



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
www.cslab.ece.ntua.gr

ΣΥΣΤΗΜΑΤΑ ΠΑΡΑΛΛΗΛΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ 9ο εξάμηνο ΗΜΜΥ, ακαδημαϊκό έτος 2025-2026

Εξαμηνιαία Άσκηση

1 Εξοικείωση με το περιβάλλον προγραμματισμού

1.1 Σκοπός της Άσκησης

Σκοπός της συγκεκριμένης άσκησης είναι η εξοικείωση με τις υποδομές του εργαστηρίου (πρόσβαση στα συστήματα, μεταγλώττιση προγραμμάτων, υποβολή εργασιών κλπ) μέσα από την παραλληλοποίηση ενός απλού προβλήματος σε αρχιτεκτονικές κοινής μνήμης.

1.2 Conway's Game of Life

Το *Παιχνίδι της Ζωής* (*Conway's Game of Life*) λαμβάνει χώρα σε ένα ταμπλό με κελιά δύο διαστάσεων. Το περιεχόμενο κάθε κελιού μπορεί να είναι γεμάτο (alive) ή κενό (dead), αντικατοπτρίζοντας την ύπαρξη ή όχι ζωντανού οργανισμού σε αυτό, και μπορεί να μεταβεί από τη μία κατάσταση στην άλλη μία φορά εντός συγκεκριμένου χρονικού διαστήματος. Σε κάθε βήμα (χρονικό διάστημα), κάθε κελί εξετάζει την κατάστασή του και αυτή των γειτόνων του (δεξιά, αριστερά, πάνω, κάτω και διαγώνια) και ακολουθεί τους παρακάτω κανόνες για να ενημερώσει την κατάστασή του:

- Αν ένα κελί είναι ζωντανό και έχει λιγότερους από 2 γείτονες πεθαίνει από μοναξιά.
- Αν ένα κελί είναι ζωντανό και έχει περισσότερους από 3 γείτονες πεθαίνει λόγω υπερπληθυσμού.
- Αν ένα κελί είναι ζωντανό και έχει 2 ή 3 γείτονες επιβιώνει μέχρι την επόμενη γενιά.
- Αν ένα κελί είναι νεκρό και έχει ακριβώς 3 γείτονες γίνεται ζωντανό (λόγω αναπαραγωγής).

1.3 Δεδομένα

Σας δίνεται η σειριακή υλοποίηση του παιχνιδιού της ζωής, στον scirouter, στο αρχείο `/home/parallel/pps/2025-2026/a1/Game_Of_Life.c`.

1.4 Ζητούμενα

1. Λάβετε έγκαιρα τον κωδικό πρόσβασης στα συστήματα και επιλύστε τυχόν προβλήματα σύνδεσης.
2. Εξοικειωθείτε με τον τρόπο μεταγλώττισης και υποβολής εργασιών στις συστοιχίες. Για οδηγίες σύνδεσης, μεταγλώττισης, εκτέλεσης κ.λ.π. των προγραμμάτων σας συμβουλευτείτε τον "Οδηγό Εργαστηρίου", που είναι διαθέσιμος στο site του helios.

3. Αναπτύξτε παράλληλο πρόγραμμα στο μοντέλο κοινού χώρου διευθύνσεων (shared address space) με τη χρήση του OpenMP.
4. Πραγματοποιείτε μετρήσεις επίδοσης σε έναν από τους κόμβους της συστοιχίας των clones για 1,2,4,6,8 πυρήνες και μεγέθη ταμπλώ 64×64 , 1024×1024 και 4096×4096 (σε όλες τις περιπτώσεις τρέξτε το παιχνίδι για 1000 γενιές).
5. Συγκεντρώστε τα αποτελέσματα, τις συγκρίσεις και τα σχόλιά σας στην Τελική Αναφορά. Για κάθε μέγεθος ταμπλό, δημιουργήστε ένα διάγραμμα χρόνου και ένα διάγραμμα speedup.
6. **Προαιρετικά:** Αναζητήστε στο διαδίκτυο ειδικές αρχικοποιήσεις του ταμπλώ που οδηγούν σε ενδιαφέροντα οπτικά αποτελέσματα και δώστε την εξέλιξη των γενιών σε μορφή κινούμενης εικόνας.

2 Παραλληλοποίηση και βελτιστοποίηση αλγορίθμων σε αρχιτεκτονικές κοινής μνήμης

2.1 Παραλληλοποίηση και βελτιστοποίηση του αλγορίθμου K-means

Σκοπός της άσκησης

Στόχος της άσκησης είναι η ανάπτυξη δυο παράλληλων εκδόσεων του αλγορίθμου K-means στο προγραμματιστικό μοντέλο του κοινού χώρου διευθύνσεων με τη χρήση του προγραμματιστικού εργαλείου OpenMP. Προς αυτό το σκοπό, θα δοκιμάσετε κάποιες συνήθεις βελτιώσεις υλοποίησης παράλληλου κώδικα και θα αξιολογήσετε την τελική επίδοση του παράλληλου προγράμματος.

Ο αλγόριθμος K-means

Σκοπός του αλγορίθμου είναι ο διαχωρισμός N αντικειμένων σε k μη επικαλυπτόμενες ομάδες (συστάδες - clusters). Ο αλγόριθμος λειτουργεί ως εξής:

```
until convergence (or fixed loops)
  for each object
    find nearest cluster
  for each cluster
    calculate new cluster center coordinates.
```

Οι εκδόσεις που καλείστε να αναπτύξετε είναι δύο: μια βασική έκδοση που παραλληλοποιεί τον σειριακό αλγόριθμο με προσθήκη καταλλήλων εντολών συγχρονισμού ώστε να γίνει ορθά η παράλληλη ενημέρωση του πίνακα `newClusters` με τις νέες υπολογιζόμενες συντεταγμένες των κέντρων των συστάδων, και μια πιο εξελιγμένη έκδοση που χρησιμοποιεί τοπικούς (copied) πίνακες για κάθε νήμα, ώστε να μπορεί να γίνεται η εγγραφή χωρίς συγχρονισμό. Στη δεύτερη έκδοση μετά τους τοπικούς υπολογισμούς τα αποτελέσματα των τοπικών αυτών πινάκων θα πρέπει να συνδυάζονται (reduction) στον πίνακα `newClusters`.

Δεδομένα

Για την υλοποίηση του αλγορίθμου σας δίνεται ο σειριακός αλγόριθμος, καθώς και σκελετοί και για τις δύο παράλληλες υλοποιήσεις στο `/home/parallel/pps/2025-2026/a2/kmeans/`, στους οποίους καλείστε να συμπληρώσετε τα κενά. Η κλήση όλων των υλοποιήσεων γίνεται μέσω της `main.c`, η οποία είναι υπεύθυνη για την δημιουργία ενός ζητούμενου μεγέθους dataset (καλώντας την `file_io.c`) και την κλήση του κατάλληλου αλγορίθμου K-means.

Ζητούμενα

Για τις 2 παράλληλες εκδόσεις που σας δίνονται:

shared clusters

1. Παραλληλοποιήστε αυτή την έκδοση του αλγορίθμου, προσέχοντας το διαμοιρασμό των δεδομένων μεταξύ των threads, χρησιμοποιώντας το OpenMP και πραγματοποιήστε μετρήσεις για το configuration `{Size, Coords, Clusters, Loops} = {256, 16, 32, 10}` για `threads = {1, 2, 4, 8, 16, 32, 64}` στο μηχανήμα *sandman*. Δημιουργήστε το barplot διάγραμμα χρόνου εκτέλεσης που προκύπτει (x-axis = sequential και threads, y-axis = time), και το αντιστοίχο speedup plot (x-axis = sequential και threads, y-axis = seq_time/time). Πώς μπορείτε να εξηγήσετε την επίδοση της παράλληλης έκδοσης σε σχέση με την σειριακή;
2. Αναζητήστε πληροφορίες για τη χρήση της μεταβλητής περιβάλλοντος `GOMP_CPU_AFFINITY` (και χρησιμοποιήστε τη σε όλα τα υπόλοιπα ερωτήματα), η οποία "προσδένει" τα threads σε συγκεκριμένους πυρήνες για όλη την εκτέλεση (thread binding) και επαναλάβετε τις μετρήσεις και τα διαγράμματα. Τι διαφορά βλέπετε και πώς τη δικαιολογείτε;

copied clusters and reduce

1. Παραλληλοποιήστε αυτή την έκδοση του αλγορίθμου (αγνοώντας τα *"hints για false-sharing"*) και επαναλάβετε τις προαναφερόμενες μετρήσεις και διαγράμματα. Συγκρίνετε την επίδοση και τον παραλληλισμό σε σχέση με την προηγούμενη έκδοση. Επιτυγχάνει η συγκεκριμένη έκδοση ικανοποιητική επίδοση;
2. Δοκιμάστε το ακόλουθο configuration $\{\text{Size, Coords, Clusters, Loops}\} = \{256, 1, 4, 10\}$, και συγκρίνετε το scalability με το προηγούμενο $\{\text{Size, Coords, Clusters, Loops}\} = \{256, 16, 32, 10\}$. Τι διαφορά παρατηρείτε στα scalability plots; Αναζητήστε πληροφορίες για την πολιτική **first-touch** του Linux και για τα φαινόμενα **false-sharing**. Με βάση αυτά, τι πρόβλημα δημιουργείται στην πραγματική τοποθέτηση των `local_newClusters` δεδομένων στη μνήμη; Προσπαθήστε να ξεπεράσετε αυτό το πρόβλημα (Hint: οι memory allocators προσπαθούν να αποφύγουν τον κατακερματισμό της μνήμης). Αναφέρετε τον καλύτερο χρόνο που επιτύχατε!
3. **Προαιρετικά (bonus)** - NUMA aware allocation: Στο configuration $\{\text{Size, Coords, Clusters, Loops}\} = \{256, 1, 4, 10\}$ του προηγούμενου ερωτήματος δοκιμάστε να βελτιώσετε την επίδοση του προγράμματος, λαμβάνοντας υπόψη τα NUMA χαρακτηριστικά του μηχανήματος και το διαμορισμό του πίνακα *objects* στα memory nodes, ο οποίος αρχικοποιείται στο αρχείο `"file_io.c"` (Hint: αξιοποιήστε την πολιτική first-touch των Linux που είδαμε πιο πάνω). Δοκιμάστε και το προηγούμενο configuration $\{\text{Size, Coords, Clusters, Loops}\} = \{256, 16, 32, 10\}$ με την νέα εκδοχή. Τι παρατηρείτε; Ποιο είναι το κυρίαρχο bottleneck σε κάθε configuration;

Αμοιβαίος Αποκλεισμός - Κλειδώματα **NEW!**

Στο ερώτημα αυτό καλείστε να αξιολογήσετε διαφορετικούς τρόπους υλοποίησης κλειδωμάτων για αμοιβαίο αποκλεισμό. Το κρίσιμο τμήμα που θα χρειαστεί να προστατευτεί είναι η ενημέρωση του πίνακα `newClusters` στην `"shared-clusters"` υλοποίηση του K-means. Συγκεκριμένα, θα μελετήσετε τα παρακάτω είδη κλειδωμάτων:

- **nosync_lock:** Η συγκεκριμένη υλοποίηση δεν παρέχει αμοιβαίο αποκλεισμό οπότε δεν παράγει και σωστά αποτελέσματα. Ωστόσο, θα χρησιμοποιηθεί ως άνω όριο για την αξιολόγηση της επίδοσης των υπόλοιπων κλειδωμάτων.
- **pthread_mutex_lock:** Το `pthread_mutex_t` κλειδωμά που παρέχει η βιβλιοθήκη Pthreads.
- **pthread_spin_lock:** Το `pthread_spinlock_t` κλειδωμά που παρέχει η βιβλιοθήκη Pthreads.
- **tas_lock:** Το *test-and-set* κλειδωμά όπως περιγράφεται στις διαφάνειες του μαθήματος.
- **ttas_lock:** Το *test-and-test-and-set* κλειδωμά όπως περιγράφεται στις διαφάνειες του μαθήματος.
- **array_lock:** Το array-based κλειδωμά όπως περιγράφεται στις διαφάνειες του μαθήματος.
- **clh_lock:** Ένα είδος κλειδώματος που στηρίζεται στη χρήση μίας συνδεδεμένης λίστας. Αναλυτικές πληροφορίες μπορείτε να βρείτε στο Κεφάλαιο 7 του βιβλίου *"The Art of Multiprocessor Programming"*.

Στο path `/home/parallel/pps/2025-2026/a2/kmeans_locks/` θα βρείτε (επιπλέον σε σχέση με την προηγούμενη άσκηση), το Makefile για την παραγωγή των εκτελέσιμων της μορφής `kmeans_omp_<lock-name>_lock`, για κάθε κλειδωμά ξεχωριστά, καθώς και τα αρχεία:

- `omp_lock_kmeans.c` : χρησιμοποιούνται τα διάφορα κλειδώματα για την προστασία του κρίσιμου τμήματος.
- `omp_critical_kmeans.c` : χρησιμοποιείται το OpenMP directive `#pragma omp critical` για την προστασία του κρίσιμου τμήματος.

Στο subdirectory `locks` θα βρείτε έτοιμες τις υλοποιήσεις των κλειδωμάτων.

1. Συγκεντρώστε μετρήσεις για κάθε είδος lock με το configuration $\{\text{Size, Coords, Clusters, Loops}\} = \{32, 16, 32, 10\}$ για `threads = \{1, 2, 4, 8, 16, 32, 64\}` στο μηχανήμα `sandman` (εφαρμόζοντας `thread-binding`). Συγκρίνετε τους χρόνους εκτέλεσης (τόσο των κλειδωμάτων, όσο και

του `omp critical`) με την `shared-clusters` υλοποίησή σας από την προηγούμενη άσκηση όπου για την προστασία του κρίσιμου τμήματος χρησιμοποιήσατε τα OpenMP directives `#pragma omp critical` ή/και `#pragma omp atomic`. Εξηγήστε τη λειτουργία του κάθε κλειδώματος. Πώς αυτή επηρεάζει την επίδοση του;

2.2 Παραλληλοποίηση του αλγορίθμου Floyd-Warshall

Σκοπός της άσκησης

Στόχος της άσκησης είναι να εξοικειωθείτε με τη χρήση των OpenMP tasks παράλληλοποιώντας τον αλγόριθμο Floyd-Warshall για αρχιτεκτονικές κοινής μνήμης.

Ο αλγόριθμος Floyd-Warshall

Ο αλγόριθμος των Floyd-Warshall (FW) υπολογίζει τα ελάχιστα μονοπάτια ανάμεσα σε όλα τα ζεύγη των N κόμβων ενός γράφου (all-pairs shortest path). Θεωρώντας το γράφο αποθηκευμένο στον πίνακα γειτνίασης A , ο αλγόριθμος έχει ως εξής:

```
for (k=0; k<N; k++)
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      A[i][j] = min(A[i][j], A[i][k]+A[k][j]);
```

Εκτός από την *standard* έκδοση του αλγορίθμου, έχουν προταθεί άλλες δύο εκδόσεις, μία *αναδρομική* (*recursive*) και μία *tiled*. Σε αυτή την άσκηση καλείστε να παραλληλοποιήσετε την **recursive** υλοποίηση, η οποία δεν μπορεί να παραλληλοποιηθεί με `parallel for` αλλά απαιτείται για αυτή την περίπτωση η χρήση `tasks`.

Δεδομένα

Η σειριακή έκδοση του αλγορίθμου Floyd-Warshall (3 versions) σας δίνεται στον `scirouter` στο φάκελο `/home/parallel/pps/2025-2026/a2/FW/`.

Ζητούμενα

- Υλοποιήστε μια παράλληλη έκδοση του **recursive** αλγορίθμου Floyd-Warshall χρησιμοποιώντας OpenMP tasks και πραγματοποιήστε μετρήσεις για μεγέθη πινάκων 1024×1024 , 2048×2048 και 4096×4096 για `threads = {1, 2, 4, 8, 16, 32, 64}` στο μηχανήμα *sandman*. Δημιουργήστε τα 3 `barplot` διαγράμματα χρόνου εκτέλεσης που προκύπτουν (x -axis = sequential και threads, y -axis = time).
- Προαιρετικά (bonus):** Υλοποιήστε παράλληλη έκδοση του **tiled** αλγορίθμου Floyd-Warshall και συγκεντρώστε μετρήσεις για όμοια μεγέθη πινάκων. Συγκρίνετε επιδόσεις με την recursive έκδοση. Τι παρατηρείτε;

2.3 Περιβάλλον εκτέλεσης

- Για την άσκηση αυτή, θα χρησιμοποιήσετε το μηχανήμα *sandman*, που ανήκει στην ουρά *serial* (βλ. παρουσίαση άσκησης). Για να υποβάλλετε ένα script, έστω `script.sh`, στο μηχανήμα, δίνετε την εντολή `qsub` ως εξής:

```
$ qsub -q serial -l nodes=sandman:ppn=64 script.sh
```
- ΠΡΟΣΟΧΗ: Μπορείτε να χρησιμοποιείτε τα μηχανήματα της συστοιχίας *parlab* για την ανάπτυξη και τον έλεγχο παράλληλων προγραμμάτων.