ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Παράλληλα Συστήματα - Εργασία 1

Χριστόφορος-Μάριος Μαμαλούκας
Εθνικό Καποδιστριακό Πανεπιστήμιο Αθηνών - Τμήμα Πληροφορικής και Τηλεπικοινωνιών

ΠΕΡΙΛΗΨΗ

Σε αυτήν την εργασία εφαρμόζουμε διάφορους αλγορίθμους σε σειριακή και παράλληλη μορφή και τους συγκρίνουμε μεταξύ τους.

Όλες οι ασκήσεις εκτελέστηκαν πάνω στο σύστημα linux01.di.uoa.gr του Πανεπιστημίου Αθηνών, το οποίο διαθέτει:

- τετραπύρηνο επεξεργαστή Intel Core i5-6500 @ 3.20GHz,
- 4GB μνήμη RAM,
- λειτουργικό σύστημα Ubuntu 20.04.5 LTS,
- έκδοση πυρήνα 5.4.0-216-generic,
- έκδοση μεταγλωττιστή GCC 9.4.0

Κάθε άσκηση εκτελέστηκε 4 φορές με την μέση τιμή του χρόνου εκτέλεσης να υπολογίζεται από το ίδιο το εκτελέσιμο.

Κάθε άσκηση εκτελείται από το εκτελέσιμο ως εξής:

```
./exe -e <exercise_number> [FLAGS...]
```

όπου το `<exercise_number>` είναι ο αριθμός της άσκησης που θέλουμε να εκτελέσουμε και τα `FLAGS` είναι οι παράμετροι (οι οποίες έχουν το πρόθημα `-f`) που θέλουμε να δώσουμε στην εκάστοτε άσκηση.

ΑΣΚΗΣΗ 1

Σε αυτήν την άσκηση εφαρμόζουμε αλγόριθμο βασισμένο στην μέθοδο Monte Carlo για την εκτίμηση του π.

Για να το πετύχουμε αυτό, παράγουμε τυχαία σημεία εντός ενός τετραγώνου πλευράς 1.

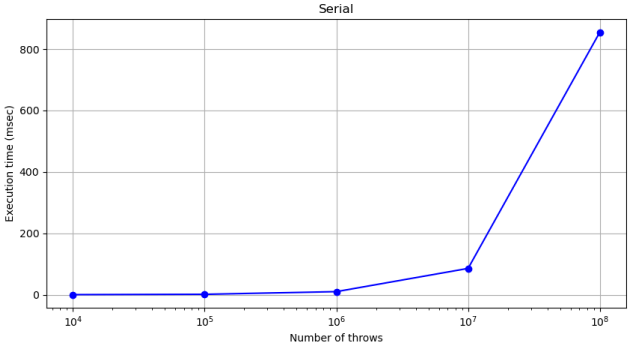
Για όσα σημεία $q = (x, y)$ ισχύει ότι $x^2 + y^2 \leq 1$, τα σημεία αυτά βρίσκονται εντός του κύκλου ακτίνας 1. Τότε, αν A είναι το πλήθος όλων των ρίψεων και B το πλήθος των ρίψεων που βρίσκονται εντός του κύκλου, τότε η εκτίμηση του π είναι:

$$\pi \approx 4\frac{B}{A}$$

Ο πηγαίος κώδικας της άσκησης βρίσκεται στο αρχείο `src/pi_calc.c`.

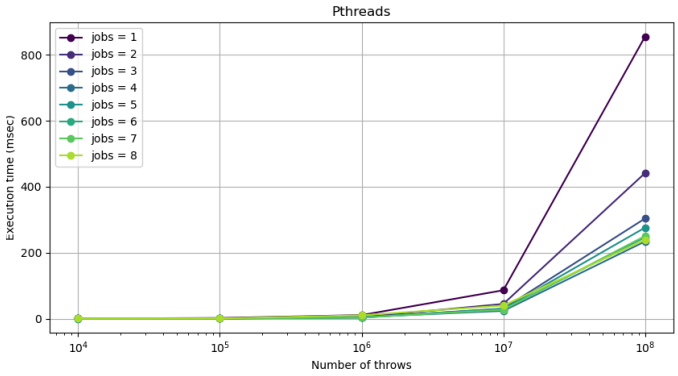
Η άσκηση δέχεται ως ορίσματα το πλήθος των συνολικών ρίψεων (`-fn=<number>`), ποια υλοποίηση θα εκτελεστεί (`-fs` για την σειριακή, `-fp` για την παράλληλη που χρησιμοποιεί pthreads και `-fomp` για την παράλληλη που χρησιμοποιεί openmp) και το πλήθος των νημάτων που θα χρησιμοποιηθούν (`-fj=<number>`). Η έξοδος είναι το εκτιμώμενο π και ο χρόνος εκτέλεσης του αλγορίθμου, όπως και η υλοποίηση που χρησιμοποιήθηκε αλλά και το πλήθος των ρίψεων, η οποία προεραιτικά μπορεί να αποθηκευτεί σε αρχείο με το όρισμα `-ff=<filename>`.

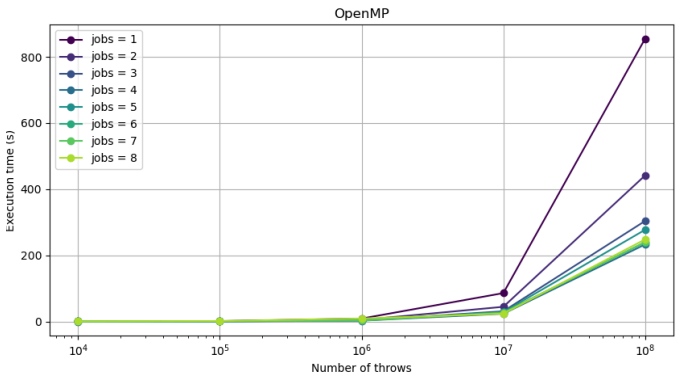
Η σειριακή υλοποίηση έχει την εξής επίδοση:



Όπως αναμένουμε, επειδή ο αλγόριθμος χρησιμοποιεί έναν βρόχο για να υπολογίσει το π, η επίδοσή του είναι γραμμική ($O(n)$) (στο διάγραμμα ο αριθμός των ρίψεων είναι σε λογαριθμική κλίμακα).

Τώρα, ας δούμε τις παράλληλες υλοποιήσεις. Για να τις παραλληλοποιήσουμε, απλά χωρίσαμε τις ρίψεις ανά νήμα, με το κάθε νήμα να υπολογίζει το πλήθος $\frac{n}{j}$ ρίψεων, όπου j είναι το πλήθος των νημάτων. Στο τέλος, τα νήματα συγχρονίζονται για να υπολογίσουν το τελικό αποτέλεσμα. Οι υλοποιήσεις έχουν τις εξής επιδόσεις:





Από άποψη των επιδόσεων μεταξύ τους, δεν παρατηρείται μεγάλη διαφορά (κάτι που ίσως να ερμηνευθεί και ως θετικό για την OpenMP, αφού παρά το γεγονός ότι είναι πολύ υψηλότερου επιπέδου από την pthreads, οι επιδόσεις της είναι συγκρίσιμες).

Ωστόσο, αυτό που παρατηρούμε και στις δύο υλοποιήσεις είναι το εξής: όταν ανεβούμε πάνω από τα 4 νήματα, οι επιδόσεις *δεν* βελτιώνονται· αντ᾽αυτού, μένουν (περίπου) στάσιμες. Αυτό εύκολα ερμηνεύεται από το γεγονός ότι το σύστημα το οποίο χρησιμοποιήθηκε για την εκτέλεση της άσκησης έχει μόνον 4 φυσικούς πυρήνες. Ως εκ τούτου, (λόγω της αρχής του περιστερώνα) τουλάχιστον 2 νήματα θα πρέπει να εκτελεστούν σε έναν πυρήνα, το οποίο σημαίνει ότι θα έχουμε σειριοποίηση της εκτέλεσης για νήματα περισσότερα από 4.

Να σημειωθεί ότι στην παράλληλη υλοποίηση χρησιμοποιήθηκε συγχρονισμός, αλλά μόνον για την ενημέρωση του τελικού μετρητή

των ρίψεων (δηλαδή όταν ένα νήμα είχε ήδη πραγματοποιήσει όλες τις ρίψεις ανατεθιμένες σε αυτό). Αυτό βοήθησε στην μείωση πιθανών καθυστερήσεων λόγω αναμονής για ενημέρωση μιας κοινόχρηστης μεταβλητής.

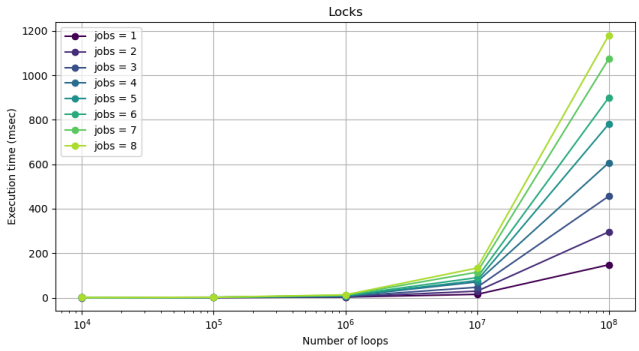
ΑΣΚΗΣΗ 2

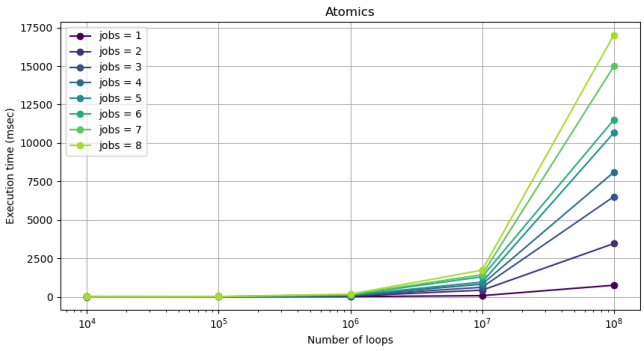
Σε αυτήν την άσκηση, βάζουμε νήματα να ανανεώσουν μία κοινόχρηστη μεταβλητή (η οποία αρχικοποιείται στο 0), για έναν καθορισμένο αριθμό επαναλήψεων. Αναμένουμε ότι στο τέλος της εκτέλεσης, η μεταβλητή αυτή θα έχει την τιμή $j \times n$, όπου j είναι το πλήθος των νημάτων και n το πλήθος των επαναλήψεων. Για να το επιτύχουμε αυτό, χρησιμοποιούμε είτε κλειδώματα (mutexes) είτε ατομικές εντολές (atomics) και για να επιβεβαιώσουμε την λειτουργικότητα του συγχρονισμού, τοποθετούμε μία εντολή `assert` στο τέλος του προγράμματος, η οποία ελέγχει αυτήν την συνθήκη.

Ο πηγαίος κώδικας της άσκησης βρίσκεται στο αρχείο `src/shared_var.c`.

Η άσκηση δέχεται ως ορίσματα το πλήθος των επαναλήψεων (`-fn=<number>`), ποια υλοποίηση θα εκτελεστεί (`-fl` για την χρήση mutexes και `-fa` για την χρήση ατομικών εντολών) και το πλήθος των νημάτων που θα χρησιμοποιηθούν (`-fj=<number>`). Η έξοδος είναι ο χρόνος εκτέλεσης του αλγορίθμου, όπως και η υλοποίηση που χρησιμοποιήθηκε αλλά και το πλήθος των επαναλήψεων, η οποία προεραιτικά μπορεί να αποθηκευτεί σε αρχείο με το όρισμα `-ff=<filename>`.

Ας συγκρίνουμε τις επιδόσεις των δύο υλοποιήσεων:





Δύο είναι οι παρατηρήσεις μας:

1. Η υλοποίηση με atomics είναι κατά πολύ αργότερη από την υλοποίηση με mutexes.
2. Και οι δύο υλοποιήσεις είναι βραδύτερες όσο αυξάνεται το πλήθος των νημάτων.

Ο λόγος που οι ατομικές εντολές εδώ βγήκαν βραδύτερες των mutexes είναι, μάλλον, λόγω του γεγονότος ότι έχουμε πολύ μεγάλο ανταγωνισμό για την κοινόχρηστη μεταβλητή, ειδικά όσο αυξάνεται ο αριθμός των νημάτων (περισσότερα νήματα θέλουν να ενημερώσουν πολλές φορές την ίδια μεταβλητή). Γενικά, τα atomics χρησιμεύουν σε καταστάσεις όπου ο ανταγωνισμός είναι σχετικά μικρός, αφού αυτά πρακτικά είναι ένα είδος busy waiting.

ΑΣΚΗΣΗ 3

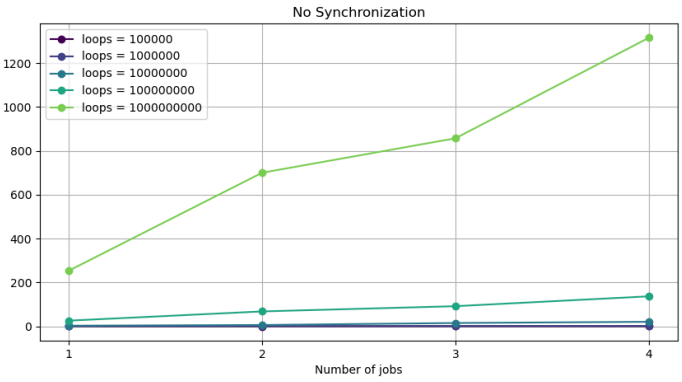
Σε αυτήν την άσκηση βάζουμε νήματα να ανανεώνουν, πλέον, όχι την ίδια κοινόχρηστη μεταβλητή, αλλά το καθένα την δική του, η οποία βρίσκεται σε ένα κοινόχρηστο πίνακα (το μέγεθος του οποίου είναι ίσο με το πλήθος των νημάτων). Το ποια θέση του πίνακα θα ενημερώνει κάθε νήμα καθορίζεται από το ID του νήματος. Εδώ, δοκιμάζουμε τόσο την χρήση συγχρονισμού (είτε με mutex είτε με atomic) όσο και την απουσία του. Όπως και στην προηγούμενη άσκηση, ελέγχουμε την ορθότητα του προγράμματος με την εντολή `assert`, η οποία ελέγχει ότι η τιμή κάθε θέσης στον πίνακα είναι ίση με το πλήθος των επαναλήψεων.

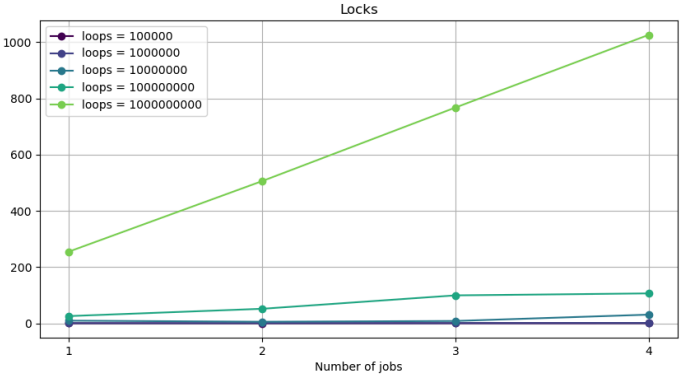
Ο πηγαίος κώδικας της άσκησης βρίσκεται στο αρχείο `src/false_sharing.c`.

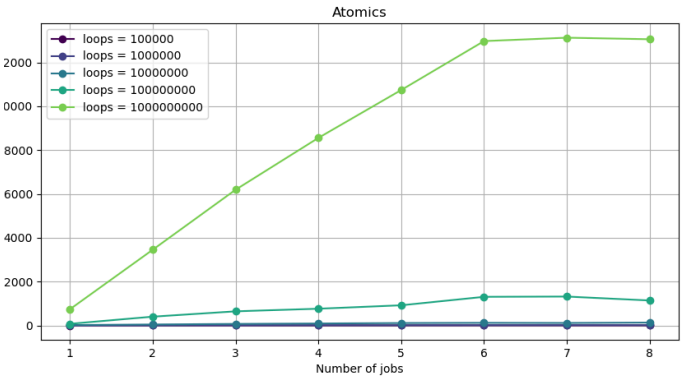
Η άσκηση δέχεται ακριβώς τα ίδια ορίσματα με την προηγούμενη, με την προσθήκη ενός επιπλέον, του `-fns` για την απουσία συγχρονισμού. Η έξοδος είναι παρόμοια με την προηγούμενη άσκηση, με την διαφορά ότι εδώ έχουμε και την επιλογή να μην χρησιμοποιήσουμε καθόλου συγχρονισμό. Στην περίπτωση όμως που η απουσία του συγχρονισμού δεν περάσει την `assert`, το πρόγραμμα εξάγει αρχείο με σφάλμα, όπου καταγράφονται οι παράμετροι οι οποίες χρησιμοποιήθηκαν στην αποτυχημένη προσπάθεια.

Κατά την εκτέλεση της άσκησης, οι υλοποιήσεις με κλειδώματα και χωρίς συγχρονισμό απέτυχαν τον έλεγχο για νήματα περισσότερα από 4 (αριθμός μάλιστα ίσος με τον αριθμό των πυρήνων του επεξεργαστή μας).

Οι συλλεχθείσες μετρήσεις για την υλοποίηση με κλειδώματα είναι οι εξής:







Προφανώς, η υλοποίηση χωρίς συγχρονισμό είναι η ταχύτερη, αφού δεν έχουμε overhead από χρήση μεθόδων συγχρονισμού.

Μια ενδιαφέρουσα παρατήρηση είναι ότι η υλοποίηση με atomics σταθεροποιείται από άποψη χρόνου, όταν το πλήθος των νημάτων αρχίζει να αυξάνει κατά πολύ.

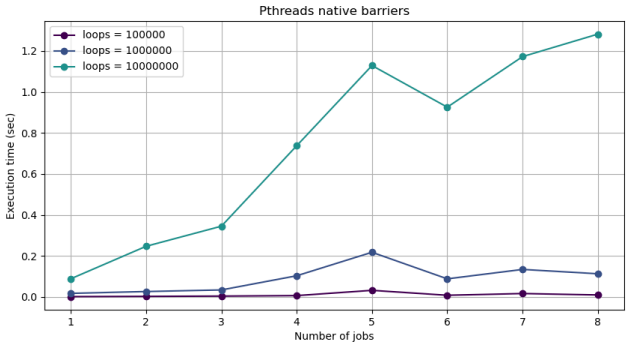
ΑΣΚΗΣΗ 4

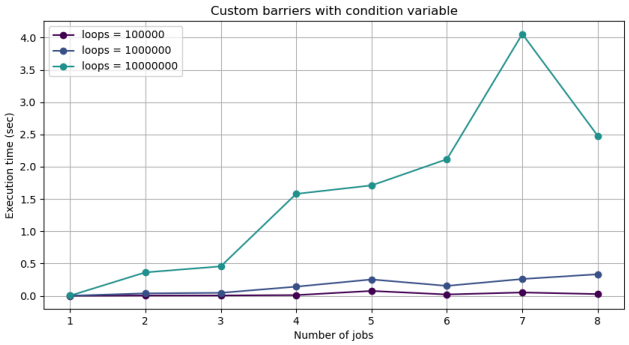
Στην άσκηση αυτήν υλοποιούμε φράγματα (barriers) με τη χρήση της αναμονής σε εγρήγορση (busy waiting) και με την χρήση μεταβλητών συνθήκης (condition variables) και τις συγκρίνουμε με τα εγγενή φράγματα του pthreads. Για την σύγκριση, κάθε νήμα τρέχει έναν ορισμένο αριθμό επαναλήψεων, όπου αναμένει το φράγμα.

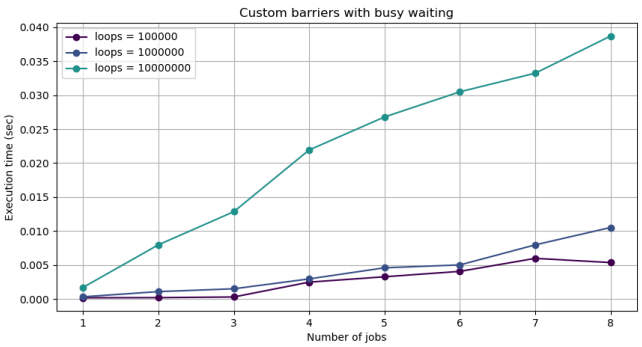
Ο πηγαίος κώδικας της άσκησης βρίσκεται στο αρχείο `src/barriers.c`.

Η άσκηση δέχεται ως ορίσματα το πλήθος των επαναλήψεων (`-fn=<number>`), ποια υλοποίηση θα εκτελεστεί (`-fb` για την χρήση busy waiting, `-fc` για την χρήση condition variables και `-fp` για την χρήση του εγγενούς φράγματος του pthreads) και το πλήθος των νημάτων που θα χρησιμοποιηθούν (`-fj=<number>`). Η έξοδος είναι ο χρόνος εκτέλεσης του αλγορίθμου, όπως και η υλοποίηση που χρησιμοποιήθηκε αλλά και το πλήθος των επαναλήψεων, η οποία προεραιτικά μπορεί να αποθηκευτεί σε αρχείο με το όρισμα `-ff=<filename>`.

Λαμβάνουμε τα εξής αποτελέσματα:







Μία κοινή τάση και στις τρεις υλοποιήσεις είναι η εξής:

- Όσα περισσότερα νήματα χρησιμοποιούμε, τόσο πιο αργά εκτελούνται οι υλοποιήσεις. Αυτό είναι αναμενόμενο, καθώς το φράγμα απαιτεί από όλα τα νήματα να περιμένουν το ένα το άλλο, οπότε όσο περισσότερα νήματα έχουμε, τόσο περισσότερο χρόνο θα χρειαστούν για να περιμένουν. Προφανώς, όταν έχουμε περισσότερα νήματα από φυσικούς πυρήνες, ο χρόνος αναμονής αυξάνεται και στις τρεις περιπτώσεις, αφού το σύστημα δεν μπορεί να εκτελέσει τα νήματα ταυτόχρονα.
- Όσες περισσότερες επαναλήψεις έχουμε, τόσο πιο αργά εκτελούνται οι υλοποιήσεις, το οποίο είναι επίσης αναμενόμενο, γιατί η αναμονή για το φράγμα είναι μία χρονοβόρα διαδικασία.

Το τρίτο που παρατηρούμε είναι ότι η ταχύτερη υλοποίηση είναι αυτή που χρησιμοποιεί αναμονή σε εγρήγορση ενώ η αργότερη αυτή που χρησιμοποιεί μεταβλητές συνθήκης.

ΑΣΚΗΣΗ 5

Στην άσκηση αυτήν ασχολούμαστε με την βελτιστοποίηση ενός αλγορίθμου πολλαπλασιασμού πίνακα με άνυσμα για την περίπτωση που ο πίνακας είναι τριγωνικός. Για να το επιτύχουμε αυτό, αξιοποιήσαμε τον όρο `schedule()` της OpenMP, ο οποίος μας επιτρέπει να αναδιανείμουμε νέο έργο στα νήματα, όταν αυτά τελειώσουν το τρέχον έργο τους. Αυτό είναι χρήσιμο, δεδομένου ότι κάθε νήμα εκτελεί τον εξής αλγόριθμο:

```
for (uint32_t j = i; j < n; j++)
{
  sum += A[i][j] * x[j];
}
```

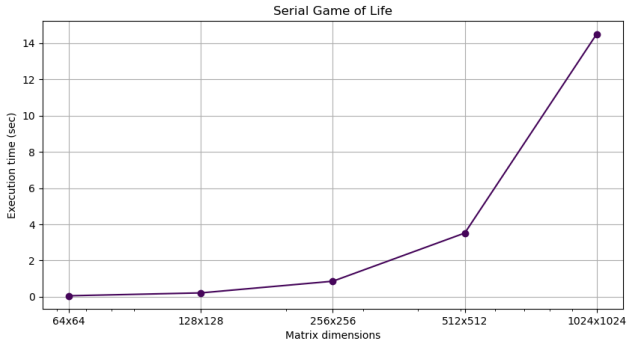
όπου A είναι ο πίνακας και x το διάνυσμα, i η ταυτότητα του νήματος και η τρέχουσα γραμμή που διαχειρίζεται και n το πλήθος των γραμμών του πίνακα (που είναι ίσος με το πλήθος των στηλών, αφού οι τριγωνικοί πίνακες είναι τετράγωνοι). Για νήματα τα οποία αναλαμβάνουν τα «κάτω» μέρη του πίνακα, βλέπουμε η δουλειά τους θα τελειώσει σχετικά γρήγορα. Ως εκ τούτου, θα αποτελούσε μεγάλη σπατάλη να μην τα αξιοποιούσαμε για να αναλάβουν και κάποιους άλλους υπολογισμούς.

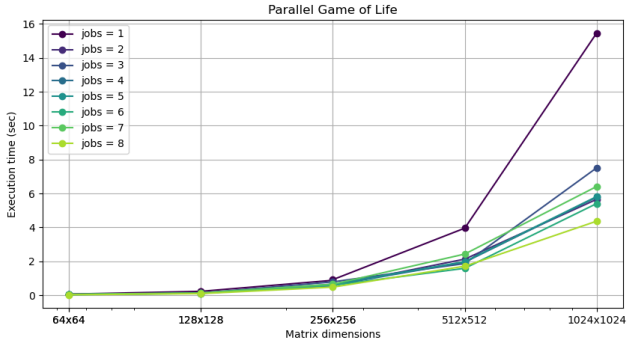
Ο πηγαίος κώδικας της άσκησης βρίσκεται στο αρχείο `src/better_mul.c`.

Η άσκηση δέχεται ως ορίσματα το πλήθος των νημάτων που θα χρησιμοποιηθούν (`-fj=<number>`), το μέγεθος του πίνακα (`-fm=<number>x<number>`) και αν αυτός είναι τριγωνικός (`-fu`) και, τέλος, αν θα χρησιμοποιηθεί η πρωτότυπη (`-f00`) ή η βελτιστοποιημένη υλοποίηση (`-f01`). Η έξοδος περιέχει τον χρόνο εκτέλεσης του αλγορίθμου, όπως και την υλοποίηση που χρησιμοποιήθηκε αλλά και το μέγεθος του πίνακα, η οποία προεραιτικά μπορεί να αποθηκευτεί σε αρχείο με το όρισμα `-ff=<filename>`.

Λόγω περιορισμών μνήμης, η άσκηση δεν μπορεί να εκτελεστεί με πίνακες μεταλύτερους από 10000×10000 (ήδη ένας πίνακας 100000×100000 δεκαδικών διπλής ακρίβειας απαιτεί 80 GB μνήμης). Παράλληλα, για πίνακες μικρότερους από 10000×10000, η βελτιστοποίηση δεν έχει νόημα, καθώς ο χρόνος εκτέλεσης είναι πολύ μικρός. Έτσι, ασχοληθήκαμε μόνο με πίνακες μεγέθους 10000×10000, για διαφορετικό αριθμό νημάτων.

Πράγματι όμως, η σκέψη μας ως προς την χρήση του `schedule()` αποδείχθηκε σωστή, καθώς η βελτιστοποιημένη υλοποίηση είναι ταχύτερη από την πρωτότυπη. Ας δούμε τα αποτελέσματα:





Βλέπουμε ότι μόλις βάλουμε τον αλγόριθμο να εκτελείται από δύο νήματα, ο χρόνος υλοποίησης μειώνεται στο μισό. Στα 8 νήματα, ο χρόνος εκτέλεσης είναι περίπου 1/4 του χρόνου εκτέλεσης της σειριακής υλοποίησης. Αυτό συμβαίνει γιατί ο αλγόριθμος είναι πολύ απλός και δεν έχει καθόλου συγχρονισμό (τα νήματα δεν γράφουν στον ίδιο πίνακα που αναγιγνώσκουν). Το ενδιαφέρον εδώ είναι το γεγονός ότι παρά το ότι το σύστημα διαθέτει 4 φυσικούς πυρήνες μόνο, έχουμε παρ'όλα αυτά βελτίωση του χρόνου εκτέλεσης όταν υπερβούμε τα 4 νήματα.

ΑΣΚΗΣΗ 6

Στην άσκηση αυτήν υλοποιούμε το Παιχνίδι της Ζωής (Conway's Game of Life) του John Conway, σε σειριακή και παράλληλη μορφή (χρησιμοποιώντας την OpenMP). Το παιχνίδι περιλαμβάνει έναν πίνακα κυττάρων τα οποία ζουν ή πεθαίνουν βάσει ορισμένων κανόνων. Παρακάτω παρατίθεται μία εικόνα με τους κανόνες του παιχνιδιού (Figure 1):

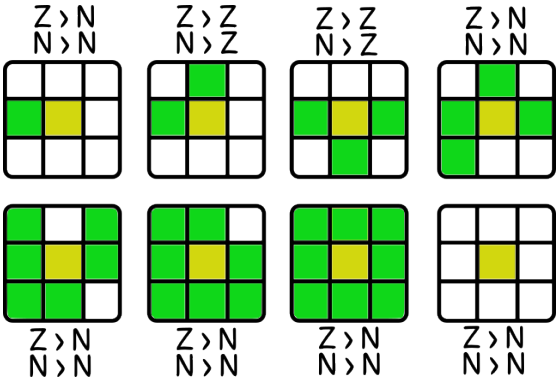


Figure 1: Κανόνες του Παιχνιδιού της Ζωής. Ο συμβολισμός $\mathrm{A} > \mathrm{B}$ σημαίνει ότι το κεντρικό κύτταρο μεταβαίνει από την κατάσταση A στην B. Τα πράσινα κύτταρα περί αυτού δηλώνουν ζωντανά κύτταρα. Z είναι η ζωντανή κατάσταση και N η νεκρή.

Ο πηγαίος κώδικας της άσκησης βρίσκεται στο αρχείο `src/game_of_life.c`.

Η άσκηση δέχεται ως ορίσματα το πλήθος των γεννεών για τις οποίες θα τρέξει το παιχνίδι (`-fg=<number>`), το πλήθος των νημάτων που θα χρησιμοποιηθούν (`-fj=<number>`), το μέγεθος του πίνακα (`-fm=<number>`, ο πίνακας είναι τετραγωνικός), και αν θα χρησιμοποιηθεί η σειριακή ή η παράλληλη υλοποίηση (`-fs` για την σειριακή και `-fp` για την παράλληλη). Η έξοδος περιέχει τον χρόνο εκτέλεσης του αλγορίθμου, όπως και την υλοποίηση που χρησιμοποιήθηκε αλλά και το πλήθος των γεννεών και το μέγεθος του πίνακα, η οποία προεραιτικά μπορεί να αποθηκευτεί σε αρχείο με το όρισμα `-ff=<filename>`.

Θεωρήσαμε τον αριθμό γεννεών σταθερό και ίσο με 1000. Τα αποτελέσματα της άσκησης είναι τα εξής:

ΑΣΚΗΣΗ 7

Στην άσκηση αυτήν προσπαθούμε να υλοποιήσουμε την μέθοδο του Gauss για την επίλυση γραμμικών συστημάτων εξισώσεων. Συγκεκριμένα, θεωρούμε ότι το σύστημά μας είναι ήδη σε τριγωνική μορφή και άρα μπορούμε να βρούμε τους αγνώστους μέσω των κάτωθι αναδρομικών τύπων:

$$x_i = \frac{b_i - \sum_{j=i}^{n} A_{ij} x_j}{A_{ii}}$$

$$x_i = \frac{b_i}{A_{ii}} - \sum_{j=0}^{i} A_{ij} x_j$$

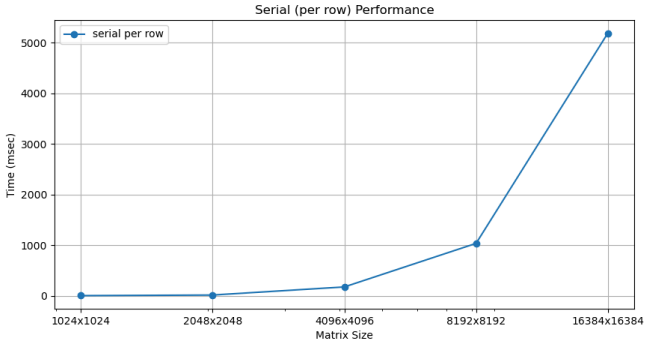
Η πρώτη μέθοδος ονομάζεται «κατά γραμμή» και η δεύτερη «κατά στήλη». Λόγω της αναδρομικότητας των τύπων αυτών, η παραλληλοποίησή τους είναι ιδιαιτέρως δύσκολη. Για την άσκηση αυτήν, δοκιμάστηκε η παραλληλοποίηση του αθροίσματος που εμφανίζεται στους τύπους, με τον υπολογισμό του όλου τύπου να γίνεται «σειριακά» (δηλαδή το x_{i+1} υπολογίζεται αφού υπολογιστεί το x_i).

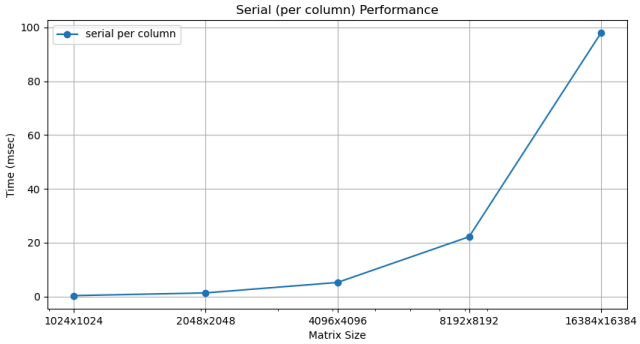
Ο πηγαίος κώδικας της άσκησης βρίσκεται στο αρχείο `src/gaussian_elimination.c`.

Η άσκηση δέχεται ως ορίσματα το πλήθος των νημάτων που θα χρησιμοποιηθούν (`-fj=<number>`), το μέγεθος του πίνακα (`-fm=<number>`, ο πίνακας είναι τετραγωνικός), αν θα χρησιμοποιηθεί η σειριακή ή η παράλληλη υλοποίηση (`-fs` για την σειριακή και `-fp` για την παράλληλη) και αν θα χρησιμοποιηθεί η κατά γραμμή (`-fpr`) ή η

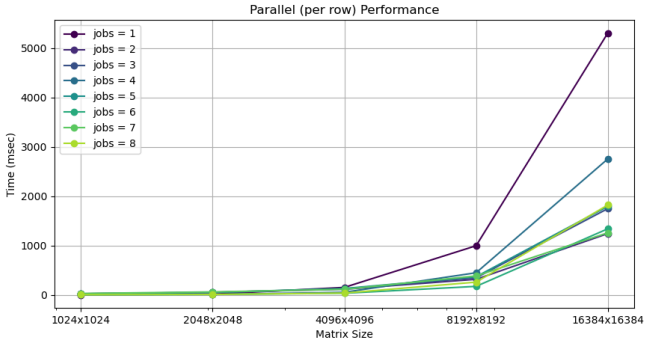
κατά στήλη (`-fpc`) μέθοδος. Η έξοδος περιέχει τον χρόνο εκτέλεσης του αλγορίθμου, όπως και την υλοποίηση που χρησιμοποιήθηκε αλλά και το μέγεθος του πίνακα, η οποία προεραιτικά μπορεί να αποθηκευτεί σε αρχείο με το όρισμα `-ff=<filename>`.

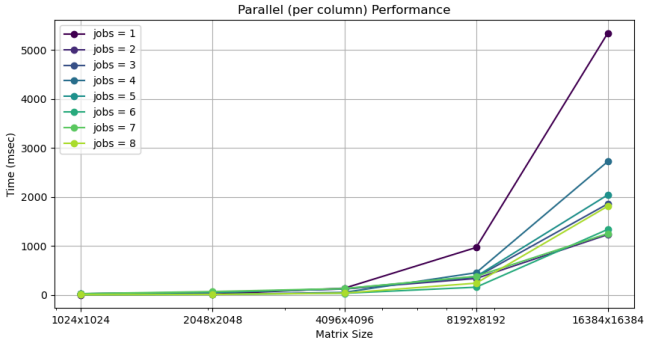
Λαμβάνουμε, λοιπόν, τα εξής αποτελέσματα για την σειριακή:





και για την παράλληλη εκτέλεση:





Από την μία, βλέπουμε πως η παραλληλοποίηση της μεθόδου «κατά γραμμή» απέδωσε καρπούς, καθώς καταφέραμε να μειώσουμε τουλάχιστον στο μισό τον χρόνο εκτέλεσης. Από την άλλη, η παραλληλοποίηση της μεθόδου «κατά στήλη» απέτυχε παταγωδώς, καθώς όχι μόνο δεν είδαμε βελτίωση, αλλά αντ᾽αυτού, *έως και πενταπλάσια* αύξηση του χρόνου εκτέλεσης *ακόμη και στην περίπτωση χρήσης ενός νήματος*.

ΑΣΚΗΣΗ 8