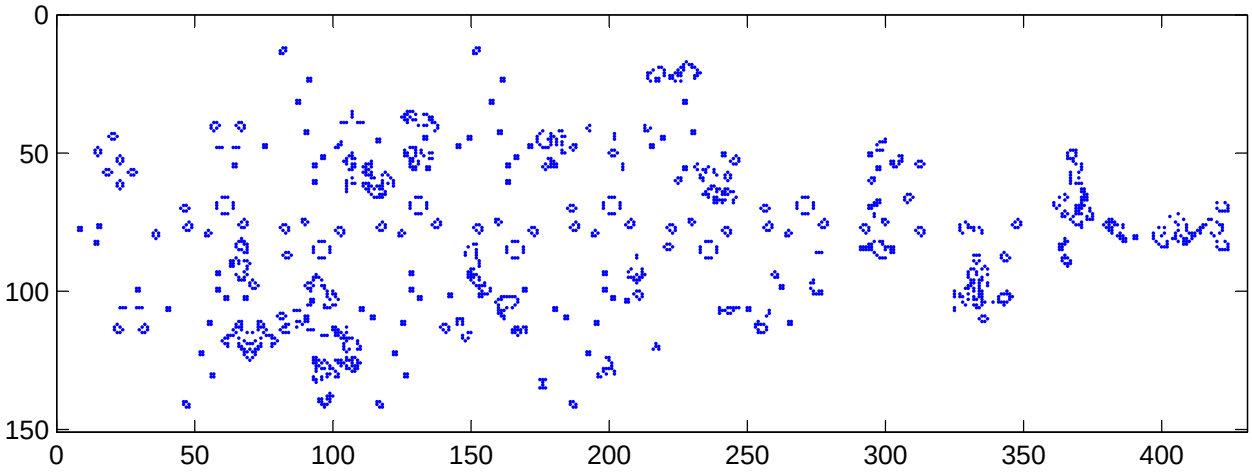
Εθνικο και Καποδιστριακο Πανεπιστημιο Αθηνων
Τμημα Πληροφορικης και Τηλεπικοινωνιων
ΠΜΣ Υπολογιστικη Επιστημη

Μάθημα: *«Τεχνολογία Παράλληλων Υπολογιστικών Συστημάτων»*
Διδάσκων: *Ιωάννης Κοτρώνης*
Χειμερινό Εξάμηνο 2006-2007

Το «Παιχνίδι της Ζωής» του **John Conway**

Παράλληλη υλοποίηση

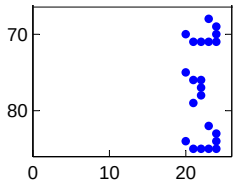


Αθανάσιος Πολυμενέας, M686
grad0686@di.uoa.gr

Άγγελος Μαντζαφλάρης, M901
amantzaf@math.uoa.gr

Σεπτέμβριος 2007

Εικόνα Εξωφύλλου: Το κλασσικό σχέδιο(pattern) *καπνός τρένου* (puffer train) μετά από 800 γενεές (δηλαδή εφαρμογές των κανόνων μετάβασης του *Παιχνιδιού της Ζωής*). Στο δεξί άκρο φαίνεται ένα ακριβές αντίγραφο του αρχικού σχεδίου, μετατοπισμένο 400 θέσεις δεξιά από το σημείο που ξεκίνησε, δηλαδή το αριστερό άκρο της εικόνας. Η ονομασία puffer train έχει να κάνει με τον τρόπο που το αρχικό σχέδιο μετακινείται προς τα εμπρός αφήνοντας ίχνη «καπνού» στο διάβα του. Το στιγμιότυπο υπολογίστηκε με την υλοποίησή μας σε ένα πλέγμα 2×3 επεξεργαστών. Το αρχικό σχέδιο φαίνεται παρακάτω σε μεγέθυνση.



Περίληψη

Στο *Παιχνίδι της Ζωής*(Game of Life), ένα παιχνίδι κυτταρικών αυτομάτων(cellular automata) για υπολογιστή, έχουμε ένα πλέγμα από *μπλε ή λευκά* τετράγωνα που αλλάζουν συνεχώς χρώμα ακολουθώντας κάποιους μηχανικούς κανόνες. Τελικά, αυτό που προκύπτει είναι ένας εντυπωσιακός, συνεχώς μεταβαλλόμενος κόσμος στην οθόνη του υπολογιστή μας. Σε αυτήν την εργασία υλοποιούμε το Παιχνίδι *της Ζωής* στο SPMD μοντέλο παράλληλου προγραμματισμού, χρησιμοποιώντας τη γλώσσα C και τη βιβλιοθήκη MPI. Μετά το μεθοδολογικό σχεδιασμό παρουσιάζουμε τα κυριότερα σημεία της πλήρως παραμετροποιήσιμης υλοποίησής μας καθώς και (θεωρητικές και πειραματικές) μετρήσεις απόδοσης και επιτάχυνσης.

1 Εισαγωγή

Το *Παιχνίδι της Ζωής* αναπτύχθηκε το 1970 από τον John Horton Conway στο Πανεπιστήμιο του Cambridge. Το *Παιχνίδι* είναι ένα μαθηματικό θαύμα όπου από τη μια επαναλαμβάνονται μερικοί άβουλοι κανόνες και από την άλλη δημιουργούν μία θεαματική, ζωντανή πολυπλοκότητα. Το *παιχνίδι* παίζεται σε ένα περιοδικό πλέγμα $N \times M$ θέσεων. Κάθε θέση έχει 8 γειτονικές και μπορεί είτε να είναι κατειλημμένη από έναν οργανισμό είτε όχι. Ξεκινώντας από κάποιες αρχικές τιμές, η επόμενη κατάσταση υπολογίζεται χρησιμοποιώντας τους ακόλουθους κανόνες:

- Εάν ένας οργανισμός (κατειλημμένη θέση) έχει 0 ή 1 γειτονικούς οργανισμούς, ο οργανισμός πεθαίνει από μοναξιά.
- Εάν ένας οργανισμός έχει 2 ή 3 γειτονικούς οργανισμούς, ο οργανισμός επιζεί στην επόμενη γενεά.
- Εάν ένας οργανισμός έχει 4 έως 8 γειτονικούς οργανισμούς, ο οργανισμός πεθαίνει λόγω υπερπληθυσμού.
- Εάν μία μη κατειλημμένη θέση έχει ακριβώς 3 γειτονικούς οργανισμούς, αυτή η θέση θα καταληφθεί στην επόμενη γενεά από έναν νέο οργανισμό, δηλαδή ένας οργανισμός γεννιέται.

Οι κανόνες αυτοί εφαρμόζονται σε όλα τα κύτταρα ταυτόχρονα. Κάθε φορά όλος ο πίνακας υπολογίζεται εκ νέου, με αποτέλεσμα να δημιουργείται μια νέα συνολική κατάσταση που ονομάσαμε γενεά. Καθώς το πρόγραμμα υπολογίζει διαδοχικές γενεές, τα κύτταρα φαίνεται να συμπεριφέρονται με μη τυχαίο τρόπο. Αν ξεκινήσουμε από διαφορετικές αρχικές καταστάσεις, θα εμφανιστούν διαφορετικά πρότυπα συμπεριφοράς για αυτά, όπως ομάδες παλλόμενων κυττάρων (περιοδικά συστήματα), ομάδες που εξαφανίζονται (ασταθή συστήματα) και σχηματισμοί που εξελίσσονται με την πάροδο του χρόνου. Η χρήση των κυτταρικών αυτομάτων μας επιτρέπει να «παρατηρήσουμε» περιοδικά ή ασταθή συστήματα, αφού αυτά δεν συναντιούνται στη φύση (όλοι οι οργανισμοί θεωρούνται ευσταθή συστήματα).

Στην ενότητα 2 αναπτύσσουμε το σχεδιασμό της εφαρμογής και στην ενότητα 3 προχωρούμε στην υλοποίηση στο MPI. Στην ενότητα 4 αναλύουμε τον παράλληλο αλγόριθμο σε θεωρητικό επίπεδο και ακολουθούν μετρήσεις, γραφικές παραστάσεις, αξιολόγηση του κώδικα και σύντομα συμπεράσματα στις ενότητες 5 και 6.

Παραθέτουμε μερικά παραδείγματα εκτέλεσης του προγράμματος στο Α′, τον πλήρη πηγαίο κώδικα στο Β′ και τους πίνακες μετρήσεων στο παράρτημα Γ′.

2 Σχεδιασμός

Ο σχεδιασμός που παρουσιάζεται ακολουθεί τη μεθοδολογία που αναπτύσσεται στο [7]. Πιο συγκεκριμένα, προσανατολιζόμαστε στο SPMD(Single Program Multiple Data) μοντέλο παραλληλίας.

Αρχικά διατυπώνουμε το μαθηματικό πρόβλημα και τον ακολουθιακό(sequential) αλγόριθμο που απορρέει από αυτό. Κατόπιν παραλληλοποιούμε τον αλγόριθμο διευθετώντας τα ζητήματα του διαμερισμού, της επικοινωνίας και της ανάθεσης στους επεξεργαστές. Οι επιλογές γίνονται με γνώμονα την αποδοτικότητα, την ταχύτητα και την κλιμάκωση.

2.1 Ακολουθιακός αλγόριθμος

Προχωρούμε στη διατύπωση του μαθηματικού μοντέλου και του ακολουθιακού αλγορίθμου. Η είσοδος του προβλήματός μας είναι ένας πίνακας $C \in \{0,1\}^{N\times M}$ με 0 (για τις ελεύθερες θέσεις) και 1 (για τις θέσεις κατειλημμένες με οργανισμούς). Ο αλγόριθμος πρέπει να υπολογίζει μια ακολουθία πινάκων $C_1, C_2, \ldots, C_i, C_{i+1}, \ldots$ κάθε ένας από τους οποίους αναπαριστά μια γενεά. Δεν υπάρχει ανάγκη για αποθήκευση όλων των γενεών, όμως είναι απαραίτητο να αποθηκεύσουμε δυο διαδοχικούς πίνακες στη μνήμη, διότι η αποθήκευση των τιμών του C_{i+1} δε μπορεί να γίνει στον πίνακα C_i, αφού η παλιά τιμή πρέπει να είναι διαθέσιμη για την εύρεση της νέας τιμής των γειτονικών θέσεων. Έτσι διατηρούμε δυο διαδοχικές γενεές σε δυο πίνακες $C,\ C'$, επικαλύπτοντας κάθε φορά την προγενέστερη γενεά με τη νέα. Η γενεά C' προκύπτει από την C από τη σχέση:

$$c'_{ij} = \begin{cases} 0 & , \text{ αν } s_{ij} \leq 1 \ \text{ ή } \ s_{ij} \geq 4 \\ c_{ij} & , \text{ αν } s_{ij} = 2 \\ 1 & , \text{ αν } s_{ij} = 3 \end{cases} \qquad i = 0,1,\ldots,N-1,\ j = 0,1,\ldots,M-1$$

όπου

$$s_{ij} = c_{i-1,j} + c_{i-1,j+1} + c_{i,j+1} + c_{i+1,j+1} + c_{i+1,j} + c_{i+1,j-1} + c_{i,j-1} + c_{i-1,j-1}$$

το άθροισμα των οκτώ γειτονικών θέσεων του πίνακα. Επειδή το πλέγμα είναι περιοδικό, οι δείκτες στους όρους του αθροίσματος λαμβάνονται modulo την αντίστοιχη διάσταση (δηλαδή $i := i \mod N,\ j := j \mod M$) ώστε να μην εμφανίζονται όροι εκτός του πίνακα. Η επόμενη γενεά υπολογίζεται με χρήση του C' και επικαλύπτει τον πίνακα C.

Μπορούμε τώρα να διατυπώσουμε τον ακολουθιακό αλγόριθμο:

1. Ανάγνωση αρχικού $N \times M$ πίνακα C από αρχείο ή δημιουργία τυχαίου $0-1$ πίνακα.

2. Εκτέλεση υπολογισμών, δηλαδή υπολογισμός του αθροίσματος s_{ij} και κατόπιν της τιμής $c'_{ij},\ i = 0,1,\ldots,N-1,\ j = 0,1,\ldots,M-1$ και αποθήκευση στον πίνακα C'.

3. Έλεγχος για στασιμότητα ($C' = C$) ή για αφανισμό του πληθυσμού($C' = \mathbf{0}$).

4. Εναλλαγή των πινάκων $C,\ C'$ και επιστροφή στο βήμα 2(συνέχεια στην επόμενη γενιά).

Το υπολογιστικό κόστος για μια γενεά είναι $T_1 = t_c NM$, όπου t_c το κόστος υπολογισμού του στοιχείου c'_{ij}. Οι κύριες απαιτήσεις σε μνήμη είναι $2NM$ για τους πίνακες $C,\ C'$.

2.2 Διαμερισμός προβλήματος (Decomposition)

Η πιο φυσική επιλογή για το πρόβλημά μας είναι η διαμέριση δεδομένων(domain decomposition) και ειδικότερα ο διαμερισμός σε πλέγμα (η εναλλακτική τεχνική της διαμέρισης διαδικασιών−functional decomposition τελικά οδηγεί στις ίδιες επιλογές). Ειδικότερα, επιλέγουμε μια διδιάστατη διαμέριση του πίνακα C, δηλαδή τον χωρίζουμε σε blocks. Με τον τρόπο αυτό ευνοείται η κλιμάκωση(scalability) του αλγορίθμου μας, όπως θα δούμε αργότερα.

Θεωρούμε μια στοιχειώδη εργασία(task) για κάθε θέση c_{ij} του πίνακα(Σχήμα 1). Το υπολογιστικό φορτίο που αντιστοιχεί σε αυτή τη θέση, άρα και στην εργασία μας, είναι ο υπολογισμός της νέας τιμής c'_{ij} κατά την επόμενη γενιά. Με τον τρόπο αυτό ορίζουμε NM εργασίες.

Όλες οι στοιχειώδεις εργασίες είναι ισομεγέθεις, και έχουν πλήθος ακριβώς ίσο με το μέγεθος του προβλήματος. Ο διαμερισμός που έγινε εισάγει αρκετές τάξεις περισσότερες στοιχειώδεις εργασίες σε σχέση με το πλήθος των επεξεργαστών, αφού το τελευταίο θα είναι πολύ μικρότερο από το μέγεθος του προβλήματος. Δεν υπάρχουν πλεονάζοντες υπολογισμοί ή επιπρόσθετη ανάγκη αποθήκευσης σε σχέση με τον ακολουθιακό αλγόριθμο, ενώ το μέγεθος των στοιχειωδών εργασιών παραμένει το ίδιο με την αύξηση του μεγέθους του προβλήματος. Οι ιδιότητες αυτές είναι επιθυμητές και υποδηλώνουν ότι η επιλογή μας θα αποβεί αποδοτική.

2.3 Επικοινωνία (Communication)

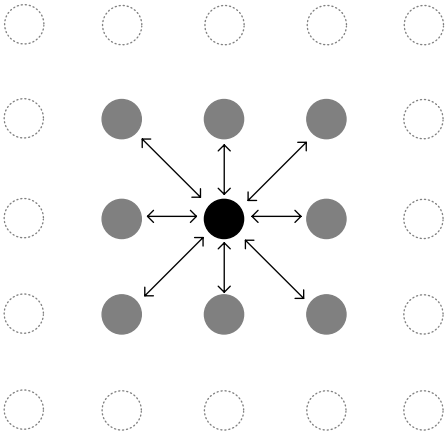
Η επικοινωνία που απαιτεί ο αλγόριθμος είναι δυο ειδών· διεξάγονται *ένας προς έναν* επικοινωνίες μεταξύ γειτονικών εργασιών για τον υπολογισμό της νέας τιμής, αλλά και δυο καθολικές(global) *όλοι προς όλους* επικοινωνίες για τον έλεγχο των συνθηκών τερματισμού.

Η *ένας προς έναν* επικοινωνία διεξάγεται τοπικά. Επειδή κάθε στοιχειώδης εργασία απαιτεί δεδομένα από οκτώ γειτονικές, ορίζουμε οκτώ κανάλια επικοινωνίας(Σχήμα 1) μεταξύ αυτής και των γειτόνων της. Από κάθε κανάλι διεξάγεται μια αποστολή και μια λήψη σε κάθε χρονική στιγμή, δηλαδή κάθε στοιχειώδης εργασία παράγει οκτώ μηνύματα και δέχεται οκτώ μηνύματα για τον υπολογισμό μιας γενεάς. Η επικοινωνία είναι δομημένη(structured), αφού οι στοιχειώδεις εργασίες βρίσκονται σε πλέγμα. Η γειτονία δε μεταβάλλεται με την πάροδο του χρόνου (δηλαδή των γενεών), άρα η επικοινωνία μας είναι στατική(static). Σε κάθε γενεά οι αποστολές και οι λήψεις μηνυμάτων διεξάγονται αμοιβαία μεταξύ των εργασιών, δηλαδή η επικοινωνία είναι σύγχρονη(synchronous). Τα μηνύματα είναι συνολικά $8NM$ έχουν μέγεθος ίσο με μια λέξη(word) του υπολογιστή.

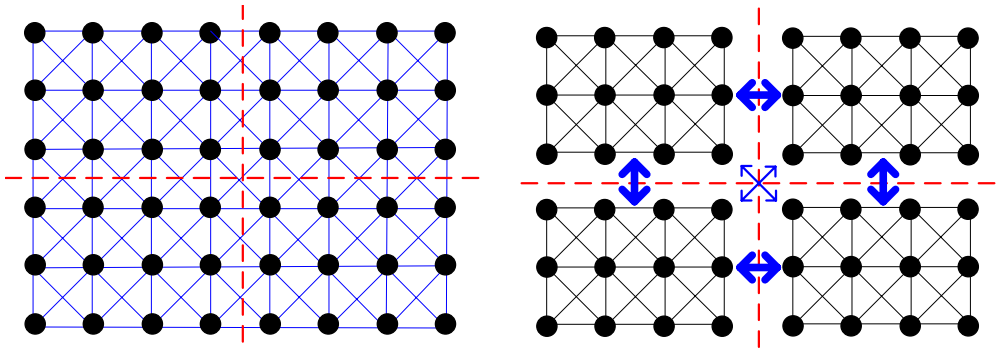
Η καθολική επικοινωνία αποτελείται από δυο μηνύματα από κάθε διεργασία, τα οποία έχουν αποδέκτη όλες τις υπόλοιπες διεργασίες. Δεν πρόκειται για επικεντρωμένη επικοινωνία, καθώς όλες οι εργασίες λαμβάνουν και στέλνουν μηνύματα, και μάλιστα δομημένα, στατικά και σύγχρονα. Κατά την υλοποίηση η επικοινωνία αυτή εκφράζεται με την εύρεση (σε όλους τους κόμβους) ενός μέγιστου και ενός ελάχιστου μεταξύ τιμών που βρίσκονται κατανεμημένες σε όλες τις διεργασίες. Για τη βελτιστοποίηση αυτών των υπολογισμών επιστρατεύουμε (τις ενσωματωμένες στη βιβλιοθήκη MPI) τεχνικές διαίρει και βασίλευε (divide and conquer) ώστε να ενισχύσουμε την παραλληλία.

2.4 Συσσώρευση & Ανάθεση (Agglomeration & Mapping)

Επειδή ο σχεδιασμός μας ακολουθεί το μοντέλο SPMD κρίθηκε επαρκές να διευθετήσουμε τα βήματα της συσσώρευσης και ανάθεσης μαζί.



Σχήμα 1: Δομή στοιχειωδών εργασιών(tasks) και κανάλια επικοινωνίας στο *Παιχνίδι της Ζωής*. Κάθε κύτταρο λαμβάνεται ως ένα task το οποίο απαιτεί επικοινωνία με οκτώ γειτονικά του. Στο σχήμα φαίνονται τα κανάλια επικοινωνίας που απαιτεί το μεσαίο (μαύρο) κύτταρο.



Σχήμα 2: Ανάθεση εργασιών (mapping) σε ένα 2×2 πλέγμα επεξεργαστών. Οι κόκκινες γραμμές υποδεικνύουν τα όρια των επεξεργαστών. Αριστερά βλέπουμε ότι το υπολογιστικό φορτίο κατανέμεται ομοιόμορφα (αν δε λάβουμε υπόψιν τυχόν άνιση κατανομή των ζωντανών κυττάρων). Δεξιά φαίνεται η μείωση του αριθμού των μηνυμάτων μεταξύ των επεξεργαστών με την αύξηση του μεγέθους του μηνύματος. Για απλότητα δεν καταγράφουμε τις οριακές επικοινωνίες του αναδιπλούμενου (περιοδικού) πλέγματος.

Παρατηρούμε ότι οι στοιχειώδεις εργασίες έχουν το ίδιο υπολογιστικό κόστος και απαιτούν την ίδια επικοινωνία. Η (διδιάστατη) συσσώρευση τους σε μεγαλύτερα tasks μειώνει τον αριθμό των μηνυμάτων αυξάνοντας το μέγεθός τους στην οριζόντια και κάθετη διάσταση, ενώ η διαγώνια επικοινωνία παραμένει μοναδιαίου μεγέθους. Επιπλέον αυτή η αύξηση της συσπείρωσης (granularity) και στις δυο διαστάσεις δεν περιορίζει τη δυνατότητα κλιμάκωσης. Για τους λόγους αυτούς επιλέγουμε να ακολουθήσουμε αυτή τη στρατηγική για το σχηματισμό διεργασιών και την ανάθεση στους επεξεργαστές(Σχήμα 2).

Η εν λόγω ανάθεση καθιστά το κόστος παραλληλοποίησης του ακολουθιακού αλγορίθμου μικρό· ο πυρήνας των υπολογισμών παραμένει ο ίδιος, απλώς εφαρμόζεται στο τοπικό κομμάτι αντί όλου του πίνακα. Το επιπλέον προγραμματιστικό κόστος αποτελείται από την υλοποίηση της επικοινωνίας, το οποίο είναι επίσης μικρό λόγω της καλά δομημένης τοπικής επικοινωνίας του αλγορίθμου. Τέλος δεν υπάρχει ανάγκη για επανάληψη υπολογισμών με άυξηση της άλω, τουλάχιστον σε πρώτη φάση.

3 Ανάπτυξη κώδικα MPI

Με το σχεδιασμό που προηγήθηκε μπορούμε να χωρίσουμε τον αλγόριθμο στα εξής βήματα:

1. Ανάγνωση αρχικού πίνακα από αρχείο ή δημιουργία τυχαίου $0-1$ πίνακα και διαμοιρασμός στους επεξεργαστές.
2. Αποστολή οριακών τιμών του τοπικού κομματιού (προς οκτώ κατευθύνσεις).
3. Εκτέλεση τοπικών υπολογισμών.
4. Εκτέλεση οριακών υπολογισμών εφόσον έχουν ληφθεί οι απομακρυσμένες οριακές τιμές.
5. Έλεγχος για στασιμότητα ή για αφανισμό του πληθυσμού.
6. Συνέχεια στην επόμενη γενιά (επιστροφή στο βήμα 2).

Θα περιγράψουμε αναλυτικά τα βασικότερα σημεία της υλοποίησης, δίνοντας τις κυριότερες εντολές MPI που χρησιμοποιήθηκαν. Ολόκληρος ο πηγαίος κώδικας δίνεται στο Παράρτημα Β΄.

Διαμοιρασμός στους επεξεργαστές: Για την ανάπτυξη του κώδικα καταρχάς θεωρούμε μια τοπολογία πλέγματος (grid) διάστασης $r \times s$. Έτσι κάθε επεξεργαστής αποκτά μια θέση στο πλέγμα, η οποία σχετίζεται με την αρχική τάξη (rank) μέσω της απεικόνισης

$$k \longmapsto (i,j) = \Big(k \ \text{ quo } s \ , \ k \mod s\Big) \quad , \quad k = 0, 1, \ldots, rs-1$$

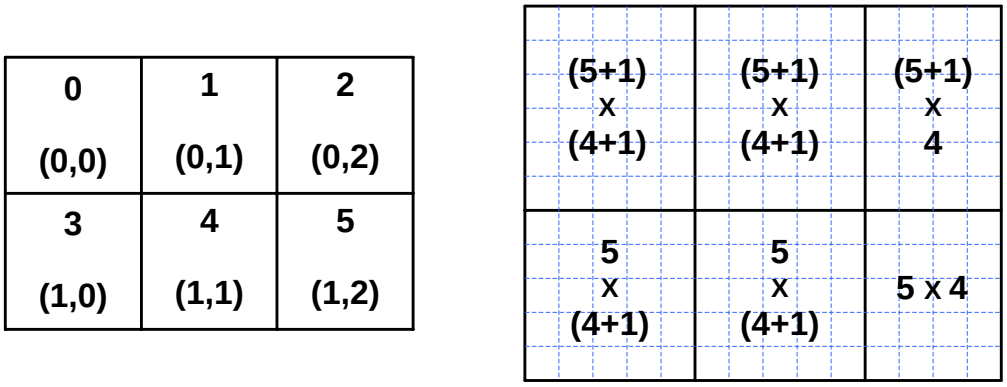
όπου quo και mod το ευκλείδειο πηλίκο και υπόλοιπο αντίστοιχα(Σχήμα 3α). Η υλοποίηση στο περιβάλλον του MPI έγινε με χρήση μιας εικονικής τοπολογίας καρτεσιανών συντεταγμένων (cartesian virtual topology):

```
MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 0, &cart_comm);
MPI_Comm_rank(cart_comm, &my_rank);
MPI_Cart_coords(cart_comm, my_rank, 2, mycoords);
```

όπου το διάνυσμα `periods` είναι μοναδιαίο, δηλαδή το πλέγμα μας είναι αναδιπλούμενο και στις δυο διαστάσεις. Οι καρτεσιανές συντεταγμένες κάθε επεξεργαστή είναι αποθηκευμένες στο διάνυσμα `mycoords`.

Ο διαμερισμός των δεδομένων γίνεται δυναμικά. Κάθε επεξεργαστής λαμβάνει ένα κομμάτι του πίνακα μεγέθους $(N \ \text{ quo } r) \times (M \ \text{ quo } s)$. Αυτό το μέγεθος προσαυξάνεται κατά ένα για τους πρώτους $N \mod r$ επεξεργαστές στην κάθετη διάσταση και τους πρώτους $M \mod s$ επεξεργαστές στην οριζόντια διάσταση. Με τον τρόπο αυτό πετυχαίνουμε ομοιόμορφη απορρόφηση των υπολοίπων στους επεξεργαστές. Ένα παράδειγμα του τρόπου διαμοιρασμού δίνεται στο Σχήμα 3β. Για τις ανάγκες επανασύνθεσης του πίνακα C υπολογίζουμε και τη μετατόπιση (offset) για κάθε διάσταση, η οποία συνδέει τις τοπικές με τις καθολικές (global) συντεταγμένες όπως περιγράφεται στο Σχήμα 4. Η μετατόπιση σε σχέση με τις καθολικές συντεταγμένες του αρχικού πίνακα διορθώνεται ανάλογα σε κάθε διάσταση:

```
nrows = n/dims[0] + ( mycoords[0] < (n%dims[0]) ? 1 : 0 ) + 2;
ncols = m/dims[1] + ( mycoords[1] < (m%dims[1]) ? 1 : 0 ) + 2;
```



(α) (β)

Σχήμα 3: (α) Καρτεσιανές συντεταγμένες επεξεργαστών σε ένα 2×3 πλέγμα επεξεργαστών. Ο k επεξεργαστής έχει συντεταγμένες (i, j) αν $k = 3 \cdot i + j$. (β) Διαμοιρασμός ενός πίνακα διάστασης 11×14 σε ένα πλέγμα επεξεργαστών 2×3. Είναι $11 = 5 \cdot 2 + 1$ άρα η πρώτη γραμμή επεξεργαστών θα λάβει $5 + 1 = 6$ γραμμές του πίνακα. Όμοια $14 = 4 \cdot 3 + 2$ άρα οι δυο πρώτες στήλες επεξεργαστών θα λάβουν $4 + 1 = 5$ στήλες του πίνακα.

```
xoffset = mycoords[0]*(n%dims[0]);
yoffset = mycoords[1]*(m%dims[1]);
if (mycoords[0]>n%dims[0]) xoffset+=n%dims[0]; else xoffset+=mycoords[0];
if (mycoords[1]>m%dims[1]) yoffset+=m%dims[1]; else yoffset+=mycoords[1];
```

Ο υπολογισμός του τοπικού κομματιού και ο διαμοιρασμός γίνεται με χρήση δυο συναρτήσεων, των `getboard` ή `rand_board` για διάβασμα από αρχείο ή δημιουργία τυχαίου πίνακα αντίστοιχα.

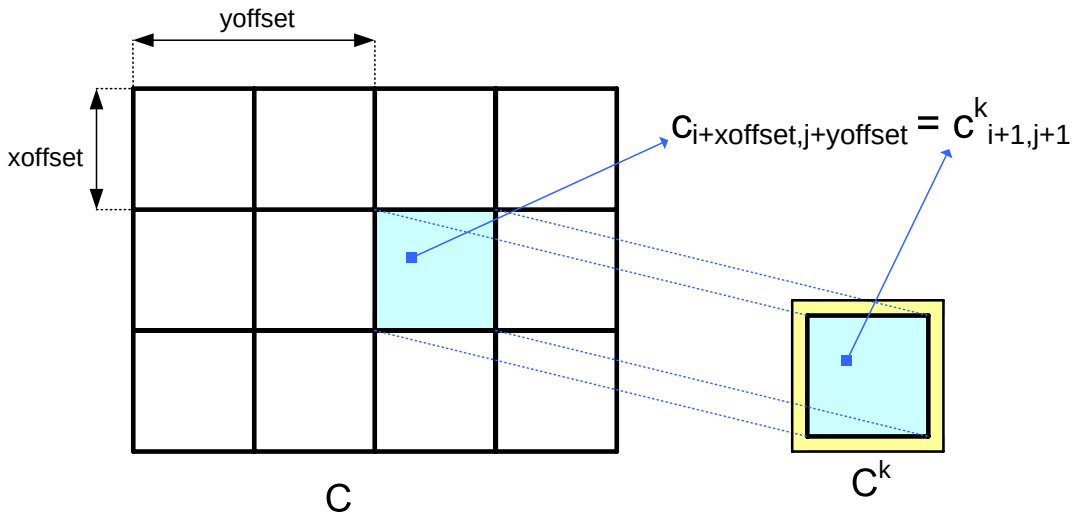
Τοπική Επικοινωνία: Για την επικοινωνία μεταξύ των γειτονικών επεξεργαστών δε χρησιμοποιείται επιπλέον μνήμη· όλες οι αποστολές και λήψεις γίνονται άμεσα στον τοπικό πίνακα κάθε επεξεργαστή. Για να γίνει αυτό εφικτό χρησιμοποιήθηκε ένας επαγόμενος τύπος δεδομένων (derived datatype) για τη μεταφορά στηλών*:

```
MPI_Type_vector(nrows-2, 1, ncols, MPI_INT, &column);
MPI_Type_commit(&column);
```

Κάθε επεξεργαστής ανταλλάζει οκτώ μηνύματα με τους γειτονικούς του επεξεργαστές. Τα γωνιακά στοιχεία μεταφέρονται συνολικά σε τρεις γειτονικούς επεξεργαστές. Για τη λήψη των οριακών δεδομένων από γειτονικούς επεξεργαστές χρησιμοποιείται μια άλως(halo) η οποία περιβάλλει τον τοπικό πίνακα, δηλαδή για τον τοπικό πίνακα έχουν δεσμευτεί δυο επιπλέον γραμμές και στήλες. Η διαδικασία της αποστολής και λήψης των οριακών δεδομένων φαίνεται στο Σχήμα 5. Επειδή η επικοινωνία είναι στατική, αρκεί οι συντεταγμένες των γειτονικών επεξεργαστών να υπολογιστούν μια φορά. Οι γειτονικοί επεξεργαστές υπολογίζονται καλώντας τη ρουτίνα `neighbors`. Για παράδειγμα οι γείτονες `left` και `right` δίνονται ως εξής:

```
tmp_coords[0]=mycoords[0]; tmp_coords[1]=mycoords[1]-1;
MPI_Cart_rank(cart_comm, tmp_coords, left);
tmp_coords[1]+=2;
```

*Κατά τη δέσμευση μνήμης επιλέξαμε τις γραμμές του πίνακα να βρίσκονται σε συνεχόμενες θέσεις μνήμης. Αυτό σημαίνει πως τα στοιχεία μιας στήλης βρίσκονται στη μνήμη ανά nrows−2 θέσεις.



Σχήμα 4: Σχέση τοπικών και καθολικών(global) συντεταγμένων. Η μετατόπιση(offset) υπολογίζεται για κάθε διάσταση εύκολα δοθέντων των καρτεσιανών συντεταγμένων του επεξεργαστή (στο παράδειγμά μας οι συντεταγμένες του επεξεργαστή k είναι $(1, 2)$). Αν ένα στοιχείο του πίνακα C^k έχει καθολικές συντεταγμένες (i_0, j_0) οι τοπικές συντεταγμένες του στον επεξεργαστή k είναι $(i + 1, j + 1)$, όπου $i = i_0 - \text{xoffset}$ και $j = j_0 - \text{yoffset}$. Η πρόσθεση μιας μονάδας στις τοπικές συντεταγμένες γίνεται λόγω της άλω (με κίτρινο χρωματισμό). Όμοια μπορούμε να επιστρέψουμε στις καθολικές συντεταγμένες με τους τύπους $i_0 = i + \text{xoffset}$ και $j_0 = j + \text{yoffset}$.

```
MPI_Cart_rank(cart_comm, tmp_coords, right);
```

Ενδεικτικά δίνουμε τον κώδικα για τα μηνύματα που ανταλλάσσονται οριζόντια:

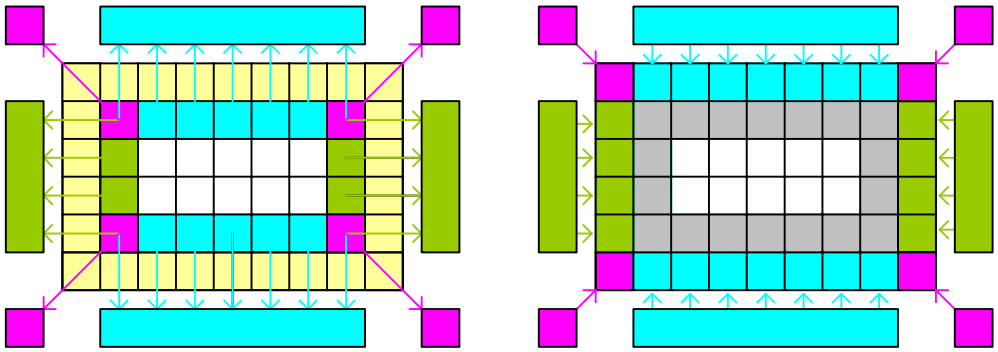
```
MPI_Isend(&(main_board[1][1]),1,column,left,0,cart_comm,&send_req[0]);
MPI_Irecv(&(main_board[1][0]),1,column,left,1,cart_comm,&recv_req[0]);

MPI_Isend(&(main_board[1][ncols-2]),1,column,right,1,cart_comm,&send_req[1]);
MPI_Irecv(&(main_board[1][ncols-1]),1,column,right,0,cart_comm,&recv_req[1]);
```

Η επικοινωνία είναι non-blocking, ώστε να υπάρξει η επιθυμητή επικάλυψη υπολογισμών και επικοινωνίας. Παρατηρήστε την εναλλαγή των ετικετών (tags) στις αποστολές και τις λήψεις: το μήνυμα που αποστέλλεται προς τα αριστερά λαμβάνεται από τα δεξιά του παραλήπτη και αντίστροφα.

Υπολογισμοί: Ο υπολογισμός της νέας τιμής ενός στοιχείου του πίνακα απαιτεί τον υπολογισμό του αθροίσματος των οκτώ γειτονικών τιμών και κατόπιν την εύρεση της νέας τιμής με βάση αυτό το άθροισμα. Για την εύρεση της νέας τιμής αναδιατυπώνουμε τους κανόνες του *Παιχνιδιού* ως εξής:

- Αν το άθροισμα των γειτονικών τιμών είναι ίσο με 3, η νέα τιμή είναι 1.
- Αν το άθροισμα δεν είναι 2 ή 3 τότε η νέα τιμή είναι 0, διαφορετικά δεν υπάρχει αλλαγή.

Ελέγχοντας κατάλληλα την τρέχουσα τιμή του στοιχείου μπορούμε εύκολα να ενσωματώσουμε στους παραπάνω κανόνες έναν έλεγχο για τον αν η νέα τιμή έχει διαφοροποιηθεί από την παλιά, δηλαδή αν $c^k_{ij} = c'^k_{i,j}$ (αυτό απαιτείται έτσι ώστε να τερματίζει το πρόγραμμα αν $C = C'$). Έτσι αποφεύγουμε να διατρέξουμε εκ νέου τους δυο πίνακες για να διεξάγουμε



Σχήμα 5: Επικοινωνία: (α) Αποστολή ακραίων στηλών (β) λήψη γειτονικών δεδομένων στην άλω (halo) του πίνακα.

αυτόν τον έλεγχο. Τελικά ο πυρήνας των υπολογισμών για ένα στοιχείο υλοποιήθηκε ως εξής:

```
crowd_count = main_board[row-1][col-1] + main_board[row-1][col]
+ main_board[row-1][col+1] + main_board[row][col-1] +
main_board[row][col+1] + main_board[row+1][col-1]
+ main_board[row+1][col] + main_board[row+1][col+1];

if(crowd_count==3)
  { if(!main_board[row][col]) changed=1;
    tmp_board[row][col]=1;}
else if(crowd_count!=2 && main_board[row][col]==1)
        { changed=1; tmp_board[row][col]=0;}
else tmp_board[row][col]=main_board[row][col];
```

Η μεταβλητή `changed` αρχικοποιείται στην τιμή 0 και στο τέλος όλων των υπολογισμών υποδηλώνει αν ο πίνακας της νέας γενεάς C_{i+1}^k έχει διαφοροποιηθεί σε σχέση με τον προηγούμενο.

Οι υπολογισμοί χωρίζονται σε δυο μέρη: Οι εσωτερικοί υπολογισμοί μπορούν να εκτελεστούν ακόμη κι αν δεν έχουν ολοκληρωθεί οι λήψεις από τους γειτονικούς επεξεργαστές. Όμως ο υπολογισμός των ακραίων τιμών του πίνακα απαιτεί τις γειτονικές τιμές. Για το λόγο αυτό οι εσωτερικοί υπολογισμοί ξεκινούν ανεξάρτητα από το αν έχουν τελειώσει οι λήψεις. Κατόπιν ακολουθεί η εντολή

```
MPI_Waitall(8, recv_req, MPI_STATUS_IGNORE);
```

πριν ξεκινήσει ο υπολογισμός των ακραίων τιμών. Με τον τρόπο αυτό επικαλύπτουμε τους υπολογισμούς με την επικοινωνία, ώστε να μειώσουμε το χρόνο αδράνειας του επεξεργαστή.

Καθολική Επικοινωνία: Μετά το τέλος των υπολογισμών πρέπει να ελέγξουμε αν ο C_{i+1} ταυτίζεται με τον C_i και αν είναι μηδενικός. Σε αυτές τις περιπτώσεις η εκτέλεση του προγράμματος τερματίζεται. Ο έλεγχος αυτός απαιτεί δυο καθολικές(όλοι με όλους) επικοινωνίες. Για την υλοποίηση αυτών των διαδικασιών χρησιμοποιήθηκαν οι έτοιμες «reduce» εντολές του MPI για την εύρεση ενός μεγίστου κι ενός ελαχίστου, τα οποία εκφράζουν τη διάζευξη και σύζευξη αντίστοιχα των μεταβλητών `changed` ,`empty` του κάθε επεξεργαστή.

```
MPI_Allreduce(&changed, &global_changed, 1, MPI_INT, MPI_MAX, cart_comm);
if (global_changed==0)
```

```
{printf("%d (gen %d): Board Unchanged\n",myrank,gen_count); break;}

empty=1;
for (row=1;row<nrows-1;row++)
    for (col=1;col<ncols-1;col++)
        if (main_board[row][col]){empty=0; break; break;}
MPI_Allreduce(&empty, &global_changed, 1, MPI_INT, MPI_MIN, cart_comm);
if (global_changed==1)
{printf("%d (gen %d): Board Dead\n",myrank,gen_count); break; }
```

Γραφική αναπαράσταση κυττάρων: Για την γραφική αναπαράσταση σε πραγματικό χρόνο χρησιμοποιούμε τη βιβλιοθήκη MPE. Κάθε στοιχείο-κύτταρο αναπαριστάται ως ένα pixel, εκτός αν χρησιμοποιείται η παράμετρος z, με την οποία κάθε κύτταρο μεγεθύνεται ανάλογα με τον παράγοντα μεγέθυνσης που εισάγεται.

```
// Άνοιγμα παραθύρου
MPE_Open_graphics(&graph,cart_comm, dsplayname,-1,-1,n*zoom,m*zoom,0);
// ⋮

// Ανανέωση γενεάς
for(row=1;row<nrows-1;row++)
  for(col=1;col<ncols-1;col++){
          graphx=( xoffset + row )*zoom;
          graphy=( yoffset + col )*zoom;
          MPE_Fill_rectangle(graph,graphx,graphy,zoom,zoom,tmp_board[row][col]);
    }
MPE_Update(graph);
// ⋮

// Κλείσιμο παραθύρου
MPE_Close_graphics(&graph);
```

Για να εμφανιστεί σωστά το παράθυρο και να έχουν δικαιώματα εγγραφής όλοι οι υπολογιστές στην τοπική οθόνη απαιτούνται τα παρακάτω βήματα[†] (έστω για εκτέλεση από τον υπολογιστή aimon):

```
% setenv DISPLAY ‘‘aimon.di.uoa.gr’’:0.0
```

ή αντίστοιχα για κέλυφος bourne:

```
$ DISPLAY=‘‘aimon.di.uoa.gr’’:0.0
$ export DISPLAY
```

Οδηγίες εκτέλεσης: Για να εκτελεστεί το πρόγραμμα χρειάζεται μεταγλώττιση με την εντολή mpicc και έπειτα εκτέλεση με συγκεκριμένες παραμέτρους χρησιμοποιώντας την εντολή mpirun. Για λόγους συντόμευσης, δημιουργήσαμε ένα πρόγραμμα κελύφους(script) με όνομα go που εκτελεί τη μεταγλώτιση (compilation) και εκτελεί το πρόγραμμα. Όλες οι παράμετροι που δίνει ο χρήστης στο script μεταφέρονται στο πρόγραμμα κατά την εκτέλεση του mpirun. Με αυτόν τον τρόπο η μόνη αλλαγή που ενδεχομένως χρειάζεται στο αρχείο go είναι ο αριθμός των επεξεργαστών στην παράμετρο -np:

[†]Επίσης για την ορθή λειτουργία της βιβλιοθήκης MPE λόγω διαφόρων προβλημάτων με τους υπολογιστές στο εργαστήριο SUNs, αναγκαστήκαμε να αφαιρέσουμε τους pegasus και asterios από το αρχείο των διαθέσιμων μηχανών machines.gol.

```
#!/bin/sh
echo "Compiling..."
mpicc -o gol gol.c -lmpe -L/usr/X11R6/lib -L/usr/openwin/lib
                         -lX11 -lm -R/home/appl/gcc-3.1/lib
echo "DONE"
echo "Running..." `date`
mpirun -machinefile machines.gol -np 9 gol $*
echo "DONE" `date`
echo '\007\c'
```

Οι παράμετροι του τελικού προγράμματος είναι όλες προαιρετικές και παρουσιάζονται στον παρακάτω πίνακα:

Προαιρετικές παράμετροι	
`m`	Εκτέλεση του προγράμματος με χρήση της βιβλιοθήκης γραφικών MPE
`x <int> y <int>`	Διαστάσεις του πίνακα του προβλήματος (για τυχαίο πρόβλημα)
`X <int> Y <int>`	Διαστάσεις του πλέγματος των επεξεργαστών
`g <int>`	Ο μέγιστος αριθμός των γενεών εκτέλεσης
`c <int>`	Ο αριθμός των γενεών ανά τις οποίες θα γίνεται έλεγχος τερματισμού
`z <int>`	Παράγοντας μεγέθυνσης (με χρήση MPE μόνο)
`f <file>`	Διάβασμα του προβλήματος από το αρχείο file
`v`	Εκτέλεση με επιπλέον πληροφορίες (verbose)

Για παράδειγμα, το στιγμιότυπο του εξωφύλλου υπολογίζεται ως εξής:

```
$ DISPLAY=``aimon.di.uoa.gr'':0.0
$ export DISPLAY
$ ./go c 10 g 800 f puffer.txt
```

Για έναν τυχαίο πίνακα 200×200, έλεγχο τερματισμού κάθε 10 γενεές, μέγιστο αριθμό γενεών 5000 και χρήση MPE δίνουμε:

```
$ ./go m x 200 y 200 g 5000 z 10
```

4 Ανάλυση απόδοσης

Σε αυτήν την ενότητα θα επιχειρήσουμε μια θεωρητική ανάλυση του παράλληλου αλγορίθμου. Για απλότητα θα θεωρήσουμε ότι ο αλγόριθμος εκτελείται σε ένα τετραγωνικό $\sqrt{P} \times \sqrt{P}$ πλέγμα επεξεργαστών.

Κάνουμε τις εξής συμβάσεις:

- Δε λαμβάνουμε υπόψιν το χρόνο προεπεξεργασίας, πχ το χρόνο που χρειάζεται για να λάβει κάθε επεξεργαστής το τοπικό κομμάτι πίνακα ή το χρόνο υπολογισμού των γειτονικών επεξεργαστών.
- Θεωρούμε ότι δεν υπάρχει επικάλυψη υπολογισμών και επικοινωνίας.
- Αγνοούμε τυχόν χρόνους αδράνειας (idle) των επεξεργαστών.
- Δε λαμβάνουμε υπόψιν την καθολική επικοινωνία, καθώς η συχνότητά της ορίζεται παραμετρικά από το χρήστη. Η επίδρασή της στο χρόνο εκτέλεσης θα ερευνηθεί πειραματικά στην επόμενη ενότητα.

Για τον υπολογισμό μιας γενεάς, ο ακολουθιακός αλγόριθμος για το *Παιχνίδι της Ζωής* τρέχει σε χρόνο

$$T_1 = t_c N M$$

Από τους χρόνους των μετρήσεων εκτιμήσαμε ότι η τιμή του στοιχειώδους υπολογιστικού κόστους για το παράλληλο σύστημα των SUNs είναι $t_c \cong 0.23\mu$sec.

Ο παράλληλος αλγόριθμος δεν εκτελεί κανέναν παραπάνω υπολογισμό, πράγματι

$$T^k_{\text{comp}} = t_c \frac{N}{\sqrt{P}} \cdot \frac{M}{\sqrt{P}} \quad , \quad k = 0, 1, \ldots, P-1$$

δηλαδή το κόστος των υπολογισμών μένει το ίδιο με αυτό του ακολουθιακού αλγορίθμου $T_{\text{comp}} = t_c N M$.

Η επικοινωνία αποτελείται από την αποστολή οκτώ μηνυμάτων από κάθε επεξεργαστή. Δυο από αυτά (οριζόντια) έχουν μέγεθος $L = \frac{N}{\sqrt{P}}$, δυο (κάθετα) έχουν μέγεθος $L = \frac{M}{\sqrt{P}}$ και άλλα τέσσερα (διαγώνια) έχουν μέγεθος μια λέξη. Με χρήση του τύπου $T_{\text{msg}} = t_s + t_w L$ βρίσκουμε το χρόνο επικοινωνίας

$$T_{\text{comm}} = P\Big(8t_s + 2\frac{N}{\sqrt{P}}t_w + 2\frac{M}{\sqrt{P}}t_w + 4t_w\Big) = 8Pt_s + 2P\left(\frac{N}{\sqrt{P}} + \frac{M}{\sqrt{P}} + 2\right)t_w$$

Έτσι ο παράλληλος χρόνος είναι

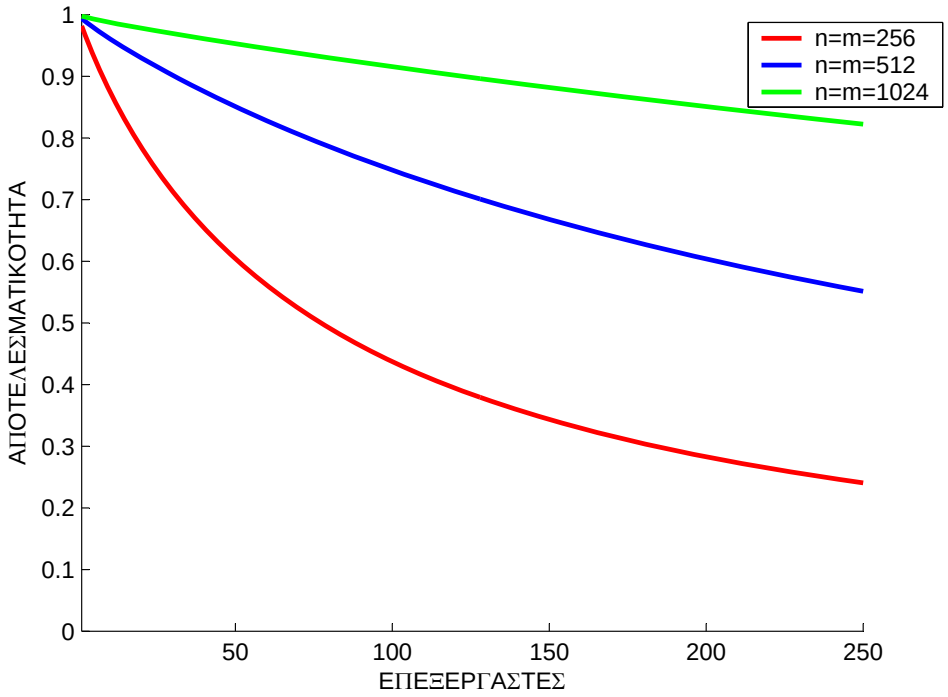
$$T_P = \frac{T_{\text{comp}} + T_{\text{comm}}}{P} = \frac{t_c N M}{P} + 8t_s + 2\left(\frac{N}{\sqrt{P}} + \frac{M}{\sqrt{P}} + 2\right)t_w$$

Η απόδοση (efficiency) δίνεται από τον τύπο

$$E = \frac{T_1}{PT_P} = \frac{t_c N M}{t_c N M + 8Pt_s + 2P\left(\frac{N}{\sqrt{P}} + \frac{M}{\sqrt{P}} + 2\right)t_w}$$

άρα η επιτάχυνση (speedup) είναι

$$S = P \cdot E = \frac{t_c N M}{\frac{t_c N M}{P} + 8t_s + 2\left(\frac{N}{\sqrt{P}} + \frac{M}{\sqrt{P}} + 2\right)t_w}$$



Σχήμα 6: Για αρκετά μεγάλο μέγεθος προβλήματος η συνάρτηση απόδοσης που υπολογίστηκε και σχεδιάζεται εδώ εξασφαλίζει καλή κλιμάκωση(θέσαμε $t_c = 0.2\mu\text{sec},\ t_w = 0.4\mu\text{sec},\ t_s = 100\mu\text{sec}$).

Προχωρούμε στον υπολογισμό της ισοαποτελεσματικότητας(isoefficiency). Διατηρώντας την απόδοση σταθερή, αναζητούμε συνάρτηση του P ώστε καθώς αυξάνεται το P να ισχύει

$$t_c NM \cong E\Big(t_c NM + 8Pt_s + 2\sqrt{P}(N + M + \frac{2}{\sqrt{P}})t_w\Big)$$

Η συνάρτηση $N = M = \sqrt{P}$ ανταποκρίνεται σε αυτήν την απαίτηση, αφού:

$$t_c P \cong E\Big(t_c P + 8Pt_s + 2\sqrt{P}(2\sqrt{P} + \frac{2}{\sqrt{P}})t_w\Big)$$

ή διαιρώντας με P

$$t_c \cong E\Big(t_c + 8t_s + 2(2 + \frac{2}{P})t_w\Big)$$

δηλαδή όταν το P είναι αρκετά μεγαλύτερο του t_w ο όρος $4t_w/P$ γίνεται αμελητέος και έχουμε

$$t_c \cong E\Big(t_c + 8t_s + 4t_w\Big)$$

Συμπεραίνουμε ότι είναι $T_{\text{comp}} = \mathcal{O}(NM) = \mathcal{O}(P)$ άρα η συνάρτηση ισοαποτελεσματικότητας είναι $\mathcal{O}(P)$. Μπορούμε να δούμε και γραφικά ότι ο αλγόριθμος έχει αρκετά καλή κλιμάκωση κάνοντας τη γραφική παράσταση της απόδοσης(Σχήμα 6).

5 Μετρήσεις

Οι μετρήσεις έγιναν στο εργαστήριο SUNs του Τμήματος Πληροφορικής του Πανεπιστημίου Αθηνών, σε ένα cluster εννέα τερματικών[‡] (με λειτουργικό σύστημα Solaris) διασυνδεδεμένων με ethernet[§].

Επειδή το παράλληλο σύστημα δε χρησιμοποιείται αποκλειστικά από εμάς, επαναλάβαμε τις μετρήσεις αρκετές φορές[¶] και σε ώρες που πιθανότατα το σύστημα βρισκόταν σε αδράνεια. Κάθε τιμή χρόνου είναι ο μέσος χρόνος ανά γενεά και ανά επεξεργαστή από μια εκτέλεση 100 γενεών σε τυχαίο πίνακα. Δηλαδή οι χρόνοι υπολογίστηκαν με τον τύπο

$$\text{μέσος χρόνος ανά γενεά} \;=\; \frac{\sum_{k=1}^{P} T_P^k}{gP}$$

όπου g ο αριθμός των γενεών. Κάθε μέτρηση έγινε τρεις φορές και όλοι οι χρόνοι που παρουσιάζονται παρακάτω είναι ο μέσος όρος των μετρήσεων αυτών.

Η καταγραφή των χρόνων γίνεται μέσα στον κώδικα με το γενικό σχήμα:

```
tmp_time= MPI_Wtime();

// Υπολογισμοί

mytime = MPI_Wtime()-tmp_time;
```

Μετρούνται οι εξής χρόνοι:

- Χρόνος τοπικής επικοινωνίας
- Χρόνος υπολογισμών στο εσωτερικό κομμάτι του τοπικού πίνακα
- Χρόνος ακραίων υπολογισμών
- Χρόνος καθολικής επικοινωνίας
- Συνολικός χρόνος για τον υπολογισμό μιας γενεάς

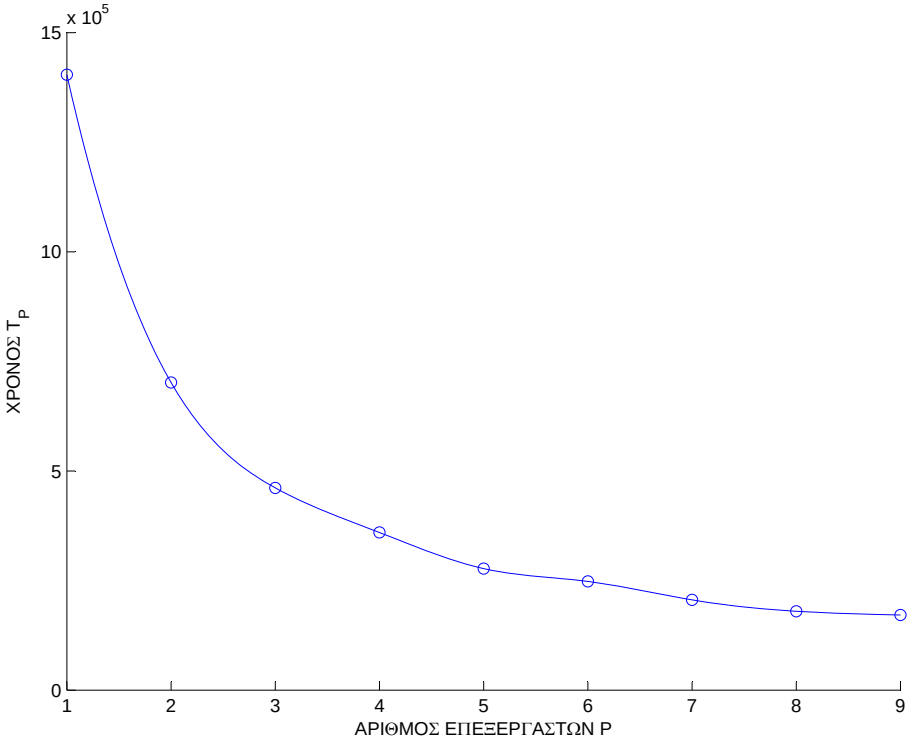
Οι συνολικοί χρόνοι που πήραμε διαφέρουν από 5 ως 15μsec από το άθροισμα των επιμέρους χρόνων. Αυτή η διαφορά οφείλεται σε επιμέρους εργασίες (πχ αύξηση του μετρητή της γενεάς ή ανάθεση τιμών και υπολογισμός των επιμέρους μετρητών) και κρίνεται αμελητέα. Επίσης η επιπλέον δραστηριότητα του συστήματος (που είναι αδύνατο να μηδενιστεί) επιφέρει διαφορές της τάξης των 100μsec στις μετρήσεις. Για το λόγο αυτό επιλέξαμε μεγάλα στιγμιότυπα ώστε οι διακυμάνσεις αυτές να μην αλλοιώσουν τα αποτελέσματα. Οι πίνακες μετρήσεων δίνονται στο Παράρτημα Γ′.

Στο Σχήμα 7 φαίνεται ο συνολικός χρόνος που πήραμε εκτελώντας ένα στιγμιότυπο 1600×1600 σε $P = 1, 2, \ldots, 9$ επεξεργαστές. Η γραφική παράσταση βρίσκεται σε μεγάλη συνάφεια με το νόμο του Amdahl, και μάλιστα για πολύ μικρό καθαρά ακολουθιακό κομμάτι. Πράγματι, η απόδοση μένει σε ιδιαίτερα υψηλά επίπεδα, όπως φαίνεται στο Σχήμα 8. Μια μικρή καμπή στην επιτάχυνση και την απόδοση φαίνεται για $P = 9$. Συμπεραίνουμε ότι το πρόγραμμα έχει εξαιρετικά καλές επιδόσεις, γεγονός που επαληθεύει και η παραπάνω θεωρητική μας ανάλυση.

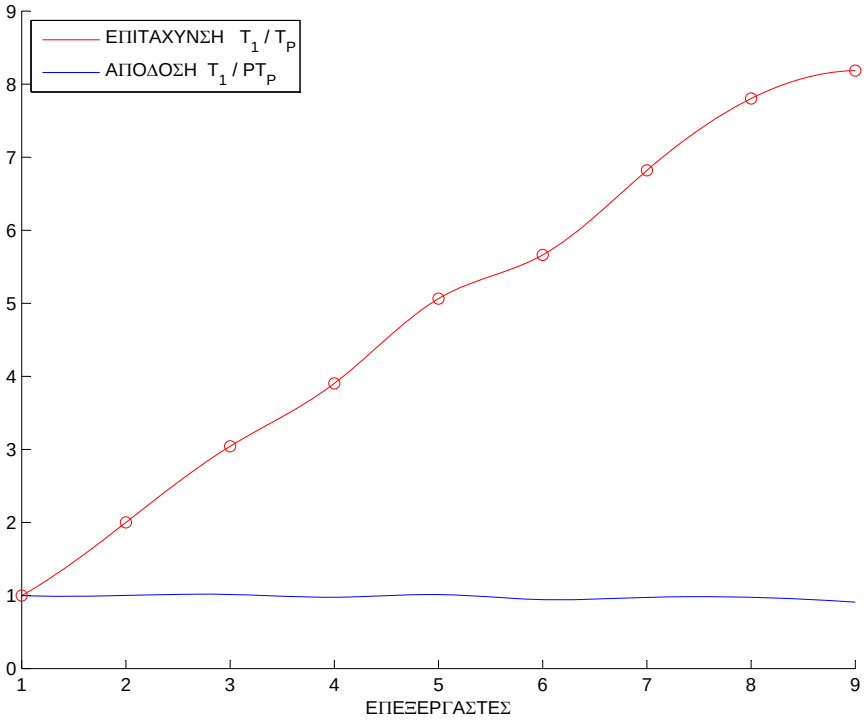
[‡]Τα δικτυακά ονόματα των μηχανών παρατίθενται στο αρχείο machines.gol

[§]Η καθυστέρηση και η μέγιστη διαμεταγωγή του δικτύου εκτιμήθηκαν με το διαγνωστικό πρόγραμμα mpptest (Σχήμα 11)

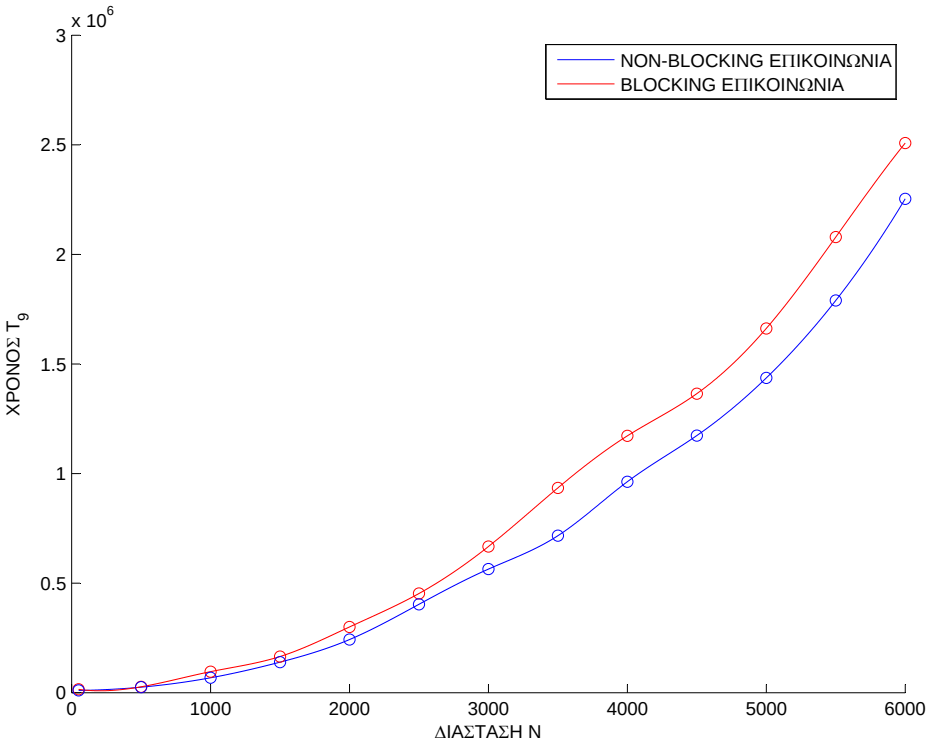
[¶]Όταν παρατηρούνταν μεγάλες αποκλίσεις μεταξύ δυο μετρήσεων οι μετρήσεις επαναλαμβάνονταν, αφού οι αποκλίσεις οφείλονταν σε τυχόν άλλες δραστηριότητες του συστήματος



Σχήμα 7: Παράλληλος χρόνος συναρτήσει του αριθμού των επεξεργαστών (Πίνακας 1.2).



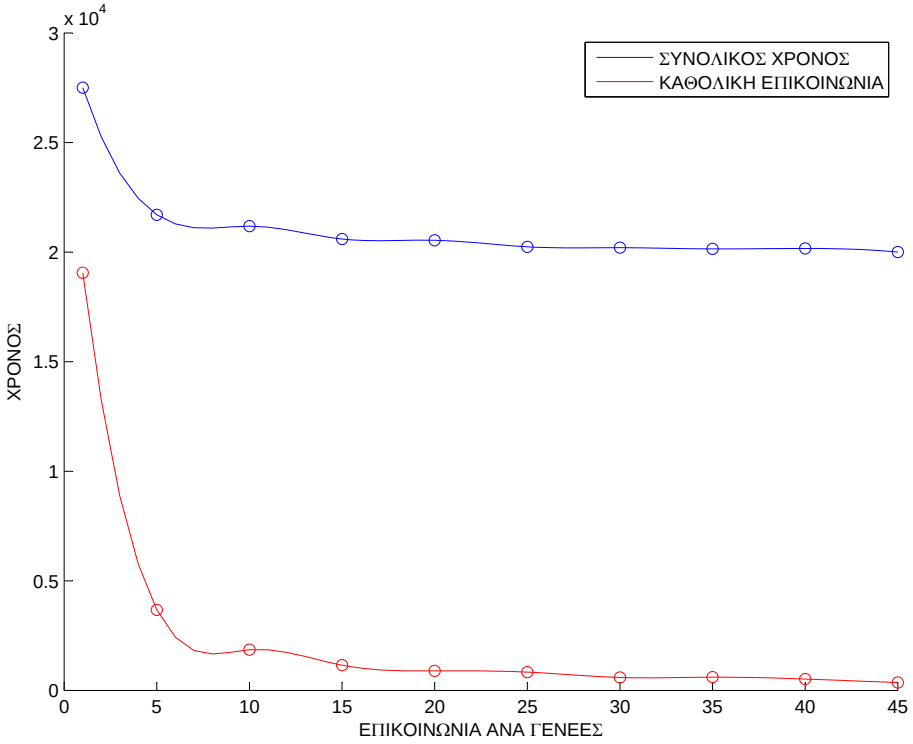
Σχήμα 8: Επιτάχυνση (speedup) και απόδοση (efficiency). Οι ελαφρές ανωμαλίες στην καμπύλη της επιτάχυνσης οφείλονται στο γεγονός ότι για κάποια πλήθη επεξεργαστών ο διαμερισμός δε γίνεται ισομερώς, πχ για $P = 7$ το πλέγμα που χρησιμοποιείται είναι 7×1. (Πίνακας 1.2).



Σχήμα 9: Κέρδος από τη χρήση non-blocking επικοινωνίας (Πίνακες 1.1 και 2.1).

Για να εκτιμήσουμε το κέρδος από την επικάλυψη επικοινωνίας και υπολογισμών, δηλαδή της χρήσης non-blocking επικοινωνίας, τροποποιήσαμε τον κώδικα ώστε να πάρουμε χρόνους όταν χρησιμοποιούμε blocking επικοινωνία. Ο συνολικός χρόνος εκτέλεσης για τις δυο περιπτώσεις φαίνεται στο Σχήμα 9. Παρατηρούμε πως για αρκετά μεγάλα μεγέθη πίνακα η χρήση non-blocking επικοινωνίας έχει ένα αρκετά σημαντικό προβάδισμα (της τάξης των 200msec για $3500 \leq N \leq 6000$).

Για να ερευνήσουμε πειραματικά τις συνέπειες της καθολικής επικοινωνίας στο χρόνο εκτέλεσης εκτελέσαμε ένα στιγμιότυπο μεγέθους 512×512 μεταβάλλοντας την παράμετρο c η οποία ρυθμίζει τη συχνότητα του ελέγχου για τερματισμό (δηλαδή την καθολική επικοινωνία). Ο συνολικός χρόνος εκτέλεσης και ο χρόνος καθολικής επικοινωνίας (που αναλογεί ανά υπολογιζόμενη γενεά) φαίνονται συναρτήσει της παραμέτρου c στο Σχήμα 10. Βλέπουμε ότι η καθολική επικοινωνία έχει δραματικές συνέπειες στο χρόνο εκτέλεσης. Για $c \leq 5$ το μεγαλύτερο ποσοστό του χρόνου εκτέλεσης καταλογίζεται σε αυτή. Η επιρροή όμως αυτή πέφτει κατακόρυφα για μικρότερες συχνότητες (δηλαδή μεγαλύτερες τιμές του c). Αυτή η απότομη συμπεριφορά μπορεί να αιτιολογηθεί από το φαινόμενο της συμφόρησης(contention) του δικτύου.



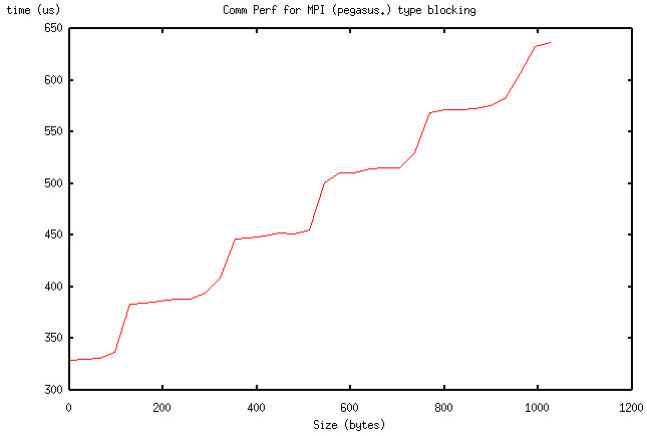
Σχήμα 10: Συνέπειες της καθολικής επικοινωνίας στο χρόνο εκτέλεσης (Πίνακας 2.2).

6 Συμπεράσματα

Είδαμε ότι ο σωστός μεθοδολογικός σχεδιασμός οδήγησε σε ένα καλά κλιμακούμενο παράλληλο πρόγραμμα, αν και οι μετρήσεις έγιναν σε ένα σύστημα διασυνδεδεμένο με ethernet, που έχει αρκετά μεγάλη καθυστέρηση(Σχήμα 11). Η συμφόρηση του δικτύου τόσο από τις διεργασίες του προγράμματός μας, όσο και από τυχόν άλλη δραστηριότητα του συστήματος ήταν εμφανής σε πολλά από τα πειράματα που πραγματοποιήσαμε. Παρόλα αυτά η συμπεριφορά του προγράμματός μας ήταν φανερά ικανοποιητική.

Μια παράμετρο του προβλήματος που δε λάβαμε υπόψιν είναι το είδος των στιγμιοτύπων εισόδου που εισάγονται στον αλγόριθμο. Αν τα στιγμιότυπα είναι αρκετά πυκνά σε κατειλημμένες θέσεις, το πρόγραμμά μας θα δουλέψει αποτελεσματικά. Σε αραιά στιγμιότυπα όμως (πχ Σχήμα 12), ενδέχεται να υπάρχουν επεξεργαστές που δουλεύουν σε κομμάτια του πίνακα γεμάτα μηδενικά. Σε αυτήν την περίπτωση μπορούν να επιστρατευτούν τεχνικές εξισορρόπησης φορτίου (load balancing) προκειμένου να βελτιωθούν οι επιδόσεις. Μια άμεση τέτοια βελτίωση για το πρόγραμμά μας είναι να δημιουργήσουμε κατά την εκτέλεση αριθμό διεργασιών πολλαπλάσιο του αριθμού των επεξεργαστών και να διαμοιράσουμε τον πίνακα με κυκλική ανάθεση σε αυτές. Με τον τρόπο αυτό μειώνεται η πιθανότητα να υπάρχουν επεξεργαστές με μηδενικό κομμάτι.

Μια τελευταία παρατήρηση είναι ότι παρά την αρκετά εύκολη παραλληλοποίηση του αλγορίθμου σε λογικό επίπεδο, η υλοποίηση (καθώς και η εκσφαλμάτωση) στο MPI εμπεριείχε διάφορες λεπτομέρειες που απαίτησαν αρκετή προγραμματιστική δουλειά. Ενδεχομένως μια υλοποίηση με χρήση του προτύπου OpenMP να αποβεί το ίδιο αποτελεσματική και με σημαντικά μικρότερο προγραμματιστικό κόστος.



Σχήμα 11: Αποτελέσματα από την εκτέλεση του διαγνωστικού προγράμματος mpptest στον κόμβο Pegasus.di.uoa.gr. Η καθυστέρηση(latency) του δικτύου είναι $t_s \cong 330\mu$sec.

Παράρτημα Α΄. Παραδείγματα εκτέλεσης

Για τον έλεγχο της ορθότητας της υλοποίησης πειραματιστήκαμε με μερικά ενδιαφέροντα patterns. Η είσοδός τους στο πρόγραμμα γίνεται με τη μορφή αρχείου κειμένου ASCII. Στην πρώτη γραμμή του αρχείου πρέπει να υπάρχουν οι διαστάσεις του πίνακα (χωρισμένες με κενό) και στις επόμενες οι συντεταγμένες των «ζωντανών» οργανισμών. Για παράδειγμα η είσοδος για το στιγμιότυπο του εξωφύλλου δίνεται στο αρχείο(puffer.txt):

$$\begin{array}{l} 150\ 450 \\ 70\ 20 \\ 71\ 21 \\ \quad \vdots \\ 82\ 23 \end{array}$$

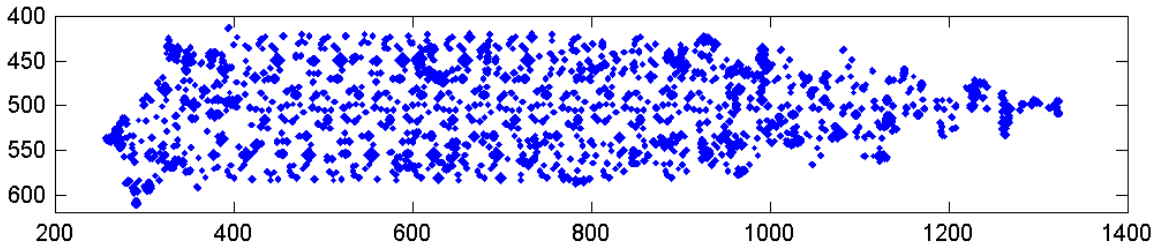
δηλαδή $N = 150$, $M = 450$ και $c_{70,20} = c_{71,21} = \cdots = c_{82,23} = 1$. Για τη δημιουργία του εξωφύλλου υπολογίστηκαν 400 γενεές αυτού του σχεδίου. Ένας μεγαλύτερος πίνακας με το ίδιο σχέδιο δίνεται στο puffer2000.txt. Στο Σχήμα 12 δίνουμε γραφικά τη γενεά 2000 που πήραμε με είσοδο αυτό το αρχείο.

Ο σχηματισμός του αρχείου spaceship.txt «μετακινείται» προς τα πάνω περνώντας από τέσσερις ενδιάμεσες καταστάσεις, δηλαδή με μια περίοδο πέντε γενεών το σχέδιο(Σχήμα 14) εμφανίζεται και πάλι στον πίνακα, μετατοπισμένο προς τα πάνω. Λόγω της περιοδικότητας του πλέγματος το «διαστημόπλοιο» εξαφανίζεται προοδευτικά όταν «φτάσει» στο πάνω μέρος του πίνακα (δηλαδή στην πρώτη γραμμή) και εμφανίζεται στην τελευταία γραμμή.

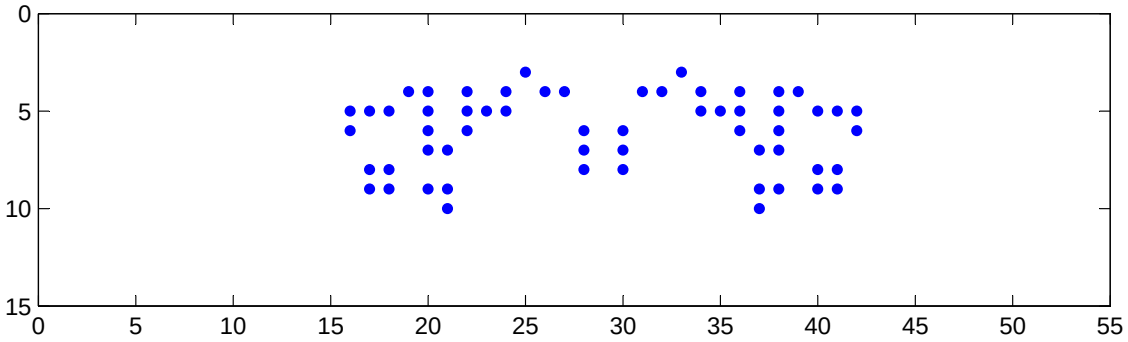
Ένα σχέδιο που εξελίσσεται μετατοπίζοντας τον εαυτό του παράλληλα με τη διαγώνιο του πίνακα φαίνεται στο Σχήμα 14. Λόγω της περιοδικότητας του πλέγματος η κίνηση αυτή επαναλαμβάνεται περιοδικά επ΄ άπειρον.

Τέλος στο αρχείο cross.txt δίνεται ένας πίνακας διάστασης 100×100 με μη μηδενικά κελιά τα $\{\ c_{ij} \mid i = 50 \text{ ή } j = 50\}$. Η πολύ δραστήρια εξέλιξή του φαίνεται στο Σχήμα 15.

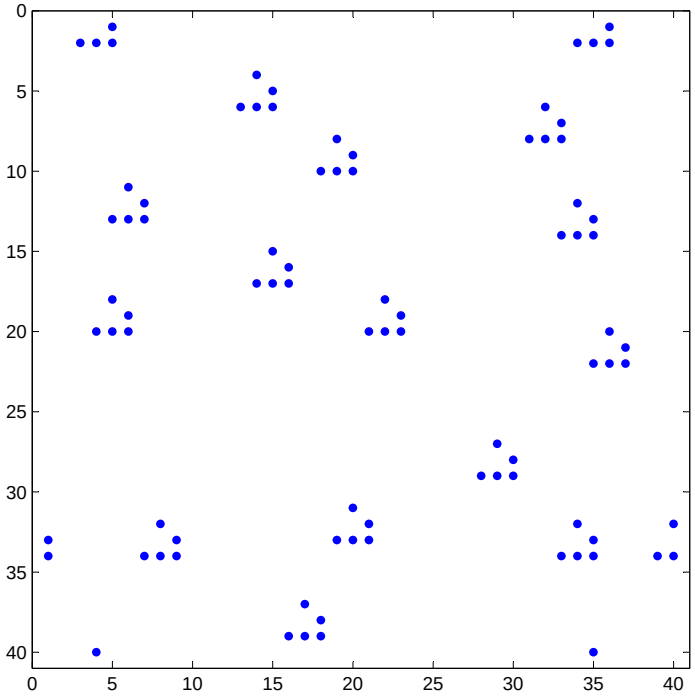
Ενδιαφέρον παρουσιάζει και η εκτέλεση με τυχαίο πίνακα· κάποιες φορές ο πίνακας σταθεροποιείται μετά από λίγο, ενώ άλλες φορές η εξέλιξη συνεχίζει με περιοδικά ή ακανόνιστα φαινόμενα. Στο διαδίκτυο υπάρχει μια πληθώρα σχεδίων τα οποία επιδεικνύουν πολλά ακόμη εντυπωσιακά φαινόμενα.



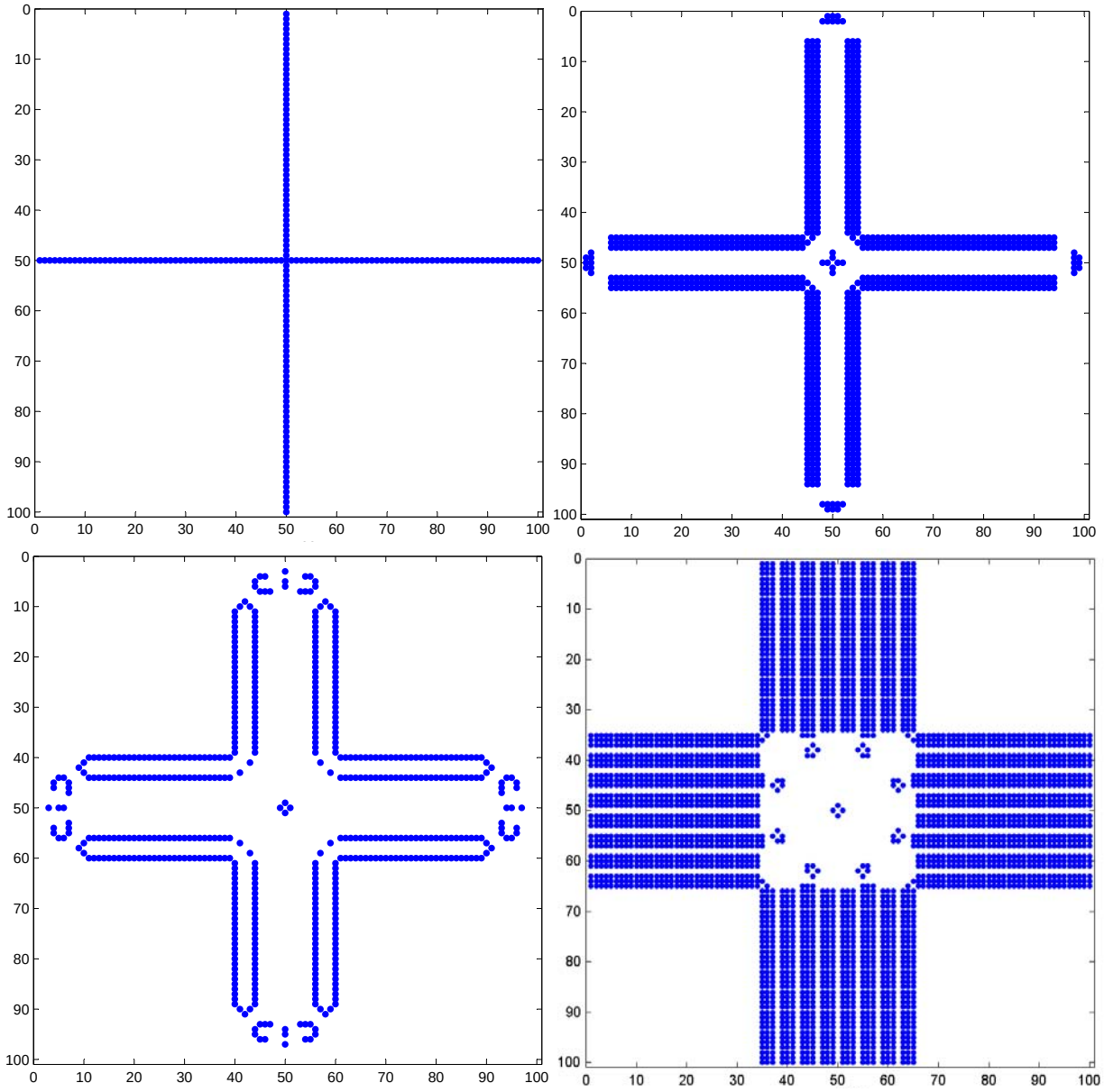
Σχήμα 12: Ο «καπνός τρένου» μετά από 2000 γενεές. Σε αυτόν τον πίνακα διάστασης 1000×1800 οι 7400 κατειλημμένες θέσεις προέκυψαν από μόλις 22 στην αρχική γενεά και είναι συγκεντρωμένες στη ζώνη $400 \leq i \leq 650$. Ο υπόλοιπος πίνακας παραμένει κενός με το πέρασμα των γενεών(puffer2000.txt).



Σχήμα 13: Ένα «διαστημόπλοιο» που κινείται επ' άπειρον παράλληλα με τον κάθετο άξονα. Κάθε πέμπτη γενεά ο πίνακας περιέχει μόνο ένα αντίγραφο του αρχικού σχεδίου (το οποίο φαίνεται στην εικόνα) μετατοπισμένο προς τα πάνω (spaceship.txt).



Σχήμα 14: Αυτό το «σμήνος πουλιών» κινείται διαγώνια στον πίνακα επ' άπειρον (sminos.txt).



Σχήμα 15: Σταυρός. Πάνω: Αρχικό pattern(αριστερά), πέμπτη γενεά(δεξιά). Κάτω: δέκατη γενεά(αριστερά), 15η γενεά(δεξιά). Ένα πραγματικά εκρηκτικό σχέδιο(cross.txt).

Παράρτημα Β'. Κώδικας C/MPI

```
/* To Paixnidi tis Zwhs toy John Conlway
 * Parallili Ylopoihsh
 *
 * A8anasios Polymeneas & Aggelos Mantzaflaris
 * 2007
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include "mpi.h"
#include "mpe.h"
#include "mpe_graphics.h"

/*Neighbor Function*/
void neighbors(MPI_Comm cart_comm, int*mycoords,int*up, int*down, int*left,
               int*right, int*uleft, int*uright, int*dleft, int*dright);
/*Board delivery Functions*/
int **getboard(int*dims,int*mycoords,int*n,int*m,int*xoffset,int*yoffset,
               int*ncols,int*nrows,char * input_filename);
int **rand_board(int*dims,int*mycoords,int n,int m,int*xoffset,int*yoffset,int*ncols,int*nrows);

int main(int argc, char *argv[])
{

/* Command Line Parsing Variables (default values) */
char flag;
int verbose=0;
int rand=0;
int rand_n=50,rand_m=50;
int grid_x=0,grid_y=0;
int checkpoint=1;
int zoom=1;
char *input_filename;

/* Program Variables */

MPE_XGraph graph;
char * displayname = 0;
int graphx,graphy;
int enable_mpe=0;

int i,row,col;
int **main_board, **tmp_board , **token_board;      /*Board Pointers*/
int n, m;                                           /*Global Dimensions*/
int nrows,ncols, xoffset, yoffset;     /*Local dimensions and offsets*/

int gen_count, generations=100;                     /*Generation Number*/
int changed, global_changed, empty;          /*Termination Conditions*/

int processors, myrank;          /*Number of processors & Process rank*/
int up, down, left, right, uleft, uright, dleft, dright; /*Neighbors*/
int crowd_count;                                    /*Neighbor Sum*/
double tmp_time,avg,gen,mytime[5]={0,0,0,0,0};              /*Timers*/

MPI_Comm cart_comm;               /*Virtual Grid topology communicator*/
int mycoords[2],periods[2],dims[2]={0,0};           /*Grid parameters*/

MPI_Datatype column;                          /*Column derived datatype*/
MPI_Request recv_req[8], send_req[8];    /*Send-recv acknowledgements*/

/* INITIATE MPI */
MPI_Init(&argc, &argv);

/* PARSE COMMAND LINE */
input_filename=malloc(128*sizeof(char));
srand((unsigned)time(NULL));


for(i=1;i<argc;i++){
        flag=(char)(argv[i][0]);
        switch(flag){
                case 'z' :
                        zoom=atoi(argv[i+1]);
                        i++;
                        break;
                case 'x' :
                        rand_n=atoi(argv[i+1]);
                        i++;
                        break;
                case 'y' :
                        rand_m=atoi(argv[i+1]);
                        i++;
                        break;
                case 'X' :
                        grid_x=atoi(argv[i+1]);
                        i++;
                        break;
                case 'Y' :
                        grid_y=atoi(argv[i+1]);
                        i++;
                        break;
                case 'g' :
                        generations=atoi(argv[i+1]);
```

```
                        i++;
                        break;
                case 'm' :
                        enable_mpe=1;
                        break;
                case 'c' :
                        checkpoint=atoi(argv[i+1]);
                        i++;
                        break;
                case 'v' :
                        verbose=1;
                        break;
                case 'f' :
                        strcpy(input_filename,argv[i+1]);
                        i++;
                        break;
                case 'h' :
printf("Proper use: go [v] [x XDIM y YDIM] [X XDIM Y YDIM] [g GENERATIONS] [m] [f FILE] [h]\n");
printf("Flags:\n");
printf("v : Verbose.\n");
printf("f : Load game data from file FILE. If -f is not set, random values are used.\n");
printf("x : Set the gameboard's X dimension for random games.\n");
printf("y : Set the gameboard's Y dimension for random games.\n");
printf("g : Set the maximum number of generations.\n");
printf("h : Display this help.\n");
exit(1);
                        break;
                default :
                        break;
        }
}
/* ENTER RANDOM MODE BY DEFAULT */
if( strlen(input_filename)==0 ) {
        rand=1;
        n=rand_n;
        m=rand_m;
}
/* END OF COMMAND LINE PARSE */


/* GET NUMBER OF PROCS */
MPI_Comm_size(MPI_COMM_WORLD,&processors);

if(grid_x*grid_y==processors){
        dims[0]=grid_x;
        dims[1]=grid_y;
}
else{
        dims[0]=sqrt(processors);
        dims[1]=sqrt(processors);
}

/* CARTESIAN TOPOLOGY */
periods[0]=1;
periods[1]=1;
        MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 0, &cart_comm);
        MPI_Comm_rank(cart_comm, &myrank);
        MPI_Cart_coords(cart_comm, myrank, 2, mycoords);

if(myrank==0 && verbose==1){
        printf("Grid Size : %dx%d\n",dims[0],dims[1]);
        printf("Board Size : %dx%d\n",n,m);
        printf("Max Generations : %d\n",generations);
        if(strlen(input_filename)>0){
                printf("Input method : FILE %s\n",input_filename);
        }
        else{
                printf("Input method : Random\n");
        }
}

/* FIND NEIGHBORS*/
neighbors(cart_comm, mycoords, &up, &down, &left, &right, &uleft, &uright, &dleft, &dright);

/* GET MAIN BOARD*/
if (rand==1)    main_board= rand_board(dims, mycoords, n, m, &xoffset, &yoffset, &ncols, &nrows);
else main_board= getboard(dims, mycoords, &n, &m, &xoffset, &yoffset, &ncols, &nrows,input_filename);

/*ALLOCATE MEMORY FOR tmp_board*/
tmp_board = malloc(ncols * sizeof(int*));
tmp_board[0] = malloc(nrows*ncols*sizeof(int));
for( i = 1; i<nrows ; i++){ tmp_board[i]=tmp_board[0]+i*ncols;}

/* DERIVED DATA TYPE FOR TRANSFERING COLUMNS*/
MPI_Type_vector(nrows-2, 1, ncols, MPI_INT, &column);
MPI_Type_commit(&column);

/* MPE GRAPHICS */
if(enable_mpe) MPE_Open_graphics( &graph, cart_comm, displayname, -1, -1, n*zoom, m*zoom, 0 );

/* START GENERATION */
MPI_Barrier(cart_comm);
mytime[4]=MPI_Wtime();

for(gen_count=1;gen_count<=generations;gen_count++){
/*-------------------------------------Communication---------------------------------*/
tmp_time= MPI_Wtime();
        /*LEFT*/
MPI_Isend( &(main_board[1][1]), 1, column, left, 0, cart_comm, &send_req[0]);
```

```
MPI_Irecv( &(main_board[1][0]), 1, column, left, 1, cart_comm, &recv_req[0]);
        /*RIGHT*/
MPI_Isend( &(main_board[1][ncols-2]), 1, column, right, 1, cart_comm, &send_req[1]);
MPI_Irecv( &(main_board[1][ncols-1]), 1, column, right, 0, cart_comm, &recv_req[1]);
        /*UP*/
MPI_Isend( &(main_board[1][1]), ncols-2, MPI_INT, up, 2, cart_comm, &send_req[2]);
MPI_Irecv( &(main_board[0][1]), ncols-2, MPI_INT, up, 3, cart_comm, &recv_req[2]);
        /*DOWN*/
MPI_Isend( &(main_board[nrows-2][1]), ncols-2, MPI_INT, down, 3, cart_comm, &send_req[3]);
MPI_Irecv( &(main_board[nrows-1][1]), ncols-2, MPI_INT, down, 2, cart_comm, &recv_req[3]);
        /*UP LEFT*/
MPI_Isend(&(main_board[1][1]), 1, MPI_INT, uleft, 4, cart_comm, &send_req[4]);
MPI_Irecv(&(main_board[0][0]), 1, MPI_INT, uleft, 5, cart_comm, &recv_req[4]);
        /*DOWN RIGHT*/
MPI_Isend(&(main_board[nrows-2][ncols-2]), 1, MPI_INT, dright, 5, cart_comm, &send_req[5]);
MPI_Irecv(&(main_board[nrows-1][ncols-1]), 1, MPI_INT, dright, 4, cart_comm, &recv_req[5]);
    /*UP RIGHT*/
MPI_Isend(&(main_board[1][ncols-2]), 1, MPI_INT, uright, 6, cart_comm, &send_req[6]);
MPI_Irecv(&(main_board[0][ncols-1]), 1, MPI_INT, uright, 7, cart_comm, &recv_req[6]);
    /*DOWN LEFT*/
MPI_Isend(&(main_board[nrows-2][1]), 1, MPI_INT, dleft, 7, cart_comm, &send_req[7]);
MPI_Irecv(&(main_board[nrows-1][0]), 1, MPI_INT, dleft, 6, cart_comm, &recv_req[7]);
/* MPI_Waitall(8, recv_req, MPI_STATUS_IGNORE);*/
mytime[0]+= MPI_Wtime()-tmp_time;
/*------------------------------------ Computation------------------------------------*/
        changed=0;
tmp_time= MPI_Wtime();
        /* COMPUTE LOCAL WHILE WAITING*/
     for(row=2;row<nrows-2;row++){
                for(col=2;col<ncols-2;col++){
        crowd_count = main_board[row-1][col-1] + main_board[row-1][col] +
      main_board[row-1][col+1] + main_board[row][col-1] + main_board[row][col+1] +
      main_board[row+1][col-1] + main_board[row+1][col] + main_board[row+1][col+1];

                        if(crowd_count==3)
                          {  if(!main_board[row][col]) changed=1;
                    tmp_board[row][col]=1;}
                        else if(crowd_count!=2 && main_board[row][col]==1)
                    { changed=1; tmp_board[row][col]=0;}
                        else tmp_board[row][col]=main_board[row][col];
                    }
    }
mytime[1]+= MPI_Wtime()-tmp_time;

        /* WAIT TO RECIEVE BORDER INFORMATION**/
tmp_time= MPI_Wtime();
        MPI_Waitall(8, recv_req, MPI_STATUS_IGNORE);

        /* NON-LOCAL COMPUTATIONS**/
   for(row=1;row<nrows-1;row++){
                /*col=1*/
                crowd_count = main_board[row-1][0] + main_board[row-1][1] + main_board[row-1][2]
                                + main_board[row][0] + main_board[row][2] + main_board[row+1][0]
                                + main_board[row+1][1] + main_board[row+1][2];

                        if(crowd_count==3)
                          {  if(!main_board[row][1]) changed=1;
                    tmp_board[row][1]=1;}
                        else if(crowd_count!=2 && main_board[row][1]==1)
                    { changed=1; tmp_board[row][1]=0;}
                        else tmp_board[row][1]=main_board[row][1];
                /*col=ncols-2*/
                crowd_count = main_board[row-1][ncols-3] + main_board[row-1][ncols-2] +
       main_board[row-1][ncols-1] + main_board[row][ncols-3] + main_board[row][ncols-1] +
       main_board[row+1][ncols-3] + main_board[row+1][ncols-2] + main_board[row+1][ncols-1];

                        if(crowd_count==3)
                          {  if(!main_board[row][ncols-2]) changed=1;
                    tmp_board[row][ncols-2]=1;}
                        else if(crowd_count!=2 && main_board[row][ncols-2]==1)
                    { changed=1; tmp_board[row][ncols-2]=0;}
                        else tmp_board[row][ncols-2]=main_board[row][ncols-2];
   }
   for(col=1;col<ncols-1;col++){
                /*row=1*/
                crowd_count = main_board[0][col-1] + main_board[0][col] + main_board[0][col+1]
                                + main_board[1][col-1] + main_board[1][col+1] + main_board[2][col-1]
                                + main_board[2][col] + main_board[2][col+1];

                        if(crowd_count==3)
                          {  if(!main_board[1][col]) changed=1;
                    tmp_board[1][col]=1;}
                        else if(crowd_count!=2 && main_board[1][col]==1)
                    { changed=1; tmp_board[1][col]=0;}
                        else tmp_board[1][col]=main_board[1][col];
                /*row=nrows-2*/
                crowd_count = main_board[nrows-3][col-1] + main_board[nrows-3][col] +
       main_board[nrows-3][col+1] + main_board[nrows-2][col-1] + main_board[nrows-2][col+1] +
        main_board[nrows-1][col-1] + main_board[nrows-1][col] + main_board[nrows-1][col+1];

                        if(crowd_count==3)
                          {  if(!main_board[nrows-2][col]) changed=1;
                    tmp_board[nrows-2][col]=1;}
                        else if(crowd_count!=2 && main_board[nrows-2][col]==1)
                    { changed=1; tmp_board[nrows-2][col]=0;}
                        else tmp_board[nrows-2][col]=main_board[nrows-2][col];
    }

mytime[2]+= MPI_Wtime()-tmp_time;
```

```
/* USE MPE */
if(enable_mpe) {
for(row=1;row<nrows-1;row++){
for(col=1;col<ncols-1;col++){
        graphx=( xoffset + row )*zoom;
        graphy=( yoffset + col )*zoom;
        MPE_Fill_rectangle(graph, graphx, graphy, zoom,zoom, tmp_board[row][col]);
}
}
 MPE_Update(graph);
}

    /* SWAP BOARDS*/
        token_board=tmp_board;
        tmp_board=main_board;
        main_board=token_board;

if(gen_count % checkpoint==0){

empty=1;
    for (row=1;row<nrows-1;row++)
        for (col=1;col<ncols-1;col++)
            if (main_board[row][col]){empty=0; break; break;}

tmp_time= MPI_Wtime();
        /* CHECK FOR CHANGES*/
MPI_Allreduce(&changed, &global_changed, 1, MPI_INT, MPI_MAX, cart_comm);
if (global_changed==0 && verbose==1) { printf("%d (gen %d): Board Unchanged\n",myrank,gen_count); break;}
    /*CHECK FOR ZERO BOARD*/
    MPI_Allreduce(&empty, &global_changed, 1, MPI_INT, MPI_MIN, cart_comm);
if (global_changed==1 && verbose==1) { printf("%d (gen %d): Board Dead\n",myrank,gen_count); break; }
mytime[3]+= MPI_Wtime()-tmp_time;
}

}/* END GENERATION */
mytime[4]= MPI_Wtime()-mytime[4];

if(enable_mpe) MPE_Close_graphics(&graph);

/* RUNNING TIMES */
if(myrank==0 && verbose==1)
printf("Generations run: %d/%d\n",((gen_count-1==generations)?gen_count-1:gen_count),generations);
if(myrank==0 && verbose==1)     printf("Timers:\n");
        avg= dims[0]*dims[1];
        gen=((gen_count-1==generations)?gen_count-1:gen_count);

MPI_Reduce(&mytime[1], &tmp_time, 1, MPI_DOUBLE, MPI_SUM, 0, cart_comm);
if(myrank==0 && verbose==1)     printf("Local: avg %f, total %f\n",tmp_time/(avg*gen),tmp_time/avg);

MPI_Reduce(&mytime[0], &tmp_time, 1, MPI_DOUBLE, MPI_SUM, 0, cart_comm);
if(myrank==0 && verbose==1)     printf("Comm: avg %f, total %f\n",tmp_time/(avg*gen),tmp_time/avg);

MPI_Reduce(&mytime[2], &tmp_time, 1, MPI_DOUBLE, MPI_SUM, 0, cart_comm);
if(myrank==0 && verbose==1)     printf("Non-local: avg %f, total %f\n",tmp_time/(avg*gen),tmp_time/avg);

MPI_Reduce(&mytime[3], &tmp_time, 1, MPI_DOUBLE, MPI_SUM, 0, cart_comm);
if(myrank==0 && verbose==1) printf("Global comm: avg %f, total %f\n",tmp_time/(avg*gen),tmp_time/avg);
if(myrank==0 && verbose==1) printf("Total time: %f, (%f per generation)\n",mytime[4],mytime[4]/gen);

MPI_Barrier(cart_comm);
MPI_Comm_free(&cart_comm);
MPI_Finalize();
return 0;
}

/*======= Neighbor Function =======*/
void neighbors(MPI_Comm cart_comm, int*mycoords,int*up, int*down,int*left, int*right,
                int*uleft, int*uright, int*dleft, int*dright)
{
     int tmp_coords[2];

     /*HORIZONTAL*/
     tmp_coords[0]=mycoords[0]; tmp_coords[1]=mycoords[1]-1;
     MPI_Cart_rank(cart_comm, tmp_coords, left);
     tmp_coords[1]+=2;
     MPI_Cart_rank(cart_comm, tmp_coords, right);
     /*VERTICAL*/
     tmp_coords[0]=mycoords[0]-1; tmp_coords[1]=mycoords[1];
     MPI_Cart_rank(cart_comm, tmp_coords, up);
     tmp_coords[0]+=2;
     MPI_Cart_rank(cart_comm, tmp_coords, down);
     /*1st DIAGONAL*/
     tmp_coords[0]=mycoords[0]-1; tmp_coords[1]=mycoords[1]-1;
     MPI_Cart_rank(cart_comm, tmp_coords, dleft);
     tmp_coords[0]+=2; tmp_coords[1]+=2;
     MPI_Cart_rank(cart_comm, tmp_coords, uright);
     /*2nd DIAGONAL*/
     tmp_coords[0]=mycoords[0]-1; tmp_coords[1]=mycoords[1]+1;
     MPI_Cart_rank(cart_comm, tmp_coords, uleft);
     tmp_coords[0]+=2; tmp_coords[1]-=2;
     MPI_Cart_rank(cart_comm, tmp_coords, dright);
}

/*======= Board delivery Functions =======*/
/*###### Load board from file ######*/
int **getboard(int*dims,int*mycoords,int*n,int*m,int*xoffset,int*yoffset,
                int*ncols,int*nrows,char * input_filename)
{
     int **main_board;
```

```
      int row,col,i;
      FILE * board; /*Input file*/
board = fopen(input_filename,"r");

/*READ DIMENSIONS */
if (!fscanf(board,"%d",n)) printf("Error Reading dimension N\n");
if (!fscanf(board,"%d",m)) printf("Error Reading dimension M\n");

/* LOCAL ROWS AND COLS */
*nrows    = (*n)/dims[0] + ( mycoords[0] < ((*n)%dims[0]) ? 1 : 0 ) + 2;
*ncols    = (*m)/dims[1] + ( mycoords[1] < ((*m)%dims[1]) ? 1 : 0 ) + 2;

/* OFFSETS */
*xoffset = mycoords[0]*(*n/dims[0]);
*yoffset = mycoords[1]*(*m/dims[1]);
if (mycoords[0]>(*n)%dims[0]) xoffset+=(*n)%dims[0]; else *xoffset+=mycoords[0];
if (mycoords[1]>(*m)%dims[1]) yoffset+=(*m)%dims[1]; else *yoffset+=mycoords[1];

/*DYNAMICALLY ALLOCATED MULTIDIM ARRAY*/
main_board = malloc(*ncols * sizeof(int*));
main_board[0] = malloc((*nrows)*(*ncols)*sizeof(int));
for( i = 1; i<(*nrows) ; i++){ main_board[i]=main_board[0]+i*(*ncols);}

for (row=0;row<*nrows;row++)for(col=0;col<*ncols;col++) main_board[row][col]=0;

/*READ THE DATA */
if(board){
   while(fscanf(board,"%d",&row)>0){
        fscanf(board,"%d",&col);
            if(row>*xoffset && col>*yoffset && (row<=*xoffset+*nrows-2) && (col<=*yoffset+*ncols-2)){
                main_board[row-*xoffset][col-*yoffset]=1; }
   }
}
else{
        printf("Error loading game file\n");
        exit(-1);
}
fclose(board);

return main_board;
}


/###### Make Random board ######*/
int **rand_board(int*dims,int*mycoords,int n,int m,int*xoffset,int*yoffset,int*ncols,int*nrows)
{
      int **main_board;
      int row,col,i;

/* LOCAL ROWS AND COLS */
*nrows    = n/dims[0] + ( mycoords[0] < ( n%dims[0]) ? 1 : 0 ) + 2;
*ncols    = m/dims[1] + ( mycoords[1] < ( m%dims[1]) ? 1 : 0 ) + 2;

/* OFFSETS */
*xoffset = mycoords[0]*( n/dims[0]);
*yoffset = mycoords[1]*( m/dims[1]);
if (mycoords[0]> n%dims[0]) xoffset+= n%dims[0]; else *xoffset+=mycoords[0];
if (mycoords[1]> m%dims[1]) yoffset+= m%dims[1]; else *yoffset+=mycoords[1];

/*DYNAMICALLY ALLOCATED MULTIDIM ARRAY*/
main_board = malloc(*ncols * sizeof(int*));
main_board[0] = malloc((*nrows)*(*ncols)*sizeof(int));
for( i = 1; i<(*nrows) ; i++){ main_board[i]=main_board[0]+i*(*ncols);}

/*RANDOM FILL*/
for (row=0;row<(*nrows);row++)
    for(col=0;col<(*ncols);col++)
       main_board[row][col]=rand()%2;

return main_board;
}
```

Παράρτημα Γ΄. Πίνακες μετρήσεων

Όλοι οι χρόνοι δίνονται σε εκατομμυριοστά του δευτερολέπτου.

1. Non-blocking επικοινωνία

Πίνακας 1.1
Εκτέλεση σε πλέγμα 3×3 επεξεργαστών

N	Τοπική Επ.	Τοπικοί Υπολ.	Ακραίοι Υπολ.	Καθολική Επ.	Συνολικός Χρόνος
50	481	57	1825	8014	10384
500	554	5392	713	18537	25203
1000	625	21649	964	44798	68043
1500	1034	49033	1341	87999	139414
2000	1080	87979	1666	151696	242430
2500	1688	144686	3032	253897	403313
3000	1535	206509	2415	353450	563917
3500	1769	269354	2564	442570	716266
4000	1907	354157	3076	603498	962647
4500	2143	444038	3344	723703	1173238
5000	2256	547672	3726	882473	1436136
5500	2798	668072	4402	1114231	1789513
6000	2629	811452	4451	1434905	2253446

Πίνακας 1.2
Εκτέλεση στιγμιοτύπου 1600×1600

P	Πλέγμα	Τοπική Επ.	Τοπικοί Υπολ.	Ακραίοι Υπολ.	Καθολική Επ.	Χρόνος
1	1×1	1381	1392110	5303	175	1403918
2	2×1	2506	418500	1799	1306	701905
3	3×1	1810	214870	1135	2833	461390
4	2×2	1790	126912	791	2461	359673
5	5×1	1164	100256	766	4488	277232
6	3×2	1247	80116	534	4040	247972
7	7×1	910	63612	644	3534	205872
8	4×2	910	52923	421	6672	179922
9	3×3	930	56286	465	7474	171522

2. Blocking επικοινωνία

Πίνακας 2.1
Εκτέλεση σε πλέγμα 3×3 επεξεργαστών

N	Τοπική Επ.	Τοπικοί Υπολ.	Ακραίοι Υπολ.	Καθολική Επ.	Συνολικός Χρόνος
50	2761	47	32	12516	15363
500	2627	5435	271	18610	26950
1000	4194	24296	566	66991	96056
1500	4190	51529	967	108331	165026
2000	4746	94840	1397	198959	299951
2500	7045	148997	1618	294853	452521
3000	6495	220084	2001	438256	666845
3500	9265	305116	2567	617537	934494
4000	7824	398558	2992	762572	1171955
4500	8870	487500	3799	884312	1364491
5000	10006	622617	3856	1025505	1661993
5500	11162	761387	4715	1201472	2078745
6000	13211	965543	5555	1523496	2507814

Πίνακας 2.2
Εκτέλεση στιγμιοτύπου 512×512 με έλεγχο τερματισμού ανά c γενεές

c	Τοπική Επ.	Τοπικοί Υπολ.	Ακραίοι Υπολ.	Καθολική Επ.	Συνολικός Χρόνος
1	2500	5668	269	19059	27505
5	12055	5670	299	3672	21701
10	13344	5682	295	1856	21182
15	13500	5665	278	1151	20598
20	13670	5680	293	891	20539
25	13453	5674	273	835	20240
30	13683	5646	274	592	20200
35	13418	5739	278	604	20144
40	13489	5879	275	520	20168
45	13645	5702	281	429	20062

Βιβλιογραφία

[1] Marc Snir, William Gropp, Steve Otto, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, David Walker, Jack J. Dongarra, *MPI: The Complete Reference*, Scientific and Engineering Computation, The MIT Press, second edition, 1998

[2] Michael J. Quinn, *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill, 2003

[3] Neil MacDonald, Elspeth Minty, Joel Malard, Tim Harding, Simon Brown, Mario Antonioletti, *Writing Message Passing Parallel Programs with MPI Course notes*, Edinburgh Parallel Computing Center

[4] Elspeth Minty, Robert Davey, Alan Simpson, David Henty, *Decomposing the Potentially Parallel Course notes*, Edinburgh Parallel Computing Center

[5] Lawrence A. Crowl, *How to Measure, Present, and Compare Parallel Performance*, IEEE Parallel & Distributed Technology: Systems & Technology archive, Vol. 2, Issue 1, 1994, pages 9-25

[6] Mark Baker(Ed.), *Cluster Computing White Paper*, Final Release, Portsmouth, 2000

[7] Ian Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison-Weslay, 1995

[8] Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar, *Introduction to Parallel Computing*, Addison Wesley, second edition, 2003

[9] D. Sima, T.J. Fountain, P. Kacsuk, *Advanced Computer Architectures*, Addison-Wesley, 1997

[10] Karl Sigmund, *Games of Life: Explorations in Ecology, Evolution, and Behavior*, Penguin, 1995

[11] Martin Gardner, *Mathematical Games*, Scientific American 223(4), October 1970, pages 120-123