ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΡΟΣ ΜΙΑ ΜΕΘΟΔΟΛΟΓΙΑ ΓΙΑ ΑΠΟΔΟΤΙΚΗ ΠΑΡΑΛΛΗΛΗ ΥΛΟΠΟΙΗΣΗ ΔΙΕΡΓΑΣΙΩΝ ΑΝΑΛΥΣΗΣ ΕΙΚΟΝΩΝ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

ΤΟΥ

ΑΔΑΜ ΚΩΝ. ΔΑΜΙΑΝΑΚΗΣ

ΑΥΓΟΥΣΤΟΣ 1992

Προς μία μεθοδολογία για αποδοτική παράλληλη υλοποίηση διεργασιών ανάλυσης εικόνων

Αδάμ Κ. Δαμιανάκης

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Αύγουστος 1992

Προς μία Μεθοδολογία για Αποδοτική Παράλληλη Υλοποίηση Διεργασιών Ανάλυσης Εικόνων

Διατριβή

που υποβλήθηκε στο Τμήμα Επιστήμης Υπολογιστών

της Σχολής Θετικών Επιστημών του

Πανεπιστημίου Κρήτης

από τον

Αδάμ Κων. Δαμιανάκη

για την Υποψηφιότητα του Τίτλου

του Διδάκτορος

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Προς μία Μεθοδολογια για Αποδοτική Παράλληλη Υλοποίηση Διεργασιών Ανάλυσης Εικόνων

Διατριβή
που υποβλήθηκε ως μερική απαίτηση για την απόκτηση
του τίτλου του Διδάκτορος από τον

Αδάμ Κων. Δαμιανάκη

Ηράκλειο, 17 Φεβρουαρίου 1992

Ο υποψήφιος Διδάκτορας

Εισηγητική Επιτροπή

Καθηγητής Στέλιος Ορφανουδάκης, Πανεπιστήμιο Κρήτης, Επόπτης

Αν. Καθηγητής Απόστολος Τραγανίτης, Πανεπιστήμιο Κρήτης, Μέλος

Επ. Καθηγητής Κώστας Κουρκουμπέτης, Πανεπιστήμιο Κρήτης, Μέλος

Καθηγητής Γεώργιος Καραγιάννης, ΕΜΠ, Μέλος

Καθηγητής Θεόδωρος Παπαθεοδώρου, Πανεπιστήμιο Πατρών, Μέλος

Γίνεται δεκτή:

Καθηγητής Στέλιος Ορφανουδάκης, Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Ευχαριστίες

Κατά τη διάρκεια των μεταπτυχιακών μου σπουδών είχα την ευκαιρία (και συχνά την τύχη) να γνωρίσω και να συνεργαστώ με αξιόλογους ανθρώπους και ιδιαίτερα σημαντικούς επιστήμονες που έπαιξαν καθοριστικό ρόλο στην ερευνητική μου πορεία.

Ανάμεσα σ'αυτούς, πρώτα από όλους θα ήθελα να ξεχωρίσω και να ευχαριστήσω το σύμβουλο-καθηγητή μου κ. Στέλιο Ορφανουδάκη με τον οποίο συνεργαστήκαμε από τον πρώτο χρόνο που ιδρύθηκε το Τμήμα Επιστήμης Υπολογιστών αλλά και το Εργαστήριο Ανάλυσης Εικόνων. Ολα αυτά τα χρόνια και ειδικά τα πρώτα και δύσκολα, υπήρξε ο ακούραστος συνεργάτης και καθοδηγητής, που με την αυστηρά ακαδημαϊκή, αλλά και συγχρόνως φιλική του συμπεριφορά, με την άκαμπτη απαιτητικότητα, αλλά και την κατανόησή του, έκανε τα προβλήματα λιγότερα και τους στόχους πιο εφικτούς. Μαζί του θα ήθελα να μοιραστώ ό,τι θετικό ή αρνητικό έχει η εργασία αυτή.

Ακόμα, θα ήθελα να ευχαριστήσω τον καθηγητή του τμήματος Επιστήμης Υπολογιστών κ. Κώστα Κουρκουμπέτη, ο οποίος σαν καθηγητής και συνάμα σαν φίλος ήταν πάντα πρόθυμος να συζητήσει κάθε πρόβλημα, να δώσει νέες ιδέες, να δείξει νέες κατευθύνσεις.

Ευχαριστώ επίσης τα υπόλοιπα μέλη της πενταμελούς επιτροπής κρίσης κ.κ. Τραγανίτη Απόστολο, καθηγητή του Παν. Κρήτης, Παπαθεοδώρου Θεόδωρο, καθηγητή του Παν. Πατρών και Καραγιάννη Γεώργιο, καθηγητή του ΕΜΠ για τα εποικοδομητικά τους σχόλια, καθώς και όλους τους καθηγητές του τμήματος Επιστήμης Υπολογιστών για ό,τι μου προσέφεραν είτε κατά τη διάρκεια μαθημάτων είτε από τις συζητήσεις που κατά καιρούς είχαμε. Σημαντική ακόμα υπήρξε η βοήθεια του Πανεπιστημίου του Yale όσο και της Thinking Machines Corporation που διέθεσαν απρόσκοπτα τη χρήση των iPSC/2 και του Connection Machine αντίστοιχα. Ιδιαίτερα μάλιστα ευχαριστώ τον καθηγητή Lennart Johnsson και την ομάδα του για τα εποικοδομητικά τους σχόλια σε θέματα του CM-2. Ακόμα το Δημήτρη Γερογιάννη για την πληροφόρηση που μου διέθεσε αλλά και για τις συζητήσεις που είχαμε στον ένα χρόνο που φιλοξενήθηκα από το Πανεπιστήμιο του Yale. Επίσης θα ήθελα να ευχαριστήσω τον καθηγητή του Πανεπιστημίου της California Dr. Tsutomo Shibata, ο οποίος υπήρξε πάντα ένας πρόθυμος συζητητής, και εποικοδομητικός κριτής, που, εκτός από τη συνεχή πληροφόρηση και το ενδιαφέρον του, δεν έπαυε να μου επαναλαμβάνει πως "*if there is a will, there is a way*".

Ποικιλόμορφη όμως ήταν η συμβολή και πολλών συναδέλφων μεταπτυχιακών φοιτητών στο Πανεπιστήμιο της Κρήτης, και ιδιαίτερα του Κωστή Βεζερίδη με τον οποίο και σχεδιάσαμε ένα πρώτο μοντέλο για το Εμπειρο Σύστημα που αναφέρεται στο 7ο κεφάλαιο, του Αντώνη Αργυρού που υλοποίησε τον αλγόριθμο του τμήματος 6.3.3 αλλά και του Ευριπίδη Πετράκη με τον οποίο εργαστήκαμε επί μακρό χρονικό διάστημα στην ίδια ερευνητική περιοχή. Οι συζητήσεις μαζί τους αλλά και τα σχόλιά τους υπήρξαν ιδιαίτερα χρήσιμα σε όλες τις φάσεις των μεταπτυχιακών μου σπουδών.

Αν κάτι όμως έκανε τα πράγματα πιο ευχάριστα σε καθημερινή βάση αυτά τα χρόνια, ήταν οι συζητήσεις και η παρέα της φίλης και συναδέλφου μου Μαίρης Μαμαλάκη, της πάντα ευχάριστης και πρόθυμης Γραμματέας του Τμήματος Ρένας Καλαϊτζάκη καθώς και της Μαρίας Σταυρακάκη.

Ευχαριστώ επίσης τη Διοίκηση του Πανεπιστήμιου Κρήτης, του Ιδρύματος Τεχνολογίας και Ερευνας και ειδικότερα του Ινστιτούτου Πληροφορικής, που όλα αυτά τα χρόνια μου διέθεσαν ανελλιπώς την "*εκ των ων ουκ άνευ*" υλική υποδομή, καθώς και το προσωπικό τους, που ήταν πάντα πρόθυμο να εξυπηρετήσει σε κάθε διαδικαστικό θέμα.

Ομως ευχαριστώ και όλους εκείνους που δεν είναι δυνατόν να αναφέρω στον περιορισμένο αυτό χώρο και οι οποίοι με το θετικό και σε μερικές περιπτώσεις αρνητικό ρόλο τους, συνέβαλλαν ουσιαστικά στην εργασία αυτή.

Ευχαριστώ ακόμα τους γονείς μου και τα αδέλφια μου για τη συνεχή και ποικιλόμορφη υποστήριξή τους, από τα πρώτα στάδια των σπουδών μου.

Ομως η ολοκλήρωση των ευχαριστιών, δεν μπορεί γίνει παρά με εκείνη προς την Ελενα Πατάκη. Την ικανότατη συνάδελφο αλλά και την ανεκτίμητη φίλη της οποίας οι συμβουλές, η εμπιστοσύνη, η γνώμη και η αγάπη υπήρξαν ανεξάντλητοι τροφοδότες στην πολύμορφη προσπάθειά μου, ιδιαίτερα μάλιστα στα τρία τελευταία και κρισιμότερα χρόνια.

Την αφιερώνω

στην Ελενα

και στους γονείς μου, Κώστα και Πελαγία

Περίληψη

Στην παράλληλη υλοποίηση των διεργασιών ανάλυσης εικόνων διακρίνει κανείς τρείς κύριες συνιστώσες. Συγκεκριμένα την αρχιτεκτονική που χρησιμοποιείται, τη διεργασία που χρειάζεται να υλοποιηθεί και το περιεχόμενο της εικόνας. Μια τέταρτη συνιστώσα, λιγότερο βασική που όμως συχνά επηρεάζει σημαντικά την απόδοση μιας παράλληλης υλοποίησης, είναι ο αλγόριθμος ανακατανομής του υπολογιστικού φορτίου των επεξεργαστών.

Κάθε μία από τις συνιστώσες αυτές χαρακτηρίζεται από ένα πλήθος παραμέτρων που εκφράζουν διάφορα χαρακτηριστικά όπως για παράδειγμα τις επικοινωνιακές και υπολογιστικές επιδόσεις και απαιτήσεις. Οι παράμετροι αυτές των συνιστωσών θα πρέπει να έχουν τις κατάλληλες τιμές προκειμένου να επιτύχουμε αποδοτική παράλληλη υλοποίηση των διεργασιών ανάλυσης εικόνων.

Στην εργασία αυτή προσεγγίζουμε μια μέθοδο για την επιλογή των αποδοτικότερων παράλληλων υλοποιήσεων διεργασιών ανάλυσης εικόνων, δοθέντων των υπολογιστικών και επικοινωνιακών απαιτήσεων διαφορετικών αλγορίθμων για κάθε διεργασία, της κατανομής και αναπαράστασης του περιεχόμενου της εικόνας στα διάφορα επίπεδα ανάλυσής της, καθώς επίσης των επικοινωνιακών και υπολογιστικών χαρακτηριστικών των αρχιτεκτονικών.

Μια τέτοια μέθοδος είναι ιδιαίτερα ενδιαφέρουσα στην περίπτωση που ο χρήστης είναι ελεύθερος να διαλέξει μεταξύ είτε πολλών αλγορίθμων για την εκτέλεση της ίδιας διεργασίας, είτε πολλών παράλληλων μηχανών με διαφορετικά χαρακτηριστικά αρχιτεκτονικής είτε τέλος εναλλακτικών αλγορίθμων ανακατανομής φορτίου σε σχέση πάντα με τα χαρακτηριστικά και την κατανομή του περιεχόμενου των εικόνων πάνω στην δομή που τις αναπαριστάνει.

Προκειμένου να επαληθεύσουμε τις υποθέσεις μας και τις προβλέψεις της μεθόδου που προτείνουμε, πειραματιστήκαμε με υλοποιήσεις διεργασιών ανάλυσης εικόνων που εμπίπτουν στην κατηγορία ανάλυσης μέσου επιπέδου. Οι διεργασίες που επιλέξαμε έχουν χαρακτηριστικά που τις καθιστούν αντιπροσωπευτικές μιας ευρύτερης ομάδας διεργασιών ανάλυσης εικόνων. Για τις διεργασίες αυτές υλοποιήθηκαν διάφοροι αλγόριθμοι σε δύο αρχιτεκτονικές κατανεμημένης μνήμης με σημαντικές διαφορές στα επικοινωνιακά και υπολογιστικά χαρακτηριστικά τους. Η πρώτη είναι η Connection Machine, μιά SIMD αρχιτεκτονική με χιλιάδες αδύνα-

τους επεξεργαστές, ενώ η δεύτερη είναι ο iPSC/2 της INTEL, ένας MIMD υπερκύβος με 64 ισχυρούς επεξεργαστές.

Παρουσιάζει ενδιαφέρον ο διαφορετικός τρόπος με τον οποίο το κάθε χαρακτηριστικό του περιεχομένου της εικόνας επιδρά στην απόδοση των αλγορίθμων αλλά και το γεγονός ότι χαρακτηριστικά με σημαντικές επιπτώσεις σ'έναν αλγόριθμο ελάχιστα επηρεάζουν άλλους. Το γεγονός αυτό μας επιτρέπει, γνωρίζοντας το περιεχόμενο μιας εικόνας, να μπορούμε να κάνουμε εκ των προτέρων επιλογή του αλγορίθμου που θα χρησιμοποιήσουμε, μεταξύ αλγορίθμων για την ίδια διεργασία. Διαμορφώνονται ακόμα κλειστοί τύποι, οι οποίοι, υπό συγκεκριμένες συνθήκες, μπορούν να εισηγούνται ποια αρχιτεκτονική και ποιός αλγόριθμος αντίστοιχα, είναι εκείνοι με την καλύτερη αναμενόμενη απόδοση. Και στις δύο περιπτώσεις τα αναλυτικά αποτελέσματα μπορούν να χρησιμοποιηθούν για να γίνει διαμερισμός του παραμετρικού χώρου (parametric space) που αντιπροσωπεύει την απόδοση διαφόρων αλγορίθμων ή αρχιτεκτονικών αντίστοιχα. Προβλέψεις που μπορούν να γίνουν με βάση την ανάλυση, επαληθεύονται από τα πειραματικά αποτελέσματα.

Τα συμπεράσματα που βγαίνουν από τα πειραματικά και αναλυτικά αποτελέσματα μας δημιουργούν την πεποίθηση πως είναι δυνατό να μπει τάξη στη χαοτική φύση του προβλήματος και ότι είμαστε σε θέση να παίρνουμε αποφάσεις βασισμένες σε ένα όχι μεγάλο αριθμό παραγόντων, τους οποίους θα γνωρίζουμε ή για τους οποίους μπορούμε να κάνουμε υποθέσεις, με δραστικές τελικά επιπτώσεις στην επίδοση των παράλληλων συστημάτων στο συγκεκριμένο πεδίο εφαρμογών.

Abstract

The efficient implementation of intermediate level image analysis tasks on parallel architectures constitutes a difficult problem which requires careful consideration of the computational characteristics and communication requirements of such tasks, the resources provided by different parallel architectures, as well as the dependence of load distribution at different levels of image analysis on image content. Specifically, optimum performance is difficult to achieve for three main reasons: 1) Imperfect match of the task graph with the topology of the parallel machine used, resulting in large communication overhead; 2) Unbalanced computational load on the processors; and 3) The degree to which the algorithm used to implement a particular task is inherently sequential.

An important consideration at the intermediate level of image analysis is the data reduction that characterizes tasks at this level and the different data structures needed to represent intermediate results at each processing step, while processing shifts to information-dense regions of interest in the image array. These regions depend on image content and result in unbalanced load distribution.

Developing a general methodology for the efficient implementation of image analysis tasks on parallel architectures is particularly important when the user is free to choose among many algorithms for the execution of the same task, parallel machines with different architectural features, and alternative algorithms for balancing the image content dependent computational load.

The components which play a major role in parallel image analysis are the *architecture, image content, imaging task* and *load balancing*. These components are not independent of each other and interdependencies are very important in any attempt at optimizing performance. With regard to load balancing, the basic question is whether it is expected to improve performance or not. In general, the answer depends on the computational and communication overhead of the load redistribution algorithm, the expected optimization of load distribution, and the dependence of the algorithm on the architecture, image content and task characteristics. Finally, one must define computational load for each task and determine how to best redistribute it without incurring substantial computational and

communication overhead.

Objectives

The primary goal of this work is the development of a methodology for the efficient parallel implementation of image analysis tasks given the computational and communication requirements of different algorithms, the content-dependent load distribution and representation at each level of image analysis, and specific architectural features. As a first step towards this goal, we describe the basic features of each component. Subsequently, we describe a model for selecting the best available parallel implementation of an imaging task. The model suggests an appropriate match between algorithms and architectures, given a parametric characterization of image content. Thus, based on image content, one may decide to implement a particular task on parallel architectures with significantly different characteristics and using algorithms which are properly matched to the architecture of choice.

Outline of the Thesis

In the first chapter, selected research results from biology and physiology are considered to argue that the human visual system uses parallel processing.

The second chapter introduces the key issues related to parallel processing, describes the objectives of this work and concludes with an outline of the thesis.

In the third chapter we present issues related to parallel image processing, and discuss the main features of the different levels of image analysis. We focus our attention on intermediate level image processing, which plays the role of bridging the gap between the low and the higher level. This focus is due to the special issues and problems that arise at this level, such as the dynamic change in the data structures needed, data reduction, content driven processing, and load balancing considerations. An extended review of the literature on the parallel architectures and systems related to parallel image analysis, parallel algorithms for

several tasks, and the problem of load balancing and task partitioning, is also provided.

In the fourth chapter, the four components of parallel image processing and their most important features are described. Architectural features include the topology, power and memory capacity of the processors, the communication features of the interconnection network, and available computational and communication primitives. Image content can be characterized in terms of important features, their quantity, magnitude, and spatial distribution over the image. In addition, one must consider the data structures needed to represent these features in a dynamic sense, while processing is in progress. An imaging task is characterized by its complexity and parallelization, as well as the data structures and communication pattern(s) required.

In the fifth chapter we give a general description of a model for the choice of the best available implementation of an imaging task. This choice is based on the parametric representation of the components of parallel image analysis. The system takes into account their interaction and influence on the overall performance. The architecture of this model is depicted in Figure 5.1. It is composed of three main parts: the first part requires the user to supply the system with relevant information regarding the task that is to be executed, information about the specific image or the class of images to which the task is to be applied, as well as the parallel system(s) that is (are) to be used. The second part consists of three knowledge bases containing statistical information about image features, information about all available algorithms, and information about available architectures respectively. The third part supports decisions related to balancing the load when overall performance is expected to improve and selecting an appropriate load balancing algorithm. These parts are linked in such a way that a decision tree can be used to select the best possible implementation.

In order to illustrate and evaluate this approach, in the sixth chapter we present results from its application on two common imaging tasks with different computational and communication requirements. These are the computation of the convex hull of a set of pixels and connected component labeling. We consider these tasks to be representative of a wide class of imaging tasks for which pixels are the basic data units, while being characterized by homogeneous communication, extended parallelization, and the requirement for the exact

location of each pixel to be known to processors other than the one holding it. On the other hand, the computation of the convex hull is different from the task of connected component labeling in that it consists of two-phases, each requiring different data structures and communication patterns, and involves more computation per data unit. These differences between the two tasks extend the generality of our conclusions.

Implementations of the above tasks have been carried out using algorithms designed to exploit their computational and communication characteristics as well as the built-in primitives of each parallel architecture. In each case, performance has been measured with respect to the complexity of image content, the granulatity of image partitions, and relevant architectural features. In this chapter it is also demonstrated that our approach can be applied to other image analysis tasks and the required analytical expressions of expected performance are derived.

In the seventh chapter we state the conclusions of our work and suggest possible directions of future research. A general description of an expert system that would take into account the experience gained from previous implementations involving a variety of tasks, algorithms and architectures, and would suggest the best possible implementation for a new set of conditions is also provided.

In the Appendix, the main features of the two architectures that have been used in our experiments (the CM-2 of Thinking Machines Corporation and iPSC/2 of Intel), are described and detailed benchmarks, obtained for the purpose of verifying the analytical predictions, are presented.

Conclusions

In this thesis, we study issues and problems which arise in parallel image analysis. Intermediate level vision is emphasized and solutions to selected problems are provided. Analytical and experimental results on the performance of specific parallel implementations of characteristic intermediate level vision tasks have been obtained, compared and used to find a proper match between algorithms and architectures, given a parametric character-

ization of image content. Thus, based on image content, one may decide to implement a particular task on parallel architectures with significantly different characteristics. It is expected that similar analysis of many other intermediate level vision tasks and related algorithms will lead to the development of a more general methodology for efficient parallel implementations of such tasks. However, such a methodology would be useful from a practical point of view if it could be applied to parallel implementations of image analysis tasks on reconfigurable architectures which support coarse and fine grain computations, the efficient embedding of different communication patterns, and a variety of different data structures, as well as efficient transformations of one data structure to another.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Ανθρώπινη Οραση και Παράλληλη Επεξεργασία

Η ανθρώπινη όραση είναι μια εξαιρετικά πολύπλοκη διεργασία με πλήθος από ανεξερεύνητες πτυχές. Εφοδιασμένη με ισχυρούς μηχανισμούς για την ταχύτατη εκτέλεση τόσο των απλών πράξεων όσο και σύνθετων υπολογισμών, έχει αποτελέσει αντικείμενο εκτεταμένων ερευνών ενώ έχει εμπνεύσει ποικίλες θεωρίες και έχει ενεργοποιήσει προσπάθειες προσομείωσής της με ικανοποιητικά πολλές φορές, αλλά όχι εξίσου τέλεια αποτελέσματα. Ιδιαίτερα φαίνεται να υπάρχει μια ισχυρή αλληλεπίδραση μεταξύ της Επιστήμης των Υπολογιστών, της Φυσιολογίας και των Νευροεπιστημών στο χώρο της έρευνας. Η πρώτη φαίνεται να προτείνει θεωρητικά μοντέλα για τον τρόπο λειτουργίας του εγκεφάλου και γενικότερα της ανθρώπινης αντίληψης στη Φυσιολογία και τις νευροεπιστήμες, ενώ οι τελευταίες τροφοδοτούν την Επιστήμη των Υπολογιστών με νέες ιδέες για αρχιτεκτονικές και τρόπους προγραμματισμού.

Αρκετοί ερευνητές στο χώρο της Φυσιολογίας υποστηρίζουν πως δεν μπορεί να υπάρξουν υπολογιστικά συστήματα, όσο έξυπνα και να είναι, που να μπορέσουν να έχουν την ίδια αποτελεσματικότητα με το ανθρώπινο μυαλό. Στηρίζουν την άποψη αυτή στο ότι η πολύπλοκη λειτουργία του στηρίζεται στην ικανότητα που έχει να επεξεργάζεται τεράστιες ποσότητες πληροφοριών με διαφορετικούς τρόπους παράλληλα. Ευρήματα από έρευνες που έχουν γίνει στην περιοχή της ανθρώπινης όρασης έχουν δείξει πως το ανθρώπινο σύστημα δεν αποτελείται από

ένα απλό ιεραρχικό σύστημα, αλλά είναι διαχωρισμένο σε τουλάχιστον τρία διαφορετικά υποσυστήματα, κάθε ένα από τα οποία επιτελεί διαφορετικές διεργασίες. Ενα από αυτά ειδικεύεται στην αναγνώριση σχημάτων, άλλο στην επεξεργασία των χρωμάτων ενώ το τρίτο ευαισθητοποιείται και επεξεργάζεται την κίνηση, τη χωρική κατανομή των αντικειμένων και την απόστασή τους από τον παρατηρητή. Ηδη από τα μέσα του 1800 ήταν γνωστό πως το κεντρικό οπτικό νεύρο διαχωρίζεται σε διάφορα υποτμήματα μετά την έξοδό του από τον οφθαλμό [1].

Ομως πολλά ερωτηματικά εγείρονται καθώς διερευνά κανείς τους μηχανισμούς της ανθρώπινης αντίληψης. Στην [2] για παράδειγμα, παρατίθεται ένα πλήθος από πειράματα που έχουν γίνει κατά καιρούς από ερευνητές στο χώρο της ανθρώπινης αντίληψης. Πρόκειται για πειράματα με ιδιαίτερο ενδιαφέρον διότι καταδεικνύουν τον περίεργο (και μέχρι τώρα ανεξήγητο) τρόπο με τον οποίο αντιλαμβανόμαστε. Χαρακτηριστικά παραδείγματα είναι τα τρίγωνα του Kanizsa, που τα βλέπουμε ενώ στην πραγματικότητα δεν υπάρχουν, το πλέγμα του Ehrenstein όπου εμφανίζονται μικροί κύκλοι εκεί όπου συμβάλλουν γραμμές που δεν ακουμπούν μεταξύ τους, το "δρόμο" του Ehrenstein, τα σχήματα του Fraser κτλ. Διάφορες θεωρίες έχουν διατυπωθεί κατά καιρούς προσπαθώντας να εξηγήσουν το γιατί βλέπουμε τέτοια ανύπαρκτα σχήματα. Μία άποψη στηρίζεται στην εντύπωση που δημιουργεί η αντίθεση της φωτεινότητας ειδικά στις γωνίες. Αλλη εξήγηση σχετίζεται με την αλληλεπίδραση των επιμέρους μηχανισμών του οπτικού συστήματος που ανιχνεύουν, ο καθένας χωριστά -και παράλληλα όλοι μαζί- ακμές, γραμμές, γωνίες κτλ. Μια τρίτη άποψη ισχυρίζεται πως το ανθρώπινο μυαλό προσπαθεί να συσχετίσει τα νέα περίεργα σχήματα που βλέπει με άλλα που ήδη γνωρίζει. Την τελευταία αυτή άποψη μπορούμε να τη συσχετίσουμε με τη θεωρία που διατύπωσε ο Marr το 1969 [3] και η οποία αναφέρει πως η απλή και κανονική δομή του φλοιού του εγκεφάλου (cortical structure) μπορεί να θεωρηθεί μια απλή αλλά ισχυρή μηχανή απομνημόνευσης που βοηθάει τη μάθηση, διότι κάθε ένα από τα 15 εκατομμύρια κύτταρα Purkinje στον εγκέφαλο (cerebellum) είναι σε θέση να μάθει μέχρι και 200 διαφορετικά σχήματα (patterns) και να ξεχωρίζει κάθε ένα από αυτά από σχήματα που δεν γνωρίζει.

Από το 1953 ο Barlow είχε βρει πως ορισμένα γάγγλια στους βατράχους παίζουν το ρόλο αποκλειστικά του "διερευνητή εντόμων" αντιδρώντας σε ό,τι μοιάζει με ένα κινούμενο μικρό μαύρο δίσκο. Και δεν είναι ο μοναδικός ερευνητής που διαπίστωσε πως υπάρχουν γάγγλια που αντιδρούν πάντα (και μόνο) στο ίδιο ερέθισμα. Ο ίδιος πάντως ολοκλήρωσε αυτά τα ευρήματά

του διατυπώνοντας το 1972 την άποψη ότι οι νευρώνες (neurons) όχι μόνο αναπαριστούν τη φωτεινότητα που τους ερεθίζει αλλά εκτελούν από μόνοι τους και άλλες πολύ πιο πολύπλοκες διεργασίες, όπως αναγνώριση σχημάτων (patterns), διάκριση του βάθους κά. Ακόμα, αγνοούν τα άσχετα ερεθίσματα και είναι τακτοποιημένοι σε μια μυστηριώδη ιεραρχία. Ο ίδιος διατύπωσε 5 δόγματα [4] το πρώτο από τα οποία αναφέρει: *Η περιγραφή της δράσης ενός νευρικού κυττάρου που μεταδίδεται σε άλλα κύτταρα και τα επηρεάζει, καθώς και η απόκρισή του σε τέτοιου είδους επιρροές από άλλα, είναι μια πλήρης περιγραφή της λειτουργίας της αντίληψης από το νευρικό σύστημα. Δεν υπάρχει τίποτα άλλο που να παρακολουθεί ή να ελέγχει αυτή τη λειτουργία, η οποία και θα πρέπει να θεωρείται η βάση για την κατανόηση του μηχανισμού με τον οποίο το μυαλό ελέγχει τη συμπεριφορά.*

Εντούτοις, όλες αυτές οι θεωρίες που διατυπώνονται δεν έχουν καταφέρει να εξηγήσουν πλήρως τους μηχανισμούς της ανθρώπινης αντίληψης. Κάτι τέτοιο όμως είναι απαραίτητο αν θέλουμε να δημιουργήσουμε μηχανικά συστήματα που θα την προσομοιώνουν ικανοποιητικά. Διότι δεν θα πρέπει να περιμένουμε να το επιτύχουμε αν τα χαρακτηριστικά των μηχανών που κατασκευάζουμε, αλλά και ο τρόπος που τα χρησιμοποιούμε δεν είναι όμοιος με εκείνον της φύσης. Βλέπουμε όμως πως αυτό που σχηματοποιείται από τις προηγούμενες θεωρίες που αναφέρθηκαν είναι ένα Ιεραρχικό σχήμα επεξεργασίας των πληροφοριών του περιβάλλοντος, όπου σε κάθε επόμενο επίπεδο της ιεραρχίας αυτής χρησιμοποιούνται λιγότεροι αλλά περισσότερο ισχυροί επεξεργαστές. Ειδικά στο πρώτο επίπεδο υπάρχουν εκατομμύρια μικροί "επεξεργαστές", κάθε ένας από τους οποίους έχει μεγάλη μνήμη ενώ είναι αρκετά "έξυπνοι" ώστε να ερεθίζουν και να αντιδρούν σε ερεθίσματα. Η συλλογή των πληροφοριών (data acquisition) γίνεται κατευθείαν από αυτά ενώ λειτουργούν παράλληλα, χωρίς ωστόσο να είναι στο πρώτο επίπεδο επεξεργασίας απαραίτητη η ιεραχική δομή και ο συγκεντρωτικός έλεγχός τους.

Μια πολύ εμπεριστατωμένη ανασκόπηση της εξέλιξης των ευρημάτων που σχετίζονται με τη λειτουργία της ανθρώπινης αντίληψης μπορεί κανείς να δει στο βιβλίο "Vision" του Marr [5], πολλά στοιχεία από το οποίο συμπεριελήφθησαν στο κεφάλαιο αυτό.

Ο B. Julesz [6] προκειμένου να πειραματιστεί στη λειτουργία της στερεοψίας (Stereo vision) δημιούργησε στερεογράμματα με κηλίδες σε τυχαίες θέσεις. Τα πειράματα αυτά οδήγησαν στο συμπέρασμα πως η ανάλυση της πληροφορίας για τη στερεοψία όπως και η ανάλυση των ερεθισμάτων από την κίνηση αντικειμένων μπορούν να γίνονται ανεξάρτητα, ακόμα και

με απουσία άλλης σχετικής πληροφορίας. Η σπουδαιότητα του συμπεράσματος αυτού, όπως αναφέρει ο Marr [5], είναι ότι μας δίνει τη δυνατότητα να διαχωρίσουμε τη μελέτη της ανθρώπινης αντίληψης σε επιμέρους τμήματα, γεγονός που μας επιτρέπει να πιστεύουμε στη σημασία της παράλληλης επεξεργασίας. Ομως η προσομοίωση της ανθρώπινης όρασης πρέπει να γίνει όχι μόνο στον τομέα της αρχιτεκτονικής αλλά και στους αλγόριθμους που χρησιμοποιεί ο ανθρώπινος εγκέφαλος για να επεξεργαστεί την πληροφορία που συλλέγει.

Οι Feldeman και Ballard διατύπωσαν πρώτοι τον "κανόνα των 100 βημάτων" που λέει πως *αφού οι νευρώνες λειτουργούν με συχνότητα περίπου 1000Hz και αφού τα απλά φαινόμενα που αντιλαμβανόμαστε συμβαίνουν σε χρόνο περίπου 100 milliseconds, τότε οι 'βιολογικοί αλγόριθμοι' δεν πρέπει να περνούν τα 100 βήματα*" [7]. Πρόσφατα, ο Tsotsos [8], ξεκινώντας από την ανάλυση της πολυπλοκότητας που απαιτούν οι διεργασίες που γίνονται προκειμένου να αντιληφθούμε διάφορα φαινόμενα γύρω μας, οδηγήθηκε στη διατύπωση περιορισμών για την αρχιτεκτονική τόσο του βιολογικού "εξοπλισμού" όσο και των υπολογιστικών συστημάτων που χρησιμοποιούνται για να προσομειώσουν τον πρώτο. Διατυπώνει δε τη θέση πως εκτός από το υπολογιστικό επίπεδο ανάλυσης θα πρέπει να θεωρούμε και το επίπεδο πολυπλοκότητας (Complexity level) ενώ πρέπει να ικανοποιείται στις προσεγγίσεις μας πάντα ο κανόνας "Μέγιστη Ισχύς με Ελάχιστο Κόστος".

Ενώ στη Μηχανική Οραση (Computer Vision) η αναγνώριση του χώρου στηρίζεται κατά κανόνα στη μοντελοποίησή του, ο άνθρωπος έχει τη δυνατότητα να αναγνωρίζει αντικείμενα που δεν τα έχει δει στο παρελθόν σε κλάσματα δευτερολέπτου, χρόνος που αντιστοιχεί σε μερικούς κύκλους επεξεργασίας του νευρικού του συστήματος. Εχει αποδειχτεί με πειράματα πως η αναγνώριση αυτή εξαρτάται σημαντικά από την προσδοκία και το γενικότερο περιεχόμενο (context) μέσα στο οποίο γίνεται η αναγνώριση αυτή. Ο Rosenfeld [9] δίνοντας έμφαση στη γεωμετρική περιγραφή των αντικειμένων κάνει ορισμένες εικασίες για τον τρόπο λειτουργίας του μηχανισμού αναγνώρισης αντικειμένων στον άνθρωπο και καταλήγει προτείνοντας ένα υπολογιστικό μοντέλο, που θα μπορούσε να κάνει ικανοποιητική αναγνώριση μη αναμενόμενων αντικειμένων (unexpected objects), εάν υλοποιηθεί σε κατάλληλες αρχιτεκτονικές με παράλληλο χαρακτήρα τις οποίες και προδιαγράφει.

Αν και υπάρχει, λοιπόν, η άποψη πως "*χρόνος είναι ο τρόπος που έχει η Φύση να εμποδίζει όλα τα γεγονότα να συμβαίνουν ταυτόχρονα*" [10] υπάρχουν πάρα πολλές περιπτώσεις που

η ίδια η φύση κάνει το αντίθετο, προκειμένου να εκτελέσει τις πιο πολύπλοκες διεργασίες της, ενώ ο άνθρωπος καταβάλλει κάθε προσπάθεια για να τη μιμηθεί εφαρμόζοντας αυτό που αποκαλούμε *Παράλληλη Επεξεργασία*.

Κεφάλαιο 2

Εισαγωγή

Σήμερα οι υπολογιστές καλούνται να υποστηρίξουν ένα ευρύ φάσμα εφαρμογών με αυξημένες απαιτήσεις σε υπολογιστική ισχύ. Τέτοιες είναι η Μηχανική Οραση, η Ρομποτική, η Διαγνωστική Ιατρική και άλλες εφαρμογές πραγματικού χρόνου (real time). Συχνά οι απαιτήσεις ταχύτητας των εφαρμογών αυτών είναι μερικές τάξεις μεγέθους μεγαλύτερες από την ικανότητα των σειριακών μηχανών, γεγονός που οδήγησε στη διέξοδο της Παράλληλης Επεξεργασίας, δηλαδή της ταυτόχρονης ενασχόλησης πολλών επεξεργαστών με την επίλυση του ίδιου προβλήματος.

Η Παράλληλη Επεξεργασία μπορεί να γίνει είτε χωρίζοντας τα δεδομένα σε κομμάτια και δίνοντας στον κάθε επεξεργαστή να επεξεργαστεί ένα από αυτά (παραλληλία ως προς τα δεδομένα), είτε χωρίζοντας τον αλγόριθμο σε λειτουργικώς ανεξάρτητα τμήματα και δίνοντας στον κάθε επεξεργαστή να τρέξει ένα από αυτά πάνω στα (όχι κατ'ανάγκη ίδια) δεδομένα (παραλληλία ως προς τον κώδικα). Το δεύτερο προϋποθέτει πως ο αλγόριθμος αποτελείται από τμήματα που από τη φύση τους μπορούν να εκτελεστούν παράλληλα. Η πρώτη μέθοδος εφαρμόζεται σε περιπτώσεις που έχουμε μεγάλες ποσότητες δεδομένων πάνω στα οποία εκτελούνται οι ίδιες πράξεις, όπως για παράδειγμα στη χαμηλού επιπέδου επεξεργασία ψηφιακών εικόνων, ενώ η δεύτερη στην περίπτωση που έχουμε πολύπλοκο αλγόριθμο με απαιτήσεις για μεγάλη υπολογιστική δύναμη και μικρές ποσότητες δεδομένων.

Πολύ συχνά αυτή η διάσπαση του προβλήματος εισάγει ένα πρόσθετο κόστος (overhead),

το οποίο όχι μόνο δεν είναι πάντα αμελητέο, αλλά μερικές φορές είναι απαγορευτικό για τη χρήση της παραλληλίας. Ετσι, δημιουργείται η ανάγκη να υπάρξει ένα μέτρο της αποτελεσματικότητας ενός παράλληλου συστήματος και γι'αυτό υπάρχουν δύο βασικά ποσοτικά μεγέθη. Το πρώτο είναι η *Επιτάχυνση* (Speedup), που εκφράζει το λόγο του χρόνου που χρειάζεται ο βέλτιστος γνωστός σειριακός αλγόριθμος προς το χρόνο επεξεργασίας στην παράλληλη υλοποίηση. Δηλαδή πόσο γρηγορότερα θα γίνει η επεξεργασία αν την υλοποιήσουμε παράλληλα. Το δεύτερο μέγεθος είναι η *Απόδοση* (Efficiency), που ισούται με την επιτάχυνση προς το πλήθος των επεξεργαστών και εκφράζει το πόσο καλή εκμετάλλευση της παράλληλης μηχανής έγινε για ένα συγκεκριμένο πρόβλημα. Στην ιδανική περίπτωση η επιτάχυνση είναι ίση με τον αριθμό των επεξεργαστών και η απόδοση ίση με τη μονάδα. Οι παραπάνω βέλτιστες επιδόσεις στις περισσότερες περιπτώσεις δεν είναι δυνατόν να επιτευχθούν για διάφορους λόγους, οι κυριότεροι των οποίων είναι: 1) η ακατάλληλη προσαρμογή του γράφου του προβλήματος στο γράφο της τοπολογίας του συστήματος που έχει σαν συνέπεια καθυστερήσεις στο δίκτυο επικοινωνίας, 2) η άνιση κατανομή φόρτου εργασίας στους επεξεργαστές και 3) ο βαθμός στον οποίο η ίδια η διεργασία είναι εγγενώς σειριακή.

Ο νόμος του Amdahl [11] προσδιορίζει πως σε κάθε αλγόριθμο υπάρχουν τμήματα τα οποία πρέπει να εκτελεστούν σειριακά. Γι'αυτό το λόγο η παράλληλη επεξεργασία μπορεί να φτάσει χρόνο εκτέλεσης (αν αθροίσουμε τους χρόνους εκτέλεσης σε όλους τους επεξεργαστές της παράλληλης μηχανής) όχι λιγότερο από το χρόνο της σειριακής εκτέλεσης όλου του αλγορίθμου σε ένα σειριακό επεξεργαστή. Προκειμένου να χαρακτηρίσουμε την παραλληλοποιησιμότητα ενός αλγορίθμου, θα πρέπει να τον αναλύσουμε σε διάφορα επίπεδα διαμέρισης ως προς τη μαζικότητα του αποτελέσματος. Τα κυριότερα επίπεδα διαμέρισης είναι α) το επίπεδο εργασίας (job) β) διεργασίας (task) γ) επιμέρους διεργασίας (process) δ) εντολής (instruction) ε) μικροεντολής (microinstruction) στ) διαχείρισης εκχωρητών (register transfer) και τέλος ζ) το χαμηλότερο επίπεδο του λογικού σχεδιασμού (logic devices).

Μια άλλη μορφή Παράλληλης Επεξεργασίας είναι η Κατανεμημένη Επεξεργασία (Distributed Processing) η οποία αναφέρεται σε παραλληλοποίηση του συστήματος σε επίπεδο εργασίας ή διεργασίας με χρήση κατά κανόνα πολλών υπολογιστών που βρίσκονται πάνω σε ένα δίκτυο. Στη διατριβή αυτή θα αναφερόμαστε μόνο στην Παράλληλη Επεξεργασία.

Είναι προφανές πως κάθε προσπάθεια για βελτιστοποίηση της απόδοσης του Συστήμα-

τος θα είχε σαν προϋπόθεση τον ιδανικό χειρισμό των παραμέτρων και των συνιστωσών που συνθέτουν το πρόβλημα. Ετσι για παράδειγμα, το σύστημα επικοινωνίας θα πρέπει να μπορεί να ανταποκριθεί ικανοποιητικά στις απαιτήσεις για τη μετακίνηση των αναγκαίων ποσοτήτων πληροφοριών, θα πρέπει να γίνεται επιλογή των κατάλληλων τοπολογιών για κάθε κατηγορία προβλημάτων, θα πρέπει να μεγιστοποιείται η παραλληλοποίηση του αλγόριθμου και να υπάρχουν οι κατάλληλοι μηχανισμοί με τους οποίους θα επιτυγχάνεται η ισοκατανομή στα φορτία των επεξεργαστών ενώ τέλος θα πρέπει να χρησιμοποιούνται οι κατάλληλες δομές δεδομένων. Με τα προβλήματα αυτά ασχολείται η διατριβή αυτή, ο αντικειμενικός σκοπός της οποίας παρουσιάζεται στην επόμενη παράγραφο.

2.1 Αντικείμενο της έρευνας αυτής

Η παράλληλη υλοποίηση διεργασιών δικαιώνεται μόνον εφόσον επιτύχει να βελτιώσει το χρόνο της υλοποίησης σημαντικά, σε σχέση με τους πόρους (resources) που διατέθηκαν για το σκοπό αυτό, αλλά και σε σχέση με την ταχύτητα που απαιτείται για τη συγκεκριμένη εφαρμογή.

Δεν είναι λίγες οι φορές που χρήστες καταβάλλουν σημαντική προσπάθεια προκειμένου να επιτύχουν έστω και μικρή βελτίωση της απόδοσης του παράλληλου συστήματος, που χρησιμοποιούν για μια συγκεκριμένη εφαρμογή, και αντίστοιχη αύξηση της επιτάχυνσης στην επεξεργασία. Διαθέτουν δε για το σκοπό αυτό όλο και περισσότερους πόρους με ανάλογη επιβάρυνση στο κόστος του συστήματος. Είναι μάλιστα αξιοσημείωτο πως πολύ συχνά τόσο η υπέρμετρη προσπάθεια όσο και το αυξημένο κόστος δεν επιφέρουν τα αναμενόμενα αποτελέσματα.

Το πρόβλημα της βελτίωσης της απόδοσης ενός τέτοιου συστήματος είναι δύσκολο από μόνο του, εκείνο όμως που το κάνει ακόμα πολυπλοκότερο, είναι η δυναμική μετάλλαξη των συνθηκών που επικρατούν κατά τη διάρκεια παράλληλων υλοποιήσεων σε συγκεκριμένες εφαρμογές. Τα δεδομένα αλλάζουν, εξελίσσονται, διαμορφώνονται δυναμικά. Οι διεργασίες που πρέπει να εφαρμοστούν πάνω σ'αυτά διαδέχονται η μία την άλλη διαμορφώνοντας ένα ακόμα πιο έντονα εξελισσόμενο περιβάλλον.

Ο κύριος στόχος της εργασίας αυτής είναι η ανάπτυξη μιας μεθοδολογίας για την αποδοτική παράλληλη υλοποίηση διεργασιών ανάλυσης εικόνων, βοηθώντων των υπολογιστικών και

επικοινωνιακών απαιτήσεων διαφορετικών αλγορίθμων, της κατανομής και αναπαράστασης του περιεχόμενου της εικόνας στα διάφορα επίπεδα ανάλυσής της καθώς επίσης και των ειδικών χαρακτηριστικών των αρχιτεκτονικών.

Σαν πρώτο βήμα συγκεκριμενοποιούμε τις τέσσερις συνιστώσες που συνθέτουν το πρόβλημα της παράλληλης επεξεργασίας εικόνων και περιγράφουμε τα βασικά χαρακτηριστικά τους. Οι συνιστώσες αυτές είναι: η *αρχιτεκτονική* που θα χρησιμοποιηθεί, η *διεργασία* που θέλουμε να υλοποιήσουμε, το *περιεχόμενο* της εικόνας πάνω στην οποία θα εφαρμόσουμε τη διεργασία και τέλος η *ανακατανομή* του υπολογιστικού φορτίου πάνω στους επεξεργαστές.

Στη συνέχεια παρουσιάζουμε ένα μοντέλο για την επιλογή της βέλτιστης παράλληλης υλοποίησης, η λειτουργία του οποίου στηρίζεται στην παραμετροποιημένη αναπαράσταση των συνιστωσών που αναφέρθηκαν νωρίτερα. Περιγράφουμε τα επιμέρους τμήματά του και εξηγούμε τον τρόπο λειτουργίας του.

Μια τέτοια μέθοδος είναι ιδιαίτερα ενδιαφέρουσα στην περίπτωση που ο χρήστης είναι ελεύθερος να διαλέξει μεταξύ πολλών αλγορίθμων για την εκτέλεση της ίδιας διεργασίας, πολλών παράλληλων μηχανών με διαφορετικά χαρακτηριστικά αρχιτεκτονικής ή τέλος εναλλακτικών αλγορίθμων ανακατανομής φορτίου σε σχέση πάντα με τα χαρακτηριστικά και την κατανομή του περιεχόμενου των εικόνων.

Ειδικότερα για τις τρεις πρώτες συνιστώσες, η μελέτη της αλληλεπίδρασής τους θα μας επιτρέψει να βρίσκουμε κάθε φορά μια "βέλτιστη" λύση, ξεκινώντας από τα στοιχεία που έχουμε δεδομένα και καθορίζοντας στη συνέχεια τις απροσδιόριστες παραμέτρους. Για παράδειγμα, αν γνωρίζουμε τη διεργασία και το περιεχόμενο της εικόνας, μπορούμε να επιλέξουμε την καταλληλότερη αρχιτεκτονική, με κάποιο βαθμό ελευθερίας αν δεν είναι όλες οι παράμετροι της διεργασίας ή του περιεχόμενου της εικόνας καθορισμένες. Παρομοίως, αν έχουμε δεδομένα την αρχιτεκτονική και το περιεχόμενο της εικόνας μπορούμε να κάνουμε την καλύτερη δυνατή επιλογή για τον αλγόριθμο υλοποίησης της διεργασίας. Τελικά, ο χαρακτηρισμός των συνιστωσών αυτών θα μας επιτρέψει να δημιουργήσουμε ομάδες αλγορίθμων με παρόμοια συμπεριφορά ανάλογα με τα χαρακτηριστικά του περιεχομένου της εικόνας και τελικά να κάνουμε προβλέψιμη, με εφικτές και ταχύτερες μεθόδους, την αποδοτικότητα του συστήματος.

Προκειμένου να δείξουμε τον τρόπο λειτουργίας αλλά και τις δυνατότητες της μεθόδου παρουσιάζουμε διεξοδικά την εφαρμογή και τα αποτελέσματά της σε δύο διεργασίες ανάλυσης εικόνων, ενώ κάνουμε συνοπτική παρουσίαση για ορισμένες άλλες. Αν και αναπτύσσουμε νέους αλγορίθμους ή τροποποιούμε υπάρχοντες για τις διεργασίες που εξετάζουμε, δεν αποτελεί κύριο στόχο μας η ανάπτυξη βέλτιστων θεωρητικά αλγορίθμων. Έτσι, όπως φαίνεται και από τα αναλυτικά και πειραματικά αποτελέσματα στο έκτο κεφάλαιο, η εφαρμογή της μεθόδου, ανάλογα με το περιεχόμενο της εικόνας είναι δυνατό να εισηγηθεί την υλοποίηση της διεργασίας σε αρχιτεκτονικές με σημαντικές διαφορές στα χαρακτηριστικά τους ή τη χρήση διαφορετικών αλγορίθμων ή ακόμα την απαίτηση για χρήση διαφορετικών δομών δεδομένων εισόδου.

Επόμενο βήμα είναι η ανάπτυξη μεθόδων εκτίμησης των επιδόσεων των διαφόρων παράλληλων υλοποιήσεων και εξαγωγής συμπερασμάτων. Τα συμπεράσματα αυτά θα είναι σχετικά με την αλληλεξάρτηση των χαρακτηριστικών των τεσσάρων συνιστωσών και θα χρησιμοποιηθούν σαν ανάδραση (feedback) στην προσπάθειά μας να βελτιώσουμε το σύστημα λήψης αποφάσεων ώστε να προσαρμόσουμε κατά τον καλύτερο τρόπο τις επόμενες υλοποιήσεις μας στους διαθέσιμους κατά περίπτωση πόρους.

Η ανάπτυξη όλων αυτών των τεχνικών και μεθόδων καθώς και η συλλογή πειραματικών δεδομένων και συμπερασμάτων, έχουμε την πεποίθηση πως συνεισφέρει σημαντικά στην κατεύθυνση ανάπτυξης μιας γενικότερης μεθοδολογίας που θα μας επιτρέπει τη γρήγορη λήψη αποφάσεων και επιλογών κατά τρόπο δυναμικό, ώστε να επιτυγχάνουμε κατά περίπτωση την καλύτερη δυνατή επίδοση των διαθέσιμων συστημάτων. Η ανάπτυξη αυτής της μεθοδολογίας πιστεύουμε πως οδηγεί τελικά στην ανάπτυξη ενός έμπειρου συστήματος, που θα μπορεί να αποφασίζει γρήγορα εκτιμώντας, σύμφωνα με τη διαμορφωμένη γνώση που θα διαθέτει, ποιες θα είναι κάθε φορά οι βέλτιστες επιλογές.

2.2 Το περίγραμμα της Διατριβής

Στη συνέχεια η εργασία εξελίσσεται ως εξής: στο τρίτο κεφάλαιο παρουσιάζεται το θέμα της παράλληλης υλοποίησης των διεργασιών ανάλυσης εικόνων. Σκιαγραφούνται οι τρεις βασικές κατηγορίες στις οποίες χωρίζονται οι διεργασίες, ανάλογα με το επίπεδο αφαίρεσης των

δομών δεδομένων και την επεξεργασία που λαμβάνει χώρα. Ακολούθως γίνεται ανασκόπηση της σχετικής βιβλιογραφίας πάνω στις παράλληλες αρχιτεκτονικές και τα συστήματα που έχουν σχεδιαστεί για ανάλυση εικόνων, δίνοντας συνάμα μια γενικότερη ιστορική αναδρομή της εξέλιξης της παράλληλης επεξεργασίας. Η παρουσίαση της βιβλιογραφίας συμπληρώνεται με το θέμα των παράλληλων αλγορίθμων και εφαρμογών, που σχετίζονται με την εργασία αυτή και ολοκληρώνεται με τη βιβλιογραφία που αναφέρεται τόσο σε τεχνικές ανακατανομής του φορτίου των επεξεργαστών (Load Balancing) των παράλληλων αρχιτεκτονικών, όσο και σε μεθόδους κατάτμησης των διεργασιών (Task Partitioning).

Στο τέταρτο κεφάλαιο παρουσιάζονται αναλυτικά οι τέσσερις συνιστώσες του προβλήματος και τα χαρακτηριστικά γνωρίσματά τους. Οι συνιστώσες αυτές είναι η διεργασία ανάλυσης εικόνων, η αρχιτεκτονική, ο αλγόριθμος ισοκατανομής του φορτίου και το περιεχόμενο της εικόνας.

Στο πέμπτο κεφάλαιο δίνουμε μια γενική περιγραφή ενός μοντέλου για την επιλογή της αποδοτικότερης παράλληλης υλοποίησης μιας διεργασίας ανάλυσης εικόνων. Η επιλογή αυτή θα στηρίζεται στην παραμετροποιημένη αναπαράσταση των συνιστωσών που συνθέτουν το πρόβλημα και της αλληλεπίδρασής τους. Γίνεται επίσης σύγκριση της αρχιτεκτονικής και των αλγορίθμων με τη μέθοδο της εναπόθεσης των γράφων (Graph Matching), που εμφανίζεται στη βιβλιογραφία.

Στο έκτο κεφάλαιο διερευνάται πειραματικά η αποτελεσματικότητα από την εφαρμογή μιας τέτοιας μεθόδου. Συγκεκριμένα παρουσιάζεται η υλοποίηση γνωστών προβλημάτων ανάλυσης εικόνων, αρχίζοντας από τον υπολογισμό του κυρτού περιγράμματος (Convex Hull) ενός συνόλου σημείων. Αναπτύσσονται αλγόριθμοι για δύο παράλληλες μηχανές με διαφορετικά χαρακτηριστικά και αναλύονται σύμφωνα με τη μέθοδο που προτείνεται στο προηγούμενο κεφάλαιο. Η αλληλεπίδραση των τεσσάρων συνιστωσών και η επίπτωση των χαρακτηριστικών τους στην επιλογή της αρχιτεκτονικής και του αλγορίθμου, όπως είχε αρχικά προβλεφτεί, επιβεβαιώνεται πειραματικά. Ακολουθεί ανάλογη προσέγγιση για το πρόβλημα της εύρεσης Συνδεδεμένων Συστατικών (Connected Components). Επίσης επιβεβαιώνεται η επίδραση των επιμέρους γνωρισμάτων στις επιλογές για τη βέλτιστη υλοποίηση των αλγορίθμων. Το κεφάλαιο ολοκληρώνεται με συνοπτική αντίστοιχη ανάλυση και άλλων γνωστών διεργασιών ανάλυσης εικόνων ενώ σχολιάζεται η υλοποίηση των αλγορίθμων ισοκατανομής του φορτίου για τους αλγορίθμους

αυτούς.

Στο έβδομο κεφάλαιο παρουσιάζονται τα συμπεράσματα και η συνεισφορά της διατριβής στην επίλυση του προβλήματος της αποδοτικής παράλληλης ανάλυσης εικόνων, ενώ διατυπώνονται οι μελλοντικές προοπτικές που διαφαίνονται από την προσέγγισή μας και την ανάπτυξη μιας γενικότερης μεθοδολογίας, που συμπεριλαμβάνει το σχεδιασμό ενός έμπειρου συστήματος για αυτόματη επιλογή των κατάλληλων υλοποιήσεων.

Τέλος η εργασία κλείνει με ένα παράρτημα στο οποίο παρουσιάζονται συνοπτικά οι αρχιτεκτονικές που χρησιμοποιήθηκαν καθώς και μετρήσεις των υπολογιστικών και επικοινωνιακών επιδόσεών τους (benchmarks) ενώ παρατίθεται η βιβλιογραφία στην οποία γίνονται αναφορές στην εργασία αυτή.

Κεφάλαιο 3

Παράλληλη Υλοποίηση των Διεργασιών Ανάλυσης Εικόνων

3.1 Εισαγωγή

Μια περιοχή εφαρμογών στην οποία η Παράλληλη Επεξεργασία έχει αποκτήσει ιδιαίτερη σημασία είναι η Ανάλυση Εικόνων και γενικότερα η Μηχανική Οραση. Η Μηχανική Οραση υλοποιείται από διεργασίες που εφαρμόζονται σε διάφορες δομές δεδομένων που παριστούν εικόνες και το περιεχόμενό τους. Οι διεργασίες αυτές χωρίζονται σε τρία επίπεδα σε κάθε ένα από τα οποία η Παράλληλη Επεξεργασία εφαρμόζεται κατά τρόπο διαφορετικό τόσο λόγω της φύσης των διεργασιών όσο και λόγω των διαφορετικών δομών που απαιτούνται. Τα επίπεδα αυτά ορίζουν μια ιεραρχία από την άποψη ότι οι διεργασίες ενός επιπέδου χρησιμοποιούν τα αποτελέσματα από πολλές διεργασίες των χαμηλοτέρων επιπέδων. Από την άλλη μεριά, πολύ συχνά, διεργασίες του ίδιου επιπέδου χρειάζεται να συνεργαστούν προκειμένου να ολοκληρώσουν το έργο τους, ενώ δεν είναι σπάνιο το γεγονός υψηλότερου επιπέδου διεργασίες να κατευθύνουν την ενεργοποίηση άλλων, χαμηλότερου επιπέδου διεργασιών, εστιάζοντας έτσι το ενδιαφέρον του συστήματος σε ορισμένες περιοχές της εικόνας και στην εξαγωγή από αυτές συγκεκριμένων χαρακτηριστικών που είναι απαραίτητα προκειμένου να ολοκληρωθεί κάποια λειτουργία. Ετσι, πολύ ενδιαφέροντα ερωτήματα τίθενται σχετικά με τον τρόπο αναπαράστασης της εικόνας στα διάφορα επίπεδα λόγω των διαφορετικών επιπέδων αφαίρεσης (abstraction

levels) αλλά και το πώς θα γίνεται η μετατροπή από τον ένα τρόπο αναπαράστασης στον άλλο ώστε να επιτυγχάνεται η μεγαλύτερη δυνατή απόδοση του συστήματος.

Τα τρία διαφορετικά επίπεδα στα οποία χωρίζονται γενικά οι διεργασίες είναι:

α) Οι **Χαμηλού επιπέδου** διεργασίες (Low level tasks) που είναι κυρίως αριθμητικές πράξεις πάνω σε πίνακες οι οποίοι αντιπροσωπεύουν διακριτές τιμές των εντάσεων των σημείων (pixels) μιας εικόνας. Οι πίνακες μπορεί να είναι δύο ή τριών διαστάσεων. Οι διεργασίες αυτές μπορούν από τη φύση τους να υλοποιηθούν παράλληλα, ενώ τα δεδομένα μπορούν χωρίς ιδιαίτερο πρόβλημα να χωριστούν σε μικρότερα τμήματα και δεν έχουν έντονες απαιτήσεις ενδοεπικοινωνίας μεταξύ των επεξεργαστών στους οποίους καταμερίζεται το πρόβλημα. Οι διεργασίες αυτές ταιριάζουν με πολύ φυσικό τρόπο στην αρχιτεκτονική του ορθογωνικού πλέγματος (Mesh Connected Architecture).

β) Ακολουθούν οι διεργασίες **Μέσου επιπέδου** (Indermediate level). Με το πέρας των διεργασιών χαμηλού επιπέδου έχει γίνει συρρίκνωση στα δεδομένα (Data Reduction), ιδιαίτερα στις περιοχές εκείνες όπου δεν υπάρχει ιδιαίτερα πολλή πληροφορία. Η πληροφορία αντιπροσωπεύει χαρακτηριστικά της εικόνας και παριστάνεται με διάφορες δομές όπως γραμμική λίστα, δένδρο, δισδιάστατο πίνακα μικρότερου μεγέθους από τον αρχικό που μπορεί να μην παριστάνει πλέον τις εντάσεις της εικόνας αλλά μια διαφορετική παραμετροποιημένη αναπαράσταση (π.χ. ο μετασχηματισμός Hough) κ.ά.

Οπως ήδη αναφέρθηκε, για καλύτερη απόδοση του συστήματος θα πρέπει όλοι οι επεξεργαστές να έχουν περίπου το ίδιο φορτίο. Ομως ο διαχωρισμός των δομών αυτών σε τμήματα με ίσες ανάγκες επεξεργασίας (ίσο υπολογιστικό φορτίο) για την παράλληλη επεξεργασία τους δεν είναι πάντα αυτονόητος ή εύκολος. Ετσι παρουσιάζεται το πρόβλημα της άνισης κατανομής του φορτίου στους επεξεργαστές του συστήματος με αρνητικές συνέπειες στην απόδοσή του. Πολύ συχνά επίσης απαιτείται στενή συνεργασία μεταξύ των επεξεργαστών που έχουν αναλάβει διαφορετικά τμήματα της εικόνας προκειμένου να εξαχθεί το συνολικό αποτέλεσμα της διεργασίας. Αυτό δημιουργεί έντονες ανάγκες ενδοεπικοινωνίας με αποτέλεσμα σημαντικές πολλές φορές καθυστερήσεις στο δίκτυο.

Η έξοδος (output) από αυτό το επίπεδο είναι κάποια νέα δομή που παριστάνει κατά κανόνα

τμηματοποιημένη (segmented) την εικόνα ως προς ορισμένα χαρακτηριστικά. Τα χαρακτηριστικά αυτά μπορεί να είναι γραμμικά τμήματα (line segments), η ένταση της εικόνας, η υφή (texture), κίνηση (motion) κ.ά. Τα χαρακτηριστικά αυτά, με τα σχετικά μεγέθη τους, τη χωρική τους (spatial) κατανομή και τις σχέσεις που υπάρχουν μεταξύ τους, εκφράζουν αυτό που αποκαλούμε *"περιεχόμενο"* (content) της εικόνας.

γ) Οι διεργασίες **Υψηλού επιπέδου** (Higher level) επεξεργάζονται σημασιολογικά το περιεχόμενο της εικόνας, όπως δίνεται από το προηγούμενο επίπεδο. Αναλύουν τα χαρακτηριστικά της που δίνονται σε συμβολική αναπαράσταση καθώς και τις σχέσεις τους όπως βρέθηκαν στο προηγούμενο επίπεδο. Αποτέλεσμα αυτού του τελευταίου επιπέδου είναι η ερμηνεία της εικόνας με κάποιο τρόπο. Για το σκοπό αυτό χρησιμοποιείται εκτός των μέχρι τότε εξαχθέντων χαρακτηριστικών, και οποιαδήποτε άλλη πληροφορία είναι διαθέσιμη για το περιεχόμενο της εικόνας και η οποία θα χρησίμευε στο να περιορίσει το πεδίο έρευνας (search space).

Η παράλληλη υλοποίηση των διεργασιών όλων των επιπέδων έχει ιδιαίτερο ενδιαφέρον καθώς εμφανίζονται διάφορα προβλήματα στην προσπάθεια να επιτύχουμε υψηλές αποδόσεις. Πολλές διεργασίες και ιδιαίτερα οι χαμηλού επιπέδου, μπορούν να υλοποιηθούν παράλληλα κατά τρόπο άμεσο, κατά κανόνα στις αρχιτεκτονικές ορθογωνικού πλέγματος καθώς απαιτούν απλές δομές δεδομένων, με ομοιόμορφη κατανομή του υπολογιστικού φορτίου στους επεξεργαστές που εκτελούν σύγχρονα τις εντολές. Οσο όμως εξελίσσεται η επεξεργασία και πηγαίνουμε σε διεργασίες μέσου και υψηλότερου επιπέδου δημιουργείται μια ανομοιογένεια, τόσο στις δομές δεδομένων όσο και στις υπολογιστικές και επικοινωνιακές ανάγκες. Οι επικοινωνίες που απαιτούνται μεταξύ των επεξεργαστών χάνουν ολοένα και περισσότερο τον τοπικό και κανονικό χαρακτήρα τους και μεταβάλλονται σε καθολικές (global) επικοινωνίες με επιλεκτική και συχνά μη προβλέψιμη συμπεριφορά. Απαιτούν προσπέλαση όχι μόνο στα τοπικά δεδομένα, δηλαδή τα δεδομένα του ίδιου του επεξεργαστή και των γειτονικών του, αλλά και στα δεδομένα άλλων επεξεργαστών που βρίσκονται σε μεγαλύτερη φυσική απόσταση πάνω στην τοπολογία. Αυτά τα φαινόμενα είναι συνέπεια της παρατηρούμενης συρρίκνωσης των δεδομένων. Αυτό αρχίζει σιγά σιγά να δίνει προτεραιότητα στις αρχιτεκτονικές με κοινή μνήμη (shared memory) σε αντίθεση με τις αρχιτεκτονικές κατανεμημένης μνήμης (distributed memory) που ικανοποιούν τις ανάγκες των διεργασιών στα δύο πρώτα στάδια. Επιπροσθέτως η ανομοιογένεια των υπολο-

γιστικών αναγκών στα στάδια αυτά, εξυπηρετείται συχνά πιο ικανοποιητικά από αρχιτεκτονικές MIMD (Multiple Instruction stream - Multiple Data stream).

Οι διεργασίες που αναφέρθηκαν παραπάνω, κατά κανόνα μπορούν να διασπαστούν σε άλλες επιμέρους διεργασίες που θα τις ονομάζουμε υποδιεργασίες (subtasks). Αυτό γίνεται κυρίως για λόγους απλοποίησης της εφαρμογής. Μερικές από τις υποδιεργασίες μπορούν να εκτελεστούν παράλληλα, ενώ οι υπόλοιπες σειριακά. Το πως θα γίνει ο διαχωρισμός ενός προβλήματος σε υποδιεργασίες και διεργασίες εξαρτάται από την πολυπλοκότητά του.

Αναφέρθηκε προηγουμένως ότι σε ορισμένες διεργασίες, ιδιαίτερα μέσου και υψηλού επιπέδου, εμφανίζεται το φαινόμενο μερικοί επεξεργαστές να έχουν να εκτελέσουν περισσότερο υπολογιστικό φορτίο από άλλους. Ορισμένοι μάλιστα είναι πιθανό να μην έχουν καθόλου φορτίο, με αποτέλεσμα να παραμένουν αδρανείς. Αυτό έχει σαν συνέπεια την πλημμελή χρησιμοποίηση του συστήματος. Σ'αυτές τις περιπτώσεις εφαρμόζονται διεργασίες ανακατανομής του φορτίου που το μεταφέρουν από τους υπερφορτωμένους σε άλλους λιγότερο φορτωμένους επεξεργαστές επιδιώκοντας έτσι την επίτευξη ομοιόμορφης κατανομής (Load Balancing) και τη βελτιστοποίηση της απόδοσης του συστήματος. Οι αλγόριθμοι ανακατανομής θα αποτελέσουν αντικείμενο ιδιαίτερης αναφοράς σε επόμενες παραγράφους. Θέματα που εγείρονται σχετικά με αυτούς είναι το τι ακριβώς είναι υπολογιστικό φορτίο για κάθε διεργασία, τι αντιπροσωπεύει, πώς παριστάνεται, πώς υπολογίζεται και βέβαια πώς μπορεί να μεταφερθεί με μικρό πρόσθετο κόστος (overhead) από τον ένα επεξεργαστή στον άλλο. Το φορτίο πάντως των επεξεργαστών έχει σχέση με τη συγκεκριμένη διεργασία και με το περιεχόμενο της εικόνας, δηλαδή με την ποσότητα, το είδος και την κατανομή της πληροφορίας που θέλουμε να επεξεργαστούμε.

Το γενικό συμπέρασμα που μπορούμε να βγάλουμε από όσα αναφέρθηκαν μέχρι τώρα είναι ότι για να επιτύχουμε την καλύτερη δυνατή απόδοση θα πρέπει χρησιμοποιήσουμε την κατάλληλη σύνθεση των επιμέρους συνιστωσών του προβλήματος. Ας υποθέσουμε για παράδειγμα ότι η αρχιτεκτονική, που θα χρησιμοποιήσουμε για κάποια συγκεκριμένη διεργασία είναι υψηλής διαμέρισης (fine granularity), δηλαδή αποτελείται από μεγάλο αριθμό επεξεργαστών. Εάν και η διεργασία είναι δυνατό από τη φύση της να σπάσει σε πολλά μέρη χωρίς πρόσθετο υπολογιστικό κόστος ή πρόσθετο κόστος επικοινωνίας (computational or communication overhead) τότε τα μεγέθη της επιτάχυνσης και της απόδοσης του συστήματος μπορεί να πάρουν υψηλές τιμές. Αντίθετα, αν η υλοποίησή της σε αρχιτεκτονική υψηλής διαμέρισης απαιτεί μεγάλο πρό-

οθετο κόστος ενδοεπικοινωνίας και υπολογισμών τότε πέφτει η απόδοση του συστήματος. Στην περίπτωση αυτή το πιθανότερο είναι ότι θα ήταν αποδοτικότερη η υλοποίησή του σε συστήματα με λίγους επεξεργαστές ενώ σε ακραίες περιπτώσεις ίσως να είναι προτιμότερη η σειριακή υλοποίηση.

Προκειμένου να επιτύχουμε τα επιθυμητά αποτελέσματα με την εφαρμογή της παράλληλης επεξεργασίας στην Ανάλυση Εικόνων θα πρέπει να κάνουμε την κατάλληλη επιλογή των τεσσάρων βασικών παραγόντων. Το πόσο καλή είναι η συνεργασία τους θα εκτιμηθεί με ποσοτικά μεγέθη όπως η επιτάχυνση του συστήματος, η απόδοσή του ή το απαιτούμενο κόστος του συστήματος. Ποιοτικά μεγέθη όπως η δυνατότητα επαναχρησιμοποίησης του αλγόριθμου (Reusability), η ευκολία προγραμματισμού (Programmability) κ.ά. μπορούν επίσης να χρησιμοποιηθούν σαν κριτήρια χαρακτηρισμού μιάς παράλληλης υλοποίησης σαν ικανοποιητικής ή μη.

Οπως ήδη αναφέρθηκε, οι επιμέρους παράγοντες έχουν ερευνηθεί εκτενώς και μια πληθώρα από αλγορίθμους και από υλοποιήσεις παρέχουν πλούσια εμπειρία και πολλά συμπεράσματα για τον τρόπο λειτουργίας της παράλληλης επεξεργασίας για την Ανάλυση Εικόνων. 'Ομως αυτό που παρατηρούμε είναι πως, παρά την πληθώρα αυτών των υλοποιήσεων και των εφαρμογών, διακρίνεται μια αδυναμία στη γενίκευση των επιμέρους συμπερασμάτων και στην ολοκλήρωση (integration) των επί μέρους μεθόδων. Με άλλα λόγια κάθε τμήμα ενός μεγάλου προβλήματος αντιμετωπίζεται αποσπασματικά σαν ένα ειδικό επιμέρους πρόβλημα. Θα ήταν όμως επιθυμητό να έχει κανείς στη διάθεσή του ένα σύστημα Ανάλυσης Εικόνων το οποίο θα καλύπτει όλες τις διεργασίες που συναντά κανείς στο θέμα αυτό. Ενα τέτοιο σύστημα θα έπρεπε να διαθέτει την αναγκαία ευελιξία και προσαρμοστικότητα ώστε να αντιμετωπίζει με ικανοποιητική αποτελεσματικότητα μια αλυσίδα διεργασιών με διαφοροποιημένες απαιτήσεις από τις οποίες συνίσταται κατά κανόνα ένα μεγάλο πρόβλημα. Να αναφέρουμε για παράδειγμα τη γενική διεργασία της περιγραφής του περιεχομένου μιας εικόνας. Είναι μια πολυσύνθετη διεργασία υψηλού επιπέδου που προϋποθέτει την ολοκλήρωση μιας αλυσίδας άλλων διεργασιών μέσου και χαμηλού επιπέδου. Μια ενδεικτική τέτοια αλυσίδα διεργασιών για αναγνώριση αντικειμένων βάσει μοντέλου είναι η εξής: η διεργασία αφαίρεσης του θορύβου (noise reduction, low pass filtering), η αναγνώριση ακμών (edge detection), η εκλέπτυνση των ακμών (thinning), η σύνδεση των τμημάτων των ακμών που για κάποιο λόγο έχουν "σπάσει" (edge linking), στη συνέχεια

μια τμηματοποίησή τους (segmentation), περιγραφή των χαρακτηριστικών των περιγραμμάτων των αντικειμένων που σχηματίζονται και, τέλος, σύγκρισή τους με κάποια μοντέλα αντικειμένων (model matching). Κάθε μία από τις επιμέρους αυτές διεργασίες έχει διαφορετικές απαιτήσεις αρχιτεκτονικής για τη βέλτιστη υλοποίησή της τόσο σε υπολογιστική ισχύ όσο και σε ενδοεπικοινωνία, χρησιμοποιεί διαφορετικές δομές δεδομένων ενώ υπάρχει πιθανότητα να χρειάζεται ή να μην χρειάζεται την εφαρμογή αλγορίθμου για ισοκατανομή του φορτίου. Επίσης για πολλές από αυτές υπάρχουν διάφοροι αλγόριθμοι που τις υλοποιούν ενώ μπορεί να διαφέρει σημαντικά και ο τρόπος που ο κάθε ένας από αυτούς υλοποιείται σε κάποια συγκεκριμένη αρχιτεκτονική.

Στόχος της εργασίας αυτής θα είναι να συμβάλλουμε στη δημιουργία ενός ολοκληρωμένου περιβάλλοντος Ανάλυσης Εικόνων δημιουργώντας μία μέθοδο-πλαίσιο που θα δίνει τη δυνατότητα για πρόβλεψη της απόδοσης ενός συστήματος με βάση τα χαρακτηριστικά των βασικών παραγόντων. Εχοντας ακόμη δεδομένα τα χαρακτηριστικά ορισμένων από αυτούς να μπορούμε να καθορίσουμε τις απαιτήσεις μας για τους υπόλοιπους ώστε να έχουμε βελτιστοποίηση στην απόδοση του συστήματος. Αν θεωρήσουμε ότι η απόδοση του συστήματος (ή η επιτάχυνση) είναι μια συνάρτηση $E = f(p_1, p_2, p_3 ... p_n)$ n παραμέτρων, θα πρέπει αρχικά να ορίσουμε τις παραμέτρους αυτές και στη συνέχεια να βρούμε πώς αντιστοιχούν τα χαρακτηριστικά των τεσσάρων παραγόντων του προβλήματος σ'αυτές. Για να γίνει πιο συγκεκριμένο αυτό ας δώσουμε ένα παράδειγμα. Μια γενική έκφραση της απόδοσης είναι : $Απόδοση = \frac{T_{sequential}}{N*(M*T_m+Y*T_y+T_{lb}+T_{ov})}$. Ο αριθμητής είναι ο χρόνος βέλτιστης σειριακής επεξεργασίας και μπορεί να θεωρηθεί σταθερός. Οι παράμετροι του παρανομαστή είναι, N: ο αριθμός των επεξεργαστών, M: το μέγιστο πλήθος των μηνυμάτων που ανταλλάσσει ένας επεξεργαστής, T_m: ο χρόνος που χρειάζεται να μεταφερθεί το ένα μήνυμα, Y: ο αριθμός υπολογιστικών πράξεων από ένα επεξεργαστή, T_y: ο χρόνος εκτέλεσης μίας τέτοιας πράξης, T_{lb}: ο χρόνος που απαιτεί η ανακατανομή του φορτίου (εάν γίνει) και T_{ov}: ο χρόνος που σπαταλάται από τους επεξεργαστές για διάφορους άλλους λόγους (overhead). Η αντιστοίχηση των χαρακτηριστικών των βασικών παραγόντων του προβλήματος στις παραμέτρους αυτές μπορεί να γίνει ως εξής: Το N αντιστοιχεί άμεσα στο πλήθος των επεξεργαστών της αρχιτεκτονικής. Αν το σύστημα έχει λεπτή διαμέριση τότε αυξάνει το N. Επίσης από τους επεξεργαστές του συστήματος εξαρτάται το T_y ενώ από την ταχύτητα και τη συνδεσμολογία του συστήματος το T_m. Από τη διεργασία εξαρτάται αν πρέπει να ανταλλαγούν λιγότερα ή περισσότερα μηνύματα M μεταξύ των επεξεργαστών καθώς επίσης το πλήθος των υπολογισμών Y για κάθε μονάδα δεδομένων. Μονάδα δεδομένων μπορεί να

είναι ένα σημείο της εικόνας (pixel), ένα τμήμα της (segment), ένα χαρακτηριστικό (feature) κ.ά. Τα δύο τελευταία αυτά μεγέθη εξαρτώνται όχι μόνο από τη διεργασία αλλά συχνά και από το ποιο είναι το περιεχόμενο της εικόνας. Αν ορίσουμε με όσο το δυνατόν μεγαλύτερη ακρίβεια τη σχέση αυτή που υπάρχει μεταξύ των χαρακτηριστικών των τεσσάρων παραγόντων ή κλάσεων των παραγόντων με όμοια χαρακτηριστικά και των παραμέτρων της απόδοσης θα μπορούμε να κάνουμε πρόβλεψη της απόδοσης του συστήματος.

3.2 Ανασκόπηση της σχετικής βιβλιογραφίας

Οπως ήδη αναφέραμε, δύο είναι τα βασικά ποσοτικά μεγέθη με τα οποία μετράμε την αποδοτικότητα ενός παράλληλου συστήματος: η Επιτάχυνση, και η Απόδοσή του. Η ιδανική αποτελεσματικότητα ενός Πολυεπεξεργαστή χαρακτηρίζεται από Επιτάχυνση ίση με τον αριθμό των επεξεργαστών που τον συνθέτουν και Απόδοση ίση με τη μονάδα. Οι παραπάνω όμως βέλτιστες επιδόσεις δεν είναι δυνατόν να επιτευχθούν για διάφορους λόγους.

Διάφορες εργασίες έχουν δημοσιευθεί κατά καιρούς υποστηρίζοντας τη δυνατότητα υπεργραμμικής απόδοσης, (απόδοση > 1) [12] [13] , για ειδικές περιπτώσεις αλγορίθμων και με ειδικές παραδοχές για την αρχιτεκτονική των επεξεργαστών. Οι εργασίες αυτές έχουν βρει αντίλογο [14] και [15]. Οι Flatt και Kennedy [16] αποδεικνύουν πως, υπό ορισμένες συνθήκες, οι οποίες θεωρούνται ρεαλιστικές, υπάρχουν άνω όρια στην ισχύ των παράλληλων μηχανών. Ορίζουν δε μία συνάρτηση που συνδέει την απόδοση, την επιτάχυνση και τον αριθμό των επεξεργαστών του συστήματος για να βρουν το πλήθος των επεξεργαστών που βελτιστοποιεί το λόγο επιτάχυνση προς κόστος.

Οι μέχρι τώρα δημοσιευμένες ερευνητικές εργασίες στο χώρο της παράλληλης επεξεργασίας για Ανάλυση Εικόνων αναφέρονται κυρίως στην αντιμετώπιση επιμέρους προβλημάτων. Ιδιαίτερη έμφαση έχει δοθεί στη μελέτη αρχιτεκτονικών υπολογιστών, κατάλληλων για την υλοποίηση παράλληλων αλγορίθμων για Ανάλυση Εικόνων: αρχιτεκτονικές που είτε είναι οι καθιερωμένες γενικού σκοπού (general purpose), που χρησιμοποιούνται και σε πολλές άλλες περιοχές εφαρμογών, είτε αρχιτεκτονικές για εξειδικευμένες εφαρμογές (special purpose architectures) που γίνονται όλο και πιο προσιτές χάρη στις εξελίξεις στον τομέα των ολοκληρω-

μένων κυκλωμάτων. Η μεγάλη ποικιλία των συστημάτων που έχουν δημιουργηθεί κατά καιρούς δείχνει ακριβώς τη διαφοροποίηση στις απαιτήσεις των διεργασιών που εμπλέκονται στην περιοχή της Ανάλυσης Εικόνων. Οι διαφοροποιήσεις αυτές σχετίζονται τόσο με τις υπολογιστικές απαιτήσεις όσο και με τις δομές δεδομένων και τις ανάγκες ενδοεπικοινωνίας. Επίσης γίνεται φανερό πως οι διάφορες διατάξεις και αρχιτεκτονικές που δημιουργούνται επιτυγχάνουν ίσως βελτιστοποίηση σε ένα μέρος της επιθυμητής επεξεργασίας αλλά φαίνεται να είναι δύσκολο να επιτύχουν συνολική βελτιστοποίηση.

Είναι χαρακτηριστικό ότι η επόμενη γενιά Υπερυπολογιστών που αναμένεται να είναι διαθέσιμη περί το 1995 [17] θα έχει υπολογιστική ισχύ τουλάχιστον 1 τρισεκατομμύριο πράξεις το δευτερόλεπτο, κύρια μνήμη εκατό Gbytes και μερικές μυριάδες στοιχεία επεξεργασίας-μνήμης (processing-memory elements). Ο προγραμματισμός τους θα χαρακτηρίζεται από αφαίρεση υψηλού βαθμού ενώ τα βασικά θέματα που θα απασχολούν τους ερευνητές θα σχετίζονται με την επικοινωνία και την ισοκατανομή του υπολογιστικού φορτίου των επεξεργαστών, εξαιτίας της κατανομής των δεδομένων σε όλο αυτό το σύστημα.

Εκτενής επίσης βιβλιογραφία υπάρχει για την παράλληλη υλοποίηση πολλών αλγορίθμων για Ανάλυση Εικόνων, καθώς επίσης και αλγορίθμων για την ανακατανομή του υπολογιστικού φορτίου των επεξεργαστών. Εκεί που φαίνεται να υπάρχει περιορισμένη ερευνητική εργασία είναι στην περιγραφή και τον ορισμό του περιεχομένου της εικόνας, τη σχέση του με το υπολογιστικό φορτίο των επεξεργαστών και στο πώς επηρεάζει τον τρόπο υλοποίησης των αλγορίθμων σε κάθε αρχιτεκτονική.

Οι Flatt και Kennedy [16] μελέτησαν το πώς επηρεάζεται η απόδοση ενός συστήματος γενικά, από το βαθμό διαμέρισης του προβλήματος και εισάγουν την έννοια της κλιμακωτής επιτάχυνσης (scaled speedup).

Ο Stout [18] ασχολείται με το θέμα της εναπόθεσης (mapping) αλγορίθμων Ανάλυσης Εικόνων σε παράλληλες αρχιτεκτονικές και προτείνει ορισμένες μεθόδους όπως την προσομείωση μίας αρχιτεκτονικής, για την οποία έχουν σχεδιαστεί ορισμένοι αλγόριθμοι, πάνω σε άλλη αρχιτεκτονική ώστε οι ίδιοι αλγόριθμοι να μπορούν να τρέχουν και στη δεύτερη αρχιτεκτονική. Άλλη προσέγγιση είναι το να σχεδιάσει κανείς ιδανικούς αλγορίθμους για την ιδανική εκάστοτε αρχιτεκτονική, καθώς επίσης το να χρησιμοποιεί γενικές πράξεις μεταφοράς δεδομένων κατά

το σχεδιασμό των αλγορίθμων, φροντίζοντας κάθε φορά να χρησιμοποιεί αρχιτεκτονικές όπου οι πράξεις αυτές υλοποιούνται βέλτιστα. Παρουσιάζει εφαρμογές όπου γίνεται φανερό πως κάθε τέτοια προσέγγιση δεν αποδίδει σε όλες τις περιπτώσεις, ενώ καταλήγει στο συμπέρασμα πως έως ότου γίνει πραγματικότητα η οικονομικά προσιτή και με τις κατάλληλες προδιαγραφές Μνήμη Παράλληλης Τυχαίας Προσπέλασης (PRAM) θα πρέπει να σχεδιάζεται με προσοχή χωριστά ο κάθε αλγόριθμος για κάθε αρχιτεκτονική. Οι Γερογιάννης και Ορφανουδάκης [19] δείχνουν πως η απόδοση ενός αλγορίθμου εξαρτάται από τον τρόπο προσαρμογής (embedding) του στη συγκεκριμένη παράλληλη μηχανή και μελετούν την υλοποίηση συγκεκριμένης διεργασίας μέσου επιπέδου με διαφορετικούς τρόπους.

Στην [20] μελετώνται δύο αλγόριθμοι για τον υπολογισμό του Ιστογράμματος μιας εικόνας, εκ των οποίων ο ένας είναι εξαρτώμενος και ο άλλος ανεξάρτητος από τα δεδομένα, με ενδιαφέροντα συμπεράσματα αναφορικά με την εξάρτηση της απόδοσης των συγκεκριμένων αλγορίθμων από την ποσότητα των σημείων της εικόνας και τη διαμέριση του ζητούμενου Ιστογράμματος. Εχουν γίνει και παλιότερα εργασίες στις οποίες γίνεται προσπάθεια να διερευνηθεί η εξάρτηση συγκεκριμένων αλγορίθμων από τα δεδομένα. Για παράδειγμα, στην [21] μελετάται η εξάρτηση της απόδοσης ενός πολυεπεξεργαστή στην υλοποίηση της διεργασίας εύρεσης ακμών από το πλήθος των σημείων που ανήκουν σε ακμές. Επίσης, από τους ίδιους συγγραφείς, στην [22] μελετάται η εξάρτηση συγκεκριμένου αλγόριθμου ανακατανομής του φορτίου των επεξεργαστών από τις υπολογιστικές και επικοινωνιακές απαιτήσεις δύο συγκεκριμένων διεργασιών καθώς επίσης και από τα χαρακτηριστικά των αρχιτεκτονικών που χρησιμοποιούνται (Connection Machine και iPSC/2). Στην [23] αναπτύσσονται δύο στρατηγικές για την ανάπτυξη παράλληλων διεργασιών για Ανάλυση Εικόνων, κατάλληλες για διεργασίες που χρησιμοποιούν στατικές δομές δεδομένων. Η επιλογή της μίας ή της άλλης στρατηγικής γίνεται με κλειστούς τύπους (closed forms) ανάλογα με συγκεκριμένα χαρακτηριστικά του περιεχόμενου της εικόνας.

Στις επόμενες παραγράφους παρουσιάζουμε μια ανασκόπηση της βιβλιογραφίας σχετικά με τις αρχιτεκτονικές για παράλληλη Ανάλυση Εικόνων, τους αλγόριθμους και τεχνικές ανακατανομής του φορτίου των επεξεργαστών καθώς και αλγορίθμων για τη συγκεκριμένη περιοχή εφαρμογών. Επειδή η σχετική με τα τρία αυτά θέματα βιβλιογραφία είναι εν γένει εκτενέστατη, κάνουμε προσπάθεια να εστιάσουμε την προσοχή μας σε περιοχές που σχετίζονται με τη διατριβή αυτή, ενώ παράλληλα γίνεται ένα είδος ιστορικής αναδρομής. Σε καμμία περίπτωση

όμως, δεν εννοούμε πως η ανασκόπηση αυτή καλύπτει όλη την περιοχή. Αλλωστε δεν θα ήταν δυνατόν.

3.2.1 Αρχιτεκτονική

Οταν στο εξής θα μιλάμε για Αρχιτεκτονική, εκτός αν αναφέρεται κάτι διαφορετικό, θα εννοούμε την τοπολογία (topology) του συστήματος δηλαδή την παράσταση του γράφου διασύνδεσης των επί μέρους επεξεργαστών του συστήματος. Στην περίπτωση των υπολοίπων χαρακτηριστικών όπως μέγεθος μνήμης, τύπος επεξεργαστών, υπολογιστική τους δύναμη κτλ. θα γίνεται ειδική αναφορά.

Κυτταρικά Πλέγματα

Η πρώτη δημοσιευμένη πρόταση για παράληλη αρχιτεκτονική ειδικά για επεξεργασία εικόνων εμφανίστηκε το 1958 [24] που πρότεινε ένα διδιάστατο πίνακα (two dimensional array) σαν μία φυσική αρχιτεκτονική για τη συγκεκριμένη εφαρμογή. Ηταν η πρώτη πρόταση που αφορούσε τα πλέγματα κυτταρικών επεξεργαστών (Cellular Arrays). Η αρχιτεκτονική αυτή έμελε να απασχολήσει πάρα πολύ τους ερευνητές στο χώρο της Παράλληλης Επεξεργασίας Εικόνων και αναφερόταν από τους Danielsson και Levialdi σαν "Παράλληλη Μηχανή Εικόνων" (Image Parallel Machine) [25] μιας και η βασική δομή αναπαράστασης των εικόνων, τουλάχιστον για τις διεργασίες χαμηλού επιπέδου, είναι ο διδιάστατος πίνακας, που προσαρμόζεται κατά πολύ φυσικό τρόπο πάνω σ'αυτή την αρχιτεκτονική. Ενας μεγάλος αριθμός από βασικές διεργασίες που χρησιμοποιούνται για την επεξεργασία εικόνων υλοποιούνται άμεσα σ'αυτή την αρχιτεκτονική. Η τοπολογία αυτή αναφέρεται και σαν Πλέγμα (Grid) και αποτελείται από NxN επεξεργαστές διαταγμένους σε ένα ορθογωνικό πλέγμα που ο κάθε ένας επικοινωνεί κατά κανόνα με τους τέσσερις γειτονικούς του (Mesh Connected). Από τα πρώτα τέτοια συστήματα που έγιναν γνωστά είναι τα Illiac IV και Illiac III με 8x8 και 36x36 επεξεργαστές αντίστοιχα στο Πανεπιστήμιο του Illinois [26]. Αλλα τέτοια συστήματα ήταν τα CLIP [27] και DAP [28].

Πάρα πολλοί αλγόριθμοι έχουν αναπτυχθεί για διεργασίες επεξεργασίας εικόνων πάνω σ'αυτή την αρχιτεκτονική. Μερικοί από αυτούς στηρίζονται στο θεωρητικό μοντέλο του "Περιο-

ρισμένου Κυτταρικού Αυτόματου" (Bounded Cellular Automaton - BCA) που θέλει το κάθε κύτταρο να έχει περιορισμένο μέγεθος μνήμης [29] [30] [31]. Αλλοι δέχονται πως το μέγεθος της μνήμης του κάθε κυττάρου μπορεί να αυξάνει με το μέγεθος του συστήματος δηλ. με το πλήθος των κυττάρων, υπόθεση που εκτός του ότι είναι ρεαλιστική για τις σημερινές ικανότητες της τεχνολογίας, απλοποιεί και τους απαιτούμενους αλγορίθμους. Ετσι, έχουν αναπτυχθεί πολλοί βασικοί αλγόριθμοι με ικανοποιητική υπολογιστική πολυπλοκότητα, όπως ο υπολογισμός τοπικών ιδιοτήτων, ο υπολογισμός ιστογράμματος, ο υπολογισμός ροπών και μετασχηματισμών (Fourier, Hadamard κτλ.) σε χρόνο O(N) για εικόνα μεγέθους NxN [32]. Αλλοι αλγόριθμοι είναι για τον υπολογισμό Συνδεμένων Συστατικών (Connected Components), Αναπαράστασης Περιοχών (Region representation)[33] κ.ά.

Πλέγματα με πολύ μεγαλύτερο πλήθος επεξεργαστών έχουν κατασκευαστεί κατά καιρούς για συγκεκριμένες εφαρμογές. Τέτοιο παράδειγμα είναι ο "Μαζικά Παράλληλος Επεξεργαστής" (Massivelly Parallel Processor - MPP) που κατασκευάστηκε για να επεξεργάζεται πολύ γρήγορα δορυφορικές εικόνες και αποτελείται από 16384 επεξεργαστές [34].

Για συγκεκριμένους σκοπούς έχουν προταθεί διάφορες παραλλαγές των κυττάρων ορθογωνικού πλέγματος. Ετσι, για μικρότερο κόστος προτάθηκαν και υπάρχουν αλγόριθμοι για υπολογιστικά κύτταρα σε μονοδιάστατο πίνακα (One-dimensional cellular arrays) [35]. Για να αυξηθεί η ταχύτητα στην περίπτωση που οι αλγόριθμοι δεν έχουν τοπικότητα αλλά χρειάζονται μεταφορά πληροφοριών από ένα μέρος της εικόνας σε άλλο, όχι γειτονικό, έχουν προταθεί λύσεις που επεκτείνουν τα διδιάστατα πλέγματα εμπλουτίζοντας τις συνδέσεις μεταξύ των επεξεργαστών. Δημιουργούνται έτσι κύτταρα με σύνδεση Υπερκύβου, Πυραμίδας, [32] [36] [37] γράφου [38] [39] κ.ά. Ακόμη έχουν προταθεί "Επανασυνδεόμενοι Γράφοι Υπολογιστικών Κυττάρων" (Reconfigurable cellular graphs) που μπορούν να ανασχηματίζουν τις μεταξύ τους συνδέσεις ανάλογα με την εξέλιξη του αλγορίθμου που υλοποιείται [40] [41]. Τέτοιο σύστημα είναι το ZMOB που μπορεί να προσομοιώσει ένα γράφο με 256 κόμβους και διάφορες συνδέσεις μεταξύ τους [42] υλοποιώντας με αυτό τον τρόπο τις βασικότερες δομές, όπως δένδρο,πλέγμα, πυραμίδα κ.ά. Παρόμοιο σύστημα με δυνατότητα επανασύνδεσης των επεξεργαστών είναι το FLIP [43] που διαθέτει 16 επεξεργαστές. Και στα δύο συστήματα η ενδοεπικοινωνία επιτυγχάνεται με ένα γρήγορο διάδρομο ροής δεδομένων. Επίσης έχουν δημοσιευτεί εργασίες για αρχιτεκτονικές που στηρίζουν την ενδοεπικοινωνία τους αποκλειστικά σε ένα διάδρομο ροής

δεδομένων (Bus) και χειρίζονται μικρό αριθμό επεξεργαστών (coarse granularity) ενώ μπορούν να ικανοποιούν μικρές απαιτήσεις σε ενδοεπικοινωνία. Παράδειγμα υλοποίησης τέτοιας αρχιτεκτονικής είναι το σύστημα PICAP II στο οποίο 6 επεξεργαστές συνδέονται μεταξύ τους με ένα γρήγορο τέτοιο διάδρομο. Αλλο τέτοιο είναι το HBA [44] το οποίο διαθέτει 24 επεξεργαστές που μέσω ενός διαδρόμου δημιουργούν ένα ιεραρχικό σύστημα. Είναι προσανατολισμένο σε εφαρμογές Ανάλυσης Εικόνων και χρησιμοποιεί σαν Host υπολογιστή ένα Sun 3.

Ενα πολύ γνωστό εμπορικό σύστημα της αρχιτεκτονικής πλέγματος που αποτελείται από δομικά στοιχεία των 4 επεξεργαστών το κάθε ένα είναι τα Transputers [45]. Συνθέτοντας τέτοια δομικά στοιχεία μπορεί να δημιουργήσει κανείς ένα αρκετά σύνθετο παράλληλο σύστημα με μικρό κόστος. Τα Transputers έχουν δική τους γλώσσα προγραμματισμού, την Occam [46], αλλά πρόσφατα έχουν εμφανιστεί μεταφραστές γνωστών γλωσσών υψηλού επιπέδου (C, Fortran κ.ά) στη γλώσσα Occam. Τα Transputers, κυρίως λόγω της προσιτής τιμής τους έχουν χρησιμοποιηθεί για αρκετές εφαρμογές [47] [48]

Συστολικά Συστήματα

Τα Συστολικά Συστήματα (Systolic Arrays) είναι μία άλλη μορφή παράλληλου συστήματος που προτάθηκε στα τέλη της δεκαετίας του '70. Η ιδέα που υλοποιείται εδώ είναι να διατηρηθεί μια συνεχής ροή δεδομένων διαμέσου ενός συστήματος επεξεργαστών που εκτελούν διαδοχικά συγκεκριμένες διεργασίες (Pipeline computation) δημιουργώντας έτσι ένα φτηνό και αποδοτικό παράλληλο σύστημα [49] [50]. Ο όρος Πίνακας (Array) είχε υιοθετηθεί από τη συσχέτιση του Συστολικού Πίνακα με ένα ορθογωνικό πλέγμα (Grid) στο οποίο κάθε κόμβος αντιστοιχεί σε έναν επεξεργαστή και κάθε γραμμή αντιστοιχεί σε ένα σύνδεσμο μεταξύ των διαδοχικών επεξεργαστών. Μέσω αυτών των συνδέσμων "ρέουν" τα δεδομένα από το ένα άκρο του πλέγματος στο αντιδιαμετρικό του. Η εξέλιξη όμως των Συστολικών Συστημάτων δεν τα περιόρισε στη διάταξη του ορθογωνικού πλέγματος. Κατά καιρούς έχουν παρουσιαστεί και άλλα με τριγωνική ή εξαγωνική διάταξη για καλύτερη απόδοση ανάλογα με τις εφαρμογές για τις οποίες σχεδιάζονται.

Οι χαμηλού επιπέδου διεργασίες της Μηχανικής 'Ορασης υλοποιούνται ικανοποιητικά πάνω σε αρχιτεκτονικές όπως αυτές που έχουμε ήδη αναφέρει. 'Ομως ένα ολοκληρωμένο σύστημα είναι απαραίτητο να μπορεί να δέχεται εφαρμογές με διαδικασίες μέσου και υψηλού επιπέδου οι οποίες έχουν διαφορετικές απαιτήσεις. Για το σκοπό αυτό έχουν μελετηθεί [51] [52] και έχουν κατασκευαστεί τέτοια συστήματα όπως το MACSYM [53], που υλοποιήθηκε για την κατανόηση γραπτών κειμένων, εφημερίδων κ.ά. Αυτά τα συστήματα διαθέτουν μια ιεραρχική διάταξη στη συνδεσμολογία των επεξεργαστών προκειμένου να μπορούν να υλοποιήσουν διεργασίες σε διάφορα επίπεδα. Οι Ιεραρχικές Διατάξεις ονομάζονται συχνά και Πυραμίδες. Διαθέτουν διάφορα επίπεδα ιεραρχίας και σε κάθε ένα εκτελούνται διεργασίες με διαφορετικά χαρακτηριστικά. Πολλοί αλγόριθμοι έχουν υλοποιηθεί ειδικά για τέτοιες αρχιτεκτονικές [54] [55] Ενα τέτοιο σύστημα είναι το PIPE [56] το οποίο διαθέτει 3 διαδρομές pipelining με διαφορετικό τρόπο λειτουργίας και η κάθε μία μπορεί να επεξεργαστεί εικόνες δημιουργώντας μια ιεραρχική διάταξη 8 επιπέδων στους επεξεργαστές του. Μπορεί επίσης εναλλακτικά να χειρίζεται τις εικόνες κατά μη ιεραρχικό τρόπο. Για την αρχιτεκτονική αυτή έχει δημιουργηθεί μία ιεραρχική λογική (hierarchical cellular logic) η οποία τυποποιεί τις δομές (data-objects) που χρησιμοποιεί η PIPE και επιτυγχάνει καλύτερη απόδοση στη λειτουργία της. Επίσης έχουν γίνει συγκριτικές εργασίες για ομοειδείς τέτοιες τοπολογίες ειδικά για την Μηχανική Οραση [57]. Η FLASH είναι μια αρχιτεκτονική αναπτυγμένη για Μηχανική Οραση σε περιβάλλον που χαρακτηρίζεται από αβεβαιότητα [58].

Μια άλλη μορφή Ιεραρχικής Διάταξης είναι εκείνη που αναφέρεται σαν Διάταξη Πολλαπλής Διαμέρισης (Multi resolution) και που χρησιμοποιείται ευρέως στις εφαρμογές της Μηχανικής Οραση ς.

Συγκρίσεις που έχουν δημοσιευθεί για χρήση των παραπάνω αρχιτεκτονικών για εφαρμογές στην Ανάλυση Εικόνων, δείχνουν πως δεν είναι εύκολο να καταλήξει κανείς σε ξεκάθαρη ιεράρχηση της αξίας τους χωρίς να αναφέρεται σε συγκεκριμένες εφαρμογές [59].

Αλλες αρχιτεκτονικές που διαθέτουν πολύ πλουσιότερο δίκτυο ενδοεπικοινωνίας μεταξύ των επεξεργαστών και έχουν πολύ μικρότερη διάμετρο δικτύου είναι η αρχιτεκτονική του Υπερκύβου (Hypercube) και η αρχιτεκτονική της Πεταλούδας (Butterfly). Ενας υπερκύβος απ-

οτελείται από N κόμβους, όπου N είναι δύναμη του 2. Κάθε κόμβος συνδέεται με κάποιον άλλον, εάν και μόνο, οι δυαδικές αναπαραστάσεις των διευθύνσεών τους διαφέρουν κατά ένα μόνο bit. Έτσι ο κάθε ένας διαθέτει log(N) γείτονες. Υπάρχουν σε λειτουργία πολλά τέτοια συστήματα όπως το CosmicCube [60], Mark II,III, iPSC, το NCube κ.ά. ενώ θεωρείται η πλέον διαδεδομένη τοπολογία γενικού σκοπού [61]. Πολλοί αλγόριθμοι Ανάλυσης Εικόνων έχουν δημιουργηθεί ειδικά γι'αυτή την αρχιτεκτονική [62] [63] κ.ά. ενώ έχουν μελετηθεί και μέθοδοι απεικόνισης άλλων αρχιτεκτονικών, όπως δένδρο, διδιάστατο και τριοδιάστατο πλέγμα, δαχτυλίδι (ring) κ.ά πάνω σε υπερκύβο [61].

Η τοπολογία της Πεταλούδας δανείζεται το όνομά της από τη μορφή που παίρνουν οι συνδέσεις μεταξύ των επεξεργαστών. Ένα τέτοιο σύστημα, το Butterfly, έχει κατασκευαστεί και λειτουργεί [64] με 128 όμοιους επεξεργαστές σε αρχιτεκτονική MIMD.

Διαφορές μεταξύ SIMD και MIMD

Μια βασική διαχωριστική τομή που μπορεί να γίνει μεταξύ των διαφόρων παράλληλων αρχιτεκτονικών είναι ως προς τον χαρακτηρισμό τους σαν SIMD ή MIMD. Στην πρώτη κατηγορία ανήκουν εκείνες στις οποίες κάθε επεξεργαστής εκτελεί ακριβώς την ίδια εντολή σε διαφορετικά όμως δεδομένα (Single Instruction stream Multi Data stream), με μόνη εναλλακτική επιλογή να παραμένει αδρανής. Τις εντολές τις δίνει ένας κεντρικός επεξεργαστής που ονομάζεται Host. Οι επεξεργαστές εργάζονται συγχρονισμένα και ο κάθε ένας έχει αναλάβει να επεξεργαστεί ένα τμήμα των δεδομένων. Στη δεύτερη κατηγορία, κάθε επεξεργαστής του παράλληλου συστήματος έχει στη διάθεσή του για να εκτελέσει ένα διαφορετικό (γενικά) τμήμα εντολών πάνω σε ένα επίσης διαφορετικό τμήμα δεδομένων (Multi Instruction stream, Multi Data stream). Οι επεξεργαστές εργάζονται ασύγχρονα και ένα σοβαρό πρόβλημα είναι το ποιος με ποιον, πότε και πόσο συχνά θα πρέπει να επικοινωνούν ώστε να μην δημιουργείται πρόβλημα στην ενδοεπικοινωνία. Η επικοινωνία γίνεται με ανταλλαγή μηνυμάτων (message passing). Με τον όρο αυτό εννοούμε πως δύο επεξεργαστές A και B επικοινωνούν μεταξύ τους στέλνοντας ο πρώτος στο δεύτερο ένα μήνυμα με κώδικα ή δεδομένα. Το μήνυμα μεταφέρεται διαδοχικά μεταξύ επεξεργαστών με φυσική σύνδεση, ακολουθώντας μια διαδρομή που ξεκινά από τον A επεξεργαστή και τελειώνει στον B. Οι αλγόριθμοι στην περίπτωση αυτή, προκειμένου να είναι αποδοτικοί, δεν θα πρέπει να έχουν μεγάλες απαιτήσεις επικοινωνίας. Παράδειγμα

τέτοιας αρχιτεκτονικής είναι το σύστημα Multicluster [65], το Ultracomputer [66] στο οποίο οι επεξεργαστές, μέσω ενός *Omega Network*, έχουν προσπέλαση στην κοινή τους μνήμη, που είναι διασπασμένη σε ισάριθμα με τους επεξεργαστές τμήματα, το Cosmic Cube [60] στο οποίο κάθε επεξεργαστής διαθέτει τη δική του μνήμη (Distributed Memory) ενώ στην κατηγορία αυτή μπορούμε να συμπεριλάβουμε και τις μηχανές "Ροής Δεδομένων" (Data Flow Machines) [67].

Ο τρόπος προγραμματισμού στους δύο αυτούς τύπους αρχιτεκτονικών διαφέρει ουσιαστικά και ως εκ τούτου διαφορετικοί αλγόριθμοι έχουν αναπτυχθεί ειδικά για την κάθε περίπτωση [68] [69], ενώ έχουν μελετηθεί χωριστά και τα βασικά χαρακτηριστικά τους [70] [71]. Στην προσπάθεια για εκμετάλλευση των πλεονεκτημάτων των δύο αρχιτεκτονικών έχουν μελετηθεί και κατασκευαστεί συστήματα που μπορούν να λειτουργούν και στη μια μορφή και στην άλλη. Χαρακτηριστικό τέτοιο παράδειγμα είναι το σύστημα PASM [72] που δημιουργήθηκε για εφαρμογές Μηχανικής Όρασης. Επίσης το σύστημα PUMPS στο οποίο έχουν προσαρμοστεί τμήματα VLSI τα οποία εκτελούν ειδικές διεργασίες επεξεργασίας εικόνων ενώ οι επεξεργαστές μέσω ενός δικτύου έχουν προσπέλαση σε μια Κατανεμημένη μνήμη (Shared memory).

Αρχιτεκτονικές PRAM

Ένα ζήτημα με μεγάλη σημασία στις παράλληλες μηχανές αφορά τα χαρακτηριστικά της μνήμης. Προβλήματα ασυνέπειας (inconcistency), ορθότητας (integrity), καθώς και έντονου ανταγωνισμού στις προσπελάσεις (contention) σε ορισμένα σημεία της μνήμης (hot spots) από πολλούς επεξεργαστές συγχρόνως, έχουν γίνει κίνητρο για προβληματισμό και έρευνα εξαιτίας της σημαντικής επίδρασης που έχουν στην απόδοση του συστήματος [73]. Ένα μοντέλο που έχει προταθεί τα τελευταία χρόνια είναι αυτό της Παράλληλης Μνήμης Τυχαίας Προσπέλασης (Parallel Random Access Memory - PRAM) που είναι μια νέα κλάση αρχιτεκτονικής υπολογιστών [18].

Δύο συστήματα που θεωρείται πως χρησιμοποιούν προσομοίωση αυτής της μνήμης λόγω της ταχύτητας ενδοεπικοινωνίας που διαθέτουν, είναι το RP3, που δημιουργήθηκε στα πλαίσια ενός ερευνητικού προγράμματος της IBM, και ένα εμπορικό προϊόν, το Connection Machine της TMC. Το πρώτο αποτελείται από 512 ισχυρούς επεξεργαστές που μοιράζονται μία κοινή

μνήμη (Shared Memory) [74] [75], είναι αρχιτεκτονικής MIMD και είναι εφοδια
νισμούς που ενισχύουν τη λειτουργία του δίκτυου ενδοεπικοινωνίας μειώνοντας
που προέρχονται από αυξημένη ζήτηση ενός συγκεκριμένου σημείου της μνήμι

Το Connection Machine είναι μία παράλληλη μηχανή βασισμένη στα κυτ
(cellular automata) [76] [77]. Αποτελείται, στην πλήρη του διάταξη, από 6
στές του ενός bit και ο κάθε ένας διαθέτει μνήμη 1Kbyte δυνατότητα για ε
λεπτής διαμέρισης (fine granularity) [78]. Στο παράρτημα γίνεται μια σ
σίαση των κυριότερων χαρακτηριστικών της αρχιτεκτονικής αυτής, η οποία
στις υλοποιήσεις της διατριβής αυτής. Το πολύ πλούσιο δίκτυο διασύνδεσι
του, του επιτρέπουν να θεωρείται ως προς τη μνήμη, καλή προσομοίωση του
τανεμημένης Παράλληλης Μνήμης Τυχαίας Προσπέλασης (Distributed P
διασύνδεσης θεωρείται επίσης προγραμματιζόμενο. Το Connection Mach
αυτών των χαρακτηριστικών του, έχει τραβήξει το ενδιαφέρον πολλών ερ
δημιουργηθεί πολλοί αλγόριθμοι για εφαρμογές στην Μηχανική 'Οραση
αυτό, όπως και άλλα, έχουν ενσωματωμένες στον εξοπλισμό τους κάποι
(primitives) τις οποίες εκτελούν σε πολύ μικρό χρόνο ορισμένες κοινές σ
σίες, όπως η ταξινόμηση, εύρεση μέγιστης και ελάχιστης τιμής μίας μεταβ
όλοι οι επεξεργαστές του συστήματος κ.ά. Αυτές λέγονται πράξεις σάρε
operations). Ακόμα, τέτοιες στοιχειώδεις πράξεις μπορούν να συνδυαστε
και να συνθέσουν πολυπλοκότερες πράξεις [80].

Νευρωνικά Δίκτυα

Μια άλλη αρχιτεκτονική, που έχει εισαχθεί από τη δεκαετία του 50, αλλά
πρόσφατα, είναι τα Νευρωνικά Δίκτυα (Neural Networks) [81] [82].
δανείζεται τη φιλοσοφία της από τη βιολογία και προσπαθεί να εξομει
ανθρώπινου εγκεφάλου κατά τρόπο μη αλγοριθμικό [83]. Σχηματίζετα
δυνατότητα να μαθαίνει και το οποίο, στις πιο εξελιγμένες μορφές τ
μεγάλο πλήθος από επεξεργαστές, οι οποίοι συνδέονται κατά μη γραμμ
πλούσιου δικτύου επικοινωνίας. Πολλές εφαρμογές έχουν γίνει με Ν
στην περιοχή της μηχανικής αναγνώρισης αντικειμένων και φωνής [84

Κατανεμημένη Επεξεργασία

Η παράλληλη επεξεργασία εμφανίζεται και με μια διαφορετική μορφή που λέγεται Κατανεμημένη Επεξεργασία (Distributed Processing), όπου έχουμε τη συνεργασία πολλών όμοιων (Ομοιογενές Σύστημα) ή διαφορετικών (Ετερογενές Σύστημα) υπολογιστικών συστημάτων, που συνδέονται μέσω ενός δικτύου. Η Κατανεμημένη Επεξεργασία έχει διαφοροποιημένα χαρακτηριστικά από την Παράλληλη Επεξεργασία. Εχει μελετηθεί αρκετά ενώ έχουν υλοποιηθεί πάρα πολλά Κατανεμημένα Συστήματα σε Πανεπιστήμια, Ερευνητικά Κέντρα και βιομηχανίες και οι αναφορές που θα μπορούσαν να γίνουν σε βιβλιογραφία είναι εξαιρετικά εκτενείς [86]. Ομως δεν συμπεριλαμβάνεται στο πεδίο εξέτασης της συγκεκριμένης έρευνας.

Τα δίκτυα επικοινωνίας των επεξεργαστών έχουν τύχει ιδιαίτερης μελέτης εξαιτίας της επίδρασής τους στην απόδοση τόσο των Κατανεμημένων όσο και των Παράλληλων Συστημάτων. Πολλές φορές παρατηρούνται σημαντικές καθυστερήσεις σ'αυτά κατά την εκτέλεση διεργασιών γενικά αλλά και ειδικότερα διεργασιών για την Ανάλυση Εικόνων και αυτό υπήρξε κίνητρο για έρευνα [87] [88]. Επίσης έχει δημιουργηθεί λειτουργικό σύστημα (MOS) για κατανεμημένο σύστημα υπολογιστών [89].

3.2.2 Αλγόριθμοι

Δόθηκε ήδη η ευκαιρία, μιλώντας για ερευνητική δουλειά που έχει γίνει πάνω σε αρχιτεκτονικές υπολογιστικών συστημάτων, να αναφερθούμε και σε εργασίες που έγιναν για παράλληλους αλγόριθμους. Η βιβλιογραφία που υπάρχει πάνω στο θέμα είναι ιδιαίτερα εκτενής. Μεγάλος αριθμός αλγορίθμων για πολλές διεργασίες και για πολλές εναλλακτικές λύσεις εφαρμογών σε διάφορες αρχιτεκτονικές έχουν υλοποιηθεί. Εδώ θα περιοριστούμε σε ορισμένες αναφορές και παρατηρήσεις με γενικότερη ισχύ.

Ο Fox [90] κάνοντας μια ανασκόπηση σε 84 από τις κυριότερες εφαρμογές παράλληλων μηχανών προσπαθεί να ταξινομήσει τα προβλήματα σε Σύγχρονα, Ασύγχρονα και Ελαστικά Σύγχρονα (loosely synchronous). Η ταξινόμηση αυτή χωρίζει τα προβλήματα σε εκείνα που είναι κατάλληλα για αρχιτεκτονικές SIMD ή MIMD. Επιβεβαιώνει την άποψη πως τα ασύγχρονα προβλήματα είναι εκείνα που χρειάζονται τη μεγαλύτερη διερεύνηση για την παραλλη-

λοποίηοή τους, ενώ οημειώνει πως όλες οι επιτυχημένες εφαρμογές αναφέρονται σε σύγχρονα προβλήματα. Ακόμη παρατηρεί πως οι μισές περίπου από τις επιστημονικές εφαρμογές εφαρμόζονται πολύ καλά σε SIMD μηχανές, ενώ οι άλλες μισές θα μπορούσαν να εκμεταλλευτούν τα πλεονεκτήματα των MIMD.

Στη βιβλιογραφία αναφέρονται αλγόριθμοι για συγκεκριμένη διεργασία και για συγκεκριμένη αρχιτεκτονική, π.χ. του Υπερκύβου [91] [62] [92], και άλλοι που εξετάζουν ομοειδείς διεργασίες, όπως ταξινόμηση (sorting), σύνδεση λιστών (merging) κ.ά. σε ομοειδείς τοπολογίες, όπως δένδρα και πυραμίδες [54]. Επίσης εμφανίζονται διάφοροι αλγόριθμοι που είναι βέλτιστοι για την ίδια διεργασία αλλά προσαρμοσμένοι σε διαφορετικές τοπολογίες [54] [93] και [94] [95].

Για ορισμένες από τις βασικές αρχιτεκτονικές έχει γίνει πάρα πολλή ερευνητική εργασία στην προσπάθεια να διελευκανθούν όλες οι πτυχές των προβλημάτων που εμφανίζονται στις υλοποιήσεις ευρέως χρησιμοποιούμενων διεργασιών. Τέτοιες αρχιτεκτονικές είναι αυτές του Ορθογωνικού Πλέγματος [96] [97] [98][99] [100] και του Υπερκύβου, για τις οποίες υπάρχουν παράλληλοι αλγόριθμοι για γράφους [101], για Υπολογιστική Γεωμετρία [102] [103], καθώς και συγκριτικές μελέτες των δύο τοπολογιών για τέτοια προβλήματα [104].

Ειδικά για την Υπολογιστική γεωμετρία υπάρχει ένας μεγάλος όγκος βιβλιογραφίας για την επίλυση γεωμετρικών προβλημάτων σε παράλληλα συστήματα με τοπολογία πλέγματος. Οι περισσότεροι από αυτούς είναι βέλτιστοι (δηλαδή τρέχουν σε χρόνο $O(N^{\frac{1}{d}})$ σε ένα πλέγμα d διαστάσεων με N επεξεργαστές.

Στην [105] αναπτύσσονται αλγόριθμοι O(N) για τον προσδιορισμό γεωμετρικών ιδιοτήτων σχημάτων που εμφανίζονται σε μια δυαδική εικόνα NxN. Οι αλγόριθμοι αυτοί είναι αναπτυγμένοι για αρχιτεκτονικές πλέγματος (Mesh connected).

Στην [102] αναπτύσσεται αλγόριθμος με πολυπλοκότητα $O(log^3 n)$ για την κατασκευή του διαγράμματος Voronoi n σημείων σε αρχιτεκτονική με κοινή CREW (Concurrent Read - Exclussive Write) μνήμη και $O(log^2 n)$ για αρχιτεκτονική με μνήμη CRCW (Concurrent Read - Concurrent Write). Χρησιμοποιώντας μια νέα παράλληλη "Διαίρει και Βασίλευε" ("Devide and Conquer") τεχνική οι Atallah και Goodrich αναπτύσσουν αλγορίθμους για διάφορα

προβλήματα Υπολογιστικής Γεωμετρίας σε παράλληλες μηχανές με κοινή CREW (Concurrent Read - Exclussive Write) μνήμη [106].

Ο Goodrich παρουσιάζει δύο βέλτιστους αλγορίθμους για αρχιτεκτονική CREW-PRAM για το πρόβλημα της τακτοποίησης (arrangement) N γραμμών στο επίπεδο [107]. Ο Atallah κάνει μια ενδιαφέρουσα επισκόπηση τεχνικών που έχουν αναπτυχθεί για προβλήματα Υπολογιστικής Γεωμετρίας σε παράλληλα Συστήματα κοινής και κατανεμημένης Μνήμης [108] Οι Atallah και Tsay αποδεικνύουν πως μπορεί να επιτευχθεί η ίδια επιτάχυνση για ορισμένα προβλήματα Υπολογιστικής Γεωμετρίας, ακόμα και αν χρησιμοποιήσουμε λιγότερους επεξεργαστές από το μέγεθος του πλέγματος[109].

Στην [110] γίνεται μια ανασκόπηση των συστολικών αλγορίθμων για την περιοχή αυτή των προβλημάτων σε διάφορα είδη παράλληλων επεξεργαστών. Στην[111] παρουσιάζονται βέλτιστοι αλγόριθμοι για διάφορα προβλήματα, όπως η εύρεση του Κυρτού Περιγράμματος, υπολογισμού του Ιστογράμματος, Συνδεδεμένων Συστατικών κ.ά. σε συγκεκριμένη αρχιτεκτονική με επεξεργαστές σε γραμμική διάταξη. Εξάλλου ο Γκίμπας [112] παρουσίασε νέες τεχνικές της μορφής "Διαίρει και Βασίλευε", για προβλήματα Υπολογιστικής Γεωμετρίας, όπου χρησιμοποιείται η τυχαία δειγματοληψία προκειμένου να αποφευχθεί η υπερδιάσπαση του προβλήματος αφενός και να κρατηθεί μια ισοκατανομή στο βάρος των υποπροβλημάτων αφετέρου.

Ομως και συγκεκριμένα προβλήματα έχουν τύχει εκτεταμένης διερεύνησης και επίλυσης για διάφορες αρχιτεκτονικές. Για παράδειγμα ο υπολογισμός του μετασχηματισμού Hough που έχει αντιμετωπιστεί για ένα μονοδιάστατο πίνακα (Array) επεξεργαστων [113], σε MIMD ασύγχρονο σύστημα γενικού σκοπού [114], σε πλέγμα επεξεργαστών SIMD [96] και [68], σε Multiple-Single Instruction-Multiple Data (MSIMD) επεξεργαστές σε ιεραρχική τοπολογία [115] κ.ά.

Οπως αντίστροφα έχει γίνει η διερεύνηση της υλοποίησης πλήθους αλγορίθμων σε συγκεκριμένες γνωστές τοπολογίες, όπως για παράδειγμα εκείνης του Υπερκύβου (Hypercube), τόσο για διεργασίες ανάλύσης εικόνων [62] [69],[91],[116],[103] όσο και για διεργασίες γενικότερου ενδιαφέροντος [63], [91],[92],[117],[118]. Ανάλογα υπάρχουν πολλοί αλγόριθμοι για την τοπολογία του πλέγματος (Mesh) [119][120], [100],[98],[93], [121],[122],[104],[68],[96] κ.ά.

Το πώς επηρεάζει η διαμέριση του προβλήματος, αλλά και η απεικόνισή του πάνω στην τοπολογία, την απόδοση της μηχανής είναι ένα θέμα που έχει μελετηθεί σε κάποιο βαθμό [123] [16] ενώ στην [124] παρουσιάζεται μια γλώσσα που σκοπεύει στην καλύτερη συνεργασία αρχιτεκτονικής και διεργασίας. Στην [125] μελετάται η επίδραση των χαρακτηριστικών ορισμένων αλγορίθμων, όπως Ταξινόμησης, επίλυσης Μερικών Διαφορικών Εξισώσεων, FFT και ο εξομοιωτής ενός σιδηροδρομικού δικτύου στην απόδοσή τους σε μια συγκεκριμένη αρχιτεκτονική, την CM*, με 50 επεξεργαστές σε ιεραρχική διάταξη.

Πρόσφατη εργασία [126] αναφέρεται στη δημιουργία ενός ολοκληρωμένου τεστ (benchmark) για παράλληλη επεξεργασία στην Ανάλυση Εικόνων. Για το σκοπό αυτό ορίσθηκαν ορισμένες διεργασίες μέσου και υψηλού επιπέδου που αποτελούσαν τα διαδοχικά βήματα ενός προβλήματος αναγνώρισης αντικειμένων. Σκοπός της εργασίας αυτής ήταν να γίνουν περισσότερο κατανοητές οι απαιτήσεις από τις αρχιτεκτονικές για την Μηχανική 'Οραση και να εντοπιστούν τα σημεία στα οποία υπάρχουν προβλήματα για ορισμένες αρχιτεκτονικές ώστε οι τελευταίες να μπορούν στο μέλλον να προδιαγραφούν καλύτερα.

Προκειμένου να ελαττώσουν την απαιτούμενη υπολογιστική ισχύ ορισμένων αλγορίθμων οι Fischler και Firschein προτείνουν ένα φορμαλισμό για τους αλγορίθμους ανάλυσης εικόνων που στηρίζεται στη δυνατότητα που υπάρχει στις περιπτώσεις αυτές να μαντεύεται, με παράλληλες μεθόδους, μια καλή απάντηση. Διότι, όπως ισχυρίζονται, συχνά είναι πολύ γρηγορότερο να επαληθεύεις μια υπόθεση από το να υπολογίζεις εξαρχής την απάντηση [127]

Ενα βασικό πάντως συμπέρασμα είναι πως πριν εξάγουμε συγκεκριμένα όρια για την απόδοση των αλγορίθμων μας, θα πρέπει να κοιτάξουμε τα χαρακτηριστικά των μηχανών στις οποίες αναφέρονται, διότι είναι πολύ πιθανό να υπάρχουν σε ορισμένες περιπτώσεις πολύ καλύτερα όρια [128]

3.2.3 Ισοκατανομή Φορτίου (Load Balancing)

Εχει αποδειχθεί ότι σε ένα μεγάλο δίκτυο από αυτόνομους κόμβους (επεξεργαστές) και υπό έντονη χρήση, υπάρχει μεγάλη πιθανότητα ένας τουλάχιστον κόμβος να παραμένει αδρανής, ενώ σε κάποιον άλλο να συσσωρεύονται εργασίες (Tasks) [129]. Αυτό έχει σαν αποτέλεσμα τη

συνολική πτώση της απόδοσης του συστήματος. Ως εκ τούτου υπάρχει η ανάγκη για ανάπτυξη αλγορίθμων που θα έχουν σκοπό την ισοκατανομή των φορτίων μεταξύ των επεξεργαστών.

Τρεις είναι οι κύριες συνιστώσες μιας πολιτικής για ΙΦ :

* Η γενική απόφαση για το αν πρέπει ή όχι να εφαρμοστεί κάποιος αλγόριθμος ανακατανομής του φορτίου των επεξεργαστών.

* Η πολιτική μεταφοράς που καθορίζει αν θα πρέπει το φορτίο, δεδομένα (data) ή διεργασία (task), να παραμείνει στον συγκεκριμένο επεξεργαστή (locally), ή να σταλεί σε κάποιον άλλο (remotely)

* Η πολιτική εύρεσης του αποδέκτη, που καθορίζει τον επεξεργαστή στον οποίο θα σταλεί το φορτίο.

Μπορούν να χρησιμοποιηθούν από απλές μέχρι πολύ πολύπλοκες μέθοδοι για ΙΦ. Οι πιο πολύπλοκες μπορεί να έχουν δυνατότητες για καλύτερη ΙΦ, όμως απαιτούν πολύπλοκους αλγορίθμους, καθώς επίσης συλλογή και φύλαξη (bookeeping) μεγάλων ποσοτήτων πληροφοριών. Γενικά, είναι συζητήσιμη η διαφορά των αποτελεσμάτων τους σε σχέση με αυτά των απλών (και λιγότερο "έξυπνων") αλγορίθμων. Οι τελευταίοι φαίνεται να έχουν σαν αποτέλεσμα σημαντική βελτίωση στην απόδοση ενός Συστήματος Πολυεπεξεργαστών και μάλιστα για ένα μεγάλο εύρος παραμέτρων του Συστήματος [129]. Αυτή η σύγκριση είναι γενικά δύσκολη, λόγω των παραμέτρων που εμπλέκονται και που είναι μεταξύ άλλων :

* Ο μέσος χρόνος εξυπηρέτησης μιας διεργασίας

* Το μέσο κόστος μεταφοράς φορτίων

* Το πλήθος των επεξεργαστών

* Η τοπολογία και οι καθυστερήσεις στο δίκτυο επικοινωνίας

* Η φύση του φορτίου (δεδομένα, κώδικας κτλ.)

Ως προς το πότε θα γίνεται η ανακατανομή του υπολογιστικού φορτίου υπάρχουν δύο

βασικές επιλογές. Εκείνη της Στατικής Ισοκατανομής Φορτίου (Static Load Balancing) και εκείνη της Δυναμικής Ισοκατανομής Φορτίου (Dynamic Load Balancing)

Στη Στατική Ισοκατανομή, η απόφαση για μεταφορά φορτίου (εργασιών ή δεδομένων) από έναν επεξεργαστή σε άλλον, δεν εξαρτάται από την κατάσταση του Συστήματος την κάθε στιγμή αλλά μάλλον από μία μέση κατάστασή του [129][130]. Στην [130] παρουσιάζεται αλγόριθμος για τη μελέτη της επίπτωσης των αποτελεσμάτων της ταχύτητας του Δικτύου στην απόδοση αλγορίθμων ισοκατανομής του φορτίου. Στη βιβλιογραφία αναφέρονται αρκετοί στατικοί αλγόριθμοι [131], [132] ο πρώτος εκ των οποίων είναι για ανομοιόμορφα προβλήματα στα οποία επιτυγχάνεται η ισοκατανομή με μια δυαδικό αναδρομικό διαχωρισμού των δεδομένων στα οποία θα εφαρμοστεί η διεργασίας.

Στη Δυναμική ή Προσαρμοζόμενη (Adaptive) Ισοκατανομή Φορτίων η ανακατανομή των φορτίων των επεξεργαστών ενός Συστήματος γίνεται λαμβάνοντας υπόψη την εκάστοτε κατάστασή του. Νέα ανακατανομή των υπολογιστικών φορτίων είναι δυνατή ακόμα και μετά την εκκίνηση της επεξεργασίας. Η βασική ιδέα είναι ότι υπάρχει ένας κρίσιμος παράγοντας φορτίου (critical Load Factor), τέτοιος ώστε, όταν το φορτίο σε έναν επεξεργαστή πέσει κάτω από την τιμή αυτού του παράγοντα, τότε ένα τμήμα του (module) M εκχωρείται σ'αυτόν τον επεξεργαστή. Η εκχώρηση αυτή δεν θα πρέπει να δημιουργεί υπερχείλιση μνήμης (overflow) στον αποδέκτη επεξεργαστή. Ο χρόνος εκτέλεσης ενός τμήματος του προγράμματος είναι γνωστός για κάθε επεξεργαστή και από αυτόν εξαρτάται ο κρίσιμος παράγοντας. Επίσης είναι γνωστό το κόστος μεταφοράς του φορτίου από τον ένα επεξεργαστή στον άλλο [133]. Η απαίτηση όμως να αντιδρά το σύστημα στις μεταβολές της κατάστασης των φορτίων του, σημαίνει πως η ενδογενής έλλειψη ακρίβειας και οι γρήγορες μερικές φορές εναλλαγές της κατάστασης αυτής, μπορεί να έχουν σαν αποτέλεσμα την ασταθή συμπεριφορά της πολιτικής αυτής.

Η απόδοση μιας Στατικής τεχνικής Ισοκατανομής Φορτίου, εξαρτάται από την ακρίβεια της τεχνικής, της αρχικής (και μόνης) εκχώρησης φορτίων. Αντίθετα, στη Δυναμική Ισοκατανομή, η αποδοτικότητά της εξαρτάται από την αποτελεσματικότητα της τεχνικής της αποστολής των φορτίων ανάμεσα στους επεξεργαστές. Συγκριτική μελέτη μεταξύ Δυναμικών και Στατικών μεθόδων Ισοκαταναμής Φορτίων [134] έδειξε πως μπορούμε να επιτύχουμε απλές και ικανοποιητικά γρήγορες, αν και υποβέλτιστες (sub-optimal) μεθόδους, ενώ δυναμικές μέθοδοι που αποδίδουν ήδη αρκετά καλά μπορούν να αποδώσουν ακόμα καλύτερα αν συνδυαστούν αρχικά

με κάποια στατική μέθοδο.

Μοντέλα ουρών (Queue models) και στοχαστικά μοντέλα για την κατανομή και κυκλοφορία (routing) των φορτίων έχουν χρησιμοποιηθεί ευρέως. Εχουν δε δημοσιευθεί και συγκριτικές εργασίες με αιτιοκρατικές (Deterministic) μεθόδους [135]. Οι Nicol και Saltz διερεύνησαν έναν εμπειρικό αλγόριθμο δυναμικής ανακατανομής του φορτίου [136], που εφαρμόζεται σε δύο στοχαστικά μοντέλα χρησιμοποιώντας αλυσίδες Markov. Η βέλτιστη λύση στην περίπτωση αυτή προϋποθέτει ορισμένες κανονικές (regular) συνθήκες και απαιτεί την επίλυση ενός συστήματος εξισώσεων. Σε περίπτωση μη ικανοποίησης των συνθηκών η βέλτιστη λύση γίνεται υπο-βέλτιστη (sub-optimal), ενώ όταν το πρόβλημα μεγαλώνει, η πολυπλοκότητα της επίλυσης του συστήματος την καθιστά αναποτελεσματική. Γι'αυτό το λόγο εξετάζουν μια ευρηματική λύση (heuristics) που προσπαθεί να ελαχιστοποιήσει το μέσο χρόνο που ένας επεξεργαστής παραμένει αδρανής σε κάθε βήμα της επεξεργασίας. Οι ίδιοι πρότειναν έναν άλλο ευρηματικό αλγόριθμο δυναμικής ισοκατανομής του φορτίου για μη κανονικές (irregular) υπολογιστικές απαιτήσεις [137] . Το μοντέλο του προβλήματος δίνεται σε δύο μορφές. Στην πρώτη χωρίζεται το πεδίο των δεδομένων σε μη κανονικά τμήματα. Στη δεύτερη, διατηρούνται ίδια τα τμήματα αλλά μεταβάλλεται η συχνότητα των συγχρονισμών τους. Θεωρούν ακόμα πως γνωρίζουν εκ των προτέρων την υπολογιστική εξέλιξη του προβλήματος καθώς επίσης το επικοινωνιακό του κόστος.

Τεχνικές ισοκατανομής του φορτίου έχουν μελετηθεί και γενικά για ορισμένες τοπολογίες γενικού σκοπού όπως εκείνη του Υπερκύβου. Στην [138] γίνεται μια ανασκόπηση τέτοιων τεχνικών για την τοπολογία αυτή ενώ στις [132, 139]. Στις [140, 141] γίνεται ανάλυση της επίπτωσης που έχει η ανισοκατανομή του φορτίου στην απόδοση πολυεπεξεργαστών τύπου SPMD (Single Program - Multiple Data) εξαιτίας μη αλγοριθμικών αιτιών όπως η έλλειψη συγχρονισμού τους.

Πειραματισμοί ειδικά σε εφαρμογές Ανάλυσης Εικόνων με διεργασίες μέσου και υψηλού επιπέδου σε ένα MIMD σύστημα έδειξαν πως οι επεξεργαστές θα πρέπει να κατευθύνονται δυναμικά σε εκείνα τα τμήματα της εικόνας που έχουν το μεγαλύτερο ενδιαφέρον [142].

Ενας αλγόριθμος που εφαρμόζεται σε διεργασίες ανάλυσης εικόνων και παρουσιάζει ενδιαφέρον είναι εκείνος που υλοποιείται στην [22] όπου δυναμικά ανακατανέμεται το φορτίο των

επεξεργαστών κάνοντας χρήση των ειδικών δυνατοτήτων των αρχιτεκτονικών στις οποίες εφαρμόζεται και με την προϋπόθεση πως κάθε επεξεργαστής είναι σε θέση να γνωρίζει το φορτίο του.

Αλγόριθμοι ισοκατανομής του φορτίου έχουν αναπτυχθεί και ειδικά για Κατανεμημένα Συστήματα. Για παράδειγμα στην [143] γίνεται μια από τις πρώτες προσεγγίσεις του προβλήματος για ένα κατανεμημένο σύστημα με δύο επεξεργαστές. Επόμενες εργασίες εμφανίζονται τόσο για ομογενή κατανεμημένα συστήματα [144, 145, 130, 129, 146, 147] , όσο και για ετερογενή [135].

Τμηματοποίηση των Διεργασιών

Η Τμηματοποίηση των Διεργασιών (Task Partitioning) είναι μια ειδική περίπτωση επίλυσης του προβλήματος της Στατικής Ισοκατανομής Φορτίου και συνίσταται στο να κάνει κανείς βέλτιστη τοποθέτηση (allocation) των επιμέρους εργασιών ενός παράλληλου προγράμματος στους επεξεργαστές ενός παράλληλου συστήματος με σκοπό να ελαχιστοποιήσει το συνολικό χρόνο εκτέλεσης. Το πρόβλημα είναι NP-complete, εκτός ορισμένων περιπτώσεων με μάλλον όχι ρεαλιστικούς περιορισμούς [148]. Πάνω σ'αυτό το θέμα έχει γίνει πολύ ερευνητική δουλειά με μελέτες σε ορισμένα μοντέλα και για συγκεκριμένες εφαρμογές. Η διάσπαση (decomposition) ενός προβλήματος με σκοπό την ισοκατανομή του σε ένα παράλληλο σύστημα μπορεί να αναχθεί σε γραφοθεωρητικό (graph theory) πρόβλημα και συγκεκριμένα στη βέλτιστη τμηματοποίηση (partitioning) ενός γράφου σε $N = 2^k$ υπογράφους [149, 150, 151, 152, 133, 153, 154, 155, 156]. Στατικοί ευρηματικοί υποβέλτιστοι αλγόριθμοι κατανεμημένων ομογενών συστημάτων φαίνεται επίσης να δίνουν καλά αποτελέσματα [144, 157, 158, 159]. Έχει δημιουργηθεί δίκτυο υπολογιστών στηριγμένο στο λειτουργικό σύστημα Unix με δυνατότητες ισοκατανομής των φορτίων μεταξύ των υπολογιστών [160]. Ενώ υπάρχουν αντιμετωπίσεις του προβλήματος για συγκεκριμένες αρχιτεκτονικές [161]

Για ένα κατανεμημένο τοπικό σύστημα επεξεργαστών προτείνεται ένα πρωτόκολλο ανακατανομής του φορτίου που έχει σταθερό επιπρόσθετο κόστος (overhead) και στηρίζει την ανακατανομή στην εύρεση ενός μέγιστου αριθμού μεταξύ των υπολογιστών. Ο αριθμός αυτός αντιπροσωπεύει το φορτίο τους [146]. Ακόμη, ένας ευρηματικός αλγόριθμος που κατανέμει

δυναμικά διεργασίες σε ένα τοπικά κατανεμημένο σύστημα υπολογιστών δημιουργήθηκε για την εξυπηρέτηση ενός συστήματος κατανεμημένης βάσης δεδομένων (distributed data base) [147]. Οι Nicol και Saltz δίνουν ένα αλγόριθμο στον οποίο γίνεται η ανάθεση των επιμέρους διεργασιών στους επεξεργαστές για διεργασίες με το χαρακτηριστικό ότι το φορτίο που θα δημιουργήσουν δεν μπορεί ή κοστίζει πολύ να προβλεφθεί [162] Στην [163] αντιμετωπίζεται το πρόβλημα της δυναμικά εκ νέου ανάθεσης (remapping) διεργασιών στους επεξεργαστές όταν η εξέλιξη χωρίζεται σε μια σειρά από φάσεις, στη διάρκεια της κάθε μίας από τις οποίες η απαιτούμενη υπολογιστική ισχύς παραμένει σταθερή, αλλά είναι μη προβλέψιμη για την επόμενη φάση, και αντιμετωπίζεται πιθανοκρατικά.

Το Transparent Process Cloning (TPC) είναι μία τεχνική λειτουργικού συστήματος που επιτρέπει την αναπαράσταση μιας σειριακής διεργασίας σαν ένα σύνολο παράλληλων στιγμιότυπων. Το σύστημα αυτό έχει έναν κεντρικό έλεγχο στο σύστημα επιτυγχάνοντας την αύξηση της παραλληλίας και τη μείωση της επικοινωνίας μεταξύ τους [164].

Κεφάλαιο 4

Συνιστώσες της Παράλληλης Υλοποίησης Διεργασιών Ανάλυσης Εικόνων

4.1 Γενικά

Στο κεφάλαιο αυτό θα παρουσιάσουμε αναλυτικά τις τέσσερις βασικές συνιστώσες (components) του προβλήματος της παράλληλης υλοποίησης διεργασιών Ανάλυσης Εικόνων, με τα ιδιαίτερα χαρακτηριστικά τους, τα οποία θα χρησιμοποιήσουμε στο επόμενο κεφάλαιο για την παραμετροποίηση του προτεινόμενου μοντέλου βελτίωσης της απόδοσης ενός παράλληλου συστήματος επεξεργασίας εικόνων. Οι συνιστώσες αυτές είναι:

1. Η συγκεκριμένη διεργασία ανάλυσης της εικόνας (Image Analysis Task).
2. Η χρησιμοποιούμενη παράλληλη αρχιτεκτονική (Architecture - Topology).
3. Το περιεχόμενο της εικόνας (Image Content).

4. Η σκοπιμότητα και δυνατότητα γρήγορης ανακατανομής του φορτίου ανάμεσα στους επεξεργαστές, με σκοπό την ισοκατανομή του (Load Balancing).

Κάθε μία από τις 4 αυτές συνιστώσες, ανάλογα με το πόσο καλά συνεργάζεται με τις υπόλοιπες, επηρεάζει θετικά ή αρνητικά τη συνολική αποτελεσματικότητα (performance) της υλοποίησης. Για τη μέτρηση αυτής της αποτελεσματικότητας δύο είναι τα βασικά ποσοτικά μεγέθη που αναφέραμε ήδη, η *επιτάχυνση* και η *απόδοση*. Όμως έχουν προταθεί και διάφορα άλλα μεγέθη που αντιπροσωπεύουν το πόσο καλά χρησιμοποιείται το παράλληλο σύστημα. Τέτοια είναι *ο χρόνος εκτέλεσης (execution time)*, *η ταχύτητα (speed)* εκτέλεσης μονάδων δεδομένων στη μονάδα του χρόνου και *η χρησιμοποίηση (utilization)* των επεξεργαστών, ενώ έχουν επίσης προταθεί μεγέθη που συσχετίζουν την απόδοση μιάς μηχανής με το κόστος της και έχουν χρησιμοποιηθεί για να διερευνηθεί η αλληλεπίδραση του μεγέθους της μηχανής, του μεγέθους του προβλήματος και του τύπου του δικτύου [165].

Κάθε μια από τις επιμέρους διεργασίες για τη βέλτιστη υλοποίησή της σε ένα παράλληλο σύστημα έχει διαφορετικές απαιτήσεις αρχιτεκτονικής, τόσο σε υπολογιστική ισχύ όσο και σε ενδοεπικοινωνία, χρησιμοποιεί διαφορετικές δομές δεδομένων, ενώ μπορεί να χρειάζεται ή να μην χρειάζεται την εφαρμογή αλγορίθμου για ισοκατανομή του φορτίου.

Το περιεχόμενο μιας εικόνας αποκτά διαφορετική έννοια ανάλογα με τη διεργασία που πρόκειται να εκτελέσουμε. Για κάθε διεργασία, ορισμένα από τα χαρακτηριστικά που συνθέτουν το περιεχόμενο της εικόνας είναι σημαντικά ενώ άλλα είναι αδιάφορα. Το ίδιο ισχύει και για τις τρεις άλλες συνιστώσες. Η γνώση των χαρακτηριστικών τους μπορεί να δώσει ένα μέτρο της συμβατότητάς τους και να συμβάλλει στην πρόβλεψη της αρμονικής ή όχι συνεργασίας τους. Ας υποθέσουμε, για παράδειγμα, ότι χρησιμοποιούμε αρχιτεκτονική που αποτελείται από μεγάλο αριθμό επεξεργαστών. Εάν η φύση της διεργασίας της επιτρέπει να χωριστεί σε πολλά μέρη χωρίς πρόσθετο υπολογιστικό κόστος ή πρόσθετο κόστος επικοινωνίας (computational or communication overhead) τότε τα μεγέθη της επιτάχυνσης και της απόδοσης του συστήματος μπορεί να πάρουν υψηλές τιμές. Αντίθετα, αν η υλοποίησή της σε αρχιτεκτονική υψηλής διαμέρισης απαιτεί μεγάλη πρόσθετη ενδοεπικοινωνία και υπολογισμούς, τότε μειώνεται η απόδοση του συστήματος. Στην περίπτωση αυτή είναι πολύ πιθανό πως θα ήταν αποδοτικότερη η υλοποίηση της διεργασίας σε συστήματα με λίγους επεξεργαστές. Προκειμένου λοιπόν να επιτύχουμε την αποτελεσματική παράλληλη υλοποίηση διεργασιών Ανάλυσης Εικόνων, είναι

σημαντικό να ληφθούν υπόψη τα χαρακτηριστικά των τεσσάρων βασικών συνιστωσών.

Στις επόμενες παραγράφους περιγράφουμε τα βασικότερα γνωρίσματα των τεσσάρων συνιστωσών του προβλήματος αρχίζοντας από εκείνα των διεργασιών Ανάλυσης Εικόνων.

4.2 Γνωρίσματα των Διεργασιών Ανάλυσης Εικόνων

Τα χαρακτηριστικά των διεργασιών (Task) που χρησιμοποιούνται στην Ανάλυση Εικόνων χωρίζονται σε 3 βασικές κατηγορίες: Η πρώτη περιλαμβάνει τα χαρακτηριστικά εκείνα που αναφέρονται στις απαιτήσεις μιας διεργασίας σε υπολογισμούς και την πολυπλοκότητα τους (Computation Characteristics). Η δεύτερη αποτελείται από εκείνα τα χαρακτηριστικά που αφορούν στις ανάγκες της διεργασίας σε ενδοεπικοινωνία μεταξύ των επεξεργαστών που έχουν αναλάβει την εκτέλεσή της (Communication Characteristics). Τέτοια χαρακτηριστικά έχουν να κάνουν τόσο με την ενδοεπικοινωνία που δημιουργείται όταν διαμερίσουμε ένα πρόβλημα σε P επεξεργαστές όσο και με την επιπλέον ενδοεπικοινωνία (overhead) που δημιουργείται αν αυτή η διαμέριση δεν γίνει κατάλληλα. Αυτό συμβαίνει με την άνιση κατανομή του φορτίου στους επεξεργαστές, την υπερβολική διαμέριση του προβλήματος (overspliting) κ.ά. Τρίτη τέλος κατηγορία χαρακτηριστικών αποτελούν εκείνα που αναφέρονται στη μορφή που πρέπει να έχουν τα δεδομένα για μια συγκεκριμένη υλοποίηση της διεργασίας. Συγκεκριμένα μας ενδιαφέρουν χαρακτηριστικά που σχετίζονται με τις απαιτούμενες δομές δεδομένων, με το ποια είναι η ελάχιστη μονάδα δεδομένων πάνω στα οποία μπορεί να υλοποιηθεί η διεργασία, καθώς επίσης και εκείνα που σχετίζονται με τη συμπεριφορά και την πιθανή μετάλλαξή τους στη διάρκεια εκτέλεσης της διεργασίας κτλ. Σημαντικά δε είναι και η δομή, μορφή και η θέση προορισμού πάνω στο υπολογιστικό σύστημα των δεδομένων εξόδου.

4.2.1 Υπολογιστικά Χαρακτηριστικά

Τα βασικά υπολογιστικά χαρακτηριστικά των διεργασιών Ανάλυσης Εικόνων είναι τα εξής:

Υπολογιστική Πολυπλοκότητα. Ο όρος αυτός εκφράζει το πλήθος των υπολογιστικών πράξεων που πρέπει να γίνουν προκειμένου να εξαχθεί το τελικό αποτέλεσμα και μας ενδιαφέρει σε δύο μορφές. Πρώτον ο συνολικός υπολογιστικός χρόνος που θα απαιτήσει η διεργασία για κάθε τμήμα στο οποίο έχει διαμεριστεί. Δεύτερο, πώς αυξάνει ο υπολογιστικός χρόνος (overhead) σαν συνέπεια της υπερβολικής διαμέρισης του προβλήματος, είτε αυτή ανάγεται σε τεμαχισμό των δεδομένων είτε σε καταμερισμό του κώδικα.

Σειριακότητα. Κάθε αλγόριθμος έχει ένα τμήμα του το οποίο μπορεί να εκτελεστεί παράλληλα και ένα άλλο το οποίο πρέπει υποχρεωτικά να εκτελεστεί σειριακά. Δηλαδή κάποιο μέρος του θα πρέπει να προηγηθεί κάποιου άλλου. Αν T_{seq} είναι ο χρόνος που απαιτεί για σειριακή εκτέλεση μέρος του αλγορίθμου και T_{tot} είναι ο συνολικός σειριακός χρόνος που απαιτεί η εκτέλεσή του, τότε η σειριακότητα ισούται με $\Sigma = \frac{T_{seq}}{T_{tot}} < 1$ ενώ $\frac{1}{\Sigma}$ είναι η μέγιστη επιτάχυνση που μπορεί να επιτευχθεί με παράλληλη υλοποίηση αν έχουμε στη διάθεσή μας απεριόριστο αριθμό επεξεργαστών. Αντίστοιχα, η *παραλληλοποιησιμότητα* της διεργασίας Π = (1 - σειριακότητα).

Λόγος υπολογιστικού χρόνου προς χρόνο επικοινωνίας Οπως έχει ήδη αναφερθεί, κάθε αλγόριθμος που υλοποιείται παράλληλα αφιερώνει ένα μέρος του χρόνου του σε υπολογιστική επεξεργασία των δεδομένων (αριθμητική ή λογική) ενώ το υπόλοιπο το αφιερώνει σε ενδοεπικοινωνία προκειμένου να ανταλλάξει πληροφορίες με άλλους επεξεργαστές που ασχολούνται με το ίδιο πρόβλημα. Τις πληροφορίες αυτές θα χρησιμοποιήσουν στη συνέχεια κάποιοι από αυτούς προκειμένου να συνεχίσουν την επεξεργασία. Το ποσοστό του χρόνου που πρέπει να αφιερώσει ο επεξεργαστής τόσο για υπολογισμούς όσο και για επικοινωνία χαρακτηρίζει τη διεργασία. Το χαρακτηριστικό αυτό εξαρτάται από τη μηχανή στην οποία γίνεται η υλοποίηση.

4.2.2 Χαρακτηριστικά επικοινωνίας

Ομοιογένεια. Υπάρχουν διεργασίες που χρειάζονται ενδοεπικοινωνία μεταξύ των επιμέρους επεξεργαστών κατά τρόπο που δεν διαφέρει από τον ένα στον άλλο ούτε τοπολογικά ούτε χρονικά. Κάθε επεξεργαστής έχει δύο επιλογές: είτε να επικοινωνήσει με κάποιον άλλο ή να παραμείνει αδρανής. Αν ο επεξεργαστής-αποδέκτης του μηνύματος μπορεί να περιγρα-

φεί κατά γενικό τρόπο για όλους τους επεξεργαστές, τότε η επικοινωνία χαρακτηρίζεται από ομοιογένεια και έχει σημαντικά πλεονεκτήματα έναντι μιας άλλης στην οποία κάθε επεξεργαστής θα έπρεπε να επικοινωνήσει με διαφορετικό τρόπο. Ακόμα λιγότερο ομοιογενής είναι η επικοινωνία εάν κάθε επεξεργαστής αποφασίζει χωριστά με ποίον άλλο επεξεργαστή θα πρέπει να επικοινωνίσει, ανάλογα με την εξέλιξη της επεξεργασίας των δεδομένων που διαθέτει. Στην περίπτωση αυτή απαιτείται ένα δίκτυο επικοινωνίας πολύ πιο ισχυρό και πιο γρήγορο το οποίο κατά συνέπεια δεν θα είχε τη δυνατότητα να είναι πλούσιο.

Σχήμα Επικοινωνίας (Communication Pattern.) το οποίο απαιτείται προκειμένου να ανταλλάξουν τις πληροφορίες τους οι επεξεργαστές ή να συλλεχθούν τα αποτελέσματα. Συχνά, τέτοια επικοινωνιακά σχήματα μπορούν να υλοποιηθούν με χρήση ορισμένων στοιχειωδών πράξεων (primitives), που είναι ενσωματωμένες (built in) στο υπολογιστικό σύστημα. Τέτοιες πράξεις μπορούν να χρησιμοποιηθούν για να υλοποιήσουν με πολύ αποτελεσματικό τρόπο μεταφορά πληροφοριών που διαφορετικά θα έκανε συμβατική χρήση του δικτύου και θα το επιβάρυνε. Είναι δυνατό ένας αλγόριθμος να χρησιμοποιεί εναλλάξ περισσότερα από ένα τέτοια σχήματα.

Ποσότητα ανταλασσόμενων δεδομένων Υπάρχουν διεργασίες που χρειάζονται συχνή ανταλλαγή πληροφοριών μεταξύ των επεξεργαστών ενώ σε άλλες η ανταλλαγή γίνεται μόνο σε συγκεκριμένες φάσεις της επεξεργασίας, όπως, για παράδειγμα, μετά το πέρας ενός τμήματος υπολογισμών ή στην αρχή κάποιου άλλου. Ακόμα, τα μηνύματα μπορεί να είναι μεγαλύτερα ή μικρότερα σε μέγεθος. Τα δύο παραπάνω ποσοτικά μεγέθη δηλαδή η *συχνότητα* και το *μέγεθος* παίζουν ιδιαίτερο ρόλο στην απόδοση μιας διεργασίας διότι επηρεάζονται από την ικανότητα μεταφοράς (bandwidth) του δικτύου. Από την άλλη μεριά κάθε μήνυμα προκειμένου να μεταδοθεί απαιτεί εν γένει χρόνο $T_m = t_{startup} + m * t_b$, δηλαδή είναι το άθροισμα ενός χρόνου εκκινήσεως $t_{startup}$ που είναι ανεξάρτητος του μεγέθους του μηνύματος και ενός άλλου $m * t_b$ που εξαρτάται από τα m bytes από τα οποία αποτελείται το μήνυμα για κάθε ένα από τα οποία χρειάζεται χρόνο t_b. Δηλαδή ο χρόνος επικοινωνίας εξαρτάται από το πλήθος των μηνυμάτων, από το πόσο μεγάλος είναι ο χρόνος $t_{startup}$ και από το μέγεθος του μηνύματος με το συντελεστή t_b.

Σειριακότητα επικοινωνίας Πολλές φορές τα μηνύματα σε μια διεργασία δεν μπορούν να σταλούν ή να ληφθούν παράλληλα. Κάποιο μήνυμα μπορεί να σταλεί αφού ληφθεί (ή

αφού σταλεί) κάποιο άλλο, ιδιαίτερα μάλιστα όταν πολλά μηνύματα έχουν κοινό αποδέκτη ή αποστολέα επεξεργαστή. Αυτό συνεπάγεται τη μείωση της παραλληλοποίησης του αλγόριθμου και επηρεάζει σημαντικά την απόδοσή του.

4.2.3 Χαρακτηριστικά Δεδομένων

Τα χαρακτηριστικά που θα περιγραφούν στη συνέχεια έχουν σχέση όχι μόνο με τα δεδομένα εισόδου, αλλά και με τον τρόπο που τα χειρίζεται το σύστημα, την εξέλιξη της μορφής και της ποσότητά τους κ.ά. έως ότου εξαχθούν τα τελικά δεδομένα εξόδου της διεργασίας.

Η Μονάδα των Δεδομένων. Μονάδα Δεδομένων είναι η ελάχιστη ποσότητα δεδομένων, στα οποία μπορεί να εκτελεστεί η διεργασία. Πολλές διεργασίες στην Ανάλυση Εικόνων εκτελούνται πάνω σε δομές που περιγράφουν τη θέση και την ένταση σημείων της εικόνας. Άλλες απαιτούν δεδομένα υψηλότερου επιπέδου, δηλαδή είναι πιο περιγραφικά. Για παράδειγμα είναι δυνατόν ορισμένες διεργασίες να εκτελούνται έχοντας σαν δεδομένα την περιγραφή αντικειμένων, ευθειών κτλ. Στην περίπτωση αυτή οι μονάδες δεδομένων που θα επεξεργαστούν είναι αριθμητικά λιγότερες ενώ η επεξεργασία τους πιο πολύπλοκη. Είναι χαρακτηριστικό πως η παραλληλία ως προς τα δεδομένα μπορεί να υλοποιηθεί μέχρι το βαθμό που κάθε επεξεργαστής αναλαμβάνει μία τέτοια μονάδα επεξεργασίας. Αν προχωρήσουμε σε παραλληλοποίηση μεγαλύτερου βαθμού τότε θα πρέπει να διασπαστεί ο κώδικας για κάθε μονάδα κάνοντας αντίτυπα των δεδομένων σε διαφορετικούς επεξεργαστές. Αυτό αλλάσει το χαρακτήρα της παράλληλης επεξεργασίας ενώ δημιουργεί πρόσθετο υπολογιστικό και επικοινωνιακό κόστος για να συλλεχθούν τα επιμέρους αποτελέσματα εκ των υστέρων για κάθε μονάδα επεξεργασίας.

Δομή Δεδομένων Εισόδου. Είναι σημαντικό χαρακτηριστικό για κάθε διεργασία οι απαιτήσεις που έχει για τη δομή των δεδομένων πάνω στην οποία θα εργαστεί. Υπάρχουν διεργασίες στην ταχύτητα εκτέλεσης των οποίων η δομή είναι σχεδόν αδιάφορη ενώ άλλες στις οποίες παίζει καθοριστικό ρόλο. Στην τελευταία περίπτωση η δομή περιέχει πολύ περισσότερες πληροφορίες, με την έννοια ότι χρειάζεται όχι μόνο να αναπαριστά ποσοτικά τις μονάδες δεδομένων αλλά επιπλέον και σχέσεις που υπάρχουν μεταξύ τους. Συμπεριλαμβάνει δηλαδή και άλλες πληροφορίες όπως για παράδειγμα την τοπολογική τους διάταξη. Ελλειψη της

συγκεκριμένης δομής έχει σαν συνέπεια πολλές φορές τη δραματική αύξηση της πολυπλοκότητας της διεργασίας προκειμένου να εξαχθούν (εάν βέβαια είναι δυνατό) οι πληροφορίες που χρειάζονται. Επίσης υπάρχουν δομές *στατικές* όπως για παράδειγμα οι δομές δεδομένων εξόδου που δημιουργούνται από διεργασίες όπως τον μετασχηματισμό Hough, τον υπολογισμό του ιστογράμματος των εντάσεων, ανύσματα (vectors) χαρακτηριστικών του περιγράμματος (shape) αντικειμένων κτλ. όπως επίσης υπάρχουν και οι *δυναμικές* δομές δεδομένων όπως τα δέντρα, τα *τετραπλα δέντρα (quadtrees)*, τις *δυναμικές λίστες (linked lists)* κ.ά.

Εξέλιξη της Δομής των Δεδομένων Κατά τη διάρκεια της επεξεργασίας, πολύ συχνά εμφανίζεται το φαινόμενο οι αρχικές δομές να τροποποιούνται τόσο ως προς την μορφή όσο και ως προς την ποσότητα και τη θέση τους. Το γεγονός αυτό είναι ιδιαίτερα σημαντικό για την αποδοτικότερη υλοποίησή τους καθώς θα πρέπει να αντιμετωπιστούν προβλήματα που σχετίζονται με την ανακατανομή των δεδομένων πάνω στους επεξεργαστές.

Τοπικότητα των Δεδομένων χαρακτηρίζεται το γεγονός κατά το οποίο η θέση στην οποία βρίσκεται κάθε μονάδα δεδομένων πρέπει να είναι γνωστή, όχι μόνο στον επεξεργαστή που τη διαθέτει, αλλά και σε όλους τους άλλους επεξεργαστές.

4.3 Γνωρίσματα της Αρχιτεκτονικής

Τα κυριότερα γνωρίσματα από τα οποία χαρακτηρίζεται η αρχιτεκτονική ενός παράλληλου συστήματος είναι:

Το πλήθος των επεξεργαστών. Ενα βασικό γνώρισμα μίας αρχιτεκτονικής είναι το πλήθος των επεξεργαστών που διαθέτει και το οποίο την χαρακτηρίζει σαν αρχιτεκτονική υψηλής ή χαμηλής διαμέρισης (fine or coarse granularity).

Τεχνικά χαρακτηριστικά των επεξεργαστών. Η υπολογιστική ικανότητα, η χωρητικότητα της μνήμης, οι θύρες επικοινωνίας (fan-in, fan-out), και τα άλλα τεχνικά χαρακτηριστικά των επεξεργαστών έχουν ιδιαίτερη σημασία καθώς θα πρέπει να ικανοποιούν τις αντίστοιχες απαιτήσεις της διεργασίας.

Μνήμη. Εκτός της ποσότητας της μνήμης που αναφέρθηκε, υπάρχουν και άλλα χαρακτηριστικά που σχετίζονται με τη μνήμη με ιδιαίτερα σημαντικό ρόλο. Για παράδειγμα, αν είναι *κοινή* (shared) ή *κατανεμημένη* (distributed) και αν μπορεί να διαβάζει και να γράφει ένας μόνο κάθε φορά ή πολλοί επεξεργαστές συγχρόνως (CRCW,CREW,EREW - Concurrent Read/Write, Exclusive Read/Write).

Η Τοπολογία. Ο τρόπος με τον οποίο είναι συνδεδεμένοι οι επεξεργαστές μεταξύ τους παίζει ιδιαίτερο ρόλο στην εξυπηρέτηση της απαιτούμενης ενδοεπικοινωνίας. Ακόμα, είναι σημαντικό το πόσο εύκολα προσομειώνει η τοπολογία αυτή άλλες τοπολογίες.

Το δίκτυο επικοινωνίας. Αμεση σχέση με το προηγούμενο γνώρισμα έχει το δίκτυο επικοινωνίας που χρησιμοποιείται στη σύνδεση των επεξεργαστών. Οι παράμετροι της ταχύτητάς του και η εξάρτησή της από το μέγεθος του μηνύματος, αλλά και από τη φυσική απόσταση μεταξύ αποστολέα-επεξεργαστή και παραλήπτη επεξεργαστή, ο πλούτος του σε διασυνδέσεις κύριες και αναπληρωματικές, η αξιοπιστία του (reliability) ακόμα και σε περίπτωση αποτυχίας ορισμένων συνδέσεων (link failure), τεχνικές για μείωση του επικοινωνιακού φόρτου κ.ά. παίζουν ιδιαίτερο ρόλο στην απόδοση του συστήματος σε ορισμένες περιπτώσεις.

Η μορφή MIMD ή SIMD κτλ. Ενας βασικός διαχωρισμός των αρχιτεκτονικών είναι εκείνος που αναφέρεται στο κατά πόσο οι επεξεργαστές τους εκτελούν τις ίδιες (SIMD - Single Instruction stream Multiple Data stream) ή διαφορετικές εντολές στα διαφορετικά δεδομένα που κρατούν (MIMD - Multiple Instruction stream Multiple Data stream).

Διαθέσιμες γλώσσες και περιβάλλον προγραμματισμού. Τα λειτουργικά συστήματα και οι γλώσσες προγραμματισμού που υποστηρίζουν την κάθε αρχιτεκτονική μπορεί να παίξουν καθοριστικό ρόλο στην επίδοση της υλοποίησης ενός αλγορίθμου.

4.4 Το Περιεχόμενο της Εικόνας

Η επόμενη συνιστωσα του προβλήματος, όπως αναφέρθηκε, είναι το περιεχόμενο (content)

μιας εικόνας που αποκτά διαφορετική έννοια ανάλογα με τη διεργασία που πρόκειται να εκτελέσουμε. Κάποια χαρακτηριστικά που συνθέτουν το περιεχόμενο της εικόνας είναι σημαντικά για ορισμένες διεργασίες και αδιάφορα σε άλλες. Για καθένα από τα χαρακτηριστικά αυτά, τρία μεγέθη μας ενδιαφέρουν:

Ποια είναι **τα χαρακτηριστικά** του περιεχόμενου της εικόνας που θα πάρουν μέρος στη διεργασία που θα ακολουθήσει, καθώς και **ο τρόπος παράστασής τους**.

Το πλήθος της ενδιαφέρουσας πληροφορίας, όπως αυτή παριστάνεται από κάποια ομάδα χαρακτηριστικών. Το πλήθος μας ενδιαφέρει γιατί επηρεάζει τόσο το μέγεθος της απαιτούμενης για την αποθήκευσή της μνήμης, όσο και την υπολογιστική ισχύ που χρειάζεται για την επεξεργασία τους.

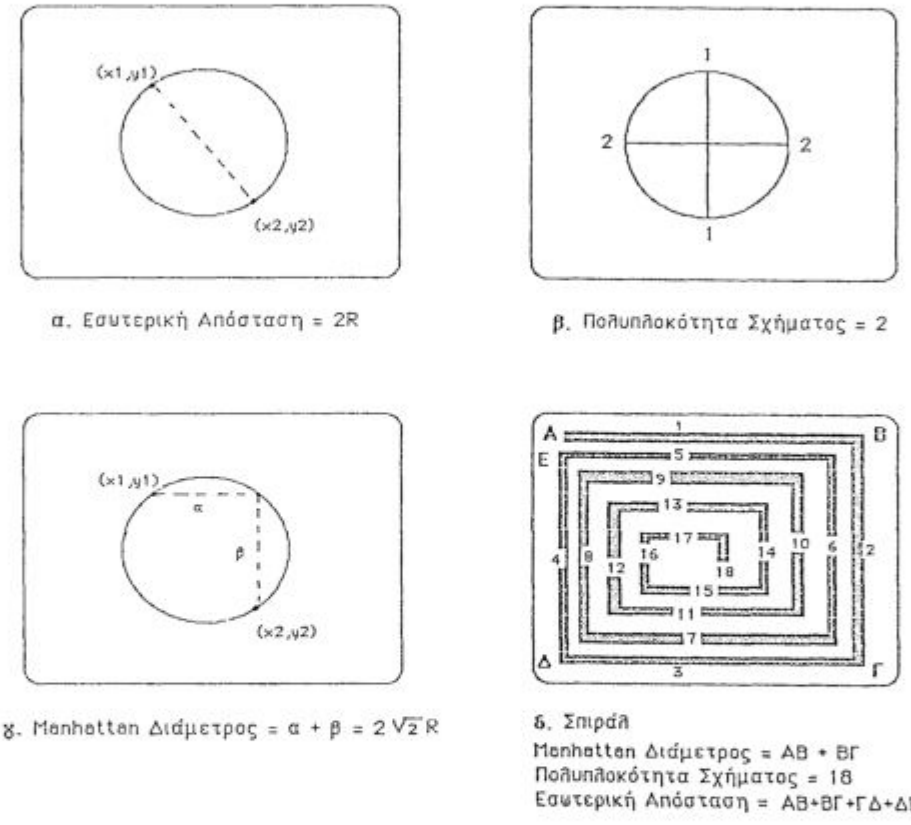
Η χωρική κατανομή (spatial distribution). Δηλαδή η περιγραφή της θέσης που κατέχουν αυτά πάνω στη συνολική δομή της εικόνας. Το χαρακτηριστικό αυτό παίζει ιδιαίτερη σημασία για την εκ των προτέρων ισοκατανομή του φορτίου των επεξεργαστών.

Τα χαρακτηριστικά του περιεχομένου της εικόνας, των οποίων την επίδραση στους αλγορίθμους μελετήσαμε, είναι: Η μέγιστη **Διάμετρος Manhattan** (ΔΜ) των αντικειμένων, που εμφανίζονται στην εικόνα, όπου Διάμετρος Manhattan = max ($| x_i - x_j | + | y_i - y_j |$) όπου (x_i, y_i) και (x_j, y_j) σημεία πάνω στο ίδιο αντικείμενο της εικόνας [166]. Μεταξύ των σχημάτων, μέγιστη ΔΜ (=2Ν) έχει ένα τετράγωνο ίσο με την εικόνα και ελάχιστη ένα σημείο (ΔΜ=0).

Απόσταση μεταξύ δύο σημείων στο ίδιο αντικείμενο ορίζεται η ελάχιστη διαδρομή μέσα στο αντικείμενο, που ενώνει τα σημεία αυτά. Τα σημεία των διαδρομών αυτών πρέπει να ανήκουν σε περιοχή αντικειμένου. Κάθε αντικείμενο στην εικόνα χαρακτηρίζεται από τη μέγιστη από τις αποστάσεις μεταξύ οποιωνδήποτε δύο σημείων που ανήκουν στο αντικείμενο αυτό. Χαρακτηριστικό της εικόνας είναι η μέγιστη από τις αποστάσεις αυτές για όλα τα αντικείμενα που παριστάνονται στην εικόνα. Το χαρακτηριστικό αυτό ονομάζουμε **μέγιστη εσωτερική απόσταση** σημείων. Το σχήμα με τη μεγαλύτερη τέτοια απόσταση για συγκεκριμένο εμβαδόν είναι η σπείρα και ελάχιστη το σημείο (=0). Για ένα παραλληλόγραμμο είναι η διαγώνιος ενώ για τον κύκλο είναι η διάμετρός του.

Ενα τρίτο χαρακτηριστικό είναι ο μέγιστος αριθμός διαδοχικών οριζόντιων και κατακό-

ρυφων σαρώσεων στο εσωτερικό του αντικειμένου που χρειάζονται για να διαδοθεί η ετικέτα με την ελάχιστη τιμή σε όλα τα σημεία του αντικειμένου. Ο αριθμός αυτός ονομάζεται **πολυπλοκότητα (C) του σχήματος** (Shape complexity). Μέγιστη πολυπλοκότητα έχει η σπείρα (Spiral) στραμμένη κατά γωνία 45 μοιρών, ενώ ελάχιστη (=2) το παραλληλόγραμμο και η έλλειψη, ανεξαρτήτα από το μέγεθος. Στην περίπτωση σημείου έχουμε επίσης πολυπλοκότητα σχήματος 0. Τα χαρακτηριστικά αυτά για ένα κύκλο αλλά και για τη σπείρα φαίνονται στο σχήμα 4.1.



α. Εσωτερική Απόσταση = 2R

β. Πολυπλοκότητα Σχήματος = 2

γ. Manhattan Διάμετρος = $\alpha + \beta = 2\sqrt{2}R$

δ. Σπιράλ
Manhattan Διάμετρος = ΑΒ + ΒΓ
Πολυπλοκότητα Σχήματος = 18
Εσωτερική Απόσταση = ΑΒ+ΒΓ+ΓΔ+ΔΕ+ ..

Σχήμα 4.1: Μερικά χαρακτηριστικά του περιεχομένου μιας εικόνας

Αλλα χαρακτηριστικά είναι το **Εμβαδόν** και η **Μέγιστη Διάμετρος** τόσο του αντικειμένου όσο και του Κυρτού του Περιγράμματος, **ο λόγος της μεγαλύτερης προς τη μικρότερη διάμετρο** (enlongation) κ.ά.

4.5 Γνωρίσματα Αλγορίθμων Ισοκατανομής Φορτίου

Υπάρχουν πολλοί αλγόριθμοι ισοκατανομής φορτίου (ΙΦ) με διάφορα χαρακτηριστικά που αφορούν στις υπολογιστικές απαιτήσεις τους, την ενδοεπικοινωνία, το αν είναι πιθανοκρατικοί (probabilistic) ή αιτιοκρατικοί (deterministic), αν είναι τοπικοί ή καθολικοί. Πολλοί αλγόριθμοι είναι απλοί αλλά δεν είναι βέλτιστοι ενώ αντίθετα οι βέλτιστοι παρουσιάζουν γενικά πολυπλοκότητα στην υλοποίηση, που τους καθιστά ασύμφορους. Στη συνέχεια δίνονται ορισμένα από τα χαρακτηριστικά αυτά, που δεσπόζουν στους αλγόριθμους ισοκατανομής. Γενικά, σε έναν αλγόριθμο ισοκατανομής φορτίου, ο στόχος της καθολικά (global) βέλτιστης ανακατανομής μπορεί να επιτευχθεί με συλλογή πληροφορίας για το φορτίο όλων των επεξεργαστών. Ως εκ τούτου ένα γρήγορο δίκτυο ή ειδικές δυνατότητες της αρχιτεκτονικής για γρήγορη συγκέντρωση και αναδιανομή πληροφορίας για το υπολογιστικό φορτίο όλων των επεξεργαστών δίνει ιδιαίτερα πλεονεκτήματα στον αλγόριθμο που εκμεταλλεύεται τέτοια χαρακτηριστικά της αρχιτεκτονικής. Τα δεσπόζοντα χαρακτηριστικά των αλγορίθμων ισοκατανομής φορτίου είναι:

Υπολογιστικές απαιτήσεις. Ο χρόνος που αφιερώνεται στην εκτέλεση του αλγορίθμου ΙΦ επιβαρύνει πρόσθετα την εκτέλεση της διεργασίας. Ως εκ τούτου ο χρόνος που αφιερώνεται σε υπολογισμούς έως ότου οι επεξεργαστές αποφασίσουν αλλά και ολοκληρώσουν την ανακατανομή θα πρέπει να είναι ο ελάχιστος δυνατός έτσι ώστε να μην επιβαρύνεται η επεξεργασία. Αντίθετα η ΙΦ πρέπει συνολικά να επιταχύνει τη διεργασία.

Επικοινωνία. Τα επικοινωνιακά χαρακτηριστικά των αλγορίθμων ΙΦ είναι όμοια με εκείνα των διεργασιών ανάλυσης εικόνων που αναφέρθηκαν στην παράγραφο 4.2.2. Δηλαδή, τα επικοινωνιακά σχήματα που χρησιμοποιεί, η συχνότητα, η τοπικότητα και το μέγεθος των μηνυμάτων, όπως και η ομοιογένεια που τη χραρακτηρίζει. Αυτό είναι ιδιαίτερα σημαντικό στην περίπτωση των αλγορίθμων ΙΦ, όπου πρέπει να συλλεχθεί πληροφορία για το φόρτο όλων των επεξεργαστών προκειμένου να επιτευχθεί μια βέλτιστη λύση, αλλά στη συνέχεια και να ανακατανεμηθεί το φορτίο.

Τοπολογία. Συχνά οι αλγόριθμοι ΙΦ είναι σχεδιασμένοι έτσι ώστε να εργάζονται στηριζόμενοι σε ορισμένα χαρακτηριστικά της τοπολογίας του παράλληλου υπολογιστικού συστήματος. Έτσι, η υλοποίησή τους σε άλλες τοπολογίες μπορεί να τις καθιστά αναποτελεσματικές, ενώ συχνά είναι αδύνατη.

Διατήρηση της Τοπικότητας στα Δεδομένα Πολλοί αλγόριθμοι ΙΦ με την ανακατανομή των δεδομένων δημιουργούν μια διάσπαση του φορτίου και καταστροφή της τοπικότητας των δεδομένων. Δεδομένα που πριν την ανακατανομή βρίσκονταν σε γειτονικούς επεξεργαστές μπορεί να βρεθούν σε οποιονδήποτε τυχαίο επεξεργαστή μετά από αυτή. Αν όμως για την εκτέλεση της διεργασίας χρειάζεται να είναι γνωστή η θέση των μονάδων δεδομένων και σε άλλους επεξεργαστές από εκείνους που τις κρατούν, δημιουργείται σημαντική επιβάρυνση στην απόδοση.

Εξαρτήσεις από το περιεχόμενο της εικόνας Περιοχές της εικόνας με περισσότερη πληροφορία, είναι λογικό να χρειάζονται περισότερη επεξεργασία και συνεπώς επιβαρύνουν ιδιαίτερα τους επεξεργαστές που τις κρατούν. Συνεπώς η ανακατανομή του φορτίου αφορά ιδιαίτερα αυτούς τους επεξεργαστές. Τόσο η δομή όσο και η ποσότητα των δεδομένων που παριστάνουν το περιεχόμενο της εικόνας επιβαρύνουν ανάλογα την απόδοση του αλγορίθμου ΙΦ.

Χρήση "καλωδιωμένης λογικής" Υπάρχουν αλγόριθμοι ισοκατανομής του φορτίου που είναι σχεδιασμένοι να εκμεταλλεύονται ικανότητες καλωδιωμένης λογικής που διαθέτει το σύστημα για να επιτύχουν γρήγορη συλλογή πληροφοριών και γρήγορη στη συνέχεια αναδιανομή του φορτίου.

Κεφάλαιο 5

Θέματα σχετικά με την Επιλογή των Παράλληλων Υλοποιήσεων

5.1 Γενικά

Στο κεφάλαιο αυτό θα παρουσιάσουμε θέματα που σχετίζονται με την επιλογή της καλύτερης μεταξύ πολλών εναλλακτικών παράλληλων υλοποιήσεων μιας διεργασίας Ανάλυσης Εικόνων. Μιά τέτοια επιλογή στηρίζεται στην παραμετροποιημένη ανάλυση όλων των παραγόντων που εμπλέκονται σε αυτή, δηλαδή των αλγορίθμων των διεργασιών που θέλουμε να εφαρμόσουμε, των αρχιτεκτονικών που έχουμε στη διάθεσή μας, του περιεχομένου των εικόνων και, τέλος, των αλγορίθμων για ανακατανομή του φορτίου. Τα προβλήματα στα οποία αναφερόμαστε είναι εκείνα τα οποία μπορούν να υλοποιηθούν παράλληλα ως προς τα δεδομένα σε SIMD μορφή. Ανάλογη μελέτη μπορεί να επεκτείνει την προσέγγισή μας για αρχιτεκτονικές MIMD.

Οι μέθοδοι που έχουν αναπτυχθεί μέχρι τώρα για το πρόβλημα αυτό στηρίζονται στο γεγονός ότι κάθε διεργασία και κάθε αρχιτεκτονική μπορεί να αναπαρασταθεί από ένα γράφο. Οι κόμβοι του γράφου της διεργασίας παριστάνουν τα υπολογιστικά τμήματα που την αποτελούν

και οι ακμές του την επικοινωνία μεταξύ αυτών των τμημάτων. Σε κάθε ακμή του γράφου μπορούμε να θέσουμε συντελεστές βάρους που αντιπροσωπεύουν το μέγεθος των μηνυμάτων που πρέπει να ανταλλαγούν μεταξύ των συγκεκριμένων υπολογιστικών τμημάτων καθώς επίσης τη συχνότητα ανταλλαγής των μηνυμάτων αυτών. Κάθε υπολογιστικό τμήμα περιγράφεται από μία έκφραση της υπολογιστικής ισχύος που απαιτεί. Η έκφραση αυτή είναι συνάρτηση διαφόρων παραγόντων, μεταξύ των οποίων και τα χαρακτηριστικά του περιεχομένου της εικόνας. Αντίστοιχα, κάθε αρχιτεκτονική μπορεί επίσης να παρασταθεί από έναν άλλο γράφο, οι κόμβοι του οποίου αντιπροσωπεύουν τους επεξεργαστές της αρχιτεκτονικής, ενώ οι ακμές αντιπροσωπεύουν τις φυσικές συνδέσεις επικοινωνίας μεταξύ τους, δηλαδή την τοπολογία της αρχιτεκτονικής. Οι γράφοι των αρχιτεκτονικών συχνά περιγράφονται και μελετώνται με βάση τη θεωρία των Γράφων [167]. Στην περίπτωση των αρχιτεκτονικών SIMD κόμβος στο γράφο της διεργασίας είναι κάθε μονάδα δεδομένων. Έτσι, σε εφαρμογές ανάλυσης εικόνων οι κόμβοι μπορεί να είναι πάρα πολλοί και ο σχηματιζόμενος γράφος της διεργασίας να είναι εξαιρετικά πολύπλοκος.

Η αποδοτική (efficient) παράλληλη υλοποίηση μιας συγκεκριμένης διεργασίας πάνω σε μια παράλληλη αρχιτεκτονική αντιστοιχεί λοιπόν στην κατάλληλη εναπόθεση (mapping) των δύο γράφων. Η εναπόθεση αυτή πρέπει να γίνει με τον καλύτερο δυνατό τρόπο προκειμένου να επιτευχθεί η μεγαλύτερη δυνατή απόδοση. Το πρόβλημα ανάγεται σε πρόβλημα ισομορφισμού γράφων, το οποίο είναι γενικά NP-complete. Διάφορες εργασίες έχουν δημοσιευτεί για το πρόβλημα αυτό, στις οποίες γίνονται ποικίλες θεωρήσεις για το μοντέλο της αρχιτεκτονικής. Μερικές από αυτές είναι γραφοθεωρητικές [168, 169] και οι αλγόριθμοι που προτείνουν είναι αρκετά πολύπλοκοι. Άλλες είναι ευρηματικές (heuristic), οι οποίες, εκτός από την απλότητά τους, φαίνονται να έχουν καλύτερη απόδοση [159] και αναφέρονται σε συγκεκριμένες αρχιτεκτονικές και τοπολογίες, ή ακόμα σε γράφους διεργασιών με συγκεκριμένα χαρακτηριστικά [170, 161, 148].

Αυτή η προσέγγιση του προβλήματος όμως, ειδικά σε γράφους με μεγάλο αριθμό κόμβων, είναι ιδιαίτερα πολύπλοκη και χρονοβόρα. Αν μάλιστα υπάρχει ομοιογένεια στις υπολογιστικές και στις επικοινωνιακές απαιτήσεις μεταξύ των υπολογιστικών τμημάτων, τότε η επίλυσή του με τη μέθοδο αυτή αποκτά μικρότερη σημασία διότι δεν εκμεταλλεύεται τη δυνατότητα να περιγραφούν τα επικοινωνιακά σχήματα με γενικούς όρους, ενώ η ομοιότητα των κόμβων καθιστά περιττή την εξουθενωτική (exhaustive) σύγκριση των πιθανών εναποθέσεων. Μια βασική

παρατήρηση που θα πρέπει να κάνουμε ακόμα για τις μεθόδους εναπόθεσης γράφων είναι ότι προϋποθέτουν τη δυνατότητα τμηματοποίησης του αλγορίθμου με τρόπο τέτοιο που ο τελικός γράφος που θα τον αναπαριστά να είναι πρακτικά χρησιμοποιήσιμος. Για παράδειγμα, στην [170] αναφέρεται ότι για ένα γράφο με 27 ακμές η εύρεση της βέλτιστης εναπόθεσης πάνω σε συγκεκριμένη αρχιτεκτονική απαιτεί 1.18 δευτερόλεπτα, για 77 ακμές 4.1 και για 230 ακμές 21.1 δευτερόλεπτα. Δεδομένου όμως ότι σε πολλές εφαρμογές η εκτέλεση μιας διαδικασίας μπορεί να διαρκεί μόνο μερικές εκατοντάδες msec, καταλαβαίνει κανείς ότι οι χρόνοι που προαναφέρθηκαν για τη βέλτιστη εναπόθεση των γράφων είναι πολύ μεγάλοι και μόνο για πολύ χρονοβόρες διαδικασίες θα μπορούσαν να χαρακτηριστούν ανεκτοί. Γίνεται λοιπόν φανερό πως θα πρέπει να αναπτυχθεί μια άλλη, πιο απλοποιημένη μέθοδος, που θα μπορεί να επιλύει το πρόβλημα αυτό όταν η υλοποίηση της ίδιας της διεργασίας απαιτεί πολύ λιγότερο χρόνο ενώ τα τμήματα στα οποία μπορεί να χωριστεί είναι πάρα πολλά. Επιπλέον, η μέθοδος αυτή θα πρέπει να λαμβάνει υπόψη και άλλους παράγοντες, που επηρεάζουν την απόδοση μιας παράλληλης διεργασίας, όπως είναι το περιεχόμενο της εικόνας και οι δομές δεδομένων που την παριστάνουν, δεν θα πρέπει να περιορίζεται σε αρχιτεκτονικές MIMD, ενώ θα πρέπει να χρησιμοποιεί εναλλακτικές λύσεις διαθέσιμων αρχιτεκτονικών και αλγορίθμων.

Στην εργασία αυτή προτείνουμε ένα μοντέλο επιλογής της βέλτιστης μεταξύ πολλών εναλλακτικών παράλληλων υλοποιήσεων για εφαρμογές ανάλυσης εικόνων, το οποίο δίνει ικανοποιητικά αποτελέσματα σε ένα ευρύ φάσμα διεργασιών μέσου επιπέδου. Εκτός από τη συσχέτιση των γράφων αρχιτεκτονικών και αλγορίθμων, λαμβάνει υπόψη του και πολλές άλλες παραμέτρους που επηρεάζουν την απόδοση μιας παράλληλης υλοποίησης.

Στο επόμενο τμήμα δίνουμε τη γενική περιγραφή για ένα μοντέλο επιλογής παράλληλων υλοποιήσεων και τη μέθοδο που στηρίζεται στο μοντέλο αυτό. Στο επόμενο κεφάλαιο θα ακολουθήσουν πειραματικά αποτελέσματα με συγκεκριμένες διεργασίες ανάλυσης εικόνων, τα οποία δείχνουν ότι το περιγραφόμενο μοντέλο είναι δυνατόν να κάνει πρόβλεψη για την καλύτερη παράλληλη υλοποίηση των διεργασιών αυτών, υπό ορισμένες συνθήκες.

5.2 Γενική περιγραφή της μέθοδου Επιλογής Παράλληλων Υλοποιήσεων

Από όσα αναφέρθηκαν μέχρι τώρα έχει γίνει φανερό πως σε ένα ολοκληρωμένο σύστημα Παράλληλης Επεξεργασίας Εικόνων εμπλέκεται μια πλειάδα από παραμέτρους που επηρεάζουν σημαντικά την απόδοσή του. Στα προηγούμενα ταξινομήσαμε τις παραμέτρους αυτές σαν γνωρίσματα των τεσσάρων βασικών συνιστωσών, που συνθέτουν το πρόβλημα της παράλληλης ανάλυσης εικόνων.

Ο χρήστης ενός παράλληλου συστήματος θα πρέπει να κάνει τις κατάλληλες επιλογές για τις τιμές των γνωρισμάτων αυτών προκειμένου να βελτιστοποιήσει τη συνολική απόδοση. Έτσι, είναι ιδιαίτερα χρήσιμη μια μεθοδολογία, η οποία, δοθέντων των τιμών ορισμένων γνωρισμάτων όσων συνιστωσών του προβλήματος θεωρούνται συγκεκριμενοποιημένες στη δεδομένη στιγμή, να μπορεί να κάνει μία εισήγηση για τις τιμές των υπολοίπων γνωρισμάτων, που θα εξασφάλιζαν τη βέλτιστη συνολική απόδοση σύμφωνα με τη συσσωρευμένη εμπειρία. Η προσέγγισή μας στο πρόβλημα υπερέχει της μεθόδου εναπόθεσης των γράφων, ιδιαίτερα στην περίπτωση που η διαδικασία παρουσιάζει ομοιογένεια, και η αρχιτεκτονική είναι τύπου SIMD. Μια σχετική εργασία για αρχιτεκτονικές MIMD είναι η [171]. Εκεί παρουσιάζονται απόψεις για τη δημιουργία μιας Βάσης Δεδομένων Αλγορίθμων, που θα υποστήριζε ένα έξυπνο Λειτουργικό Σύστημα για Ανάλυση Εικόνων σε MIMD μηχανές. Στην εργασία αυτή όμως δεν λαμβάνονται υπόψη τα επικοινωνιακά σχήματα, που απαιτεί μια διεργασία, αλλά ούτε και το περιεχόμενο της εικόνας.

Το πρόβλημα λοιπόν της παράλληλης υλοποίησης διεργασιών Ανάλυσης Εικόνων είναι πολυπαραμετρικό και ως εκ τούτου ένα μοντέλο επιλογής υπολοιήσεων είναι εξαιρετικά πολύπλοκο. Η ιδέα εδώ είναι να το αντιμετωπίσουμε παίρνοντας αποφάσεις σε πολλαπλά επίπεδα, στα οποία θα χρησιμοποιούνται διάφοροι κανόνες απόφασης (decision rules), διαμορφώνοντας έτσι ένα *Δένδρο Αποφάσεων* (*Decision Tree*). Το δέντρο αυτό θα αποτελεί το μηχανισμό κεντρικού ελέγχου του μοντέλου επιλογής παράλληλων υπολοιήσεων διεργασιών ανάλυσης εικόνων, το οποίο φαίνεται στο σχήμα 5.1. Το μοντέλο αυτό αποτελείται από ένα πρώτο "τμήμα εισόδου δεδομένων" από όπου δίνονται από το χρήστη όλες οι πληροφορίες που είναι εξαρχής γνωστές για το πρόβλημα. Αυτές οι πληροφορίες αναφέρονται στην εικόνα πάνω στην οποία πρόκειται

να γίνει η επεξεργασία, στη διεργασία που θα υλοποιηθεί και στην αρχιτεκτονική ή αρχιτεκτονικές που έχει στη διάθεσή του ο χρήστης. Το "κύριο μέρος" του συστήματος αποτελείται από τρία τμήματα. Το πρώτο είναι εκείνο όπου είναι αποθηκευμένες στατιστικές πληροφορίες για τα χαρακτηριστικά κάθε πιθανού είδους εικόνας της οποίας μπορεί να ζητηθεί η επεξεργασία. Το δεύτερο τμήμα περιέχει την περιγραφή των επικοινωνιακών και υπολογιστικών χαρακτηριστικών όσων αρχιτεκτονικών είναι διαθέσιμες. Τέλος, υπάρχει το τμήμα εκείνο στο οποίο βρίσκονται οι περιγραφές των γνωστών στο σύστημα αλγορίθμων για όσες διεργασίες είναι υποψήφιες για υλοποίηση. Ο τρόπος αναπαράστασης των πληροφοριών σε κάθε τμήμα δίνεται στις επόμενες παραγράφους.

Τα τμήματα αυτά του μοντέλου είναι αρκετά για να δώσουν μια εισήγηση για τη βέλτιστη υλοποίηση σύμφωνα με τις διαθέσιμες πληροφορίες. Εκτός από αυτά όμως υπάρχει και ένα ακόμα τμήμα, αυτό της Ισοκατανομής Φορτίου. Μια πρώτη προσέγγιση για την ανάγκη ή όχι ανακατανομής του φορτίου μπορεί να γίνει εκ των προτέρων, με βάση τη διεργασία, το περιεχόμενο της εικόνας, ή ακόμα με βάση την ήδη επιλεγμένη υλοποίηση. Ο αλγόριθμος Ισοκατανομής Φορτίου θα ενεργοποιείται στη συνέχεια δυναμικά κατά τη διάρκεια της επεξεργασίας, προκειμένου να παίρνει αποφάσεις για την ανάγκη και τη σκοπιμότητα ανακατανομής του υπολογιστικού φορτίου των επεξεργαστών κάθε στιγμή. Στη διατριβή αυτή δεν θα ασχοληθούμε εκτεταμένα με το τελευταίο αυτό τμήμα του μοντέλου. Για να δούμε πώς θα λειτουργεί ο κεντρικός μηχανισμός ελέγχου της λειτουργίας του μοντέλου, δηλαδή το δέντρο αποφάσεων, ας φανταστούμε πως θέλουμε να δημιουργήσουμε ένα έμπειρο σύστημα (expert system) ή ένα διαλογικό (interactive) σύστημα λήψης αποφάσεων. Τα ζητήματα που προκύπτουν στο σχεδιασμό ενός τέτοιου συστήματος είναι τα εξής:

-Ποιες ερωτήσεις και με ποια σειρά πρέπει να γίνονται στο χρήστη προκειμένου να καταλήξουμε σε μια απόφαση;

-Πόση πληροφορία πρέπει ο χρήστης να δώσει στο σύστημα πριν εκείνο μπορέσει να του δώσει ενδιαφέρουσες πληροφορίες;

Ακόμα, το δέντρο αυτό θα πρέπει να έχει τα εξής χαρακτηριστικά:

-Οι κανόνες σε ένα επίπεδο δεν θα πρέπει να αναιρούν ή να έρχονται σε αντίθεση με κανόνες αποφάσεων στα επόμενα επίπεδα.
-Σε περίπτωση αόριστων πληροφοριών εισόδου θα πρέπει να δίνει πιθανοκρατικές (probabilistic) εισηγήσεις και θα πρέπει να έχει δυνατότητα αναπροσαρμογής.

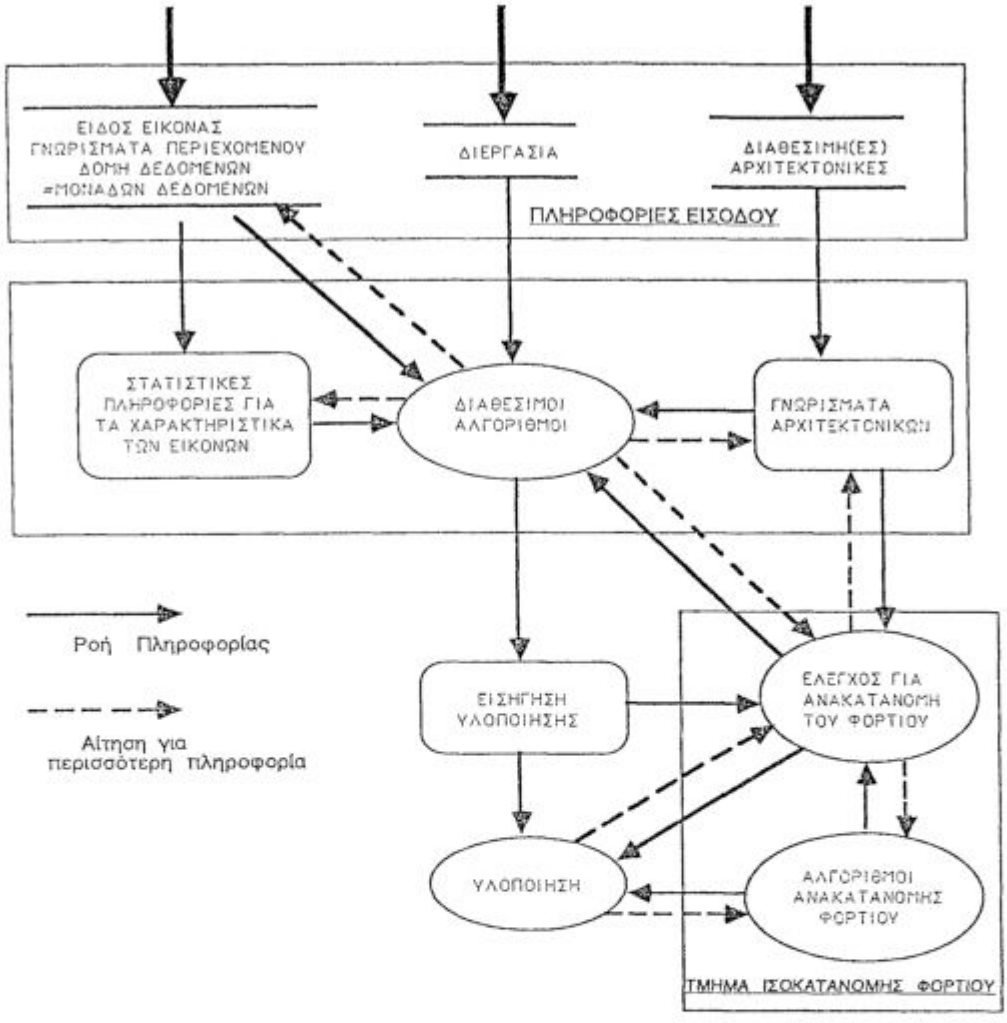
Προκειμένου να γίνει λειτουργική η μέθοδός μας θα πρέπει να απλοποιηθούν (και να επιμεριστούν) οι πολύπλοκοι υπολογισμοί που απαιτεί το αναλυτικό μοντέλο λήψης αποφάσεων, προκειμένου να βγάλει την απόφασή του. Με άλλα λόγια κάνουμε μια απλοποιημένη προσέγγιση (approximation) του αναλυτικού αυτού μοντέλου. Με την απλοποίηση του μοντέλου, ουσιαστικά περνούμε τις αρχικά διαθέσιμες πληροφορίες από ένα φίλτρο, προκειμένου να συρρικνώσουμε το πεδίο επιλογών και να εστιάσουμε την αναζήτηση της καλύτερης υλοποίησης σε μια υποπεριοχή του πεδίου των λύσεων.

Η ανάπτυξη του μοντέλου που προτείνουμε επικεντρώνει το ενδιαφέρον της σε δύο σημεία, τα οποία γίνονται και αντικείμενα ανάλυσης στις επόμενες παραγράφους. Το πρώτο είναι το πώς αναλύουμε και περιγράφουμε τους αλγορίθμους προκειμένου να τους εντάξουμε σε ένα τέτοιο δέντρο αποφάσεων. Στην περιγραφή αυτή συμπεριλαμβάνεται και η συσχέτιση των αλγορίθμων με τις άλλες συνιστώσες του προβλήματος. Το δεύτερο σημείο ενδιαφέροντος είναι το πώς σχεδιάζουμε τη λειτουργία του δένδρου αποφάσεων, ώστε, δοθείσης μια διεργασίας, να επιτυγχάνουμε μια αποτελεσματική πλοήγηση πάνω σ'αυτό, που θα δώσει, σε ικανοποιητικό χρόνο, μια απόφαση για τον τρόπο υλοποίησης της διεργασίας.

Στις επόμενες παραγράφους, δίνουμε τη δομή των επιμέρους τμημάτων, με έμφαση στην αναπαράσταση των διαθέσιμων αλγορίθμων και παρουσιάζουμε τη λειτουργία του δέντρου αποφάσεων. Με τα πειραματικά αποτελέσματα που ακολουθούν επιβεβαιώνουμε τις θεωρήσεις μας όσον αφορά στην αλληλεξάρτηση των επιμέρους παραμέτρων και την ανάγκη ανάλυσής τους προκειμένου να επιλέξουμε την υλοποίηση με τη βέλτιστη κάθε φορά απόδοση.

5.2.1 Περιγραφή των αλγορίθμων

Ο χρόνος που απαιτεί μία διαδικασία Ανάλυσης Εικόνων στην παράλληλη υλοποίησή της συμπεριλαμβάνει το χρόνο που χρειάζεται για υπολογισμούς και εκείνον που απαιτεί για



Σχήμα 5.1: Μοντέλο του Συστήματος Επιλογής Παράλληλων Υλοποιήσεων Διεργασιών Ανάλυσης Εικόνων

επικοινωνία. Στη συνέχεια θα αναλύσουμε τους χρόνους αυτούς αρχίζοντας από το χρόνο για την επικοινωνία.

Ο χρόνος που απαιτείται για τη μετάδοση ενός μηνύματος εξαρτάται γενικά από το χρόνο t_{st}, που απαιτείται κατά την εκκίνηση της επικοινωνίας (*start-up time*) και το χρόνο t_b, που είναι συνάρτηση του πλήθους b των bits που μεταφέρονται και του επικοινωνιακού σχήματος (communication pattern) που πρέπει να ακολουθηθεί μεταξύ των επεξεργαστών που επικοινωνούν. Έτσι ο χρόνος επικοινωνίας παριστάνεται από την εξίσωση:

$$\mathrm{T} = t_{st} + t_b(pattern, b)$$

Οι χρόνοι αυτοί εξαρτώνται από τις αρχιτεκτονικές και το δίκτυο ενδοεπικοινωνίας που χρησιμοποιούν. Υπάρχουν δίκτυα με μεγάλο χρόνο εκκίνησης t_{st}, στα οποία όμως η μεταφορά του μηνύματος δεν εξαρτάται ουσιαστικά από τη φυσική απόσταση μεταξύ των επεξεργαστών που επικοινωνούν, και δίκτυα όπου το t_{st} είναι πολύ μικρό αλλά στο χρόνο επικοινωνίας μετράει σημαντικά το πόσο απέχει ο επεξεργαστής-αποδέκτης από τον επεξεργαστή-αποστολέα. Στην πρώτη περίπτωση ο μεγάλος χρόνος εκκίνησης του μηνύματος χρησιμοποιείται προκειμένου να προετοιμαστεί η διαδρομή μεταξύ των δύο επεξεργαστών, έτσι ώστε στη συνέχεια να μην επηρεάζει τόσο πολύ ούτε το μέγεθος του μηνύματος, ούτε η απόσταση μεταξύ των επεξεργαστών. Παράδειγμα αρχιτεκτονικής της πρώτης περίπτωσης είναι το iPSC/2 της Intel ενώ της δεύτερης το Connection Machine της TMC και τα Transputers. Η φυσική απόσταση μεταξύ των επεξεργαστών που επικοινωνούν εξαρτάται από τον τρόπο με τον οποίο το επικοινωνιακό σχήμα της διεργασίας υλοποιείται στη συγκεκριμένη αρχιτεκτονική.

Κάθε αλγόριθμος θα πρέπει να αναλύεται χωριστά στις διαδοχικές *Φάσεις* στις οποίες χωρίζεται από τα διαφορετικά υπολογιστικά και επικοινωνιακά χαρακτηριστικά που απαιτεί. **Ο χρόνος επικοινωνίας** για κάθε φάση i είναι

$$T^i_{comm} = msgs^i * (t_{st} + t_b(pattern^i, b^i))$$

$msgs^i$ είναι το πλήθος των μηνυμάτων που ανταλάσσονται στη φάση αυτή. Το πλήθος αυτό συχνά εξαρτάται από τα χαρακτηριστικά του περιεχομένου της εικόνας.

Ο υπολογιστικός χρόνος που απαιτείται από σε κάθε φάση του, είναι

$$T^i_{comp}(s) = DU^i * \sum_{all\oplus} (\mathrm{a}^i_{\oplus} * t_{\oplus}(s))$$

αν στη φάση i κάθε επεξεργαστής κρατά το πολύ DU^i μονάδες δεδομένων η καθεμία από τις οποίες απαιτεί την εκτέλεση $a^i_{\oplus}$ πράξεων με τελεστή $\oplus$ για κάθε μία από τις οποίες απαιτείται μέσος υπολογιστικός χρόνος $t_{\oplus}(s)$ για μεταβλητές μεγέθους s bits.

Ετσι συνολικά για τις Φ φάσεις του αλγορίθμου, ο χρόνος εκτέλεσης θα είναι

$$T_{tot}(s) = \sum_{i=1}^{\Phi} \left(\mathcal{S}(dstr^{i-1}, dstr^i) + DU^i * \sum_{all\oplus} (a^i * t_{\oplus}(s)) + msgs^i * (t_{st} + t_b(pattern^i, b^i)) \right)$$

Ο εκθέτης i δείχνει τη φάση για την οποία υπολογίζονται τα αντίστοιχα μεγέθη. $\mathcal{S}(dstr^{i-1}, dstr^i)$ είναι μια συνάρτηση κόστους για τη μετατροπή από τη δομή δεδομένων εξόδου $(dstr^{i-1})$ από την προηγούμενη φάση *(i-1)* του συγκεκριμένου αλγορίθμου στη δομή $(dstr^i)$ που απαιτεί η φάση i.

Η έκφραση που μας δίνει τις υπολογιστικές και τις επικοινωνιακές απαιτήσεις ενός αλγορίθμου είναι πολύ συχνά συνάρτηση που εμπλέκει τα χαρακτηριστικά του περιεχομένου της εικόνας εφόσον υπάρχει εξάρτηση από αυτά. Από τη σχέση αυτή θα συγκεκριμενοποιηθούν τα χαρακτηριστικά του περιεχομένου για τα οποία θα ζητηθούν πληροφορίες. Η περιγραφή ιδιαίτερα των επικοινωνιακών αναγκών πρέπει να είναι καλά ορισμένη, έτσι ώστε να είναι δυνατόν να διευρευνηθεί η άμεση υλοποίησή τους στις υπάρχουσες αρχιτεκτονικές. Ακόμα, πρέπει να περιγράφονται όχι μόνο στην τοπολογική τους διάσταση, δηλαδή ποιο υπολογιστικό τμήμα πρέπει να επικοινωνήσει με ποιο, αλλά επίσης και στη χρονική διάσταση, δηλαδή πότε γίνεται αυτή η επικοινωνία σε σχέση με την ολοκλήρωση κάποιων υπολογισμών ή την περάτωση άλλων επικοινωνιών. Αυτό επιτυγχάνεται με την κατάλληλη ανάλυση του αλγορίθμου σε διαδοχικές φάσεις. Για παράδειγμα, στην περίπτωση της μετάδοσης μιας πληροφορίας σε όλους τους επεξεργαστές (broadcasting) μας ενδιαφέρει όχι μόνο να περιγράψουμε τη διαδρομή μέσω της οποίας θα γίνει η μεταφορά της πληροφορίας αλλά και το γεγονός ότι η μετάδοση ενός μηνύματος σε κάποιο κόμβο του γράφου επικοινωνίας εξαρτάται απόλυτα από το χρόνο ολοκλήρωσης της επικοινωνίας μέχρι τον προηγούμενό του κόμβο, πάνω στη διαδρομή αυτή.

5.2.2 Περιγραφή των Αρχιτεκτονικών

Το τμήμα αυτό περιλαμβάνει δύο τμήματα. Στο πρώτο τμήμα, για κάθε αρχιτεκτονική και για κάθε δυνατή διάταξή της υπάρχουν πληροφορίες για τις υπολογιστικές της επιδόσεις (ανά τελεστή, μέγεθος μεταβλητής, κτλ.). Στο δεύτερο μέρος δίνονται οι χρόνοι υλοποίησης μιας σειράς από επικοινωνιακά σχήματα, τα οποία χρησιμοποιούνται στους αλγόριθμους. Προκειμένου να επιτύχουμε τη γρήγορη και καλά ορισμένη περιγραφή τους απαιτείται να υπάρχει μια βιβλιοθήκη επικοινωνιακών σχημάτων η οποία είναι αποθηκευμένη στο δεύτερο τμήμα. Στα επικοινωνιακά σχήματα συμπεριλαμβάνονται και οι προθεματικές πράξεις σάρωσης (prefix operations) καθώς και οι πράξεις συρρίκνωσης (reduction operations), που αποτελούν στην ουσία επικοινωνιακά σχήματα συνδυασμένα με υπολογιστικές πράξεις. Για κάθε ένα από αυτά τα επικοινωνιακά σχήματα δίνονται οι χρόνοι υλοποίησης σαν απόλυτα νούμερα ή ακόμα, αν χρειάζεται, σαν συνάρτηση απλούστερων επικοινωνιακών σχημάτων. Τέτοια επικοινωνιακά σχήματα είναι:

Broadcast(message,condition): το περιεχόμενο του μηνύματος *message* μεταδίδεται σε όλους τους επεξεργαστές που είναι ενεργοί, επειδή ικανοποιούν τη συνθήκη *condition*.

Broadcast-directed(message,condition,direction) το περιεχόμενο του μηνύματος *message* μεταδίδεται κατά τη διεύθυνση *direction* σε όλους τους επεξεργαστές που είναι ενεργοί, επειδή ικανοποιούν τη συνθήκη *condition*.

Communication(to-decide): κάθε επεξεργαστής αποφασίζει ανεξάρτητα την επικοινωνία που χρειάζεται, ανάλογα με τα δεδομένα του.

Pack(var,condition): με την εντολή αυτή οι τιμές της μεταβλητής *var* όσων επεξεργαστών ικανοποιούν τη συνθήκη *condition* (ενεργών επεξεργαστών) τη στιγμή της εκτέλεσης της εντολής, μεταφέρονται σε διαδοχικούς επεξεργαστές, ενεργούς και μη.

Pack-directed(var,condition,direction): οι τιμές της μεταβλητής *var* όσων επεξεργαστών ικανοποιούν τη συνθήκη *condition* (ενεργών επεξεργαστών) τη στιγμή της εκτέλεσης της εντολής, μεταφέρονται σε διαδοχικούς επεξεργαστές, ενεργούς και μη, κατά τη διεύθυνση *direction*.

Prefix(operator,var,condition): προθεματική πράξη σάρωσης με τελεστή *operator* στην τιμή της μεταβλητής *var* που εκτελείται από όσους επεξεργαστές είναι ενεργοί, επειδή ικανοποιούν τη συνθήκη *condition*.

Prefix-directed(direction,operator,var,condition): προθεματική πράξη σάρωσης με τελεστή *operator* στην τιμή της μεταβλητής *var* που εκτελείται από όσους επεξεργαστές κατά τη διεύθυνση *direction* ικανοποιούν τη συνθήκη *condition* (ενεργοί επεξεργαστές).

Reduction(operator,var,condition): πράξη συρρίκνωσης με τελεστή *operator* στην τιμή της μεταβλητής *var* που εκτελείται από όσους επεξεργαστές ικανοποιούν τη συνθήκη *condition* (ενεργοί επεξεργαστές).

Reduction-directed(direction,operator,var,condition): πράξη συρρίκνωσης κατά τη διεύθυνση *direction* με τελεστή *operator* στην τιμή της μεταβλητής *var* που εκτελείται από όσους επεξεργαστές ικανοποιούν τη συνθήκη *condition* (ενεργοί επεξεργαστές).

Send(message,n): Στείλε το περιεχόμενο της μεταβλητής *message* στο συγκεκριμένο επεξεργαστή n.

Send-directed(message,direction,n): Στείλε το περιεχόμενο της μεταβλητής *message* στον n επεξεργαστή πάνω στη διεύθυνση *direction*.

Send-relative-directed(message,direction,δ): Στείλε το περιεχόμενο της μεταβλητής *message* στον επεξεργαστή που βρίσκεται σε απόσταση δ κατά τη διεύθυνση *direction*.

Shrink(var,first,condition): Μετακίνηση των τιμών της μεταβλητής *var* σε όλους τους επεξεργαστές που ικανοποιούν τη συνθήκη *condition* (ενεργοί επεξεργαστές), ώστε να καταλάβουν θέσεις σε διαδοχικούς επεξεργαστές (ενεργούς και μη) αρχίζοντας από τον *first*.

Shrink-directed(var,first,direction,condition): Μετακίνηση των τιμών της μεταβλητής *var* σε όλους τους επεξεργαστές κατά τη διεύθυνση *direction* που ικανοποιούν τη συνθήκη *condition* (ενεργοί επεξεργαστές), ώστε να καταλάβουν θέσεις σε διαδοχικούς επεξεργαστές (ενεργούς και μη) αρχίζοντας από τον *first*.

Sorting(var,condition) : Ταξινόμηση των τιμών της μεταβλητής *var* σε όσους επεξεργαστές ικανοποιούν τη συνθήκη *condition* (ενεργοί επεξεργαστές).

Sorting-directed(var,direction,condition): Ταξινόμηση των τιμών της μεταβλητής *var* κατά τη διεύθυνση *direction* σε όσους επεξεργαστές ικανοποιούν τη συνθήκη *condition* (ενεργοί επεξεργαστές).

Τα επικοινωνιακά αυτά σχήματα μπορούν να εμπλουτιστούν και με άλλα, υψηλότερου ή χαμηλότερου επιπέδου.

5.2.3 Περιγραφή του Περιεχομένου της Εικόνας

Στο τμήμα αυτό είναι συγκεντρωμένες στατιστικές πληροφορίες σχετικές με το είδος, τη συχνότητα εμφάνισης και τα σχετικά μεγέθη των χαρακτηριστικών διαφόρων ειδών εικόνων. Οι πληροφορίες αυτές θα έχουν πιθανοκρατικό χαρακτήρα και θα εξαρτώνται από το πόσο ενημερωμένο είναι το σύστημα. Οι πληροφορίες που θα παρέχονται θα αφορούν τα όρια διακύμανσης του μεγέθους και της συχνότητας εμφάνισης των χαρακτηριστικών.

5.2.4 Τμήμα Ισοκατανομής Φορτίου

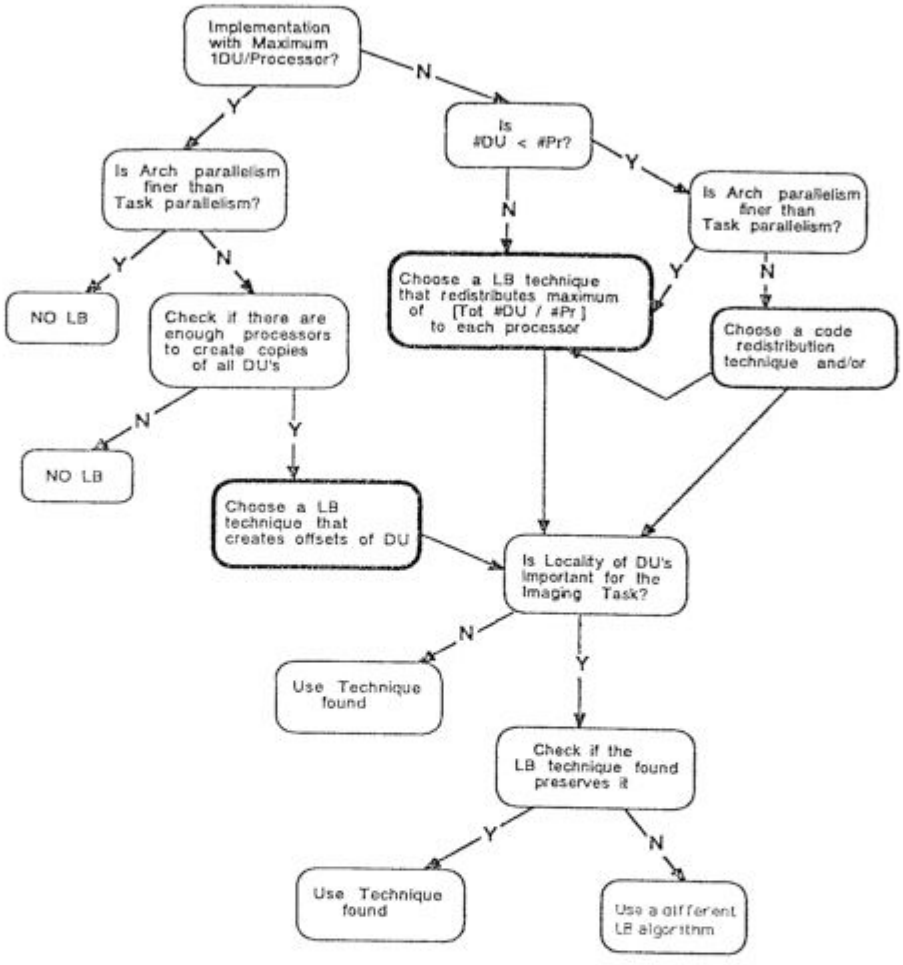
Στην παράγραφο αυτή θα περιγράψουμε ένα δέντρο αποφάσεων (σχήμα 5.2), το οποίο θα αποφασίζει αν πρέπει ή όχι να εφαρμοστεί αλγόριθμος ισοκατανομής φορτίου κατά τη διάρκεια της εκτέλεσης μιας διεργασίας, και, αν ναι, ποιες είναι οι γενικές αρχές, που πρέπει να διέπουν αυτόν τον αλγόριθμο. Το τμήμα αυτό του μοντέλου δεν διερευνάται εκτεταμένα στην παρούσα εργασία και δεν παρουσιάζουμε τον τρόπο περιγραφής των χαρακτηριστικών αυτών των αλγορίθμων. Η έρευνα αυτή θα πρέπει να γίνει μελλοντικά. Η διαδικασία επιλογής συγκεκριμένου αλγορίθμου, η οποία έχει σχέση με την υλοποίηση του συστήματος, καθώς και το αν το πρόσθετο κόστος που εισάγει ο αλγόριθμος τον κάνει μη εκμεταλλεύσιμο, για το οποίο πρέπει να λαμβάνουμε υπόψη τη συγκεκριμένη εφαρμογή, είναι ερωτήματα που δεν αντιμετωπίζονται εδώ.

Κάθε μία από τις διεργασίες που προαναφέραμε έχει τις δικές της απαιτήσεις από τους αλγόριθμους ανακατανομής του φορτίου (Load Redistribution) προκειμένου να επιτευχθεί απο-

δοτικά η ισοκατανομή του (Load Balancing). Καταρχήν, όλοι οι αλγόριθμοι ανάλυσης εικόνων, που λειτουργούν με μία μονάδα δεδομένων ανά επεξεργαστή, δεν επιδέχονται κατά κανόνα ανακατανομή του φορτίου τους, εφόσον αναφερόμαστε σε καθαρά SIMD αρχιτεκτονικές. Η μόνη περίπτωση που μπορεί να γίνει αυτό είναι όταν σε μία μονάδα δεδομένων εκτελούνται οι ίδιοι υπολογισμοί πολλές φορές (π.χ. με διαφορετικούς συντελεστές κάθε φορά). Στην περίπτωση αυτή σαν φορτίο του επεξεργαστή δεν θεωρούνται πλέον τα δεδομένα του αλλά οι υπολογισμοί που θα πρέπει να εκτελέσει. Έτσι μπορεί να στείλει ένα αντίγραφο των δεδομένων του σε κάποιον άλλο επεξεργαστή και να μοιραστεί μαζί του τους υπολογισμούς που θα πρέπει να γίνουν. Όμως θα πρέπει να υπάρχει ικανός αριθμός επεξεργαστών για να αναλάβει τα αντίγραφα όλων των μονάδων δεδομένων που βρίσκονται στην ίδια κατάσταση από πλευράς υπολογιστικών απαιτήσεων, καθώς ακόμα και αν ένας επεξεργαστής δεν βρει κάποιον άλλο για να μοιραστεί τους υπολογισμούς, θα πρέπει όλοι οι υπόλοιποι, που θα τελειώσουν γρηγορότερα, να τον περιμένουν παραμένοντας αδρανείς. Έτσι, δεν πρόκειται να υπάρξει οποιοδήποτε πλεονέκτημα από την ανακατανομή του φορτίου ενώ αντίθετα η εκτέλεση του αλγορίθμου θα επιβαρυνθεί με το πρόσθετο κόστος που θα επέβαλε αυτή η ίδια η διεργασία ανακατανομής.

Στην περίπτωση που έχουμε επεξεργασία με πολλές μονάδες δεδομένων σε κάθε επεξεργαστή, η ισοκατανομή του φορτίου αποκτά διαφορετικό χαρακτήρα. Καταρχήν μια πρώτη σκέψη για ισοκατανομή του φορτίου τους μπορεί να βασιστεί στο να μεταφερθούν μονάδες δεδομένων από επεξεργαστές που διαθέτουν πολλές σε άλλους που έχουν λιγότερες. Η τεχνική αυτή είναι σχετικά απλή και υλοποιείται από διάφορους αλγόριθμους ανάλογα με την τοπολογία και τα άλλα χαρακτηριστικά της αρχιτεκτονικής.

Μια άλλη τεχνική που θα μπορούσε να χρησιμοποιηθεί είναι εκείνη της ανακατανομής των υπολογισμών. Δηλαδή, διάφοροι επεξεργαστές εκτελούν διαφορετικούς υπολογισμούς είτε πάνω στα ίδια δεδομένα (όπως ήδη αναφέραμε) είτε πάνω σε διαφορετικά δεδομένα. Στην ίδια αυτή ομάδα τεχνικών, δηλαδή όταν ανακατανέμονται οι υπολογισμοί που πρέπει να γίνουν στα δεδομένα αντί να κατανεμηθούν τα δεδομένα αυτά καθαυτά, καθοριστικό ρόλο θα παίξει το "επίπεδο" παραλληλίας της αρχιτεκτονικής. Αυτό συμβαίνει τόσο στην περίπτωση που έχουμε μία μονάδα δεδομένων ανά επεξεργαστή όσο και στην περίπτωση που έχουμε πολλές μονάδες δεδομένων. Για να το εξηγήσουμε αυτό θα θυμηθούμε ότι υπάρχουν διάφορα επίπεδα παράλληλης διαμέρισης μιας διεργασίας, στα οποία μπορεί να λειτουργήσει η παραλληλία ως προς τον



Σχήμα 5.2: Δέντρο Αποφάσεων για την επιλογή αλγορίθμου ισοκατανομής φορτίου

κώδικα, από το επίπεδο λειτουργίας των λογικών πυλών ως το επίπεδο εκτέλεσης διεργασιών. *"Επίπεδο παράλληλης διαμέρισης ως προς κώδικα μιας διεργασίας"* ονομάζουμε το χαμηλότερο επίπεδο διαμέρισης στο οποίο μπορούν να εκτελεστούν παράλληλα οι υπολογιστικές της απαιτήσεις. Με ανάλογο τρόπο ορίζουμε το *"επίπεδο παραλληλίας της αρχιτεκτονικής"* σαν το χαμηλότερο επίπεδο διαμέρισης κώδικα στο οποίο η παράλληλη εκτέλεση από κομβικούς επεξεργαστές συγχρονίζεται από το Host. Για παράδειγμα, το Hypercube της Intel έχει επίπεδο παραλληλίας τη διεργασία, ενώ το Connection Machine τη νανοεντολή (nanoinstruction). Από τον ορισμό γίνεται φανερό πως μια αρχιτεκτονική δεν μπορεί να λειτουργήσει παράλληλα ως προς κώδικα (code parallelism) σε υψηλότερο επίπεδο διαμέρισης από το δικό της, μπορεί όμως σε χαμηλότερο. Στα χαμηλότερα επίπεδα παραλληλίας (λογικών πυλών και νανοεντολών) κάθε τι γίνεται ταυτόχρονα σε όλους τους επεξεργαστές ενώ δεν υπάρχουν περιθώρια για ανακατανομή φορτίου υπό μορφή κώδικα και εκείνο που πρέπει να εξασφαλίζεται προκειμένου να επιτυγχάνεται ισοκατανομή του φορτίου, είναι να διαθέσουμε δεδομένα σε όσο το δυνατόν περισσότερους επεξεργαστές. Το ψηλότερο επίπεδο παραλληλίας αντιστοιχεί στην πιο ελεύθερης μορφής παραλληλία και πλησιάζει την επεξεργασία από αρχιτεκτονικές MIMD. Μεταξύ των δύο αυτών επιπέδων υπάρχουν τα επίπεδα διαχείρισης καταχωρητών, εκτέλεσης μικροεντολών, εκτέλεσης εντολών και υποδιεργασίας.

Η παρατήρηση που κάνουμε σ'αυτό το σημείο είναι πως εφόσον το υπολογιστικό μέρος του αλγορίθμου είναι δυνατό να διασπαστεί σε παράλληλα εκτελέσιμα τμήματα, που αντιστοιχούν σε διαμέριση χαμηλότερου επιπέδου από το επίπεδο παραλληλίας της αρχιτεκτονικής, τότε είναι δυνατόν να εφαρμόσουμε αποδοτικά παραλληλία ως προς κώδικα και κατ'επέκταση μπορούμε να θεωρήσουμε σαν φορτίο την εκτέλεση εντολών. Διαφορετικά είναι επιβεβλημένη η παραλληλία ως προς τα δεδομένα.

Ενα βασικό χαρακτηριστικό της διεργασίας ανάλυσης εικόνων, το οποίο θα πρέπει να ληφθεί υπόψη στην τελική επιλογή του αλγορίθμου ισοκατανομής φορτίου είναι η ανάγκη να γνωρίζουν την κατανομή των δεδομένων στο χώρο και άλλοι επεξεργαστές εκτός από εκείνον που τα κρατά. Υπάρχουν διεργασίες στις οποίες οι μονάδες δεδομένων (ορισμένες ή όλες) σχετίζονται με συγκεκριμένες άλλες και ως εκ τούτου θα πρέπει αφενός μεν ο επεξεργαστής που τις επεξεργάζεται να γνωρίζει πού βρίσκονται οι τελευταίες, ώστε να ανταλάσσουν πληροφορίες που απορρέουν από την επεξεργασία τους και αφετέρου οι μονάδες δεδομένων να βρίσκονται

σε επεξεργαστές που δεν απέχουν μεγάλη φυσική απόσταση μεταξύ τους, έτσι ώστε να μην υπάρχουν μεγάλες καθυστερήσεις στην επικοινωνία. Ενα τέτοιο χαρακτηριστικό των διεργασιών περιορίζει τις τεχνικές ανακατανομής του φορτίου που θα μπορούσαν να χρησιμοποιηθούν.

5.2.5 Η λειτουργία της μεθόδου σαν ένα Δένδρο Αποφάσεων

Μέχρι τώρα, στο κεφάλαιο αυτό, παρουσιάσαμε τα μέρη του μοντέλου που περιέχουν την απαραίτητη πληροφορία για τη λήψη των αποφάσεων. Τα μέρη αυτά αναφέρονται στους αλγόριθμους που υλοποιούν τις διεργασίες, στις διαθέσιμες αρχιτεκτονικές, στο περιεχόμενο των εικόνων και τέλος στους αλγόριθμους ισοκατανομής του φορτίου, και αποτελούν το κύριο μέρος του προτεινόμενου μοντέλου. Στο τμήμα αυτό παρουσιάζουμε μία μέθοδο, που εκμεταλλεύεται τα μέρη αυτά του μοντέλου προκειμένου να κάνει εισηγήσεις για τη βέλτιστη υλοποίηση μιας συγκεκριμένης διεργασίας. Στη μέθοδο αυτή δεν λαμβάνεται υπόψη το τελευταίο τμήμα, δηλαδή αυτό της ισοκατανομής του φορτίου, μιας και η ανάλυσή του στην παρούσα εργασία ήταν μόνο θεωρητική. Η μέθοδος δίνεται με τη μορφή ενός δέντρου αποφάσεων.

Η φυσική εκκίνηση της πορείας πάνω στο δένδρο των αποφάσεων πρέπει να γίνεται δίνοντας το τι θέλουμε να κάνουμε, τι διαθέτουμε από πόρους και ποια είναι η μορφή των δεδομένων εισόδου πάνω στα οποία θα γίνει η εφαρμογή. Έτσι, η πρώτη πληροφορία που θα πρέπει να δώσουμε είναι η διεργασία που θέλουμε να υλοποιήσουμε, στη συνέχεια οι αρχιτεκτονικές που διαθέτουμε (μία ή περισσότερες) ενώ τέλος είναι απαραίτητες οι πληροφορίες για τις εικόνες και τα δεδομένα εισόδου. Για το σκοπό αυτό θα πρέπει να δώσουμε τη μορφή της μονάδας δεδομένων (pixel, ευθύγραμμα τμήματα, περιγραφές αντικειμένων κτλ.), καθώς επίσης και τη δομή και το πλήθος των δεδομένων εισόδου.

Με βάση τις πληροφορίες αυτές, το σύστημα εστιάζει την αναζήτηση σε ορισμένους αλγόριθμους οι οποίοι φαίνεται να ταιριάζουν σ'αυτά τα πρώτα δεδομένα. Αν η αρχιτεκτονική πάνω στην οποία θα γίνει η υλοποίηση είναι συγκεκριμένη, το σύστημα θα μπορεί να εστιάσει την αναζήτηση σε κάποιους από τους αλγόριθμους που έχει στη διάθεσή του, οι οποίοι θα έχουν χαρακτηριστικά κατάλληλα για την αρχιτεκτονική αυτή. Για τους αλγόριθμους στους οποίους θα καταλήξει σαν πιθανούς, ζητάει από τον χρήστη όσες πρόσθετες πληροφορίες χρειάζεται σχετικά με το περιεχόμενο της εικόνας, που πιθανά επηρεάζει την απόδοσή του. Από την πε-

ριγραφή των αλγορίθμων αυτών, το σύστημα γνωρίζει ποιοι είναι οι παράγοντες που επηρεάζουν την απόδοση του καθενός και χρησιμοποιεί αυτή τη γνώση στην επιλογή των ερωτήσεών του. Ακόμα, υπολογίζει τη σχέση $t_{st} + t_b(pattern, b)$ του επικοινωνιακού σχήματος (ή των επικοινωνιακών σχημάτων) κάθε αλγορίθμου για τη συγκεκριμένη αρχιτεκτονική (ή αρχιτεκτονικές) που του αναφέρθηκαν ή που επέλεξε.

Το σύστημα, αφού έχει μαζέψει όλες τις απαραίτητες πληροφορίες, κάνει μια πρώτη πρόβλεψη της απόδοσης κάθε αλγορίθμου και αποφασίζει ποιους από αυτούς θα πρέπει να εισηγηθεί στο χρήστη. Σε περίπτωση που ορισμένα δεδομένα δεν είναι γνωστά, είναι δυνατόν να κάνει υποθέσεις στηριγμένες στην εμπειρία του και να δώσει μια σειρά από εισηγήσεις με κάποιο βαθμό προτίμησης για την κάθε μία. Συγχρόνως, θα πρέπει να βγουν συμπεράσματα για την ανάγκη χρησιμοποίησης αλγόριθμου ισοκατανομής φορτίου και, σε περίπτωση θετικής ενδείξεως, θα πρέπει να προσδιορίσει τα χαρακτηριστικά του.

Σε περίπτωση που έχουμε μια ακολουθία από διεργασίες, η επιλογή του αλγόριθμου για μια διεργασία επηρεάζεται από την επιλογή για την προηγούμενή της, ιδιαίτερα στην περίπτωση που τα δεδομένα εισόδου της διεργασίας είναι τα δεδομένα εξόδου της προηγούμενης.

Από τα όσα είπαμε μέχρι τώρα, γίνεται φανερό ότι η διαδικασία που περιγράψαμε αντιστοιχεί στη δυναμική δημιουργία και διάσχιση ενός δένδρου αποφάσεων. Ξεκινώντας από τη ρίζα του δένδρου, το σύστημα έχει να επιλέξει ανάμεσα από όλους τους γνωστούς του αλγόριθμους για κάθε διεργασία. Μόλις δώσει ο χρήστης τις αρχικές πληροφορίες, το σύστημα περιορίζει το πεδίο της έρευνάς του (search space) και καταλήγει σε ένα νέο κόμβο του δένδρου. Εκεί το σύστημα μπορεί να προτείνει κάποιους αλγόριθμους που υλοποιούν την εν λόγω διεργασία. Για να προχωρήσει το σύστημα στην επιλογή, μεταξύ αυτών των αλγορίθμων, εκείνου (ή εκείνων) με την καλύτερη δυνατή απόδοση χρειάζεται επιπλέον πληροφορία που την παίρνει αφού υποβάλλει στο χρήστη κάποιες νέες ερωτήσεις. Πρέπει να πούμε εδώ ότι το δένδρο αποφάσεων δεν είναι πλήρως σχηματισμένο από την αρχή, αλλά δημιουργείται δυναμικά βάσει κανόνων που απορρέουν από την περιγραφή των αλγορίθμων, οι οποίοι καθορίζουν το πώς θα δημιουργηθεί κάθε φορά ένα νέο υποδένδρο κάτω από τον κόμβο στον οποίο βρίσκεται το σύστημα. Οι απαντήσεις του χρήστη χρησιμοποιούνται για να παρθεί η απόφαση για το πώς θα κινηθούμε μέσα σε αυτό το υποδένδρο.

Στο επόμενο κεφάλαιο παρουσιάζουμε αλγόριθμους για την επίλυση ορισμένων βασικών προβλημάτων Επεξεργασίας Εικόνων και τους αναλύουμε σύμφωνα με τη μεθοδολογία που αναπτύξαμε στο κεφάλαιο αυτό. Επίσης διερευνούμε αναλυτικά και πειραματικά την αλληλεπίδραση των γνωρισμάτων των αρχιτεκτονικών με εκείνα του περιεχόμενου των εικόνων καθώς επίσης την επίπτωση που έχει η αλληλεπίδραση αυτή στην απόδοση συγκεκριμένων αλγορίθμων. Οι συσχετίσεις αυτές μας οδηγούν σε κλειστούς τύπους (closed forms) που υπαγορεύουν τη βέλτιστη υλοποίηση σε κάθε περίπτωση. Οι κλειστοί αυτοί τύποι είναι εκείνοι που θα πρέπει να χρησιμοποιηθούν προκειμένου να παίρνονται οι αποφάσεις για την πλοήγηση πάνω στο δέντρο των αποφάσεων, με άλλα λόγια αποτελούν τα κλειδιά για τη λειτουργία της μεθόδου.

Τα βήματα που ακολουθούμε είναι τα εξής: παρουσιάζουμε αρχικά τους αλγόριθμους για κάθε διεργασία και την ανάλυσή τους για υλοποίηση σε αρχιτεκτονική υψηλής διαμέρισης (fine granularity) με μια μονάδα δεδομένων ανά επεξεργαστή. Στη συνέχεια γίνεται το ανάλογο για υλοποίηση σε αρχιτεκτονική χαμηλής διαμέρισης (coarse granularity) με πολλές μονάδες δεδομένων ανά επεξεργαστή. Σε αρκετές περιπτώσεις γίνεται υλοποίηση περισσότερων του ενός αλγορίθμων. Ακολουθούν τα πειραματικά δεδομένα από κάθε υλοποίηση και εξάγονται οι υπολογιστικοί μηχανισμοί για την πρόβλεψη των επιδόσεων οι οποίοι επαληθεύονται πειραματικά.

5.3 Συμπεράσματα

Στο κεφάλαιο αυτό περιγράψαμε γενικά ένα μοντέλο επιλογής παράλληλων υλοποιήσεων διεργασιών ανάλυσης εικόνων με τα επιμέρους τμήματά του, καθώς επίσης τον τρόπο λειτουργίας ενός δέντρου αποφάσεων που αποτελεί το μηχανισμό κεντρικού ελέγχου του μοντέλου αυτού. Από την παρουσίαση αυτή γίνεται φανερός ο παραμετρικός τρόπος με τον οποίο θα πρέπει να αναπαριστούμε τις συνιστώσες της παράλληλης υλοποίησης διεργασιών ανάλυσης εικόνων προκειμένου να επιτύχουμε τη βέλτιστη συνεργασία τους.

Άλλες σχετικές μέθοδοι που έχουν μελετηθεί αναφέρονται σε ισομορφισμούς γράφων και είναι κατάλληλες για αρχιτεκτονικές MIMD. Το μοντέλο που περιγράφουμε εκμεταλλεύεται τη δυνατότητα υλοποίησης μιας διεργασίας με διαφορετικούς αλγόριθμους και λαμβάνει υπόψη τις επικοινωνιακές και υπολογιστικές απαιτήσεις των αλγορίθμων αυτών, το περιεχόμενο της

εικόνας και, σε συνδυασμό με τις αντίστοιχες επιδόσεις της αρχιτεκτονικής, καταλήγει στη βέλτιστη κάθε φορά εισήγηση υλοποίησης.

Στο κεφάλαιο που ακολουθεί παρουσιάζουμε την εφαρμογή της μεθόδου σε συγκεκριμένες διεργασίες, από όπου φαίνεται η επίδραση που μπορεί να έχουν όλοι αυτοί οι παράγοντες στην τελική επίδοση μιας παράλληλης υλοποίησης.

Κεφάλαιο 6

Πειραματική Διερεύνηση Παράλληλων Υλοποιήσεων για Ανάλυση Εικόνων

6.1 Γενικά

Στο κεφάλαιο αυτό παρουσιάζεται η πειραματική διερεύνηση της λειτουργίας μιάς μεθόδου για την επιλογή της καλύτερης παράλληλης υλοποίησης διεργασιών ανάλυσης εικόνων, όπως αυτή που περιγράφηκε στο προηγούμενο κεφάλαιο. Τα αποτελέσματα εξήχθησαν από την υλοποίηση και τη μελέτη δύο μέσου επιπέδου διεργασιών ανάλυσης εικόνων: 1) τον υπολογισμό του Κυρτού Περιγράμματος (Convex Hull) ενός συνόλου σημείων, και 2) την εύρεση των Συνδεδεμένων Συστατικών μιας εικόνας. Θεωρούμε ότι οι δύο παραπάνω διεργασίες είναι αντιπροσωπευτικές μιας ευρύτερης κλάσης διεργασιών ανάλυσης εικόνων, για τις οποίες η βασική μονάδα δεδομένων είναι το σημείο (pixel) ενώ χαρακτηρίζονται από ομογενή επικοινωνία, ικανοποιητική παραλληλοποιησιμότητα και την απαίτηση η θέση κάθε σημείου που ανήκει σε

αντικείμενο να είναι γνωστή και σε άλλους επεξεργαστές εκτός από αυτόν που το επεξεργάζεται. Απ'την άλλη μεριά, ο υπολογισμός του κυρτού περιγράμματος διαφέρει από τη διεργασία της εύρεσης των συνδεδεμένων συστατικών ως προς το γεγονός ότι η πρώτη ολοκληρώνεται σε δύο φάσεις, η καθεμία από τις οποίες απαιτεί διαφορετικές δομές δεδομένων και διαφορετικά επικοινωνιακά σχήματα, ενώ χρειάζεται πολλή περισσότερη υπολογιστική ισχύ. Αυτές οι διαφορές μεταξύ των δύο διεργασιών ενισχύουν τη γενικότητα των συμπερασμάτων. Προκειμένου να δείξουμε τη γενικότερη εφαρμογή της μεθόδου, αναλύουμε στη συνέχεια και άλλες διεργασίες, που είναι αντιπροσωπευτικές κλάσεων με διαφορετικά χαρακτηριστικά.

Οι αλγόριθμοι που αναπτύσσονται σ' αυτό το κεφάλαιο για κάθε διεργασία υλοποιούνται με δύο τρόπους: *πρώτον* σε αρχιτεκτονική υψηλής διαμέρισης με μία μονάδα δεδομένων ανά επεξεργαστή (στο Connection Machine) και δεύτερον σε αρχιτεκτονική χαμηλής διαμέρισης με πολλές μονάδες δεδομένων ανά επεξεργαστή (στο iPSC/2).

6.2 Διεργασία υπολογισμού κυρτού περιγράμματος (Convex Hull)

Σαν Κυρτό Περίγραμμα (ή Περίβλημα) (ΚΠ) ενός συνόλου S από τυχαία σημεία στο επίπεδο ορίζεται το κυρτό πολύγωνο, ελάχιστου εμβαδού, που τα περιέχει [172]. Στην παρούσα εργασία θα ασχοληθούμε με τη διεργασία αυτή στο επίπεδο αλλά, με κατάλληλες επεκτάσεις, οι αλγόριθμοι μπορούν να εφαρμοστούν και σε περισσότερες διαστάσεις.

Η διεργασία του Κυρτού Περιγράμματος έχει ένα ευρύ φάσμα εφαρμογών. Μια από τις σημαντικότερες είναι στην Ανάλυση Εικόνων, όπου χρησιμοποιείται ευρέως για την περιγραφή και αναγνώριση αντικειμένων. Προκειμένου να γίνει η αναγνώριση των αντικειμένων που εμφανίζονται σε μια εικόνα, τα ταξινομούμε ως προς διάφορα χαρακτηριστικά, πολλά από τα οποία μπορούν να υπολογιστούν εύκολα αν γνωρίζουμε το ΚΠ τους. Για παράδειγμα, το μέγεθος της διαμέτρου του αντικειμένου υπολογίζεται εύκολα από το Κυρτό του Περίγραμμα, αφού η διάμετρος ενός πεπερασμένου συνόλου σημείων είναι η διάμετρος του Κυρτού Περιγράμματός του [Θ. 4.16] [172]. Μια άλλη περιοχή εφαρμογών είναι η Στατιστική με την οποία η Γεωμετρία είναι αλληλένδετη αφού κάθε δείγμα στατιστικό μπορεί να θεωρηθεί και σαν σημείο στον

Ευκλείδειο χώρο. Διάφορα προβλήματα στατιστικής, τα οποία ανάγονται στην εύρεση του ΚΠ ενός τμήματος του δειγματικού χώρου, αναφέρονται στη βιβλιογραφία.

6.2.1 Σειριακή επίλυση του προβλήματος

Υπάρχουν διάφοροι αλγόριθμοι για τη λύση αυτού του προβλήματος, τόσο σειριακοί όσο και παράλληλοι. Εν γένει, οι αλγόριθμοι αυτοί λειτουργούν σε δύο φάσεις. Στην πρώτη φάση εντοπίζονται τα σημεία που βρίσκονται στο περίγραμμα του αντικειμένου και στη συνέχεια, από τα σημεία αυτά απορρίπτονται εκείνα που δημιουργούν κοίλες γωνίες. Για την εργασία αυτή σχεδιάσαμε ένα σειριακό αλγόριθμο, ο οποίος θα μπορεί εύκολα να προσαρμοσθεί σ' ένα περιβάλλον παράλληλης επεξεργασίας, εκμεταλλευόμενος τις δυνατότητες των αρχιτεκτονικών που χρησιμοποιούμε. Μοιάζει με εκείνον που ανέπτυξε ο Andrew [173] στο ότι και οι δύο υπολογίζουν δύο ημιπεριγράμματα και επίσης παρουσιάζει κοινά σημεία με εκείνον του Graham [174] αφού, όπως και οι περισσότεροι γνωστοί αλγόριθμοι, χρησιμοποιούν στο δεύτερο μέρος τους τον αλγόριθμο *Graham-scan*. Ο τελευταίος έδωσε μία από τις πρώτες δημοσιεύσεις για το πρόβλημα περιγράφοντας έναν αλγόριθμο με πολυπλοκότητα $O(N\log N)$ για την κατασκευή του κυρτού περιγράμματος N σημείων. Ο σειριακός αλγόριθμος που θα παρουσιάσουμε στη συνέχεια δέχεται στην είσοδό του ένα πίνακα PxP και έχει υπολογιστική πολυπλοκότητα $O(P^2)$ και πολυπλοκότητα μνήμης επίσης $O(P^2)$ στη σειριακή μορφή του.

Ας υποθέσουμε πως για κάθε σημείο A ενός αντικειμένου έχουμε τις συντεταγμένες του x_A, y_A σε ένα ορθογώνιο σύστημα αξόνων O_{xy}.

Ξεκινάμε από τρεις παρατηρήσεις:

Παρατήρηση Α: *Κάθε ευθεία παράλληλη στον άξονα x που περνά από το εσωτερικό ενός Κυρτού Περιγράμματος θα το τέμνει σε δύο το πολύ σημεία.*

Αυτό αποδεικνύεται ουσιαστικά από το θεώρημα [Θ.3.5] [172] με ακραία περίπτωση η ευθεία να εφάπτεται του περιγράμματος οπότε έχουμε ένα κοινό σημείο μεταξύ τους. Αλλη ακραία περίπτωση είναι το να συμπίπτει η ευθεία με κάποια οριζόντια πλευρά του περιγράμματος. Στην περίπτωση αυτή όλα τα κοινά τους σημεία ανήκουν στο ΚΠ.

Παρατήρηση Β: *Αν τα δύο σημεία είναι τα A και B, με* $x_A < x_B$ και $y_A = y_B$ *τότε* $x_A < x_i < x_B$ *για κάθε σημείο* $i \in S$ *με* $y_A = y_i = y_B$.

Παρατήρηση Γ: *Τα δύο (τουλάχιστον) ακραία σημεία κατά την κατεύθυνση X και τα άλλα δύο (τουλάχιστον) ακραία κατά την κατεύθυνση Y ανήκουν στο ΚΠ του S*

Ο αλγόριθμος τότε έχει ως εξής:

Ας υποθέσουμε πως έχουμε ένα σύνολο S σημείων *(i,j)* πάνω σε μία εικόνα διαστάσεων *PxP* στα οποία η ένταση $I_{i,j} > 0$. Σε πρώτη φάση, χρησιμοποιώντας τον αλγόριθμο Contour του σχήματος 6.1, δημιουργούμε δύο μονοδιάστατους πίνακες $MIN_x[P]$ και $MAX_x[P]$ με τις ελάχιστες και μέγιστες συντεταγμένες των σημείων της εικόνας κατά τη x διάσταση για κάθε μία από τις P γραμμές της εικόνας. Ουσιαστικά οι δύο αυτοί πίνακες θα κρατούν το αριστερό και δεξί ημιπερίγραμμα (semi-contour) του αντικειμένου. Στο Σχήμα 6.5 βλέπουμε τα σημεία ενός αντικειμένου μιας εικόνας διαστάσεων 16x16. Στο επόμενο Σχ. 6.6 υπάρχουν τα σημεία μόνο του περιγράμματος ενώ εκατέρωθεν φαίνονται οι τιμές των πινάκων MIN_x και MAX_x.

```
        Procedure Contour
(1)     For i := 1 to P
(2)         j := 0
(3)         while ((I_{i,j+1} = 0) and (j < P)) j := j + 1
(4)         if ((j ≤ P) and (j > 0)) then  MIN_x[i] := j
(5)         j := P+1
(6)         while ((I_{i,j-1} > 0) and (j > 0)) j := j - 1
(7)         if ((j ≤ P) and (j > 0)) then  MAX_x[i] := j
(8)     EndFor
```

Σχήμα 6.1: Ο σειριακός αλγόριθμος *Contour*

Αν σε κάποια γραμμή y δεν υπάρχει κανένα σημείο, οι αντίστοιχες θέσεις στους πίνακες έχουν τιμή μηδέν. Αν υπάρχει ένα μόνο σημείο το περιλαμβάνουν και οι δύο πίνακες. Επίσης για λόγους, που εξυπηρετούν την απλότητα του αλγορίθμου, υποθέτουμε ότι κάθε στοιχείο των

```
        Procedure Graham-scan
(1)     current = index of first non-zero element of MIN_x[P];
(2)     prev = PREV(current);
(3)     next = NEXT(current);
(4)     While ( prev ≤ N )
(5)         if ∠(prev, current, next) is concave then
(6)                 DISCARD(current);
(7)                 current = prev
(8)                 prev = PREV(current);
(9)                 next = NEXT(current);
(10)            else
(11)                current = next
(12)                prev = PREV(current);
(13)                next = NEXT(current);
(14)    EndWhile
```

Σχήμα 6.2: Ο σειριακός αλγόριθμος *Graham-scan*

δύο αυτών πινάκων με μη μηδενική τιμή, γνωρίζει τη θέση του προηγούμενού του μη μηδενικού στοιχείου, καθώς επίσης και του επόμενού του. Για το σκοπό αυτό χρησιμοποιούνται βοηθητικοί πίνακες που για λόγους απλοποίησης δεν θα αναφέρονται στο εξής. Τις τιμές αυτές για κάποιο στοιχείο i ας υποθέσουμε πως τις επιστρέφουν οι συναρτήσεις PREV(i) και NEXT(i) αντίστοιχα.

Στη δεύτερη φάση εφαρμόζεται το δεύτερο στάδιο του αλγόριθμου Graham-Hull που εφαρμόζει την Graham-scan [172] σε κάθε ένα χωριστά από τους δύο πίνακες MIN_x[P] και MAX_x[P]. Ο αλγόριθμος αυτός απαλείφει κάθε σημείο που δημιουργεί κοίλη γωνία με το προηγούμενο και το επόμενό του και ως εκ τούτου δεν ανήκει στο περίγραμμα.

Ο αλγόριθμος **Graham-scan** φαίνεται στο Σχ. 6.2. Ο αλγόριθμος εκεί αναφέρεται μόνο στον πίνακα MIN_x[P] αλλά εντελώς ανάλογα λειτουργεί και για τον πίνακα MAX_x[P].

Η συνάρτηση DISCARD(*current*) είναι η εξής:

Procedure **DISCARD** (*current*)

(1) PUT-NEXT(PREV(*current*), NEXT(*current*);

(2) PUT-PREV(NEXT(*current*), PREV(*current*);

(3) $MIN_x[current] = 0$;

Οι διαδικασίες PUT-NEXT(i,n) και PUT-PREV(i,p) τοποθετούν τις τιμές n και p στους δείκτες για το επόμενο και προηγούμενο αντίστοιχα μη μηδενικό στοιχείο του στοιχείου i.

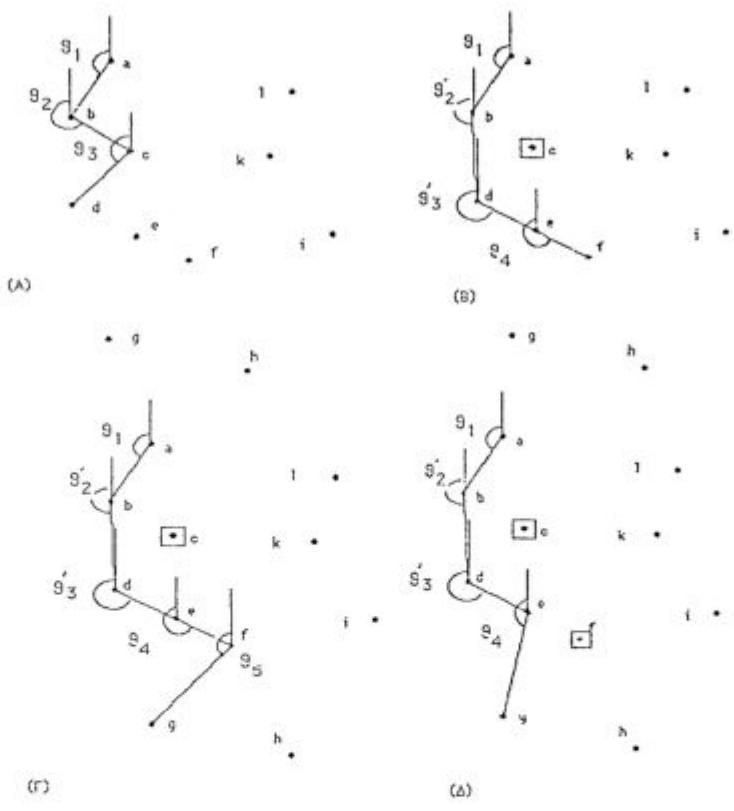
Με το πέρας της διαδικασίας **Graham-scan** οι πίνακες MIN_x[P] και MAX_x[P] διαθέτουν αντίστοιχα το αριστερό και δεξί κυρτό ημιδιάγραμμα του αντικειμένου. Το τελικό ΚΠ δημιουργείται με την ένωση των δύο αυτών κυρτών ημιδιαγραμμάτων.

Τα Σχήματα 6.3 και 6.4 δείχνουν την εξέλιξη του αλγορίθμου του Graham-scan για τα σημεία a,b,c,d,e,f,g,h,i,k,l. Στο 6.3.(Α) φαίνονται οι διαδοχικές πλευρές του περιγράμματος που ελέγχονται. Οταν ο έλεγχος φτάνει στο c αυτό εξαλείφεται και ο έλεγχος αρχίζει ξανά από το b (Σχ. 6.3.(Β)).

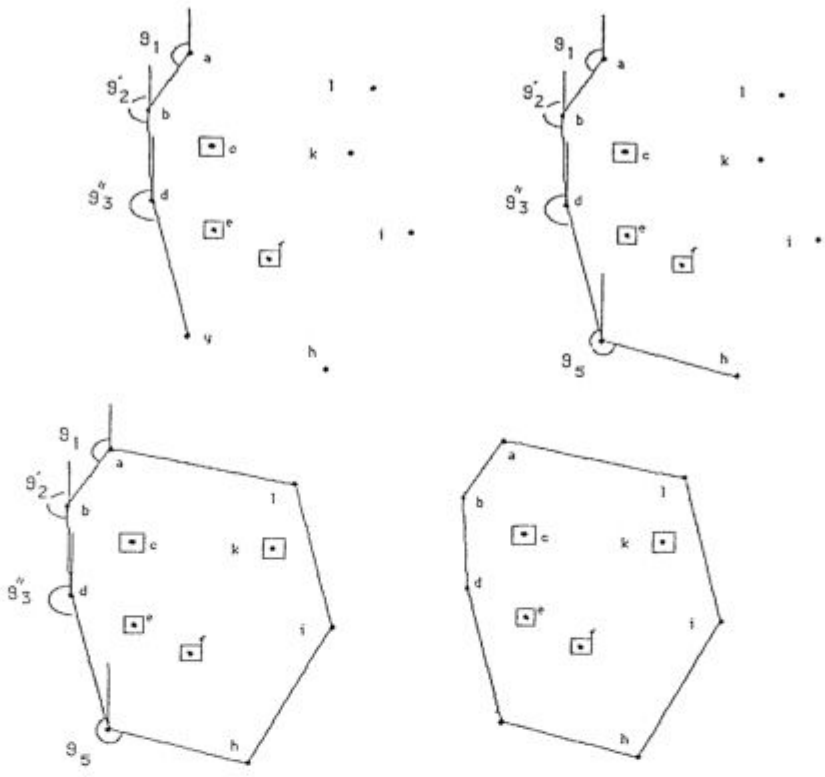
Ομοίως, όταν ο έλεγχος φτάνει στο f αυτό εξαλείφεται, επανεξετάζεται το e και εξαλείφεται κι αυτό. Η διαδικασία συνεχίζει όπως φαίνεται στα επόμενα Σχήματα 6.4.(Α) και 6.4.(Β). Μετά από παρόμοια επεξεργασία του δεξιού ημιπεριγράμματος, καταλήγουμε στο ΚΠ του Σχ. 6.4.(Δ). Μέσα σε τετράγωνα είναι τα σημεία του περιγράμματος που απαλείφθηκαν.

Ο αλγόριθμος που παρουσιάστηκε διαφέρει από τους προαναφερθέντες μόνο ως προς το πρώτο μέρος. Ο Graham στο πρώτο στάδιο ταξινομεί τα σημεία $\pi_1, \pi_2 ... \pi_N$ του αντικειμένου ως προς τις πολικές τους γωνίες προς ένα εσωτερικό σημείο Ε του αντικειμένου. Μολονότι η πολυπλοκότητα μνήμης είναι επίσης O(N), η υπολογιστική πολυπλοκότητα, λόγω της ταξινόμησης την οποία αποφεύγουν οι δύο άλλοι, είναι O(NlogN).

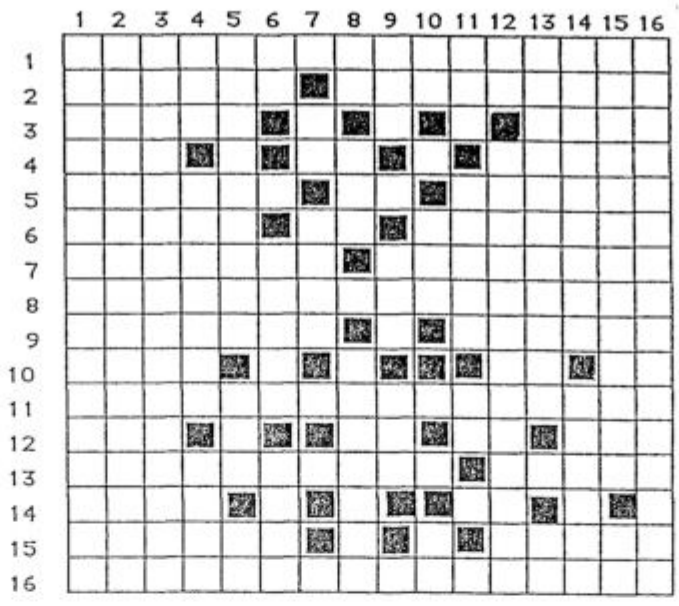
Και οι τρεις αλγόριθμοι χρησιμοποιούν στο δεύτερο στάδιό τους τον αλγόριθμο Graham-scan. Στα επόμενα Σχ. δίνεται παραστατικά η λειτουργία του αλγορίθμου: Το Σχήμα 6.5 δείχνει τα σημεία ενός αντικειμένου σε μια εικόνα 16x16. Το Σχήμα 6.6 έχει μόνο τα σημεία



Σχήμα 6.3: Η εξέλιξη του Αλγόριθμου Graham-scan



Σχήμα 6.4: Η εξέλιξη του Αλγόριθμου Graham-scan (συνέχεια)

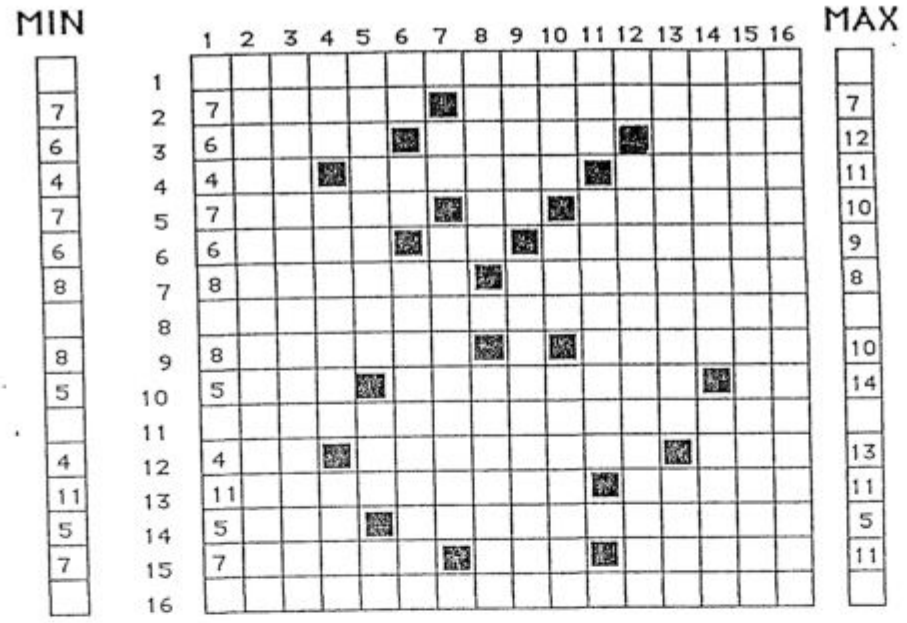


Σχήμα 6.5: Ενα αντικείμενο σε μια εικόνα 16x16

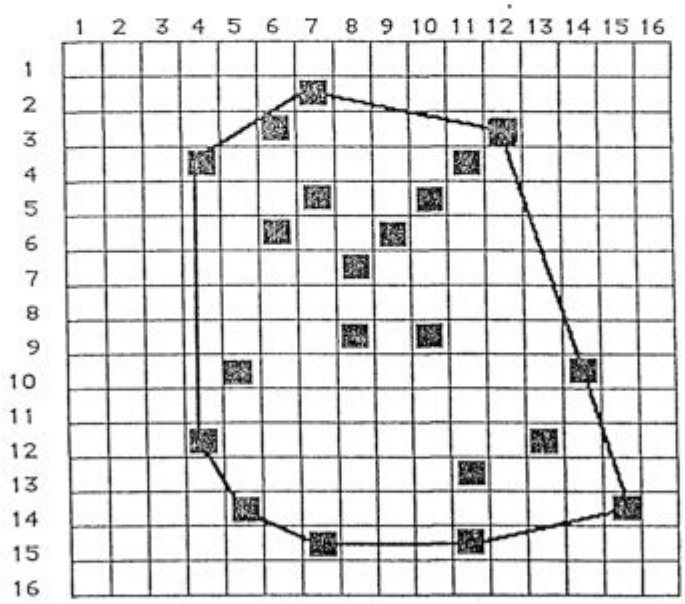
εκείνα που ανήκουν στο περίγραμμα του αντικειμένου ενωμένα για παραστατικούς λόγους, ενώ φαίνονται και οι αντίστοιχοι πίνακες MIN_x και MAX_x. Στο Σχήμα 6.7 φαίνεται τέλος το ΚΠ του αντικειμένου διαγραμμισμένο.

Οι αλγόριθμοι που θα παρουσιάσουμε στο επόμενο τμήμα αποτελούν δύο εκδόσεις του ίδιου κατά βάσει αλγορίθμου, και διαφέρουν μεταξύ τους στην πρώτη φάση της προεπεξεργασίας των δεδομένων εισόδου. Η πρώτη έκδοση δέχεται σαν δομή δεδομένων εισόδου μία λίστα με N σημεία και έχει πολυπλοκότητα $O(N \log N)$, ενώ η δεύτερη έκδοση δέχεται στην είσοδό της ένα πίνακα PxP όπως περιγράφτηκε νωρίτερα.

Τα πειραματικά αποτελέσματα που ακολουθούν προέρχονται από παράλληλες υλοποιήσεις των αλγορίθμων αυτών, τόσο σε αρχιτεκτονική αδρής διαμέρισης (coarsed grained, iPSC/2) με πολλές μονάδες επεξεργασίας ανά επεξεργαστή, όσο και σε αρχιτεκτονική λεπτής διαμέρισης (fine grained, Connection Machine) με μία μονάδα δεδομένων ανά επεξεργαστή.



Σχήμα 6.6: Το περίγραμμά του και οι αντίστοιχοι πίνακες MIN_x και MAX_x



Σχήμα 6.7: Το Κυρτό του Περίγραμμα διαγραμμισμένο

6.2.2 Παράλληλη υλοποίηση με μία μονάδα δεδομένων ανά επεξεργαστή

Ο αλγόριθμος που υλοποιήθηκε στο Connection Machine λαμβάνει υπόψη τις ιδιαίτερες δυνατότητες και χαρακτηριστικά της αρχιτεκτονικής, έτσι ώστε να πετύχει βέλτιστη απόδοση. Συγκεκριμένα κάνει χρήση των προθεματικών πράξεων σάρωσης (prefix operations) που διαθέτει το σύστημα καθώς και του μεγάλου αριθμού των επεξεργαστών του. Δέχεται σαν είσοδο ένα διδιάστατο πίνακα PxP σημείων που αντιπροσωπεύουν την εικόνα και δίνει σαν έξοδο το Κυρτό Περίγραμμα των αντικειμένων. Υλοποιείται σε δύο στάδια. Στο πρώτο στάδιο βρίσκει το εξωτερικό περίγραμμα του αντικειμένου που δεν είναι κατ'ανάγκη κυρτό:

Κάθε επεξεργαστής έχει μία μεταβλητή *pixel* που αντιπροσωπεύει την τιμή του αντίστοιχου σημείου της εικόνας. Η μεταβλητή *pixel* παίρνει τιμή 1 για σημείο που ανήκει στο αντικείμενο ή τιμή 0 για σημείο που ανήκει στο φόντο (background). p_{ij} είναι ο επεξεργαστής στη θέση (i,j) του πλέγματος και έχει σαν γείτονές του τους επεξεργαστές $p_{i-1,j}$ και $p_{i+1,j}$ στην κατακόρυφη διεύθυνση και $p_{i,j-1}$ και $p_{i,j+1}$ στην οριζόντια. Αντίστοιχα, $pixel_{ij}$ είναι η τιμή της μεταβλητής *pixel* που κρατά ο επεξεργαστής p_{ij} του πλέγματος. Γενικά var_{ij} είναι η τιμή της μεταβλητής *var* στον επεξεργαστή p_{ij}.

Οι εντολές (3-4) και (6-7) του σειριακού προγράμματος του Σχ. 6.1, εκτελούνται με εφαρμογή μίας προθεματικής πράξης σάρωσης **scan-max** και μίας **scan-copy-backwards** η κάθε μία. Ενώ η ανακύκλωση (1)-(8) ουσιαστικά δεν υλοποιείται αφού οι εντολές (2) έως και (7) εκτελούνται παράλληλα για κάθε γραμμή P της εικόνας.

Οι βασικές εντολές του παράλληλου αλγορίθμου είναι οι παρακάτω και εκτελούνται παράλληλα από κάθε επεξεργαστή p_{ij}. Η παράλληλη μορφή μιας εντολής συμβολίζεται με ένα * στην αρχή της ή με !! στο τέλος της.

(1) **if!!** ($pixel_{ij} = 1$) **then** $tempx_{ij}$:= x-coordinate
(2) (**set!!** *max-x* (**scan-grid!!** *tempx* **max!!** :dimension :x))
(3) (**set!!** *min-x* (**scan-grid!!** *tempx* **min!!** :dimension :x :direction :backward))

Με την εντολή (1), κάθε επεξεργαστής p_{ij} που κρατάει μη μηδενικό στοιχείο της εικόνας, εκχωρεί την τιμή της x-συντεταγμένης του στο ορθογώνιο πλέγμα του συστήματος στη μεταβλητή

$tempx_{ij}$.

Η (2) εκτελεί μια πράξη σάρωσης κατά τη x κατεύθυνση και εκχωρεί στη μεταβλητή $max - x_{iP}$ του τελευταίου επεξεργαστή κάθε γραμμής i του πλέγματος των επεξεργαστών (δηλαδή στον p_{iP}), τη μεγαλύτερη τιμή της μεταβλητής $tempx_{ij}$ όπου P είναι το μέγεθος της x διάστασης του πλέγματος, που συμπίπτει με το μέγεθος της αντίστοιχης διάστασης της εικόνας. Έτσι στη γραμμή G για παράδειγμα, ο επεξεργαστής p_{GP}, θα έχει στην τιμή της μεταβλητής $max\text{-}x_{GP}$ τη συντεταγμένη x του τελευταίου προς τα δεξιά μη μηδενικού σημείου, που υπάρχει στη γραμμή G της εικόνας.

Ανάλογα, η εντολή (3) εκχωρεί στη μεταβλητή $min - x_{1i}$ του πρώτου επεξεργαστή κάθε γραμμής i, τη συντεταγμένη x του πρώτου από αριστερά μη μηδενικού σημείου της γραμμής.

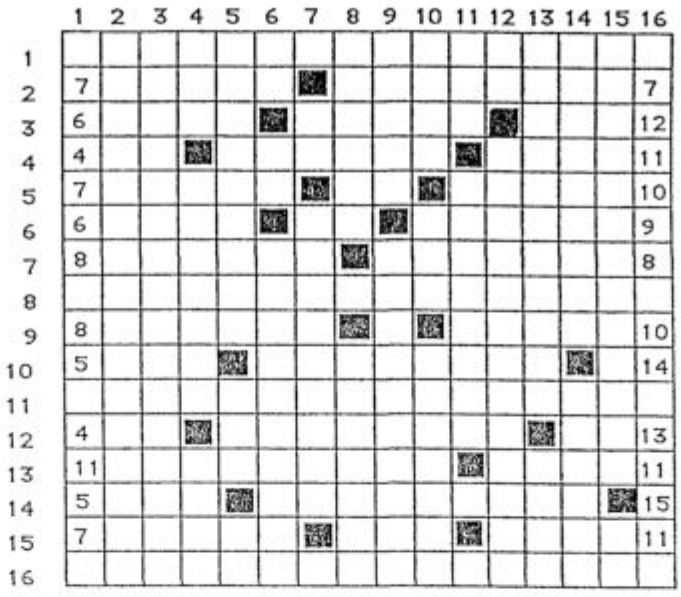
Με το πέρας των εντολών αυτών οι επεξεργαστές με συντεταγμένη 1 κατά τη διάσταση x στο ορθογώνιο πλέγμα τους, κρατούν τον πίνακα MIN_x ενώ εκείνοι με συντεταγμένη P κρατούν τον πίνακα MAX_x. Ισχύει δε, $\text{MIN}_x[\text{i}] = min - x_{i1}$ και $\text{MAX}_x[\text{i}] = max - x_{iP}$.

Το Σχήμα 6.8 δείχνει την 16x16 εικόνα του Σχ. 6.5 σε ένα σύστημα 16x16 επεξεργαστών και ποιοι από αυτούς κρατούν τα στοιχεία των πινάκων MIN_x και MAX_x.

Η υλοποίηση του Graham-scan γίνεται στη συνέχεια με χρήση ενός συνδυασμού προθεματικών πράξεων αλλά και πράξεων επικοινωνίας κατά την κατεύθυνση y αυτή τη φορά.

Οι βασικές εντολές υλοποίησης αυτού του αλγόριθμου στο Connection Machine φαίνονται στο Σχ. 6.9.

Με την εντολή (1) κάθε επεξεργαστής που κρατάει ένα μη μηδενικό στοιχείο του πίνακα MIN_x ή του πίνακα MAX_x, σηκώνει μια σημαία που θα χρησιμοποιήσει αργότερα. Στο εξής κάθε επεξεργαστής με τιμή TRUE στη μεταβλητή *flag* θα λέγεται **ενεργός**. Κάτι ανάλογο γίνεται με την εντολή (2) όπου κάθε ενεργός επεξεργαστής ελέγχει αν είναι στο δεξί ήμιου του ορθογωνικού πλέγματος οπότε και σηκώνει μια δεύτερη σημαία $flag_{right}$ Η εντολή (3) μεταφέρει την τιμή που κρατά κάθε επεξεργαστής στον επόμενο ενεργό. Επειδή από τη λειτουργία της εντολής αυτής η τιμή φτάνει μέχρι τον προηγούμενο επεξεργαστή από εκείνον που έχει τιμή TRUE στην *flag*, η εντολή (3) ολοκληρώνεται από την (4). Στην εντολή αυτή κάθε ενεργός



Σχήμα 6.8: Τα σημεία του περιγράμματος και οι πίνακες MIN_x και MAX_x στο πλέγμα των επεξεργαστών

επεξεργαστής p_{ij} παίρνει την τιμή της προσωρινής μεταβλητής $temp_x$ από τον προηγούμενό του κατά την y διάσταση επεξεργαστή και έτσι αποκτά την τιμή της συντεταγμένης x του προηγούμενου μη μηδενικού σημείου. Οι τελευταίοι επεξεργαστές που δεν έχουν προηγούμενο θέτουν στη μεταβλητή $temp_x$ την τιμή *border-val* που είναι προκαθορισμένη.

Ομοίως μεταδίδονται και τα στοιχεία του πίνακα MAX_x στις εντολές (5) και (6) , ενώ με τρόπο ανάλογο διαδίδεται και η συντεταγμένη y του κάθε ενεργού επεξεργαστή από τις εντολές (7) και (8).

Στη συνέχεια, κάθε επεξεργαστής, χρησιμοποιώντας τις συντεταγμένες του προηγούμενου σημείου καθώς και τις δικές του, υπολογίζει τη γωνία του ευθύγραμμου τμήματος με άκρα τα σημεία (x,y) και $(temp_x, temp_y)$ ως προς τον άξονα y καλώντας την παράλληλη συνάρτηση **ESTIMATE-ANGLE!!**. Για να αποφασίσει αν βρίσκεται σε κορυφή με κοίλη ή κυρτή γωνία θα πρέπει να διαθέτει την αντίστοιχη γωνία με την επόμενη κορυφή. Η γωνία αυτή έχει υπολογιστεί στον επεξεργαστή που κρατά το επόμενο μη μηδενικό σημείο του ημιπεριγράμματος. Με τις επόμενες δύο εντολές (10) και (11), κάθε επεξεργαστής στέλνει στον προηγούμενό του ενεργό επεξεργαστή αυτή τη γωνία.

Procedure **Graham-scan**

do while ($criterion_{PP}$ = TRUE)

(1) **if!!** (($min\text{-}x > 0$) **or** ($max\text{-}x <$ P+1)) **then** *flag* = TRUE

(2) **if!!** ((*flag* = TRUE) **and** ($x > P/2$)) **then** $flag_{right}$ = TRUE

(3) $temp_x$ = (**scan-grid!!** $min\text{-}x$ **copy!!** :dimension :y :segment-pvar *flag*)

(4) **if!!** (*flag* = TRUE) **then** $prev_x$ = (**news-border!!** $temp_x$ *border-val* 0 -1)

(5) $temp_{max-x}$ = (**scan-grid!!** $max\text{-}x$ **copy!!** :dimension :y :segment-pvar *flag*)

(6) **if!!** ($flag_{right}$ = TRUE) **then** $prev_x$ = (**news-border!!** $temp_{max-x}$ *border-val* 0 -1)

(7) $temp_y$ = (**scan-grid!!** y_{coord} **copy!!** :dimension :y :segment-pvar *flag*)

(8) **if!!** (*flag* = TRUE) **then** $prev_y$ = (**news-border!!** $temp_y$ *border-val* 0 -1)

(9) **ESTIMATE-ANGLE!!**(x,y,$temp_x$,$temp_y$,$angle_{ij}$)

(10) $temp_{angle}$ =

(**scan-grid!!** *angle* **copy!!** :dimension :y :segment-pvar *flag* :direction :backward)

(11) **if!!** (*flag* = TRUE) **then** $next_{angle}$ = (**news-border!!** $temp_{angle}$ *border-val* 0 1)

(12) **if!!** (*angle*< $next_{angle}$) **then** *discarded* = TRUE

(13) **if!!** (*discarded*= TRUE) **then**

$min\text{-}x = 0$; $max\text{-}x$ = P+1

(14) *criterion* = (**scan!!** *discarded* **or!!**)

end while

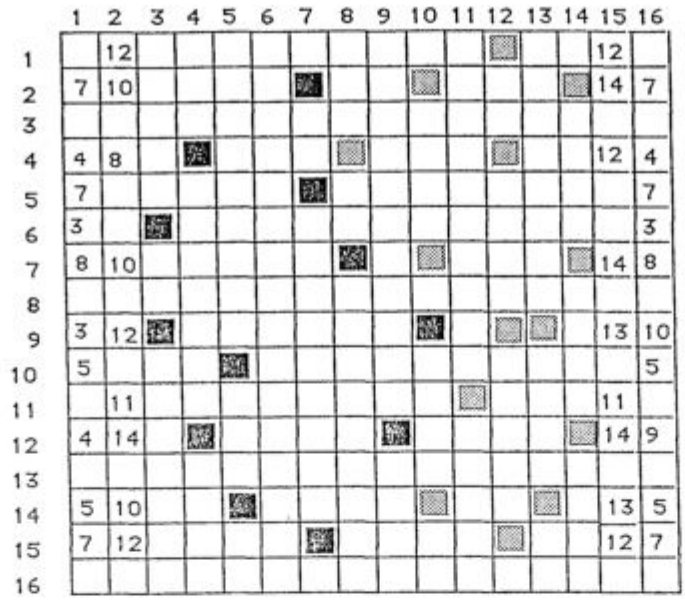
Σχήμα 6.9: Η υλοποίηση του αλγόριθμου *Graham-scan* στο Connection Machine

Ο αλγόριθμος τελειώνει με τρεις εντολές (12-14). Κάθε επεξεργαστής, που μετά τον έλεγχο των γωνιών αποφασίζει πως πρέπει να απαλείψει το σημείο που κρατάει σηκώνει τη σημαία *discarded* και μηδενίζει τις συντεταγμένες του περιγράμματος που κρατούσε ενώ τελικά εκτελείται μία καθολική προθεματική πράξη σάρωσης με τελεστή **or** πάνω στη μεταβλητή *discarded*. Αν το αποτέλεσμα είναι TRUE στον τελευταίο επεξεργαστή p_{PP}, σημαίνει πως τουλάχιστον ένας επεξεργαστής άλλαξε κατάσταση, άρα πρέπει να επαναληφθεί η εκτέλεση της διαδικασίας αυτής, πράγμα που ελέγχει το **do while**. Αν είναι FALSE, σημαίνει πως κανένα σημείο δεν απαλείφθηκε και κατά συνέπεια έχει βρεθεί η τελική μορφή του ΚΠ.

Ο Αλγόριθμος για πολλά αντικείμενα στην ίδια εικόνα

Στον αλγόριθμο που παρουσιάσαμε κάναμε την υπόθεση ότι η εικόνα περιέχει σημεία τα οποία ανήκουν σε ένα αντικείμενο. Ομως δεν είναι πάντα έτσι, γιατί πολύ συχνά στην εικόνα παριστάνονται περισσότερα από ένα αντικείμενα. Θα δείξουμε στη συνέχεια μερικές μετατροπές που χρειάζεται ο αλγόριθμος της προηγούμενης παραγράφου για να βρίσκει το ΚΠ όλων των αντικειμένων που εμφανίζονται στην εικόνα. Δεχόμαστε πως τα στοιχεία (pixels) που ανήκουν σε κάθε αντικείμενο έχουν, αντί για 1, την τιμή της ετικέτας του αντικειμένου και πως η αρίθμηση των αντικειμένων αρχίζει από το 1 και γίνεται κατά συνεχή τρόπο. Αυτή είναι και η έξοδος (output) από τον αλγόριθμο για την εύρεση Συνδεδεμένων Συστατικών που παρουσιάζεται σε επόμενο τμήμα Αν δεν ισχύουν αυτές οι συνθήκες, θα πρέπει να γίνουν ορισμένες προσθήκες στον αλγόριθμο, οι οποίες θα τις εξασφαλίζουν. Η διεργασία αυτή γίνεται με τρόπο απλό.

Για την επίλυση λοιπόν του προβλήματος των πολλών αντικειμένων θα τροποποιήσουμε εν μέρει, μόνο τον αλγόριθμο **Contour** ως εξής: Στην αρχική μορφή του, η πρώτη στήλη των επεξεργαστών κρατούσε τον πίνακα $MIN_x[P]$ και η τελευταία τον $MAX_x[P]$. Είχαμε μόνο ένα ζεύγος από τέτοιους πίνακες που αντιστοιχούσαν στο μοναδικό αντικείμενο της εικόνας. Αν έχουμε όμως πολλά αντικείμενα θα δημιουργηθούν και τα αντίστοιχα ζεύγη πινάκων των ημιπεριγραμμάτων τους. Για L αντικείμενα θα έχουμε L πίνακες $MIN_x[P]$ και L πίνακες $MAX_x[P]$. Δημιουργούμε λοιπόν τα L αυτά ζεύγη πινάκων και τους τοποθετούμε σε διαδοχικές στήλες επεξεργαστών. Έτσι, ο πίνακας $MIN_x[P]^1$ του αντικειμένου 1 θα τοποθετηθεί στην πρώτη στήλη επεξεργαστών και ο $MAX_x[P]^1$ στην P στήλη, ο $MIN_x[P]^2$ του αντικειμένου 2 στη 2 στήλη και ο $MAX_x[P]^2$ στην P-1, ο $MIN_x[P]^3$ του αντικειμένου 3 στην 3 στήλη και ο $MAX_x[P]^3$ στην

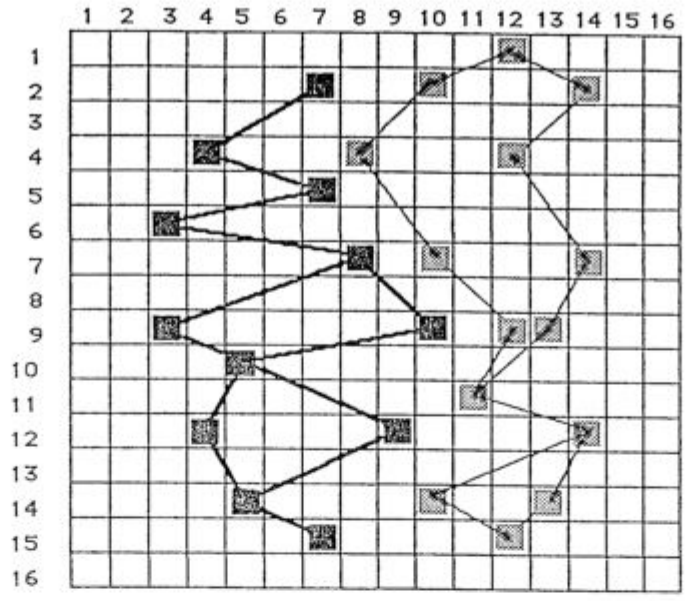


Σχήμα 6.10: Τα περιγράμματα 2 αντικειμένων και οι πίνακες MIN_x και MAX_x πάνω στο πλέγμα

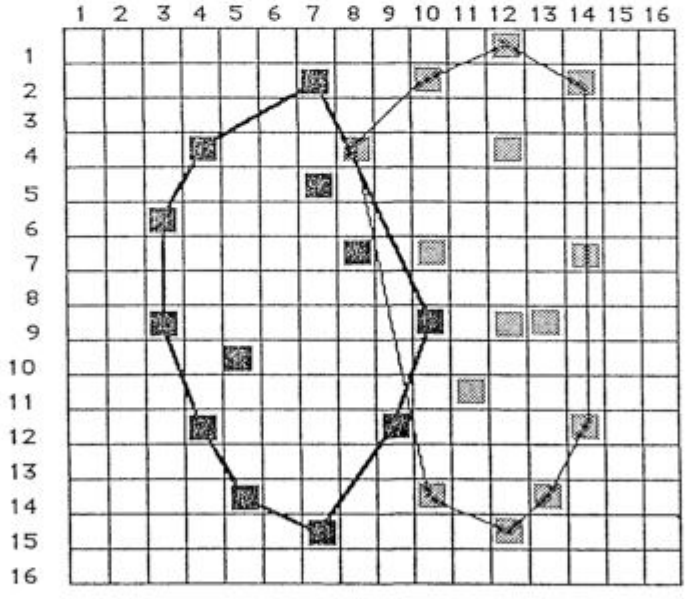
P-2 κ.ό.κ. Στη συνέχεια, παράλληλα σε όλους τους επεξεργαστές θα υλοποιηθεί η διαδικασία **Graham-scan** ακριβώς με τον ίδιο τρόπο που παρουσιάστηκε προηγούμενα.

Είναι φανερό πως η διαδικασία **Contour** θα εφαρμοστεί διαδοχικά για κάθε αντικείμενο. Εισάγεται έτσι μια σειριοποίηση, όπως θα φανεί στην ανάλυση.

Τα επόμενα σχήματα είναι για μια εικόνα 16x16 με δύο αντικείμενα σε ένα ορθογώνιο πλέγμα 16x16 επεξεργαστών. Το Σχήμα 6.10 έχει τα περιγράμματα των 2 αντικειμένων και τις θέσεις που κρατούνται τα στοιχεία των πινάκων $\mathrm{MIN}_x[P]^1$,$\mathrm{MIN}_x[P]^2$ και $\mathrm{MAX}_x[P]^1$ και $\mathrm{MAX}_x[P]^2$. Στα Σχήματα 6.11 και 6.12 βλέπουμε διαγραμμισμένα τα περιγράμματα των δύο αντικειμένων και τα κυρτά τους περιγράμματα, αντίστοιχα.



Σχήμα 6.11: Διαγραμμισμένα τα περιγράμματα των 2 αντικειμένων



Σχήμα 6.12: Τα κυρτά περιγράμματα των 2 αντικειμένων

Procedure **Contour**

(1) **Labels** = ***max** ($pixel_{ij}$) ;
(2) **For** $k := 1$ *to* **Labels**
(3) $tempx_{ij} = 0$
(4) **if!!** ($pixel_{ij}$ = k) **then** $tempx_{ij}$:= x-coordinate
(5) *min-x* = (**scan-grid!!** *tempx* **min!!** :dimension :x :direction :backward)
(6) $tempx_{ij}$ = P + 1
(7) **if!!** ($pixel_{ij}$ = k) **then** $tempx_{ij}$:= x-coordinate
(8) *max-x* = (**scan-grid!!** *tempx* **max!!** :dimension :x)
(9) *position* = *k* - 1;
(10) **if!!** (x_{coord} = *position*) **then** (**news!!** $temp_x$ 0 (1 - *k*))
(11) *position* = P - *k* + 1;
(12) **if!!** (x_{coord} = *position*) **then** (**news!!** $temp_x$ 0 (*k* -1))
(13) **EndFor**

Σχήμα 6.13: Η διεργασία *Contour* στο Connection Machine για πολλά αντικείμενα στην ίδια εικόνα

Στο Σχήμα 6.13 φαίνεται πώς γίνεται η υλοποίηση της διεργασίας **Contour** με χρήση των πράξεων σάρωσης στο Connection Machine.

Στην εντολή (1) χρησιμοποιούμε μια βασική πράξη (primitive) που έχει το σύστημα. Η πράξη συρρίκνωσης (reduction operation) ***max**(*var*) επιστρέφει στον Host επεξεργαστή τη μεγαλύτερη τιμή που έχει η μεταβλητή *var* σε όλους τους επεξεργαστές. Έτσι, η μεταβλητή **Labels** στον Host υπολογιστή, ισούται με το πλήθος των αντικειμένων που εμφανίζονται στην εικόνα. Στη συνέχεια, όπως περιγράψαμε νωρίτερα, βρίσκουμε για κάθε αντικείμενο τα δύο ημιπεριγράμματά του. Η εντολή (9) υπολογίζει τη στήλη στην οποία πρέπει να μεταφερθεί το αριστερό ημιπερίγραμμα του αντικειμένου *k* και η (10) κάνει αυτή τη μεταφορά. Το ίδιο κάνουν οι (11) και (12) για το δεξί ημιπερίγραμμα κάθε αντικειμένου.

Οταν ολοκληρωθεί η ανακύκλωση των εντολών (2) έως (13) έχουν βρεθεί τα περιγράμματα όλων των αντικειμένων, οπότε εφαρμόζεται η διαδικασία **Graham-scan** που βρίσκει παράλληλα τα Κυρτά Περιγράμματά τους.

Ανάλυση του αλγορίθμου

Ο συνολικός χρόνος που απαιτείται για τον υπολογισμό του Κυρτού Περιγράμματος όλων των αντικειμένων στην εικόνα μπορεί να εκφραστεί σαν:

$$\mathrm{T}_{tot}^{CM} = T_{init} + L * \mathrm{T}_{contour} + T_{Graham} \tag{6.1}$$

Η ανάλυση των επιμέρους αυτών παραγόντων έχει ως εξής:

T_{init} είναι ο χρόνος που απαιτείται προκειμένου κάθε επεξεργαστής να δώσει τις αρχικές τιμές σε ορισμένες μεταβλητές που θα του χρειαστούν αργότερα.

$$T_{init} = n_a t_a$$

οπου n_a είναι το πλήθος των αριθμητικών πράξεων που εκτελούνται στη φάση αυτή και t_a ο χρόνος εκτέλεσης κάθε μιας από αυτές.

Ακολουθεί η φάση εύρεσης του περιγράμματος του αντικειμένου, η οποία επαναλαμβάνεται L φορές, δηλαδή όσο είναι το πλήθος των αντικειμένων στην εικόνα. Η διαδικασία αυτή χρειάζεται χρόνο $\mathrm{T}_{contour}$ για κάθε αντικείμενο.

$$\mathrm{T}_{contour} = 2 * t_{scan-x-max} + n_\beta t_a + 2 * t_{msg},$$

όπου $t_{scan-x-max}$ είναι ο χρόνος για την πράξη σάρωσης κατά τη x διάσταση και t_{msg} είναι ο χρόνος για να σταλεί ένα μήνυμα χρησιμοποιώντας τις σχετικές θέσεις των επεξεργαστών πάνω στο ορθογώνιο πλέγμα τους. Το μέγεθος των μεταβλητών που συμμετέχουν στις πράξεις σάρωσης είναι $log\mathrm{P}$ για μια εικόνα PxP.

T_{Graham} είναι ο χρόνος για την υλοποίηση της διαδικασίας **Graham-scan** και αναλύεται ως εξής :

$$T_{Graham} = T_1 + C * (\mathrm{T}_2 + \mathrm{T}_3 + \mathrm{T}_4 + \mathrm{T}_5 + T_6)$$

Κάνουμε την ανάλυση του αλγορίθμου στις έξη φάσεις χωριστά:

T_1 είναι ο χρόνος για ορισμένες αρχικές πράξεις που απαιτούνται, T_2 ο χρόνος για να στείλει κάθε επεξεργαστής τις συντεταγμένες του σημείου που κρατάει στον επόμενο ενεργό επεξεργαστή, T_3 ο χρόνος που χρειάζεται ο κάθε επεξεργαστής να υπολογίσει τη σχετική γωνία της κορυφής που κρατά με την προηγούμενη κορυφή, T_4 ο χρόνος για να στείλει τη γωνία αυτή στον προηγούμενό του ενεργό επεξεργαστή, T_5 ο χρόνος για να συγκρίνει τις δύο γωνίες και να αποφασίσει αν το σημείο που κρατάει πρέπει να απαλειφθεί ή όχι. Τέλος, T_6 είναι ο χρόνος που χρειάζεται το σύστημα να ελέγξει αν στην προηγούμενη σάρωση ακυρώθηκαν κορυφές προκειμένου να αποφασίσουμε αν ολοκληρώθηκε ο αλγόριθμος και έχει βρεθεί το κυρτό περίγραμμα. C (Curvature) είναι το μέγιστο πλήθος σημείων που ανήκουν σε κοίλο τμήμα του περιγράμματος των αντικειμένων.

$$T_1 = n_\gamma t_a$$

$$\mathrm{T}_2 = 3 * t_{scan-y-segm-copy} + 3 * t_{msg-border}$$

$$\mathrm{T}_3 = n_\delta t_a$$

$$\mathrm{T}_4 = n_\varepsilon t_a + (t_{scan-y-segm-copy} + t_{msg-border})$$

$$\mathrm{T}_5 = n_\zeta t_a$$

$$\mathrm{T}_6 = n_\theta t_a + t_{scan-or} + t_{msg}$$

Με μία τμηματική πράξη σάρωσης κάθε επεξεργαστής διαβίβει κατά την y κατεύθυνση τις τιμές x και y, με σκοπό να φτάσουν στον επόμενο επεξεργαστή που κρατά υποψήφια κορυφή. Με την πράξη σάρωσης η τιμή φτάνει μέχρι τον προηγούμενο επεξεργαστή από τον οποίο ο ενδιαφερόμενος την παίρνει εύκολα σε χρόνο t_{msg}. Γενικά $t_{scan-operator}$ είναι ο χρόνος για την εκτέλεση μιας σάρωσης με τον τελεστή *operator*. $t_{msg-border}$ είναι ο χρόνος εκτέλεσης μιας πράξης με σχετικές διευθύνσεις, κατά την οποία οι επεξεργαστές που βρίσκονται στα όρια του πλέγματος αντιμετωπίζονται με ειδικό τρόπο. Τέλος $n_\gamma, n_\delta, n_\varepsilon, n_\zeta$, και n_θ είναι τα πλήθη

των αριθμητικών πράξεων σε κάθε τμήμα $T_1 \cdots T_6$ του αλγορίθμου, για κάθε μία πρ
τις οποίες απαιτείται χρόνος t_a.

Επειδή σε κάθε επανάληψη απαλείφεται ένα σημείο από κάθε κοιλότητα του περιγρ
το πλήθος των επαναλήψεων της Graham-scan εξαρτάται από το μέγιστο πλήθος (C)
που βρίσκονται στην ίδια κοιλότητα πάνω στο περίγραμμα των αντικειμένων.

Με όσα αναφέρθηκαν παραπάνω, γίνεται φανερό πως ο συνολικός χρόνος της πα
λης εκτέλεσης της διαδικασίας αυτής εξαρτάται από το πλήθος (L) των αντικειμένων μό
πρώτη φάση (Contour). Στη φάση αυτή έχουμε μετασχηματισμό της αρχικής δομής δεδ
από το διδιάστατο πίνακα όλης της εικόνας στους γραμμικούς πίνακες (λίστες) των περ
μάτων των αντικειμένων που αναπαριστά. Στη δεύτερη φάση, η επεξεργασία των λιστών
γίνεται παράλληλα (εφόσον $L \leq \frac{P}{2}$ που πρακτικά ικανοποιείται) αλλά εδώ έχουμε εξ
του χρόνου ολοκλήρωσης της διαδικασίας από το μέγιστο βάθος της κοιλότητας (C) που
νίζεται στα αντικείμενα. Ακόμα, από το γεγονός ότι ο χρόνος εκτέλεσης μιας πράξης σά
είναι ανεξάρτητος ουσιαστικά από το πλήθος και το μέγεθος των τμημάτων, στην περ
που έχουμε τμηματική σάρωση, γίνεται φανερό πως ο χρόνος ολοκλήρωσης της διεργασί
εξαρτάται ούτε από το μέγεθος ούτε από το πλήθος και τη θέση των αντικειμένων μέσα
εικόνα.

6.2.3 Παρατηρήσεις επί της υλοποίησης

Οπως αναφέρεται στο παράρτημα, κάθε φυσικός επεξεργαστής μπορεί να προσομε
ένα πλέγμα από ιδεατούς (virtual) επεξεργαστές, έτσι ώστε να αυξηθεί το μέγεθος του
λικού πλέγματος στο οποίο θα γίνει η επεξεργασία. Το ιδεατό αυτό πλέγμα δεν είναι π
συμμετρικό δεδομένου ότι και το φυσικό πλέγμα δεν είναι πάντα συμμετρικό. Για παράδε
αν έχουμε 8K επεξεργαστές, αυτοί διατάσσονται σε ένα φυσικό πλέγμα 64x128. Δηλαδ
γραμμές και 128 στήλες. Αν θέλουμε να δημιουργήσουμε τελικά ένα πλέγμα για μι
κόνα 128x128, κάθε επεξεργαστής θα πρέπει να προσομειώσει ένα ιδεατό πλέγμα 2x1 α
επεξεργαστές (ιδεατούς επίσης).

Αν εκτελέσουμε μια πράξη σάρωσης $scan^{x}_{max}$ κατά τη x κατεύθυνση, επειδή ουσιαο

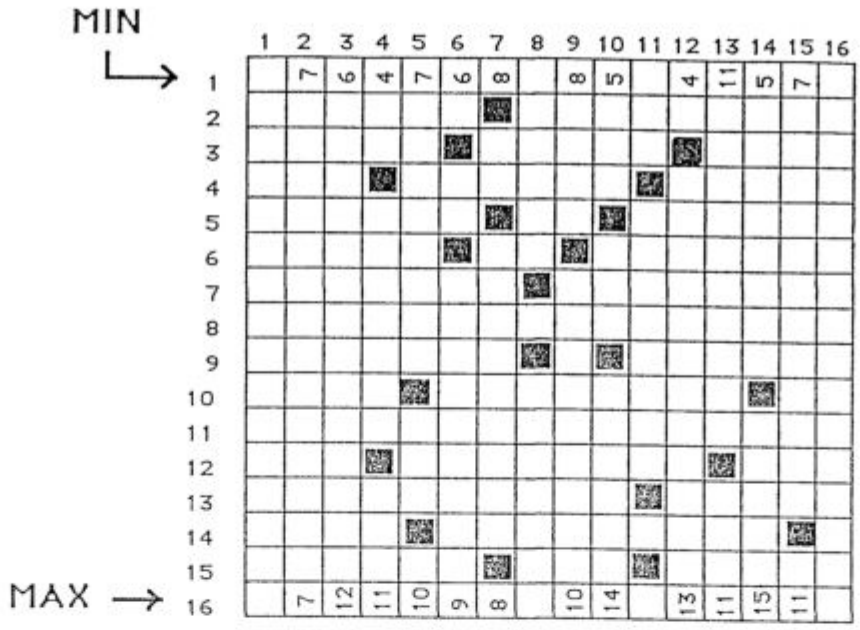
κάθε επεξεργαστής προσομειώνει 2 επεξεργαστές κατά αυτή την κατεύθυνση, θα γίνουν 2 αντί μιας πράξεις σάρωσης. Αν εκτελέσουμε μία $scan^{y}_{max}$, κάθε επεξεργαστής θα πρέπει να εκτελέσει την πράξη *max* στις 2 τιμές των 2 ιδεατών επεξεργαστών που κρατάει κατά την y κατεύθυνση και πάνω στο αποτέλεσμα να εφαρμοστεί η πράξη $scan^{y}_{max}$ του συστήματος. Παρουσιάζει δηλαδή η ίδια πράξη μια ανισοτροπία ως προς την κατεύθυνση που εκτελείται. Αυτό φαίνεται και από τις μετρήσεις των επιδόσεων του CM, στο Παράρτημα A.2.

Προκειμένου να εκμεταλλευτούμε τη διαφορά στην ταχύτητα της εκτέλεσης της σάρωσης κατά τις δύο διευθύνσεις, μπορούμε να τροποποιήσουμε λίγο την υλοποίηση του αλγορίθμου μας, έτσι ώστε να ανάγουμε τα περισσότερα $scan^{y}_{max}$ σε $scan^{x}_{max}$, που έχουν μικρότερο χρόνο εκτέλεσης, επιτυγχάνοντας σημαντική βελτίωση της απόδοσης του αλγορίθμου.

Η εύρεση του περιγράμματος των αντικειμένων χρησιμοποιεί σαρώσεις μόνο κατά τη x κατεύθυνση, ενώ η διαδικασία Graham χρησιμοποιεί σαρώσεις μόνο κατά την y. Αυτό που μπορούμε να κάνουμε ως εκ τούτου, είναι να τοποθετήσουμε τα σημεία των περιγραμμάτων των αντικειμένων στις γραμμές, αντί για τις στήλες, του πλέγματος. Η μετατροπή αυτή της διάταξης γίνεται με ελάχιστο κόστος. Τα ημιπεριγράμματα κάθε αντικειμένου, όπως περιγράφτηκε στην παράγραφο 6.2.2 βρίσκονται αρχικά στις στήλες 1 και P, και από κει μεταφέρονται παράλληλα στις στήλες L και P-L. (L ο αριθμός του αντικειμένου). Στο σημείο αυτό, αντί να μετατεθούν λοιπόν στις στήλες L και P-L μπορούν να μεταφερθούν αντίστοιχα στις γραμμές L και P-L (Σχ. 6.14). Το επιπλέον κόστος επικοινωνίας που απαιτείται είναι πολύ μικρό και επιπλέον γίνεται μία μόνο φορά. Στη συνέχεια ο αλγόριθμος υλοποιείται όπως περιγράφτηκε ήδη, αλλά με τη διαφορά ότι η διαδικασία Graham-scan εφαρμόζεται στις γραμμές αντί για τις στήλες.

Μια επόμενη σημαντική παρατήρηση που κάνει τον αλγόριθμο αποτελεσματικότερο είναι ότι, εκτός του ότι δεν επηρεάζεται η απόδοσή του πρακτικά από το πλήθος των αντικειμένων που εμφανίζονται στην εικόνα, δεν επηρεάζεται καθόλου και από τη σχετική θέση τους. Έτσι, δύο ή περισσότερα αντικείμενα μπορεί να είναι αλληλοεπικαλυπτόμενα ή το ένα να περικλείει το άλλο. Η απόδοση του αλγορίθμου είναι ακριβώς η ίδια. Στην περίπτωση που έχουμε πλήθος αντικειμένων $L > \frac{P}{2}$ χωρίζουμε τα αντικείμενα σε ομάδες των $\frac{P}{2}$ και στη συνέχεια εφαρμόζεται ο αλγόριθμος εν σειρά σ'αυτές με τον ίδιο τρόπο.

Η παράλληλη μορφή του αλγορίθμου, όσον αφορά τόσο στην υλοποίηση όσο και στην ανά-

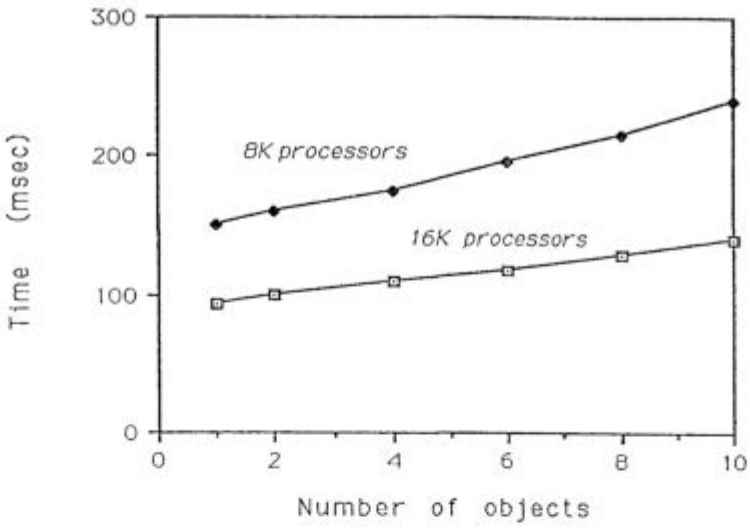


Σχήμα 6.14: Το περίγραμμα, με τους πίνακες MIN_x και MAX_x σε γραμμές του πλέγματος των επεξεργαστών

λυσή του, απλοποιείται πάρα πολύ, επειδή τα επικοινωνιακά σχήματα που απαιτεί βρίσκουν πολύ καλή ταύτιση στους γράφους επικοινωνίας της συγκεκριμένης αρχιτεκτονικής και στους μηχανισμούς που αυτή διαθέτει. Έτσι, τα πειραματικά αποτελέσματα προσεγγίζουν τους αναλυτικούς χρόνους πολύ καλά.

Οσον αφορά το χρόνο που χρειάζεται στο ξεκίνημά του κάθε αλγόριθμος, αυτός δεν είναι σταθερός και προφανώς εξαρτάται από την κατάσταση της μηχανής εκείνη τη στιγμή.

Το φόρτωμα των δεδομένων δεν απαιτεί ιδιαίτερα πολύ χρόνο. Αν συγκριθεί μάλιστα με το χρόνο ολοκλήρωσης των διεργασιών είναι ένα μικρό ποσοστό. Η ταχύτητα επικοινωνίας του κεντρικού υπολογιστή (Host) με τους κομβικούς επεξεργαστές είναι $5x10^8$ bits/sec [79] που σημαίνει ότι για το φόρτωμα μιας εικόνας 1024x1024 bytes χρειάζεται 17 msec, για 512x512 4 msec και για 256x256 2 msec. Αν οι εικόνες είναι δυαδικές και κάθε σημείο αντιπροσωπεύεται από ένα bit, τότε αντίστοιχα χρειάζονται 2.5 msec, 0.5 msec και 0.1 msec.



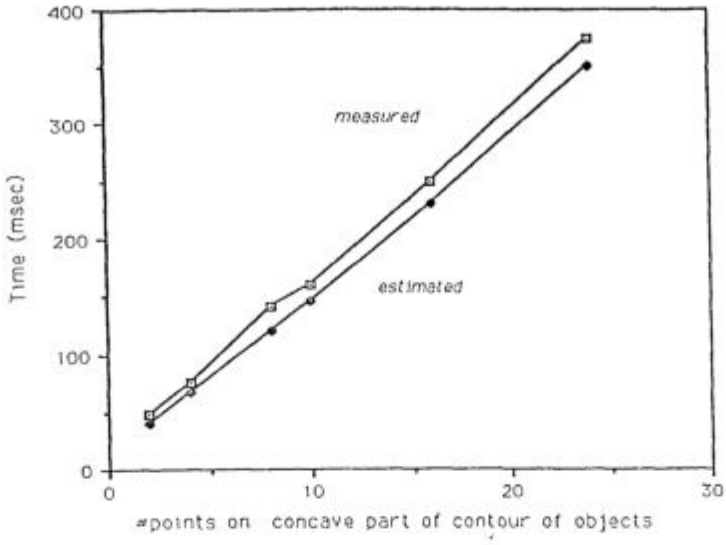
Σχήμα 6.15: ΚΠ:Μετρημένος χρόνος σε 8K και16K επεξεργαστές συναρτήσει του πλήθους των αντικειμένων

6.2.4 Πειραματικά αποτελέσματα

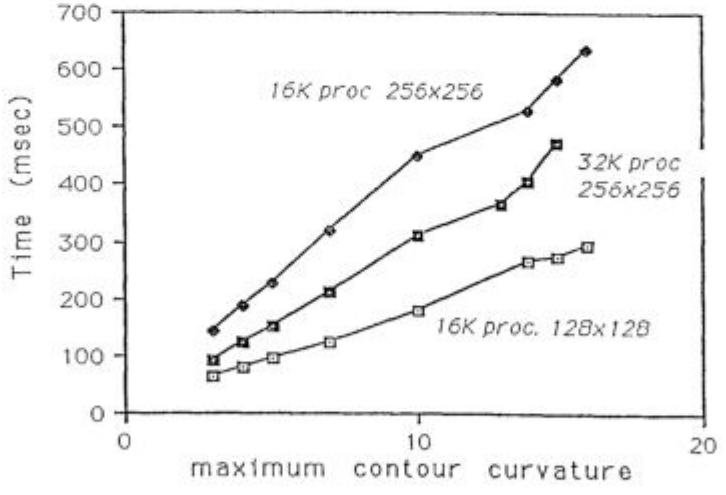
Στο Σχήμα 6.15 βλέπουμε τον πραγματικό χρόνο, που μετρήθηκε στο σύστημα με 8K και 16K φυσικούς επεξεργαστές, συναρτήσει του πλήθους των αντικειμένων που εμφανίζονται σε μια εικόνα 128x128.

Το επόμενο Σχ. 6.16 δείχνει τον ανεμενόμενο και μετρημένο χρόνο στο σύστημα με 16K επεξεργαστές, σε μια εικόνα 128x128, συναρτήσει της μέγιστης κοιλότητας που εμφανίζει το περίγραμμα των αντικειμένων. Το μέγεθος μιάς κοιλότητας εκφράζεται από το πλήθος των σημείων που τη δημιουργούν. Το πλήθος αυτών των σημείων επηρεάζει το χρόνο εκτέλεσης αφού σε κάθε επανάληψη εξαλείφεται ένα μόνο τέτοιο σημείο από κάθε κοιλότητα συγχρόνως. Ετσι μας ενδιαφέρει το μέγιστο πλήθος σημείων σε μιά οποιαδήποτε από όλες τις κοιλότητες όλων των αντικειμένων που εμφανίζονται στην εικόνα.

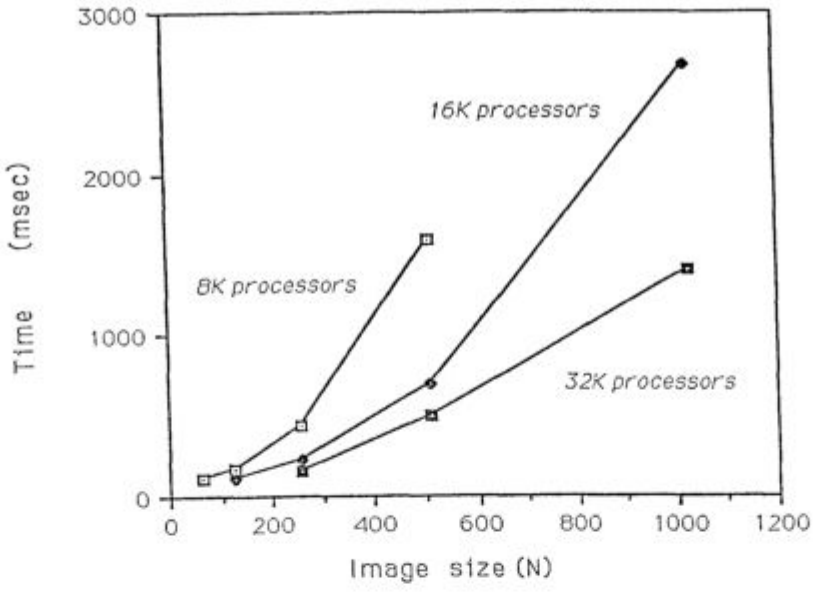
Στο Σχήμα 6.17 φαίνονται οι μετρημένοι χρόνοι για πλέγματα 128 και 256 σε 16K και 32K φυσικούς επεξεργαστές.



Σχήμα 6.16: ΚΠ:Μετρημένος χρόνος σε 16Κ επεξεργαστές συναρτήσει της κοιλότητας των αντικειμένων



Σχήμα 6.17: Υπολογισμός Κυρτού Περιγράμματος σε 16Κ και 32Κ επεξεργαστές



Σχήμα 6.18: Υπολογισμός ΚΠ σε εικόνες με 6 αντικείμενα μέγιστης κοιλότητας 4 συναρτήσει του μεγέθους της εικόνας

Στο Σχήμα 6.18 φαίνονται οι χρόνοι για την εύρεση των περιγραμμάτων σε εικόνες με 6 αντικείμενα και με μέγιστη κοιλότητα 4, συναρτήσει του μεγέθους της εικόνας. Υπάρχουν τρεις καμπύλες που αντιστοιχούν σε συστήματα με 8K, 16K και 32K επεξεργαστές. Παρατηρούμε πως ο χρόνος αυξάνει σημαντικά όσο μεταβαίνουμε σε μεγαλύτερα ιδεατά πλέγματα.

6.2.5 Παράλληλη υλοποίηση με πολλές μονάδες δεδομένων ανά επεξεργαστή

Στην περίπτωση αυτή χρησιμοποιήθηκε η αρχιτεκτονική iPSC/2. Η παράλληλη υλοποίησή του σειριακού αλγορίθμου του τμήματος 6.2.1, βασίζεται στην αρχή "Διαίρει και Βασίλευε" (devide and conquer).

Κάθε επεξεργαστής από τους E συνολικά, αναλαμβάνει ένα μέρος από τα αρχικά δεδομένα, βρίσκει το περίγραμμά τους (contour) και στη συνέχεια, ανά δύο οι επεξεργαστές συνθέτουν αυτά τα επιμέρους περιγράμματα ακολουθώντας ένα δενδροειδές σχήμα επικοινωνίας για να καταλήξουν μετά από $\log E$ βήματα στο συνολικό περίγραμμα το οποίο κρατάει

ένας επεξεργαστής. Ο τελευταίος αυτός εφαρμόζει τον Αλγόριθμο Graham-Scan και βρίσκει το τελικό Κυρτό Περίγραμμα. Ο αλγόριθμος αυτός έχει παρόμοια φιλοσοφία με εκείνον που εμφανίζεται στην [172]. Ο αλγόριθμος εκείνος παρουσιάζεται στη σειριακή μορφή του και σε κάθε υποπεριοχή της εικόνας δημιουργεί το επιμέρους Κυρτό Περίγραμμα σε αντίθεση με τον αλγόριθμο που παρουσιάζουμε εδώ και όπου δημιουργούμε τα απλά (όχι κατ'ανάγκη κυρτά) περιγράμματα. Η ίδια διαφορά ισχύει σε όλη τη διαδικασία της σύνθεσης των επιμέρους τμημάτων. Υλοποιήθηκαν και οι δύο αλγόριθμοι. Τα αποτελέσματα με την πρώτη μορφή του αλγορίθμου ήταν λιγότερο καλά και δεν παρουσιάζονται εδώ.

Τον αλγόριθμο αυτό θα τον δώσουμε σε δύο μορφές, οι οποίες διαφέρουν μόνο στη δομή των δεδομένων εισόδου, που χρησιμοποιούν. Ο πρώτος (Αλγόριθμος Α) χρησιμοποιεί μία λίστα εκείνων των σημείων μόνο που έχουν μη μηδενική τιμή. Τα σημεία αυτά μπορεί να έχουν βρεθεί από προηγούμενη διαδικασία. Κάθε επεξεργαστής έχει τότε μια υπολίστα της συνολικής λίστας. Στο δεύτερο (Αλγόριθμος Β) δίνεται σε κάθε επεξεργαστή ένας υποπίνακας της συνολικής εικόνας.

Η διαφορά στις δύο περιπτώσεις είναι στο πρώτο στάδιο του αλγορίθμου, δηλαδή στην εύρεση του περιγράμματος των αντικειμένων σε κάθε υποεικόνα. Το δεύτερο στάδιο της σύνθεσης των επιμέρους περιγραμμάτων και το τρίτο της εύρεσης του ΚΠ δεν παρουσιάζουν καμιά διαφοροποίηση.

Στις επόμενες παραγράφους θα παρουσιάσουμε τις υλοποιήσεις των δύο αυτών αλγορίθμων.

Αλγόριθμος Α - Δομή Δεδομένων εισόδου: Λίστα

Ο κάθε κομβικός επεξεργαστής έχει μια λίστα *Pixel[M]* με τις συντεταγμένες x και y ενός μέρους των σημείων του αντικειμένου. Για να βρει το περίγραμμά τους εφαρμόζει τον αλγόριθμο Contour του Σχ. 6.19.

Ο αλγόριθμος αυτός αρχικά ταξινομεί τα σημεία που διαθέτει ως προς τη y συντεταγμένη τους και στη συνέχεια δημιουργεί τους δύο πίνακες $\mathrm{MIN}[i]_x$ και $\mathrm{MAX}[i]_x$, κρατώντας για κάθε τιμή της διάστασης y τη μικρότερη και μεγαλύτερη τιμή της διάστασης x.

```
        Procedure Contour
(1)     Sort (pixel[M] over y-coordinate)
(2)         index_cont = 1
            index = 1
(3)         MIN[ index_cont] = pixel[1]
(4)         MAX[ index_cont] = pixel[1]
(5)     Do While (not-end-of-list)
(6)         if (pixel[index].y ≠ 0)
(7)             if (pixel[index].y > MIN[ index_cont].y) then
(8)                 index_cont = index_cont + 1
(9)                 MIN[ index_cont] = pixel[index]
(10)                MAX[ index_cont] = pixel[index]
                    else
(11)                    if (pixel[index].x <MIN[index_cont].x) then
(12)                        MIN[ index_cont] = pixel[index]
(13)                    else
(14)                        if (pixel[index].x >MAX[index_cont].x) then
(15)                            MAX[ index_cont] = pixel[index]
(16)                    endif
(17)            index = index + 1
(18)            endif
(19)        endif
(20)    End While
```

Σχήμα 6.19: *Contour* στο iPSC/2: Αλγόριθμος Α

Το επόμενο στάδιο είναι εκείνο της σύνθεσης των επιμέρους περιγραμμάτων, *Compose-Partial-Contours*, που φαίνεται στο Σx. 6.20. Το στάδιο αυτό επαναλαμβάνεται $\log E$ φορές, όπου E το πλήθος των επεξεργαστών στους οποίους είναι κατανεμημένη η εικόνα. Υποθέτουμε ότι $(b_{\log E-1}, b_{\log E-2}, \ldots, b_1, b_0)$ είναι τα $\log E$ bits της δυαδικής μορφής της ταυτότητας του κάθε επεξεργαστή και $\bar{b_i}$ είναι το $not(b_i)$.

Η διαδικασία **Merge**, που χρησιμοποιείται στο Σx. 6.20, συνθέτει δύο ημιπεριγράμματα δίνοντας ένα νέο ημιπερίγραμμα που συμπεριλαμβάνει τα δύο προηγούμενα. Δηλαδή, ελέγχει και απαλείφει τα σημεία των δύο ημιπεριγραμμάτων που ανήκουν στην ίδια γραμμή, κρατώντας μόνο τα ακραία.

Ανάλυση του Αλγορίθμου Α

Αν N είναι τα σημεία της (PxP) εικόνας και E οι επεξεργαστές, τότε ο χρόνος που απαιτεί η ταξινόμηση των $\frac{N}{\mathrm{E}}$ σημείων που έχει κάθε επεξεργαστής είναι:

$$\mathrm{T}_1 = \frac{N}{\mathrm{E}} \log \frac{N}{\mathrm{E}} t_{comp}$$

όπου t_{comp} ο χρόνος που απαιτεί κάθε σύγκριση και αντικατάσταση στοιχείων.

Το επόμενο στάδιο είναι εκείνο της ενδοεπικοινωνίας. Αρχικά, κάθε επεξεργαστής, αφού βρει το περίγραμμα των σημείων που κρατάει, αποφασίζει ποιο από τα δύο περιγράμματα θα κρατήσει ενώ στέλνει το άλλο στο γειτονικό του. Οι πρώτοι επεξεργαστές αρχικά διέθεταν $\frac{N}{\mathrm{E}}$ σημεία ο καθένας. Ομως μετά την επεξεργασία που θα κάνουν σ'αυτά δεχόμαστε πως θα μειωθούν με ένα συντελεστή $\mathrm{a}_0 < 1$ και συνεπώς κάθε ημιπερίγραμμα που θα πρέπει να σταλεί θα περιέχει κατά μέσο όρο τις συντεταγμένες $\frac{1}{2}\frac{N}{\mathrm{E}}\mathrm{a}_0$ σημείων δηλαδή θα έχει μέγεθος $4\frac{1}{2}\frac{N}{\mathrm{E}}\mathrm{a}_0$ bytes υποθέτοντας 2 bytes για κάθε συντεταγμένη. Ο χρόνος που απαιτεί ένα τέτοιο μήνυμα είναι

$$\mathrm{T}_2^0 = (t_{startup} + 4\frac{1}{2}\frac{N}{\mathrm{E}}\mathrm{a}_0 t_b) = (t_{startup} + 2\frac{N}{\mathrm{E}}\mathrm{a}_0 t_b)$$

όπου $t_{startup}$ είναι ο χρόνος προετοιμασίας και εκκίνησης του μηνύματος, και t_b ο χρόνος μεταφοράς ενός byte από έναν επεξεργαστή στο γειτονικό του. Είναι προφανές ότι στα πρώτα βήματα το a_i θα είναι μικρό ενώ θα τείνει προς τη μονάδα καθώς θα αυξάνει το i.

Στα επόμενα βήματα, κάθε επεξεργαστής που θα τελειώνει τη διαδικασία εύρεσης του

Procedure **Compose-Partial-Contours**

(1) $receiver = (b_{\log E-1}, b_{\log E-2}, \ldots, b_1, \bar{b_0})$

(2) **if** $(b_0 = 0)$ **then**

(3) **send** (*Left-partial-Contour,receiver*)

(4) **receive** (*New-partial-Right-Contour*)

(5) **Merge** (*Right-partial-Contour,New-partial-Right-Contour*)

end if

(6) **if** $(b_0 = 1)$ **then**

(7) **send** (*Right-partial-Contour,receiver*)

(8) **receive** (*New-partial-Left-Contour*)

(9) **Merge** (*Left-partial-Contour,New-partial-Left-Contour*)

end if

(10) **For** i := 0 to $\log E$ - 1

(11) $receiver = (b_{\log E-1}, b_{\log E-2}, \ldots, \bar{b_i}, \ldots, b_1, b_0)$

(12) **if** $((b_i = 1) \wedge (b_{i+1} = 0))$ **then**

(13) **if** $(b_0 = 1)$ **then**

(14) **send** (*Right-partial-Contour,receiver*)

(15) **receive**(*New-partial-Left-Contour*)

(16) **Merge** (*Left-partial-Contour,New-partial-Left-Contour*)

(17) **endif**

(18) **if** $(b_0 = 0)$ **then**

(19) **send** (*Left-partial-Contour,receiver*)

(20) **receive**(*New-partial-Right-Contour*)

(21) **Merge** (*Right-partial-Contour,New-partial-Right-Contour*)

(22) **endif**

(23) **endif**

(24) **EndFor**

Σχήμα 6.20: Ο αλγόριθμος *Compose-Partial-Contours* στο iPSC/2

περιγράμματος, θα στέλνει στον επόμενό του τις συντεταγμένες των σημείων που κρατά. Αυτό επαναλαμβάνεται $\log \mathrm{E}$ φορές έως ότου φτάσουν στον τελευταίο επεξεργαστή όλα τα υπολογισμένα σημεία. Στο βήμα i κάθε επεξεργαστής που έχει σειρά να στείλει στον επόμενο του θα στείλει τις συντεταγμένες $(\frac{1}{2}2^i\frac{N}{\mathrm{E}}\prod_{k=1}^{i}a_k)$ σημείων δηλαδή $(4\mathrm{x}2^{i-1}\frac{N}{\mathrm{E}}\prod_{k=1}^{i}a_k)$ bytes, όπου a_k είναι ένας συντελεστής (όπως ο a_0), που αναπαριστά τη μέση μείωση του συνολικού πλήθους σημείων του περιγράμματος μετά την k-1 σύνθεση των επιμέρους περιγραμμάτων. Ο χρόνος επικοινωνίας που θα απαιτήσει συνολικά το βήμα αυτό είναι:

$$T_2 = (t_{startup} + 2\frac{N}{\mathrm{E}}a_0 t_b) + \sum_{i=1}^{\log \mathrm{E}-1}(t_{startup} + 2^{i+1}\frac{N}{\mathrm{E}}\prod_{k=1}^{i}a_k t_b)$$

Αν υποθέσουμε ότι $a_i = \bar{a}$ για όλα τα $i = 0 \cdots \log \mathrm{E} - 1$, η προηγούμενη σχέση γίνεται:

$$T_2 = \log \mathrm{E} * t_{startup} + 2t_b\frac{N}{E}\sum_{i=0}^{\log \mathrm{E}-1}2^i\bar{a}^i$$

Το επόμενο στάδιο είναι εκείνο της σύνδεσης της λίστας που δέχεται ο κάθε επεξεργαστής με τη δική του. Αν t_{merge} είναι ο χρόνος για την επεξεργασία ενός στοιχείου κάθε λίστας, ο συνολικός χρόνος αυτού του στάδιου είναι:

$$T_3 = \sum_{i=1}^{\log \mathrm{E}}\frac{N}{E}2^i\bar{a}^i t_{merge}$$

Αφού ολοκληρωθεί η μεταφορά των ημιπεριγραμμάτων στους δύο τελευταίους επεξεργαστές, κάθε ένας θα εφαρμόσει πάνω σ'αυτό που κρατά τη διαδικασία Graham-scan.

Ο χρόνος για το τελευταίο αυτό στάδιο θα είναι ο μεγαλύτερος των χρόνων που θα απαιτηθούν για την εύρεση του αριστερού και του δεξιού κυρτού ημιπεριγράμματος, αφού τα δύο δημιουργούνται παράλληλα. Ετσι:

$$T_4^{left} = (\frac{1}{2}\frac{N}{E}\bar{a}^{\log \mathrm{E}} + \sum_{i=1}^{left-concaveparts} c_i)t_{Graham}$$

και

$$T_4^{right} = (\frac{1}{2}\frac{N}{E}\bar{a}^{\log E} + \sum_{i=1}^{rigth-concaveparts} c_i)t_{Graham}$$

$$T_4 = \max(T_4^{left}, T_4^{right})$$

Το c_i είναι το πλήθος των σημείων που ανήκουν στην κοιλότητα i του περιγράμματος. Τα σημεία αυτά προκαλούν παλινδρόμηση στη λειτουργία του αλγορίθμου με συνέπεια την αντίστοιχη επιβάρυνση στο χρόνο εκτέλεσης.

Ο συνολικός χρόνος εκτέλεσης είναι λοιπόν:

$$T_{tot}^A = T_1 + T_2 + T_3 + T_4 = \tag{6.2}$$

$$\frac{N}{E}\log\frac{N}{E}t_{comp} + \log E * t_{startup} + 2t_b\frac{N}{E}\sum_{i=0}^{\log E-1} 2^i\bar{a}^i + \sum_{i=1}^{\log E-1}\frac{N}{E}\bar{a}^i t_{merge} + \max(T_4^{left}, T_4^{right})$$

Βλέπουμε λοιπόν, ότι η απόδοση του αλγορίθμου εκτός από τα υπολογιστικά χαρακτηριστικά των επεξεργαστών και του δικτύου, εξαρτάται και από το περιεχόμενο της εικόνας. Συγκεκριμένα από το πλήθος N των σημείων, καθώς και από το πόσο ομοιόμορφα είναι κατανεμημένα στο συνολικό εμβαδόν του αντικειμένου $\bar{a}$, πόσα σημεία βρίσκονται πάνω στο ΚΠ και ακόμα από το άθροισμα όλων των σημείων που ανήκουν σε κοίλα τμήματα του περιγράμματος ($\sum_{allconcaveparts} c_{concave}$).

Αλγόριθμος Β - δομή δεδομένων εισόδου: Πίνακας

Ο αλγόριθμος αυτός διαφέρει από τον προηγούμενο μόνο στο πρώτο στάδιο, κατά το οποίο δημιουργούνται τα περιγράμματα από τα δεδομένα του κάθε επεξεργαστή, και συγκεκριμένα ως προς τη δομή των δεδομένων εισόδου που χρησιμοποιεί.

Κάθε επεξεργαστής έχει στη διάθεσή του έναν πίνακα MxM, υποπίνακα της συνολικής εικόνας. Πάνω σ'αυτόν εφαρμόζει τον αλγόριθμο της παραγράφου 6.2.1 και υπολογίζει το

τοπικό περίγραμμα των σημείων που βρίσκονται σ'αυτή την υποπεριοχή της εικόνας. Ο αλγόριθμος αυτός έχει πολυπλοκότητα O(MxM) αφού αυτό που κάνει είναι μια σάρωση χωρίς παλινδρομήσεις σε όλα τα στοιχεία του πίνακα.

Ανάλυση του Αλγορίθμου B

Η ανάλυση των χρόνων του αλγόριθμου αυτού είναι παρόμοια με εκείνη του προηγούμενου αλγόριθμου. Διαφέρουν μόνο στο πρώτο στάδιο, αφού έχουμε διαφορετική δομή των δεδομένων εισόδου. Ο χρόνος για το πρώτο στάδιο γίνεται:

$$T_1 = \frac{P\mathrm{x}P}{\mathrm{E}} t_{compare}$$

όπου αυτή τη φορά $t_{compare}$ είναι ο χρόνος για να συγκρίνει τη x συντεταγμένη κάθε μη μηδενικού σημείου στον πίνακα, με την αντίστοιχη τιμή του MIN[i] και MAX[i]. Ο χρόνος $t_{compare}$ είναι προφανώς κατά πολύ μικρότερος του t_{comp} του αλγόριθμου A. Εφαρμόζεται όμως σε πολύ περισσότερα σημεία. Ο συνολικός χρόνος εκτέλεσης του αλγόριθμου αυτού γίνεται:

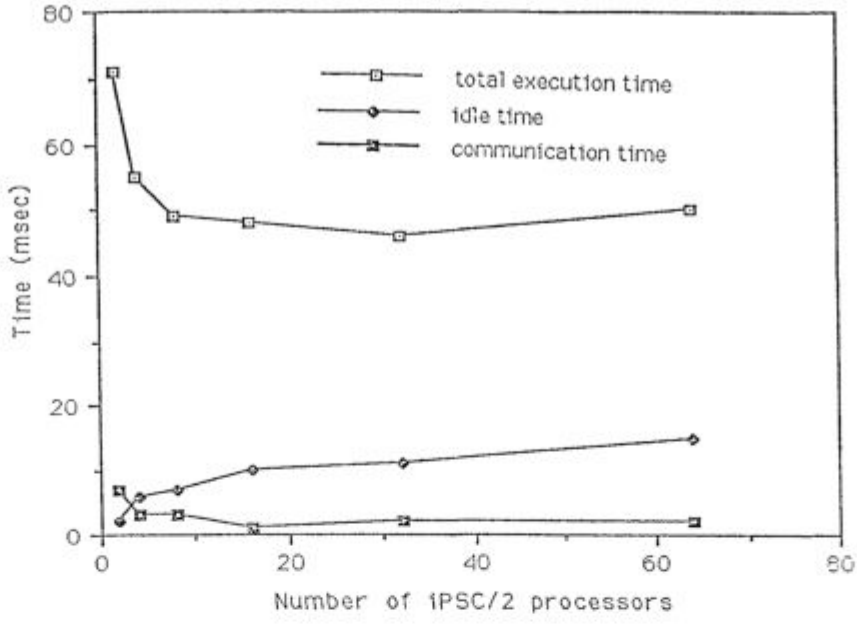
$$T_{tot}^{B} = \frac{M\mathrm{x}M}{\mathrm{E}} t_{compare} + \log \mathrm{E} * t_{startup} + 2t_b \frac{N}{E} \sum_{i=0}^{\log \mathrm{E}-1} 2^i \bar{\mathrm{a}}^i +$$

$$\sum_{i=1}^{\log \mathrm{E}-1} \frac{N}{E} \bar{\mathrm{a}}^i t_{merge} + \max(\mathrm{T}_4^{left}, T_4^{right}) \qquad (6.3)$$

Η εξάρτηση του αλγορίθμου από τα χαρακτηριστικά της αρχιτεκτονικής είναι όμοια με εκείνη του αλγορίθμου A. Οσον αφορά την εξάρτηση από την εικόνα ισχύει ό,τι για τον αλγόριθμο A με μόνη διαφορά ότι αντί του πλήθους των σημείων, εξαρτάται από το μέγεθος της εικόνας.

6.2.6 Πειραματικά αποτελέσματα (iPSC/2)

Στο Σχήμα 6.21 βλέπουμε τους μετρημένους πειραματικά συνολικό και επιμέρους χρόνους εκτέλεσης του αλγορίθμου A σε μία λίστα με 1024 σημεία, συναρτήσει του πλήθους των



Σχήμα 6.21: iPSC/2: Αλγόριθμος Α, χρόνος για 1024 σημεία ως προς το πλήθος των επεξεργαστών

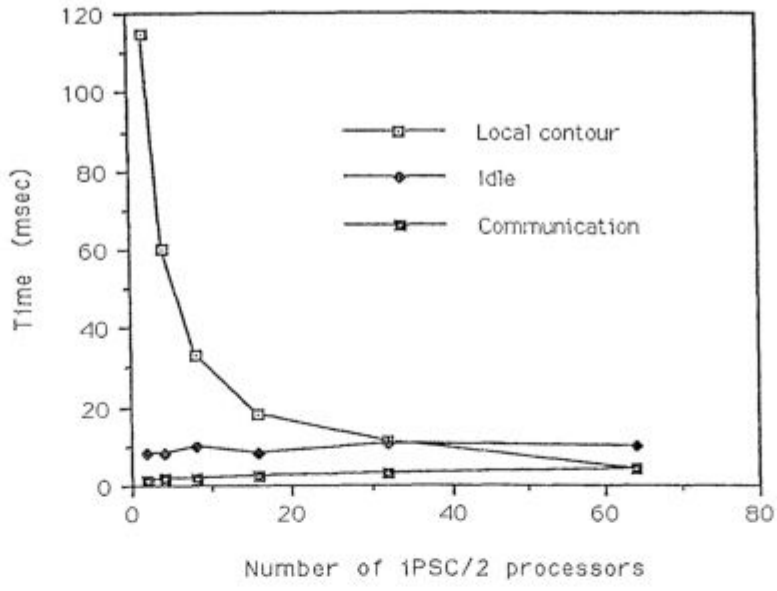
επεξεργαστών.

Στο Σχ. 6.22 η πρώτη καμπύλη (local contours) δείχνει το χρόνο που απαιτείται για τον υπολογισμό των τοπικών περιγραμμάτων από κάθε επεξεργαστή όταν ο συνολικός αριθμός σημείων είναι 3350. Η δεύτερη καμπύλη δείχνει το μέγιστο χρόνο που παραμένει αδρανής ένας επεξεργαστής, ενώ η τρίτη καμπύλη δείχνει τον μέγιστο χρόνο επικοινωνίας για ένα επεξεργαστή. Οι χρόνοι αυτοί δίνονται συναρτήσει του πλήθους των επεξεργαστών.

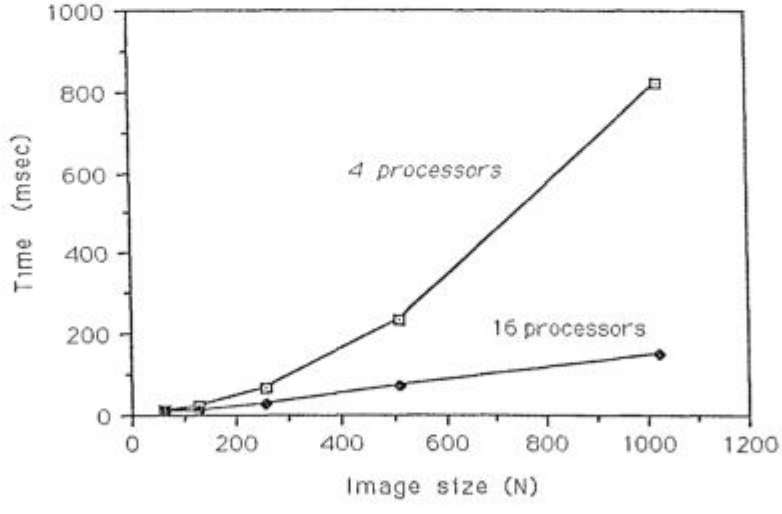
Ακολουθεί το Σχ. 6.23 με τους χρόνους επεξεργασίας της ίδιας εικόνας με τον αλγόριθμο Β, με 4 και 16 επεξεργαστές, συναρτήσει του μεγέθους της εικόνας. Η ανάλυση των χρόνων του Σχ. αυτού, για 4 επεξεργαστές, φαίνεται στο Σχ. 6.24.

Η σύγκριση των χρόνων των δύο αλγορίθμων για εικόνα 128x128 συναρτήσει του πλήθους των επεξεργαστών δίνεται στο επόμενο Σχ. 6.25 .

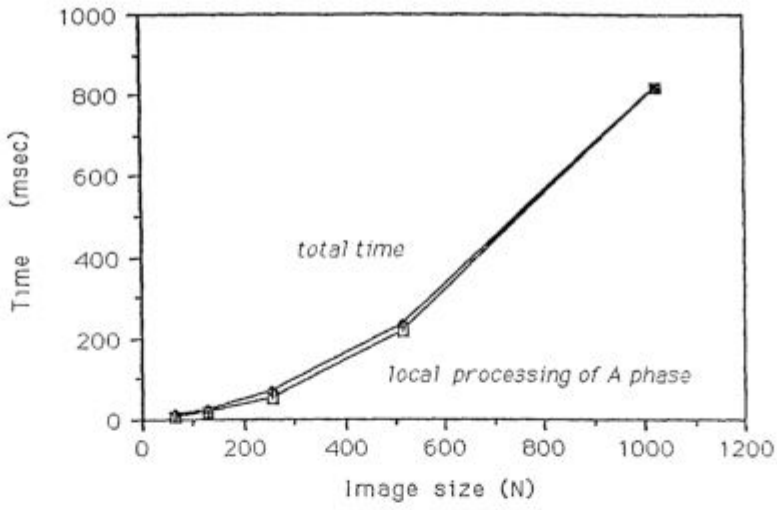
Στο Σχήμα 6.26 βλέπουμε τις επιδόσεις των δύο αλγορίθμων για μια εικόνα 256x256, με 3350 σημεία του αντικειμένου, ως προς το πλήθος των επεξεργαστών. Λόγω του μεγάλου πλήθους των σημείων, βλέπουμε την υπεροχή του δεύτερου αλγόριθμου η οποία όμως μειώνεται



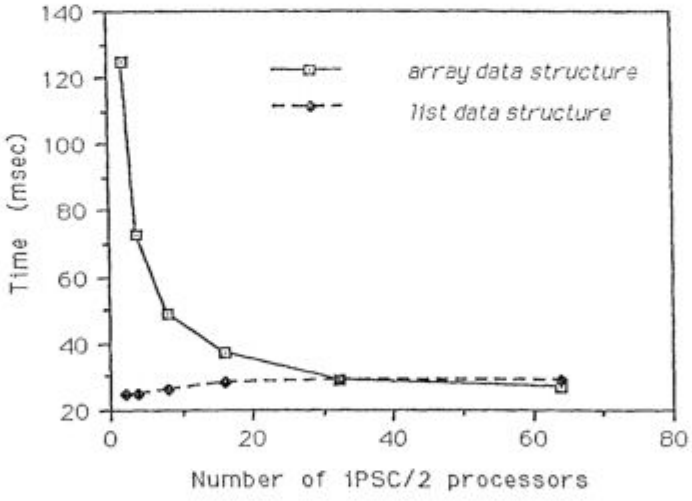
Σχήμα 6.22: iPSC/2:Υπολογισμός ΚΠ με τον αλγόριθμο Α. Ανάλυση επιμέρους χρόνων για 3350 σημεία



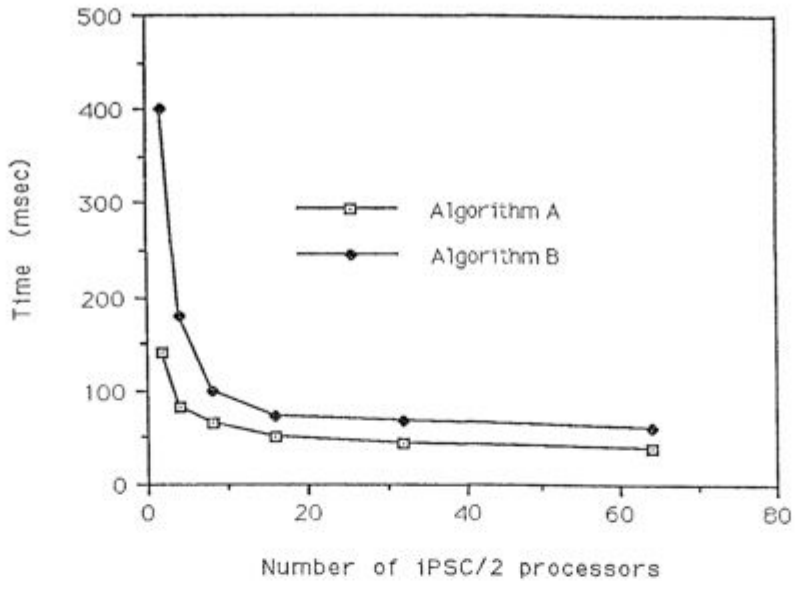
Σχήμα 6.23: iPSC/2: Χρόνος υπολογισμού ΚΠ με τον αλγόριθμο Β, ως προς το μέγεθος της εικόνας



Σχήμα 6.24: iPSC/2: ΚΠ, χρόνοι τοπικής και ολικής επεξεργασίας με τον Αλγόριθμο Β και 3500 σημεία



Σχήμα 6.25: ΚΠ: οι χρόνοι των αλγόριθμων Α και Β σε εικόνα με 256 σημεία, ως προς το πλήθος των επεξεργαστών



Σχήμα 6.26: ΚΠ: Οι Αλγόριθμοι Α και Β για 256x256 εικόνα με 3350 σημεία ως προς το πλήθος των επεξεργαστών

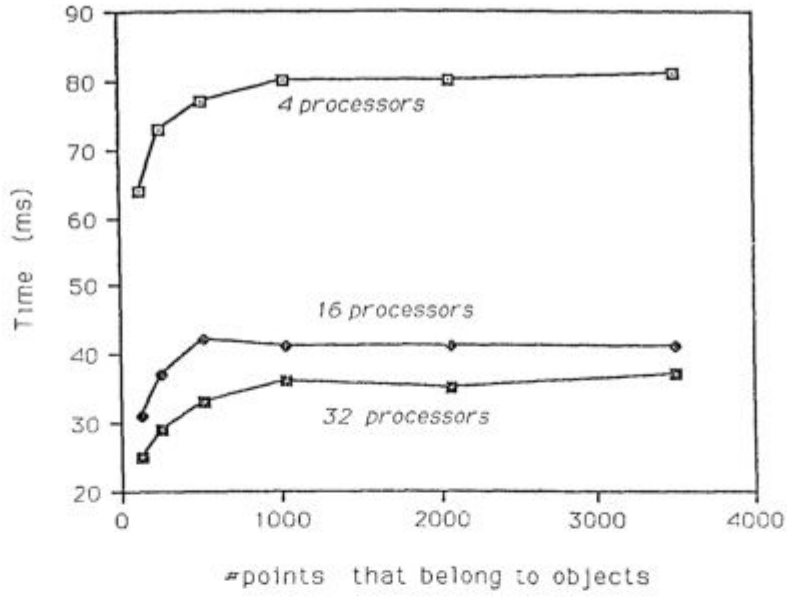
όσο αυξάνει το πλήθος των επεξεργαστών.

Για να έχουμε μια πληρέστερη εικόνα της απόδοσης των δύο αλγορίθμων, παραθέτουμε τα επόμενα Σχ. 6.27 και 6.28 στα οποία φαίνονται οι επιδόσεις τους ως προς το πλήθος των σημείων του αντικειμένου που εμφανίζεται στην εικόνα σε 4, 16 και 32 επεξεργαστές. Οι εικόνες έχουν μέγεθος 256x256.

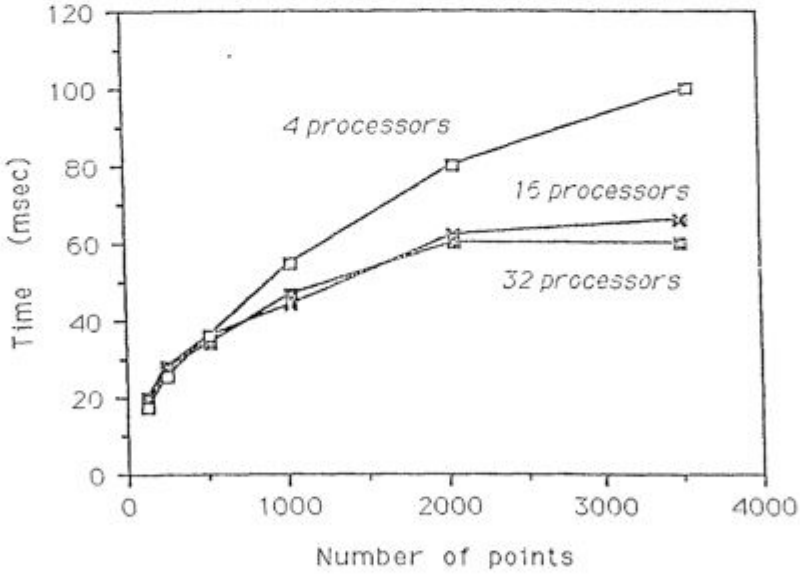
Τέλος παραθέτουμε το Σχ. 6.29 με τους αναμενόμενους και μετρημένους χρόνους για τον αλγόριθμο Β.

6.2.7 Υπολογισμός κυρτού περιγράμματος: Επιλογές υλοποίησης

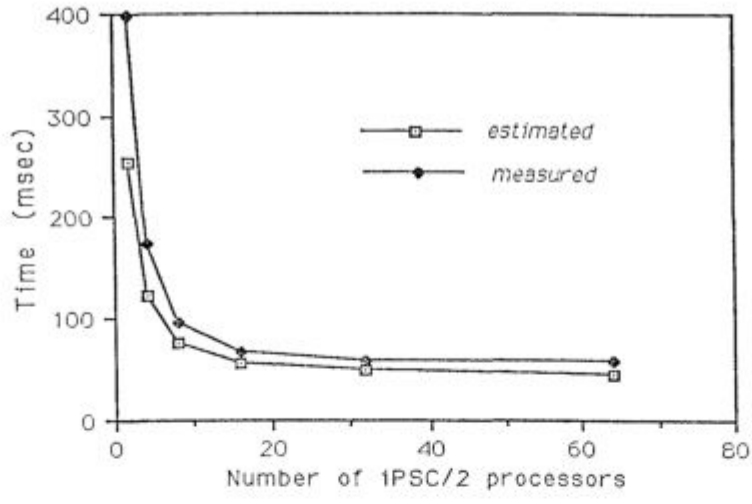
Οι επιλογές υλοποίησης μπορούν να βασιστούν στη σύγκριση των αναλυτικά υπολογισμένων ολικών χρόνων εκτέλεσης T_{tot}^{CM}, T_{tot}^{A} και T_{tot}^{B}, που αντιστοιχούν στον υπολογισμό του κυρτού περιγράμματος πρώτον στο CM, και δεύτερον στο iPSC/2 με χρήση λίστας ή πίνακα σαν δομή των δεδομένων εισόδου. Τέτοιες συγκρίσεις μπορούν να γίνουν για διάφορες τιμές των παραμέτρων, που χρησιμοποιήσαμε προκειμένου να χαρακτηρίσουμε το περιεχόμενο των εικόνων



Σχήμα 6.27: ΚΠ: Ο αλγόριθμος Β συναρτήσει του πλήθους των σημείων



Σχήμα 6.28: ΚΠ: Ο αλγόριθμος Α συναρτήσει του πλήθους των σημείων

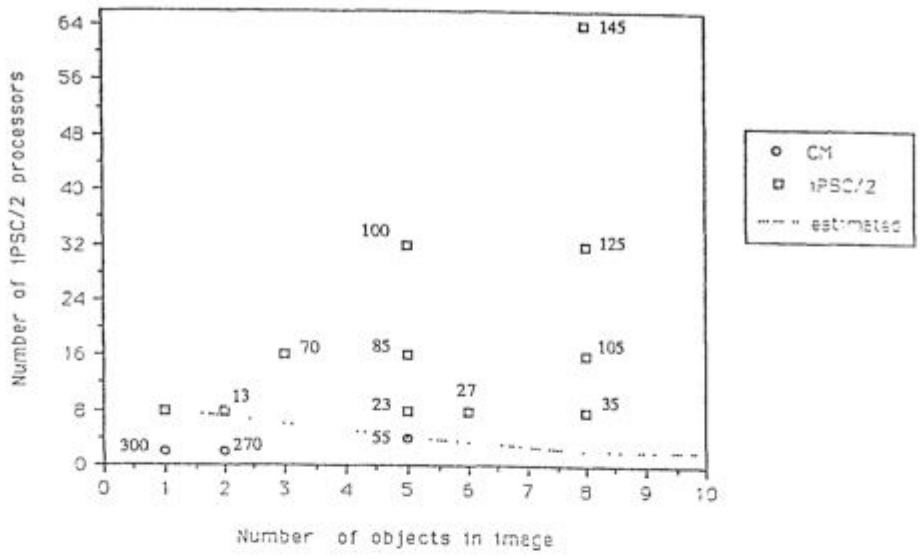


Σχήμα 6.29: ΚΠ: Αναμενόμενοι και πραγματικοί χρόνοι του Αλγόριθμου Β

και τις αρχιτεκτονικές.

Στα επόμενα σχήματα φαίνονται τα αποτελέσματα από τέτοιες συγκρίσεις για διάφορες τιμές γνωρισμάτων των αρχιτεκτονικών ή του περιεχόμενου της εικόνας τα οποία εμπλέκονται στις αναλυτικές εκφράσεις, όπως τη μέγιστη καμπυλότητα του περιγράμματος (contour curvature), του πλήθους των επεξεργαστών στο iPSC/2 ή του πλήθους των αντικειμένων στην εικόνα. Σε κάθε περίπτωση, εξάγεται μία καμπύλη που αντιστοιχεί σε ίσους συνολικούς χρόνους εκτέλεσης στο CM και στο iPSC/2, και αυτή χρησιμοποιείται για να χωρίσει το χώρο των παραμέτρων σε δύο περιοχές, για τις οποίες το CM ή το iPSC/2 είναι η καλύτερη επιλογή ή ακόμα η μία δομή δεδομένων εισόδου ή η άλλη. Εκτός από τις αναλυτικές αυτές καμπύλες, στα αντίστοιχα διαγράμματα φαίνονται επίσης μεμονωμένα σημεία που αντιπροσωπεύουν την πειραματική σύγκριση των αντίστοιχων υλοποιήσεων. Το σύμβολο του σημείου αντιπροσωπεύει την προτεινόμενη κάθε φορά επιλογή, ενώ η αριθμητική ενδειξη, το ποσοστιαίο κέρδος σε χρόνο εκτέλεσης από την επιλογή αυτή.

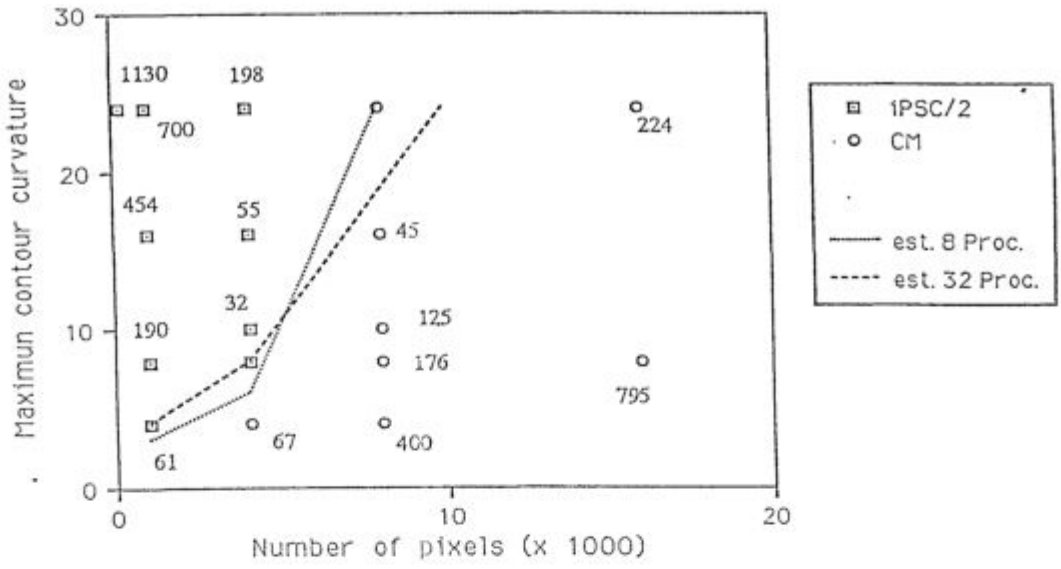
Στο Σχήμα 6.30 δείχνει τον διαμερισμό του παραμετρικού χωρου *Πλήθος αντικειμένων-Πλήθος επεξεργαστών του iPSC/2* με σύγκριση των χρόνων T_{tot}^{CM} και T_{tot}^{Λ} όταν η δομή των δεδομένων εισόδου είναι η λίστα, χρησιμοποιούνται 32Κ φυσικοί επεξεργαστές στο CM και το



Σχήμα 6.30: ΚΠ: Διαμερισμός τού χώρου (πλήθος επεξεργαστών iPSC/2 - πλήθος αντικειμένων) για επιλογή αρχιτεκτονικής

μέγιστο πλήθος σημείων σε μία κοιλότητα του περιγράμματος των αντικειμένων ήταν 4. Το συνολικό πλήθος των μη μηδενικών σημείων ήταν 3360. Παρατηρούμε πως εν γένει υπερέχει η υλοποίηση στο iPSC/2 με εξαίρεση τις περιπτώσεις που χρησιμοποιούμε την αρχιτεκτονική αυτή με λίγους επεξεργαστές. Οσο μάλιστα αυξάνει το πλήθος των αντικειμένων μέσα στην εικόνα, τόσο η υπεροχή του αλγορίθμου στο iPSC/2 γίνεται επικρατέστερη. Από αυτά τις αναλυτικές σχέσεις των υπολογισμού των χρόνων μπορούμε επίσης να συμπεράνουμε μία εξάρτηση από το πλήθος των μη μηδενικών σημείων της εικόνας (pixels), που ανήκουν σε αντικείμενο και όχι στο φόντο (background). Περιμένουμε πως, όσο αυτό το πλήθος μεγαλώνει, τα σχετικά πλεονεκτήματα του iPSC/2 θα μειώνονται.

Στο Σχήμα 6.31, συγκρίνονται οι πειραματικά εξαχθέντες συνολικοί χρόνοι εκτέλεσης T_{tot}^{CM} και T_{tot}^{A} και διαμερίζουν το χώρο *Πλήθος μη μηδενικών σημείων - Μέγιστη κοιλότητα περιγράμματος* σε δύο ημιχώρους από τους οποίους μπορούμε να επιλέξουμε αρχιτεκτονική όταν η δομή των δεδομένων εισόδου στις υλοποιήσεις στο iPSC/2 είναι η λίστα. Στα πειράματα αυτά χρησιμοποιήθηκαν 16K φυσικοί επεξεργαστές του CM διατεταγμένοι σε ένα πλέγμα 128x128, ενώ για το iPSC/2 λαμβανόταν ο καλύτερος χρόνος με χρήση οποιουδήποτε αριθμού επεξεργαστών. Αυτό έγινε διότι, όπως βλέπουμε και στα σχετικά διαγράμματα, η απόδοση

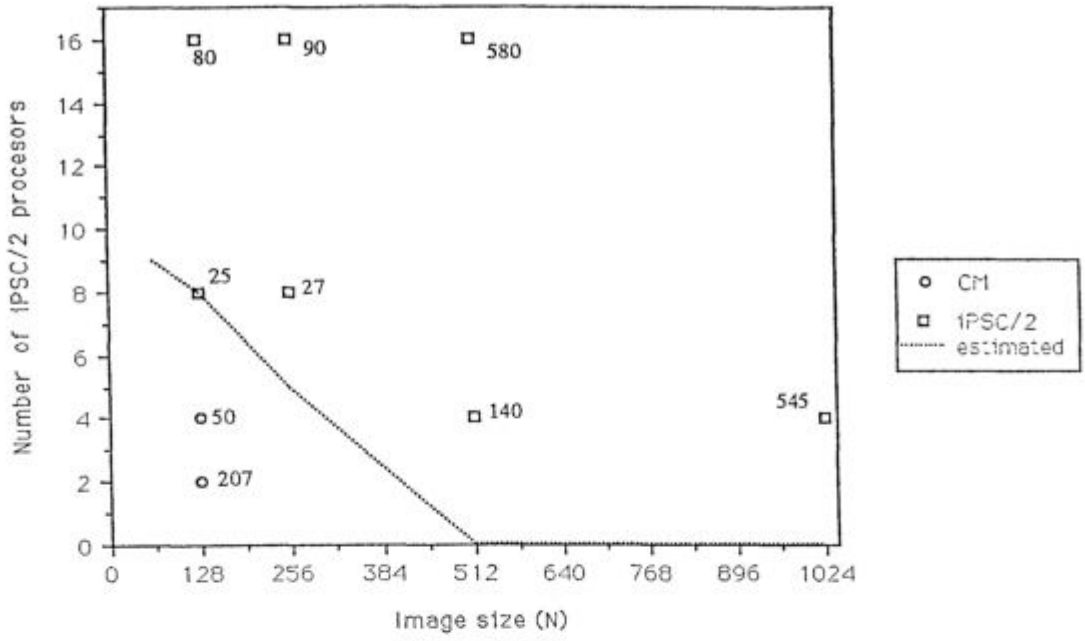


Σχήμα 6.31: ΚΠ: Διαμερισμός τού χώρου (πλήθος σημείων - μέγιστη κοιλότητα) για επιλογή αρχιτεκτονικής

του αλγορίθμου στο iPSC/2, δεν αυξάνει μονότονα με το πλήθος των χρησιμοποιούμενων επεξεργαστών. Οι αναλυτικές καμπύλες είναι για 8 και 32 επεξεργαστές, και βλέπουμε οτι ο χώρος διαμερίζεται και στις δύο περιπτώσεις περίπου κατά τον ίδιο τρόπο. Παρατηρούμε πως μετά από μερικές χιλιάδες σημείων, η επίδοση της υλοποίησης του αλγορίθμου στο Connection Machine είναι καλύτερη από εκείνη στο iPSC/2, ανεξάρτητα από το πλήθος των επεξεργαστών του τελευταίου. Αντίθετα για πολύ λίγα σημεία και αντικείμενα με μεγάλο αριθμό σημείων σε καμπυλότητες του περιγράμματός τους έχει σημαντικά καλλίτερη επίδοση.

Στο Σχήμα 6.32 παριστάνεται ένας διαμερισμός του χώρου *πλήθος επεξεργαστών iPSC/2 - μέγεθος εικόνας* σε δύο περιοχές στις οποίες είναι προτιμότερη η μία ή άλλη αρχιτεκτονική. Στα πειράματα που έγιναν χρησιμοποιήθηκαν 32K επεξεργαστές του CM, ενώ το πλήθος των μη μηδενικών σημείων ήταν 3360. Παρατηρούμε ξανά πως εν γένει είναι προτιμότερο το iPSC/2 ειδικά για εικόνες μέ μέγεθος μεγαλύτερο από 512x512.

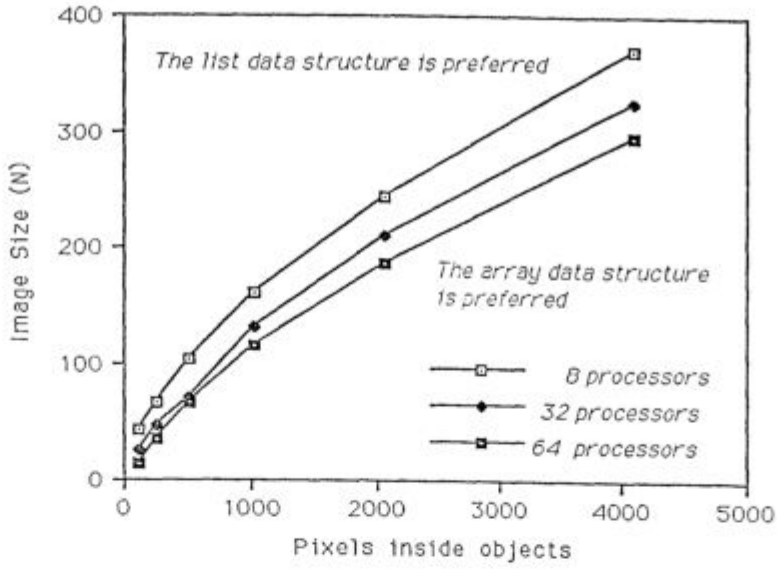
Τέλος, στο Σχήμα 6.33 βλέπουμε 3 καμπύλες υπολογισμένες αναλυτικά με σύγκριση των χρόνων T^{B}_{tot} και T^{A}_{tot} από τις σχέσεις (6.2) και (6.3) οι οποίες διαχωρίζουν τον παραμετρικό χώρο *μεγέθους εικόνας - πλήθους σημείων πάνω σε αντικείμενο* σε δύο ημίχωρους,



Σχήμα 6.32: ΚΠ: Επιλογή αρχιτεκτονικής στο χώρο (πλήθος επεξεργαστών iPSC/2 - μέγεθος εικόνας)

υποδεικνύοντας ότι στις παράλληλες υλοποιήσεις στο iPSC/2, η λίστα είναι προτιμότερη σαν δομή των δεδομένων εισόδου, όταν το μέγεθος της εικόνας είναι μεγάλο και το μέγεθος της εικόνας είναι σχετικά μικρό, ενώ ο πίνακας προτιμάται όταν το συνολικό πλήθος των σημείων της εικόνας είναι μικρό και το πλήθος των σημείων της εικόνας που ανήκουν σε αντικείμενα σχετικά μεγάλο. Τα τρία διαφορετικά διαγράμματα αντιστοιχούν στις περιπτώσεις που χρησιμοποιούμε 8, 32 ή 64 επεξεργαστές του iPSC/2. Παρατηρούμε ότι το πλήθος των επεξεργαστών του iPSC/2 δεν έχει σημαντική επίδραση στο διαχωρισμό του χώρου των παραμέτρων.

Τα παραδείγματα των επιλογών υλοποίησης που παρουσιάστηκαν σ'αυτή το τμήμα σχετίζονται με μερικές μόνο από τις παραμέτρους των χαρακτηριστικών του περιεχομένου της εικόνας και των παράλληλων αρχιτεκτονικών. Σε κάθε παράδειγμα, εκχωρήθηκαν τυπικές τιμές στις υπόλοιπες παραμέτρους. Είναι προφανές πως, αν αυτές οι τιμές αλλάξουν, ο χώρος των παραμέτρων θα διαχωριστεί διαφορετικά.



Σχήμα 6.33: ΚΠ: Διαμερισμός παραμετρικού χώρου για επιλογή δομής δεδομένων εισόδου στο iPSC/2

6.3 Εύρεση Συνδεδεμένων Συστατικών (Connected Component Labeling)

Μία δεύτερη βασική διεργασία στην ανάλυση δυαδικών εικόνων είναι εκείνη της εύρεσης Συνδεδεμένων Συστατικών (Connected Components) που αποτελεί συχνά απαραίτητο στάδιο προεπεξεργασίας της εικόνας για ανωτέρου επιπέδου διεργασίες [175].

Το πρόβλημα εύρεσης Συνδεδεμένων Συστατικών είναι εκείνο της εκχώρησης της ίδιας ετικέτας (label) σε όλα τα επιμέρους συστατικά που ανήκουν στο ίδιο αντικείμενο της εικόνας.

Στις υλοποιήσεις που θα παρουσιάσουμε, συστατικά που ανήκουν στο ίδιο αντικείμενο είναι τα σημεία της δυαδικής εικόνας (pixels) τα οποία θεωρούμε συνδεδεμένα με βάση έναν από τους ορισμούς συνεκτικότητας των pixels μίας εικόνας. Δύο κοινοί ορισμοί της συνεκτικότητας είναι η 4-συνεκτικότητα (ένα στοιχείο της εικόνας είναι συνδεδεμένο με τους 4 γείτονές του στην οριζόντια και κάθετη διεύθυνση) και η 8-συνεκτικότητα (συμπεριλαμβάνονται και οι δύο διαγώνιες κατευθύνσεις).

Οι γνωστοί αλγόριθμοι για εύρεση συνδεδεμένων συστατικών μπορούν να ταξινομηθούν σε 2 κατηγορίες: τους τοπικούς αλγόριθμους και τους αλγόριθμους τυχαίας πρόσβασης. Οι τοπικοί αλγόριθμοι επαναληπτικά αλλάζουν την ετικέτα κάθε σημείου βασιζόμενοι στις ετικέτες των γειτονικών σημείων. Με τους αλγόριθμους τυχαίας πρόσβασης η αλλαγή των ετικετών των σημείων της εικόνας γίνεται με χρήση καθολικής (global) πληροφορίας πράγμα που προγραμματιστικά επιτυγχάνεται καταφεύγοντας σε πολύπλοκους χειρισμούς δεικτών.

Οι τοπικοί αλγόριθμοι είναι πιο εύκολο να κωδικοποιηθούν, αλλά έχουν μάλλον λιγότερο καλή απόδοση στη γενική περίπτωση [166]. Οι αλγόριθμοι τυχαίας πρόσβασης επιτυγχάνουν καλύτερη απόδοση αλλά είναι πιο πολύπλοκοι στην υλοποίησή τους.

Στο [142] οι Shiloach-Vishkin προτείνουν έναν αλγόριθμο Συνδεδεμένων Συστατικών με πολυπλοκότητα O(logN) για SIMD μηχανή που διαθέτει κοινή μνήμη (shared memory). Στην ίδια εργασία μετατρέπεται ο παραπάνω αλγόριθμος για υλοποίηση σε MIMD αρχιτεκτονική με κοινόχρηστη μνήμη ενώ τον υλοποιούν στο Ultracomputer του NYU [66], όπου χρησιμοποιείται η πράξη του Fetch-and-Add. Η λειτουργία του αλγορίθμου παρουσιάζει μια δυναμική ανάθεση εργασίας στους επεξεργαστές, αλλά δεν δίνονται πειραματικά αποτελέσματα για την απόδοση σε χρόνο του αλγορίθμου. Επιπλέον, ο αλγόριθμος αυτός προϋποθέτει μια αρχική δομή γράφου μεταξύ των σημείων της εικόνας που δεν είναι προφανές πώς δημιουργείται αρχικά και με ποιο υπολογιστικό κόστος.

Στο CM αναπτύχθηκαν δύο αλγόριθμοι (που θα αναφέρονται σαν αλγόριθμοι A και B) που ανήκουν στην κατηγορία των τοπικών αλγορίθμων, ενώ ο αλγόριθμος που αναπτύχθηκε για το iPSC/2 έχει στην πρώτη του φάση τοπικό χαρακτήρα, ενώ ολοκληρώνεται με χρήση καθολικής (global) επικοινωνίας.

6.3.1 Παράλληλη υλοποίηση με μία μονάδα δεδομένων ανά επεξεργαστή

Συνδεδεμένα Συστατικά: Αλγόριθμος A

Στο CM η υλοποίηση είναι του τύπου "ένα στοιχείο εικόνας ανά επεξεργαστή" (pixel per processor) ενώ χρησιμοποιεί επικοινωνιακά σχήματα τα οποία εκμεταλλεύονται τις ιδιαίτερες

δυνατότητες της αρχιτεκτονικής και συγκεκριμένα τις προθεματικές πράξεις (prefix operations). Ο αλγόριθμος αυτός αναφέρεται στην [79] και δουλεύει ως εξής:

Κάθε επεξεργαστής έχει μία μεταβλητή *pixel* που αντιπροσωπεύει την τιμή του αντίστοιχου σημείου της εικόνας. Η μεταβλητή *pixel* παίρνει τιμή 1 για σημείο που ανήκει σε κάποιο αντικείμενο ή τιμή 0 για σημείο που ανήκει στο background. p_{ij} είναι ο επεξεργαστής στη θέση (i,j) του πλέγματος και έχει σαν γείτονές του τους επεξεργαστές $p_{i-1,j}$ και $p_{i+1,j}$ στην κατακόρυφη διεύθυνση και $p_{i,j-1}$ και $p_{i,j+1}$ στην οριζόντια. Αντίστοιχα, $pixel_{ij}$ είναι η τιμή της μεταβλητής *pixel* που κρατά ο επεξεργαστής p_{ij} του πλέγματος. Προκειμένου ο επεξεργαστής p_{ij} να αποκτήσει την τιμή μιας μεταβλητής γειτονικού επεξεργαστή θα πρέπει να εκτελέσει μια εντολή επικοινωνίας. Για λόγους απλότητας οι εντολές αυτές δεν σημειώνονται, αλλά εννοούνται στον αλγόριθμο που ακολουθεί. Οι εντολές που φαίνονται στο Σx. 6.34 εκτελούνται παράλληλα από κάθε επεξεργαστή: p_{ij}.

Procedure **Connected Components A**

(1) **if!!** $(pixel_{i,j} = 1)$ $label_{i,j}$:= processor-address
$flag_{i,j}$:= TRUE

(2) **if!!** $((flag_{i,j})$ **and** $(\text{not}(flag_{i-1,j})))$ **then** $region - up - limit_{i,j}$:= TRUE

(3) **if** $((flag_{ij})$ **and** **(not** $(flag_{i+1,j})))$ **then** $region - down - limit_{i,j}$:= TRUE

(4) **if** $((flag_{ij})$ **and** **(not** $(flag_{i,j-1})))$ **then** $region - left - limit_{i,j}$:= TRUE

(5) **if** $((flag_{ij})$ **and** **(not** $(flag_{i,j+1})))$ **then** $region - right - limit_{i,j}$:= TRUE

(6) **scan-min** (*direction* : x, var : *label*, *segmented-flag* : *region-left-limit*)

(7) **scan-copy-backwards** (*direction* : x, var : *label*, *segmented-flag* : *region-right-limit*)

(8) **scan-min** (*direction* : y, var : *label*, *segmented-flag* : *region-up-limit*)

(9) **scan-copy-backwards** (*direction* : y, var : *label*, *segmented-flag* : *region-down-limit*)

Σχήμα 6.34: Ο Αλγόριθμος *Connected Components A* για το Connection Machine

Με την εντολή 1 κάθε επεξεργαστής που κρατάει μη μηδενικό σημείο της εικόνας, (δηλαδή σημείο σε αντικείμενο), δίνει μία τιμή στην μεταβλητή $label_{ij}$. Η αρχική τιμή κάθε σημείου είναι η ταυτότητα του επεξεργαστή που το κρατά και είναι μοναδική. Με το τέλος του αλγόριθμου

θα πρέπει όλα τα σημεία της κάθε περιοχής να αποκτήσουν την ίδια τιμή στην $label_{ij}$ η οποία μάλιστα θα είναι η μικρότερη που εμφανίζεται στην περιοχή. Επίσης "σηκώνει" τη σημαία **flag** που θα τον χαρακτηρίζει στο εξής σαν ενεργό επεξεργαστή. Οι εντολές 2 έως 5 σηκώνουν τις τέσσερις αντίστοιχες σημαίες ορίζοντας κατά αυτό τον τρόπο τα όρια της περιοχής της κάθε συνεκτικής περιοχής που εμφανίζεται στην εικόνα. Η εντολή 6 διαβίβει στη x κατεύθυνση την μικρότερη τιμή της μεταβλητής $label_{ij}$ για κάθε συνεκτική περιοχή και σε κάθε γραμμή της εικόνας. Η εντολή αυτή συμπληρώνεται από την 7. Ομοίως οι 8 και 9 διαβίβουν τη μικρότερη τιμή κατά την y διεύθυνση αυτή την φορά. Οι εντολές 6-7 και 8-9 επαναλαμβάνονται διαδοχικά έως ότου διαδοθεί η μικρότερη τιμή της μεταβλητής $label_{ij}$ σε όλη τη συνεκτική περιοχή. Το πλήθος των επαναλήψεων των τεσσάρων εντολών 6-9 εξαρτάται από την πολυπλοκότητα του σχήματος (shape complexity) των αντικειμένων. Κάθε επανάληψή τους αντιστοιχεί σε πολυπλοκότητα 2.

Ανάλυση του αλγόριθμου A

Στη συνέχεια δίνουμε την ανάλυση του αλγορίθμου αυτού χωριστά για κάθε μία από τις τρεις φάσεις στις οποίες μπορούμε να τον χωρίσουμε. Δηλαδή τον υπολογισμό των συνθηκών εκκίνησης, τη δεύτερη φάση με τις επικοινωνίες μεταξύ των γειτονικών επεξεργαστών, και τέλος τη διάδοση των ετικετών κατά τις διευθύνσεις X και Y:

$$
\begin{aligned}
&1 && : T_1 = n_a t_a \\
&2\text{-}5 && : T_2 = n_\beta t_a + 4 * t_{msg} \\
&6\text{-}9 && : T_3 = n_\gamma t_a + 4 * t_{scan-segm-min}
\end{aligned}
$$

Ο χρόνος ολοκλήρωσης T_A^{CM} της διεργασίας εξαρτάται από την πολυπλοκότητα C του σχήματος, όπως αυτή έχει οριστεί, και θα είναι:

$$
\begin{aligned}
T_A^{CM} &= T_1 + T_2 + \frac{C}{2} * T_3 \\
T_A^{CM} &= (n_a + n_\beta + \frac{C}{2} * n_\gamma) t_a + 4 * t_{msg} + 2 * C * t_{scan-segm-min} \qquad (6.4)
\end{aligned}
$$

όπου $n_\alpha, n_\beta, n_\gamma$ το πλήθος των αριθμητικών και λογικών πράξεων που απαιτεί σε κάθε φάση, κάθε μία από τις οποίες χρειάζεται χρόνο t_a κατά μέσο όρο, t_{msg} ο χρόνος που χρειάζεται ένας επεξεργαστής για να στείλει ένα μήνυμα σε ένα γειτονικό του επεξεργαστή, $t_{scan-segm-min}$ ο χρόνος εκτέλεσης της τμηματοποιημένης πράξης σάρωσης με τελεστή scan, και C η μέγιστη πολυπλοκότητα σχήματος των συνεκτικών περιοχών που εμφανίζονται στην εικόνα.

Συνδεδεμένα Συστατικά: Αλγόριθμος Β

Ενας δεύτερος αλγόριθμος που υλοποιήθηκε στο Connection Machine είναι αυτός που περιγράφεται στο [166]. Στηρίζεται στην επεξεργασία "ενός σημείου ανά επεξεργαστή", δεν χρησιμοποιεί προθεματικές πράξεις σάρωσης ή άλλες βασικές πράξεις (primitives) του συστήματος, έχει πολυπλοκότητα O(N) για μια εικόνα NxN και χρησιμοποιεί ένα μονοδιάστατο πίνακα μήκους 2N για κάθε επεξεργαστή στον οποίο κρατάει τη διαδοχική εξέλιξη της τιμής του σημείου του αντίστοιχου επεξεργαστή με τη διαδοχική εφαρμογή σε όλους, μιας πράξης συρρίκνωσης (Shrinking primitive). Αυτή η πράξη συρρίκνωσης υλοποιείται με απλές επικοινωνίες μεταξύ γειτονικών επεξεργαστών, (nearest neighbor), και έχει ενδιαφέρουσες ιδιότητες [176]. Ο αλγόριθμος έχει την πολύ καλή αυτή πολυπλοκότητα αλλά εξαρτάται από τη Manhattan διάμετρο των αντικειμένων της εικόνας, όπως αυτή περιγράφεται στο τμήμα 4.4. Αυτό σημαίνει πως για μεγάλες εικόνες, ή ακριβέστερα για μεγάλα αντικείμενα, πρακτικά η απόδοσή του είναι πολύ χαμηλή. Αντίθετα γίνεται ενδιαφέρων για αντικείμενα με μεγάλη πολυπλοκότητα σχήματος όπως η σπείρα (Spiral).

Η ανάλυση του αλγορίθμου αυτού δίνει:

$$T_B = (n_a D - 5)t_a + (n_b D - 6) * t_{msg} \tag{6.5}$$

από την οποία φαίνεται ότι η απόδοση του αλγορίθμου αυτού εξαρτάται από τη διάμετρο Manhattan D. θα πρέπει επίσης να σημειωθεί η ανάγκη διατήρησης ενός πίνακα μεγέθους 2P σε κάθε επεξεργαστή, για πλέγμα επεξεργαστών PxP. Το μέγεθος αυτό της μνήμης είναι σημαντικό για συστήματα με τόσο μεγάλο πλήθος επεξεργαστών.

Από την ανάλυση, αλλά και από τα πειραματικά αποτελέσματα φαίνεται πως ο αλγόριθμος,

παρά τη μικρή θεωρητικά πολυπλοκότητά του, έχει πολύ μεγάλο χρόνο εκτέλεσης λόγω του πλήθους των επικοινωνιών που απαιτεί αλλά και λόγω της μικρής υπολογιστικής ισχύος των επεξεργαστών.

6.3.2 Πειραματικά αποτελέσματα (Connection Machine)

Οι υλοποιήσεις που έγιναν στο CM είχαν τα χαρακτηριστικά της λεπτής διαμέρισης (λόγω του πλήθους των επεξεργαστών), χρησιμοποίησαν μόνο τοπικές επικοινωνίες (nearest neighbor), ενώ ο πρώτος από αυτούς εκμεταλλεύτηκε τις πράξεις σάρωσης του συστήματος.

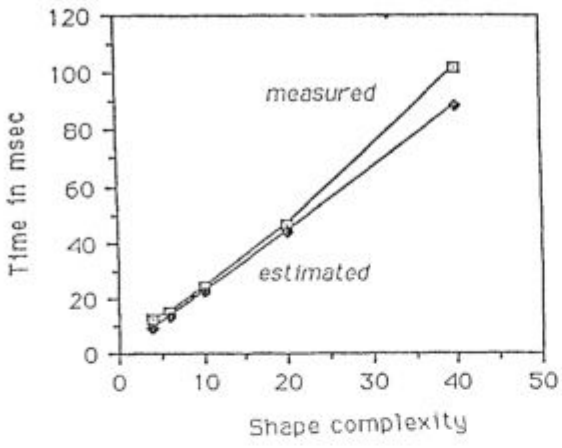
Το Σχήμα 6.35(α) δείχνει τον αναμενόμενο και τον πραγματικό χρόνο συναρτήσει της πολυπλοκότητας του σχήματος. Η αύξηση του χρόνου, όπως δείχνουν και οι αντίστοιχοι τύποι υπαγορεύεται από το πλήθος των πράξεων σάρωσης που χρειάζονται. Παρατηρούμε πολύ καλή σχέση μεταξύ της αναμενόμενης και της πραγματικής απόδοσης. Στο Σχήμα 6.35(β) φαίνονται οι χρόνοι για μια εικόνα με 4 αντικείμενα με πολυπλοκότητα 4 σε αυξανόμενο μέγεθος εικόνας. Από 32x32 η εικόνα γίνεται 512x512 χρησιμοποιώντας κάθε φορά έναν επεξεργαστή για κάθε σημείο της εικόνας. Στο πείραμα αυτό, επειδή το πλήθος των επεξεργαστών δεν αρκούσε για επεξεργασία με "ένα σημείο ανά επεξεργαστή", χρησιμοποιήθηκαν ιδεατοί επεξεργαστές του CM. Στις πρώτες τέσσερις διαμερίσεις με 32x32, 64x64, 128x128 και 256x256 επεξεργαστές χρησιμοποιήθηκε ένας Sequencer του CM που διαθέτει 8K επεξεργαστές. Έτσι, η τελευταία διαμέριση που χρειάζεται 64K επεξεργαστές υλοποιήθηκε χρησιμοποιώντας τον κάθε επεξεργαστή σαν 8 ιδεατούς. Στη συνέχεια χρησιμοποιήθηκαν 2 Sequencers που διέθεταν συνολικά 16K επεξεργαστές και έτσι η προηγούμενη διαμέριση υλοποιήθηκε από 4 ιδεατούς επεξεργαστές για κάθε πραγματικό. Αυτό δικαιολογεί την κατακόρυφη μείωση του χρόνου περίπου κατά 35%. Λογικά θα περίμενε κανείς μεγαλύτερη μείωση μιας και κάθε επεξεργαστής εκτελούσε χρέη για 8 αντί για 4 ιδεατούς αλλά δεν πρέπει να παραβλέψουμε το γεγονός ότι η επικοινωνία είναι μειωμένη στην πρώτη περίπτωση αφού στην πραγματικότητα μεταξύ των ιδεατών επεξεργαστών δεν στέλνονται μηνύματα. Το ίδιο φαινόμενο παρατηρείται στην συνέχεια όταν χρησιμοποιούνται οι 16K επεξεργαστές για διαμέριση 256x256 και για 512x512. Στην πρώτη περίπτωση έχουμε 4 ιδεατούς επεξεργαστές ενώ στη δεύτερη 16 ανά πραγματικό επεξεργαστή. Τέλος επαναλαμβάνεται το πείραμα χρησιμοποιώντας και τους 4 Sequencers που είχαμε διαθέσιμους με 32K επεξεργαστές και παρατηρούμε ξανά μια πτώση στο χρόνο της τάξης του 40%.

Στο τελευταίο Σχήμα 6.35(γ) βλέπουμε τον πραγματικό χρόνο του δεύτερου αλγόριθμου συναρτήσει της Διαμέτρου Manhattan. Βλέπουμε ξανά μια πολύ καλή πρόβλεψη του χρόνου ο οποίος αυξάνει ανάλογα με τη ΔΜ και φτάνει σε πολύ μεγάλες τιμές. Έτσι για ΔΜ=128 που αντιστοιχεί σε ένα αντικείμενο 64x64 ο χρόνος είναι περίπου 260 msec, ο οποίος για τον πρώτο αλγόριθμο αντιστοιχεί σε πολυπλοκότητα σχήματος πάνω από 120. Ομως σχήματα με τέτοια πολυπλοκότητα σε μια εικόνα 64x64 δεν αποτελούν τη συνήθη περίπτωση. Θα πρέπει το σχήμα να έχει πολύ λεπτά τμήματα με διαγώνια κατεύθυνση. Γίνεται λοιπόν προφανές πως ο πρώτος αλγόριθμος έχει σαφή υπεροχή στην απόδοση για απλά σχήματα, ανεξάρτητα από το μέγεθος και το πλήθος τους.

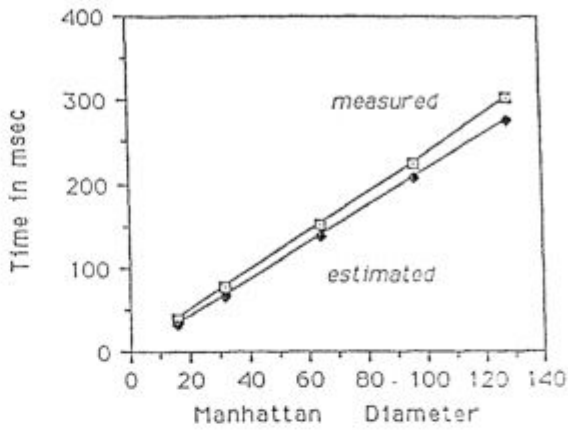
Τέλος, παρατηρούμε πως το πλήθος των αντικειμένων που εμφανίζονται στην εικόνα είναι και στις δύο περιπτώσεις σχεδόν αδιάφορο. Το ίδιο συμβαίνει και με το μέγεθος της εικόνας, με την επιφύλαξη πως η μεγαλύτερη εικόνα αφήνει περιθώρια για μεγαλύτερα αντικείμενα με αντίστοιχα μεγαλύτερες τιμές ΔΜ.

6.3.3 Παράλληλη υλοποίηση με πολλές μονάδες δεδομένων ανά επεξεργαστή

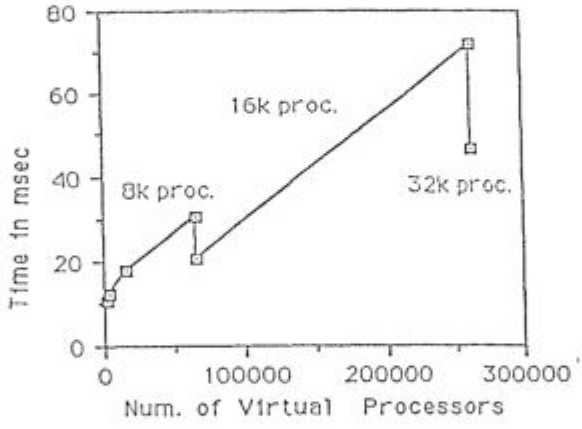
Στην περίπτωση αυτή, χωρίζουμε την εικόνα σε τόσες λωρίδες, όσοι είναι οι διαθέσιμοι επεξεργαστές της αρχιτεκτονικής. Για Ε επεξεργαστές, και μέγεθος εικόνας εισόδου PxP, χωρίζουμε την εικόνα σε Ε οριζόντιες λωρίδες εμβαδού $\frac{P*P}{E}$ pixels η καθεμία. Γειτονικές λωρίδες εικόνας στέλνονται σε γειτονικούς επεξεργαστές. Κάθε επεξεργαστής, εκτελεί ένα σειριακό αλγόριθμο πάνω στη λωρίδα εικόνας που του έχει εκχωρηθεί. Σε περίπτωση που ο αρχικός διαμερισμός της εικόνας ήταν τέτοιος ώστε ένα συνδεδεμένο συστατικό να διανεμηθεί σε δύο ή περισσότερους επεξεργαστές, τότε στο συστατικό αυτό, μετά την πρώτη επεξεργασία, θα έχουν εκχωρηθεί περισσότερες από μια ετικέτες. Αυτές οι διπλές εκχωρήσεις μπορούν να ανιχνευθούν τοπικά με σύγκριση των οριακών γραμμών των λωρίδων της εικόνας που μοιράστηκαν αρχικά στους επεξεργαστές. Για το σκοπό αυτό, κάθε επεξεργαστής στέλνει στον επόμενό του την τελευταία γραμμή από τη λωρίδα εικόνας που επεξεργάστηκε. Μετά από αυτό, κάθε επεξεργαστής είναι σε θέση να βρει τις τοπικές ισοδυναμίες ετικετών. Στη συνέχεια, οι τοπικές αυτές ισοδυναμίες συγκεντρώνονται σε $\log E$ βήματα σε ένα επεξεργαστή ο οποίος υπολογίζει



(α)



(β)



(γ)

Σχήμα 6.35: Μετρήσεις για τη διεργασία εύρεσης των συνδεδεμένων συστατικών στο CM

τις καθολικές ισοδυναμίες οι οποίες διανέμονται πάλι σε όλους τους επεξεργαστές. Ετσι, κάθε επεξεργαστής έχοντας το σύνολο των ισοδυναμιών των ετικετών ενημερώνει σωστά πλέον τις ετικέτες των σημείων της περιοχής που του αντιστοιχεί. Συνοπτικά ο αλγόριθμος φαίνεται στο Σχ. 6.36. Ο αλγόριθμος αυτός έχει αρκετές ομοιότητες με εκείνον που παρουσιάζεται στην [177] με τη διαφορά ότι στον τελευταίο η εικόνα χωρίζεται σε τετράγωνες περιοχές και χρησιμοποιούνται τα περιγράμματα των αντικειμένων για να ανιχνευθεί η συνέχειά τους στις διαδοχικές υποεικόνες.

Procedure **Connected Components**

(1) **Sequential Connnected Component Labeling** *on the* $\frac{P^2}{E}$ *subimage*
(2) **if** (*id* < *E-1*) **then**
(3) **send** (*last row of local subimage*, **to:***id+1*)
(4) **if** (*id* > *0*) **then**
(5) **receive**(*last row of previous subimage*)
(6) **Estimate local equalities of Labels**
(7) **Send** (*local equalities*, **to:***0*)
(8) **if** (*id* =*0*) **then**
(9) **Estimate Global label equalities**
(10) **Send** (*Global equalities*, **to:***all-processors*)
(11) **Update** *local pixels according to Global equalities*
End

Σχήμα 6.36: Ο αλγόριθμος εύρεσης συνδεδεμένων συστατικών στο iPSC/2

Με την εντολή 1 κάθε επεξεργαστής βρίσκει σειριακά τα συνδεδεμένα τμήματα πάνω στη λωρίδα εικόνας που του εκχωρήθηκε. Κάθε επεξεργαστής χαρακτηρίζεται μοναδικά από την τιμή της μεταβλητής *id* που παίρνει τιμές 1 έως E, έτσι ώστε στις επόμενες εντολές (2-3) κάθε επεξεργαστής εκτός από τον τελευταίο, στέλνει στον επόμενό του την τελευταία γραμμή από την επεξεργασμένη λωρίδα εικόνας. Στη συνέχεια (4-7) κάθε επεξεργαστής εκτός από τον πρώτο δέχεται τη γραμμή αυτή, υπολογίζει τις τοπικές ισοδυναμίες των ετικετών, και τις στέλνει στον επεξεργαστή 0. Ο επεξεργαστής 0 έχοντας συγκεντρώσει όλες αυτές τις τοπικές

ισοδυναμίες υπολογίζει τις καθολικές τις οποίες στέλνει σε όλους τους επεξεργαστές. Η διαδικασία ολοκληρώνεται στην εντολή (11) όταν κάθε επεξεργαστής ενημερώνει τις ετικέτες που έχει αποδώσει στα σημεία του τμήματος της εικόνας που του έχει εκχωρηθεί από την αρχή της διεργασίας.

Ανάλυση του αλγορίθμου

Στη συνέχεια παραθέτουμε την ανάλυση του αλγορίθμου για τις διαδοχικές φάσεις του:

Ο σειριακός αλγόριθμος εφαρμοζόμενος στη λωρίδα εικόνας διαστάσεων $\frac{P^2}{E}$ απαιτεί χρόνο:

$$\mathrm{T}_1 = \frac{P^2}{E} * t_a + \frac{\mathrm{N}}{E} * t_{comp}$$

όπου t_{comp} είναι ο χρόνος που χρειάζεται για να θέσει την ετικέτα σε κάθε ένα από τα N στοιχεία (pixels) της εικόνας που ανήκουν σε αντικείμενα ανάλογα με τις ετικέτες των γειτονικών του σημείων, ενώ t_a είναι ο χρόνος εκτέλεσης μιας πράξης *if* προκειμένου να διαπιστωθεί αν το κάθε σημείο ανήκει σε αντικείμενο.

Δεδομένου ότι οι μεταφορές δεδομένων ανάμεσα στους επεξεργαστές γίνονται παράλληλα, ο χρόνος που απαιτείται στο βήμα αυτό του αλγορίθμου είναι:

$$\mathrm{T}_2 = t_{startup} + P * t_b$$

Εστω ότι ο αριθμός ισοδύναμων ετικετών που ανιχνεύονται από ένα επεξεργαστή p_i είναι e_i. Εάν t_{eq} είναι ο χρόνος που απαιτείται για να ελέγξει τις ισοδυναμίες μεταξύ ετικετών που βρίσκονται εκατέρωθεν των διαχωριστικών γραμμών μεταξύ υποπεριοχών της εικόνας, τότε ο συνολικός χρόνος που απαιτείται για το βήμα του αλγορίθμου καθορίζεται από τον χρόνο του επεξεργαστή με τις περισσότερες ισοδυναμίες. Εστω e_{max} το μέγιστο των e_i. Τότε

$$T_3 = e_{max} * t_{eq}$$

Η συγκέντρωση των τοπικών ισοδυναμιών σε ένα επεξεργαστή γίνεται σε $\log E$ διαδοχικά

βήματα (δηλαδή όση η διάσταση του υπερκύβου). Η μεταφορά των τοπικών ισοδυναμιών γίνεται κατά μήκος μίας διάστασης του υπερκύβου κάθε φορά. Ο χρόνος επικοινωνίας που απαιτείται για τον i επεξεργαστή είναι:

$$T_4 = \sum_{j=1}^{\log \mathrm{E}} (t_{startup} + 2 * (\sum_{i=1}^{j} e_i) * t_b) = \log \mathrm{E} * t_{startup} + 2t_b \sum_{j=1}^{\log \mathrm{E}} \sum_{i=1}^{j} e_i$$

Ο επεξεργαστής με $id = 0$ υπολογίζει τις καθολικές ταυτότητες των τοπικών ετικετών, συγκρίνοντας τες. Για κάθε σύγκριση χρειάζεται χρόνο $t_{compare}$.

$$T_5 = \sum_{j=1}^{\log \mathrm{E}} e_j \log E (\sum_{j=1}^{\log \mathrm{E}} e_j) t_{compare}$$

Στη συνέχεια επαναλαμβάνεται η επικοινωνία του τέταρτου βήματος, αλλά με αντίστροφη φορά, δηλαδή το σύνολο των ισοδυναμιών μοιράζεται σε όλους τους επεξεργαστές. Επομένως:

$$T_6 = \log E * (t_{startup} + 2 * (\sum_{i=1}^{E-1} e_i) * t_b)$$

Εάν t_{update} είναι ο χρόνος που απαιτείται από ένα επεξεργαστή για να αντικαταστα θεί η ετικέτα ενός σημείου από εκείνη που υπολογίσθηκε τελικά, τότε ο χρόνος εκτέλεσης του βήματος αυτού είναι:

$$T_7 = \frac{P^2}{\mathrm{E}} * t_a + \frac{\mathrm{N}}{\mathrm{E}} * t_{update}$$

Σύμφωνα με τα παραπάνω, ο ολικός χρόνος εκτέλεσης του αλγορίθμου με E επεξεργαστές είναι:

$$T_E^{iPSC/2} = T_1 + T_2 + T_3 + T_4 + T_5 + T_6 + T_7$$

$$= \frac{P^2}{E} * t_a + \frac{\mathrm{N}}{E} * t_{comp} + e_{max} * t_{eq} + (2 \log \mathrm{E} + 1) t_{startup} + 2t_b (\sum_{j=1}^{\log \mathrm{E}} \sum_{i=1}^{j} e_i + \sum_{i=1}^{E-1} e_i) +$$

$$\sum_{j=1}^{\log \mathrm{E}} e_j \log E (\sum_{j=1}^{\log \mathrm{E}} e_j) t_{compare} + \frac{P^2}{\mathrm{E}} * t_{update} \qquad (6.6)$$

Οπως βλέπουμε η επίδοση του αλγορίθμου εξαρτάται καταρχήν από τα υπολογιστικά και επικοινωνιακά χαρακτηριστικά του συστήματος. Οσον αφορά το περιεχόμενο της εικόνας,

εξαρτάται από το πλήθος N των σημείων του αντικειμένου και από τον αριθμό e_{max} τμημάτων του αντικειμένου, τα οποία τέμνουν τα όρια των υποπεριοχών της εικόνας.

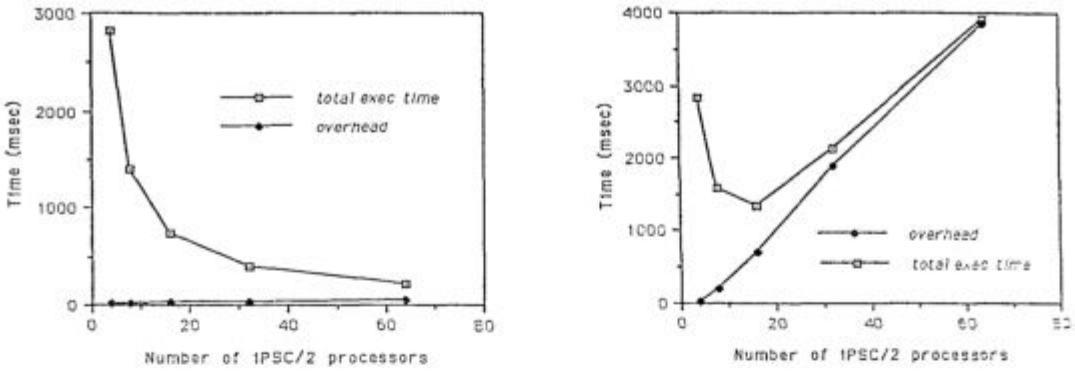
6.3.4 Πειραματικά αποτελέσματα (iPSC/2)

Ο αλγόριθμος που αναπτύχθηκε στο iPSC/2 δοκιμάστηκε πάνω σε εικόνες με διαφορετικά χαρακτηριστικά. Οι διαστάσεις των εικόνων αυτών είναι 512X512. Η πρώτη εικόνα περιέχει ένα συμμετρικό αντικείμενο το οποίο καταλαμβάνει το μεγαλύτερο μέρος της ενώ η δεύτερη παρουσιάζει ένα σπειροειδές αντικείμενο.

Τα χαρακτηριστικά της πρώτης εικόνας που επηρεάζει την απόδοση του αλγορίθμου, είναι πρώτο ότι έχουμε ισομερή κατανομή των σημείων σε κάθε λωρίδα που παίρνουν οι επεξεργαστές ανεξάρτητα από το πλήθος τους. Δηλαδή υπάρχει ισοκατανομή του φορτίου. Δεύτερο ότι ο αριθμός ισοδυνάμων ετικετών μεταξύ δύο γειτονικών λωρίδων παραμένει μικρός (ο μεγιστος αριθμός είναι δύο). Επομένως τα βήματα 4, 5 και 6 του αλγορίθμου δεν απαιτούν μεγάλο χρόνο για την εκτέλεσή τους.

Τα παραπάνω χαρακτηριστικά της εικόνας, ευνοούν τη γρήγορη εκτέλεση του αλγορίθμου. Οι χρόνοι εκτέλεσης και επικοινωνίας για διάφορους αριθμούς επεξεργαστών φαίνονται στο Σχ. 6.37.(α). Παρατηρούμε ότι ο υπολογιστικός χρόνος αρχικά είναι πολύ μεγαλύτερος από τον χρόνο επικοινωνίας. Αποτέλεσμα αυτού είναι ότι η επιτάχυνση του αλγορίθμου εξελίσσεται ικανοποιητικά για μέχρι και 16 επεξεργαστές. Για περισσότερους από 16 επεξεργαστές, υπάρχει βελτίωση στο χρόνο εκτέλεσης του αλγορίθμου, η οποία ωστόσο δεν είναι ανάλογη του αριθμού των χρησιμοποιούμενων επεξεργαστών. Σε εικόνες μικρότερου μεγέθους η διαφορά μεταξύ του χρόνου επικοινωνίας και του υπολογιστικού χρόνου είναι μικρότερη. Έτσι, οι αρνητικές συνέπειες που έχει η αύξηση του χρόνου επικοινωνίας στην επιτάχυνση του αλγορίθμου είναι περισσότερο εμφανείς.

Η σπειροειδής εικόνα που επίσης χρησιμοποιήθηκε αποτελεί την χειρότερη περίπτωση από άποψη χρόνου εκτέλεσης του αλγορίθμου. Οποιοσδήποτε διαμερισμός της εικόνας σε λωρίδες, έχει σαν αποτέλεσμα το μέγιστο αριθμό τομών των ορίων των υποπεριοχών της εικόνας με το αντικείμενο. Αποτέλεσμα αυτού είναι ότι αύξηση του αριθμού επεξεργαστών έχει σαν συνέπεια



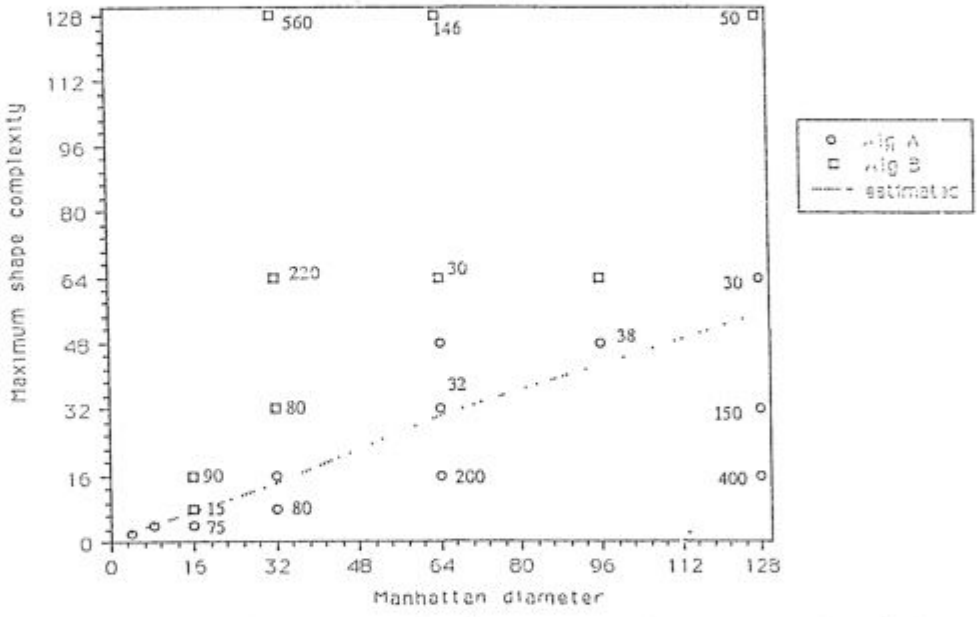
Σχήμα 6.37: Χρόνοι εκτέλεσης αλγορίθμου συνδεδεμένων συστατικών στο iPSC/2

την μεγάλη αύξηση των παραπάνω χρόνων. Οι χρόνοι εκτέλεσης του αλγορίθμου για την εικόνα αυτή συναρτήσει του αριθμού των χρησιμοποιούμενων επεξεργαστών, φαίνονται στο Σχ. 6.37.(β). Για 32 επεξεργαστές, ο αυξανόμενος χρόνος επικοινωνίας και υπολογισμού των ολικών ισοδυναμιών έχει καταλυτικά αποτελέσματα. Ο χρόνος εκτέλεσης του αλγορίθμου είναι μεγαλύτερος από εκείνον που απαιτείται για 16 επεξεργαστές. Το γεγονός ότι συνολικά, η επεξεργασία της πρώτης εικόνας απαιτεί περισσότερο χρόνο απ' ότι η επεξεργασία της δεύτερης, οφείλεται απλά στο γεγονός ότι ο αριθμός των σημείων που ανήκουν σε αντικειμένο διαφέρει σημαντικά μεταξύ των δύο εικόνων.

6.3.5 Συνδεδεμένα Συστατικά : Επιλογές υλοποίησης

Οι επιλογές των υλοποιήσεων για τη διεργασία της εύρεσης των συνδεδεμένων συστατικών σε μία εικόνα, γίνεται στη βάση της ανά ζεύγος σύγκρισης των αναλυτικών εκφράσεων των χρόνων εκτέλεσης T_A^{CM}, T_B^{CM}, και $T^{iPSC/2}$ των εξισώσεων (6.4), (6.5) και (6.6) για διαφορετικές τιμές των παραμέτρων που έχουν χρησιμοποιηθεί για το χαρακτηρισμό του περιεχομένου των εικόνων. Εκτός από τις αναλυτικές αυτές καμπύλες, στα αντίστοιχα διαγράμματα φαίνονται επίσης μεμονωμένα σημεία που αντιπροσωπεύουν την πειραματική σύγκριση των αντίστοιχων υλοποιήσεων. Το σύμβολο του σημείου αντιπροσωπεύει την προτεινόμενη κάθε φορά επιλογή, ενώ η αριθμητική ενδειξη, το ποσοστιαίο κέρδος σε χρόνο εκτέλεσης από την επιλογή αυτή.

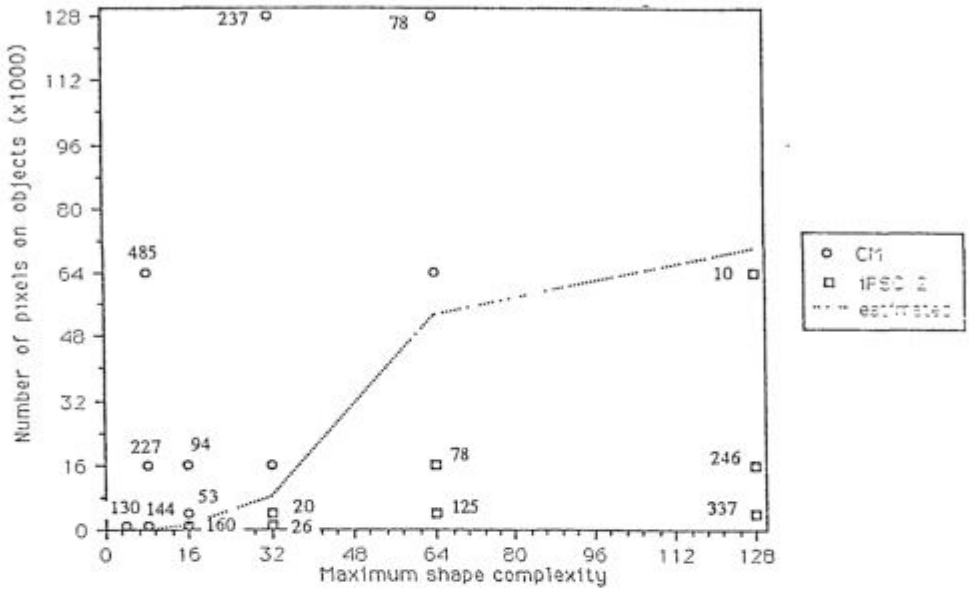
Στο Σχήμα 6.38 η σύγκριση των T_A^{CM} και T_B^{CM} υπαγορεύει την τμηματοποίηση του παραμετρικού χώρου της μέγιστης πολυπλοκότητας σχήματος και της διαμέτρου Manhattan σε περιοχές στις οποίες είναι προτιμητέος ο Α ή ο Β αλγόριθμος στην αρχιτεκτονική του CM. Τα πειράματα έγιναν με χρήση 16Κ φυσικών επεξεργαστών διατεταγμένων σε πλέγμα 256x256. Παρατηρούμε πως ανάλογα με την σχέση των συγκεκριμένων χαρακτηριστικών η επιτυγχανόμενη επιτάχυνση, υπό αυτές της συνθήκες, μπορεί να είναι και εξαπλάσια. Η αναλυτικά υπολογισμένη καμπύλη αντιπροσωπεύει τις σχέσεις στις τιμές των δύο χαρακτηριστικών για τις οποίες οι επιδόσεις των δύο αλγορίθμων είναι ίσες. Τα σημεία δίπλα στα οποία δεν υπάρχουν ποσοστά, αντιστοιχούν σε επιδόσεις που διαφέρουν λιγότερο από 10%. Αν, λοιπόν, η σχέση των δύο χαρακτηριστικών αντιστοιχεί σε σημείο που βρίσκεται πάνω από την καμπύλη θα πρέπει να διαλέξουμε τον αλγόριθμο Β, ενώ αν βρίσκεται κάτω θα διαλέξουμε τον αλγόριθμο Α.



Σχήμα 6.38: ΣΣ: Διαμερισμός παραμετρικού χώρου που επιτρέπει την επιλογή αλγόριθμου στο CM

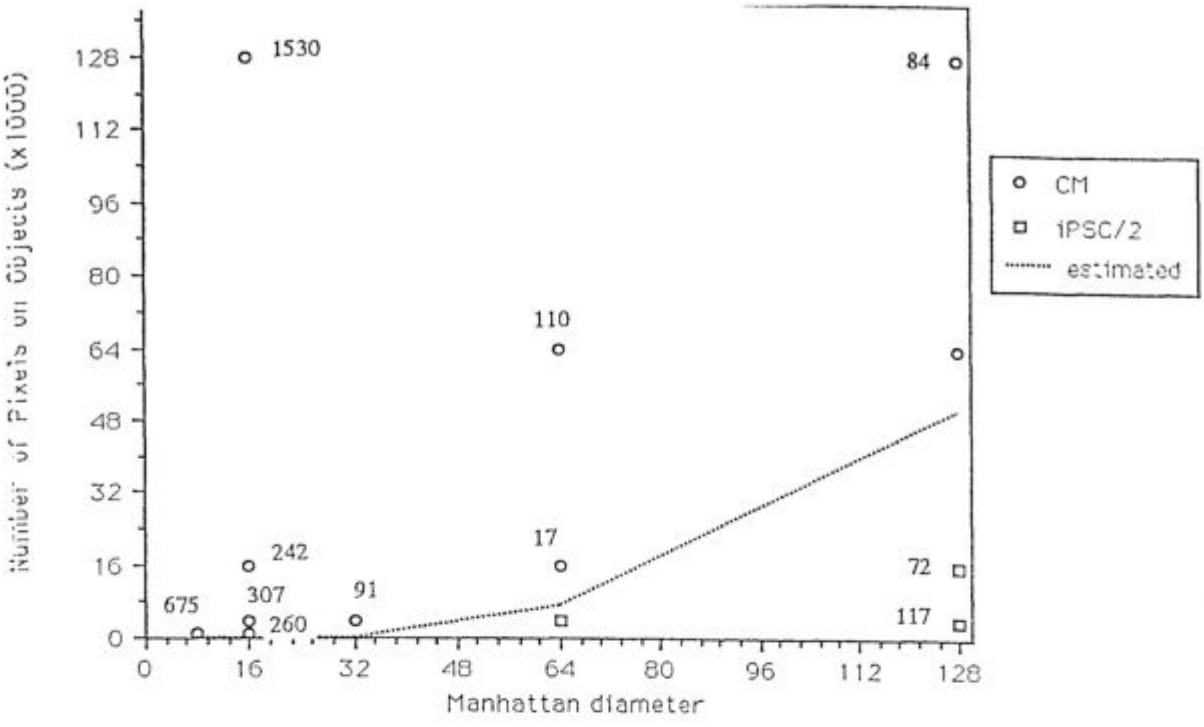
Στο επόμενο Σχ. 6.39 φαίνεται η σχέση μεταξύ των επιδόσεων του αλγορίθμου Α στο CM με τον αλγόριθμο στο iPSC/2. Η σύγκριση των επιδόσεων T_A^{CM} και $T^{iPSC/2}$ από τις εξισώσεις (6.4) και (6.6), επιτρέπει ξανά τον διαμερισμό του παραμετρικού χώρου σε δύο περιοχές στις οποίες είναι προτιμότερη η μία ή η άλλη αρχιτεκτονική. Η απόδοση του Αλγόριθμου Α στο CM εξαρτάται από την πολυπλοκότητα του σχήματος, ενώ η απόδοση του αλγόριθμου στο iPSC-2

εξαρτάται από τον αριθμό των σημείων που ανήκουν στα αντικείμενα της εικόνας. Παρατηρούμε πως για μεγάλα σχήματα με πολύ μικρή πολυπλοκότητα οι επιδόσεις στο CM μπορεί να είναι και πέντε φορές μεγαλύτερες, ενώ αντίθετα για λίγα αντικείμενα με μικρό εμβαδόν και με σημαντική πολυπλοκότητα, η υλοποίηση στο iPSC/2 μπορεί να είναι μέχρι και τέσσερις φορές καλύτερη. Η υλοποίηση έγινε σε πλέγμα 512x512 με 16K επεξεργαστές για το CM και με 8 επεξεργαστές για το iPSC/2.



Σχήμα 6.39: ΣΣ: Διαμερισμός παραμετρικού χώρου για επιλογή Αρχιτεκτονικής

Ομοίως στο Σχ. του Σχ. 6.40 φαίνεται η σχέση μεταξύ των επιδόσεων του αλγορίθμου Β στο CM με τον αλγόριθμο στο iPSC/2. Η αναλυτική διερεύνηση γίνεται με χρήση των εξισώσεων (6.5) και (6.6) που δίνουν τους χρόνους εκτέλεσης T_B^{CM} και $T^{iPSC/2}$. Η επίδοση του αλγόριθμου Β στο CM εξαρτάται από τη μέγιστη διάμετρο Manhattan των αντικειμένων της εικόνας ενώ η απόδοση του αλγόριθμου στο iPSC/2 από το συνολικά αριθμό σημείων που ανήκουν σε αντικείμενα. Παρατηρούμε πως ο συγκεκριμένος διαμερισμός υπαγορεύει μέχρι και δεκαπενταπλάσια ταχύτητα στο CM αν τα αντικείμενα είναι πάρα πολλά και μικρά. Αντίθετα για αντικείμενα με μικρό εμβαδόν αλλά με επίμηκες σχήμα, που ανεβάζει την τιμή της διαμέτρου Manhattan, η αρχιτεκτονική του iPSC/2 γίνεται πλεονεκτικότερη. Στα πειράματα αυτά το πλέγμα ήταν 512x512, χρησιμοποιήθηκαν 32K επεξεργαστές του CM και 8 επεξεργαστές του iPSC/2.



Σχήμα 6.40: ΣΣ: Ακόμα ένας διαμερισμός παραμετρικού χώρου για επιλογή αρχιτεκτονικής

6.4 Αλλες διεργασίες

Στην παράγραφο αυτή παρουσιάζονται συνοπτικά ορισμένες άλλες διεργασίες ανάλυσης εικόνων με διάφορα υπολογιστικά και επικοινωνιακά χαρακτηριστικά. Οι διεργασίες αυτές μπορούν να θεωρηθούν αντιπροσωπευτικές μεγάλων ομάδων διεργασιών. Οι αλγόριθμοι των δύο πρώτων είναι επαναληπτικοί και χαρακτηρίζονται από σημαντικές απαιτήσεις σε υπολογιστική ισχύ και ομοιογενή επικοινωνία τοπικού χαρακτήρα κατά τη διάρκεια της εξέλιξής τους, ενώ χρειάζονται την εφαρμογή μιας πράξης συρρίκνωσης προκειμένου να αποφασιστεί εάν έχουν ολοκληρωθεί.

Οι επόμενες δύο διεργασίες, η χαλαρωτική ταξινόμηση και η εκλέπτυνση γραμμών, είναι αντιπροσωπευτικές μιας ευρύτερης κλάσης διεργασιών που χαρακτηρίζονται σαν "διεργασίες ψηφίσματος" (voting processes). Τα βασικά γνωρίσματα αυτής της κλάσης διεργασιών είναι ότι απαιτούν κάποια υπολογιστική ισχύ προκειμένου να αποφασίσει κάθε μονάδα δεδομένων την ή τις "ψήφους" της. Στη συνέχεια απαιτείται ένα επικοινωνιακό σχήμα που θα συλλέγει γρήγορα τις ψήφους αυτές. Ο κατάλληλος γράφος αρχιτεκτονικής για ένα τέτοιο σχήμα είναι ένα δένδρο ή η πεταλούδα (Butterfly). Μια βασική διαφορά στις δύο διεργασίες που

παρουσιάζονται εδώ είναι ότι στη μεν διεργασία του υπολογισμού του μετασχηματισμού Hough η συλλογή των "ψήφων" είναι κοινή για όλες τις μονάδες δεδομένων, ενώ στη διεργασία του υπολογισμού των ροπών περιοχών η συλλογή των "ψήφων" γίνεται ανά περιοχή. Αυτό σημαίνει ότι η αρχιτεκτονική στην οποία θα υλοποιηθεί η διεργασία θα πρέπει να έχει τη δυνατότητα να υλοποιεί τους γράφους ανεξάρτητα για κάθε ομάδα δεδομένων, δηλαδή να επιτρέπει την παράλληλη συλλογή των "ψήφων" για όλες τις ομάδες δεδομένων ταυτόχρονα. Διαφορετικά, η συλλογή θα πρέπει να γίνει διαδοχικά για κάθε ομάδα και θα έχει σαν αποτέλεσμα τη σειριοποίηση του αλγορίθμου. Μια διερεύνηση εναλλακτικών τρόπων υλοποίησης του προβλήματος αυτού στο Connection Machine γίνεται στην [22], όπου διατυπώνονται συγκεκριμένα κριτήρια επιλογής της καλύτερης υλοποίησης με βάση ορισμένα χαρακτηριστικά του περιεχομένου. Αλλη διεργασία που εμπίπτει στην κλάση των "διεργασιών ψηφίσματος" είναι ο υπολογισμός του ιστογράμματος μιας εικόνας, ο οποίος επίσης μπορεί να γίνει συνολικά για όλη την εικόνα ή τμηματικά ανά περιοχή. Για κάθε διεργασία δίνεται συνοπτικά η ανάλυση της υλοποίησή της, όπως έγινε και με τις δύο προηγούμενες. Στο τέλος παραθέτουμε πίνακες με τα ιδιαίτερα χαρακτηριστικά της κάθε μίας, όπως θα μπορούσαν να χρησιμοποιηθούν από το σύστημα που προτείνουμε.

6.4.1 Χαλαρωτική Ταξινόμηση (Relaxation Labeling)

Χαλαρωτική Ταξινόμηση είναι η διεργασία εκείνη που εκχωρεί μία ετικέτα l από ένα σύνολο L από πιθανές ετικέτες σε κάθε ένα από τα n αντικείμενα ενός συνόλου A αντικειμένων. Κάθε μία από τις ετικέτες αυτές αντιπροσωπεύει μία από τις κλάσεις στις οποίες θα ταξινομηθούν τα αντικείμενα που εμφανίζονται στην εικόνα. Η μέθοδος είναι επαναληπτική και χρησιμοποιεί ορισμένους συντελεστές συμβατότητας (Compatibility Coefficients) μεταξύ των ετικετών γειτονικών αντικειμένων, προκειμένου να εξαλείψει την ασάφεια που υπάρχει στην αρχική εκχώρηση ετικετών [178].

Οι συντελεστές αυτοί τους οποίους συμβολίζουμε με $c(i, j; h, k)$ συσχετίζουν το αντικείμενο i με G γειτονικά του και καθορίζει πόσο συμβατό είναι το γεγονός: (*Το αντικείμενο* **i** *ανήκει στην κλάση* **j**) με το γεγονός: (*Το αντικείμενο* **h** *ανήκει στην κλάση* **k**). Οι συντελεστές συμβατότητας υπολογίζονται με διάφορες μεθόδους ανάλογα με τις οποίες μπορεί να παίρνουν διάφορες τιμές. Εδώ θα θεωρήσουμε την περίπτωση όπου οι τιμές που παίρνουν είναι στο

διάστημα [-1,1]. Οι τιμές αυτές είναι πραγματικοί αριθμοί και το μέγεθός τους δείχνει το μέγεθος της συμβατότητας ή ασυμβατότητας τους. Έτσι αν ο συντελεστής είναι ίσος με 1 τότε είναι απόλυτα σίγουρη η γειτνίαση αντικειμένων που ανήκουν στις συγκεκριμένες κλάσεις, αν είναι 0 τότε δεν υπάρχει συσχέτιση μεταξύ των κλάσεων των δύο γειτονικών αντικειμένων, ενώ αν είναι -1 τότε είναι αδύνατη η γειτνίαση τέτοιων κλάσεων. Ενδιάμεσες τιμές αντιπροσωπεύουν ενδιάμεσες καταστάσεις.

Αν το πλήθος των πιθανών κλάσεων είναι L και p_{ij}^r είναι η πιθανότητα το αντικείμενο i να ανήκει στην κλάση j στην επανάληψη r, τότε η επαναληπτική διαδικασία καθορίζεται από τους επόμενους τύπους [178]:

$$p_{ij}^{r+1} = \frac{p_{ij}^r * (1 + q_{ij}^r)}{\sum_{j=1}^{L} p_{ij}^r * (1 + q_{ij}^r)}$$

όπου:

$$q_{ij}^r = \frac{1}{g} \sum_{h=1}^{G} \sum_{k=1}^{L} c(i,j;h,k) p_{hk}^r$$

Για κάθε αντικείμενο **i**, όταν η πιθανότητα p_{ij}^r για κάποιο **j** περάσει ένα ορισμένο όριο (π.χ το 0.95), εκχωρείται η ετικέτα **j** στο αντικείμενο **i** και σταματάει η επαναληπτική διαδικασία για αυτό το συγκεκριμένο αντικείμενο. Ο αλγόριθμος τελειώνει όταν ικανοποιηθεί κάποιο κριτήριο. Ενα τέτοιο είναι η ελαχιστοποίηση της εντροπίας που χαρακτηρίζει την αμφιβολία στις εκχωρήσεις [179]. Η διεργασία μπορεί να θεωρηθεί αντιπροσωπευτική μιας ευρύτερης ομάδας διεργασιών με ιδιαίτερα μεγάλες απαιτήσεις σε υπολογιστική ισχύ, ενώ η ενδοεπικοινωνία που απαιτείται είναι εν γένει τοπικού χαρακτήρα (μεταξύ των γειτονικών αντικειμένων).

Η μέθοδος βρίσκει πολλές εφαρμογές τόσο σε ανάλυση χαμηλού, όσο και σε ανάλυση μέσου και υψηλού επιπέδου. Τα αντικείμενα μπορεί να είναι τα σημεία (pixels) μιας εικόνας, ή τα διάφορα αντικείμενα που εμφανίζονται μέσα σ'αυτήν και τα οποία θα πρέπει να αναγνωρίσουμε. Οι κλάσεις μπορεί να είναι δύο, συνήθως οι (*αντικείμενο, φόντο*), ή περισσότερες (*αντικείμενο-1, αντικείμενο-2, ... , αντικείμενο-n*). Στην περίπτωση αυτή, η συσχέτιση μεταξύ των αντικειμένων έχει πιο πολύπλοκη μορφή και ως εκ τούτου ο γράφος του αλγορίθμου εξαρτάται από το πρόβλημα.

Υλοποίηση με μία μονάδα δεδομένων ανά επεξεργαστή

Η υλοποίηση του αλγορίθμου στην περίπτωση αυτή γίνεται ως εξής: Σε κάθε επεξεργαστή εκχωρούνται οι αρχικές ταξινομήσεις ενός αντικειμένου. Στη συνέχεια κάθε επεξεργαστής επικοινωνεί με τους G γειτονικούς του και ενημερώνεται για τις ταξινομήσεις τους. Ακολουθεί μια επαναληπτική διαδικασία σε κάθε επανάληψη της οποίας κάθε επεξεργαστής υπολογίζει τις νέες τιμές των πιθανοτήτων των ταξινομήσεων και στις συνέχεια τις ανταλλάσσει με τους γειτονικούς του.

Αν αναλύσουμε τους τύπους των προηγουμένων παραγράφων τότε προκύπτει πως η απαιτούμενη υπολογιστική ισχύς είναι:

$$\mathrm{T}^1_{comp} = I * L * (L * (G+1) * t^1_m + (2L+G) * t^1_a)$$

όπου I το πλήθος των επαναλήψεων που απαιτούνται, G το πλήθος των γειτονικών του αντικειμένων από τα οποία εξαρτάται το κάθε ένα, L το πλήθος των πιθανών κλάσεων και t^1_m ο χρόνος που απαιτεί η εκτέλεση ενός πολλαπλασιασμού πραγματικών αριθμών, και t^1_a είναι ο χρόνος για μία πρόσθεση. Ο πρώτος παράγοντας είναι συνάρτηση της ασάφειας που έχει το περιεχόμενο της εικόνας, ενώ οι άλλοι δύο εξαρτώνται από τη φύση του προβλήματος. Ο επικοινωνιακός χρόνος θα είναι:

$$\mathrm{T}^1_{comm} = I * L * G * t^1_{msg}$$

όπου t^1_{msg} ο χρόνος για τη μεταφορά ενός μηνύματος σε γειτονικό του επεξεργαστή.

Υλοποίηση με πολλές μονάδες δεδομένων ανά επεξεργαστή

Η διαφοροποίηση στην περίπτωση αυτή είναι ότι κάθε επεξεργαστής υπολογίζει τις νέες τιμές των πιθανοτήτων των εκχωρήσεων από εκείνες του προηγούμενου βήματος για όλες τις μονάδες δεδομένων που έχει στη διάθεσή του. Στη συνέχεια επικοινωνεί με γειτονικούς του και ενημερώνει για Q από τα αντικείμενά του τα οποία εξαρτώνται από αντικείμενα που υπάρχουν σε άλλους επεξεργαστές.

Αυτό που διαφοροποιείται στην περίπτωση αυτή όσον αφορά τον υπολογιστικό χρόνο είναι το πόσες μονάδες δεδομένων από τις N του προβλήματος έχει ο κάθε επεξεργαστής E. Η

απαιτούμενη υπολογιστική ισχύς στην περίπτωση αυτή θα είναι:

$$\mathrm{T}^{\mathrm{N}}_{comp} = \lceil \frac{\mathrm{N}}{E} \rceil * I * L * (L * (G+1) * t^N_m + (2L+G) * t^N_a)$$

Οσον αφορά στο χρόνο για επικοινωνία, αυτό που θα τον επηρεάσει είναι ο συνολικός αριθμός Q των γειτονικών επεξεργαστών με τους οποίους πρέπει να επικοινωνήσει ο κάθε ένας. Ετσι ο χρόνος αυτός θα είναι:

$$\mathrm{T}^{\mathrm{N}}_{comm} = I * L * Q * t^N_{msg}$$

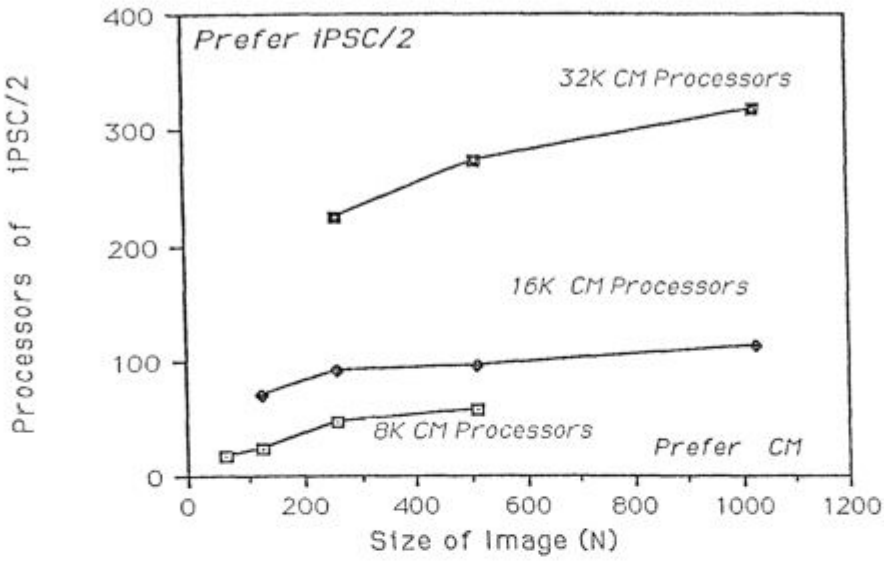
Ο αριθμός Q εξαρτάται από τον τρόπο που γίνεται η εναπόθεση (mapping) του γράφου που παριστάνει τη δομή του προβλήματος στο γράφο της αρχιτεκτονικής.

Επιλογές Υλοποίησης

Η δεύτερη υλοποίηση εξαρτάται από το πλήθος P των μονάδων δεδομένων στις οποίες πρέπει να γίνει η επεξεργασία. Ομως αντίστοιχα, θα πρέπει να σημειώσουμε πως στην πρώτη υλοποίηση θα πρέπει οι επεξεργαστές E να είναι ίσοι με το πλήθος P. Ο γράφος της διεργασίας εξαρτάται από τη φύση του προβλήματος. Αν η διαδικασία εφαρμόζεται σε εικόνα που παριστάνεται από έναν πίνακα PxP, τότε ο γράφος είναι αντίστοιχα ένας πίνακας από κόμβους όπου ο κάθε ένας συνδέεται με το γειτονικό του (nearest neighbor). Αν εφαρμόζεται για υψηλότερου επιπέδου ανάλυση εικόνας, τότε ο γράφος έχει προφανώς λιγότερους κόμβους, κάθε ένας από τους οποίους αντιπροσωπεύει ένα αντικείμενο στην εικόνα, το οποίο πρέπει να αναγνωριστεί. Το πλήθος των ακμών του γράφου εξαρτάται από το βαθμό αλληλεξάρτησης των αντικειμένων αυτών.

Στο Σχήμα 6.41 βλέπουμε τα αναλυτικά αποτελέσματα για την επιλογή αρχιτεκτονικής συναρτήσει του μεγέθους της εικόνας και του πλήθους των επεξεργαστών στο iPSC/2, όταν αντικείμενα είναι τα pixels και με την υπόθεση ότι επεξεργαζόμαστε όλα τα pixels μέχρι το τέλος της διαδικασίας. Κάθε σημείο της εικόνας αντιστοιχεί σε ένα πραγματικό ή ιδεατό επεξεργαστή του CM. Στο χώρο πάνω από κάθε καμπύλη είναι προτιμητέο το iPSC/2 ενώ από κάτω το CM. Στην περίπτωση αυτή παρατηρούμε το Connection Machine να έχει σοβαρό

προβάδισμα παρά τη χαμηλή υπολογιστική ισχύ των επεξεργαστών. Αυτό είναι αποτέλεσμα της υψηλής παραλληλοποιησιμότητας της διεργασίας. Αν όμως μεγάλος αριθμός σημείων φτάνει στην τελική κατάσταση νωρίτερα από άλλα, τότε θα περιμένουμε πως το iPSC/2 θα αρχίσει να αποδίδει καλύτερα αφού η χρησιμοποίηση (utilization) των επεξεργαστών του CM θα πέφτει.



Σχήμα 6.41: Επιλογή αρχιτεκτονικής για την υλοποίηση της διεργασίας της Χαλαρωτικής Ταξινόμησης

6.4.2 Εκλέπτυνση Γραμμών (Thinning)

Η διαδικασία αυτή εφαρμόζεται προκειμένου να λεπτύνουμε τις υπάρχουσες γραμμές ώστε το πάχος τους να είναι τελικά ένα σημείο. Το σχήμα που δημιουργείται τελικά λέγεται και *σκελετός* του αντικειμένου. Υπάρχει στη βιβλιογραφία μια πληθώρα αλγορίθμων για τη διαδικασία αυτή από τις οποίες επιλέγουμε να παρουσιάσουμε έναν κλασικό αλγόριθμο που παρουσιάζεται από τον Παυλίδη [180] και λειτουργεί επαναληπτικά ως εξής: Κάθε σημείο ελέγχει αν δημιουργεί με τα 8 γειτονικά του κάποιο από 6 συγκεκριμένα σχήματα (patterns). Τον έλεγχο αυτό επαναλαμβάνει 4 φορές κάθε φορά από τις οποίες εξετάζει τα σχήματα αυτά αφού τα περιστρέψει κατά γωνία 45 μοιρών. Ανάλογα με το αν βλέπει να δημιουργείται στη γειτονιά του κάποιο από αυτά, αποφασίζει αν είναι σημείο του σκελετού, αν θα πρέπει να σβηστεί ή αν θα πρέπει να συνεχίσει να συμμετέχει στη διαδικασία. Το πλήθος των επαναλήψεων ισούται

με τη μέγιστη ακτίνα R των κύκλων που μπορούν να εγγραφούν μέσα στα αντικείμενα που εμφανίζονται στην εικόνα.

Ο αλγόριθμος ανήκει στην ίδια κατηγορία με εκείνον της προγούμενης παραγράφου, δηλαδή απαιτεί σημαντική υπολογιστική ισχύ και τοπικού χαρακτήρα επικοινωνίες. Η ανάλυσή του έχει ως εξής:

$$\begin{aligned} \mathrm{T}_{comp} &= R * (t_a + 4(t_a + \frac{P^2}{E}(5t_a + \frac{\bar{B}}{E} * N * t_{match}) + 2t_{msg})) \\ &= R * (5 + 20\lceil\frac{P^2}{E}\rceil + 4\lceil\frac{P^2}{E^2}\rceil\bar{B} * N * A) * t_a + 8Rt_{msg} \end{aligned}$$

Οπου t_a είναι ο μέσος χρόνος για την εκτέλεση μιας πράξης, P το μέγεθος της εικόνας, $\bar{B}$ ο μέσος αριθμός σημείων που ανήκουν σε αντικείμενα κατά τη διάρκεια των R επαναλήψεων. N είναι το πλήθος των σχημάτων (patterns) με τα οποία γίνεται η σύγκριση και $t_{match} = \mathrm{A} * t_a$ ο χρόνος ελέγχου για το αν σχηματίζεται ένα από τα 6 γνωστά σχήματα. Τελος, t_{msg} είναι ο χρόνος επικοινωνίας μεταξύ γειτονικών επεξεργαστών. Το απαιτούμενο επικοινωνιακό σχήμα είναι εκείνο της επικοινωνίας με τον κοντινότερο γείτονα (nearest neighbor) πάνω σε ορθογωνικό πλέγμα.

Για υλοποίηση με μία μονάδα δεδομένων ανά επεξεργαστή η συνολική επίδοση του αλγορίθμου δίνεται από την επόμενη σχέση, όπου ο εκθέτης 1 υποδηλώνει την υλοποίηση με μία μονάδα δεδομένων ανά επεξεργαστή:

$$\mathrm{T}^1 = R * 4(6t_a^1 + N * t_{match} + 8 * t_{msg}^1)$$

Για την υλοποίησή του με πολλές μονάδες δεδομένων ανά επεξεργαστή δημιουργείται η σχέση:

$$\mathrm{T}^{\mathrm{N}} = R * (5 + 20\lceil\frac{P^2}{E}\rceil + 4\lceil\frac{P^2}{E^2}\rceil\bar{B} * N * A) * t_a^{\mathrm{N}} + 8Rt_{msg}^{\mathrm{N}}$$

Βλέπουμε ότι η απόδοση των αλγορίθμων εξαρτάται από τα τεχνικά χαραστηριστικά των

αρχιτεκτονικών (t_a, t_{msg}) και από την ακτίνα R. Επιπλέον η δεύτερη υλοποίηση εξαρτάται από το εμβαδόν των αντικειμένων, το οποίο εκφράζεται από το $\bar{B}$. Αν συγκρίνουμε τις δύο επιδόσεις, η ακτίνα R θα απαλειφθεί, οπότε η επιλογή των αρχιτεκτονικών θα γίνει κυρίως βάση των τεχνικών χαραστηριστικών και του παράγοντα $\bar{B}$.

6.4.3 Μετασχηματισμός Hough (Hough Transform)

Ο μετασχηματισμός Hough μετασχηματίζει την εικόνα από το πεδίο του χώρου σε ένα νέο πεδίο παραμέτρων, ανάλογα με τις οποίες μπορεί να διερευνά την ύπαρξη διαφόρων παραστάσεων (κύκλων, ελλείψεων, ευθειών, κτλ.). Στην παράγραφο αυτή θα ασχοληθούμε με τη χρήση του για τον εντοπισμό ευθειών μέσα στην εικόνα. Για το σκοπό αυτό υπολογίζεται η ακόλουθη παράσταση για κάθε σημείο της εικόνας:

$$\rho = x sin(\theta \pm \frac{\pi}{2}) - y cos(\theta \pm \frac{\pi}{2})$$

Η παράσταση υπολογίζεται για διάφορες τιμές της γωνίας θ, και κάθε μία από αυτές τις τιμές αποτελεί μία "ψήφο" του σημείου αυτού για την ευθεία που περνά από το σημείο (x,y) και έχει κατεύθυνση θ. Οποιο ζεύγος (ρ,θ) συλλέξει ικανοποιητικό αριθμό τέτοιων ψήφων αντιπροσωπεύει το μετασχηματισμό της αντίστοιχης ευθείας.

Η παράλληλη υλοποίηση, λοιπόν, του αλγορίθμου αυτού συνίσταται στον "εν παραλλήλω" υπολογισμό των ποσοτήτων αυτών για τις διάφορες τιμές της γωνίας θ. Για κάθε τέτοιο υπολογισμό καταχωρούνται οι τιμές του ρ που προκύπτουν για το κάθε σημείο. Οταν ολοκληρωθούν οι υπολογισμοί για όλες τις πιθανές γωνίες, από τις θέσεις που είναι καταχωρημένες οι "ψήφοι" για τα ζεύγη (ρ,θ) προκύπτει αν και ποιες ευθείες υπάρχουν.

Για υλοποίηση με μία μονάδα δεδομένων ανά επεξεργαστή κάθε επεξεργαστής υπολογίζει τη συνεισφορά του σημείου της εικόνας που κρατά για τις διάφορες γωνίες θ.

Υπάρχουν διάφοροι τρόποι υλοποίησης αυτής της διεργασίας ανάλογα με το πού θέλουμε να συγκεντρώσουμε τα δεδομένα εξόδου. Κατάλληλα σχήματα επικοινωνίας για τη διεργασία αυτή είναι εκείνα που συλλέγουν τις τιμές κάποιας μεταβλητής σε ένα σημείο (επεξεργαστή) του συστήματος. Τέτοια είναι το *Δένδρο* και η *Πεταλούδα* (Butterfly).

Για υλοποίηση της διεργασίας με πολλές μονάδες δεδομένων ανά επεξεργαστή, κάθε ένας αναλαμβάνει μια υποπεριοχή της εικόνας, δημιουργεί τους τοπικούς μετασχηματισμούς και στη συνέχεια όλοι οι επιμέρους μετασχηματισμοί συντίθενται προκειμένου να δώσουν τον τελικό. Η επίδοση του αλγορίθμου στην περίπτωση αυτή εξαρτάται από το πλήθος των σημείων που υπάρχουν πάνω στην εικόνα, το πλήθος των γωνιών θ για τις οποίες διερευνάται η ύπαρξη ευθειών, ο αριθμός των επεξεργαστών στους οποίους διαμοιράζεται η εικόνα, και τέλος τα τεχνικά χαρακτηριστικά της αρχιτεκτονικής, μεταξύ των οποίων είναι και τα επικοινωνιακά σχήματα που υποστηρίζει αποτελεσματικά.

6.4.4 Ροπές περιοχών (Region Moments)

Η διαδικασία αυτή υπολογίζει τις ροπές αδράνειας των επιφανειών των αντικειμένων που εμφανίζονται πάνω στην εικόνα. Ενας βασικός τύπος υπολογισμού τους είναι ο εξής:

$$m_{kl} = \sum_{x} \sum_{y} x^k y^l \mathrm{I}(x, y)$$

με x,y τις συντεταγμένες κάθε σημείου της εικόνας I. Ανάλογα με την εφαρμογή θα πρέπει να καθορίσουμε την τάξη των ροπών που θα πρέπει να υπολογίσουμε. Τάξη της ροπής είναι το άθροισμα $n=k+l$. Οταν λέμε ροπές τάξης n εννοούμε την εφαρμογή του παραπάνω τύπου, με όλους εκείνους τους συνδιασμούς των k,l για τους οποίους ισχύει $k+l < n$.

Η διαδικασία αυτή απαιτεί ισχυρή υπολογιστική ισχύ ενώ δεν απαιτεί ενδοεπικοινωνία μεταξύ των επεξεργαστών, παρά μόνο για τη συλλογή και άθροιση των επιμέρους αποτελεσμάτων χωριστά για κάθε αντικείμενο στην εικόνα.

Τα επικοινωνιακά σχήματα που απαιτεί είναι τα ίδια με εκείνα της προηγούμενης διεργασίας του μετασχηματισμού Hough. Η διαφορά εδώ είναι πως μπορεί να έχουμε διάφορα αντικείμενα για τα οποία θα πρέπει να υπολογίσουμε χωριστά τις ροπές. Στην περίπτωση αυτή υπάρχουν διάφοροι τρόποι υπολογισμού των ροπών, ένας από τους οποίους είναι να υπολογιστούν διαδοχικά οι ροπές για κάθε ένα αντικείμενο ενώ ένας άλλος είναι να αναδιαταχτούν τα σημεία των αντικειμένων ώστε να υπολογιστούν παράλληλα οι ροπές τους εκμεταλλευόμενοι

συγκεκριμένες δυνατότητες του συστήματος [19].

Για υλοποίηση με μία μονάδα δεδομένων ανά επεξεργαστή, κάθε επεξεργαστής αναλαμβάνει από ένα σημείο και υπολογίζει τη συνεισφορά του στις ροπές του αντικειμένου στο οποίο ανήκει. Αντίθετα για υλοποίηση με πολλές μονάδες δεδομένων ανά επεξεργαστή, κάθε ένας αναλαμβάνει μια υποπεριοχή της εικόνας και υπολογίζει, για κάθε ένα σημείο που βρίσκεται σ'αυτή, τη συνεισφορά του στις ροπές του αντικειμένου στο οποίο ανήκει. Ενας άλλος τρόπος υλοποίησης είναι να αναλάβει κάθε επεξεργαστής τον υπολογισμό όλων των ροπών ενός αντικειμένου. Η υλοποίηση αυτή προϋποθέτει ανακατανομή των δεδομένων, έτσι ώστε τα σημεία του ίδιου αντικειμένου να βρίσκονται στον ίδιο επεξεργαστή. Αυτό εξαρτάται από τη σχέση του αριθμού των αντικειμένων με το πλήθος των επεξεργαστών.

Η επιλογή της βέλτιστης υλοποίησης θα εξαρτηθεί από τα τεχνικά χαρακτηριστικά των υποψήφιων αρχιτεκτονικών, το πλήθος των αντικειμένων που εμφανίζονται στην εικόνα, το πλήθος των σημείων που τα αποτελούν και τον αριθμό των επεξεργαστών του συστήματος αλλά και τον αριθμό των ροπών που θέλουμε να υπολογίσουμε.

6.4.5 Τα χαρακτηριστικά των διεργασιών και των εικόνων

Από την ανάλυση των αλγορίθμων των διεργασιών που παρουσιάσαμε στις προηγούμενες παραγράφους, δημιουργήσαμε τους επόμενους πίνακες, που αναφέρονται τόσο στα χαρακτηριστικά των αλγορίθμων, όσο και στην εξάρτησή τους από συγκεκριμένα χαρακτηριστικά του περιεχόμενου της εικόνας.

Στον πίνακα 6.1 βλέπουμε τα χαρακτηριστικά των διεργασιών, στις οποίες έχουμε αναφερθεί μέχρι τώρα. Για κάθε διεργασία υπάρχει μία γραμμή που αντιστοιχεί σε 1 Μονάδα δεδομένων ανά επεξεργαστή και μία άλλη που αντιστοιχεί σε πολλές (N) Μονάδες δεδομένων ανά επεξεργαστή. Ειδικά για τη διεργασία του κυρτού περιγράμματος (CH) και των συνδεδεμένων συστατικών (CC) οι δείκτες A και B που υπάρχουν στη στήλη αυτή υποδηλώνουν τις αντίστοιχες δύο υλοποιήσεις. Στη στήλη της απαίτησης σε μνήμη δίνεται η αντίστοιχη πολυπλοκότητα. Στη στήλη του λόγου $\frac{t_{comp}}{t_{comm}}$, δηλαδή του χρόνου που απαιτείται για υπολογισμούς προς το χρόνο που απαιτείται για επικοινωνία, οι διαβαθμίσεις D,C,B και A αντιστοιχούν στις τιμές

(*0.00-0.25, 0.25-0.50, 0.50-0.75 και 0.75-1.00*). Στα σχήματα επικοινωνίας (Communication Patterns) ο συμβολισμός που ακολουθείται είναι ο εξής: *Tree* είναι το δένδρο πάνω σε όλους τους επεξεργαστές του συστήματος. *Pref* είναι η προθεματική πράξη σάρωσης πάνω σε όλους τους επεξεργαστές. Τέλος *nn* είναι το σχήμα επικοινωνίας με τον κοντινότερο γείτονα του κάθε επεξεργαστή (nearest neighbor). Οταν στα παραπάνω σχήματα ο συμβολισμός ακολουθείται από το *-X*, σημαίνει ότι οι επεξεργαστές είναι διατεταγμένοι σε ορθογωνικό πλέγμα και το σχήμα αυτό υλοποιείται σε κάθε γραμμή του. Το πρόθεμα *cl-* υποδηλώνει ότι το αντίστοιχο επικοινωνιακό σχήμα θα πρέπει να έχει τη δυνατότητα να εφαρμοστεί χωριστά για διαφορετικές ομάδες (clusters) μονάδων δεδομένων. *Problem Graph* σημαίνει ότι εξαρτάται από το γράφο της συγκεκριμένης εφαρμογής και μπορεί να είναι ο,τιδήποτε.

Features⇒		Memory	$\frac{t_{comp}}{t_{comm}}$	Comm.	Data	Data	Paral-	Cont.
Task⇓	$\frac{DU}{proc}$			Pattern	Unit	Struct.	lelism	Depend.
CH	1	D	C	cl-Tree-X	Pixel	2D-Array	B	Yes
		O(1)		nn-X		1D-Array	B	Yes
	N_A	O($\frac{N}{E}$)	C	Tree	Pixel	List	B	Yes
	N_B	O($\frac{P^2}{E}$)	B	Tree	Pixel	Array List	B	Yes
CC.	1_A	O(1)	D	cl-Pref-X	Pixel	Array	A	Yes
	1_B	O(P)	D	nn-X	Pixel	Array	A	Yes
	N	O($\frac{P^2}{E}$)	A	Tree	Pixel	Array	C	Yes
RL.	1	O(1)	B	Pr.Graph	Object	Pr.Graph	A	Yes
	N	O($\frac{P^2}{E}$)	A	Pr.Graph	Object	Pr.Graph	A	Yes
Hough	1	O(1)	B	Tree	Pixel	Free	B	Yes
	N	O($\frac{N}{E}$)	B	Tree	Pixel	Free	C	Yes
Moments	1	O(1)	A	cl-Tree	Pixel	Free	B	Yes
	N	O($\frac{P^2}{E}$)	A	cl-Tree	Pixel	Free	C	Yes
Thinning	1	O(1)	A	nn	Pixel	Array	A	Yes
	N	O($\frac{P^2}{E}$)	B	nn	Pixel	Array	A	Yes

Πίνακας 6.1: Βάση πληροφοριών με τα χαρακτηριστικά των διεργασιών

Στον πίνακα 6.2 βλέπουμε την εξάρτηση κάθε μίας από τις διεργασίες αυτές από συγκεκριμένα χαρακτηριστικά του περιεχόμενου της εικόνας. Τα ιδιαίτερα χαρακτηριστικά του περιεχομένου της εικόνας που αναφέρονται στον πίνακα αυτό είναι το πλήθος των σημείων που ανήκουν σε ένα αντικείμενο *(#Pix-obj)*, το πλήθος των σημείων που βρίσκονται στο περίγραμμα ενός αντικειμένου *(#Pix-contour)*, το άθροισμα των σημείων του περιγράμματος που ανήκουν σε κοίλα τμήματά του (Σcurvature), ο μέγιστος αριθμός σημείων που ανήκουν σε ένα κοίλο τμήμα *max(Curvature)*, και η μέγιστη πολυπλοκότητα του σχήματος (Shape Complexity), όπως αυτή περιγράφεται στο τμήμα 4.4, η διάμετρος Manhattan, η μέγιστη ακτίνα R κύκλου που μπορεί να εγγραφεί στα αντικείμενα της εικόνας, το μέγιστο πλήθος τομών μεταξύ αντικειμένων και των ορίων των υποπεριοχών στις οποίες χωρίζεται μια εικόνα. Ακόμα, η διακριτική ικανότητα που περιμένουμε να έχει μια εικόνα ως προς τις διαβαθμίσεις των γωνιών (#θ) των ευθειών που αναπαριστά και των θέσεών τους (#ρ). Επίσης υπάρχουν ποιοτικά χαρακτηριστικά τα οποία επηρεάζουν την απόδοση των αλγορίθμων, όπως η αρχική αμφιβολία (*Initial Ambiguity*) που εμπεριέχει η αρχική ταξινόμηση των αντικειμένων της, ο βαθμός της αλληλεξάρτησής τους (*Interdependence*) καθώς επίσης η πολυπλοκότητα που χαρακτηρίζει την εικόνα σαν αναπαράσταση ενός θέματος (*Scene Complexity*). Παρατηρούμε ότι οι περισσότεροι αλγόριθμοι εξαρτώνται από το πλήθος των αντικειμένων που εμφανίζονται στην εικόνα ενώ δεν έχουν ίδια εξάρτηση οι αλγόριθμοι της διεργασίας που υλοποιούνται με λεπτή (fine) και αδρή (coarse) διαμέριση.

6.5 Συμπεράσματα

Στο κεφάλαιο αυτό επιδείχθηκε ο τρόπος εμπλουτισμού των βάσεων πληροφοριών του μοντέλου επιλογής παράλληλων υλοποιήσεων διεργασιών ανάλυσης εικόνων. Συγκεκριμένα, περιγράψαμε τον τρόπο ανάλυσης των αλγορίθμων προκειμένου να προσθέσουμε γνώση στο τμήμα του μοντέλου που αφορά την περιγραφή τους. Μέσα από τις διαφορετικές υλοποιήσεις για διάφορες ομάδες διεργασιών φάνηκαν τα αποτελέσματα της αλληλεπίδρασης μεταξύ των συνιστωσών του προβλήματος. Έγινε φανερό τόσο από τα αναλυτικά όσο και από τα πειραματικά αποτελέσματα πως ένας μικρός αριθμός παραμέτρων είναι δυνατόν να καθορίσει τη υλοποίηση με την καλλίτερη απόδοση μεταξύ γνωστών υλοποιήσεων για διάφορες διεργασίες επεξεργασίας εικόνων.

Task ⇒	*CH*			*CC*			*RL*		*Hough*		*Moments*		*Thinn.*	
Features ⇓	1	N_A	N_B	1_A	1_B	N	1	N	1	N	1	N	1	N
#Pix-obj						*		*		*		*		*
#Pix-contour		*												
#Objects	*	*	*	*		*	*	*			*	*		
Σ(Curvature)		*	*											
max(Curvature)	*													
Shape Compl.				*										
Manhattan D.					*									
$\mathcal{R}$													*	*
Local Equal.						*								
$\#\theta$									*	*				
$\#\rho$									*	*				
Init. Ambig.							*	*						
Interdepend.							*	*						
Scene Compl.							*	*						

Πίνακας 6.2: Βάση πληροφοριών για την εξάρτηση αλγορίθμων από χαρακτηριστικά του περιεχόμενου των εικόνων

Κεφάλαιο 7

Επίλογος

7.1 Συζήτηση και Συμπεράσματα

Από την αναλυτική παρουσίαση των παραγόντων που εμπλέκονται στην παράλληλη υλοποίηση διεργασιών ανάλυσης εικόνων στα προηγούμενα κεφάλαια, γίνεται φανερό πως η βέλτιστη υλοποίησή τους, αποτελεί ένα πολύπλοκο και δυσεπίλυτο πρόβλημα. Οι σχετικές παράμετροι είναι πολλές, οι τιμές των οποίων μάλιστα δεν είναι πάντα γνωστές. Οι εργασίες που εμφανίζονται στη βιβλιογραφία δίνουν έμφαση, η κάθε μία χωριστά, στις επιμέρους συνιστώσες του προβλήματος. Ποικίλες αρχιτεκτονικές με διάφορα χαρακτηριστικά και δυνατότητες η κάθε μία σχεδιάζονται συχνά για να αντιμετωπίσουν ένα συγκεκριμένο πρόβλημα. Αναπτύσσονται αλγόριθμοι για τη μία ή την άλλη επιμέρους διεργασία ανάλυσης εικόνων ή σχεδιάζονται αλγόριθμοι ισοκατανομής φορτίου που επιτυγχάνουν κατά περίπτωση βελτιστοποίηση της απόδοσης. Ομως το πρόβλημα της ανάλυσης εικόνων δεν εστιάζεται στην υλοποίηση ενός ή δύο επιμέρους διεργασιών. Οι ολοκληρωμένες εφαρμογές που απαιτούν υψηλές ταχύτητες είναι εκείνες για τις οποίες έχει ουσιαστική έννοια και διερευνάται η χρήση της παραλληλίας. Αυτές, λοιπόν, οι ολοκληρωμένες εφαρμογές συνίστανται κατά κανόνα από μια ακολουθία (ή πλέγμα) διεργασιών, γεγονός που όχι μόνο δεν θα πρέπει να αγνοηθεί, αλλά προσθέτει ένα ακόμα βαθμό πολυπλοκότητας στο πρόβλημα. Η συνέχεια που πρέπει να εξασφαλίζεται μεταξύ των διεργασιών και ο βαθμός στον οποίο αυτές είναι συμβατές μεταξύ τους δημιουργούν πρόσθετα προβλήματα. Αυτό που δεν φαίνεται να έχει διερευνηθεί ιδιαίτερα μέχρι τώρα είναι ποιες είναι

οι βασικές συνιστώσες που συνθέτουν το πρόβλημα, πώς αυτές αλληλοεπηρεάζονται ανάλογα με τα επιμέρους χαρακτηριστικά τους, και τι επίπτωση έχει η αλληλεπίδραση αυτή στη συνολική απόδοση του συστήματος. Ερωτήματα όπως ποιοι αλγόριθμοι είναι κατάλληλοι για κάποιες διαθέσιμες αρχιτεκτονικές, ή τι χαρακτηριστικά πρέπει να έχει η αρχιτεκτονική προκειμένου να έχει καλή απόδοση ο αλγόριθμος που θέλουμε να υλοποιήσουμε για κάποια συγκεκριμένη διεργασία στερούνται συστηματικής απαντήσεως. Εκείνο μάλιστα το οποίο φαίνεται να έχει μελετηθεί λιγότερο από όλα είναι το περιεχόμενο της εικόνας και η σχέση του με την επίδοση ενός συστήματος. Μια τέτοια ανάλυση ενταγμένη στον τρόπο λειτουργίας ενός παράλληλου συστήματος, όπως φαίνεται από τα αναλυτικά και πειραματικά αποτελέσματα του έκτου κεφαλαίου, θα μπορούσε να αυξήσει σημαντικά την απόδοση ενός παράλληλου συστήματος.

Στην εργασία αυτή προσεγγίσαμε αναλυτικά και πειραματικά τη σχέση και την αλληλεξάρτηση των βασικών συνιστωσών που, σύμφωνα με την εμπειρία μας και την διερεύνηση που κάναμε, κρίθηκαν αρκετές ώστε να περιγράψουν το πρόβλημα σε όλες του τις διαστάσεις. Σαν πρώτο βήμα, καθορίστηκαν τα βασικά χαρακτηριστικά κάθε μίας από τις τέσσερις συνιστώσες του προβλήματος, οι οποίες είναι οι αλγόριθμοι υλοποίησης των διεργασιών, οι αρχιτεκτονικές, το περιεχόμενο των εικόνων και οι αλγόριθμοι ισοκατανομής του φορτίου. Στη συνέχεια, χρησιμοποιήσαμε τα χαρακτηριστικά αυτά σε μια συστηματική αντιμετώπιση του προβλήματος της πρόβλεψης της απόδοσης των παράλληλων υλοποιήσεων διεργασιών ανάλυσης εικόνων. Παρουσιάσαμε ένα μοντέλο κατάλληλο για την επιλογή της βέλτιστης μεταξύ πολλών εναλλακτικών παράλληλων υλοποιήσεων, το οποίο είναι κατάλληλο για ένα ευρύ φάσμα διεργασιών, αρχιτεκτονικών, εικόνων και αλγορίθμων ισοκατανομής του φορτίου.

Προκειμένου να διερευνήσουμε την αλληλεπίδραση των συνιστωσών μιάς παράλληλης υλοποίησης, την επίπτωση που θα έχει στην συνολική απόδοση και τη δυνατότητα να προβλέπουμε την απόδοση αυτή, πειραματιστήκαμε με υλοποιήσεις διεργασιών ανάλυσης εικόνων που εμπίπτουν στην κατηγορία του μέσου επιπέδου. Οι διεργασίες μέσου επιπέδου παρουσιάζουν διάφορες ιδιαιτερότητες, όπως το γεγονός ότι απαιτούν πολύπλοκα σχήματα επικοινωνίας, πολλές φορές όχι ομοιογενή σε όλη τη δομή των δεδομένων, οι υπολογιστικές τους απαιτήσεις εξαρτώνται από το περιεχόμενο της εικόνας ενώ υλοποιούνται σε δομές δεδομένων πολύπλοκες και δυναμικά εξελισσόμενες σε αντίθεση με την κανονική δομή του πίνακα (array). Οι αρχιτεκτονικές στις οποίες αναφέρεται η εργασία αυτή είναι τύπου SIMD με κατανεμημένη

μνήμη.

Οι διεργασίες που χρησιμοποιήθηκαν σαν παραδείγματα, παρουσιάζουν διάφορα υπολογιστικά και επικοινωνιακά χαρακτηριστικά τα οποία τις κάνουν αντιπροσωπευτικές μεγάλων ομάδων διεργασιών. Ορισμένοι από τους αλγόριθμους που υλοποιήθηκαν είναι σχεδιασμένοι εξαρχής, ενώ άλλοι αποτελούν την παράλληλη υλοποίηση γνωστών σειριακών αλγορίθμων ή τέλος είναι γνωστοί παράλληλοι αλγόριθμοι δανεισμένοι από τη βιβλιογραφία.

Ιδιαίτερα, διερευνήθηκαν σημαντικά χαρακτηριστικά του περιεχομένου της εικόνας και πώς καθένα από αυτά επηρεάζει την απόδοση του κάθε αλγορίθμου. Παρουσιάζει ενδιαφέρον ο διαφορετικός τρόπος με τον οποίο το κάθε χαρακτηριστικό επιδρά στην απόδοση των αλγορίθμων αλλά και το γεγονός ότι χαρακτηριστικά με σημαντικές επιπτώσεις σε έναν αλγόριθμο έχουν ουδέτερο ρόλο σε άλλο. Το γεγονός αυτό μας επιτρέπει, γνωρίζοντας το περιεχόμενο μιας εικόνας, να μπορούμε να κάνουμε εκ των προτέρων επιλογή του αλγορίθμου που θα χρησιμοποιήσουμε, μεταξύ αλγορίθμων για την ίδια διεργασία. Διαμορφώνονται επίσης κλειστοί τύποι, οι οποίοι, υπό συγκεκριμένες συνθήκες, μπορούν να εισηγούνται ποια αρχιτεκτονική είναι εκείνη με την καλύτερη αναμενόμενη απόδοση ή τέλος να υποδεικνύουν τη συγκεκριμένη υλοποίηση.

Τα συμπεράσματα που βγαίνουν από τα πειραματικά και αναλυτικά αποτελέσματα του πέμπτου και έκτου κεφαλαίου μας δημιουργούν την πεποίθηση πως είναι δυνατόν να μπει τάξη στη χαοτική φύση του προβλήματος και ότι είμαστε σε θέση να παίρνουμε αποφάσεις βασισμένες σε ένα όχι μεγάλο αριθμό παραγόντων, τους οποίους γνωρίζουμε ή για τους οποίους μπορούμε να κάνουμε υποθέσεις, με δραστικές τελικά επιπτώσεις στην επίδοση των παράλληλων υλοποιήσεων.

Τα συμπεράσματα αυτά θα αποκτήσουν ιδιαίτερο ενδιαφέρον αν συμπληρωθούν με την αντίστοιχη μελέτη των αλγορίθμων ανακατανομής του φορτίου στην οποία δεν αναφέρεται εκτεταμένα η διατριβή αυτή. Επίσης το μοντέλο μπορεί να επεκταθεί ανάλογα σε αρχιτεκτονικές MIMD καθώς επίσης σε αρχιτεκτονικές με διαφορετικά μοντέλα μνήμης όπως η PRAM (Parallel Random Access Memory), η κοινή μνήμη (shared memory) κ.ά. Η μέθοδος που περιγράψαμε, αναμένεται να αποδώσει εξίσου καλά, ανεξάρτητα από την εκάστοτε χρησιμοποιούμενη αρχιτεκτονική, αρκεί κάθε αρχιτεκτονική να περιγράφεται κατάλληλα. Επίσης, δεν επηρεά-

ζεται από τον τρόπο διαμέρισης των αρχικών δεδομένων στους επεξεργαστές (π.χ. λωρίδα, ορθογώνιες περιοχές κ.τ.λ.) χωρίς να αναφερόμαστε εδώ στις συνέπειες που η διαμέριση αυτή μπορεί να έχει στην άνιση αρχική κατανομή του φορτίου. Αυτό που αλλάζει στην περίπτωση των διαφορετικών διαμερίσεων, είναι πιθανά ο γράφος των απαιτούμενων επικοινωνιών ο οποίος θα πρέπει να φροντίζουμε να περιγράφεται σωστά και να είναι τέτοιος που να μπορεί να εξυπηρετηθεί από τον γράφο(-ους) επικοινωνίας της αρχιτεκτονικής.

Εντούτοις, υπάρχουν κατηγορίες διεργασιών ανάλυσης εικόνων στο μέσο επίπεδο για τις οποίες δεν είναι προφανής ο τρόπος που θα μπορούσε να εφαρμοστεί η προσέγγισή μας ώστε να δώσει ικανοποιητικά αποτελέσματα. Τέτοιες διεργασίες είναι εκείνες στις οποίες οι υπολογιστικές αλλά κυρίως οι επικοινωνιακές απαιτήσεις έχουν έντονα δυναμικό χαρακτήρα και ως εκ τούτου δεν είναι προβλέψιμες ή περιγράψιμες με αποτελεσματικό τρόπο εξ αρχής. Ακόμα δεν αντιμετωπίζεται αποτελεσματικά η περίπτωση να μην υπάρχει ικανοποιητικά ακριβής περιγραφή του περιεχομένου της εικόνας.

Η ανάπτυξη όλων αυτών των τεχνικών και μεθόδων καθώς και η συλλογή πειραματικών δεδομένων και συμπερασμάτων, έχουμε την πεποίθηση πως συνεισφέρει σημαντικά στην κατεύθυνση ανάπτυξης μιας γενικότερης μεθοδολογίας. Μια τέτοια μεθοδολογία θα ήταν χρήσιμη από πρακτική άποψη, εφόσον θα μπορούσε να χρησιμοποιηθεί για την υλοποίηση διεργασιών ανάλυσης εικόνων σε αναδιατασσόμενες (reconfigurable) αρχιτεκτονικές που θα υποστήριζαν υπολογισμούς τόσο λεπτής όσο και αδρής διαμέρισης, καθώς επίσης την αποτελεσματική εφαρμογή ποικίλων επικοινωνιακών σχημάτων αλλά και τον ευέλικτο μετασχηματισμό από μία δομή δεδομένων σε άλλη.

7.2 Ερευνητικές Επεκτάσεις

7.2.1 Γενικές κατευθύνσεις

Στην εργασία αυτή δόθηκε έμφαση στη μελέτη της αλληλεπίδρασης τριών από τις βασικές συνιστώσες που συνθέτουν το πρόβλημα της αποδοτικής παράλληλης υλοποίησης διεργασιών ανάλυσης εικόνων. Ενα από τα επόμενα βήματα προς την κατεύθυνση της ολοκλήρωσης της

μεθόδου είναι η ανάλογη ανάλυση και διερεύνηση της τέταρτης συνιστώσας του προβλήματος, δηλαδή των αλγορίθμων ισοκατανομής του φορτίου. Για το σκοπό αυτό, θα πρέπει να καθοριστούν μέθοδοι ορισμού του φορτίου ενός επεξεργαστή σε σχέση με τη διεργασία και το περιεχόμενο της εικόνας. Ερωτήματα όπως τι είναι υπολογιστικό φορτίο για μια συγκεκριμένη διεργασία, πώς μπορεί να μεταφέρεται αποτελεσματικά μεταξύ των επεξεργαστών, ποια είναι τα προβλήματα που δημιουργούνται αν το υπερδιασπάσουμε (overfragmentation) προκειμένου να το διαμοιράσουμε σε μεγάλο αριθμό επεξεργαστών κ.ά. είναι μεταξύ εκείνων που θα πρέπει να διερευνηθούν προκειμένου να αντιμετωπιστεί το πρόβλημα πιο ολοκληρωμένα. Η απάντηση σε αυτά τα ερωτήματα είναι επίσης σημαντική προκειμένου να εφαρμοστούν αποδοτικές τεχνικές ανακατανομής του υπολογιστικού φορτίου στους επεξεργαστές αλλά και για να δημιουργηθούν κανόνες και κριτήρια κατάλληλα για την εκ των προτέρων πρόβλεψη της απόδοσης τέτοιων τεχνικών.

Είναι προφανές πως η λειτουργία ενός τέτοιου συστήματος θα προϋπέθετε τον αποτελεσματικό σχεδιασμό των μηχανισμών που θα υλοποιούν το δέντρο των αποφάσεων, ενώ είναι αναγκαίος ο εμπλουτισμός του μοντέλου επιλογής υλοποιήσεων με μεγάλο αριθμό αλγορίθμων και υλοποιήσεων διαφόρων διεργασιών. Παρ'όλα αυτά, πολλές φορές δεν είναι δυνατόν να γνωρίζουμε επακριβώς τις τιμές όλων των γνωρισμάτων από τα οποία εξαρτάται η απόδοση της παράλληλης υλοποίησης, και ως εκ τούτου ένας τέτοιος μηχανισμός θα είναι ατελής, καθώς η συμπεριφορά ενός παράλληλου συστήματος φαίνεται να παρουσιάζει νεφελώδη (Fuzzy) στατιστική απόδοση.

Στην επόμενη παράγραφο θα περιγράψουμε συνοπτικά το σχεδιασμό του μοντέλου ενός *Εμπειρου Συστήματος* (Expert System) το οποίο θα αποφασίζει τον τρόπο υλοποίησης παράλληλων διεργασιών ανάλυσης εικόνων προκειμένου να επιτευχθεί η βέλτιστη απόδοσή τους. Πιστεύουμε πως το δέντρο των αποφάσεων θα μπορεί να κάνει μια πρώτη προσέγγιση στη βέλτιστη υλοποίηση, ενώ στη συνέχεια το Εμπειρο Σύστημα θα περιγράφει πολύ γρήγορα και με μεγαλύτερη ακρίβεια την τελική εισήγηση. Η μέθοδος του πέμπτου κεφαλαίου εμπλουτίζεται ως εκ τούτου, εκτός των άλλων, με ανάδραση. Βεβαίως, ένα τέτοιο σύστημα προϋποθέτει τη συλλογή μεγάλου αριθμού δεδομένων από υλοποιήσεις και από πειραματικά αποτελέσματα.

7.2.2 Ενα μοντέλο Εμπειρου Συστήματος για αποδοτική Παράλληλη Ανάλυση Εικόνων

Το σύστημα που προτείνεται θα αποτελείται από τρία βασικά τμήματα: Τον **Παραγωγό Κανόνων** (Rule Generator), που επεξεργάζεται την εμπειρία που καταχωρείται σχετικά με τις επιλογές υλοποίησης που έχουν γίνει για διάφορους συνδυασμούς εικόνων, διεργασιών και αρχιτεκτονικών και παράγει νέους ή βελτιώνει υπάρχοντες κανόνες. Οι κανόνες παριστάνονται ως Κανόνες Παραγωγής (Production Rules). Το **Λήπτη Αποφάσεων** (Decision Maker), ο οποίος, βασιζόμενος στη Βάση Κανόνων και στα δεδομένα του "προβήματος" που του τίθεται, αναζητά τον κανόνα με τη βέλτιστη εισήγηση υλοποίησης. Ως **πρόβλημα** μπορεί να οριστεί ο συνδυασμός της εικόνας και της επιδιωκόμενης επεξεργασίας. Δεχόμαστε ότι εικόνα και επεξεργασία μπορούν να περιγραφούν από μία σειρά από γνωρίσματα, με πεδία τιμών που μπορεί να είναι *συνεχή*, *διακριτά* ή *δομημένα*, και κάθε ένα από τα οποία αποτελεί μια διάσταση στο χώρο του προβλήματος. Αντίστοιχα, τα γνωρίσματα για τα οποία θα γίνει η εισήγηση αποτελούν τις διαστάσεις στο χώρο λύσης. Τέλος το σύστημα περιέχει και έναν **Εκτιμητή** (Evaluator), που κρίνει την προηγούμενη απόφαση με βάση την εκ των υστέρων μέτρηση της επίδοση της εφαρμογής της. Για τον τελευταίο χρειάζεται ο ορισμός μιας αντικειμενικής συνάρτησης. Ως πρώτη προσέγγιση μπορούν να χρησιμοποιηθούν μονοδιάστατα μεγέθη, όπως ο συνολικός χρόνος εκτέλεσης, η επιτάχυνση ή η απόδοση του συστήματος. Ο Παραγωγός Κανόνων και ο Εκτιμητής ενεργοποιούνται περιοδικά, όχι κατά τη διάρκεια της κύριας διεργασίας, προκειμένου να βελτιώσουν και να εμπλουτίσουν τη βάση κανόνων. Ο Λήπτης Αποφάσεων ενεργοποιείται δυναμικά κατά τη διάρκεια της επεξεργασίας και ως εκ τούτου ο χρόνος εκτέλεσής του πρέπει να είναι ο ελάχιστος δυνατός σε σχέση με τη συνολική διάρκεια της εκτέλεσης της διεργασίας ανάλυσης της εικόνας.

Το Εμπειρο Σύστημα αρχικά τροφοδοτείται με *εκπαιδευτικά στιγμιότυπα* και παράγει κανόνες. Η δομή των κανόνων πρέπει να λαμβάνει υπόψη το γεγονός ότι η συνθήκη (**IF part**) στους κανόνες παραγωγής μπορεί να αναφέρεται όχι μόνο στις διαστάσεις του προβλήματος, αλλά και σε εκείνες της λύσης, εφόσον αυτές είναι συχνά συσχετισμένες και αλληλεξαρτώμενες, όπως για παράδειγμα στην περίπτωση που έχουμε τη δέσμευση να χρησιμοποιήσουμε κάποια συγκεκριμένη αρχιτεκτονική. Η **συνθήκη** λοιπόν των κανόνων παραγωγής ορίζει έναν υποχώρο του χώρου στιγμιοτύπων που είναι το καρτεσιανό γινόμενο υποσυνόλων των πεδίων τιμών των

διαστάσεων του προβλήματος και της λύσης. Ενα τέτοιο καρτεσιανό γινόμενο ονομάζεται ομάδα (cluster). Η απουσία μίας διάστασης απο την πρόταση της συνθήκης σημαίνει ότι η διάσταση είναι αδιάφορη για την επιλογή που εισηγείται ο συγκεκριμένος κανόνας και είναι ισοδύναμο με το να θεωρούμε ως υποσύνολο αποδεκτών τιμών ολόκληρη τη διάσταση.

Με δεδομένο το πλαίσιο αυτό, προχωρούμε στη διατύπωση δύο βασικών προβλημάτων:

Πρόβλημα 1: Δοθέντος ενός συνόλου εκπαιδευτικών στιγμιοτύπων υλοποιήσεων, να οριστούν ομάδες στο χώρο στιγμιοτύπων που να αντιστοιχούν στις διάφορες επιλογές λύσεων (πρόβλημα Παραγωγού Κανόνων).

Πρόβλημα 2: Δοθείσης μιας βάσης κανόνων, να βρεθεί τρόπος ώστε, για ένα επερχόμενο στιγμιότυπο, να αποφασίζεται τάχιστα ο βέλτιστος για την περίπτωση κανόνας και να εισηγείται τη σχετική λύση (πρόβλημα Λήπτη Αποφάσεων).

Η υλοποίηση και των τριών τμημάτων του μοντέλου αυτού παρουσιάζει μεγάλο ενδιαφέρον και αποτελεί αντικείμενο για μελλοντική έρευνα. Μερική υλοποίηση του Λήπτη Αποφάσεων, ο οποίος όπως αναφέραμε χρειάζεται να αποφασίζει δυναμικά, έγινε ήδη παράλληλα στο Connection Machine και οι χρόνοι λήψης των αποφάσεων δικαιολόγησαν απόλυτα την επιλογή αυτή, ιδιαίτερα στην περίπτωση που το Εμπειρο Σύστημα θα έχει εμπλουτιστεί σημαντικά. Ο τρόπος υλοποίησης του Λήπτη Αποφάσεων δεν θα πρέπει να τον περιορίζει στο να μεταβάλλει κατά περίπτωση τους χώρους του Προβλήματος και της Λύσης (Problem and Solution Space) με τρόπο αυτόματο καθώς ο Host αποφασίζει ποια από τα γνωρίσματα που περιγράφουν τους κανόνες είναι γνωστά και ποια είναι τα ζητούμενα.

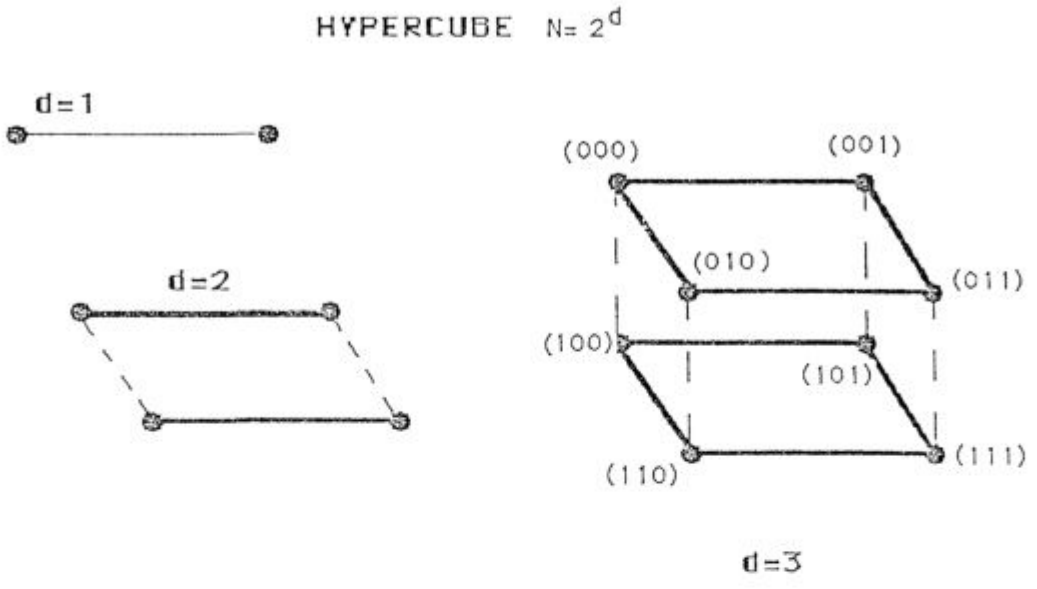
Παράρτημα Α

Α.1 Οι Αρχιτεκτονικές που χρησιμοποιήθηκαν

Στο πειραματικό μέρος της εργασίας αυτής χρησιμοποιήθηκαν οι αρχιτεκτονικές Connection Machine (CM) της Thinking Machines και το iPSC/2 της Intel. Και οι δύο ανήκουν στη γενική κατηγορία των αρχιτεκτονικών Υπερκύβου (Hypercube).

Μια παράλληλη αρχιτεκτονική υπερκύβου είναι ένας πολυεπεξεργαστής με $p = 2^d$ επεξεργαστές, συνδεδεμένους με τη συνδεσμολογία ενός d-διάστατου δυαδικού κύβου. Στους επεξεργαστές μιας τέτοιας αρχιτεκτονικής μπορούν να εκχωρηθούν διευθύνσεις έτσι ώστε οι διευθύνσεις άμεσα συνδεμένων επεξεργαστών να διαφέρουν κατά ένα και μόνο δυαδικό ψηφίο. Η αρχιτεκτονική υπερκύβου έχει μια αναδρομική δομή. Ενας (d+1)-διάστατος υπερκύβος μπορεί να κατασκευαστεί από δύο d-διάστατους υπερκύβους εάν 1) συνδέσουμε τους αντίστοιχους επεξεργαστές που έχουν την ίδια διεύθυνση στους δύο d-διάστατους υπερκύβους και 2) προσθέσουμε ένα δυαδικό ψηφίο στην πιο σημαντική θέση της διεύθυνσης του κάθε επεξεργαστή (1 για τους επεξεργαστές του πρώτου υπερκύβου και 0 για τους επεξεργαστές του δεύτερου). Κάθε επεξεργαστής ενός d-διάστατου υπερκύβου γειτονεύει με d άλλους επεξεργαστές. Η μέγιστη δυνατή απόσταση μεταξύ επεξεργαστών είναι d και η μέση απόσταση $\frac{d}{2}$. Ο ολικός αριθμός διασυνδέσεων ανάμεσα στους επεξεργαστές ενος d-διάστατου υπερκύβου είναι $d * 2^{(d-1)}$. Η αρχιτεκτονική υπερκύβου έχει ποικίλες ενδιαφέρουσες τοπολογικές ιδιότητες [166], και απο-

τελεί μία από τις πλέον διαδεδομένες τοπολογίες. Η τοπολογία ενός 4-διάστατου υπερκύβου φαίνεται στο Σχήμα A.1.



Σχήμα A.1: Η τοπολογία Υπερκύβου (Hypercube)

Στις επόμενες παραγράφους αναφερόμαστε πιο ειδικά στα δύο συστήματα, παρουσιάζοντας τα ιδιαίτερα χαρακτηριστικά του καθενός.

A.1.1 Connection Machine (CM-2)

Το Connection Machine είναι μία παράλληλη μηχανή βασισμένη στα κυτταρικά αυτόματα (Cellular automata) [76, 77]. Αποτελείται στην πλήρη του διάταξη από 64K επεξεργαστές που είναι "κτισμένοι" πάνω σε 4096 ολοκληρωμένα κυκλώματα (chips). Κάθε ολοκληρωμένο κύκλωμα έχει 16 επεξεργαστές διαταγμένους σε ένα πλέγμα 4x4 και το κάθε ένα από αυτά τα 4096 ολοκληρωμένα κυκλώματα βρίσκεται σε ένα κόμβο ενός υπερκύβου διάστασης 12. Το όλο σύστημα είναι ένας πολυεπεξεργαστής τύπου SIMD (Single Instruction Stream - Multiple Data Stream). Οι 64K επεξεργαστές είναι χωρισμένοι σε 8 Sequencers, κάθε ένας από τους οποίους είναι υπεύθυνος για 8K επεξεργαστές και μπορεί να λειτουργήσει ανεξάρτητα από διαφορετικούς χρήστες. Όλοι οι Sequencers κατευθύνονται από ένα Host υπολογιστή (front

end) ο οποίος τους στέλνει μακροεντολές του κώδικα (macroinstructions) και τα αντίστοιχα δεδομένα. Εκεί υπάρχουν μικροελεγκτές (microcontrollers) οι οποίοι, χρησιμοποιώντας μικροεντολές (microinstructions) του συστήματος, μετατρέπουν τις μακροεντολές σε νανοεντολές (nanoinstructions). Τις τελευταίες αυτές εκτελεί κάθε επεξεργαστής χωριστά. Προκειμένου να προγραμματιστεί, το CM διαθέτει ένα ισχυρό σύνολο μακροεντολών (macroinstructions) που ονομάζεται *Paris (Parallel Instructions)*, που αποτελεί το χαμηλότερου επιπέδου πρωτόκολο επικοινωνίας με το χρήστη. Οι υψηλού επιπέδου γλώσσες επικοινωνίας που υποστηρίζει, είναι κτισμένες πάνω στο σύνολο των εντολών *Paris* και είναι η *Lisp, C* (παράλληλη Lisp και παράλληλη C αντίστοιχα), καθώς επίσης η παράλληλη Fortran.

Το Connection Machine υποστηρίζει δύο μηχανισμούς ενδοεπικοινωνίας μεταξύ των επεξεργαστών του. Ο πιο βασικός είναι ο *Router* που επιτρέπει συγχρόνως την πολλαπλή ανταλλαγή μηνυμάτων μεταξύ όλων των επεξεργαστών του συστήματος. Ο δεύτερος μηχανισμός είναι λιγότερο γενικός αλλά συμπληρώνει τον πρώτο. Οργανώνει τους επεξεργαστές σε ένα διδιάστατο ορθογωνικό πλέγμα, διαθέτει εντολές που επιτρέπουν σε κάθε επεξεργαστή να στείλει μηνύματα σε άλλους επεξεργαστές προς τις τέσσερις κατευθύνσεις (*(N)orth, (E)ast, (W)est, (S)outh*) από τις οποίες παίρνει και το όνομά του (*NEWS*).

Η επικοινωνία μεταξύ των επεξεργαστών πάνω σε κάθε ολοκληρωμένο κύκλωμα (on-chip) γίνεται πάρα πολύ γρήγορα μέσω του ορθογωνικού πλέγματος που βρίσκεται πάνω σ'αυτό ενώ ένας από τους επεξεργαστές ρυθμίζει μέσω του δικτύου την επικοινωνία τους με επεξεργαστές που βρίσκονται έξω από αυτό (off-chip). Οι επεξεργαστές είναι του ενός bit και ο κάθε ένας διαθέτει μνήμη 1Kbyte. Ο καθένας από αυτούς μπορεί να χωριστεί και να λειτουργήσει νοητά σαν 2,4,8,16 κ.ο.κ. ιδεατούς επεξεργαστές (virtual processors) ανεβάζοντας έτσι το συνολικό αριθμό των επεξεργαστών σε περισσότερους από 1M [78], και δίνει τη δυνατότητα για επεξεργασία πολύ λεπτής διαμέρισης (fine granularity). Μπορούν επίσης να διαταχθούν λογικά (με χρήση ενός προγραμματιζόμενου δικτύου επικοινωνίας) σε διάφορες τοπολογίες. Μια σημαντική η ιδιότητά του είναι η δυνατότητά του να αυξάνει κλιμακωτά (scalability) χωρίς ουσιαστικές μετατροπές στους κώδικες.

Το πολύ πλούσιο δίκτυο διασύνδεσης και η ταχύτητά του, καθώς επίσης και η δυνατότητα προσπέλασης που έχει κάθε επεξεργαστής στη μνήμη οποιουδήποτε άλλου, του επιτρέπουν να θεωρείται καλή προσομοίωση του μοντέλου της Κατανεμημένης Παράλληλης Μνήμης Τυχαίας

Προσπέλασης (Distributed PRAM). Το Connection Machine όπως και άλλα συστήματα έχουν ενσωματωμένες στον εξοπλισμό τους κάποιες βασικές πράξεις (primitives). Οι πράξεις αυτές που λέγονται και *προθεματικές πράξεις σάρωσης* (scan or prefix operations) εκτελούν σε πολύ μικρό χρόνο ορισμένες κοινές, στοιχειώδεις διεργασίες όπως ταξινόμηση, εύρεση μεγίστων και ελαχίστων τιμών κ.ά. πάνω στις τιμές μιας μεταβλητής. Συνοπτικά ο τρόπος λειτουργίας τους είναι ο εξής: οι τιμές της μεταβλητής αυτής, στους διάφορους επεξεργαστές μπορούν να θεωρηθούν σαν τα στοιχεία ενός πίνακα A[N], κάθε στοιχείο A[i] του οποίου κρατάει ένας αντίστοιχος επεξεργαστής p_i. Μετά την εκτέλεση της πράξης ο επεξεργαστής p_n κρατάει το αποτέλεσμα της πράξης πάνω στα στοιχεία A[0] έως και A[n]. Η σάρωση μπορεί να εκτελεστεί και κατά την αντίστροφη φορά (backwards) και τότε ο επεξεργαστής p_i κρατάει το αποτέλεσμά της πάνω στα στοιχεία A[N] έως και A[i]. Οι πράξεις σάρωσης μπορούν ακόμα να συνδυαστούν και να συνθέσουν πολυπλοκότερες πράξεις [80]. Παραδείγματα της λειτουργίας των προθεματικών πράξεων σάρωσης παρουσιάζονται διαγραμματικά πάνω σε ένα πίνακα A:

A	=	[6 2 3 1 3 5]	A	=	[6 2 3 1 3 5]
scan+	=	[6 8 11 12 15 20]	scan *(copy)*	=	[6 6 6 6 6 6]
A	=	[4 2 3 7 1 4]	A	=	[4 2 3 7 3 2]
scan*(max)*	=	[4 4 4 7 7 7]	scan*(max-backwards)*	=	[7 7 7 7 3 2]

Ακόμα υπάρχει και η τμηματική μορφή των πράξεων σάρωσης (segmented scan-operations) που χρησιμοποιεί τις τιμές ενός Boolean πίνακα T[N] σαν δείκτη που διαχωρίζει τον αρχικό πίνακα A[N] σε τμήματα, σε καθένα από τα οποία εφαρμόζεται η πράξη σάρωσης ανεξάρτητα. Κάθε τέτοιο τμήμα είναι μεταξύ δύο διαδοχικών True τιμών του πίνακα T[N].

A	=	[7 2 1 3 2 1 6 5 1]
Τμηματική μεταβλ. T	=	[T F F T F F F T F]
scan*(max)*	=	[7 7 7 3 3 3 6 5 5]

Τέλος από τους χρήσιμους μηχανισμούς που διαθέτει το σύστημα είναι οι πράξεις συρ-

ρίκνωσης (*Reduction*). Τέτοιες πράξεις είναι οι ***max**(*var*) και ***min**(*var*) που επιστρέφουν στο Host τη μέγιστη και την ελάχιστη τιμή των τιμών της μεταβλητής *var* που διαθέτουν όλοι οι επεξεργαστές.

A.1.2 iPSC/2 - Hypercube

Το iPSC/2 ανήκει στους πολυεπεξεργαστές τύπου MIMD (Multiple Instructions Stream - Multiple Data Stream) αλλά μπορεί να λειτουργήσει και σαν SIMD (Single Instructions Stream - Multiple Data Stream), και αντί για πολλούς επεξεργαστές μικρής υπολογιστικής ισχύος, διαθέτει λίγους αλλά ισχυρούς. Συγκεκριμένα, αποτελείται από 1 έως και 128 επεξεργαστές συνδεδεμένους με την κλασική συνδεσμολογία του υπερκύβου, που αναπτύχθηκε νωρίτερα. Κάθε επεξεργαστής αποτελείται από έναν 80386, 32-bit μικροεπεξεργαστή και 1 έως 16 Mbytes κύριας μνήμης. Προαιρετικά κάθε κόμβος μπορεί να διαθέτει αριθμητικό συνεπεξεργαστή (numeric coprocessor), μονάδα επεξεργασίας αριθμών κινητής υποδιαστολής (floating-point unit) ή μονάδα επεξεργασίας πινάκων (vector floating point unit). Κάθε επεξεργστής μπορεί να χειρίζεται ανεξάρτητα αρχεία για ανάγνωση ή για γραφή δεδομένων. Η επικοινωνία μεταξύ επεξεργαστών γίνεται με ανταλλαγή μηνυμάτων (message passing) και επιτυγχάνεται μέσω ενός δικτύου διακοπτών (Switching Network). Όταν κάποιος επεξεργαστής επιθυμεί να επικοινωνήσει με κάποιον άλλο, μία σειρά από διακόπτες κλείνουν και έτσι καθορίζεται το μονοπάτι επικοινωνίας [181]. Η μόνη στιγμή που ένας επεξεργαστής εμπλέκεται στην ανταλλαγή μηνυμάτων είναι όταν δέχεται σαν τελικός αποδέκτης ένα μήνυμα, ή τη στιγμή που το στέλνει σαν αποστολέας.

Τυπικά, η μεταφορά ενός μηνύματος μεγέθους m bytes, από έναν επεξεργαστή προς κάποιον άλλο διαρκεί χρόνο: $t_s + t_b(n, b)$, όπου t_s είναι ο χρόνος προετοιμασίας της επικοινωνίας (setup time) και $t_b(n, b)$ ο χρόνος μεταφοράς μηνύματος μεγέθους b σε φυσική απόσταση n μεταξύ των επεξεργαστών. Ο χρόνος t_s είναι πολύ μεγάλος σε σχέση με τον $t_b(n, b)$ και αυτό κάνει τα συχνά και μικρά μηνύματα αντιοικονομικά.

Το σύστημα διαθέτει διάφορους μηχανισμούς και εντολές για το συγχρονισμό των επεξεργαστών στην υπολογιστική διαδικασία, αλλά και για την ανταλλαγή σύγχρονων και ασύγχρονων μηνυμάτων.

Ολοι οι επεξεργαστές της αρχιτεκτονικής συνδέονται με ένα κεντρικό υπολογιστή (Host), ο οποίος ελέγχει τη λειτουργία του όλου συστήματος, στέλνει τον κώδικα στους κομβικούς επεξεργαστές και δίνει την εντολή για την εκκίνηση της επεξεργασίας.

Στο λογισμικό εξοπλισμό του συμπεριλαμβάνονται μεταφραστής της C , της FORTRAN, περιβάλλον της Common LISP κ.ά. ενώ το λειτουργικό σύστημα που υποστηρίζει ο Host υπολογιστής είναι το Unix.

Διαθέτει έναν αριθμό από πράξεις συρρίκνωσης (*Reduction*) με διάφορους τελεστές (**Global-Max, Global-Min, Global-Or, Global-And** κτλ). που εκτελούνται είτε πάνω στην τιμή μιας μεταβλητής είτε σε στοιχεία πινάκων που κρατούν όλοι οι επεξεργαστές.

Στη συνέχεια παραθέτουμε τις επιδόσεις που μετρήθηκαν για την εκτέλεση υπολογιστικών και επικοινωνιακών πράξεων στα δύο συστήματα αρχίζοντας από το Connection Machine.

Α.2 Μετρήσεις Χρόνων στο CM-2 και στο iPSC/2

Εγιναν διάφορες μετρήσεις για τους χρόνους που απαιτούν οι δυο αρχιτεκτονικές για να εκτελέσουν τόσο πράξεις υπολογιστικές όσο και για ενδοεπικοινωνία. Επίσης μετρήθηκαν οι απαιτούμενοι χρόνοι για την εκτέλεση πράξεων σάρωσης. Οι μετρήσεις αυτές φροντίσαμε να καλύπτουν ένα ευρύ φάσμα παραμέτρων έτσι ώστε να πάρουμε πειραματικά την καλύτερη δυνατή εικόνα των χαρακτηριστικών των δύο αρχιτεκτονικών. Χρησιμοποιήθηκαν μεταβλητές διαφόρων μεγεθών, και ειδών. Ακέραιες (integer), πραγματικές (real), διπλές (double) κτλ. Επίσης για να μετρήσουμε επιδόσεις στην επικοινωνία, χρησιμοποιήθηκαν μηνύματα διαφόρων μεγεθών και με διάφορα σχήματα επικοινωνίας (communication patterns). Χρησιμοποιήθηκαν ακόμα διάφορα πλήθη επεξεργαστών στο κάθε σύστημα. Τα αποτελέσματα όλων αυτών των μετρήσεων παρατίθενται στις επόμενες παραγράφους και παρουσιάζουν ιδιαίτερο ενδιαφέρον γιατί δίνουν ανάγλυφα τα χαρακτηριστικά και τις διαφορές των δύο αρχιτεκτονικών στον υπολογιστικό και επικοινωνιακό τομέα. Πρώτα παραθέτουμε τις μετρήσεις που αφορούν στο Connection Machine, τόσο τις υπολογιστικές όσο και τις επικοινωνιακές, ενώ στην συνέχεια τις αντίστοιχες που αφορούν στο iPSC/2 (Hypercube). Στο τέλος δίνεται ένας συνοπτικός σχολιασμός των επιδόσεων αυτών. Σε όσες μετρήσεις υπάρχει το σημείο (-), σημαίνει πως δεν

έγινε η μέτρηση.

Θα πρέπει να σημειώσουμε πως η χρονομέτρηση αρχιτεκτονικών αποτελεί μια πολύπλοκη και ευαίσθητη διαδικασία, το αποτέλεσμα της οποίας εξαρτάται από πολλούς παράγοντες, όπως η κατάσταση της μηχανής την ώρα της χρονομέτρησης, τον υπολογιστή από τον οποίο χρησιμοποιείται (front end machine) και το φορτίο του κ.ά. Οι μετρήσεις που παρατίθενται στη συνέχεια έγιναν υπό συνθήκες ανάλογες με εκείνες των πειραμάτων που παρουσιάζονται στα προηγούμενα κεφάλαια.

A.2.1 Μετρήσεις στο Connection Machine

Το μηχάνημα που χρησιμοποιήθηκε είναι το CM-2 με 8K, 16K και 32K επεξεργαστές. Οπως έχει ήδη αναφερθεί κάθε φυσικός επεξεργαστής έχει τη δυνατότητα να προσομοιάσει ένα ιδεατό μικρό πλέγμα (virtual grid) επεξεργαστών. Μπορεί να προσομεωθεί έτσι από όλο το σύστημα των επεξεργαστών ένα μεγαλύτερο ιδεατό πλέγμα που μπορεί να αναλάβει την επεξεργασία εικόνων με ένα σημείο ανά επεξεργαστή (pixel per processor). Αν το σύστημα διαθέτει αρκετούς επεξεργαστές τότε οι κόμβοι του πλέγματος είναι φυσικοί επεξεργαστές. Οι παρακάτω μετρήσεις έγιναν για διάφορα μεγέθη τέτοιων ιδεατών πλεγμάτων. Για παράδειγμα μια εικόνα 256x256 απαιτεί 64K επεξεργαστές ενώ μία 512x512 256K. Αν διαθέτουμε 32K φυσικούς επεξεργαστές για παράδειγμα, κάθε ένας από αυτούς θα πρέπει να προσομειώσει στην πρώτη περίπτωση 2 και στη δεύτερη 8 ιδεατούς. Αντίθετα δεν μπορεί να δημιουργηθεί ιδεατό πλέγμα με λιγότερους ιδεατούς επεξεργαστές από φυσικούς. Ετσι για παράδειγμα με 32K επεξεργαστές δεν μπορεί να γίνει πλέγμα 64x64 ή 128x128, παρά μόνο βέβαια αν δεχτούμε ότι θα μείνει αδρανές ένα τμήμα της φυσικής μηχανής. Εγιναν μετρήσεις για πράξεις τόσο από τους κομβικούς επεξεργαστές (nodes) όσο και από τον κεντρικό (host) τις οποίες παραθέτουμε στις επόμενες παραγράφους αρχίζοντας από τις υπολογιστικές επιδόσεις.

Υπολογιστικές μετρήσεις στο CM-2 (Computational Benchmarks)

Ο Πίνακας A.1 δίνει σε μsec το χρόνο που απαιτεί η πράξη if σε 8K, 16K και 32K φυσικούς επεξεργαστές όταν διατάσσονται σε διάφορα μεγέθη ιδεατού πλέγματος (64x64, 128x128,

256x256, 512x512, 1024x1024) τόσο για συνθήκη με διαδική μεταβλητή (Boolean) όσο και με ακέραια μεταβλητή.

Συνθήκη με:	*Ακέραια Μεταβλητή*					*Boolean Μεταβλητή*				
Μέγ. Πλέγματος ⇒	64	128	256	512	1024	64	128	256	512	1024
Φυσικοί Επεξεργ.⇓	*μsec*					*μsec*				
8kp	98	132	245	-	-	15	18	36	-	-
16kp	-	109	164	425	-	-	15	24	61	-
32kp	-	-	121	246	788	-	-	18	36	110

Πίνακας Α.1: CM-2: Η πράξη *if* σε 8K,16K και 32K επεξεργαστές για διάφορα μεγέθη ιδεατού πλέγματος

Στον πίνακα Α.2 δίνονται οι χρόνοι για τις πράξεις πρόσθεση, αφαίρεση, διαίρεση αριθμών με 32 bits στους φυσικούς επεξεργαστές.

Operation ⇒	+	-	*	/
Virtual Grid ⇓	*Time* (*μsec*)			
1x1	36	42	225	210
2x1	70	66	245	250
2x2	125	130	1620	1570
4x4	210	215	3210	3070
8x8	1560	1610	14770	14330

Πίνακας Α.2: CM-2: Αριθμητικές πράξεις

Ο πίνακας Α.3 δίνει το χρόνο για την καθολική πράξη ***max**(*var*) η οποία επιστρέφει στον κεντρικό επεξεργαστή τη μέγιστη τιμή που έχει η μεταβλητή *var* σε όλους τους ιδεατούς ή φυσικούς επεξεργαστές. Ο χρόνος δίνεται σε μsec συναρτήσει του μεγέθους της μεταβλητής για 8K,16K και 32K επεξεργαστές για διάφορα μεγέθη ιδεατού πλέγματος.

Φυσικοί Επεξεργ.	*8K*			*16K*			*32K*		
Μέγ. Πλέγματος	64	128	256	128	256	512	256	512	1024
Μέγεθος μεταβλ.	*μsec*			*μsec*			*μsec*		
8 bits	64	85	194	68	122	340			
16 bits	116	146	350	112	214	625			
32 bits	202	267	663	198	400	1195	268		2264
64 bits	362	508	1290	379	771	2340	509		4520
128 bits	712	994	2610	736	1510	4620	994		8800
256 bits	1410	1960	5120	1450	2994	9180	1963		15200

Πίνακας A.3: CM-2: Η πράξη συρρίκνωσης (∗**max***var*) συναρτήσει του μεγέθους της μεταβλητής *var*

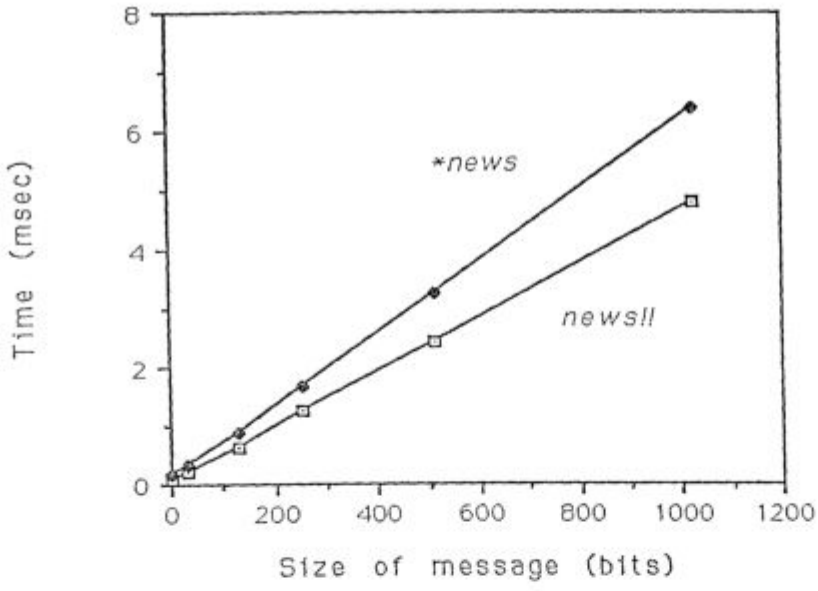
Μετρήσεις επικοινωνίας

Υπάρχει μια πολύ μεγάλη ποικιλία από εντολές για την επικοινωνία μεταξύ των επεξεργαστών του Connection Machine. Κάθε επεξεργαστής έχει μία ταυτότητα που χαρακτηρίζει τη σειρά του στο σύστημα. Η ταυτότητα αυτή είναι ένας ακέραιος αριθμός από το 0 ως το N-1 για N επεξεργαστές. Γνωρίζει επίσης τις x και y συντεταγμένες του αν τους θεωρήσουμε διατεταγμένους σε ένα ορθογωνικό πλέγμα δύο διαστάσεων. Θεωρητικά μπορούν να διαταχτούν και σε πλέγματα περισσότερων διαστάσεων αλλά δεν έχουν κατασκευαστεί ακόμα τέτοιες μηχανές. Υπάρχουν εντολές επικοινωνίας με τις οποίες κάθε ενεργός επεξεργαστής μπορεί να επικοινωνήσει με έναν άλλο επεξεργαστή δίνοντας την απόλυτη διεύθυνσή του είτε στην εν σειρά διάταξη είτε στη διάταξη πλέγματος. Με άλλες εντολές δίνεται μόνο η σχετική απόσταση κατά x και κατά y των επεξεργαστών που θέλουν να επικοινωνήσουν. Μία από τις ισχυρότερες τέτοιες εντολές είναι η *news!! ((n)orth, (e)ast, (w)est, (s)outh)* και με αυτήν κάθε επεξεργαστής διαβάζει από εκείνον με τον οποίο θέλει να επικοινωνήσει, την τιμή μιας μεταβλητής. Η σύνταξη της εντολής είναι *news!!*(*var*, *δx*, *δy*) και επιστρέφει στον επεξεργαστή που την εκτελεί, την τιμή της μεταβλητής *var* που κρατά ο επεξεργαστής που απέχει *δx* θέσεις κατά την X και *δy* θέσεις κατά την Y κατεύθυνση πάνω στο ιδεατό πλέγμα.

Ο πίνακας A.4 δείχνει τους χρόνους εκτέλεσης της εντολής αυτής για ένα σύστημα με

16K επεξεργαστές διατεταγμένους σε διάφορα μεγέθη ορθογωνικού πλέγματος. Οι χρόνοι είναι σε μsec και δίνονται συναρτήσει του μεγέθους (σε bits) της μεταβλητής. Τα τρία μέρη του πίνακα αυτού αντιστοιχούν σε 3 διαφορετικές αποστάσεις κατά τη X κατεύθυνση, μεταξύ των δύο επεξεργαστών που επικοινωνούν. Το πρώτο είναι για άμεση γειτνίαση (απόσταση 1), η δεύτερη για απόσταση 10 και η τρίτη για απόσταση 20. Στις μετρήσεις που έγιναν, όλοι οι επεξεργαστές συμμετείχαν επικοινωνώντας με κάποιον άλλο. Βλέπουμε πως ο χρόνος επικοινωνίας αυξάνει αναλογικά με το μέγεθος του μηνύματος. Επίσης αυξάνει με την απόσταση αλλά με διαφορετικό ρυθμό. Συγκεκριμένα για επικοινωνία με τον αμέσως γειτονικό επεξεργαστή οι αντίστοιχοι χρόνοι είναι πολύ μικροί σε σχέση με το χρόνο που απαιτεί η ίδια σε μέγεθος πλέγματος και μηνύματος επικοινωνία, για απόσταση=10 μεταξύ των επεξεργαστών. Ομως αυτό που βλέπουμε είναι πως για απόσταση=20 ο χρόνος επικοινωνίας δεν αυξάνει με τον ίδιο ρυθμό. Αυτό οφείλεται κατά κύριο λόγο στο ότι η επικοινωνία με τον αμέσως επόμενο επεξεργαστή γίνεται ως επί το πλείστον on-chip. Δηλαδή δεν χρειάζεται οι επεξεργαστές να χρησιμοποιήσουν το δίκτυο (Router). Αντίθετα για επικοινωνία σε απόσταση 10 και 20 θέσεις μακρύτερα γίνεται off-chip μέσω του δικτύου με τις ανάλογες καθυστερήσεις. Συγκρίνοντας τώρα την επικοινωνία για απόσταση 10 θέσεων με εκείνη για απόσταση 20 θέσεων παρατηρούμε πως δεν υπάρχει τόσο μεγάλη αύξηση. Ο λόγος είναι πως το δίκτυο είναι πολύ πλούσιο και η διπλάσια απόσταση στο ορθογωνικό πλέγμα δεν συνεπάγεται κατ'ανάγκη διπλάσια φυσική απόσταση στη τοπολογία που προσομειώνει το πλέγμα αυτό. Αντίθετα μάλιστα η σχέση στις φυσικές αποστάσεις είναι λογαριθμική στον υπερκύβο, και με αυτή τη σχέση βλέπουμε πράγματι να συμφωνούν κατά προσέγγιση οι χρόνοι.

Στον πίνακα A.5 δίνονται οι χρόνοι εκτέλεσης της ίδιας εντολής *news!!* που υλοποιείται σε 8K,16K και 32K φυσικούς επεξεργαστές όταν διατάσσονται σε διάφορα μεγέθη ιδεατού πλέγματος. Οι μετρήσεις γίνονται για επικοινωνίες σε απόσταση 1,3,5,7 και 9 θέσεων πάνω στο ιδεατό πλέγμα κατά τη X κατεύθυνση. Τα μεγέθη των μηνυμάτων στην περίπτωση αυτή ήταν διαφορετικά για τον κάθε επεξεργαστή αλλά ήταν μέχρι 32 bits το κάθε ένα. Ολοι οι επεξεργαστές εκτέλεσαν συγχρόνως την εντολή επικοινωνίας. Και στην περίπτωση αυτή μπορούμε να κάνουμε παρατηρήσεις ανάλογες με εκείνες του πίνακα A.4. Αυτό που έχει ενδιαφέρον στην περίπτωση αυτή πως για πλέγματα με μικρή διαμέριση, όταν η απόσταση γίνεται για 9 θέσεις παρατηρείται μείωση του χρόνου επικοινωνίας. Αυτό δικαιολογείται από το γεγονός ότι κατά την εναπόθεση του ορθογωνικού πλέγματος πάνω στην τοπολογία του



Σχήμα Α.2: CM-2: Οι εντολές *news!!* και **news* σε 8Κ επεξεργαστές

υπερκύβου οι επεξεργαστές σε απόσταση 9 θέσεων βρίσκονται σε μικρότερη φυσική απόσταση από εκείνους που απέχουν 7 θέσεις.

Ο πίνακας Α.6 είναι ανάλογος του Α.5 και δείχνει τους χρόνους για την εντολή **news*. Η εντολή αυτή είναι αντίστοιχη της *news!!* με τη διαφορά ότι η τιμή της μεταβλητής στέλνεται από τον επεξεργαστή που εκτελεί την εντολή στον επεξεργαστή με τον οποίο θέλει να επικοινωνήσει. Τα τρία τμήματα αντιστοιχούν στις 3 διαφορετικές αποστάσεις μεταξύ των επεξεργαστών. Οπως μπορεί να παρατηρήσει κανείς, η εντολή *news!!* είναι γρηγορότερη από την εντολή **news*. Αυτό φαίνεται και στο διάγραμμα του Σχήματος Α.2 όπου δίνονται για σύγκριση οι καμπύλες των δύο εντολών.

Μέγ. Μηνύματος (bits) ⇒		2	32	128	256	512	1024
Απόσταση ⇓	*Πλέγμα* ⇓	*Χρόνοι σε msec*					
1	128	.07	.2	.64	1.25	2.43	4.78
	256	.15	.46	1.70	3.15	6.20	12.25
	512	.32	1.33	4.60	9.00	17.80	-
	1024	.88	3.30	14.90	-	-	-
10	128	.29	1.13	4.23	8.37	16.60	33.00
	256	.40	2.32	8.90	17.60	35.00	69.70
	512	.97	5.56	20.40	40.20	79.70	-
	1024	2.91	14.46	51.30	-	-	-
20	128	.30	1.30	4.96	9.80	19.50	38.90
	256	.69	4.43	16.80	33.30	66.20	132.20
	512	1.35	9.46	35.50	70.30	139.90	-
	1024	3.66	22.20	81.00	-	-	-

Πίνακας Α.4: CM-2: *news!!* σε 16K επεξεργαστές ως προς το μέγεθος του μηνύματος και την απόσταση

Απόσταση ⇒		1	3	5	7	9
Φυσ. Επεξεργ. ⇓	*Πλέγμα* ⇓	*Χρόνοι σε μsec*				
8K	64	125	210	250	340	380
	128	150	260	330	430	390
	256	320	560	760	1020	960
	512	950	1600	2020	2660	2880
16K	128	124	207	255	350	430
	256	240	450	550	780	750
	512	595	1100	1510	2020	1960
	1024	1950	3280	4160	5490	5950
32K	256	160	270	324	460	410
	512	330	590	790	1070	1020
	1024	1000	1680	2130	2810	3050

Πίνακας Α.5: CM-2: *news!!* για μηνύματα έως 32 bits ως προς μέγεθος πλέγματος-απόστασης

Απόσταση ⇒		1	3	5	7	9
Φυσ. Επεξεργ. ⇓	*Πλέγμα* ⇓	*Χρόνοι σε μsec*				
8K	64	230	320	350	450	500
	128	300	400	450	600	550
	256	650	890	1100	1390	1330
	512	2090	2750	3200	3900	4140
16K	128	230	320	350	450	560
	256	410	640	760	990	980
	512	1180	1720	2140	2690	2640
	1024	4220	5620	6540	7940	8440
32K	256	290	420	470	700	580
	512	670	930	1110	1450	1390
	1024	2190	2910	3400	4130	4410

Πίνακας Α.6: CM-2: **news* για μηνύματα έως 32 bits ως προς μέγεθος πλέγματος-απόστασης

Απόσταση ⇒		1	3	5	7	9
Φυσ. Επεξ. ⇓	*Πλέγμα* ⇓	*Χρόνοι σε μsec*				
8K	64	130	140	150	200	260
	128	185	210	225	250	250
	256	190	320	405	530	420
	512	470	760	910	1210	1300
16K	128	98	130	140	180	250
	256	145	260	270	380	280
	512	300	525	710	950	720
	1024	850	1380	1730	2270	2400
32K	256	140	170	180	250	190
	512	200	310	400	550	420
	1024	490	740	920	1200	1300

Πίνακας A.7: CM-2: *news!!* για μήνυμα 2 bits ως προς μέγεθος πλέγματος-απόστασης

Ακολουθεί Πίνακας A.7 στον οποίο φαίνονται σε μsec οι χρόνοι εκτέλεσης της εντολής *news!!* σε 8K,16K και 32K επεξεργαστές για μέγεθος μηνύματος 2 bits, συναρτήσει του μεγέθους του ιδεατού πλέγματος που ξεκινά από 64x128 και καταλήγει σε 1024x1024. Οι επιδόσεις δίνονται συναρτήσει της απόστασης μεταξύ των επεξεργαστών που επικοινωνούν. Και στην περίπτωση αυτή συμμετείχαν όλοι οι επεξεργαστές. Ο πίνακας αυτός είναι ο αντίστοιχος του A.5 και διαφέρει μόνο στο μέγεθος των μηνυμάτων που ανταλάσσουν οι επεξεργαστές.

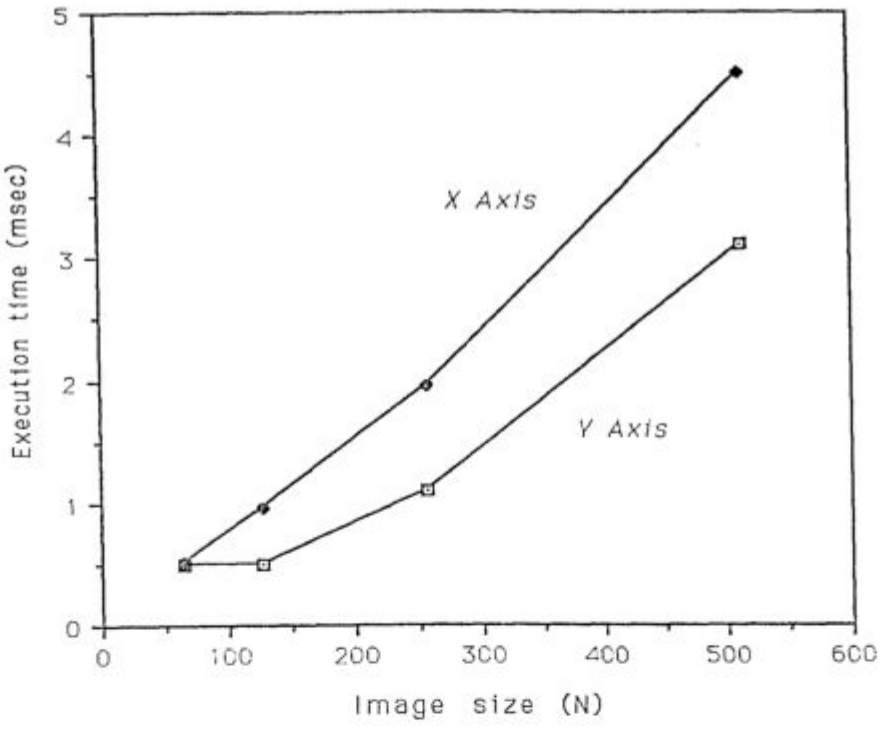
Παρομοίως ο πίνακας A.8 είναι ο αντίστοιχος για την εντολή **news*. Και εδώ το μέγεθος των μηνυμάτων είναι 2 bits, ενώ η εντολή εκτελείται συγχρόνως από όλους τους ιδεατούς επεξεργαστές.

Οπως έχουμε ήδη αναφέρει, όταν δημιουργούμε ένα ιδεατό πλέγμα σε ένα πλήθος φυσικών επεξεργαστών, είναι δυνατό το μικρό ιδεατό πλέγμα που προσομειώνει κάθε επεξεργαστής να μην έχει τις ίδιες διαστάσεις κατά τη X και κατά την Y κατεύθυνση. Αυτό έχει σαν συνέπεια οι εντολές επικοινωνίας και οι πράξεις σάρωσης που θα αναφερθούν στην επόμενη παράγραφο, να μην εκτελούνται στον ίδιο χρόνο για την κάθε κατεύθυνση. Στον πίνακα A.9 φαίνεται ακριβώς

Απόσταση ⇒		1	3	5	7	9
Φυσ. Επεξ. ⇓	*Πλέγμα* ⇓	*Χρόνοι σε μsec*				
8K	64	230	278	290	350	370
	128	290	340	355	430	370
	256	530	650	760	930	830
	512	1180	1770	2000	2350	2500
16K	128	190	240	240	295	360
	256	300	405	430	550	470
	512	680	930	1130	1370	1200
	1024	2150	2740	3100	3700	3910
32K	256	250	290	300	390	570
	512	430	540	640	790	680
	1024	1160	1470	1660	1980	2100

Πίνακας Α.8: CM-2: **news* για μήνυμα 2 bits ως προς μέγεθος πλέγματος-απόστασης

αυτή η διαφορά χρόνου εκτέλεσης της εντολής στις δύο διαστάσεις του πλέγματος. Η ίδια διαφορά παρουσιάζεται και στο διάγραμμα του σχήματος Α.3.



Σχήμα Α.3: CM-2: Η εντολή *scan* κατά τη διεύθυνση X και Y

		8K Επεξεργαστές				*16K Επεξεργαστές*			
Πλέγμα ⇒		64	128	256	512	128	256	512	1024
Απόστ.	X-Y	*Χρόνοι σε μsec*							
1	X	130	185	190	470	98	145	300	850
	Y	140	160	230	540	95	140	305	830
3	X	140	210	320	760	130	260	525	1380
	Y	148	203	440	1020	128	260	500	1330
5	X	150	225	405	910	140	270	710	1730
	Y	145	220	460	1420	136	262	670	1640
7	X	200	250	530	1210	180	380	950	2270
	Y	196	293	705	1890	183	390	890	2140
9	X	260	250	420	1300	280	290	720	2400
	Y	222	318	514	1510	402	290	716	2350

Πίνακας Α.9: CM-2: *news!!* κατά X-Y ως προς το μέγεθος του πλέγματος και της απόστασης

Μια άλλη εντολή επικοινωνίας παρόμοια με την *news!!* αλλά που διαθέτει μία ακόμα δυνατότητα είναι η *news-border!!(var border-var x y)*. Η εντολή αυτή επιστρέφει στον επεξεργαστή που την εκτελεί την τιμή της μεταβλητής *var* που κρατάει ο επεξεργαστής που απέχει από αυτόν πάνω στο ιδεατό πλέγμα x θέσεις κατά τη X κατεύθυνση και y κατά την Y. Σε περίπτωση που η θέση αυτή είναι εκτός των ορίων του πλέγματος επιστρέφει την τιμή της μεταβλητής *border-var*. Οι μετρήσεις από την εκτέλεση της εντολής αυτής δίνονται στον πίνακα A.10 για ακέραιες μεταβλητές σε 8K και 16K φυσικούς επεξεργαστές.

Πλέγμα ⇒	64	128	256	512	1024
Πράξη	*Χρόνοι σε μsec*				
8K	510	740	1400	4100	-
16K	-	520	560	1300	4300

Πίνακας A.10: CM-2: *news-border!!* για 8K και 16K επεξεργαστές

Προθεματικές πράξεις σάρωσης (Prefix operations)

Οι προθεματικές πράξεις σάρωσης είναι πράξεις χτισμένες πάνω στην αρχιτεκτονική και εκτελούνται πολύ γρήγορα σε σύγκριση με το χρόνο που θα απαιτούσαν αν τις υλοποιούσαμε λογισμικά με άλλες εντολές. Η λειτουργία τους παρουσιάστηκε σε προηγούμενη παράγραφο. Αυτό που θα ρποσθέσουμε εδώ είναι πως οι πράξεις αυτές υλοποιούνται σε όλους τους ενεργούς επεξεργαστές του συστήματος σαν να ήταν διατεταγμένοι σε ένα μονοδιάστατο πίνακα ή εναλλακτικά στους επεξεργαστές κάθε γραμμής ή κάθε στήλης του ιδεατού πλέγματος.

Στους πίνακες που ακολουθούν θα δείξουμε τις μετρήσεις που έγιναν στις εξής πράξεις σάρωσης :*scan* η καθολική πράξη σάρωσης πάνω σε όλους τους φυσικούς ή ιδεατούς επεξεργαστές. *scan-segm* η τμηματική πράξη σάρωσης πάλι σε όλους τους επεξεργαστές, *scan-x* η πράξη σάρωσης σε κάθε γραμμή του ορθογωνικού πλέγματος (x κατεύθυνση), *scan-y* επίσης σε ορθογωνικό πλέγμα αλλά στους επεξεργαστές κάθε στήλη του πλέγματος. *scan-segm-x* και *scan-segm-y* είναι οι αντίστοιχες με τις προηγούμενες αλλά στην τμηματική μορφή τους. Οι μετρήσεις έγιναν για την υλοποίηση των πράξεων αυτών με τελεστές *max* και *copy* για διάφορα μεγέθη ιδεατού πλέγματος.

Πλέγμα ⇒	64	128	256	512
Πράξη	*Χρόνοι σε msec*			
scan	.37	.45	.85	2.62
scan-segm	.45	.54	.91	2.65
scan-x	.67	.77	1.93	4.95
scan-y	.74	1.43	3.14	7.73
scan-segm-x	1.03	1.16	2.76	8.28
scan-segm-y	1.12	2.12	4.78	11.93

Πίνακας Α.11: CM-2: Σάρωση με τελεστή **max** σε 8Κ φυσ. επεξεργαστές για μεταβλητή έως 16 bits

Ο πίνακας Α.11 παρουσιάζει τους χρόνους που απαιτούν οι πράξεις αυτές για την υλοποίησή τους σε 8Κ φυσικούς επεξεργαστές με τελεστή *max*, συναρτήσει του μεγέθους του ιδεατού πλέγματος. Οι μεταβλητές ήταν μέχρι 16 bits.

Πλέγμα ⇒	128	256	512	1024
Πράξη	*Χρόνοι σε msec*			
scan	.43	.61	1.45	5.30
scan-segm	.50	.71	1.56	5.28
scan-x	.78	1.63	3.85	10.89
scan-y	.77	7.02	3.87	11.04
scan-segm-x	1.15	2.40	6.02	10.73
scan-segm-y	1.15	4.42	6.33	17.07

Πίνακας Α.12: CM-2: Σάρωση με τελεστή **max** σε 16Κ επεξεργαστές για μεταβλητή έως 16 bits

Ο πίνακας Α.12 είναι ο αντίστοιχος με τον προηγούμενο αλλά για 16Κ φυσικούς επεξεργαστές.

Οι χρόνοι αντιστοιχούν στην εκτέλεση της πράξης σε κάθε διάσταση. Όπως και στις πράξεις του προηγούμενου διαγράμματος, έτσι και εδώ παίρνουν μέρος όλοι οι επεξεργαστές.

Πλέγμα ⇒	256	512	1024
Πράξη	*Χρόνοι σε msec*		
scan	.50	.92	2.80
scan-segm	.60	1.00	2.86
scan-x	.88	1.99	5.46
scan-y	1.61	3.53	8.73
scan-segm-x	1.30	3.09	8.79
scan-segm-y	2.51	5.33	13.06

Πίνακας Α.13: CM-2: Σάρωση με τελεστή **max** σε 32Κ επεξεργαστές για μεταβλητή έως 32 bits

Ενδεικτικά δίνουμε το χρόνο για την εκτέλεση μιας πράξης σάρωσης υλοποιημένης λογισμικά που είναι για 8Κ φυσικούς επεξεργαστές διατεταγμένους σε πλέγμα 64x64, για τελεστή *max* 17 msec και για τελεστή *copy* 10 msec περίπου.Ο χρόνος εκτέλεσης της πράξης *segmented-scan-copy* σε 16Κ φυσικούς επεξεργαστές συναρτήσει του πλήθους των ομάδων στις οποίες είναι χωρισμένοι, φαίνεται στον πίνακα Α.19

Η πράξη *segmented-scan-grid-copy* σε ένα πλέγμα 128x128 επεξεργαστών σε 16Κ φυσικούς επεξεργαστές φαίνεται στον πίνακα Α.20.

Ολοκληρώνουμε την παρουσίαση των επιδόσεων στο Connection Machine με την παράθεση του πίνακα Α.21 με τις επιδόσεις τριών πράξεων σε 16Κ φυσικούς επεξεργαστές διατεταγμένους σε πλέγμα 128x128. Της πράξης *sort (var)* που ταξινομεί μεταθέτει σε διαδοχικούς επεξεργαστές τις τιμές της μεταβλητής *var* ταξινομημένες, της *sort-grid (var) dimension* που κάνει ακριβώς το ίδιο αλλά στη διάσταση *dimension* του ορθογωνικού πλέγματος, την *Rank (var)* που επιστρέφει σε κάθε επεξεργαστή τη θέση που έχει η τιμή που κρατά για τη μεταβλητή (var) στην διάταξη των τιμών που έχουν όλοι οι επεξεργαστές στην μεταβλητή αυτή και τέλος για την εντολή *Pack(var)* που μεταθέτει τις τιμές της μεταβλητής *var* που κρατούν όλοι οι ενεργοί επεξεργαστές (σύμφωνα μέ κάποια συνθήκη) κατά την εκτέλεση της εντολής αυτής, έτσι ώστε να καταλάβουν διαδοχικές θέσεις στους επεξεργαστές. Η τελευταία αυτή εντολή δεν είναι του συστήματος αλλά είναι υλοποιημένη λογισμικά. Οι επιδόσεις δίνονται για ακέραια

Πλέγμα ⇒	64	128	256	512
Πράξη	*Χρόνοι σε msec*			
scan	1.00	1.18	2.13	6.81
scan-segm	.51	.61	1.04	3.21
scan-x	1.70	1.80	4.25	12.02
scan-y	2.01	3.44	3.79	20.56
scan-segm-x	.85	1.01	2.39	6.76
scan-segm-y	1.76	1.82	4.12	10.20

Πίνακας Α.14: CM-2: Σάρωση με τελεστή **copy** σε 8K επεξεργαστές για μεταβλητή έως 32 bits

αλλά και για πραγματική τιμή της μεταβλητής *var* σε κάθε περίπτωση.

Πλέγμα ⇒	128	256	512	1024
Πράξη	*Χρόνοι σε msec*			
scan	1.09	1.62	3.85	13.57
scan-segm	.55	.79	1.86	6.47
scan-x	1.77	3.95	9.20	25.53
scan-y	1.87	4.04	10.00	27.48
scan-segm-x	.99	2.10	6.50	15.72
scan-segm-y	1.00	2.11	5.20	14.43

Πίνακας A.15: CM-2: Σάρωση με τελεστή **copy** σε 16K επεξεργαστές για μεταβλητή έως 32 bits

A.2.2 Μετρήσεις στο Hypercube

Η μηχανή στην οποία έγιναν οι επόμενες μετρήσεις είναι ένα iPSC/2 της Intel με 2 έως 64 κόμβους. Τα χαρακτηριστικά των επεξεργαστών αυτών όπως και του συστήματος ολόκληρου παρουσιάστηκαν στην προηγούμενη παράγραφο. Αυτό που παρατηρούμε από τις μετρήσεις των επιδόσεων πως οι χρόνοι εκτέλεσης πράξεων επικοινωνίας δεν εξερτώνται ουσιαστικά από το μέγεθος των μηνυμάτων και τη φυσική απόσταση μεταξύ των επεξεργαστών που επικοινωνούν. Στις επόμενες παραγράφους παρατίθενται οι χρόνοι βασικών υπολογιστικών και επικοινωνιακών εντολών.

Υπολογιστικές μετρήσεις (Computational Benchmarks)

Ο πίνακας A.22 που ακολουθεί δίνει ορισμένες μετρήσεις στις επόμενες χαρακτηριστικές πράξεις: άδειος βρόχος (*for*), άθροιση ακέραιου (*iadd*), πραγματικού (*radd*) και διπλού (*dadd*), και πολλαπλασιασμούς επίσης ακέραιου (*imul*), πραγματικού (*rmul*) και διπλού αριθμού (*dmul*). Οι τελευταίες δύο μετρήσεις είναι για πράξεις *if* με μία συνθήκη (*if1*) και με δύο συνθήκες (*if2*). Οι μετρήσεις έγιναν με 100.000 επαναλήψεις στην κάθε μία, αλλά οι χρόνοι που δίνονται στον πίνακα A.22 είναι για μία μόνο επανάληψη και είναι σε μsec.

Πλέγμα ⇒	256	512	1024
Πράξη	*Χρόνοι σε msec*		
scan	1.30	2.40	7.32
scan-segm	.68	1.19	3.53
scan-x	2.03	4.72	12.72
scan-y	3.98	8.83	21.90
scan-segm-x	1.13	2.65	7.39
scan-segm-y	2.11		

Πίνακας Α.16: CM-2: Σάρωση με τελεστή **copy** σε 32K επεξεργαστές για μεταβλητή έως 32 bits

Μετρήσεις επικοινωνίας

Μετρήσεις έγιναν για δύο διαφορετικά είδη επικοινωνίας. Στην πρώτη περίπτωση παίρνουν μέρος δύο μόνο επεξεργαστές, εστω οι Α και Β. Η επικοινωνία μεταξύ τους γίνεται ως εξής: Οι δύο επεξεργαστές αρχικά συγχρονίζονται. Ο Α στέλνει στον Β το μήνυμα και περιμένει. Ο Β, μόλις πάρει το μήνυμα από τον Α του το στέλνει πίσω ολοκληρώνοντας έτσι τη συμμετοχή του στην επικοινωνία. Ο Α τελειώνει μόλις έχει πάρει το μήνυμα από τον Β. Οι δύο επεξεργαστές απείχαν μεταξύ τους διάφορες αποστάσεις.

Στον επόμενο διάγραμμα του Σχήματος Α.4 έχουμε τους χρόνους που απαιτούνται για την επικοινωνία δύο μόνο επεξεργαστών για διάφορα μεγέθη μηνυμάτων. Οι χρόνοι δίνονται ενδεικτικά για τρείς περιπτώσεις. Για τις περιπτώσεις που το σύστημα έχει 4, 8 και 16 κόμβους. Οπως βλέπουμε οι χρόνοι δεν παρουσιάζουν ουσιαστικές διαφοροποιήσεις. Προφανώς διότι το σύστημα επικοινωνίας έχει πολύ μικρό φορτίο από τους δύο μόνο επεξεργαστές που παίρνουν μέρος στην επικοινωνία.

Στο διάγραμμα του Σχήματος Α.5 δίνονται οι χρόνοι που απαιτούνται για τέσσερα συγκεκριμένα μεγέθη μηνύματος (0, 32, 64, 128 bytes) συναρτήσει των αποστάσεων Hamming των δύο επεξεργαστών.

Στη δεύτερη περίπτωση όλοι οι επεξεργαστές παίρνουν μέρος. Κάθε ένας στέλνει σε ένα

Πλέγμα ⇒	128	256	512	1024
Πράξη	*Χρόνοι σε msec*			
scan	.25	.30	.58	1.75
scan-segm	.30	.36	.63	1.68
scan-gr-x	.57	1.15	2.60	6.70
scan-gr-y	.57	1.15	2.62	6.70
scan-gr-segm-x	.91	2.10	4.28	11.45
scan-gr-segm-y	.91	1.85	4.29	11.40

Πίνακας Α.17: CM-2: Σάρωση με τελεστή **max** σε 16K επεξεργαστές για μεταβλητή έως 2 bits

δεύτερο ένα μήνυμα, ενώ δέχεται από κάποιον τρίτο ένα άλλο μήνυμα ίδιου μεγέθους. Ο μετρούμενος χρόνος σ'αυτή την περίπτωση είναι ο χρόνος που απαιτείται για την εκτέλεση μιας εντολής *csend* (σύγχρονη αποστολή) και μιας *creceive* (σύγχρονη λήψη). Αυτό που παρατηρείται στις μετρήσεις αυτές είναι πως ο χρόνος ολοκλήρωσης μιάς επικοινωνίας ειδικά για τα μηνύματα μικρού μεγέθους, δεν εξαρτάται από το μέγεθος αυτό, διότι το Hypercube έχει μεγάλο χρόνο εκκίνησης του μηνύματος (start-up time).

Επίσης όσο μεγαλύτερα είναι τα μηνύματα, ειδικά για μεγέθη πάνω από 1-2 Kb, τόσο πιο ασταθής είναι ο μετρούμενος χρόνος. Το ίδιο συμβαίνει και στις περιπτώσεις όπου έχουμε μεγάλες φυσικές αποστάσεις μεταξύ των επεξεργαστών που επικοινωνούν. Σε επικοινωνίες όπου όλοι οι επεξεργαστές έπαιρναν μέρος, και είχαν να στείλουν μηνύματα σε όχι γειτονικούς τους, εμφανιζόταν συχνά το φαινόμενο να αυξάνει (ακόμα και να διπλασιάζεται) ο χρόνος ολοκλήρωσης της επικοινωνίας.

Το δίκτυο ενδοεπικοινωνίας με άλλα λόγια είχε σταθερή και προβλέψιμη συμπεριφορά όσο το φορτίο του κρατιόταν κάτω από ένα επίπεδο.

Πλέγμα ⇒	128	256	512	1024
Πράξη	*Χρόνοι σε msec*			
scan	.88	1.23	2.74	9.40
scan-segm	.36	.42	.76	2.15
scan-gr-x	1.80	3.31	7.92	22.42
scan-gr-y	1.70	3.60	8.53	28.50
scan-gr-segm-x	.80	1.64	3.79	10.30
scan-gr-segm-y	.82	1.62	3.95	10.05

Πίνακας Α.18: CM-2: Σάρωση με τελεστή **copy** σε 16K επεξεργαστές για μεταβλητή έως 2 bits

Πλήθος ομάδων	4	6	8	12	16	32	64	128
time (μs)	468	469	468	469	470	471	472	497

Πίνακας Α.19: CM-2: Σάρωση *segmented-scan-copy*

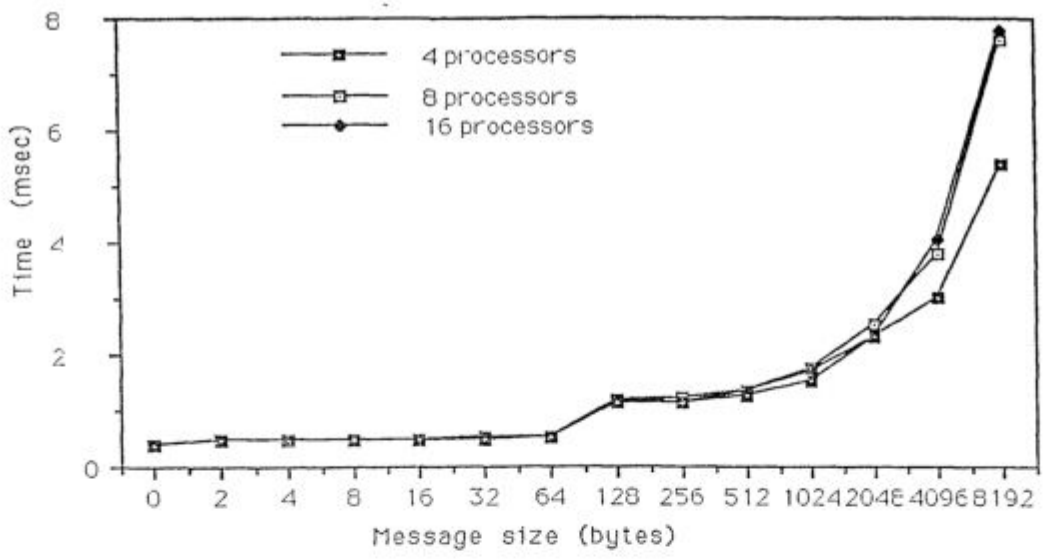
Πράξεις σάρωσης

Τα επόμενα διαγράμματα αναφέρονται στις πράξεις σάρωσης πάνω στο Hypercube για διάφορα πλήθη κόμβων. Οι χρόνοι εδώ είναι σε msec.

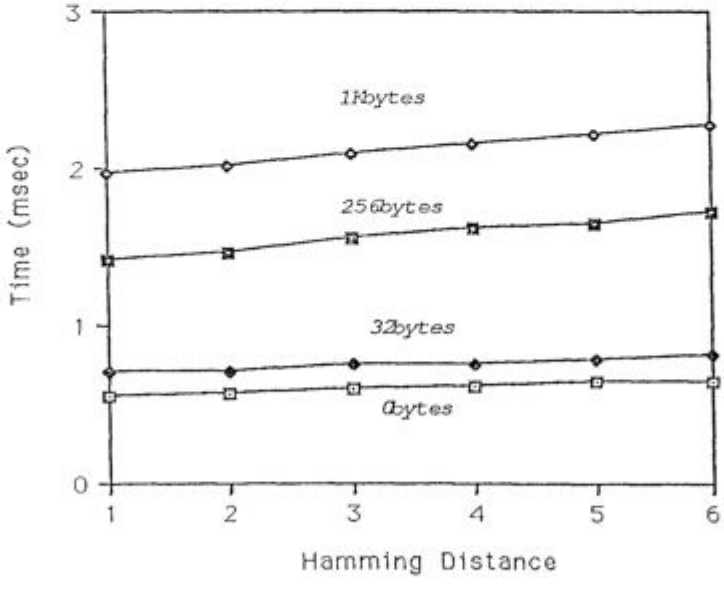
Το διάγραμμα Α.6 δείχνει τον χρόνο που απαιτεί ο συγχρονισμός των κομβικών επεξεργαστών συναρτήσει του πλήθους των. Η εντολή είναι η *gsync()* και χρησιμοποιείται για να συγχρονίσει την επεξεργασία στους κόμβους. Κάθε επεξεργαστής που φτάνει στην εντολή

Πλήθος ομάδων	4	6	8	12	16	32	64	128
time (μs)	948	947	950	951	947	946	948	948

Πίνακας Α.20: CM-2: Σάρωση *segmented-scan-copy* για μια διάσταση σε ορθογωνικό πλέγμα



Σχήμα Α.4: iPSC/2: Επικοινωνία *ένας προς ένα* με 8 και 16 κόμβους ως προς το μέγεθος του μηνύματος



Σχήμα Α.5: iPSC/2: Επικοινωνία *ένας προς ένα* ως προς μέγεθος του μηνύματος και απόστασης Hamming

	Ακέραιος	Πραγματικός
Πράξη	*χρόνοι σε msec*	
sort (var)	16.03	50.60
sort-grid	45.84	78.32
Rank	15.60	48.40
Pack	1.15	2.48

Πίνακας Α.21: CM-2: Διάφορες πράξεις

Operation	*for*	*iadd*	*radd*	*dadd*	*imul*	*rmul*	*dmul*	*if1*	*if2*
Time (μs)	1.32	0.06	1.7	4.3	0.06	1.7	4.45	1.0	1.3

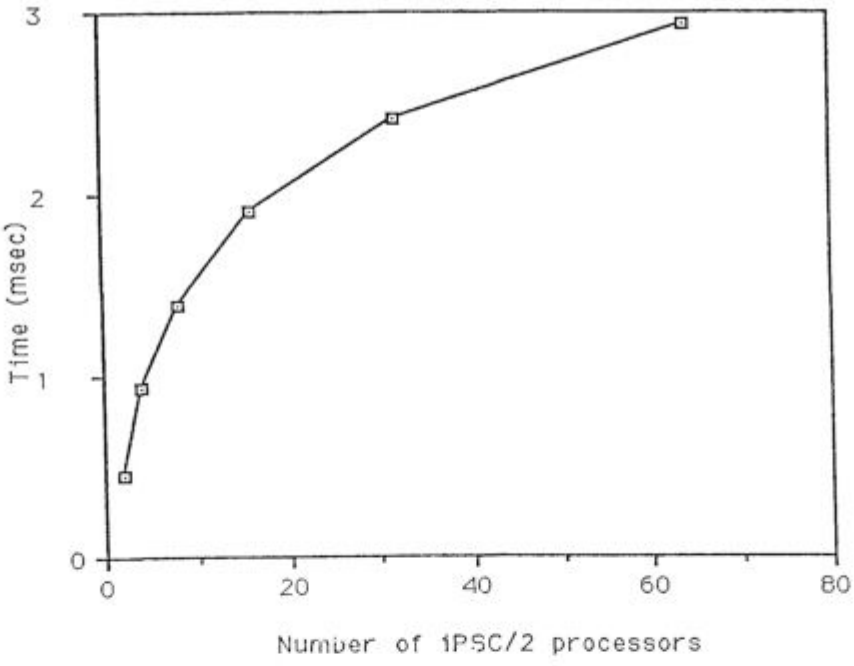
Πίνακας Α.22: iPSC-2: Αριθμητικές πράξεις

αυτή, διακόπτει την ροή της επεξεργασίας έως ότου όλοι οι κόμβοι φτάσουν επίσης στην εκτέλεση αυτής της εντολής. Είναι προφανές βέβαια πως όταν η εντολή αυτή καλείται μέσα σε μία διεργασία, ο χρόνος ολοκλήρωσής της εξαρτάται από τον πλέον καθυστερημένο κόμβικό επεξεργαστή. Οι μετρήσεις που παραθέτουμε έγιναν με μηδενικό υπολογιστικό φορτίο για όλους τους κόμβους.

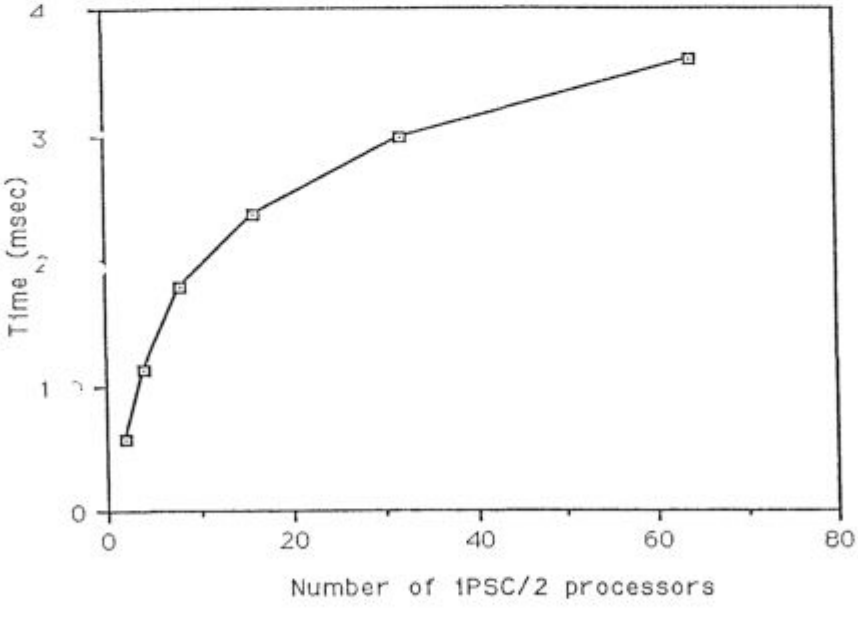
Το επόμενο διάγραμμα Α.7 είναι για την πράξη της άθροισης των τιμών μιάς ακέραιας μεταβλητής, που κρατούν όλοι οι επεξεργαστές, ενώ στον πίνακα Α.23 που ακολουθεί φαίνονται οι χρόνοι άθροισης μιας ακέραιας μεταβλητής και επιπλέον μιας πραγματικής (real) και μιας διπλής(double). Η διαφορά μεταξύ τους βλέπουμε πως δεν είναι σημαντική.

Number of Nodes	2	4	8	16	32	64
Global Sum of an integer (ms)	0.58	1.14	1.76	2.38	2.98	3.60
Global Sum of a real (ms)	0.58	1.16	1.77	2.38	2.99	3.61
Global Sum of a double (ms)	0.59	1.17	1.78	2.40	3.02	3.66

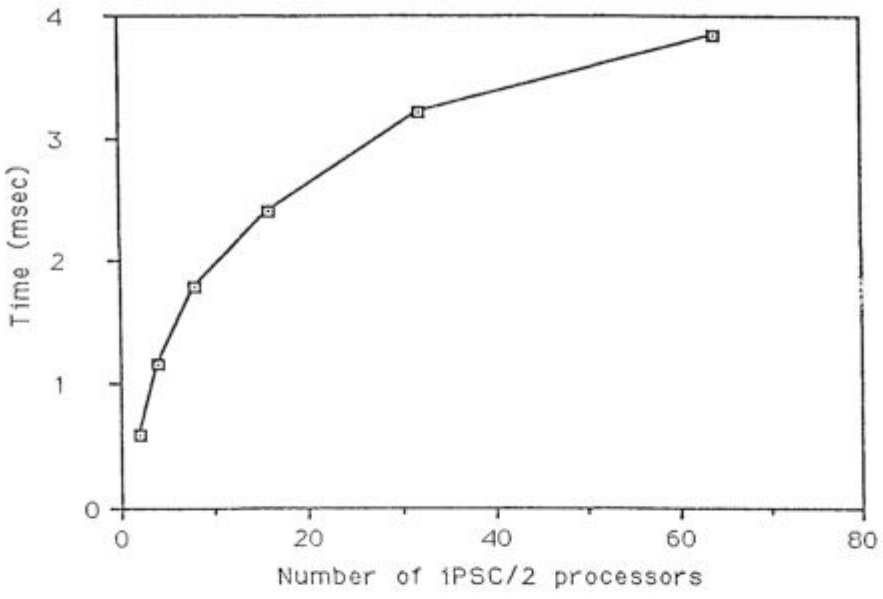
Πίνακας Α.23: iPSC/2: Καθολικές προσθέσεις μεταβλητών συναρτήσει του πλήθους των επεξεργαστών



Σχήμα A.6: iPSC/2: Η πράξη του συγχρονισμού των κόμβων συναρτήσει του πλήθους των επεξεργαστών



Σχήμα A.7: iPSC/2: Η πράξη της καθολικής (global) άθροισης ακέραιας μεταβλητής



Σχήμα Α.8: iPSC/2: Εύρεση μέγιστης τιμής ακεραίας μεταβλητής ως προς το πλήθος των επεξεργαστών

Στη συνέχεια έχουμε το διάγραμμα του Σχήματος Α.8 με τους χρόνους που απαιτεί η εύρεση της μέγιστης τιμής μιας μεταβλητής σε όλους τους κόμβους, όταν η τιμή αυτή είναι ακέραιος αριθμός, ενώ στον πίνακα Α.24 που ακολουθεί φαίνονται επιπλέον οι χρόνοι για την ίδια πράξη σάρωσης όταν η μεταβλητή είναι πραγματικός (real) και διπλός (double). Βλέπουμε πως η διαφορά είναι ξανά πολύ μικρή αλλά εν πάσει περιπτώσει η επεξεργασία των πραγματικών είναι μάλλον μικρότερη από εκείνη των ακεραίων και αυτό μάλλον λόγω του 80387 μαθηματικού συνεπεξεργαστή που διαθέτουν οι κόμβοι.

Number of Nodes	2	4	8	16	32	64
Max of integers (ms)	0.59	1.18	1.79	2.38	3.19	3.83
Max of reals (ms)	0.58	1.17	1.78	2.41	3.03	3.67
Max of doubles (ms)	0.59	1.17	1.79	2.41	3.05	3.70

Πίνακας Α.24: iPSC/2: Εύρεση μέγιστης τιμής αριθμητικών μεταβλητών ως προς το πλήθος των επεξεργαστών

Α.2.3 Συγκρίσεις των επιδόσεων των δύο μηχανών

Η υπολογιστική ισχύς κάθε ενός από τους επεξεργαστές του CM-2 είναι περίπου 16Kips. Οπως φαίνεται και από τις μετρήσεις είναι πολύ αδύνατοι με συνέπεια μεγάλες καθυστερήσεις στην επεξεργασία διεργασιών που είναι έντονα υπολογιστικές (computational intensive). Ιδιαίτερο πρόβλημα δημιουργείται για μεγάλες τιμές των μεταβλητών. Υπάρχει η δυνατότητα να ενισχυθούν οι επεξεργαστές με ειδικό Hardware (μαθηματικούς συνεπεξεργαστές), όμως επειδή αυτό θα πρέπει να επαναληφθεί χιλιάδες φορές, όσοι δηλαδή και οι επεξεργαστές, η επιβάρυνση στο κόστος του συστήματος θα είναι πολύ μεγάλη. Ακόμα εντονότερο γίνεται το πρόβλημα όταν οι επεξεργαστές έχουν να προσομοιώσουν μεγάλο ιδεατό πλέγμα. Αντίθετα οι επεξεργαστές των κόμβων του iPSC/2 είναι δύο τάξεις μεγέθους ισχυρότεροι και αντιμετωπίζουν πολύ ικανοποιητικά διεργασίες με υψηλές υπολογιστικές απαιτήσεις.

Οσον αφορά τώρα στα χαρακτηριστικά ενδοεπικοινωνίας, ο συσχετισμός των δύο μηχανών διαφοροποιείται. Το CM-2 έχει ένα πολύ γρήγορο δίκτυο με ταχύτητες μια τάξη μεγέθους υψηλότερες από το iPSC/2 και μάλιστα ανεξάρτητα σχεδόν από το πλήθος των επεξεργαστών. Ιδιαίτερα πλεονεκτικό εμφανίζεται το CM-2 στην περίπτωση μηνυμάτων μικρού μεγέθους αφού ο χρόνος εκκίνησης ενός μηνύματος είναι πολύ μικρός σε σχέση με τον χρόνο ολοκλήρωσης της επικοινωνίας. Στο iPSC/2 ο χρόνος εκκίνησης είναι περισσότερο από το 80επικοινωνίας, όπως μπορούμε να δούμε συγκρίνοντας το χρόνο μεταφοράς ενός μηνύματος χωρίς δεδομένα και το χρόνο για την μεταφορά μηνυμάτων διαφόρων μεγεθών

Στις προθεματικές πράξεις σάρωσεις που όπως μπορούμε να δούμε παίζουν ένα σημαντικό ρόλο στην επίδοση των παράλληλων αλγορίθμων υπάρχει σημαντική διαφοροποίηση τόσο όσον αφορά στην ποικιλία των πράξεων αυτών όσο και στις επιτυγχνόμενες ταχύτητες εκτέλεσης. Το CM-2 εμφανίζει πολύ ικανοποιητική ταχύτητα χωρίς μάλιστα να εξαρτάται ουσιαστικά από το πλήθος των φυσικών επεξεργαστών παρά μόνο από το πλήθος των ιδεατών στην περίπτωση που χρησιμοποιούνται τέτοιοι. Το CM-2 διαθέτει επίσης μεγαλύτερη ποικιλία τόσο από προθεματικές πράξεις όσο και από πράξεις συρρίκνωσης σε αντίθεση με το iPSC-2 που διαθέτει μόνο από τις τελευταίες. Το iPSC/2 έχει ικανοποιητική ταχύτητα στην εκτέλεση των πράξεων αυτών αλλά η ταχύτητα αυτή εξαρτάται από το πλήθος των επεξεργαστών.

Οι δύο μηχανές που διερευνήσαμε στην εργασία αυτή, παρουσιάζουν και πολλά άλλα δια-

φορετικά χαρακτηριστικά. Σε καμμία όμως περίπτωση δεν ήταν στην πρόθεσή μας η καθολική σύγκρισή τους. Πράγμα άλλωστε εκ φύσεως μάλλον αδύνατον. Θελήσαμε να δώσουμε μόνο μια χροιά των ομοιοτήτων και των διαφορών τους στον υπολογιστικό και επικοινωνιακό τομέα.

Βιβλιογραφία

[1] Livingstone MS. Art, Illusion and the Visual System. *Scientific American*, 257(11):78–85, Nov 1988.

[2] Walker J. What explains subjective-contour illusions, those bright spots that are not really there? *Scientific American*, 257(11):96–103, Nov 1988.

[3] Marr D. A theory of cerebellar cortex. *J. Physiology*, 202:437–470, 1969.

[4] Barlow HB. Single units and sensation: a neuron doctrine for perceptual psychology. *Perception*, 1:371–394, 1972.

[5] Marr D. VISION. *Freeman Publications, San Francisco*, 1982.

[6] Julesz B. Binocular depth perception of computer generated patterns. *Bell Syst. Tech. J.*, 39:1125–1162, 1960.

[7] Feldman J and Ballard D. Connectionist Models and their properties. *Cognitive Science*, 6:205–254, 1982.

[8] Tsotsos JK. A "complexity level" analysis of immediate vision. *Proc. First International Conference on Computer Vision*, pages 346–355, June 1987.

[9] Rosenfeld A. Recognizing unexpected objects: A proposed approach. *In Proc. of Image Understanding Workshop*, II:620–627, February 1987.

[10] From a graffito on a wall of a cafe in Austin, Texas. *National Geographic*, 177(3):109, March 1990.

[11] Amdahl GM. Validity of the single processor approach to achieving large scale computer capabilities. *Proc. AFIPS Spring Joint Comp. Conf. 30, Atlantic City*, pages 483–485, 1967.

[12] Janssen R. A note on superlinear speedup. *Parallel Computing*, 4:211–213, 1987.

[13] Parkinson. Parallel efficiency can be greater than unity. *Parallel Computing*, 3:261–262, 1986.

[14] Faber V, Lubeck OM, and White AB. Superlinear speedup of an efficient sequential algorithm is not possible. *Parallel Computing*, 3:259–260, 1986.

[15] Faber V, Lubeck OM, and White AB. Comments on the paper "parallel efficiency can be greater than unity". *Parallel Computing*, 4:209–210, 1987.

[16] Flatt HP and Kennedy K. Performance of Parallel Processors. *Parallel Computing*, 12:1–20, 1989.

[17] Johnsson L. Supercomputers: Past and Future. *YALEU/DCS/TR-778*, March 1990.

[18] Stout QF. Mapping vision algorithms to parallel architectures. *Proceedings of the IEEE*, 76(8):982–995, August 1988.

[19] Gerogiannis DC and Orphanoudakis SC. Efficient Embedding of Interprocessor Communications in Parallel Implementations of Intermediate Level Vision Tasks. *Proceedings of 10th International Conference on Pattern Recognition, Atlantic City, USA*, pages 368–372, June 1990.

[20] Gerogiannis DC, Orphanoudakis SC, and Johnsson SL. Histogram Computation on Distributed Memory Architectures. *Concurrency: Practice and Experience*, 1(2):219–237, December 1989.

[21] Mudge TN and Abdahl-Rahman T. Efficiency of feature dependent algorithms for the parallel processing of images. *Proceedings IEEE*, 1983.

[22] Gerogiannis DC and Orphanoudakis SC. Parallel implementation and load balancing requirements of intermediate level vision tasks. *Submitted to IEEE Trans. on Parallel and Distributed Systems*, 1991.

[23] Yalamanchili S and Aggarwal JK. A system organization for parallel image processing. *Pattern Recognition*, 18(1):17–29, 1985.

[24] Unger SH. A computer oriented toward spatial problems. *Proc. IRE*, 46:1744–1750, 1958.

[25] Levialdi S. Issues on parallel algorithms for image processing. *The Characteristics of Parallel Algorithms*, pages 191–208, 1987.

[26] Hord RM and Stevenson DK. The illiac IV architecture and its suitability for image processing. *Special Computer Architecture for Pattern Recognition, CRC Press*, pages 103–126, 1982.

[27] Duff MJ. Clip4, special computer architecture for pattern recognition. *IEEE Trans. Computers*, 29(9):836–840, September 1982.

[28] Marks P. Low level vision using an array processor. *Computer Graphics and Image Processing*, 14:281–292, 1980.

[29] Aladyev V. Mathematical theory of homogeneous structures and their applications. *Valgus, Tallinn, USSR*, 1980.

[30] Vollmer R. Algorithmen in zellularautomaten. *Teubner, Stuttgart, FRG*, 1979.

[31] Rosenfeld A. Picture languages: Formal models for picture recognition. *Academic Press*, 1979.

[32] Dyer CR and Rosenfeld A. Parallel image processing by parallel-augmented cellular automata. *IEEE Trans. PAMI*, 3:29–41, 1981.

[33] Rosenfeld A. Parallel image processing using cellular arrays. *Computer*, 16(1):14–20, January 1983.

[34] Batcher KE. Design of a massivelly parallel processor. *IEEE Transactions on Computers*, C-29(9):836–840, September 1980.

[35] Dubitizki T, Wu AY, and Ronsefeld A. Parallel region property computation by active quadtree networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-3(6):626–633, November 1981.

[36] Dyer CR and Rosenfeld A. Triangle cellular automata. *Information and Control*, 48:54–68, 1981.

[37] Rosenfeld P, Fiksel JR, and Holliger A. Intelligent graphs: Networks of finite automata capable of solving graph problems. *Graph Theory and Computing, RCR Edit.*, pages 219–265, 1979.

[38] Wu A and Rosenfeld A. Sequential and cellular graph automata. *Information Sciences*, 20:57–68, February 1980.

[39] Wu A and Rosenfeld A. Cellular graph automata. *Information and Control*, 42:305–353, 1979.

[40] Ronsefeld A and Wu AY. Parallel computers for region-level image processing. *Pattern Recognition*, 15(1):41–50, 1982.

[41] Ronsefeld A and Wu AY. Reconfigurable cellular computers. *Information and Control*, 50:64–84, 1981.

[42] Rieger C. ZMOB: Doing it in parallel. *Proc. Workshop Computer Architecture for PAIDM*, pages 273–278, 1981.

[43] Leutjen K, Gemmar P, and Ischen H. FLIP : A flexible multiprocessor system for image processing. *Proc. Fifth Int'l Conf. Pattern Recognition*, pages 326–328, 1980.

[44] Howard MD and Wallace RS. Hba vision architecture: Built and benchmarked. *IEEE trans. on PAMI*, 11(3):227–232, March 1989.

[45] De Carlini U and Villano U. A simple algorithm for clock synchronization in transputer networks. *Software-Practice and Experience*, 18(4):331–347, April 1988.

[46] Pountain D. OCCAM II. *BYTE*, pages 279–284, October 1989.

[47] Van Renterghem P. Transputers for industrial applications. *Concurrency : Practice and Experience*, 1(2):135–169, December 1989.

[48] Jones P and Murta A. Practical experience of run-time link reconfiguration in a multi-transputer machine. *Concurrency : Practice and Experience*, 1(2):171–189, December 1989.

[49] Kung HT. Why systolic architectures? *Computer*, pages 37–46, January 1982.

[50] Fortes JAB and Wah BW. Systolic arrays–from concept to implementation. *Computer*, 21(1):12–17, July 1987.

[51] Y. Bresler and A. Macovski. A hierarchical bayesian approach to reconstruction from projections of a multiple object 3-d scene. In *Proc. 7th Intl. Conf. on Pattern Recognition (ICPR)*, pages 455–457, Montreal, July 30 - August 2 1984.

[52] Nagin PA, Hanson AR, and Riseman M. Region relaxation in a parallel hierarchical architecture.

[53] Inagaki K, Kato T, Hiroshima T, and Sakai T. MASCYM: A hierarchical parallel image processing system for event-driven pattern understanding of documents. *Pattern Recognition*, 17(1):85–108, 1984.

[54] Stout QF. Sorting, merging, selecting and filtering on tree and pyramid machines. *Proceedings of Intern. Conf. on Parallel Processing*, pages 214–221, August 1983.

[55] Li Z-N and Uhr L. A pyramidal approach for the recognition of neurons using key features. *Pattern Recognition*, 19(1):55–62, 1986.

[56] Stewart CV and Dyer CR. Heyristic Scheduling Algorithms for PIPE. *Workshop on CAPAMI*, pages 75–82, October 1987.

[57] Yang YH and Sze TW. An evaluation study of six topologies of parallel computer architectures for scene matching. *Proceedings of Intern. Conf. on Parallel Processing*, pages 258–260, August 1983.

[58] Huntsberger TL and Wood WR. FLASH: A Parallel Architecture for computer vision in uncertain environments. *Proc. of IEEE Comp. Soc. Workshop on Comp. Arch. for Pattern Anal. and Image Database Management*, pages 280–283, November 1985.

[59] Fountain TJ. Array Architectures for Iconic and Symbolic Image Processing. *Proceedings CH2342-4 IEEE*, pages 24–33, 1985.

[60] Seitz CL. The cosmic cube. *Communications of the ACM*, 28(1):22–33, January 1985.

[61] Wilely P. A parallel architecture comes of age at last. *IEEE Spectrum*, 24(6):46–50, June 1987.

[62] Mudge TN. Vision algorithms for hypercube machines. *Proc. of IEEE Comp. Soc. Workshop on Comp. Arch. for Pattern Anal. and Image Database Management*, pages 225–230, November 1985.

[63] Ho C-T and Johnsson SL. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. *Tech. Rep. YALE/DCS/TR-483*, May 1986.

[64] Brown CM, Ellis CS, Feldman JA, LeBlancTJ, and Peterson GL. Research with the butterfly multicomputer. *Computer Science and Computer Engineering Review*, University of Rochester, 1984-1985.

[65] Reeves AP. Multicluster: An MIMD system for computer vision. *in Integrated Technology for Parallel Imag. Processing*, pages 39–56, 1985.

[66] Gottlieb A, Grishman R, Kruskal CP, McAuliffe KP, Rudolph L, and Snir M. The NYU ultracomputer-designing a MIMD shared memory parallel computer. *IEEE Trans. on Computers*, 32(2):175–189, February 1983.

[67] Chen S. A data flow computer architecture for markov image models. *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pages 75–79, November 1985.

[68] Rosenfeld A, Ornelas J, and Hung Y. Hough transform algorithms for mesh-connnected SIMD parallel processors. *Computer Vision, Graphics, and Image Processing*, 41:293–305, 1988.

[69] Fang Z, Li X, and Ni LM. Parallel algorithms for image template matching on hypercube SIMD computers. *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pages 33–40, November 1985.

[70] Irani KB and Wu WS. Minimazitation of interprocessor communication for parallel computations on a SIMD multicomputer interconnected with an Omega network. *Proc. of the Intern. Conf. on Parallel Processing*, pages 63–65, August 1984.

[71] Saltz J, Naik V, and Nicol D. Reduction of the effects of the communication delays in scientific algorithms on message passing MIMD architectures. *SIAM Journal of Scientific and Statistical Computing*, 8(1):118–134, January 1987.

[72] Tuomenoksa DL Kuehn JT, Siegel HJ and Adams III GB. The use and design of PASM*. *in Integrated Technology for Parallel Imag. Processing*, pages 133–152, 1985.

[73] Pfister GF and Norton VA. 'hot spot' contention and combining in multistage interconnection networks. *Proc. Int. Conf. on Paral. Proc.*, pages 790–797, August 1985.

[74] Pfister GF, Brantley WC, George DA, Harvey SL, Kleinfelder WJ, McAuliffe KP, Melton EA, Norton VA, and Weiss J. The IBM reasearch parallel processor prototype (RP3): introduction and architecture. *Proc. Int. Conf. on Paral. Proc.*, pages 764–771, August 1985.

[75] Brantley WC, McAuliffe KP, and Weiss J. RP3 processor-memory element. *Proc. Int. Conf. on Paral. Proc.*, pages 782–788, August 1985.

[76] Hillis WD. The connection machine: a computer architecture based on cellular automata. *Physica*, 10:213–228, 1984.

[77] Hillis WD. The connection machine. *MIT Press*, 1982.

[78] Connection machine: System front end: Symbolics. *The Connection Machine system*, Manual, 1987.

[79] Little JJ, Blelloch G, and Cass T. Parallel algorithms for computer vision on the connection machine. *In Proc. of Image Understanding Workshop*, II:628–638, February 1987.

[80] Blelloch GE. Scans as Primitive Parallel Operations. *IEEE trans. on Computers*, 38(11):1526–1538, November 1989.

[81] Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. *Proc. of National Academy of Science, USA*, 79:2554–2558, April 1982.

[82] Hopfield JJ and Tank DW. Computing with neural circuits: A model. *Science*, 237:625–633, August 1986.

[83] Hect-Nielsen R. Neurocomputing:picking the human brain. *IEEE Spectrum*, 25(3):36–41, March 1988.

[84] Sharma M, Patel JH, and Ahuja N. Netra: an architecture for a large scale multiprocessor vision system. *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pages 92–98, November 1985.

[85] Kohonen T. The "neural" phonetic typewriter. *Computer*, 21(3):11–21, March 1988.

[86] Stancovic JA. A perspective on distributed computer systems. *IEEE Transactions on Computers*, C-33(12):1102–1115, December 1984.

[87] Kushner TR and Rosenfeld A. A model of interprocesor communication for parallel image processing. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(4):600–618, July/August 1983.

[88] Lint B and Agerwala T. Communication issues in the design and analysis of parallel algorithms. *IEEE Transactions on Software Engineering*, SE-7(2):174–186, March 1981.

[89] Barak A and Paradise OG. MOS - a Distributed UNIX. *Dept. of Comp. Science, Hebrew Univ. Jerusalem*, June 1986.

[90] Fox GC. What Have We Learnt from Using Real Parallel Machines to Solve Real Problems. *C3P522*, California Institute of Technology, Pasadena, February 1988.

[91] Chan TF and Saad Y. Multigrid algorithms on the hypercube multiprocessor. *IEEE Transactions on Computers*, C-35(11):969–977, November 1986.

[92] Anderson S and Chen MC. Parallel branch-and-bound algorithms on the hypercube. *SIAM edition: Hypercube Multiprocessors*, pages 309–318, 1987.

[93] Schnorr CP and Shamir A. An optimal sorting algorithm for mesh connected computers. *Proceedings 0-89791-193-8 ACM*, pages 255–263, 1986.

[94] Guerra C. A VLSI algorithm for the optimal detection of a curve. *Proc. of IEEE Comp. Soc. Workshop on Comp. Arch. for Pattern Anal. and Image Database Management*, pages 197–202, November 1985.

[95] Deriche R. Optimal edge detection using recursive filtering. *First International Conference on Computer Vision*, pages 501–505, June 1987.

[96] Cypher RE, Sanz JLC, and Snyder L. The Hough transform has O(N) complexity on SIMD nxn mesh array architectures. *proc. THO203 IEEE*, pages 115–121, 1987.

[97] Lawrie DH. Access and alignment of data in an array processor. *IEEE Trans. on Computers*, 24(12):1145–1155, December 1975.

[98] May, Sen S, and Scherson ID. The distance bound for sorting on mesh-connected processor arrays is tight. *0272-5428 IEEE*, pages 255–263, 1986.

[99] O'Leary DP. Ordering schemes for parallel procesing of certain mesh problems. *SIAM J. Sci. Stat. Comput.*, 5(3):620–632, September 84.

[100] Atallah MJ and Kosaraju SR. Graph problems on a mesh-connected processor array. *Journal of ACM*, 31(3):648–666, July 1984.

[101] Das SK, Deo N, and Prasad S. Parallel graph algorithms for hypercube computers. *Parallel Computing*, 13:143–158, October 1990.

[102] Evans DJ and Stojmenovic I. On parallel computation of voronoi diagrams. *Parallel Computing*, 12:121–125, 1989.

[103] Woo J and Sahni S. Hypercube computing: Connected components. *The Journal of Supercomputing*, 3:209–234, 1989.

[104] Boxer L and Miller R. Dynamic computational geometry on meshes and hypercubes. *The Journall of Supercomputing*, 3:161–191, May 1989.

[105] Miller R and Stout QF. Geometric algorithms for digitized pictures on a Mesh-Connected computer. *IEEE PAMI*, 7(2):216–228, March 1985.

[106] Atallah MJ and Goodrich MT. Efficient parallel solutions to some geometric problems. *Purdue University*, CSD-TR-504, March 1986.

[107] Goodrich MT. Constructing arrangements optimally in Parallel. *Dep. of Comp. Science, Johns Hopkins Univ., Rep. JHU-90/06*, 1990.

[108] Atallah MJ. Parallel technics for computational geometry. *Purdue University*, CSD-TR-1020, June 1991.

[109] Atallah MJ and Tsay JJ. On the parallel decomposability of geometric problems. *Purdue University*, CSD-TR-873, March 1991.

[110] Umeo H and Asano T. Systolic Algorithms for Computational Geometry Problems-A Survey. *Computing, Springer-Verlag Ed.*, pages 19–40, 1989.

[111] Alnuweiri HM and Prasanna Kumar VK. Optimal image algorithms on an Orthogonally-Connected Memory-Based architecture. *10th Intern. Conf. on Pattern Recognition, Atlantic City, USA*, II:350–355, June 1990.

[112] Guibas L. Randomization in computational geometry. *Verbal communication*, July 1990.

[113] Fisher AL and Highnam PT. Computing the Hough Transform on a Scan Line Processor. *IEEE PAMI*, 11(3):262–265, March 1989.

[114] Ben-Tzvi D, Naqvi A, and Sandler M. Synchronous multiprocessor implementation of the Hough transform. *Comp. Vis., Graph., and Im. Processing*, 52(3):437–446, December 1990.

[115] Francis Nd, Nudd GR, Atheryon TJ, Kerbyson DJ, Packwood RA, and vaudin J. Performance evaluation of the hierarchical Hough transform on an associative M-SIMD architecture. *10th Intern. Conf. on Pattern Recognition, Atlantic City, USA*, II:509–511, June 1990.

[116] Balasubramanian V and Banerjee P. Tradeoffs in the design of efficient algorithm-based error detection schemes for hypercube multiprocessors. *IEEE trans. on Software Engineering*, 16(2):183–196, February 1990.

[117] Johnsson SL and Ho CT. Shuffle permutations on boolean cubes. *YELAU/DCS/TR-653*, October 1988.

[118] Seidel SR and George WL. A Sorting algorithm for Hypercubes with d-Port Communication. *Dep. of Math. and Comp. Sciences, Mich. Tech. Univ., CS-TR 87-02*, January 1987.

[119] Miller R and Stout QF. Mesh computer algorithms for line segments and simple polygons. *Proc. of the Intern. Conf. on Parallel Processing*, pages 282–285, August 1987.

[120] Chakrabarti C and Jaja J. A parallel algorithm for Template Matching on an SIMD mesh connected computer. *10th Intern. Conf. on Pattern Recognition, Atlantic City, USA*, II:362–367, June 1990.

[121] Prasanna-Kumar VK and Reisis DI. Image computations on meshes with multiple broadcast. *IEEE trans. on PPAMI*, 11(11):1194–1201, November 1989.

[122] Ranka S and Sahni S. Convolution on mesh connected multicomputers. *IEEE trans. on PAMI*, 12(3):315–318, March 1990.

[123] Cvetanovic Z. The Effects of Problem Partitioning, Allocation, and Granularity on the Performance of Multiple-Processor Systems. *IEEE Transactions on Computers*, C-36(4):421–432, April 1987.

[124] Li H, Wang CC, and Lavin M. Struvtered Processes: a new language attribute for better interaction of parallel architecture and language. In *Proc. IEEE 0190-3918*, pages 247–254, 1985.

[125] Deminet J. Experience with multiprocessor algorithms. *IEEE Transactions on Computers*, C-31(4):278–287, April 1982.

[126] Weems CC, Rana D, Hanson AR, Riseman EM, Shu DB, and Nash G. An overview of architecture research for image understanding at the University of Massachusetts. *10th Intern. Conf. on Pattern Recognition, Atlantic City, USA*, II:379–384, June 1990.

[127] Fischler MA and Firschein O. Parallel guessing: A strategy for high-speed computation. *Pattern Recognition*, 20(2):257–263, 1987.

[128] Fisher DC. Your favorite parallel algorithms might not be as fast as you think. *IEEE Transactions on Computers*, 37(2):211–213, February 1988.

[129] Eager DL, Lazowska ED, and Zahorjan J. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, SE-12(5):662–675, May 1986.

[130] Tantawi AN and Towsley D. Optimal static load balancing in distributed computer. *Journal of the ACM*, 32(2):445–465, April 1985.

[131] Berger MJ and Bokhari SH. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, C-36(5):570–580, May 1987.

[132] Chu E and George A. Gaussian elimination with partial pivoting and load balancing on a multiprocessor. *Parallel Computing*, 5(1+2):65–74, July 1987.

[133] Stone HS. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, SE-3(1):85–93, January 1977.

[134] Saltz JH Iqbal MA and Bokhari SH. A comparative analysis of static and dynamic load balancing strategies. *Proc. of the Intern. Conf. on Parallel Processing*, pages 1040–1051, August 1986.

[135] Chow Y and Kohler WH. Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Transactions on Computers*, c-28(5):354–361, May 1979.

[136] Nicol D and Saltz J. Dynamic remapping of parallel computations with varying resource demands. *Tech. Rep. 86-45, ICASE*, July 1986.

[137] Nicol DM and Saltz JH. Schedules for mapping irregular parallel computations. *Tech. Rep. 87-52, ICASE*, September 1987.

[138] Fox GC. A review of Automatic Load Balancing and Decomposition methods for the Hypercube. *Minessota Institute for Mathematics and its Application Workshop*, November 1986.

[139] Plaxton CG. Load Balancing on the Hypercube and Shuffle-Exchange. *Dept. Comp. Science, Stanford Univ.*, 1990.

[140] Marinescu DC and Rice JR. Non-Algorithmic load imbalance effects for domain decomposition methods on a Hypercube. *Comp. Science Dept., Purdue Univ., Techn. Rep. CSD-TR-1000*, January 1989.

[141] Marinescu DC and Rice JR. Synchronization and Load imbalance effects in Distributed memory multiprocessor systems. *Comp. Science Dept., Purdue Univ., Techn. Rep. CSD-TR-1000*, July 1990.

[142] Hummel RA and Zhang K. Dynamic processor allocation for parallel algorithms in image processing. *Courant Institute of Mathematical Sciences, Tech. Rep. No. 286, Robotics Rep. No. 94*, January 1987.

[143] Stone HS. Critical load factors in two-processor distributed systems. *IEEE trans. Soft. Eng.*, 4(3):254–259, May 1978.

[144] Gao C, Liu JWS, and Railey M. Load balancing algorithms in homogeneous distributed systems. *Proc. of the Intern. Conf. on Parallel Processing*, pages 302–306, August 1984.

[145] Barak A and Shiloh A. A Distributed Load-balancing policy for a multicomputer. *Software-Practice and Experience*, 15(9):901–913, April 1984.

[146] Baumgartner KM and Wah BW. Load balancing protocols on a local computer system with a multiaccess network. *Proc. of the Intern. Conf. on Parallel Processing*, pages 851–858, August 1987.

[147] Lu H and Carey MJ. Load -balanced task allocation in locally distributed computer systems. *Proc. of the Intern. Conf. on Parallel Processing*, pages 1037–1039, August 1986.

[148] Sadayppan P, Ercal F, and Ramanujam J. Cluster partitioning approaches to mapping parallel programs onto a hypercube. *Parallel Computing*, 13:1–16, 1990.

[149] Fox GC and Furmanski W. Load Balancing Loosely Synchronous Problems with a Neural Network. *C3P363B*, California Institute of Technology, Pasadena, February 1988.

[150] Polychronopoulos CD and Banerjee U. Speedup bounds and processor allocation for parallel programs on multiprocessors. *Proc. of the Intern. Conf. on Parallel Processing*, pages 961–968, August 1986.

[151] Bokhari SH. Partitioning problems in parallel, pipelined, and distributed computing. *IEEE Transactions on Computers*, 37(1):48–57, January 1988.

[152] Bokhari SH. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE trans. on Software Engineering*, 7(6):583–589, November 1981.

[153] Fernandez EB and Bussell B. Bounds on the number of processors and time for multiprocessor optimal schedules. *IEEE Transactions on Computers*, C-22(8):745–751, August 1973.

[154] Rao GS, Stone HS, and Hu TC. Assignment of task in a Distributed processor system with limited memory. *IEEE Trans. on Computers*, 28(4):291–299, April 79.

[155] Saltz JH. Analysis of parameterized methods for problem partioning. *Tech. Rep. YALEU/DCS/RR-537*, May 1987.

[156] McDowell CE and Appelbe WF. Processor scheduling for linearly connected parallel processors. *IEEE Transactions on Computers*, C-35(7):632–638, July 1986.

[157] Ward MO and Romero DJ. Assigning parallel-executable, intercommunicating subtasks to processors. *Proc. of the Intern. Conf. on Parallel Processing*, pages 392–394, August 1984.

[158] Polychronopoulos CD and Banerjee U. Processor allocation for horizontal and vertical parallelism and related speedup bounds. *IEEE Transactions on Computers*, C-36(4):410–420, April 1987.

[159] Efe K. Heuristic models of task assignment scheduling in distributed systems. *Computer*, 15(6):50–56, June 1982.

[160] Hwang K, Croft WJ, Goble GH, Wahn BW, Briggs FA, Simmons WR, and Coats CL. A unix-based local computer network with load balancing. *Computer*, 15(4):55–65, April 1982.

[161] Sadayappan P and Ercal F. Cluster-partitioning approaches to mapping parallel programs onto a hypercube. *Supercomputing, 1st International Conference, Athens Greece*, June 1987.

[162] Nicol DM and Saltz JH. Principles for problem aggregation and assignment in medium scale multiprocessors. *Tech. Rep. 87-39, ICASE*, September 1987.

[163] Nicol DM and Reynolds PF. Optimal dynamic remapping of data parallel computations. *IEEE Trans. on Computers*, 39(2):206–219, February 1990.

[164] Goldberg AP and Jefferson DR. Transparent process cloning: A tool for load management of distributed programs. *Proc. of the Intern. Conf. on Parallel Processing*, pages 728–734, August 1987.

[165] Siegel LJ, Siegel HJ, and Swain PH. Performance measures for evaluating algorithms for SIMD Machines. *IEEE Trans. on Software Eng.*, 8(4):319–331, July 1982.

[166] Cypher RE, Sanz JLC, and Snyder L. Algorithms for Image Component Labeling on SIMD Mesh Connected Computers. *THO203 IEEE*, pages 276–281, 1990.

[167] Saad Y and Schultz MH. Topological properties of hypercube. *IEEE Trans. on Computers*, 37(7):867–872, July 1988.

[168] Haessig K and Jenny CJ. Partitioning and allocation computational objects in distributed computing systems. *Proc. of IFIP Congress*, pages 593–598, 1980.

[169] Chu WW, Holloway LJ, Lan M-T, and Efe K. Task allocation in distributed data processing. *Computer*, pages 57–69, November 1980.

[170] Houstis CE, Houstis EN, Rice JR, Samartzis SM, and Alexandrakis DL. A user guide to the algorithm mapper: A system for modeling and evaluating parallel applications/architecture pairs. *Technical Report CSD-TR-793*, August 1988.

[171] Weil F, Jamieson L, and Delp E. Some aspects of an image understanding Database for an Intelligent Operating System. *Workshop on CAPAMI*, pages 203–208, October 1987.

[172] Preparata FP and Shamos MI. Computational Geometry, an Introduction. *Springer-Verlag*, pages 106–110, 1985.

[173] Andrews AM. Another efficient algorithm for convex hulls in two dimensions. *Info. Proc. Lett.*, (9):216–219, 1979.

[174] Graham RL. An efficient algorithm for determining the convex hull of a finite planar set. *Info. Proc. Lett.*, (1):132–133, 1972.

[175] Damianakis KA, Argyros AA, and Orphanoudakis KS. Parallel implementations of image analysis tasks. *Proceedings of 3rd Panhellenic Conference on Computer Science, Athens*, May 1991.

[176] Levialdi S. On shrinking binary picture patterns. *Communications of ACM*, 15I:7–10, 1972.

[177] Sunwoo MH, Baroody BS, and Agarwal JK. A Parallel Algorithm for Region Labeling. *Workshop on CAPAMI*, pages 27–34, October 1987.

[178] Rosenfeld A and Kak AC. Digital Picture Processing. *Academic Press*, 2:152–180, 1982.

[179] Damianakis AK. Image Analysis Using Relaxation Labeling. *Master Thesis, Comp. Science Dept., Univ. of Crete*, 1988.

[180] Pavlidis T. Algorithms for Graphics and Image Processing. *Computer Science Press*, pages 195–215, 1982.

[181] iPSC/2 User's Guide. *INTEL Corporation Manual*, March 1988.