθεί με την έκδοση πολλών αντιγράφων του δεδομένου (με ειδικό, αφοσιωμένο κύκλωμα) και την ενέργεια των διαφορετικών εντολών σε κάθε αντίγραφο. Δεν είναι αρχιτεκτονική που συναντάται συχνά.

4. *Πολλαπλών Εντολών, Πολλαπλών Δεδομένων (Multiple Instruction, Multiple Data streams - MIMD)*. Διαφορετικές εντολές εκτελούνται ταυτόχρονα σε διαφορετικές ομάδες δεδομένων. Αποτελεί την αρχιτεκτονική πάνω στην οποία στηρίζονται Κατανεμημένα Συστήματα (distributed systems), έχει τους λιγότερους περιορισμούς και είναι το μοντέλο στο οποίο στηρίζονται τα σύγχρονα υπολογιστικά συστήματα, ακόμα και αυτά που απευθύνονται στο μέσο καταναλωτή. Ένας επιπλέον διαχωρισμός αυτής της αρχιτεκτονικής, αναλόγως την απεικόνιση εκτέλεσης προγραμμάτων σε επεξεργαστές, είναι σε

α) *Μονό Πρόγραμμα, Πολλαπλά Δεδομένα (Single Programme, Multiple Data - SPMD)*, όπου κάθε επεξεργαστής εκτελεί το ίδιο πρόγραμμα (όχι απαραίτητα την ίδια χρονική στιγμή).

β) *Πολλαπλά Προγράμματα, Πολλαπλά Δεδομένα (Multiple Programmes, Multiple Data - MPMD)*, όπου κάθε επεξεργαστής εκτελεί παραπάνω από ένα πρόγραμμα ταυτόχρονα.



Ταξινόμηση κατά Flynn. Με PU συμβολίζεται η κάθε μονάδα επεξεργασίας.

3.6 Προγραμματιστική προσέγγιση παραλληλοποίησης

Ένα πρόβλημα θα πρέπει να είμαστε σε θέση να το διαιρέσουμε σε υπο-προβλήματα τα οποία και θα λύσουμε παράλληλα συμβάλλοντας στην τελική λύση. Γενικά αυτό αποτελεί δύσκολη διαδικασία και υπάρχουν αρκετοί παράγοντες που πρέπει να λάβουμε υπόψη. Αρχικά, η προσέγγισή εμπίπτει σε κάποια από τις εξής δύο κατηγορίες

1. *Παραλληλοποίηση προσανατολισμένη σε Εργασίες (Task-oriented Parallelism)*, όπου πρώτα μοιράζονται οι υπολογισμοί (απεικονίζονται σε εργασίες) και βάσει αυτών μοιράζονται τα δεδομένα.

2. *Παραλληλοποίηση προσανατολισμένη σε Δεδομένα (Data-oriented Parallelism)*, όπου πρώτα μοιράζονται τα δεδομένα και βάσει αυτών οι εργασίες.

Η πρώτη προσέγγιση είναι πιο γενική, συνήθως πιο εύκολη στη σύλληψη και ταιριάζει κυρίως σε προβλήματα που μοντελοποιούνται με ανομοιογενείς, δυναμικές και ακανόνιστες δομές δεδομένων, ενώ η δεύτερη συνήθως καταλήγει σε πιο απλές και κομψές υλοποιήσεις και ταιριάζει κυρίως σε προβλήματα που μοντελοποιούνται με ομοιογενείς και στατικές δομές δεδομένων - σε χωρία δηλαδή που εύκολα μπορούν να διαιρεθούν σε υπο-χωρία, όπου και με την ίδια προσέγγιση φτάνουμε σε λύση η οποία εξαρτάται από τα σύνορα και τις αρχικές συνθήκες του εκάστοτε υπο-χωρίου. Πάντως, και στις δύο προσεγγίσεις, καταλήγουμε να έχουμε κατανομή εργασιών και κατανομή δεδομένων.

Κοινός παράγοντας στις αποδοτικές προσεγγίσεις παραλληλοποίησης είναι το γεγονός ότι απαιτείται περισσότερος χώρος και περισσότερη εργασία από το σύστημα για τη λύση του προβλήματος σε σχέση με τη σειριακή λύση. Είναι κοινή πεποιήθηση ότι "ο παράλληλος προγραμματισμός είναι κάτι απλό όσο δεν ενδιαφερόμαστε για την επίδοση". Τις περισσότερες φορές, μια απλή και αφελής προσέγγιση οδηγεί σε κατώτερες επιδόσεις από την αντίστοιχη σειριακή λύση.

3.7 Κατανομή δεδομένων

Τα δεδομένα ανά εργασία χωρίζονται σε αποκλειστικά δεδομένα (private data), δηλαδή δεδομένα στα οποία έχει πρόσβαση μόνο η εκάστοτε εργασία, μοιραζόμενα δεδομένα (shared data), δηλαδή δεδομένα που

βρίσκονται σε χώρο διευθύνσεων μνήμης όπου διαφορετικές εργασίες έχουν πρόσβαση, κατανεμημένα δεδομένα (distributed data), τα οποία είναι δεδομένα που ανταλλάσσονται μεταξύ εργασιών μιας και δε βρίσκονται στον ίδιο χώρο διευθύνσεων μνήμης, σε καθολικά δεδομένα (uniform data) που έχουν αρχικοποιηθεί πριν την έναρξη του υπολογισμού, είναι δεδομένα μόνο προς ανάγνωση (read-only), πρόσβαση έχουν όλες οι εργασίες και ο χώρος διαμονής τους εξαρτάται από την εκάστοτε υλοποίηση και σε αντιγραμμένα δεδομένα (replicated data) που χρησιμοποιούνται για την αντιγραφή των υπολογισμών των υπο-προβλημάτων. Η διαχείριση των αποκλειστικών δεδομένων δεν παρουσιάζει επιπλοκές μιας και είναι απομονωμένα δεδομένα για κάθε εργασία. Στον αντίποδα, για τα μοιραζόμενα δεδομένα το προγραμματιστικό πλαίσιο πρέπει να παρέχει στοιχεία (primitives) συγχρονισμού, τα οποία και εκθέτει στον προγραμματιστή, για τη διαχείριση συγχρονισμού και των συνεπαγόμενων φαινομένων ανταγωνισμού (race conditions), κτλ.[18] Ακόμη, η διαχείριση της κατανεμημένης μνήμης γίνεται με ανταλλαγή μηνυμάτων μεταξύ των εργασιών, έναν αρκετά διαφανή τρόπο ταυτόχρονου προγραμματισμού που οδηγεί σε λιγότερο πολύπλοκες υλοποιήσεις και συνήθως οδηγεί σε καλύτερη κλιμάκωση (scaling) υλοποιήσεων μαζικού παραλληλισμού (massive parallelism) σε ανομοιογενή και κατανεμημένα συστήματα.

3.8 Εξαρτήσεις και συγχρονισμός

Ως τώρα σκόπιμα δεν έχει γίνει σαφής διαχωρισμός και απεικόνιση εργασιών σε διεργασίες (processes), νήματα (threads), κ.ά, μιας και αναλόγως του προγραμματιστικού μοντέλου και συνεπώς του προγραμματιστικού πλαισίου (API) που παρέχεται στον προγραμματιστή, αυτή η απεικόνιση και οι συμβάσεις της ενδέχεται να έχουν αρκετά σημεία που διαφέρουν. Το εκάστοτε λειτουργικό σύστημα σε συνδυασμό με το προγραμματιστικό πλαίσιο επιτρέπουν την υλοποίηση των διάφορων εργασιών με τη βοήθεια διεργασιών, νημάτων και άλλων δομικών στοιχείων

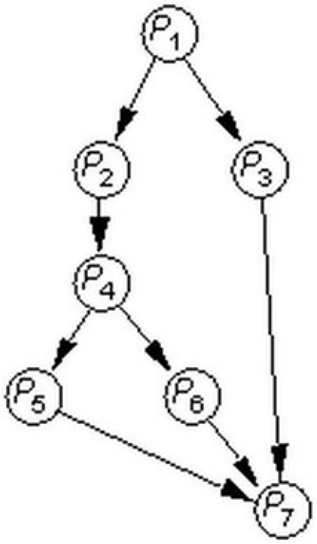
που συντονίζει ο χρονοδρομολογητής (scheduler) του λειτουργικού συστήματος όσον αφορά το χρονοπρογραμματισμό. Κάθε διεργασία έχει το δικό της αποκλειστικό χώρο διευθύνσεων στη μνήμη και μπορεί να εκκινήσει άλλες διεργασίες. Ο έλεγχος που θα έχει στις διεργασίες αυτές καθορίζεται από την αρχιτεκτονική του λειτουργικού συστήματος. Η επικοινωνία μεταξύ διεργασιών γίνεται με τις μεθόδους δια-διεργασιακής επικοινωνίας (inter-process communication) που παρέχει το λειτουργικό σύστημα. Μια διεργασία εκτελείται στο κύριο νήμα της. Μπορεί να εκκινήσει και άλλα νήματα τα οποία θα υπάρχουν στον ίδιο χώρο διευθύνσεων μνήμης. Η επικοινωνία μεταξύ νημάτων της ίδιας διεργασίας είναι άμεση. Κάθε νήμα μπορεί να εκκινήσει μία ή περισσότερες ίνες ή πράσινα νήματα ή πράσινες διεργασίες (fibers, green threads, green processes) για το χρονοπρογραμματισμό των οποίων όμως είναι υπεύθυνο. Τέτοιες μεθόδους υλοποίησης συναντάμε συχνά σε εικονικές μηχανές (virtual machines) ή σε προγράμματα που τρέχουν σε λειτουργικά συστήματα που δεν υποστηρίζουν ταυτόχρονη εκτέλεση πολλαπλών εργασιών (multi-tasking).

Ακόμη και στην απλή περίπτωση ενός σειριακού προγράμματος υπάρχουν εξαρτήσεις οι οποίες επιβάλλουν κάποια σειρά στις λειτουργίες του προγράμματος, ώστε να έχουμε ορθή εκτέλεσή του. Φερ' ειπείν αν ένας υπολογισμός χρειάζεται το αποτέλεσμα ενός προηγούμενου υπολογισμού, τότε για να έχουμε σημασιολογικά ορθό αποτέλεσμα, είναι προφανές ότι ο δεύτερος υπολογισμός πρέπει να προηγηθεί του πρώτου.

Τα παραπάνω έχουν ιδιαίτερη σημασία για τον ταυτοχρονισμό όσον αφορά την πρόσβαση σε κοινούς πόρους. Ένα απλό παράδειγμα είναι το ακόλουθο. Στην κοινή μνήμη (shared memory) όπου έχουν πρόσβαση δύο διεργασίες, την ίδια χρονική στιγμή η μία από αυτές αλλάζει την τιμή μιας μεταβλητής και η άλλη τη διαβάζει. Αν ο κοινός πόρος (η μνήμη) δεν προστατεύεται από κάποιο μηχανισμό, τότε η διεργασία που διαβάζει, ενδέχεται να διαβάσει μια εντελώς λάθος τιμή (αφού δε θα έχει τελειώσει

την εγγραφή η άλλη διεργασία). Αξίζει να σημειώσουμε ότι οι εξαρτήσεις εμφανίζονται σε όλα τα επίπεδα της υλοποίησης είτε αναφερόμαστε σε επίπεδο λογισμικού (λειτουργικό σύστημα, διεργασίες, νήματα, κτλ) είτε σε επίπεδο υλικού, δηλαδή επεξεργαστής με παραλληλισμό σε επίπεδο εντολών (Instruction Level Parallelism - ILP) που εφαρμόζεται ιδιαίτερα σε μοντέρνες αρχιτεκτονικές εκτέλεσης-εκτός-σειράς (out-of-order execution) και παραλληλισμό σε επίπεδο bits (Bit Level Parallelism), όπου η αρχιτεκτονική μας εγγυάται ατομικές ενημερώσεις (atomic updates) σε αντικείμενα που καταλαμβάνουν προκαθορισμένο χώρο στη μνήμη (συνήθως όσο και το μέγεθος της λέξης του επεξεργαστή).

Σε υψηλότερο επίπεδο, λαμβάνοντας υπόψη και διεργασίες, έχουμε μια θεώρηση για τις εξαρτήσεις που περιγράφεται με το γράφο προήγησης (precedence graph). Πρόκειται για έναν κατευθυνόμενο ακυκλικό γράφο που οι κόμβοι του αντιστοιχούν σε διεργασίες. Ο κάθε κλάδος του K_{ij} , από τον κόμβο i στον κόμβο j , δηλώνει ότι η διεργασία P_j (κόμβος j) μπορεί να εκτελεστεί μόνο μετά την ολοκλήρωση της διεργασίας P_i (κόμβος i). Ο γράφος προήγησης αποτελεί έναν τρόπο μοντελοποίησης παραλληλισμού ενός ταυτόχρονου προγράμματος. Οι εισερχόμενοι και εξερχόμενοι κλάδοι από έναν κόμβο που περιγράφουν τις εξαρτήσεις και την ενορχήστρωση της επικοινωνίας και του συγχρονισμού, στη συνέχεια υλοποιούνται με το εκάστοτε προγραμματιστικό πλαίσιο που έχει επιλεγεί.[17]



Παράδειγμα γράφου προήγησης.

Είναι φανερό ότι για την επίτευξη του ταυτοχρονισμού θα πρέπει να παρέχονται διαδικασίες για τη σωστή διαχείριση κρίσιμων τμημάτων (critical sections), ώστε να μπορούν να εκτελούνται αδιαίρετα ακολουθίες εντολών. Επίσης, θα πρέπει η δομή του προγράμματος να είναι τέτοια, ώστε να αποφεύγονται αδιέξοδα (deadlocks). Το πρόβλημα του κρίσιμου τμήματος διατυπώθηκε και λύθηκε από τον Dijkstra με τους εξής περιορισμούς:[17]

- Όταν μια διεργασία εκτελεί το κρίσιμο τμήμα της, σε καμία άλλη δεν μπορεί να επιτραπεί να εκτελεί το δικό της κρίσιμο τμήμα.

- Στην περίπτωση που οι διεργασίες προσπαθούν να εισέλθουν στο κρίσιμο τμήμα τους την ίδια χρονική στιγμή, η επιλογή γίνεται με τυχαίο τρόπο.

- Όταν μια διεργασία βρίσκεται στο μη κρίσιμο τμήμα της, δεν μπορεί να αποτρέψει άλλες διεργασίες από το να μπουν στο κρίσιμο τμήμα τους.

- Όταν πρέπει να αποφασισθεί ποια από τις διεργασίες που ανταγωνίζονται θα μπει στο κρίσιμο τμήμα της, η επιλογή δεν μπορεί να αναβάλλεται επ' άπειρο.

- Πρέπει να έχουν την ευκαιρία όλες οι διεργασίες να μπορούν να εκτελούν το κρίσιμο τμήμα τους μέσα σε λογικό χρονικό διάστημα και όχι μερικές να μονοπωλούν το σύστημα.

3.9 Μονοειδή (Monoids)

Τα μονοειδή είναι αλγεβρικές δομές με έναν τελεστή εφοδιασμένο με την προσεταιριστική ιδιότητα (associative property) και ένα ταυτοτικό στοιχείο (identity element). Μπορούν να εκφράσουν τη σύνθεση συναρτήσεων στα στοιχεία ενός συνόλου. Επίσης, διάφορες μορφές μονοειδών μπορούν να περιγράψουν Μηχανές Πεπερασμένων Καταστάσεων (Finite State Machines), λογισμό διεργασιών (process calculus) και ταυτόχρονη υπολογιστική (concurrent computing).

Ένα μονοειδές είναι ένα σύνολο S με ένα δυαδικό τελεστή $(.)$ που ικανοποιεί τα παρακάτω 3 αξιώματα:[13]

$\forall a,b \in S \ : \ (a \ . \ b) \in S \quad [\textit{εγκλεισμός}]$
$\forall a,b,c \in S \ : \ (a \ . \ b) \ . \ c = a \ . \ (b \ . \ c) \quad [\textit{προσεταιριστικότητα}]$
$\exists e \in S \ : \ \forall a \in S \ : \ e \ . \ a = a \ . \ e \quad [\textit{ταυτοτικό στοιχείο}]$

Κατά τον παραλληλισμό ένα τέτοιο σύνολο διευκολύνει αρκετά τη διαδικασία όπου παράλληλα μπορούμε να συνδυάσουμε τα στοιχεία του για την παραγωγή του αποτελέσματος, χωρίς επιπλέον εξαρτήσεις. Ακόμα περισσότερο μας διευκολύνει η περίπτωση που ο τελεστής διαθέτει και την αντιμεταθετική ιδιότητα (commutative). Απλό παράδειγμα τέτοιου τελεστή είναι η πρόσθεση δύο οποιοδήποτε πραγματικών αριθμών. Ισχύει

$\textit{για} \ \ a,b \in \mathbb{R} \ \ \textit{είναι} \ \ a+b \in \mathbb{R} \quad [\textit{εγκλεισμός}]$
$\textit{ισχύει} \ \ a+b=b+a \quad [\textit{αντιμεταθετικότητα}]$
$\textit{ισχύει} \ \ a+(b+c)=(a+b)+c \quad [\textit{προσεταιριστικότητα}]$
$\textit{ισχύει} \ \ a+0=a \quad [\textit{το } 0 \textit{ είναι ταυτοτικό στοιχείο}]$

Στην περίπτωση που ο τελεστής δεν ικανοποιεί κάποιες από τις παραπάνω ιδιότητες, προσπαθούμε να βρούμε ένα υπερ-σύνολο S' του S του οποίου ο τελεστής που θα χρησιμοποιήσουμε να τις ικανοποιεί. Στη συνέχεια εργαζόμαστε με τον παρακάτω τρόπο, ο οποίος καλείται συζυγής μετασχηματισμός (conjugate transform)[13]

- Απεικονίζουμε τα στοιχεία s_i του S σε στοιχεία s'_i του S' .

- Εργαζόμαστε με τα στοιχεία του συνόλου S' .

- Απεικονίζουμε τα στοιχεία που προέκυψαν ως αποτελέσματα του παραπάνω βήματος πίσω στο σύνολο S .

Κατά το συζυγή μετασχηματισμό χρησιμοποιούμε την ιδιότητα του καταμορφισμού (ομομορφισμός από μια αρχική άλγεβρα σε μια άλλη). Αξίζει να σημειωθεί ότι η στρατηγική του παραλληλισμού δεν απαιτεί την αντιμεταθετική ιδιότητα, αλλά μόνο τις ιδιότητες των μονοειδών.

Ένα απλό παράδειγμα χρήσης της παραπάνω τεχνικής είναι το ακόλουθο. Έστω ότι θέλουμε να ξέρουμε ανά πάσα στιγμή το ύψος ενός δέντρου ή ενός υπο-δέντρου του. Το δέντρο δεν είναι στατικό (έχουμε συνεχώς εισαγωγή και διαγραφή κόμβων). Το να διατρέχουμε κάθε φορά το εκάστοτε υπο-δέντρο είναι μια μη αποδοτική διαδικασία. Αν όμως περάσουμε από την αφηρημένη δομή δεδομένων που περιγράφει το δέντρο σε μια πιο "εμπλουτισμένη" δομή που προέρχεται από αυτήν, η επίλυση γίνεται αποδοτική. Έτσι, σε κάθε κόμβο του δέντρου, εκτός από το φορτίο (τα δεδομένα του κόμβου), συμπεριλαμβάνουμε και έναν αριθμό που δείχνει το ύψος από αυτόν τον κόμβο, οπότε σε κάθε εισαγωγή ή διαγραφή ενημερώνουμε αυτόν τον αριθμό. Η ενημέρωση δεν εισάγει παραπάνω κόστος από αυτό της πράξης (εισαγωγή ή διαγραφή).

Αν η παραπάνω διαδικασία δεν είναι εφικτή, αναγκαζόμαστε να καταφύγουμε σε λύσεις με κατάσταση (stateful), όπου πρέπει να έχουμε ένα μοντέλο περιγραφής του υπολογισμού για κάθε ζευγάρι τελεσταίων, το οποίο εξαρτάται από τους μετασχηματισμούς που έχουν προηγηθεί, οπότε και εισάγονται επιπλέον εξαρτήσεις στα δεδομένα.

3.10 Προθεματικός Συμβολισμός (Prefix Notation)

Ο προθεματικός συμβολισμός είναι ένας τρόπος συμβολισμού αλγεβρικών και λογικών εκφράσεων, κατά τον οποίο ο τελεστής τοποθετείται πριν τους τελεσταίους. Σε αντίθεση με το συνηθισμένο εντυπωμένο συμβολισμό (infix notation) όπου οι τελεστές τοποθετούνται ανάμεσα στους τελεσταίους. Με αυτόν το συμβολισμό γίνεται ξεκάθαρη η εφαρμογή του ίδιου τελεστή σε πολλούς τελεσταίους χωρίς να χρειάζεται να τον επαναλαμβάνουμε (για παράδειγμα, στις παραστάσεις $(+\ 5\ 1\ 3\ 9\ 7)$ και $(5+1+3+9+7)$ βλέπουμε πόσο πιο συνοπτικός και περιεκτικός είναι ο πρώτος συμβολισμός σε σχέση με το δεύτερο). Με αυτόν τον τρόπο μπορούμε με μαθηματικά αφαιρετικό τρόπο να εκφράσουμε κομψά και μετασχηματισμό πολλών δεδομένων με τον ίδιο τρόπο. Άλλωστε, η γλώσσα LISP χρησιμοποιεί αυτόν το συμβολισμό για τις συμβολικές εκφράσεις. Οι συμβολικές εκφράσεις είναι λίστες των οποίων το πρώτο στοιχείο αποτελεί το αναγνωριστικό της έκφρασης (π.χ. μια "ετικέτα", μια συνάρτηση - το στοιχείο που υποδεικνύει τη λειτουργία της λίστας). Επίσης, αποτελεί τρόπο συμβολισμού που ταιριάζει απόλυτα για την έκφραση του SIMD τρόπου επεξεργασίας.

3.11 Αλγοριθμικοί Σκελετοί (Algorithmic Skeletons)

Οι παραπάνω αρχές ισχύουν και στην περίπτωση που εξετάζουμε ταυτοχρονισμό με νήματα ή άλλους τρόπους υλοποίησής του. Στόχος είναι να είμαστε σε θέση να υλοποιούμε προγραμματιστικά μοτίβα τα οποία εμπεριέχουν την έννοια του συγχρονισμού, ώστε σε υψηλότερο επίπεδο να εκφράζουμε τη συνεργασία που οδηγεί στην παραλληλοποίηση του προβλήματος. Αυτά τα μοτίβα καλούνται Αλγοριθμικοί Σκελετοί. Μια λίστα με τα ευρέως γνωστά είναι η ακόλουθη:[5],[7],[2]

- *Φάρμα (Farm)*. Χρησιμοποιείται για προβλήματα πλήρως παραλληλοποιήσιμα, χωρίς εξαρτήσεις, των οποίων οι εργασίες μοιράζονται στους κόμβους της φάρμας και επιλύονται.

- *Σωλήνωση (Pipe)*. Περιγράφουμε μια ροή εργασιών που συνδυάζονται για την επίλυση του προβλήματος.

- *Για-κάθε, Όσο-Ισχύει (For, While)*. Περιγράφουν μοτίβα που μας επιτρέπουν να διατρέχουμε συλλογές (iteration) ή μια εργασία να εκτελείται για ορισμένο (ή όσο ισχύει κάποια συνθήκη) αριθμό επαναλήψεων.

- *Εάν (If)*. Υλοποιεί τη διακλάδωση με κριτήριο εάν μια συνθήκη ισχύει.

- *Απεικόνιση-Αναγωγή (Map-Reduce)*. Τα δεδομένα είναι οργανωμένα σε δομή δεδομένων που αναπαρίσταται από κάποια συλλογή αντικειμένων $(x_{1,}x_{2,},\ldots,x_n)$. Κάθε αντικείμενο x_i της συλλογής υπόκειται στο μετασχηματισμό $y_i=f(x_i)$. Αυτό είναι το στάδιο της απεικόνισης. Προκύπτει έτσι μια άλλη συλλογή αντικειμένων $(y_{1,}y_{2,},\ldots,y_n)$, της οποίας τα στοιχεία στη συνέχεια συνδυάζονται ώστε να προκύψει ένα νέο αντικείμενο z (αναγωγή). Στο στάδιο αυτό χρησιμοποιούμε τις τεχνικές που αναφέρθηκαν στην παραπάνω παράγραφο περί μονοειδών. Για τους συνδυασμούς των στοιχείων χρησιμοποιούμε συνήθως δέντρα αναγωγής (reduction trees) με αλγόριθμους για προθεματική άθροιση (prefix sum). Συνηθέστεροι αλγόριθμοι είναι του Blelloch[2] και των Hillis – Steele[7].

- *Διαίρει-και-Βασίλευε (Divide-and-Conquer)*. Κάθε εργασία διαιρείται σε αυτόνομες υπο-εργασίες, οι οποίες επιλύονται ανεξάρτητα (μερικές λύσεις). Στη συνέχεια, η λύση προκύπτει από την αναδρομική επίλυση των μερικών λύσεων.

- *Με Αποτύπωση (Stencil)*. Ένα χωρίο διαιρείται σε υπο-χωρία. Κάθε στοιχείο x_i κάθε υποχωρίου υπόκειται σε υπολογισμό για τον οποίο συμβάλλουν τα γειτονικά στοιχεία του x_i . Είναι δηλαδή $y_i=f(x_i,N(x_i))$, όπου $N(x_i)$ μια γειτονιά του x_i . Η γειτονιά αυτή έχει συγκεκριμένες διαστάσεις και καλείται πυρήνας (kernel) του υπολογισμού. Οι εξαρτήσεις εμφανίζονται με τη μορφή οριακών συνθηκών για κάθε υπο-χωρίο.

- *Επέκταση-και-Οριοθέτηση (Branch-and-Bound)*. Το πρόβλημα μοντελοποιείται με μια περιγραφή που αποτελεί την τρέχουσα κατάσταση. Μια τρέχουσα κατάσταση είναι πρόγονος επόμενων καταστάσεων, στις οποίες πρέπει να αναζητήσουμε τη λύση ή είναι μερικές λύσεις του

προβλήματος. Με κάποιο κριτήριο αποκόπτουμε (pruning) καταστάσεις οι οποίες δε συμβάλλουν στη λύση, οπότε και έχουμε την οριοθέτηση στο μέτωπο αναζήτησης.

- Ακολουθιακό (Sequential). Δεν παριστάνει παραλληλισμό. Μας διευκολύνει όμως στη διασύνδεση των δομικών στοιχείων του Σκελετού.

3.12 Ετερογενής Παραλληλισμός

Ιστορικά, θα πρέπει να αναφέρουμε ότι πάντα οι υπολογιστές μαζικής παραγωγής διέθεταν ειδικά κυκλώματα (αναλόγως το κοινό που απευθύνονταν) για εργασίες όπως η απεικόνιση γραφικών σε συσκευές (π.χ. οθόνες), η αναπαραγωγή ήχου, η Απ' ευθείας Προσπέλαση Μνήμης (Direct Memory Access - DMA), η ψηφιακή επεξεργασία σήματος, η δικτυακή επικοινωνία, κ.ά. Έτσι, το υβριδικό ή ετερογενές υπολογιστικό μοντέλο υπολογισμού είναι κάτι το οποίο οι σχεδιαστές των συστημάτων λάμβαναν υπόψη και κατά συνέπεια με διάφορα προγραμματιστικά μοντέλα (ή τις περισσότερες φορές μέσω γλώσσας μηχανής) εξέθεταν στον προγραμματιστή του συστήματος.

Όπως αναφέραμε παραπάνω, οι σχεδιαστική τάση στο υλικό είναι ξεκάθαρο πια πως είναι οι περισσότεροι πυρήνες και οι επεξεργαστές με περισσότερες δυνατότητες για επίλυση διαφορετικών προβλημάτων με τη βοήθεια παράλληλων υπολογισμών. Επειδή όμως κάθε αρχιτεκτονική δεν μπορεί να ευνοεί την αποδοτική επίλυση πάσης φύσεως προβλήματος και κατά συνέπεια όλα τα σχεδιαστικά μοτίβα που χρησιμοποιούνται, είναι φανερό ότι η συνύπαρξη διαφορετικών αρχιτεκτονικών στο ίδιο υπολογιστικό σύστημα είναι σημαντική. Το παραπάνω είναι ιδιαίτερα ενδεικτικό για τη φιλοσοφία σχεδίασης ενός συστήματος που μπορεί να παρέχει με αφαιρετικό τρόπο ευέλικτα προγραμματιστικά πλαίσια, ώστε να εκμεταλλεύεται ο χρήστης τη δύναμη ενός σύγχρονου υπολογιστικού συστήματος. Με τη σύνθεση μοτίβων όπως οι Αλγοριθμικοί Σκελετοί και την

παραδοχή της επεκτασιμότητας και κλιμάκωσης ενός συστήματος, οι λύσεις που προκύπτουν έχουν την ιδιότητα της ευελιξίας και της αντοχής στο χρόνο.

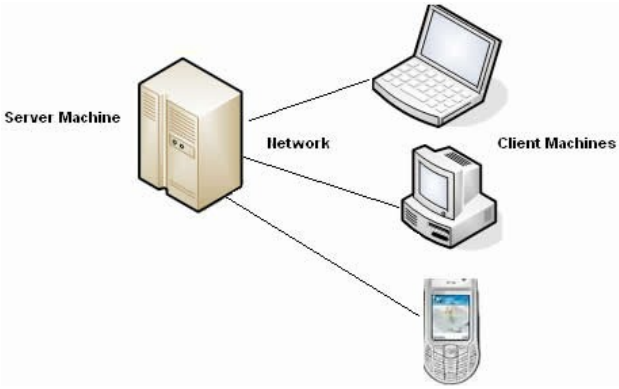
Ο ετερογενής παραλληλισμός σήμερα συναντάται σε πληθώρα συστημάτων. Από υπερ-υπολογιστές υψηλών επιδόσεων (High Performance Computing - HPC) μέχρι και κοινούς υπολογιστές καθημερινής χρήσης. Παρακάτω παραθέτουμε ενδεικτικά διάφορα πρότυπα και προγραμματιστικά μοντέλα, η σύνθεση των οποίων μας οδηγεί στην ανάπτυξη εφαρμογών ετερογενών ή υβριδικών συστημάτων. Κάποια από αυτά αναφέρονται στη διαδικασία της σχεδίασης και κάποια στη διαδικασία της υλοποίησης. Σκοπός είναι η αντίληψη της έκφρασης τεχνικών παραλληλισμού και στοιχείων υλοποίησης Αλγοριθμικών Σκελετών μέσα από το πρίσμα των σταδίων ανάπτυξης ενός τέτοιου συστήματος.

3.12.1 Μοντέλο Πελάτη-Εξυπηρετητή (Client-Server Model)

Το μοντέλο αυτό είναι ευρέως διαδεδομένο, ιδιαίτερα στη διαδικτυακή επικοινωνία όσον αφορά την παροχή υπηρεσιών. Αρχικά αναπτύχθηκε από την εταιρεία Xerox στο Xerox Palo Alto Research Center (Xerox PARC), ένα πασίγνωστο ερευνητικό κέντρο στο οποίο γεννήθηκαν αρκετές από τις μοντέρνες ιδέες στην πληροφορική, όπως τα Γραφικά Περιβάλλοντα Αλληλεπίδρασης (GUIs) με παραθυρικό περιβάλλον, η συσκευή ποντίκι και πολλά άλλα. Αποτελεί εδραιωμένο μοντέλο με πάρα πολλές εφαρμογές. Ας μην ξεχνάμε πως ο Παγκόσμιος Ιστός (World Wide Web) στηρίζεται σε αυτό το μοντέλο.

Το μοντέλο Πελάτη-Εξυπηρετητή στηρίζεται στην απλή αρχή του ότι πολλοί πελάτες (clients) στέλνουν αιτήσεις εξυπηρέτησης με τη βοήθεια ενός προσυμφωνημένου πρωτοκόλλου σε έναν εξυπηρετητή (server), ο οποίος τις λαμβάνει, τις επεξεργάζεται και τις εξυπηρετεί. Πρόκειται περί

ενός μοντέλου επικοινωνίας διαφορετικών συστημάτων το οποίο στηρίζεται στην ύπαρξη του πρωτοκόλλου για την επίτευξη της αλληλεπίδρασης. Δεν είναι απαραίτητο ο πελάτης και ο εξυπηρετητής να είναι διαφορετικά υπολογιστικά συστήματα. Μπορεί να είναι συστήματα που φιλοξενούνται στην ίδια "μηχανή". Παράδειγμα για το παραπάνω αποτελεί το παραθυρικό περιβάλλον X (X Window System), όπου ο X server μπορεί να εκτελείται στον ίδιο υπολογιστή με τους X clients, τις εφαρμογές δηλαδή που εξυπηρετεί.



Πελάτες και εξυπηρετητής.

Ο εξυπηρετητής εκφράζει τον πάροχο πόρων τους οποίους παρέχει κατόπιν αίτησης. Είναι μια σχέση συνεργασίας ανάμεσα σε διαφορετικά συστήματα, οπότε είναι εύκολα αντιληπτό πως πρόκειται για μια αφαιρετική θεώρηση αλληλεπίδρασης. Επίσης, η ταυτόχρονη εξυπηρέτηση αιτήσεων είναι απαραίτητη για ένα σύγχρονο σύστημα εξυπηρετητή, μιας και ένας εξυπηρετητής μπορεί να εξυπηρετεί πολλούς πελάτες. Αυτό δημιουργεί την ανάγκη υλοποίησης μοτίβων για την ταυτόχρονη διαχείριση εργασιών λαμβάνοντας υπόψιν το φορτίο ανάλογα με την παρεχόμενη υπηρεσία. Έτσι, υπάρχουν τεχνικές για εξισορρόπηση φορτίου (load balancing) και κατακερματισμό (sharding), ώστε να εξασφαλίζεται όσο το δυνατόν περισσότερο άμεση απόκριση στις διάφορες αιτήσεις. Ένα άλλο πολύ βασικό μέρος της υλοποίησης του εξυπηρετητή είναι η αντοχή του. Η παρεχόμενη υπηρεσία πρέπει να είναι διαθέσιμη χωρίς διακοπή και χωρίς έλλειψη στην

όποια αποθηκευμένη πληροφορία. Άρα, ο πλεονασμός (redundancy) σε αντίγραφα ασφαλείας (back-up copies), σε μέρη του συστήματος σε επίπεδο υλικού (hardware) και σε ισοδύναμα - όμοια συστήματα που ανά πάσα στιγμή μπορούν να αναλάβουν την εξυπηρέτηση σε περίπτωση βλάβης του κύριου εξυπηρετητή (replicas) αποτελούν πρακτικές που χρησιμοποιούνται εκτενώς.

Κατά τη συνεργασία μεταξύ διαφορετικών συστημάτων δεν είναι απαραίτητο ένα σύστημα να χαρακτηρίζεται πλήρως ως πελάτης ή ως εξυπηρετητής. Για κάποια υπηρεσία μπορεί να λειτουργεί ως εξυπηρετητής και ταυτόχρονα για κάποια άλλη να λειτουργεί ως πελάτης. Η επικοινωνία μεταξύ εξυπηρετητών για τη μετάδοση κάποιας πληροφορίας που είναι αναγκαίο να γνωστοποιηθεί αποτελεί ένα παράδειγμα τέτοιας λειτουργίας.

```
/*
 * server.c - A very simple server.
 *
 * %gcc -Wall -Wextra -o server server.c
 * %./server
 */


#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <time.h>


int
main(void)
{
      int listenfd = 0, connfd = 0;
      struct sockaddr_in serv_addr;
```

```
	char sendBuff[1025];
	time_t ticks;

	listenfd = socket(AF_INET, SOCK_STREAM, 0);
	memset(&serv_addr, 0x00, sizeof(serv_addr));
	memset(sendBuff, 0x00, sizeof(sendBuff));

	serv_addr.sin_family = AF_INET;
	serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
	serv_addr.sin_port = htons(5000);

	bind(listenfd, (struct sockaddr *) &serv_addr,
sizeof(serv_addr));

	listen(listenfd, 10);

	for (;;) {
		connfd = accept(listenfd, (struct sockaddr *) NULL, NULL);

		ticks = time(NULL);
		snprintf(sendBuff, sizeof(sendBuff),
			"%.24s\r\n", ctime(&ticks));
		write(connfd, sendBuff, strlen(sendBuff));

		close(connfd);
		sleep(1);
	}

	return 0;
}
```

Υλοποίηση ενός πολύ απλού εξυπηρετητή σε γλώσσα C για Unix σύστημα.

```
/*
 * client.c - A very simple client.
 *
 * %gcc -Wall -Wextra -o client client.c
 * %./client 127.0.0.1
 */


#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
```

```
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>


int
main(int argc, char *argv[])
{
      int sockfd = 0, n = 0;
      char recvBuff[1024];
      struct sockaddr_in serv_addr;

      if (2 != argc) {
            fprintf(stderr, "\nUsage: %s <ip of server>\n", argv[0]);

            return 1;
      }

      memset(recvBuff, 0x00, sizeof(recvBuff));
      if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
            fprintf(stderr, "\nError : Could not create socket\n");

            return 1;
      }

      memset(&serv_addr, 0x00, sizeof(serv_addr));

      serv_addr.sin_family = AF_INET;
      serv_addr.sin_port = htons(5000);

      if (inet_pton(AF_INET, argv[1], &serv_addr.sin_addr) <= 0) {
            fprintf(stderr, "\ninet_pton error occured\n");

            return 1;
      }

      if (connect(sockfd, (struct sockaddr *) &serv_addr,
                  sizeof(serv_addr)) < 0) {
            fprintf(stderr, "\nError : Connect Failed\n");
```

```
            return 1;
      }

      while ((n = read(sockfd, recvBuff, sizeof(recvBuff) - 1)) > 0) {
            recvBuff[n] = 0;
            if (EOF == fputs(recvBuff, stdout))
                  fprintf(stderr, "\nError : Fputs error\n");
      }

      if (n < 0)
            fprintf(stderr, "\n Read error \n");

      return 0;
}
```

Υλοποίηση ενός πολύ απλού πελάτη του άνωθεν εξυπηρετητή σε γλώσσα C για Unix σύστημα.

3.12.2 Μοντέλο Δράστη (Actor Model)

Το μοντέλο Δράστη είναι ένα μαθηματικό μοντέλο περιγραφής ταυτόχρονων υπολογισμών. Ως δομικά στοιχεία έχει τους δράστες (οντότητες με κατάσταση) και τα μηνύματα με τα οποία αλληλεπιδρούν. Ως μοντέλο περιγραφής έχει αρκετά κοινά στοιχεία με τους δράστες (agents) του τομέα της Τεχνητής Νοημοσύνης. Ένας δράστης, αναλόγως το μήνυμα που λαμβάνει παίρνει αποφάσεις (που επηρεάζουν την κατάστασή του), μπορεί να δημιουργήσει εκ νέου άλλους δράστες, να στείλει νέα μηνύματα και να διαμορφώσει κάποιο σύνολο παραμέτρων που θα συμβάλλουν στην απόφαση για το επόμενο μήνυμα που θα λάβει. Η φιλοσοφία που διέπει αυτό το μοντέλο είναι ότι τα πάντα είναι δράστες (κατ' αναλογία με το ότι τα πάντα είναι αντικείμενα που διέπει τον αντικειμενοστρεφή προγραμματισμό). Υπάρχουν πολλοί δράστες που ταυτόχρονα διεκπεραιώνουν εργασίες, οι οποίοι είναι όσο το δυνατόν απομονωμένοι μεταξύ τους (δε μοιράζονται δεδομένα, οπότε και δε χρειάζονται μηχανισμοί συγχρονισμού όπως τα κλειδώματα). Με επικοινωνία μέσω μηνυμάτων συμβάλλουν στη γενικότερη πρόοδο της επίλυσης του προβλήματος.

Περιπτώσεις ανταγωνισμού και αδιέξοδα αποφεύγονται λόγω της σχεδίασης του συγκεκριμένου μοντέλου.

Όπως αναφέραμε παραπάνω, η απόκριση ενός δράστη στη λήψη ενός μηνύματος μπορεί να είναι η εν δυνάμει ταυτόχρονη:[3]

- αποστολή διακριτού αριθμού μηνυμάτων σε άλλους δράστες
- δημιουργία διακριτού αριθμού νέων δραστών
- διαμόρφωση της εσωτερικής του κατάστασης που θα καθορίσει την απόκρισή του στη λήψη επόμενων μηνυμάτων.

Δεν υπάρχει εγγυημένη σειρά όσον αφορά την παράλληλη εκτέλεση των παραπάνω. Επίσης, δεν υπάρχει εγγύηση όσον αφορά τη διατεταγμένη σειρά κατά την οποία καταφθάνουν τα μηνύματα σε κάποιο δράστη, δηλαδή η σειρά που θα καταφθάσουν μηνύματα ταυτόχρονα είναι ακαθόριστη και δεν αποτελεί οδηγό για τη σειρά με την οποία αυτά θα επεξεργαστούν από το δράστη. Για την καταχώρηση ληφθέντων μηνυμάτων, κάθε δράστης έχει μια δομή που μοιάζει σε "ταχυδρομικό κουτί" (mailbox) το οποίο είναι συνυφασμένο με μια ταχυδρομική διεύθυνση (mail address). Με αυτόν τον τρόπο οι δράστες είναι σε θέση να γνωρίζουν τον προορισμό κάθε μηνύματος προς αποστολή (επικοινωνούν δηλαδή μόνο με τους δράστες των οποίων γνωρίζουν τη διεύθυνση) και να προβαίνουν στην επεξεργασία των ληφθέντων μηνυμάτων.

Ο μη ντετερμινισμός στη σειρά των ταυτόχρονων γεγονότων που συνθέτουν την απόκριση ενός δράστη σε κάποιο μήνυμα και οι παρενέργειες όσον αφορά την κατάστασή του κάνουν την περιγραφή του μοντέλου με το λ-λογισμό αδύνατη.

```
%% echo.erl - A simple echo server.

-module(echo).
-export([start/0]).
```

```
loop() -> receive
    {Sender, Num} ->
        Sender ! Num,
        loop()
    end.

start() -> spawn(fun loop/0).
```

Απλοϊκή υλοποίηση μοντέλου Δράστη με τη γλώσσα Erlang. Κάθε αίτηση εξυπηρετείται από ένα δράστη.

3.12.3 Σύστημα MapReduce

Το σύστημα MapReduce χρησιμοποιείται κυρίως για την επεξεργασία μεγάλων συνόλων δεδομένων (datasets) με κατανεμημένο παράλληλο αλγόριθμο σε ένα σύμπλεγμα (cluster). Είναι ένα ολοκληρωμένο σύστημα με την έννοια ότι προβλέπει την υλοποίηση σε αρκετά επίπεδα, από την εγκατάσταση και την παραμετροποίησή της μέχρι τις παρεχόμενες λειτουργίες από το προγραμματιστικό του πλαίσιο. Έχει ως βάση του τον αντίστοιχο Αλγοριθμικό Σκελετό (Map-Reduce).

Προγραμματιστικά εκθέτει τρόπο υλοποίησης συναρτήσεων για απεικόνιση (map) και αναγωγή (reduce). Όλα τα δεδομένα συνδέονται με αναγνωριστικά (κλειδιά). Ο υπολογισμός αναλύεται στα παρακάτω 5 βήματα.

1) *Προετοιμασία εισόδου απεικόνισης*. Τα δεδομένα οργανώνονται βάσει των κλειδιών τους και προετοιμάζονται για αποστολή στους αντίστοιχους επεξεργαστές, ώστε να λάβει χώρα το στάδιο της απεικόνισης.

2) *Εκτέλεση της απεικόνισης*. Ο προγραμματιστής έχει υλοποιήσει τη συνάρτηση απεικόνισης και κάθε επεξεργαστής την εκτελεί για τα δεδομένα που λαμβάνει παράγοντας νέα δεδομένα με νέα κλειδιά. Η συνάρτηση εκτελείται μία φορά για κάθε δεδομένο εισόδου και είναι της μορφής $Map(K_1, V_1) \rightarrow [K_2, V_2]$, όπου K_1 το αρχικό κλειδί, V_1 το δεδομένο εισόδου, K_2 το νέο κλειδί, V_2 το δεδομένο εξόδου. Δηλαδή για κάθε

ζεύγος $(\textit{κλειδί}, \textit{δεδομένο})$ εισόδου παράγεται μια λίστα ζευγών $(\textit{κλειδί}, \textit{δεδομένο})$ εξόδου.

3) <u>*Ανάμιξη αποτελεσμάτων απεικόνισης*</u>. Η έξοδος του προηγούμενου βήματος οργανώνεται βάσει των νέων κλειδιών και οδηγείται στους επεξεργαστές που είναι επιφορτισμένοι με την εκτέλεση του βήματος της αναγωγής.

4) <u>*Εκτέλεση της αναγωγής*</u>. Ο προγραμματιστής έχει υλοποιήσει τη συνάρτηση αναγωγής και κάθε επεξεργαστής την εκτελεί για τα ομαδοποιημένα δεδομένα της εισόδου του. Η συνάρτηση εκτελείται μία φορά για κάθε δεδομένο εισόδου και είναι της μορφής $Reduce(K_2,[V_2]) \rightarrow [V_3]$, όπου K_2 το νέο κλειδί, $[V_2]$ η λίστα (ομάδα) που συνδέεται με το κλειδί K_2, $[V_3]$ η λίστα που περιέχει το αποτέλεσμα της αποτίμησης (μπορεί να είναι είτε η κενή λίστα, είτε να έχει ένα στοιχείο είτε και περισσότερα).

5) <u>*Παραγωγή αποτελέσματος*</u>. Το σύστημα συλλέγει τα αποτελέσματα του παραπάνω βήματος και βάσει των νέων κλειδιών τα ταξινομεί, ώστε να υπολογιστεί το τελικό αποτέλεσμα.

Κατά την παραπάνω διαδικασία, η αποστολή και λήψη των δεδομένων δύναται να γίνει από διαφορετικά υπο-συστήματα μέσω δικτύου.

Μια εξαιρετικά δημοφιλής υλοποίησή του είναι το Apache Hadoop του μη-κερδοσκοπικού οργανισμού Apache Software Foundation.

3.12.4 Πρότυπο POSIX Threads

Τα POSIX Threads είναι μέρος του προτύπου συμβατότητας λειτουργικών συστημάτων POSIX. Στόχος είναι η προτυποποίηση ενός προγραμματιστικού πλαισίου για πολυνηματικό προγραμματισμό. Το πρότυπο αναφέρεται στη γλώσσα προγραμματισμού C, αλλά υπάρχουν υλοποιήσεις και για άλλες γλώσσες που στην ουσία περιτυλίγουν την υλοποίηση της C (wrappers). Είναι ένας "πρωτόγονος" τρόπος

πολυνηματικού προγραμματισμού για προγραμματισμό συστημάτων (systems programming), μιας και ο προγραμματιστής έχει πλήρη έλεγχο της εκτέλεσης. Έχουν υιοθετηθεί σε πληθώρα λειτουργικών συστημάτων και πολλές εφαρμογές τα χρησιμοποιούν. Συνδυαζόμενα με TCP/IP sockets αποτελούν την πιο βασική πλατφόρμα εργαλείων για παράλληλο προγραμματισμό.

Τα νήματα μιας διεργασίας δρουν σε κοινό χώρο διευθύνσεων μνήμης. Αποτελεί ευθύνη του προγραμματιστή η διαχείριση του συγχρονισμού και των κρίσιμων τμημάτων. Οι διαδικασίες - συναρτήσεις που εκτίθενται στον προγραμματιστή διαιρούνται σε 4 ομάδες που αφορούν[8]

α) τη διαχείριση νημάτων (δημιουργία, ένωση με το κύριο νήμα, κτλ)

β) τα κλειδώματα αμοιβαίου αποκλεισμού (mutexes)

γ) τις μεταβλητές συνθηκών (conditional variables) και

δ) δομικά στοιχεία συγχρονισμού, όπως κλειδώματα αναγνωστών - εγγραφέων (read / write locks) και φράγματα συγχρονισμού (barriers).

Οι σημαφόροι (semaphores) περιλαμβάνονται ως επεκτάσεις στο πρότυπο (δεν είναι μέρος δηλαδή του βασικού πολυνηματικού προτύπου).

```
/*
 * pth-dining_philosophers.c - Dining philosophers (pthreads)
 *   Forks are picked up atomically.
 *
 * %gcc -Wall -Wextra -o pth-dining_philosophers pth-
dining_philosophers.c \
 *   -pthread
 * %./pth-dining_philosophers
 */


#include <sys/time.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <errno.h>
```

```
#define NUMP     5


pthread_mutex_t fork_mutex[NUMP];
pthread_mutex_t eat_mutex;


void  *diner(void *arg);


void *
diner(void *arg)
{
      int *i;
      int v;
      int eating = 0;

      i = (int *) arg;

      printf("I'm diner %d\n", *i);
      v = *i;

      while (eating < NUMP) {
            printf("%d is thinking\n", v);
            sleep(v / 2);
            printf("%d is hungry\n", v);
            pthread_mutex_lock(&eat_mutex);
            pthread_mutex_lock(&fork_mutex[v]);
            pthread_mutex_lock(&fork_mutex[(v + 1) % NUMP]);
            pthread_mutex_unlock(&eat_mutex);
            printf("%d is eating\n", v);
            eating++;
            sleep(1);
            printf("%d is done eating\n", v);
            pthread_mutex_unlock(&fork_mutex[v]);
            pthread_mutex_unlock(&fork_mutex[(v + 1) % NUMP]);
      }
      pthread_exit(NULL);

      return NULL;
}

int
main(void)
```

```
{
      int i;
      pthread_t diner_thread[NUMP];
      int dn[NUMP];

      pthread_mutex_init(&eat_mutex, NULL);
      for (i = 0; i < NUMP; i++)
            pthread_mutex_init(&fork_mutex[i], NULL);

      for (i = 0; i < NUMP; i++) {
            dn[i] = i;
            pthread_create(&diner_thread[i], NULL, diner, &dn[i]);
      }

      for (i = 0; i < NUMP; i++)
            pthread_join(diner_thread[i], NULL);

      pthread_exit(0);

      return 0;
}
```

Υλοποίηση του προβλήματος των συνδαιτυμόνων φιλοσόφων με χρήση POSIX Threads.

3.12.5 Προγραμματιστικό πλαίσιο OpenMP

Το OpenMP είναι ένα προγραμματιστικό πλαίσιο πολυεπεξεργασίας για αρχιτεκτονικές κοινής μνήμης (κυρίως) με υλοποίηση για τις γλώσσες C, C++ και Fortran. Η διαχείριση γίνεται από το μη κερδοσκοπικό οργανισμό OpenMP Architecture Review Board (OpenMP ARB).

Το OpenMP βασίζεται σε οδηγίες (directives) προς το μεταγλωττιστή (που περνώντας από έναν προ-επεξεργαστή (π.χ, C pre-processor με οδηγίες `#pragma`) μετατρέπονται σε εντολές) και μπορεί να αξιοποιηθεί για την εύκολη παραλληλοποίηση ήδη υπάρχοντος σειριακού κώδικα. Αναλαμβάνει τη διαχείριση και υλοποίηση των στοιχείων που εκφράζουν τον παραλληλισμό (νήματα, συγχρονισμός), χωρίς επιπλέον εργασία από την πλευρά του προγραμματιστή (παρά μόνον των οδηγιών που εκφράζουν το

μοτίβο της παραλληλοποίησης). Αυτό αποτελεί βασικό πλεονέκτημα του OpenMP: είναι αρκετά εύκολο για κάποιον μη-προγραμματιστή ή με περιορισμένες γνώσεις πάνω στις λεπτομέρειες των παράλληλων και ταυτόχρονων συστημάτων να εκφράσει προφανείς παράλληλους υπολογισμούς σε ένα πρόγραμμα.

```
/*
 * omp-reduction.c - OpenMP example program which computes the
 *   dot product of two arrays a and b (that is sum(a[i]*b[i]))
 *using
 *   a sum reduction.
 *
 * %gcc -Wall -Wextra -fopenmp -o omp-reduction omp-reduction.c
 * %./omp-reduction
 */


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


#define N 1000


int
main(void)
{
      double a[N];
      double sum = 0.0;
      int i, n, tid;

      /* Start a number of threads */
#pragma omp parallel shared(a) private(i)
      {
            tid = omp_get_thread_num();

            /* Only one of the threads do this */
#pragma omp single
            {
                  n = omp_get_num_threads();
                  printf("Number of threads = %d\n", n);
```

```
            }

            /* Initialize a */
#pragma omp for
            for (i = 0; i < N; i++) {
                  a[i] = 1.0;
            }

            /* Parallel for loop computing the sum of a[i] */
#pragma omp for reduction(+:sum)
            for (i = 0; i < N; i++) {
                  sum = sum + (a[i]);
            }

      } /* End of parallel region */

      printf("   Sum = %2.1f\n", sum);

      return 0;
}
```

Παράδειγμα υπολογισμού εσωτερικού γινομένου με χρήση OpenMP σε γλώσσα C.

3.12.6 Προγραμματιστικό πλαίσιο Intel TBB (Intel Threading Building Blocks)

Πρόκειται για ένα υψηλού επιπέδου προγραμματιστικό πλαίσιο πολυπύρηνων επεξεργαστών σε γλώσσα C++. Αναπτύχθηκε από την εταιρεία Intel και εκθέτει στον προγραμματιστή δομές δεδομένων και αλγόριθμους για την ευκολότερη υλοποίηση Αλγοριθμικών Σκελετών, χωρίς να χρειάζεται αρκετή εργασία για τη μικρο-διαχείριση (micro-management) των λεπτομερειών της υλοποίησης. Ο προγραμματιστής αρχικοποιεί το περιβάλλον του πλαισίου, εκφράζει τη λύση του προβλήματος με εργασίες και η υλοποίηση του πλαισίου αναλαμβάνει την κατανομή των εργασιών σε επεξεργαστές, τη χρονοδρομολόγηση των εργασιών, την κατασκευή του γράφου προήγησης για τις εξαρτήσεις και την εκτέλεση του Αλγοριθμικού Σκελετού. Χρησιμοποιούνται εκτενώς τα εκμαγεία (templates) της C++ για την έκφραση γενικού προγραμματισμού (generic programming).

```
/*
 * tbb-example.cpp - A simple TBB example.
 *
 * %g++ -Wall -Wextra -o tbb-example tbb-example.cpp -ltbb
 * %./tbb-example
 */


#include <tbb/blocked_range.h>
#include <tbb/parallel_for.h>
#include <tbb/task_scheduler_init.h>
#include <iostream>
#include <vector>


using namespace std;


struct mytask {
      mytask(size_t n)
            : _n(n)
      { }

      void operator()() {
            for (int i = 0; i < 1000000; ++i) { } // Deliberately run
slow
            cerr << "[" << _n << "]";
      }

      size_t _n;
};

struct executor {
      executor(vector<mytask>& t)
            : _tasks(t)
      { }

      executor(executor &e, tbb::split)
            : _tasks(e._tasks)
      { }

      void operator()(const tbb::blocked_range<size_t>& r) const {
```

```
            for (size_t i = r.begin(); i != r.end(); ++i)
                  _tasks[i]();
      }

      vector<mytask>& _tasks;
};

int
main(void)
{
      tbb::task_scheduler_init init;  // Automatic number of threads
      // tbb::task_scheduler_init init(2);  // Explicit number of
threads

      vector<mytask> tasks;
      for (int i = 0; i < 1000; ++i)
            tasks.push_back(mytask(i));

      executor exec(tasks);
      tbb::parallel_for(tbb::blocked_range<size_t>(0, tasks.size()),
exec);
      cerr << endl;

      return 0;
}
```

Παράδειγμα χρήσης του πλαισίου Intel TBB σε γλώσσα C++.

3.12.7 Πρότυπο MPI (Message Passing Interface)

Το πρότυπο MPI είναι ένα πρωτόκολλο επικοινωνίας για παράλληλη επεξεργασία με ανταλλαγή μηνυμάτων (message passing) σε κατανεμημένα συστήματα. Αναπτύχθηκε από ομάδες ερευνητών του ακαδημαϊκού και βιομηχανικού χώρου. Ως προγραμματιστικό μοντέλο αποκρύπτει αρκετές από τις λεπτομέρειες υλοποίησης της επικοινωνίας και διευκολύνει τον προγραμματισμό κατά το μοντέλο SPMD. Εκτός από τις διαδεδομένες υλοποιήσεις για τις γλώσσες C και Fortran υπάρχουν υλοποιήσεις και για άλλες γλώσσες (Java, Python, κτλ). Για τη λειτουργία του απαιτείται η εγκαθίδρυση ενός συμπλέγματος MPI (MPI cluster). Οι κόμβοι του συμπλέγματος εκτελούν παράλληλα το πρόγραμμα.

Κάθε κόμβος του συστήματος μοντελοποιείται με μια διεργασία που δρα στην τοπική μνήμη που διαθέτει. Η ανταλλαγή δεδομένων με άλλες διεργασίες γίνεται με ανταλλαγή μηνυμάτων. Όλες οι διεργασίες εκτελούν το ίδιο πρόγραμμα. Κάθε διεργασία επεξεργάζεται υποσύνολο των δεδομένων ή διαφοροποιεί τη ροή εκτέλεσής της με βάση ένα σύμβολο, το βαθμό (rank) στην τοπολογία, που της αποδίδει το MPI. Ανταλλαγή μηνυμάτων με άλλες διεργασίες μπορεί να γίνει είτε σύγχρονα (blocking) είτε ασύγχρονα (non-blocking). Κατά την ανταλλαγή δεδομένων χρησιμοποιούνται είτε προκαθορισμένοι τύποι δεδομένων (datatypes) από το MPI (π.χ. MPI_INT για ακέραιους αριθμούς) είτε σύνθετοι τύποι δεδομένων που όμως αποτελούνται από τους παραπάνω προκαθορισμένους τύπους. Η επικοινωνία μπορεί να είναι είτε από μια διεργασία προς μια άλλη (point-to-point) είτε συλλογική (collective) όπου μια διεργασία αποστέλλει ή λαμβάνει μηνύματα από όλες τις διεργασίες που είναι οργανωμένες σε κάποια ομάδα (group). Ομάδα μπορεί να θεωρηθούν και όλες οι διεργασίες του συμπλέγματος. Συνήθως εκλέγεται μια διεργασία ως κύριος του συμπλέγματος (master ή root) και ενορχηστρώνει τη ροή εκτέλεσης. Γενικά, καλή τακτική είναι η επιδίωξη όσο το δυνατόν συμμετρικού κώδικα (όλες οι διεργασίες να εκτελούν τον ίδιο κώδικα με όσο το δυνατό λιγότερες αποκλίσεις - διακλαδώσεις αναλόγως το αναγνωριστικό τους), προς αποφυγή μεγάλης πολυπλοκότητας στον κώδικα και βέλτιστης χρήσης των πόρων του συστήματος.[18]

Νεότερη έκδοση του προτύπου MPI είναι το MPI-2, το οποίο αποτελεί επέκταση του MPI-1 που εξετάσαμε παραπάνω, εισάγοντας παράλληλη είσοδο-έξοδο (I/O), δυναμική διαχείριση διεργασιών και διαχείριση κατανεμημένης μνήμης με λειτουργίες απομακρυσμένης μνήμης. Προς το παρόν δεν έχει τύχει της ευρείας αποδοχής που έχει το MPI-1.

```
/*
 * mpi-pi.c - Simple pi calculation using MPI.
 *
```

```
 * %mpicc -Wall -Wextra -o mpi-pi mpi-pi.c
 * %mpirun -np 4 ./mpi-pi
 */


#include <mpi.h>
#include <stdio.h>
#include <math.h>


int
main(int argc, char *argv[])
{
      int n, myid, numprocs, i;
      double PI25DT = 3.14159265358979323846264643;
      double mypi, pi, h, sum, x;

      MPI_Init(&argc, &argv);
      MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
      MPI_Comm_rank(MPI_COMM_WORLD, &myid);

      for (;;) {
            if (0 == myid) {
                  printf("Enter the number of intervals (0 quits): ");
                  scanf("%d", &n);
            }

            MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

            if (0 == n)
                  break;
            else {
                  h   = 1.0 / (double) n;
                  sum = 0.0;
                  for (i = myid + 1; i <= n; i += numprocs) {
                        x = h * ((double)i - 0.5);
                        sum += (4.0 / (1.0 + x * x));
                  }
                  mypi = h * sum;

                  MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
                             MPI_COMM_WORLD);

                  if (0 == myid)
```

```
                        printf("pi is approximately %.16f, "
                               "Error is %.16f\n",
                               pi, fabs(pi - PI25DT));
            }
        }

        MPI_Finalize();

        return 0;
}
```

Υπολογισμός του αριθμού π με χρήση του MPI σε γλώσσα C.

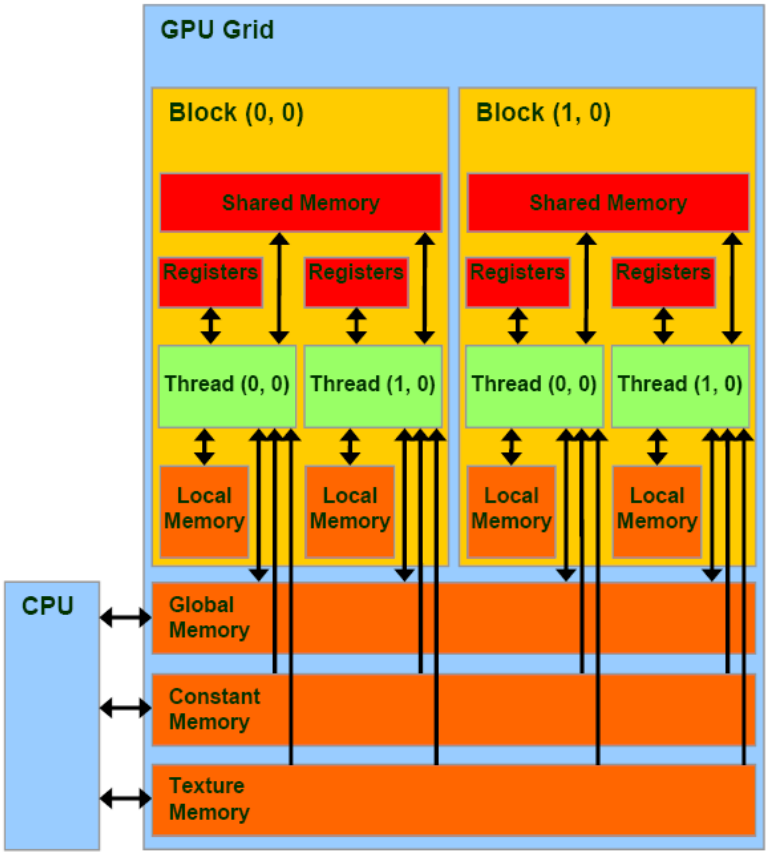
3.12.8 Πρότυπο CUDA (Compute Unified Device Architecture)

Το προγραμματιστικό μοντέλο αυτό δημιουργήθηκε από την εταιρεία NVIDIA και αποτελεί πλατφόρμα για ανάπτυξη εφαρμογών που εκμεταλλεύονται το μαζικό παραλληλισμό. Κύριο μειονέκτημά του είναι ότι είναι συμβατό μόνο με επεξεργαστές (συνήθως επεξεργαστές γραφικών) της ανωτέρω εταιρείας. Ταιριάζει σε προβλήματα που επωφελούνται από αρχιτεκτονικές τύπου SIMD. Ως μοντέλο, στηρίζεται στη μαζικά παράλληλη εκτέλεση νημάτων (threads) οργανωμένα όσον αφορά τη δρομολόγηση σε ομάδες εκτέλεσης (warps). Αντίθετα με μια "παραδοσιακή" Κεντρική Μονάδα Επεξεργασίας που ευνοεί τη σειριακή εκτέλεση λίγων νημάτων στα οποία αναθέτουμε διαφορετικές εργασίες, οπότε και η εκτέλεση μπορεί να διακλαδίζεται αναλόγως την εκάστοτε εργασία ή φορτίο, στο μοντέλο CUDA όσο λιγότερες διακλαδώσεις υπάρχουν και όσο περισσότερη ομοιογένεια στα δεδομένα έχουμε, τόσο καλύτερη επίδοση αναμένουμε να εξασφαλίσουμε. Από τα παραπάνω, εύκολα συνάγουμε ότι τέτοιου είδους μηχανές υπολογισμού δεν ταιριάζουν για υλοποιήσεις εργασιών γενικού σκοπού (όπως π.χ. κάνει ένα λειτουργικό σύστημα), αλλά περισσότερο είναι για να συνεργάζονται με μονάδες επεξεργασίας γενικού σκοπού (όπως η Κεντρική Μονάδα ενός υπολογιστικού συστήματος). Αυτό αποτελεί ένα ετερογενές υπολογιστικό μοντέλο.

Προγραμματίζοντας σε CUDA συνδυάζουμε τον προγραμματισμό της

Κεντρικής Μονάδας Επεξεργασίας με ένα ειδικό κύκλωμα επεξεργασίας ρεύματος δεδομένων, το οποίο συνήθως είναι ένας μοντέρνος επεξεργαστής γραφικών (Graphical Processing Unit - GPU). Διαιρούμε το πρόβλημα σε σειριακές ή ακολουθιακές ενότητες που θα εκτελεστούν στη CPU (host code) και ενότητες που παραλληλοποιούνται και θα εκτελεστούν στην GPU (device code) οι οποίες καλούνται πυρήνες εκτέλεσης (kernels). Όταν ξεκινά ένας πυρήνας, τα νήματα που τον αποτελούν οργανώνονται χωρικά σε ένα πλέγμα (grid) 1, 2 ή 3 διαστάσεων. Κάθε block του πλέγματος είναι οργανωμένο σε υπο-χωρία 1, 2 ή 3 διαστάσεων τα οποία και αντιστοιχούν στα νήματα εκτέλεσης. Η επικοινωνία γίνεται μέσω ενός διαύλου (συνήθως PCI Express) που συνδέει την κύρια μνήμη (RAM) της CPU και την καθολική μνήμη (global memory) της GPU. Σε κάθε κύκλο όλα τα νήματα εκτελούν την ίδια εντολή. Έτσι, εκτελούνται όλες οι διακλαδώσεις ελέγχου και οι μη έγκυρες ματαιώνονται στο τέλος.

Το μοντέλο μνήμης ενός επεξεργαστή CUDA περιλαμβάνει την καθολική μνήμη όπου όλα τα νήματα εκτέλεσης έχουν πρόσβαση ανάγνωσης και εγγραφής, τη σταθερή μνήμη (constant memory) όπου όλα τα νήματα έχουν μόνο πρόσβαση ανάγνωσης, την κοινή μνήμη (shared memory) για κάθε block, όπου τα νήματα που το αποτελούν έχουν πρόσβαση ανάγνωσης και εγγραφής, την τοπική μνήμη (local memory) για κάθε νήμα και τους καταχωρητές (registers) κάθε νήματος. Επίσης υπάρχει και η μνήμη υφής (texture memory) που είναι μόνο προς ανάγνωση από τα νήματα και χρησιμοποιείται όταν η τοπική χωρικότητα (spatial locality) είναι απαραίτητη.[11]



Το μοντέλο μνήμης ενός επεξεργαστή CUDA.

Από τα παραπάνω βλέπουμε πως το CUDA είναι μια μορφή προγραμματισμού SPMD αρχιτεκτονικής (για την ακρίβεια καλείται SIMT - Single Intruction Multiple Threads), όπου όλα τα νήματα εκτελούν τον ίδιο κώδικα έχοντας το καθένα το δικό του σύνολο δεδομένων. Το αναγνωριστικό κάθε νήματος, το οποίο είναι ένα διάνυσμα που δείχνει τη θέση του νήματος στο πλέγμα που εκκίνησε ο πυρήνας, χρησιμοποιείται για να διαφοροποιήσει την εκτέλεσή του. Οι πυρήνες εκτέλεσης γράφονται μαζί με το κύριο πρόγραμμα. Συνήθως χρησιμοποιείται η γλώσσα προγραμματισμού C με κάποιες προσθήκες για το CUDA (τα αρχεία πηγαίου κώδικα συνηθίζεται να έχουν την κατάληξη .cu σε αντίθεση με την παραδοσιακή κατάληξη .c που χρησιμοποιούμε για αρχεία πηγαίου κώδικα της C). Το πρόγραμμα περνά από ένα ειδικό μεταγλωττιστή (compiler), ο οποίος χωρίζει τον κώδικα που θα εκτελεστεί στη CPU από αυτόν που θα εκτελεστεί στην GPU. Ο πρώτος μεταγλωττίζεται με έναν "παραδοσιακό" μεταγλωττιστή, ενώ ο δεύτερος με το μεταγλωττιστή του CUDA. Υπάρχουν διαπροσωπείες και για αρκετές

άλλες γλώσσες προγραμματισμού (εκτός της C). Ένα CUDA πρόγραμμα ξεκινά με τον κώδικα που εκτελείται στη CPU να αντιγράφει τα δεδομένα του παράλληλου υπολογισμού από τη RAM στην global memory της CUDA συσκευής. Στη συνέχεια εκτελείται ο υπολογισμός (μέσω του πηρήνα εκτέλεσης) και αφού όλα τα νήματα έχουν τερματίσει (κάτι που ελέγχεται με εντολές συγχρονισμού), τα δεδομένα μεταφέρονται στη CPU από την global memory για περαιτέρω επεξεργασία στη CPU. Εδώ χρήζει προσοχής το γεγονός του κόστους της επικοινωνίας κατά την εκκίνηση και τον τερματισμό του πυρήνα εκτέλεσης, το οποίο πρέπει να λαμβάνεται υπόψη κατά την υλοποίηση. Τέλος, πολλοί πυρήνες μπορούν να εκτελεστούν παράλληλα με τα κανάλια εκτέλεσης CUDA (CUDA streams).[11]

```
/*
 * cuda-vecadd.cu - Vector addition (CUDA) example.
 *
 * %nvcc -o cuda-vecadd cuda-vecadd.cu
 * %./cuda-vecadd
 */


#include <stdio.h>
#include <stdlib.h>
#include <math.h>


// CUDA kernel. Each thread takes care of one element of c
__global__ void
vecAdd(double *a, double *b, double *c, int n)
{
      // Get our global thread ID
      int id = blockIdx.x * blockDim.x + threadIdx.x;

      // Make sure we do not go out of bounds
      if (id < n)
            c[id] = a[id] + b[id];
}

int
main(void)
{
```

```
// Size of vectors
int n = 100000;

// Host input vectors
double *h_a;
double *h_b;
//Host output vector
double *h_c;

// Device input vectors
double *d_a;
double *d_b;
//Device output vector
double *d_c;

// Size, in bytes, of each vector
size_t bytes = n*sizeof(double);

// Allocate memory for each vector on host
h_a = (double*) malloc(bytes);
h_b = (double*) malloc(bytes);
h_c = (double*) malloc(bytes);

// Allocate memory for each vector on GPU
cudaMalloc(&d_a, bytes);
cudaMalloc(&d_b, bytes);
cudaMalloc(&d_c, bytes);

int i;
// Initialize vectors on host
for( i = 0; i < n; i++ ) {
      h_a[i] = sin(i) * sin(i);
      h_b[i] = cos(i) * cos(i);
}

// Copy host vectors to device
cudaMemcpy(d_a, h_a, bytes, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, h_b, bytes, cudaMemcpyHostToDevice);

int blockSize, gridSize;

// Number of threads in each thread block
blockSize = 1024;
```

```
    // Number of thread blocks in grid
    gridSize = (int)ceil((float)n/blockSize);

    // Execute the kernel
    vecAdd<<<gridSize, blockSize>>>(d_a, d_b, d_c, n);

    // Copy array back to host
    cudaMemcpy(h_c, d_c, bytes, cudaMemcpyDeviceToHost);

    // Sum up vector c and print result divided by n,
    //   this should equal 1 within error
    double sum = 0;
    for(i=0; i<n; i++)
        sum += h_c[i];
    printf("final result: %f\n", sum/n);

    // Release device memory
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    // Release host memory
    free(h_a);
    free(h_b);
    free(h_c);

    return 0;
}
```

Παράδειγμα πρόσθεσης διανυσμάτων με CUDA.

3.12.9 Πρότυπο OpenCL (Open Computing Language)

Το πρότυπο OpenCL αρχικά αναπτύχθηκε από την εταιρεία Apple. Στη συνέχεια, η διαχείριση και η περαιτέρω ανάπτυξή του παραδόθηκε στο μη κερδοσκοπικό οργανισμό Khronos Group (υπεύθυνο και για τη διαχείριση του προτύπου OpenGL). Περιγράφει ένα προγραμματιστικό πλαίσιο για το συντονισμό παράλληλων υπολογισμών σε ετερογενείς επεξεργαστές και μια ανεξαρτήτου πλατφόρμας γλώσσα προγραμματισμού (πολύ κοντά στη

γλώσσα C) για τον προγραμματισμό των πυρήνων υπολογισμού (OpenCL kernels).

Σε γενικές γραμμές είναι αρκετά παρεμφερές ως πρότυπο με το CUDA. Κύρια διαφορά με το CUDA είναι ότι οι πυρήνες εκτέλεσης του OpenCL μπορούν να εκτελεστούν όχι μόνο σε επεξεργαστές γραφικών NVIDIA, αλλά σε οποιοδήποτε επεξεργαστή για τον οποίο υπάρχει πρόγραμμα οδήγησης (driver) για OpenCL, είτε είναι επεξεργαστής γραφικών είτε όχι - για παράδειγμα, σε CPU, σε επεξεργαστή ψηφιακού σήματος (DSP), κ.ά. Εξάλλου, το πρότυπο OpenCL ορίζει διάφορα προφίλ που σχετίζονται με την πλατφόρμα εκτέλεσης (π.χ. φορητές συσκευές, ενσωματωμένα συστήματα), ώστε να γίνεται εφικτή μια αποδοτική υλοποίηση του προτύπου σε πλατφόρμες με διαφορετικά τεχνικά χαρακτηριστικά και δυνατότητες.

Το μοντέλο μνήμης του είναι παρεμφερές με το αντίστοιχο του CUDA. Τα παραπάνω έχουν ως αποτέλεσμα κατά τον προγραμματισμό να χρησιμοποιούνται τα ίδια προγραμματιστικά μοτίβα και να υπάρχουν οι ίδιοι περιορισμοί. Δε θα επεκταθούμε περισσότερο, μιας και οι διαφορές με το CUDA δεν είναι μεγάλες όσον αφορά τη φιλοσοφία που ακολουθείται κατά τον προγραμματισμό και δεν αποτελεί σκοπό του παρόντος κεφαλαίου η ανάλυση σε βάθος των λεπτομερειών ενός προγραμματιστικού μοντέλου.

```
/*
 * ocl-vecadd.c - Vector addition (OpenCL) example.
 *
 * %gcc -Wall -Wextra -o ocl-vecadd ocl-vecadd.c -lOpenCL
 * %./ocl-vecadd
 */


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <CL/opencl.h>
```

```
// OpenCL kernel. Each work item takes care of one element of c
const char *kernelSource =                                      "\n"
           "#pragma OPENCL EXTENSION cl_khr_fp64 : enable   \n"
           "__kernel void vecAdd(__global double *a,        \n"
           "                     __global double *b,        \n"
           "                     __global double *c,        \n"
           "                       const unsigned int n)    \n"
           "{                                               \n"
           "    //Get our global thread ID                  \n"
           "    int id = get_global_id(0);                  \n"
           "                                                \n"
           "    //Make sure we do not go out of bounds      \n"
           "    if (id < n)                                 \n"
           "        c[id] = a[id] + b[id];                  \n"
           "}                                               \n"
           "\n";


int
main(void)
{
      // Length of vectors
      unsigned int n = 100000;

      // Host input vectors
      double *h_a;
      double *h_b;
      // Host output vector
      double *h_c;

      // Device input buffers
      cl_mem d_a;
      cl_mem d_b;
      // Device output buffer
      cl_mem d_c;

      cl_platform_id cpPlatform;        // OpenCL platform
      cl_device_id device_id;           // device ID
      cl_context context;               // context
      cl_command_queue queue;           // command queue
      cl_program program;               // program
      cl_kernel kernel;                 // kernel
```

```
	// Size, in bytes, of each vector
	size_t bytes = n * sizeof(double);

	// Allocate memory for each vector on host
	h_a = (double *) malloc(bytes);
	h_b = (double *) malloc(bytes);
	h_c = (double *) malloc(bytes);

	// Initialize vectors on host
	int i;
	for (i = 0; i < n; i++) {
		h_a[i] = sinf(i) * sinf(i);
		h_b[i] = cosf(i) * cosf(i);
	}

	size_t globalSize, localSize;
	cl_int err;

	// Number of work items in each local work group
	localSize = 64;

	// Number of total work items - localSize must be devisor
	globalSize = ceil(n / (float)localSize) * localSize;

	// Bind to platform
	err = clGetPlatformIDs(1, &cpPlatform, NULL);

	// Get ID for the device
	err = clGetDeviceIDs(cpPlatform, CL_DEVICE_TYPE_GPU, 1,
&device_id, NULL);

	// Create a context
	context = clCreateContext(0, 1, &device_id, NULL, NULL, &err);

	// Create a command queue
	queue = clCreateCommandQueue(context, device_id, 0, &err);

	// Create the compute program from the source buffer
	program = clCreateProgramWithSource(context, 1,
				(const char **) & kernelSource,
				NULL, &err);

	// Build the program executable
	clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
```

```
    // Create the compute kernel in the program we wish to run
    kernel = clCreateKernel(program, "vecAdd", &err);

    // Create the input and output arrays in device memory
    //   for our calculation
    d_a = clCreateBuffer(context, CL_MEM_READ_ONLY, bytes, NULL,
NULL);
    d_b = clCreateBuffer(context, CL_MEM_READ_ONLY, bytes, NULL,
NULL);
    d_c = clCreateBuffer(context, CL_MEM_WRITE_ONLY, bytes, NULL,
NULL);

    // Write our data set into the input array in device memory
    err = clEnqueueWriteBuffer(queue, d_a, CL_TRUE, 0,
                        bytes, h_a, 0, NULL, NULL);
    err |= clEnqueueWriteBuffer(queue, d_b, CL_TRUE, 0,
                         bytes, h_b, 0, NULL, NULL);

    // Set the arguments to our compute kernel
    err  = clSetKernelArg(kernel, 0, sizeof(cl_mem), &d_a);
    err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &d_b);
    err |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &d_c);
    err |= clSetKernelArg(kernel, 3, sizeof(unsigned int), &n);

    // Execute the kernel over the entire range of the data set
    err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL,
                          &globalSize, &localSize,
                          0, NULL, NULL);

    // Wait for the command queue to get serviced before
    //   reading back results
    clFinish(queue);

    // Read the results from the device
    clEnqueueReadBuffer(queue, d_c, CL_TRUE, 0,
                   bytes, h_c, 0, NULL, NULL );

    // Sum up vector c and print result divided by n,
    //   this should equal 1 within error
    double sum = 0;
    for (i = 0; i < n; i++)
         sum += h_c[i];
    printf("final result: %f\n", sum / n);
```

```
    // release OpenCL resources
    clReleaseMemObject(d_a);
    clReleaseMemObject(d_b);
    clReleaseMemObject(d_c);
    clReleaseProgram(program);
    clReleaseKernel(kernel);
    clReleaseCommandQueue(queue);
    clReleaseContext(context);

    //release host memory
    free(h_a);
    free(h_b);
    free(h_c);

    return 0;
}
```

Παράδειγμα πρόσθεσης διανυσμάτων με OpenCL.

3.12.10 Επεκτάσεις Διανυσματικής Επεξεργασίας SSE

Το σύνολο εντολών SSE (Streaming SIMD Extensions) αναπτύχθηκε από την εταιρεία Intel για την οικογένεια επεξεργαστών x86 (και κατ' επέκταση x86_64) και προήλθε από το σύνολο εντολών MMX της ίδιας εταιρείας. Έχει υποστεί αρκετές ανανεώσεις και είναι η βάση για μεταγενέστερα σύνολα εντολών με σκοπό τη διανυσματική επεξεργασία δεδομένων με το μοντέλο SIMD.

Υιοθετώντας αυτό το σύνολο εντολών, ένας επεξεργαστής (CPU), εκτός των υπόλοιπων εντολών του ρεπερτορίου του και καταχωρητών που διαθέτει, εισάγει και νέους καταχωρητές μεγαλύτερου πλάτους (καταχωρητές XMM) από τη λέξη του επεξεργαστή. Ο τρόπος λειτουργίας στηρίζεται στην τεχνική φόρτωσης αυτών των καταχωρητών με δεδομένα (ένας τέτοιος καταχωρητής μπορεί να αποθηκεύσει αρκετές λέξεις δεδομένων) και εν συνεχεία την εκτέλεση μιας εντολής για την ταυτόχρονη επεξεργασία των δεδομένων (SIMD τεχνική επεξεργασίας).[9]

Το σύνολο εντολών SSE περιλαμβάνει εντολές για επεξεργασία ακέραιων αριθμών, πραγματικών αριθμών κινητής υποδιαστολής και διάφορες άλλες εντολές (για διαχείριση κρυφής και κύριας μνήμης). Οι εντολές εκτίθενται στον προγραμματιστή είτε απ' ευθείας μέσω της γλώσσας μηχανής (assembly) είτε με εγγενείς εντολές (intrinsics) ενός μεταγλωττιστή μιας γλώσσας υψηλότερου επιπέδου.

Πλεονεκτεί σε σχέση με μοντέλα τύπου CUDA (που εκτελούν εργασίες σε άλλους επεξεργαστές του συστήματος) στο ότι δεν έχουμε επιπλέον κόστος για τη μεταφορά δεδομένων από τη CPU στον stream processor. Από την άλλη όμως οι επιδόσεις είναι αρκετά κατώτερες για μεγάλα μεγέθη δεδομένων που θα οδηγηθούν σε έναν εξειδικευμένο stream processor για μαζικά παράλληλη επεξεργασία. Επίσης υπάρχουν αρκετές διαφορετικές εκδόσεις των επεκτάσεων αυτών που σε διάφορους επεξεργαστές ενδέχεται να μην υποστηρίζονται. Αυτό όμως δεν αποτελεί εκ προοιμίου δεδομένο, μιας και το πρόγραμμα δε θα πρέπει να είναι περιορισμένο να εκτελείται σε συγκεκριμένο μοντέλο/α επεξεργαστή. Πρέπει κατά την εκτέλεση να παρθεί η απόφαση ποιες εντολές του ρεπερτορίου SSE δύναται να χρησιμοποιηθούν. Έτσι, συνήθης τρόπος αρχικοποίησης σε ένα πρόγραμμα είναι η ενσωμάτωση ενός πρελούδιου κώδικα που ελέγχει κατά την έναρξη, σε χρόνο εκτέλεσης (run-time), ποια έκδοση του συνόλου SSE υποστηρίζεται από την εκάστοτε αρχιτεκτονική που εκτελείται το πρόγραμμα. Αυτό γίνεται με εντολή γλώσσας μηχανής που επιστρέφει πληροφορίες για την αρχιτεκτονική και της δυνατότητες της CPU (εντολή CPUID).

```
/*
 * sse-vecadd.c - Add two 4D vectors (SSE) using intrinsics.
 *
 * %gcc -Wall -Wextra -o sse-vecadd sse-vecadd.c
 * %./sse-vecadd
 */
```

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


union reg128_t {
      __m128 xmm;
      float f[4];
};


int
main(void)
{
      union reg128_t v1, v2, res;

      /* Little endian, stored in reverse */
      v1.xmm = _mm_set_ps(4.0f,  3.0f,  2.0f,  1.0f);
      v2.xmm = _mm_set_ps(13.0f, 12.0f, 11.0f, 10.0f);

      /* Addition: result = vector1 + vector 2 */
      res.xmm = _mm_add_ps(v1.xmm, v2.xmm);

      printf("result = (%f, %f, %f, %f)\n",
             res.f[3], res.f[2], res.f[1], res.f[0]);

      return 0;
}
```

Παράδειγμα υπολογισμού αθροίσματος 2 τετραδιάστατων διανυσμάτων με SSE intrinsics (gcc) σε γλώσσα C.

3.13 Συνοψίζοντας

Παραπάνω είδαμε ενδεικτικά διάφορα μοντέλα, συστήματα, πλαίσια και πρότυπα αρκετά διαφορετικά μεταξύ τους τα οποία και πιθανώς να μπορούν να συνεργαστούν μεταξύ τους σε ένα ετερογενές περιβάλλον. Για παράδειγμα, ένα σύστημα που στηρίζεται σε ένα σύμπλεγμα MPI μπορεί να περιέχει υπολογισμούς που γίνονται τοπικά (σε κάθε κόμβο του) με χρήση CUDA ή / και επεκτάσεων SSE ή / και POSIX threads. Η έξοδός του μπορεί

να οδηγείται σε ένα άλλο σύστημα (το άλλο σύστημα λειτουργεί ως εξυπηρετητής σύμφωνα με το μοντέλο πελάτη – εξυπηρετητή).

Στον αντίποδα, ο ετερογενής παραλληλισμός επιτάσσει τη σωστή επιλογή των εργαλείων για την υλοποίηση μιας εφαρμογής που από την οπτική γωνία του μηχανικού πρέπει να περιλαμβάνει τη σωστή αντιστάθμιση μεταξύ κόστους και επιδόσεων. Επίσης, εφόσον πρόκειται για έναν τομέα που χαρακτηρίζεται από α) συνεχείς αλλαγές, β) νέα συστήματα και ιδέες που συνεχώς εμφανίζονται και γ) από ευρεία εφαρμογή σε διαφόρου είδους τομείς, είναι φυσικό μια απλή επισκόπηση όπως η παραπάνω να μην αποτελεί έναν πλήρη οδηγό. Τέλος, αφού το σύστημα που παρουσιάζεται στην παρούσα εργασία βασίζεται και χρησιμοποιεί συνδυασμό κάποιων εκ των παραπάνω, είναι καλό ο αναγνώστης να έχει ήδη μια επαφή με αυτά.

4. ΤΟ ΣΥΣΤΗΜΑ CG-STONE

4.1 Σκοπός

Σκοπός του συστήματος CG-Stone είναι η παροχή ενός προγραμματιστικού πλαισίου για την ανάπτυξη εφαρμογών διαχείρισης πολυμεσικού σήματος με παράλληλη επεξεργασία. Αυτό επιτυγχάνεται με χρήση τεχνικών ετερογενούς παραλληλισμού σε ένα κατανεμημένο σύστημα αρχιτεκτονικής Πελάτη-Εξυπηρετητή. Η υλοποίηση είναι σε γλώσσα C και, αναλόγως την αρχιτεκτονική του κάθε κόμβου που αποτελεί το σύστημα χρησιμοποιούνται διαφορετικά προγραμματιστικά πλαίσια για τον παραλληλισμό / ταυτοχρονισμό.

4.2 Πίνακες Κατακερματισμού (Hash Tables)

Ένας Πίνακας Κατακερματισμού (Π.Κ) περιέχει ζεύγη κλειδιών - τιμών (key - value pairs). Είναι μια Αφηρημένη Δομή Δεδομένων (Abstract Data Type) που έχει την ιδιότητα της γρήγορης εισαγωγής, αναζήτησης και διαγραφής των περιεχομένων του. Για την ακρίβεια όλες οι παραπάνω λειτουργίες εκτελούνται σε αναπόσβεστο χρόνο O(1) (amortized) με ασυμπτωτικό συμβολισμό. Η διαδικασία στηρίζεται στην αρχή ότι ένα κλειδί μπορεί να απεικονιστεί μονοσήμαντα από το πεδίο που ανήκει σε ένα πεδίο τιμών με διακεκριμένη πληθικότητα μέσω μιας συνάρτησης, η οποία καλείται συνάρτηση κατακερματισμού (hash function). Στην περίπτωση που δε γνωρίζουμε εκ προοιμίου όλες τις δυνατές τιμές του πεδίου ορισμού (ή δε θέλουμε για λόγους επίδοσης να προβούμε σε αντιστοίχιση όλων αυτών των τιμών από πριν), είναι σίγουρο το γεγονός ότι θα υπάρξουν συγκρούσεις. Δηλαδή δύο διαφορετικές τιμές εισόδου να έχουν την ίδια τιμή εξόδου.

Η υλοποίηση στο σύστημα που παρουσιάζεται στην παρούσα εργασία

είναι η ακόλουθη. Ένας πίνακας απλά συνδεδεμένων λιστών (single linked lists) αποτελεί τον Π.Κ. Αυτός ο πίνακας λιστών αντιπροσωπεύει τους κάδους (buckets) του Π.Κ. Για την εύρεση της θέσης ενός ζεύγους κλειδί - τιμή, πρώτα εφαρμόζουμε τη συνάρτηση κατακερματισμού στο κλειδί. Το πεδίο ορισμού της συνάρτησης κατακερματισμού είναι μια οποιαδήποτε ακολουθία bytes. Το πεδίο τιμών της είναι ένα υποσύνολο των φυσικών αριθμών (αριθμοί των 32 bits). Την τιμή που προκύπτει την αντιστοιχούμε στον αριθμό των κάδων παίρνοντας το υπόλοιπο της διαίρεσης "τιμή της συνάρτησης" προς "αριθμός κάδων". Έτσι, γνωρίζουμε τον κάδο στον οποίο αντιστοιχεί το συγκεκριμένο κλειδί. Στη συνέχεια διατρέχουμε τα στοιχεία της συνδεδεμένης λίστας που αντιστοιχεί σε αυτόν τον κάδο μέχρις ότου εντοπίσουμε το κλειδί. Αυτό γίνεται με τη βοήθεια μιας συνάρτησης σύγκρισης. Η συνάρτηση αυτή έχει ως είσοδο δύο κλειδιά και επιστρέφει ΝΑΙ αν τα κλειδιά είναι ίσα και ΟΧΙ αν δεν είναι. Η συνάρτηση αυτή είναι ευθύνη του προγραμματιστή να οριστεί ώστε να μπορεί να αποφανθεί αν τα δύο προς σύγκριση κλειδιά είναι σημασιολογικά ίσα. Αν το κλειδί βρεθεί, τότε επιστρέφεται η τιμή που αντιστοιχεί σε αυτό. Αλλιώς επιστρέφεται ένα σύμβολο που δηλώνει τη μη ύπαρξή του στον Π.Κ. Με τον παραπάνω τρόπο είμαστε σε θέση να υλοποιήσουμε ένα σύνολο (set), μιας και υπάρχει η εγγύηση για την ύπαρξη μίας μόνο φοράς ενός στοιχείου στη δομή.

Επί πλέον για το γρήγορο έλεγχο μη ύπαρξης ενός κλειδιού γίνεται χρήση φίλτρου Bloom (Bloom filter). Το φίλτρο Bloom αποτελείται από ακολουθία bits. Το μέγεθος της ακολουθίας είναι παραμετροποιήσιμο κατά την αρχικοποίηση του φίλτρου. Κατά το στάδιο της εφαρμογής της συνάρτησης κατακερματισμού στο κλειδί (και πριν τον εντοπισμό του κάδου στον οποίο ανήκει), βρίσκουμε το bit που αντιστοιχεί σε αυτήν την τιμή, παίρνοντας το υπόλοιπο της διαίρεσης της τιμής προς το μέγεθος του φίλτρου. Για τα κλειδιά που έχει ήδη γίνει εισαγωγή στον Π.Κ, οι αντίστοιχες τιμές των bits στο φίλτρο θα είναι 1. Το σημαντικό σημείο που χρήζει ιδιαίτερης προσοχής είναι το ότι ένα φίλτρο Bloom μπορεί να αποφανθεί μόνο για τη μη ύπαρξη ενός κλειδιού (στην περίπτωση που το bit

που αντιστοιχεί στο κλειδί έχει τιμή 0). Η τιμή 1 από την άλλη δεν αποτελεί εγγύηση ότι το κλειδί υπάρχει, λόγω του πεπερασμένου μεγέθους του φίλτρου, του γεγονότος ότι το μέγεθος αυτό δε μας συμφέρει να είναι τόσο μεγάλο όσο η πληθικότητα του πεδίου τιμών της συνάρτησης κατακερματισμού και λόγω των συγκρούσεων της συνάρτησης κατακερματισμού.

Οι λειτουργίες που υποστηρίζει ένας Π.Κ. στο παρόν σύστημα είναι:

- Δημιουργία Π.Κ. με συγκεκριμένο αριθμό κάδων, συνάρτηση κατακερματισμού, συνάρτηση σύγκρισης κλειδιών, συνάρτηση σύγκρισης τιμών, συνάρτηση υπολογισμού μεγέθους κλειδιού, συνάρτηση υπολογισμού μεγέθους τιμής, φίλτρο Bloom.
- Καταστροφή Π.Κ.
- Υπολογισμός μνήμης που καταλαμβάνει η δομή
- Εισαγωγή ζεύγους κλειδί – τιμή
- Έλεγχος ύπαρξης κλειδιού
- Αναζήτηση τιμής βάσει κλειδιού
- Αλλαγή τιμής βάσει κλειδιού με εφαρμογή συνάρτησης μετάλλαξης στην τιμή
- Διαγραφή ζεύγους κλειδί – τιμή
- Εναλλαγή τιμών δύο κλειδιών
- Εφαρμογή συνάρτησης για κάθε κλειδί που έχει ως είσοδο την τιμή του

Εξάλλου, το κάθε κλειδί, εκτός του αναγνωριστικού (της πληροφορίας - όνομα δηλαδή που το διαφοροποιεί από τα άλλα κλειδιά), μπορεί να περιέχει και μεταδεδομένα (metadata), όπως κάποια ομάδα στην οποία αντιστοιχεί, κάποια πληροφορία τοπικότητας ή / και χρονικότητας, χρόνο δημιουργίας ή τελευταίας επεξεργασίας, όνομα κατόχου, κτλ. Εννοείται πως δεν είναι αναγκαίο κάποια ή όλα τα μεταδεδομένα να λαμβάνονται υπόψιν στη συνάρτηση σύγκρισης του κλειδιού. Όμως, η ύπαρξη κάποιων εξ αυτών των μεταδεδομένων είναι αναγκαία στην περίπτωση σήματος που περιέχει,

για παράδειγμα, ακολουθία εικόνων (ώστε να μπορεί να κωδικοποιηθεί η θέση ενός πλαισίου (frame) στο χώρο και το χρόνο).

4.2.1 Ταυτόχρονοι Πίνακες Κατακερματισμού (Concurrent Hash Tables)

Το σύστημα προβλέπει όμως και την ύπαρξη Ταυτόχρονου Πίνακα Κατακερματισμού (Τ.Π.Κ), ο οποίος εκτελεί τις λειτουργίες του αντίστοιχου σειριακού Π.Κ, έχοντας όμως τη δυνατότητα να τις εκτελεί ταυτόχρονα. Έτσι, έχοντας ως δεδομένη την ύπαρξη πολλών νημάτων εκτέλεσης, οι λειτουργίες θα πρέπει να εκτελούνται ορθά (χωρίς απρόβλεπτα αποτελέσματα λόγω θεμάτων συγχρονισμού) και με όσο δυνατό μικρότερο κόστος συγχρονισμού. Η μέθοδος που επελέγει για αυτό είναι το "λεπτόκοκκο" κλείδωμα (fine-grained locking) κάδων. Κάθε κάδος έχει το δικό του κλείδωμα που προστατεύει τη λίστα που περιέχει. Έτσι αρκετά νήματα μπορούν ταυτόχρονα να εργάζονται σε διαφορετικούς κάδους. Όμως, στον ίδιο κάδο, μόνο ένα νήμα μπορεί να εργάζεται σε κάποια χρονική στιγμή. Από τις παραπάνω λειτουργίες, η δημιουργία και η καταστροφή του Τ.Π.Κ. δεν υποστηρίζονται σε πολυνηματική λειτουργία. Άρα, το κύριο νήμα λειτουργίας το οποίο αναλαμβάνει την εκκίνηση και τον τερματισμό του κόμβου που ανήκει ένας Τ.Π.Κ. θα πρέπει να κάνει τις λειτουργίες αυτές χωρίς κάποιο άλλο νήμα να εργάζεται στον Τ.Π.Κ. Επίσης, κατά την ενημέρωση του φίλτρου Bloom, η εναλλαγή ενός bit (από 0 σε 1) γίνεται με ατομική ενημέρωση (atomic update) της λέξης (word) στην οποία ανήκει.

Στο σύστημα, οι Τ.Π.Κ. λειτουργούν ως στρώμα αποθήκευσης πληροφορίας στη μνήμη (in-memory storage) και ταυτόχρονης προσπέλασης. Αρχιτεκτονικά, αυτό έχει ομοιότητες με τη χρήση κρυφής μνήμης (cache). Παρέχει γρήγορη προσπέλαση με αλλαγή των περιεχομένων σε πληροφορία όπου υπάρχει αρκετή κίνηση.

4.2.2 Μετρήσεις Τ.Π.Κ.

Μετρήσαμε την απόδοση δύο διαφορετικών τεχνικών κλειδώματος:

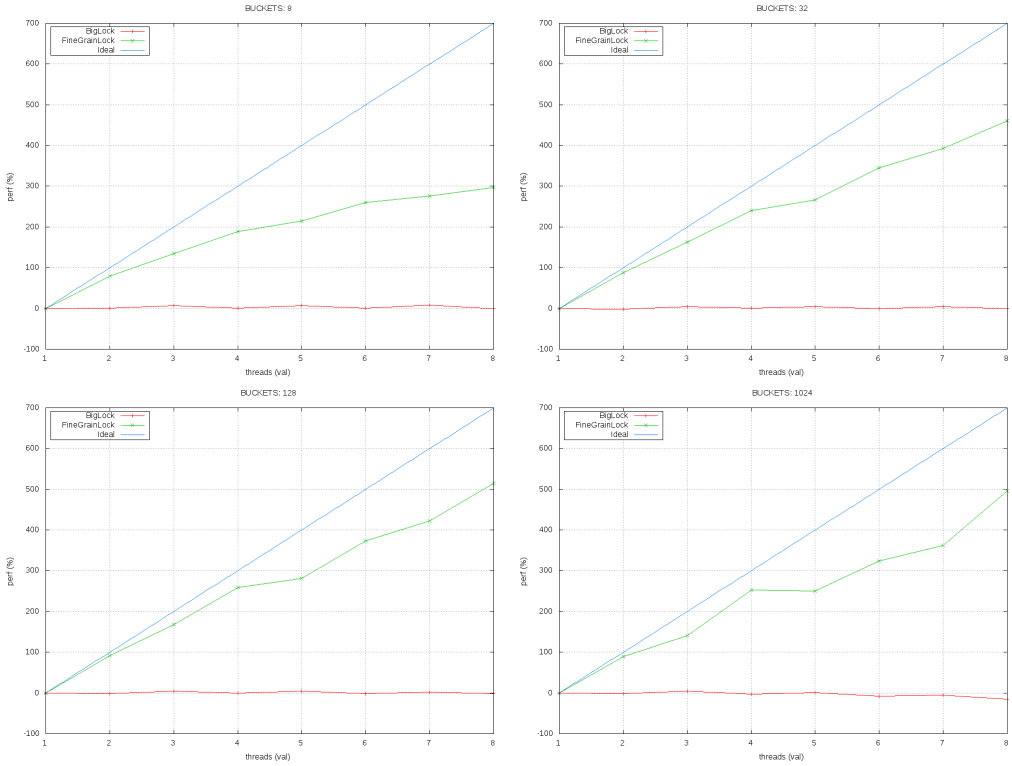
α) "μεγάλο" κλείδωμα (big lock), όπου κλειδώνουμε ολόκληρη τη δομή του Π.Κ. σε κάθε λειτουργία και

β) "λεπτόκοκκο" κλείδωμα (fine-grained locking), όπου κάθε φορά κλειδώνουμε τον/τους απαραίτητο(υ)ς κάδο(υ)ς.

Η απεικόνιση των νημάτων (affinity) σε επεξεργαστικούς πυρήνες (cores) ήταν 1 - 1 (κάθε νήμα εκτελούταν στον ίδιο πυρήνα καθ' όλη τη διάρκεια της μέτρησης). Οι μετρήσεις είναι για τις ίδιες τυχαίες λειτουργίες στον Τ.Π.Κ. (η σειρά έκδοσης των εντολών που θα επιδράσουν στον Τ.Π.Κ. είναι ίδια κάθε φορά, δηλαδή η φύτρα (seed) της ψευδοτυχαίας γεννήτριας είναι η ίδια). Οι λειτουργίες που πραγματοποιούνται είναι: εισαγωγή (insert), διαγραφή (delete), αναζήτηση (lookup) και ανταλλαγή (swap). Σε κάθε μέτρηση έγιναν 1.000.000 λειτουργίες, οπότε κατά μέσο όρο αναμένουμε να έχουμε 250.000 εγγραφές και 250.000 διαγραφές. Οδηγούμαστε στα εξής συμπεράσματα:

- Για 1 thread και οι δύο υλοποιήσεις έχουν την ίδια περίπου επίδοση (αναμενόμενο). Από εκεί και πέρα, όμως, η υλοποίηση fine-grained locking είναι σαφώς καλύτερη, λόγω παραλληλίας.

- Το fine-grained locking κλιμακώνεται αρκετά καλύτερα, όπως αναμενόταν. Μάλιστα, σε μερικές περιπτώσεις αγγίζει την ιδανική επίδοση (αν οι συνθήκες το ευνοούν - εδώ όταν ο αριθμός των buckets είναι ικανοποιητικά μεγάλος ώστε να περιορίζονται στο ελάχιστο οι συγκρούσεις).



Μετρήσεις Τ.Π.Κ.

4.3 Αποθήκευση και ανάκτηση δεδομένων

Τα περιεχόμενα ενός Τ.Π.Κ. είναι στην κύρια μνήμη. Ο προγραμματιστής δηλώνει ένα σχήμα οργάνωσης (schema) για τα ζεύγη κλειδιών - τιμών. Το σχήμα επεξεργαζόμενο από το σύστημα παράγει συρμό bytes που μπορούν να αποθηκευθούν με τους κατάλληλους οδηγούς σε κάποιον προορισμό. Ένας οδηγός είναι ένα τμήμα κώδικα που εκτελεί τη μεταφορά του συρμού των bytes στον προβλεπόμενο προορισμό. Προορισμός είναι ένα άλλο σύστημα (π.χ. μια βάση δεδομένων), ένα αρχείο σε κάποιο Σύστημα Αρχείων (File System), μια υποδοχή (socket) ή ακόμα και κάποιο ξεχωριστό τμήμα της κύριας μνήμης. Έτσι, τα περιεχόμενα μεταβιβάζονται σε κάποιον αποδέκτη εκτός συστήματος. Το αρχείο μπορεί να είναι τοπικό για τον κόμβο ή κάποιο Κατανεμημένο Σύστημα Αρχείων (Distributed File System). Η διαδικασία αυτή αποτελεί τη σειριοποίηση των δεδομένων του Τ.Π.Κ. Με αντίστοιχο τρόπο έχουμε την απο-σειριοποίηση δεδομένων από

μια πηγή, όπου πάλι βάσει του σχήματος, τα δεδομένα εισάγονται στον Τ.Π.Κ. Αξίζει να σημειωθεί πως κατά την απο-σειριοποίηση και, εφόσον αυτό είναι δυνατό, τα εισερχόμενα δεδομένα συγχωνεύονται με τα τυχόν υπάρχοντα δεδομένα στον Τ.Π.Κ. Αυτή η διαδικασία συμβαίνει ταυτόχρονα με την υπόλοιπη λειτουργία του συστήματος.

4.4 Επεξεργασία δεδομένων

Ο προγραμματιστής υλοποιεί διαπροσωπείες (interfaces) για τύπους δεδομένων που αποτελούν τις τιμές (που αναπαριστούν την πληροφορία). Οι διαπροσωπείες αυτές περιλαμβάνουν κατασκευαστές αντικειμένων (constructors) για την πληροφορία (π.χ. τα εικονοστοιχεία μιας εικόνας), καταστροφείς αντικειμένων (destructors), συναρτήσεις ανάκτησης ιδιοτήτων (getters) και συναρτήσεις επεξεργασίας - μετάλλαξης, συναρτήσεις σειριοποίησης - απο-σειριοποίησης. Σε κάθε συνάρτηση επεξεργασίας, ο προγραμματιστής έχει τη δυνατότητα να χρησιμοποιήσει στοιχεία ετερογενούς παραλληλισμού είτε μέσω συναρτήσεων που παρέχει το σύστημα είτε με επί τόπου υλοποίηση από αυτόν. Για παράδειγμα, στην περίπτωση επεξεργασίας μιας εικόνας μεγάλου μεγέθους, ο προγραμματιστής μπορεί να χρησιμοποιήσει μια ήδη υλοποιημένη συνάρτηση με το πρότυπο CUDA ή να προβεί στη δική του υλοποίηση χρησιμοποιώντας CUDA ή SSE ή POSIX threads ή κάτι άλλο. Το σύστημα παρέχει έτοιμες υλοποιήσεις για τους εξής τύπους: ακέραιοι αριθμοί (μεγέθους 64 bits), πραγματικοί αριθμοί (floats, δηλαδή απλής ακρίβειας με κινητή υποδιαστολή κατά IEEE), ανύσματα bytes (byte vectors), ανύσματα ακέραιων αριθμών (integer vectors μεγέθους 64 bits για τον κάθε ακέραιο) και ανύσματα πραγματικών αριθμών (float vectors και double vectors, δηλαδή πραγματικών αριθμών κινητής υποδιαστολής με διπλή ακρίβεια κατά IEEE). Στην περίπτωση ανυσμάτων υπάρχουν εντολές που δρουν σε όλο το άνυσμα και εντολές που δρουν σε κάποια περιοχή του, ώστε να μπορεί ο προγραμματιστής να εργαστεί σε μια περιοχή ενδιαφέροντος (region of interest).

Αντίστοιχα για τα κλειδιά των τιμών μπορεί να χρησιμοποιήσει την υπάρχουσα διαπροσωπεία ή να προβεί στη δική του υλοποίηση. Το σύστημα παρέχει κλειδιά που αναπαρίστανται με συμβολοσειρές (strings) και με μεταδεδομένα τη χρονική στιγμή δημιουργίας, τελευταίας ενημέρωσης και τελευταίας προσπέλασης. Οι χρονικές στιγμές προέρχονται από το χρόνο συστήματος Unix (Unix time) και είναι ο αριθμός των δευτερολέπτων από την αρχή της 1 Ιανουαρίου 1970 (epoch) έως τον εκάστοτε χρόνο. Έτσι, ο προγραμματιστής έχει την επιλογή της κωδικοποίησης πληροφορίας μέσα στο κλειδί της συμβολοσειράς. Για παράδειγμα, το κλειδί ενός τμήματος με όνομα "X42-Y69" μιας εικόνας με όνομα "Image1" του πλαισίου με αύξοντα αριθμό 69105 μπορεί να είναι η συμβολοσειρά "video$Image1$X42-Y69$69105". Αν υποθέσουμε ότι τα εικονοστοιχεία της εικόνας αναπαρίστανται με ένα άνυσμα bytes στο χρωματικό χώρο RGB, αυτό το ζεύγος κλειδιού - τιμής θα βρίσκεται σε ένα Τ.Π.Κ. που οι τιμές του είναι τύπου bytevector.

Στο παραπάνω παράδειγμα, μια εντολή επεξεργασίας έχει τη μορφή

```
ProcessCommand(BYTEVECTOR_TAB, "video$Image1$X42-Y69$69105", Args)
```

Τα Args είναι ένα άνυσμα που περιέχει τις παραμέτρους που θα οδηγήσουν την εντολή επεξεργασίας ProcessCommand.

Το ρεπερτόριο περιλαμβάνει και εντολές επεξεργασίας που αφορούν παραπάνω του ενός κλειδιού. Με τη βοήθεια των εντολών επεξεργασίας (και την επέκταση του παρεχόμενου ρεπερτορίου εντολών), ο προγραμματιστής είναι σε θέση να υλοποιεί τους Αλγοριθμικούς Σκελετούς με τους οποίους έχει μοντελοποιήσει το πρόβλημα προς επίλυση. Επίσης, παρέχονται εντολές θέασης δεδομένων του συστήματος (views) οι οποίες δεν εισάγουν κάποια παρενέργεια (side-effect) (δε μεταλλάσσουν τιμές) και απλώς επιστρέφουν τιμές κλειδιών.

Εξάλλου, υπάρχουν βοηθητικές εντολές, όπως για παράδειγμα η ανάγνωση ενός αρχείου εικόνας κάποιου φορμά (π.χ. TGA) και εντολές

επισκόπησης του συστήματος (για παράδειγμα εντολή που επιστρέφει την ποσότητα μνήμης που καταναλώνεται από τους Τ.Π.Κ. για αποθήκευση των δεδομένων).

```
/*
 * fixm.h - Objtab fixnum value memory.
 *
 * Written by:
 *   Dionisis G. Kakoliris
 *
 * 2012/04/01
 */


#ifndef H_FIXM_H_
#define H_FIXM_H_

#include "shared/libshared"


#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */


enum {
      FIRST_FIXM_OP_MINUS_ONE = -1,
      FIXM_SET,
      FIXM_INC,
      FIXM_ADD,
      ALL_FIXM_OPS
};


extern const struct obj_iface_t g_objtab_fixm_val_iface;
static const struct obj_oper_t g_objtab_fixm_oper_set = {
      FIXM_SET, 1, 0, 0
};
static const struct obj_oper_t g_objtab_fixm_oper_inc = {
      FIXM_INC, 1, 0, 0
};
static const struct obj_oper_t g_objtab_fixm_oper_add = {
```

```
      FIXM_ADD, 2, 1, 1
};


#ifdef __cplusplus
}
#endif /* __cplusplus */


#endif /* H_FIXM_H_ */
/*
 * fixm.c - Objtab fixnum value memory.
 *
 * Written by:
 *   Dionisis G. Kakoliris
 *
 * 2012/04/01
 */


#include "fixm.h"


static voidadd_fixm_val(void *result, void *x, const void *y);
static int compar_fixm_val(const void *x, const void *y);
static voiddelete_fixm_val(void *arg);
static voidinc_fixm_val(void *result, void *x, const void *y);
static void      *new_fixm_val(const void *arg);
static voidprint_fixm_val(const void *arg);
static voidset_fixm_val(void *result, void *x, const void *y);
static size_t    sizeof_fixm_val(const void *arg);


const struct obj_iface_t g_objtab_fixm_val_iface = {
      NULL,
      {new_fixm_val}, 1,
      delete_fixm_val,
      sizeof_fixm_val,
      NULL,
      NULL,
      {0}, 0,
      NULL,
      compar_fixm_val,
      NULL, NULL,
```

```
	{0}, 0,
	{set_fixm_val, inc_fixm_val, add_fixm_val}, 3,
	print_fixm_val, NULL
};


static void *
new_fixm_val(const void *arg)
{
	const int64_t *zero = (const int64_t *) arg;
	int64_t *val;

	if (!arg) {
		WARN_OBJ_NULL_ARGUMENTS;

		return NULL;
	}

	MALLOC_SAFE(val, sizeof(int64_t), int64_t *, "fixmval");
	*val = *zero;

	return (void *) val;
}

static void
delete_fixm_val(void *arg)
{
	if (arg)
		FREE(arg);
}

static size_t
sizeof_fixm_val(const void *arg)
{
	if (arg)
		return sizeof(int64_t);

	return 0;
}

static int
compar_fixm_val(const void *x, const void *y)
{
	const int64_t *a = (const int64_t *) x;
```

```
	const int64_t *b = (const int64_t *) y;
	int r;

	r = compar_null_obj(x, y);
	if (r)
		return r;

	if (*a < *b)
		return -1;
	else if (*a > *b)
		return 1;

	return 0;
}

static void
print_fixm_val(const void *arg)
{
	const int64_t *n = (const int64_t *) arg;

	if (n)
		printf("%" PRId64, *n);
}

static void
set_fixm_val(void *result, void *x, const void *y)
{
	const int64_t *b = (const int64_t *) y;
	int64_t *a, *r;

	if (!result) {
		WARN_OBJ_NULL_ARGUMENTS;

		return;
	}
	if (!x) {
		WARN_OBJ_NULL_ARGUMENTS;

		return;
	}
	if (!y) {
		WARN_OBJ_NULL_ARGUMENTS;

		return;
```

```
	}

	a = (int64_t *) x;
	r = (int64_t *) result;
	*a = *b;
	*r = *a;
}

#ifdef GCC_GT_40000
#ifdef GCC_GT_40600
#pragma GCC diagnostic push
#endif /* GCC_GT_40600 */
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif /* GCC_GT_40000 */
static void
inc_fixm_val(void *result, void *x, const void *y)
{
	int64_t *a, *r;

	if (!result) {
		WARN_OBJ_NULL_ARGUMENTS;

		return;
	}
	if (!x) {
		WARN_OBJ_NULL_ARGUMENTS;

		return;
	}

	a = (int64_t *) x;
	r = (int64_t *) result;
	(*a)++;
	*r = *a;
}
#ifdef GCC_GT_40000
#pragma GCC diagnostic warning "-Wunused-parameter"
#ifdef GCC_GT_40600
#pragma GCC diagnostic pop
#endif /* GCC_GT_40600 */
#endif /* GCC_GT_40000 */

static void
add_fixm_val(void *result, void *x, const void *y)
```

```
{
      const int64_t *b = (const int64_t *) y;
      int64_t *a, *r;

      if (!result) {
            WARN_OBJ_NULL_ARGUMENTS;

            return;
      }
      if (!x) {
            WARN_OBJ_NULL_ARGUMENTS;

            return;
      }
      if (!y) {
            WARN_OBJ_NULL_ARGUMENTS;

            return;
      }

      a = (int64_t *) x;
      r = (int64_t *) result;
      *a = *a + *b;
      *r = *a;
}
```

Ένα απλό interface για ακέραιους αριθμούς 64 bits.

4.5 Επικοινωνία με το περιβάλλον

Η διαπροσωπεία με το περιβάλλον γίνεται μέσω ενός πρωτοκόλλου. Ένας εξυπηρετητής δέχεται αιτήσεις με τη βοήθεια υποδοχών. Ο εξυπηρετητής διαθέτει ένα προκαθορισμένο σύνολο νημάτων (thread pool), υλοποιημένο με POSIX threads, των οποίων η δουλειά είναι η εξυπηρέτηση των αιτήσεων. Όταν φτάσει μια αίτηση, ο εξυπηρετητής την τοποθετεί σε μια ουρά (queue). Το πρώτο νήμα που είναι ελεύθερο για εξυπηρέτηση αίτησης, εξάγει από την ουρά μια αίτηση και προχωρά στην εξυπηρέτησή της. Η εξυπηρέτηση αφορά την επεξεργασία του μηνύματος (σύμφωνα με το προκαθορισμένο πρωτόκολλο), την εξαγωγή της αιτούμενης εντολής και τις

προβλεπόμενες ενέργειες που αφορούν το στρώμα αποθήκευσης δεδομένων, σύμφωνα με τα παραπάνω.

Αφαιρετικά, σε γενική θεώρηση, αυτό αποτελεί μια προσέγγιση του μοντέλου Δράστη (Actor model) στο μοντέλο Πελάτη-Εξυπηρετητή (Client-Server model). Πρέπει όμως να σημειώσουμε πως κατά το μοντέλο Δράστη, οι δράστες - νήματα που εξυπηρετούν αιτήσεις είναι συνήθως πολλοί σε αριθμό και υλοποιούνται με ελαφρά νήματα ή διεργασίες (green threads) τη διαχείριση των οποίων αναλαμβάνει μια εικονική μηχανή.

Σε μια αίτηση υπάρχει μια ομάδα εντολών (batch) με τα αντίστοιχα ορίσματα. Οι εντολές που επιδρούν στο στρώμα δεδομένων (και κατ' επέκταση στους Τ.Π.Κ) δεν επιδρούν απευθείας στους κάδους των Τ.Π.Κ. Ο λόγος είναι ότι πρέπει να αποφύγουμε όσο το δυνατόν περισσότερο τα συνεχή κλειδώματα, ειδικά όταν πρόκειται να εργαστούμε στον ίδιο κάδο. Οι εντολές μιας αίτησης ομαδοποιούνται σύμφωνα με τον κάδο που αφορούν και ανάλογα με τις εξαρτήσεις που παρουσιάζουν (π.χ. η ανάγνωση που έπεται μιας αλλαγής πρέπει να εκτελεστεί ντετερμινιστικά μετά την εγγραφή) εκτελούνται. Κατά την ολοκλήρωση της εξυπηρέτησης, όλες οι λειτουργίες που απομένουν, εκτελούνται. Προσπαθούμε δηλαδή, λειτουργίες που αφορούν τον ίδιο κάδο να εκτελεστούν στο ίδιο κρίσιμο τμήμα, χωρίς να κλειδώνουμε - απελευθερώνουμε τον ίδιο κάδο. Τη διαχείριση της ομαδοποίησης και εκτέλεσης των λειτουργιών τις αναλαμβάνει η υλοποίηση του Τ.Π.Κ.

Επίσης, με τη βοήθεια των εντολών ανάκτησης δεδομένων (getters) και των εντολών αποθήκευσης των δεδομένων σε κάποιο άλλο σύστημα, επιτυγχάνεται η διαλειτουργικότητα (interoperability) με άλλα συστήματα. Για παράδειγμα αποθηκεύοντας την κατάσταση των δεδομένων (κλειδιά - τιμές) σε κάποιο Κατανεμημένο Σύστημα Αρχείων, ένα άλλο σύστημα τύπου MapReduce (π.χ. Hadoop) μπορεί να προβεί σε περεταίρω επεξεργασία των δεδομένων.

4.6 Κατανεμημένη αρχιτεκτονική

Το σύστημα αποτελείται από κόμβους. Κάθε κόμβος είναι υπεύθυνος για συγκεκριμένα κλειδιά. Η κατάτμηση γίνεται με συνάρτηση κατακερματισμού (ώστε να αποφασιστεί σε ποιον κόμβο αντιστοιχεί κάποιο κλειδί). Ο εξυπηρετητής αναλαμβάνει να οδηγήσει τις αιτήσεις στους κατάλληλους κόμβους. Κατά το πέρας της εξυπηρέτησης μιας αίτησης, υπάρχει το στάδιο της αναγωγής (reduction) για εντολές που αφορούν περισσότερα του ενός κλειδιά, τα οποία πιθανώς βρίσκονται σε διαφορετικούς κόμβους.

4.7 Το σύστημα παραγωγής Making In Scons

Για την παραγωγή των εκτελέσιμων αρχείων και βιβλιοθηκών του συστήματος αναπτύχθηκε το σύστημα παραγωγής (build system) Making In Scons. Αποτελεί επέκταση του ήδη υπάρχοντος συστήματος παραγωγής Scons, το οποίο χρησιμοποιεί τη γλώσσα σεναρίων Python για τη συγγραφή κανόνων και εξαρτήσεων που οδηγούν στην παραγωγή εξαγόμενων αρχείων. Ο σκοπός της ανάπτυξης του συστήματος ήταν η συγγραφή των κανόνων και των εξαρτήσεων με δηλωτικό τρόπο. Γι' αυτόν το λόγο χρησιμοποιήθηκε η γλώσσα σεναρίων AWK για την ανάλυση των κανόνων και τη μετέπειτα μετατροπή σε κώδικα Python. Με δηλωτικό τρόπο εκφράζεται και η ιεραρχία του έργου (προς παραγωγή), που είναι οργανωμένο σε υπο-καταλόγους (sub-directories), οπότε αποφεύγεται η επανειλλημένη συγγραφή όμοιου κώδικα. Το σύστημα Making In Scons χρησιμοποιείται για παραγωγή εκτελέσιμων αρχείων και βιβλιοθηκών έργων που αφορούν κώδικα γραμμένο σε γλώσσα C ή C++.

```
#!/usr/bin/awk -fmakiss.awk

#
# 3rd party auxiliary hash functions.
#


OUT_MERGED_LIB_SHARED = hashaux

LIB_PATHS = {
      #/3rdparty/jsw
      #/3rdparty/fnv-5.0.3
      #/3rdparty/murmur1
      #/3rdparty/murmur2
      #/3rdparty/murmur3
      #/3rdparty/city
      #/3rdparty/city32
      #/3rdparty/spooky
      #/3rdparty/librhash
}

LIBS = {
      city
      city-c32
      fnv
      hlib
      rhash
      murmur1
      murmur2
      murmur3
      spooky-c
}
```

Παράδειγμα ενός αρχείου που οδηγεί το σύστημα παραγωγής Making In Scons για την παραγωγή μιας βιβλιοθήκης συναρτήσεων κατακερματισμού.

5. ΕΝΔΕΙΚΤΙΚΕΣ ΠΕΡΙΠΤΩΣΕΙΣ ΧΡΗΣΗΣ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ (USE CASES)

5.1 Γενικά

Στο προηγούμενο κεφάλαιο είδαμε την υλοποίηση του συστήματος. Ως προγραμματιστικό πλαίσιο μπορεί να αποτελέσει τη βάση για την ανάπτυξη πολλών και διαφορετικών μεταξύ τους συστημάτων. Ενδεικτικά, αναφέρουμε παρακάτω διάφορες από αυτές τις περιπτώσεις χρήσης, οι οποίες μπορούν να αποτελέσουν με τη σειρά τους τη βάση για την ανάπτυξη άλλων συστημάτων. Βέβαια, στην παρούσα εργασία, έμφαση έχει δοθεί ως τώρα στην επεξεργασία πολυμεσικού σήματος. Όμως, λόγω της φύσης του συστήματος, η χρήση του δεν περιορίζεται μόνο σε αυτόν τον τομέα όπως θα δούμε και παρακάτω.

5.2 Ως απλός Πίνακας Κατακερματισμού

Αποτελεί το σημείο εφαρμογής για τις περιπτώσεις χρήσης του συστήματος. Ξεκινώντας από αυτήν την απλή εφαρμογή και, εφοδιάζοντας το σύστημα με συναρτήσεις και διαπροσωπείες, μπορούμε να το χρησιμοποιήσουμε για πληθώρα εφαρμογών.

5.3 Ως εργαλείο ταυτόχρονης επεξεργασίας πολυμεσικού περιεχομένου

Αυτή η περίπτωση χρήσης υπήρξε αρχικά το εφαλτήριο για την ανάπτυξη του συστήματος στο Εργαστήριο Πολυμέσων του Ε.Μ.Π. Η ταυτόχρονη προσπέλαση πληροφορίας στους Ταυτόχρονους Πίνακες Κατακερματισμού (Τ.Π.Κ) των κόμβων του συστήματος χρησιμοποιείται για

την επίτευξη συνεργασίας όσον αφορά τη διαχείριση του περιεχομένου. Έτσι, διαφορετικοί χρήστες επιδρούν στο μέρος της πληροφορίας που θέλουν και, με τη βοήθεια του συστήματος, οι αλλαγές που κάνουν γίνονται αυτόματα ορατές στους υπόλοιπους χρήστες. Επί πλέον, το σύστημα μπορεί να διατηρεί αντίγραφα ασφαλείας - στιγμιότυπα των δεδομένων (snapshots) ανά τακτά χρονικά διαστήματα, τα οποία και μπορούν να προωθηθούν σε άλλες ομάδες χρηστών. Η κατανεμημένη αυτή ροή εργασίας παρουσιάζει αρκετές ομοιότητες με Κατανεμημένα Συστήματα Ελέγχου Αναθεωρήσεων (Distributed Version Constrol Systems - DVCS), όπου αρκετοί προγραμματιστές συνεργάζονται στην υλοποίηση ενός συστήματος με κατανεμημένο τρόπο εργασίας δουλεύοντας ταυτόχρονα σε ποικίλες διακλαδώσεις. Ακόμη, ο ετερογενής παραλληλισμός συμβάλλει στη βέλτιστη αξιοποίηση των πόρων ενός σύγχρονου υπολογιστικού συστήματος και την επίτευξη κάθετης κλιμάκωσης (vertical scaling) για το σύστημα. Από την άλλη, η κατάτμηση σε κόμβους συμβάλλει στην επίτευξη οριζόντιας κλιμάκωσης (horizontal scaling). Ιστορικά, η αρχική έκδοση του συστήματος, αφορούσε τη βελτιστοποίηση αλγορίθμων ψηφιακής επεξεργασίας εικόνας. Ύστερα, και στα πλαίσια της παρούσας Διπλωματικής Εργασίας, εξελίχθηκε σε ένα σύστημα υποστήριξης ενός προγραμματιστικού πλαισίου για τη γενικότερη επίλυση προβλημάτων που αφορούν τη διαχείριση και επεξεργασία πολυμεσικού σήματος με αποδοτικό τρόπο.

5.4 Ως εργαλείο αποθήκευσης στατιστικών δεδομένων και σημασιολογικού περιεχομένου πληροφορίας

Το σύστημα μπορεί να χρησιμοποιηθεί όχι μόνο για την επεξεργασία πολυμεσικού σήματος, αλλά και για την αποθήκευση μεταδεδομένων που αφορούν το πολυμεσικό περιεχόμενο. Διάφορα άλλα συστήματα που επεξεργάζονται πολυμεσική ή κάποιου άλλου είδους πληροφορία μπορούν να καταχωρούν στο σύστημα διάφορα αποτελέσματα, τα οποία συνήθως αφορούν σημασιολογία ή στατιστικά δεδομένα. Για παράδειγμα, μπορούμε

να αποθηκεύουμε το πόσα εικονοστοιχεία ενός συγκεκριμένου χρώματος υπάρχουν σε μια εικόνα (κλειδί: μια συμβολοσειρά που περιγράφει την εικόνα, τιμή: ο ακέραιος αριθμός που περιέχει το πλήθος των εν λόγω εικονοστοιχείων). Επί προσθέτω, αν η εικόνα συνενωθεί με κάποια άλλη εικόνα, με χρήση της λειτουργίας πρόσθεσης που παρέχει η διαπροσωπεία για τις ακέραιες τιμές, μπορούμε να επαυξήσουμε την αρχική πληροφορία, ώστε να περιέχει το άθροισμα των εικονοστοιχείων του χρώματος που μας ενδιαφέρει. Εξάλλου, για κάποιο σύστημα που προβαίνει σε απόδοση σημασιολογίας σε περιεχόμενο, το σύστημα μπορεί να χρησιμοποιηθεί για την αποθήκευση της εξαγόμενης πληροφορίας. Έτσι, μια ομάδα συστημάτων που ενεργούν σε μεγάλη ποσότητα πληροφορίας είναι σε θέση να χρησιμοποιήσουν το σύστημα ως ενδιάμεσο στρώμα αποθήκευσης και ανάκτησης πληροφοριών. Σε αυτήν την περίπτωση το σύστημα λειτουργεί ως ενδιάμεσο συστατικό στοιχείο (component) για τη διαλειτουργικότητα (interoperability) μεταξύ συνεργαζόμενων συστημάτων. Στο εργαστήριο χρησιμοποιήσαμε το σύστημα και με αυτόν το ρόλο. Ένα σύστημα μηχανικής εκμάθησης είχε ως είσοδο διάφορες εικόνες, στις οποίες υπήρχαν περιοχές ενδιαφέροντος. Στη συνέχεια μετά από επεξεργασία, τα δεδομένα μεταφέρονταν σε μια συστοιχία νευρωνικών δικτύων. Τα αποτελέσματα αποθηκεύονταν στο σύστημα, οπότε και ήταν διαθέσιμα σε άλλα συστήματα.

5.5 Ως στρώμα αποθήκευσης

Το σύστημα μπορεί να χρησιμοποιηθεί καθαρά ως στρώμα αποθήκευσης για ένα άλλο σύστημα. Μέσω δικτύου, τα δεδομένα μεταφέρονται και ανακτώνται όποτε αυτό είναι απαραίτητο, έχοντας εξασφαλίσει και την ύπαρξη αντιγράφων ασφαλείας. Με χρήση της διαπροσωπείας του τύπου για ανύσματα bytes (byte vectors) έχουμε την ουδέτερη θεώρηση της αποθηκευμένης πληροφορίας. Πρέπει όμως να σημειωθεί πως η συγκεκριμένη χρήση δεν ενδείκνυται για περιπτώσεις όπου

υπάρχει ανάγκη για συνεχή αποθήκευση περιεχομένου μεγάλου μεγέθους σε υποδομή με δίκτυο περιορισμένης διαμεταγωγής δεδομένων.

5.6 Ως σύστημα υποβοήθησης στη δειγματοληψία σημάτων

Λόγω των υψηλών επιδόσεων που οφείλονται στη βέλτιστη χρήση του υλικού και την υλοποίηση, το σύστημα μπορεί να χρησιμοποιηθεί για τη λήψη δειγμάτων ενός σήματος σε μεγάλες συχνότητες δειγματοληψίας. Από ένα σύστημα αισθητήρων, μέσω δικτύου, τα δείγματα οδηγούνται προς αποθήκευση στο σύστημα, όπου ταυτόχρονα μπορούν να εκτελούνται και λειτουργίες επεξεργασίας των δειγμάτων σε πραγματικό χρόνο. Η δυνατότητα εργασίας σε πραγματικό χρόνο προσδίδει στο σύστημα αρκετές δυνατότητες.

5.7 Ως πλαίσιο για την ταυτόχρονη εργασία σε ιεραρχικές δομές και δυναμικές τοπολογίες

Ιεραρχικές δομές (που έχουν σκελετική διάταξη) όπου οι κόμβοι τους (αρθρώσεις) αντιδρούν σε ερεθίσματα μπορούν να αναπαρασταθούν με τη βοήθεια του εν λόγω συστήματος. Ομοίως και για ιεραρχικές δομές οι οποίες δεν παρουσιάζουν στατικότητα όσον αφορά τους κόμβους από τους οποίους αποτελούνται - αποτελούν δηλαδή δυναμικές τοπολογίες. Σε κάποια χρονική στιγμή, το στιγμιότυπο (snapshot) των δεδομένων που αποτελούν τη δομή χρησιμοποιείται για την ανάπτυξη της δομής από τους κόμβους που αποτελείται σύμφωνα με την παρακάτω διαδικασία:

για κάθε κόμβο−γονέα:
  για κάθε κόμβο−παιδί του γονέα:
    για κάθε ιδιότητα i του κόμβου−παιδιού:
      ιδιότητα i = επαύξηση ιδιότητας με αντίστοιχη ιδιότητα i του κόμβου−γονέα
        ; (κληρονομικότητα από τον κόμβο−γονέα)

Η παραπάνω διαδικασία εφαρμοζόμενη για τη θέση και τον προσανατολισμό των αρθρώσεων μιας σκελετικής διάταξης ξεκινά από τον κόμβο-ρίζα (γονέας των υπόλοιπων κόμβων), ο οποίος καθορίζει τη θέση και τον προσανατολισμό της όλης διάταξης. Για κάθε κόμβο-παιδί του, η θέση του κόμβου-παιδιού προκύπτει από τη διανυσματική πρόσθεση της θέσης του με αυτήν του γονέα. Για τον προσανατολισμό, αναλόγως την υλοποίηση ενδέχεται να έχουμε πολλαπλασιασμό πινάκων (στην περίπτωση αναπαράστασης του προσανατολισμού με πίνακες μετασχηματισμών), πολλαπλασιασμούς στο χώρο των τετραδονίων (στην περίπτωση αναπαράστασης του προσανατολισμού με τετραδόνια) ή κάτι άλλο. Η διαδικασία συνεχίζεται αναδρομικά για κάθε κόμβο. Για κάθε επίπεδο της ιεραρχίας, οι υπολογισμοί μπορούν να γίνονται παράλληλα.

5.8 Ως εργαλείο αναπαράστασης γράφων

Αποτελεί έναν από τους απώτερους σκοπούς γύρω από τη σχεδίαση του συστήματος. Είναι γενίκευση της προηγούμενης περίπτωσης. Η αναπαράσταση γίνεται χρησιμοποιώντας ως αξίες τις γραμμές του πίνακα γειτνίασης του γράφου. Η ταυτόχρονη εργασία στους κόμβους του γράφου είναι εγγενής ιδιότητα του συστήματος. Η χρήση μιας αφηρημένης δομής όπως ο γραφής προσδίδει στο σύστημα τη βασική ιδιότητα της υπολογιστικής γενικού σκοπού (general purpose computing).

5.9 Ως εργαλείο προσομοίωσης για προβλήματα Τεχνητής Νοημοσύνης

Στην Τεχνητή Νοημοσύνη υπάρχουν δράστες (agents) που δρουν σε κάποιο περιβάλλον και, αναλόγως την κατάστασή τους και την αντίληψή τους για την κατάσταση του συστήματος, δρουν. Η κατάσταση κάθε δράστη κωδικοποιείται στο σύστημα, όπως και η ιστορία (history) του. Με τις

κατάλληλες συναρτήσεις για δράστες και χρησιμοποιώντας το μοντέλο Δράστη (actor model) η εξέλιξη ενός συστήματος που επιδρούν ταυτόχρονα οι νοήμονες δράστες προσομοιώνεται από το σύστημα.

5.10 Ως μηχανή χωρο-χρονικής λογικής

Για ένα σύστημα λογικής (κυρίως προτασιακής λογικής), το σύστημα παρέχει έναν τρόπο έκφρασης κατηγορημάτων (predicates) συναρτήσει του χώρου και του χρόνου. Για παράδειγμα το κατηγόρημα $P(t_{1,}t_{2,}\ldots,t_n)$, όπου $t_{1,}t_{2,}\ldots,t_n$ οι όροι (terms), μπορεί να ισχύει ή όχι για κάποιο σημείο του χώρου ή του χρόνου. Αυτό επιτυγχάνεται με την επαύξηση του ονόματος των κλειδιών (στον Τ.Π.Κ. που αποθηκεύονται οι όροι ως κλειδιά), ώστε να περιέχεται σημασιολογία για το χώρο (εκφράζεται με κάποιο όνομα) και το χρόνο (εκφράζεται ως χρονικό αναγνωριστικό (timestamp) αναλόγως την κλίμακα χρόνου που εργαζόμαστε). Ταυτόχρονοι δράστες μπορούν να μεταλλάσσουν τα περιεχόμενα της βάσης δεδομένων που αποτελεί τη γνώση του συστήματος. Εν συνεχεία, οι κόμβοι του συστήματος μπορούν να διαδίδουν πληροφορίες για τα γεγονότα που "γνωρίζουν" στους γειτονικούς τους κόμβους (gossiping). Έτσι, ο γράφος με τον οποίο μοντελοποιούμε τη μηχανή λογικής, μπορεί να εξελίσσεται στο χωρο-χρόνο με σταδιακές ενημερώσεις στα "κύτταρα" που τον αποτελούν. Αυτή η ιδιότητα έχει τις ρίζες της στην ιδέα του Σχετικιστικού Προγραμματισμού (Relativistic Programming). Ο Σχετικιστικός Προγραμματισμός στηρίζεται σε ιδέες που πηγάζουν από την Ειδική Θεωρία της Σχετικότητας, όπου ένας παρατηρητής μπορεί να παρατηρεί γεγονότα με διαφορετική διάταξη (ordering) από έναν άλλον. Κατά αυτήν την προγραμματιστική πειθαρχεία δε μας ενδιαφέρει να επιλύουμε τις συγκρούσεις που προκύπτουν λόγω ταυτοχρονισμού (π.χ. για αναγνώστες - εγγραφείς), αλλά θέλουμε κάθε παρατηρητής (που μπορεί να μοντελοποιείται φερ' ειπείν από κάποιο νήμα), να έχει μια "σωστή" όψη του κόσμου που παρατηρεί. Αυτό οδηγεί στην ανάπτυξη αλγορίθμων που

υλοποιούνται χωρίς κλειδώματα και στην πλειοψηφία των περιπτώσεων με non-blocking τρόπο.

5.11 Ως Κατανεμημένος Πίνακας Συμβόλων

Χτίζοντας στα παραπάνω, είναι δυνατή η κατασκευή ενός εργαλείου που λειτουργεί σαν Κατανεμημένος Πίνακας Συμβόλων (Distributed Symbol Table). Πίνακες συμβόλων χρησιμοποιούνται στην υλοποίηση μεταγλωττιστών, αλλά και στη δηλωτική σημασιολογία (denotational semantics). Το σύστημα μπορεί να μοντελοποιήσει εύκολα ένα περιβάλλον (environment) που αποθηκεύει τις λεκτικές δεσμεύσεις (bindings) για τη δηλωτική σημασιολογία σε γλώσσες ταυτόχρονου προγραμματισμού που μοντελοποιούνται με το μοντέλο Δράστη. Επίσης, για ένα περιβάλλον Κατανεμημένου Αντικειμενοστρεφούς Προγραμματισμού (Distributed Object-Oriented Programming), μπορεί να παρέχει το στρώμα αποθήκευσης της κατάστασης (state) των αντικειμένων του κόσμου.

6. ΜΕΛΛΟΝΤΙΚΕΣ ΠΡΟΣΘΗΚΕΣ, ΑΝΑΒΑΘΜΙΣΕΙΣ ΚΑΙ ΚΑΤΕΥΘΥΝΣΕΙΣ

6.1 Αναλυτικά

Στις παρακάτω παραγράφους παραθέτουμε βελτιώσεις που μπορούν να γίνουν στο σύστημα CG-Stone. Οι βελτιώσεις αυτές είναι στην πλειοψηφία τους ορθογωνικές μεταξύ τους, δηλαδή ανεξάρτητες μεταξύ τους. Αναφέρονται σε επίπεδο αρχιτεκτονικής, επίπεδο υλοποίησης, αλλά και σε λειτουργικά μέρη του συστήματος. Σκοπός δεν είναι η ανάλυση κάθε πρότασης, μιας και αυτές αποτελούν γενικές προτάσεις - εφαλτήρια για περαιτέρω μελέτη.

Κατ' αρχήν, η στρατηγική κλειδώματος κάδων που ακολουθείται στην υλοποίηση του Ταυτόχρονου Πίνακα Κατακερματισμού (Τ.Π.Κ) για τον ταυτοχρονισμό, επελέγει έχοντας υπόψιν την ύπαρξη πολλών ταυτόχρονων εγγραφέων μαζί με αναγνώστες. Θεωρούμε έτσι, ότι έχουμε ταυτόχρονη πληθώρα μεταλλάξεων στα περιεχόμενα του Τ.Π.Κ. Όμως, στην περίπτωση που η χρήση του συστήματος προορίζεται για περιβάλλοντα όπου οι αναγνώστες είναι αρκετά περισσότεροι από τους εγγραφείς, μια λύση χωρίς κλειδώματα (lock-free) θα ήταν προτιμότερη. Ειδικότερα, μια στρατηγική που ακολουθεί τις αρχές του Σχετικιστικού Προγραμματισμού, παρόμοια με την υλοποίηση Read-Copy-Update (RCU) που εφαρμόζεται στον πυρήνα του λειτουργικού συστήματος Linux και για την οποία υπάρχουν βιβλιοθήκες υλοποίησης σε χώρο χρήστη (userland) εκτός από το χώρο πυρήνα. Αυτή η στρατηγική εγγυάται την περάτωση κάθε λειτουργίας ανάγνωσης σε περιορισμένο αριθμό βημάτων (wait-free αλγόριθμος), χωρίς το υπεράνω κόστος (overhead) που συνεπάγεται η απόκτηση κλειδώματος.

Επί πλέον, στους Τ.Π.Κ. μια ενδιαφέρουσα αλλαγή είναι η γενικοποίηση κλειδιών και τιμών, προσθέτοντας τον τύπο τους και από το

πόσα πεδία αυτός αποτελείται (ως μεταδεδομένα), ώστε στον ίδιο Τ.Π.Κ. να μπορούν να εφαρμοστούν και να συνδυαστούν ποικίλες διαπροσωπείες, παράγοντας νέους τύπους ως αποτελέσματα. Αυτή η κατεύθυνση βέβαια, χρήζει αρκετής προσοχής λόγω της πολυπλοκότητας στο όποιο σύστημα τύπων χρησιμοποιηθεί. Στον αντίποδα όμως, το σύστημα εφοδιάζεται με "πλουσιότερους" τύπους δεδομένων που κάνουν εφικτή τη χρήση του ως μια υπολογιστική μηχανή γενικού σκοπού.

Χρησιμοποιώντας Πίνακες Κατακερματισμού για την αποθήκευση των δεδομένων στην κύρια μνήμη, η αναζήτηση κάποιας τιμής είναι αποδοτική στην περίπτωση που γνωρίζουμε την ακριβή ονομασία του κλειδιού της. Αυτό έχει ως αποτέλεσμα να μην μπορεί κάποιος να κάνει αναζήτηση για οργανωμένα κλειδιά σε κάποια ιεραρχία ή ομάδες κλειδιών με αποδοτικό τρόπο. Με χρήση άλλων δομών δεδομένων, όπως για παράδειγμα λίστες παράλειψης (skip lists) ή προθεματικά δέντρα (radix trees), μπορούμε να έχουμε ιεραρχική οργάνωση που βοηθά στην αναζήτηση. Για τις παραπάνω δομές υπάρχουν υλοποιήσεις χωρίς κλειδώματα και είναι ευρέως διαδεδομένες σε εφαρμογές τέτοιου τύπου. Ακόμη, στην περίπτωση που χρειαζόμαστε λειτουργίες ανάλυσης των αποθηκευμένων δεδομένων (analytics), αποδοτική είναι η οργάνωση κατά στήλες (column-oriented). Η αναπαράσταση αυτή υλοποιείται με στήλες (πίνακες) των οποίων τα περιεχόμενα (γραμμές) συσχετίζονται με κάποιο σχεσιακό μοντέλο σε διάταξη αστέρα (star schema). Με αυτόν τον τρόπο οργάνωσης διευκολύνονται οι λειτουργίες που αναφέρονται σε συνολικά / αθροιστικά μεγέθη (aggregate calculations) χρησιμοποιώντας ανύσματα επιλογής (selection vectors) που υπαγορεύουν ποιες γραμμές συμμετέχουν στους εκάστοτε υπολογισμούς. Επίσης, στην περίπτωση συμπιεσμένης αποθήκευσης επιτυγχάνεται μεγαλύτερο ποσοστό συμπίεσης των δεδομένων, λόγω της ομοιογένειας των περιεχομένων κάθε στήλης. Η οργάνωση κατά στήλες συναντάται συχνά σε εφαρμογές με φορτία που έχουν σχέση με ανάλυση δεδομένων (On-line Analytical Processing (OLAP) systems), όπου όψεις των δεδομένων με κύβους OLAP (OLAP cubes) βοηθούν τους

υπολογισμούς, εν αντιθέσει με εφαρμογές των οποίων τα φορτία αφορούν συναλλαγές (On-line Transaction Processing (OLTP) systems), όπου η αναπαράσταση κατά γραμμές (row-oriented) είναι η συνηθέστερη επιλογή.

Επί προσθέτω, στην περίπτωση υπολογισμών κατά το μοτίβο απεικόνισης-αναγωγής (map-reduce), μια αναπαράσταση με όχι γραμμικές λίστες των κλειδιών που συμμετέχουν στον υπολογισμό είναι προτιμότερη. Πιο συγκεκριμένα, έχουμε:

- *Στάδιο απεικόνισης*: επιλέγεται ένα υποσύνολο των κλειδιών του συστήματος που θα συμμετέχουν στον υπολογισμό και, ενδεχομένως, η τιμή κάθε κλειδιού υπόκειται σε κάποιο μετασχηματισμό.

- *Στάδιο αναγωγής*: τα μετασχηματισμένα κλειδιά συνδυαζόμενα μεταξύ τους, παράγουν το αποτέλεσμα του υπολογισμού. Η προσωρινή αποθήκευση του αποτελέσματος του σταδίου της απεικόνισης (όπως είδαμε και στο κεφάλαιο περί Ετερογενούς Παραλληλισμού) σε δέντρα αναγωγής κάνει εφικτή την παραλληλοποίηση του σταδίου της αναγωγής τηρουμένων κάποιων συνθηκών (που σχετίζονται με τον τελεστή της αναγωγής). Δηλαδή, αν το αποτέλεσμα του σταδίου της απεικόνισης το αποθηκεύσουμε σε μια γραμμική λίστα (cons list - αφηρημένη λίστα με γραμμική οργάνωση) αυτό δυσχεραίνει την παράλληλη αναγωγή. Αντίθετα, αν χρησιμοποιήσουμε μη-γραμμική λίστα (conc list - αφηρημένη λίστα με δενδροειδή οργάνωση), πετυχαίνουμε κατάτμηση των επικείμενων υπολογισμών επί της ουσίας.

Ένα άλλο σημείο που αρχιτεκτονικά είναι απαραίτητο σε ένα τέτοιο σύστημα είναι η υλοποίηση μιας μηχανής αντιγράφων (replication engine). Κατά τη σειριοποίηση των δεδομένων και την αποθήκευσή τους σε κάποιο μέσο ή κάποιο άλλο σύστημα (όχι μόνο στην κύρια μνήμη, της οποίας τα περιεχόμενα δε θεωρούνται μόνιμα), απαραίτητο είναι να υπάρχουν και αντίγραφά τους σε διαφορετικές τοποθεσίες, ώστε η ανάκτηση των δεδομένων σε περίπτωση αστοχίας κάποιου υπο-συστήματος να είναι εφικτή. Επίσης, αυτή η διαδικασία πρέπει να είναι αυτοματοποιημένη, ώστε να μην απαιτεί παρακολούθηση και επέμβαση από κάποιο χρήστη. Υπάρχουν

αρκετές αρχιτεκτονικές λύσεις και έτοιμες υλοποιήσεις για αυτήν τη λειτουργία.

Εξάλλου, η υλοποίηση του συστήματος για την παρούσα εργασία επιδέχεται αρκετές άλλες βελτιστοποιήσεις που σχετίζονται με τη χρήση της γλώσσας προγραμματισμού C, όσον αφορά τα δομικά στοιχεία που αποτελούν το σύστημα. Αυτές οι βελτιστοποιήσεις είναι εκτός εμβέλειας της συγκεκριμένης εργασίας, μιας και στο παρόν στάδιο, έμφαση δίνεται περισσότερο στην εύκολη κατανόηση της λειτουργικότητας και τον ευανάγνωστο κώδικα του συστήματος. Παρ' όλα αυτά και, για συγκεκριμένες πλατφόρμες, μπορεί να αυξήσουν τις επιδόσεις του συστήματος και να αποβούν χρήσιμες για συγκεκριμένα φορτία. Από την άλλη, παράγοντες όπως η υστέρηση που μπορεί να υπάρχει στη δικτυακή επικοινωνία, ενδέχεται να καθιστούν αυτές τις βελτιστοποιήσεις πρακτικά αόρατες στη χρήση του συστήματος.

Μια άλλη κατεύθυνση αφορά την υλοποίηση του συστήματος σε σύστημα με κώδικα bytecode. Κατά αυτήν την υλοποίηση, προβαίνουμε στη χρήση πληθώρας παρεχόμενων προγραμματιστικών πλαισίων από ποικίλες γλώσσες προγραμματισμού, έχοντας ως αποτέλεσμα "ενιαίο" κώδικα bytecode, ικανό να εκτελεστεί σε όσες αρχιτεκτονικές υποστηρίζουν το συγκεκριμένο κώδικα με τη βοήθεια εικονικής μηχανής. Παράδειγμα αποτελεί η Εικονική Μηχανή Java (Java Virtual Machine - JVM). Ειδικότερα, η γλώσσα προγραμματισμού Scala (γλώσσα αντικειμενοστρεφούς και συναρτησιακού προγραμματισμού) αποτελεί συνιστώμενη επιλογή λόγω της συναρτησιακής φύσης της, της ευκολίας που παρέχει στην υλοποίηση γλωσσών DSL (Domain-Specific Languages), των έτοιμων και χρήσιμων βιβλιοθηκών της (π.χ. Akka για την υλοποίηση του μοντέλου Δράστη), της πληθώρας υποστηριζόμενων προγραμματιστικών πλαισίων που αφορούν ανάλυση δεδομένων και της ολοένα αυξανόμενης χρήσης της. Ακόμη, ο συνδυασμός με άλλες γλώσσες που υποστηρίζει η

Εικονική Μηχανή JVM, καθιστά την υλοποίηση του συστήματος πιο εύκολη και εύστοχη.

Στην περίπτωση που δε θέλει κάποιος να χρησιμοποιήσει μια ήδη υπάρχουσα εικονική μηχανή, μπορεί να προβεί στην υλοποίηση μιας τέτοιας, με γνώμονα τις λειτουργίες και την έκφραση των διαπροσωπειών που παρέχει το σύστημα CG-Stone ή αυτών που μπορεί να υλοποιήσει ο χρήστης επεκτείνοντας το σύστημα. Αυτό βοηθά την ανάπτυξη γλωσσών σεναρίων (scripting languages) ή γλωσσών DSL για το εν λόγω σύστημα, συμβάλλοντας στην ευχρηστία του. Μια προτεινόμενη κατεύθυνση είναι η υλοποίηση γλώσσας σεναρίων λογικού προγραμματισμού (logic programming). Η Αφηρημένη Μηχανή της γλώσσας αυτής μπορεί να χρησιμοποιεί την υπάρχουσα υποδομή. Είναι αρκετά ενδιαφέρον τρόπος έκφρασης και λύσης προβλημάτων, ειδικά στην περίπτωση που θέλουμε και μίξη στοιχείων Τεχνητής Νοημοσύνης στο σύστημα. Άλλη κατεύθυνση είναι η υλοποίηση γλώσσας με χαρακτηριστικά συναρτησιακού προγραμματισμού δίνοντας έμφαση στον προθεματικό συμβολισμό, κάτι που τονίζει τον SIMD χαρακτήρα του συστήματος και βοηθά στην παραλληλοποίηση Αλγοριθμικών Σκελετών όπως Απεικόνιση-Αναγωγή.

Βελτιστοποίηση αποτελεί και η συλλογή αιτήσεων για το σύστημα και στη συνέχεια η ομαδική επεξεργασία τους (batch processing). Αυτό υλοποιείται με Δράστες-Συλλέκτες αιτήσεων που ανά τακτά χρονικά διαστήματα ή για κάποιο αριθμό συγκεντρωμένων αιτήσεων, δημιουργούν Δράστες-Παιδιά για την ομαδική επεξεργασία των αιτήσεων. Αποτελεί τακτική διαρρέοντος κάδου (leaky bucket) και βοηθά στην αποδοτικότερη διαχείριση της κίνησης.

Τέλος, μια αρκετά μακροπρόθεσμη κατεύθυνση θα ήταν η ενσωμάτωση του συστήματος σε κβαντικούς υπολογιστές (quantum computers). Η υπέρθεση στις καταστάσεις που βρίσκονται ομάδες πεπλεγμένων κβαντικών bits (entangled qubits) κατά την εκτέλεση ενός

αλγόριθμου και η μέτρηση (και, στην ουσία, δειγματοληψία) που ακολουθεί, αλλάζουν τον τρόπο που σκεφτόμαστε και λύνουμε προβλήματα ως τώρα (με τον κλασικό τρόπο). Το σύστημα CG-Stone με τις κατάλληλες επεκτάσεις, θα μπορούσε να λειτουργήσει χρησιμοποιώντας στο στρώμα αποθήκευσης κβαντικά bits (εκτός των υπόλοιπων κλασικών τιμών). Επίσης, σε μια ευρείας κλίμακας παράταξη (deployment), συλλέγοντας τεράστιο όγκο πληροφορίας, έχοντας υπόψιν ότι πολλά μεγέθη μπορούν να ακολουθούν τον Ισχυρό Νόμο των Μεγάλων Αριθμών, κατά τον οποίο, κάτω από κατάλληλες προϋποθέσεις, ο δειγματικός μέσος μιας ακολουθίας ανεξάρτητων τυχαίων μεταβλητών που ακολουθούν μία κοινή κατανομή συγκλίνει σχεδόν βεβαίως προς τον θεωρητικό μέσο (τη μέση τιμή) της κατανομής, θα μπορούσαμε να αποφανθούμε για την ισχύ αρκετών υποθέσεων που τυγχάνει να κάνουμε.

6.2 Συμπεράσματα

Στην παρούσα εργασία παρουσιάστηκε το σύστημα CG-Stone, το οποίο χρησιμοποιήσαμε για ταυτόχρονη επεξεργασία πολυμεσικού σήματος. Αποτελεί ένα προγραμματιστικό πλαίσιο για την ανάπτυξη εφαρμογών που αφορούν πολυμέσα. Εξάλλου, όπως είδαμε στο παρόν και το προηγούμενο κεφάλαιο, με κατάλληλες προσθήκες μπορεί να χρησιμοποιηθεί σε ευρύ φάσμα περιπτώσεων (όχι μόνο για πολυμέσα). Αρκετές από αυτές τις κατευθύνσεις σκοπεύω να ακολουθήσω σε μελλοντικές αναβαθμίσεις του συστήματος και είμαι σίγουρος ότι θα αποτελέσουν μια συναρπαστική και εξαιρετικά ενδιαφέρουσα αναζήτηση.

7. ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Amdahl (1967). Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities, AFIPS spring joint computer conference 1967.

2. Blelloch (1989). Scans as Primitive Parallel Operations, Carnegie Mellon University.

3. Clinger (1981). Foundations of Actor Semantics, MIT Artificial Intelligence Laboratory, Technical Report 633.

4. Flynn (1972). Some Computer Organizations and Their Effectiveness, IEEE Transactions on Computers (Volume:C-21, Issue: 9), ISSN: 0018-9340.

5. Gonzalez-Velez, Leyton (2010). A Survey of Algorithmic Skeleton Frameworks: High-level Structured Parallel Programming Enablers.

6. Haykin (1995). Συστήματα Επικοινωνίας, Εκδόσεις Παπασωτηρίου, ISBN: 960-7510-18-6.

7. Hillis, Steele (1986). Data Parallel Algorithms, ACM 000t-0782/86/1200-1170.

8. IEEE Std 1003.1c-1995. IEEE Standard for Information Technology - Portable Operating System Interface (POSIX) - System Application Program Interface (API) Amendment 2: Threads Extension (C Language).

9. Intel (2007). Intel SSE4 Programming Reference, http://software.intel.com/file/17971.

10. Jaja (1992). An Introduction to Parallel Algorithms, University of Maryland, ISBN-13: 9780201548563.

11. NVIDIA (2011). CUDA C Programming Guide, http://docs.nvidia.com/cuda/cuda-c-programing-guide/index.html.

12. Proakis, Salehi (2002). Συστήματα Τηλεπικοινωνιών, Έκδοση Ε.Κ.Π.Α, ISBN: 960-8313-04-X.

13. Steele (2009). Organizing Functional Code for Parallel Execution, Steele, ICFP 2009.

14. Vishkin (2010). Thinking in Parallel: Some Basic Data-Parallel Algorithms and Techniques, University of Maryland.

15. Γιαννούκος (2010). Αλγοριθμικές μέθοδοι επεξεργασίας/ανάλυσης πολυμεσικής πληροφορίας και μηχανικής μάθησης, Διδακτορική Διατριβή, Ε.Μ.Π.

16. Καγιάφας, Λούμος (2002). Τεχνολογία Πολυμέσων, Έκδοση Ε.Μ.Π.

17. Παπακωνσταντίνου, Μπιλάλης, Τσανάκας (1999). Λειτουργικά Συστήματα Μέρος Ι: Αρχές Λειτουργίας, Εκδόσεις Συμμετρία, ISBN: 978-960-266-013-3.

18. Σημειώσεις μαθήματος Συστήματα Παράλληλης Επεξεργασίας (2011). Εργαστήριο Υπολογιστικών Συστημάτων Ε.Μ.Π, http://www.cslab.ntua.gr/courses/pps/notes.go.