*[Ονόματα, παπατζες κλπ]*

# 1. Εξαρτήσεις Δεδομένων

Ξεκινώντας να προσεγγίσουμε την παραλληλοποίηση του αλγορίθμου FW στις διαφορετικές υλοποιήσεις που δίνονται, εντοπίσαμε ορισμένες ομοιότητες μεταξύ τους. Ο αλγόριθμος δρα και στις 3 υλοποιήσεις με μία παρόμοια σειρά εξαρτήσεων εκτέλεσης καθώς ενημερώνονται διαφορετικά σημεία του πίνακα, όπως παρουσιάζεται στις διαφάνειες για το εργαστήριο.

Αρχικά, στην κλασική εκδοχή είναι υποχρεωτική η ολοκλήρωση της k επανάληψης πριν την εκτέλεση της k+1, το οποίο υποννοεί μία 'διαγώνιο' κίνηση του αλγορίθμου μέσα στον πίνακα. Έπειτα στην αναδρομική εκδοχή η εκτέλεση των αναδρομικών κλήσεων σε κάθε τεταρτημόριο ακολουθεί τη σειρά εξαρτήσεων Α11 → {Α12, Α21} → Α22 → Α22 → {Α12, Α21} → Α11 που παρουσιάζει μία όμοια εικόνα. Ακόμη, η τρίτη εκδοχή παρουσιάζει εξάρτηση τόσο ανάμεσα στα k (που αντιπροσωπεύουν πλέον tiles μεγέθους Β) όσο και κατά τη διάρκεια της εκτέλεσης για κάθε k: Προτεραιότητα έχει η αρχική κλήση του αλγορίθμου FW, μετά την εκτέλεση της οποίας έπονται τα 4 for loops για την ενημέρωση των 'αξόνων' από tiles πάνω, κάτω και εκατέρωθεν του (k,k). Τέλος ενημερώνονται τα ορθογώνια τμήματα ανάμεσα στους 'άξονες' χρησιμοποιώντας πληροφορία από αυτούς.

# 2. Η παράμετρος Β

Οι δύο νέες υλοποιήσεις χρησιμοποιούν και οι δύο μία παράμετρο Β δεδομένη από το χρήστη, η οποία αντιπροσωπεύει το μέγεθος του υποπίνακα στο οποίο ξεκινά να εκτελείται τοπικά ο κλασικός FW. Παρατηρήσαμε μετά από δοκιμές σε διαφορετικά μηχανήματα και στο sandman ότι το μέγεθος της παραμέτρου Β που οδηγεί σε βέλτιστα χρονικά αποτελέσματα παραμένει περίπου σταθερό σε κάθε μηχάνημα ανεξαρτήτως υλοποίησης παραλληλίας και μεγέθους πίνακα εισόδου. Το αποτέλεσμα αυτό είναι αναμενόμενο, αφού το βέλτιστο Β εξαρτάται κατά κόρον από το μέγεθος της κρυφής μνήμης που διαθέτει κάθε thread.

Όταν το B είναι πολύ μικρό, ο υποπίνακας χωράει ολόκληρος στην κρυφή μνήμη που διατίθεται σε κάθε thread αλλά εκτελούνται υπερβολικά πολλές αναδρομικές κλήσεις / χωρίζεται ο πίνακας σε πολλά tiles. Αυτό έχει ως αποτέλεσμα να αναλώνονται οι πόροι του συστήματος σε δημιουργία threads στις περισσότερες παράλληλες εκδοχές μας. Αντίθετα όταν είναι υπερβολικά μεγάλο δεν χωράει στην κρυφή μνήμη κάθε thread, οπότε και αναλώνεται το σύστημα σε memory fetches.

# 3. OpenMP

Για την παραλληλοποίηση των διαφορετικών εκδοχών του αλγορίθμου αποφασίσαμε να χρησιμοποιήσουμε OpenMP λόγω της απλότητας του, θυσιάζοντας κάποια ευελιξία έναντι της χρήσης TBB. Βάση των παραπάνω παρατηρήσεων κάναμε ορισμένες 'ενημερωμένες' πειραματικές δοκιμές χρησιμοποιώντας τα constructs omp for, omp sections και omp task/taskwait.

# 4. Παραλληλοποίηση του κλασικού Floyd-Warshall

Για τον κλασσικό αλγόριθμο ο κώδικας εγγενώς δεν προσφέρεται για σημαντική βελτιστοποίηση σε παράλληλα συστήματα, οπότε η καλύτερη προσέγγιση μας ήταν η παραλληλοποίηση των υπολογισμών σε κάθε θέση του πίνακα ανά βήμα του μετρητή k:

for(k=0;k<N;k++)

#pragma omp parallel for shared(A) private(i,j) firstprivate(k, N)

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]);

*[Αποτελέσματα εδώ]*

# 5. Παραλληλοποίηση της Recursive Υλοποίησης

Για τη recursive υλοποίηση παρατηρούμε ότι η μόνη περιοχή που είναι παραλληλοποιήσιμη σε κάθε αναδρομική κλήση είναι η ενημέρωση των περιοχών Α12, Α21 καθώς δεν υπάρχει εξάρτηση μεταξύ τους. Αρχικά επιχειρήσαμε τη χρήση του omp sections construct για την παραλληλοποίηση αυτών των περιοχών:

FWR (A11, B11, C11);

#pragma omp sections

{

#pragma omp section

FWR (A12, B11, C12);

#pragma omp section

FWR (A21, B21, C11);

}

FWR (A22, B21, C12);

FWR (A22, B21, C12);

#pragma omp sections

{

#pragma omp section

FWR (A21, B21, C11);

#pragma omp section

FWR (A12, B11, C12);

}

FWR (A11, B11, C11);

*[Συντομη παρουσίαση κάποιων αποτελεσμάτων θα ελεγα, πχ για το καλύτερο Β χρόνοι για διαφορους πυρήνες αναφορικά χωρις γραφημα or something]*

Έπειτα χρησιμοποίησαμε omp tasks τα οποία όντας πιο lightweight δομές επιτρέπουν καλύτερη οργάνωση των πόρων του συστήματος κατά την δημιουργία threads:

FWR (A11, B11, C11);

#pragma omp taskwait

#pragma omp task

FWR (A12, B11, C12);

#pragma omp task

FWR (A21, B21, C11);

#pragma omp taskwait

FWR (A22, B21, C12);

FWR (A22, B21, C12);

#pragma omp taskwait

#pragma omp task

FWR (A21, B21, C11);

#pragma omp task

FWR (A12, B11, C12);

#pragma omp taskwait

FWR (A11, B11, C11);

*[Παρουσίαση καλύτερων (?) αποτελεσμάτων]*

Σημ.: Πιο πολύπλοκες προσεγγίσεις, συμπεριλαμβανομένης και της παραλληλοποίησης του τμήματος που εφαρμόζεται ο κλασικός FW αποδείχθηκαν μη αποδοτικές.

# 5. Παραλληλοποίηση της Tiled Υλοποίησης

Για την tiled εκδοχή του αλγορίθμου αρχικά μας φάνηκε προφανής μία προσέγγιση αντίστοιχη της recursive, δηλαδή task centric παραλληλοποίηση των independent τμημάτων του κώδικα:

for k = 1:N step B {

FW(k,k)

#pragma omp taskwait

#pragma omp task

for all tiles above

apply FW

#pragma omp task

for all tiles below

apply FW

#pragma omp task

for all tiles right

apply FW

#pragma omp task

for all tiles left

apply FW

#pragma omp taskwait

#pragma omp task

for all tiles up right

apply FW

#pragma omp task

for all tiles up left

apply FW

#pragma omp task

for all tiles down right

apply FW

#pragma omp task

for all tiles down left

apply FW

#pragma omp taskwait

}

*[Πιο σύντομη παρουσίαση αποτελεσμάτων]*

Ο κώδικας παρουσίασε σαφή βελτίωση με αυτή την προσέγγιση, παρ' αυτα η πιο αποδοτική στρατηγική αποδείχθηκε απλούστερη, συγκεκριμένα η εκτέλεση όλων των for loops παραλληλοποιόντας το καθένα με τη σειρα:

for k = 1:N step B {

FW(k,k)

#pragma omp for

for all tiles above

apply FW

#pragma omp for

for all tiles below

apply FW

#pragma omp for

for all tiles right

apply FW

#pragma omp for

for all tiles left

apply FW

#pragma omp for

for all tiles up right

apply FW

#pragma omp for

for all tiles up left

apply FW

#pragma omp for

for all tiles down right

apply FW

#pragma omp for

for all tiles down left

apply FW

}

*[Παρουσίαση αποτελεσμάτων]*

Για άλλη μία φορά, πιο πολύπλοκες υλοποιήσεις, συμπεριλαμβανομένου και του υβριδικού μοντέλου (παράλληλα for και κατανομή σε tasks) δεν επέφεραν καλύτερη απόδοση.