Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Συστήματα Παράλληλης Επεξεργασίας

2η Εργαστηριακή Άσκηση - Ενδιάμεση Αναφορά

9ο Εξάμηνο - Ακαδημαϊκό Έτος 2020-2021

Γιαννόπουλος Εμμανουήλ - 03117031

Παγώνης Γεώργιος - 03117030

28 Νοεμβρίου 2021

1 Εισαγωγή

Σε αυτή την άσκηση σκοπός είναι να παραλληλοποιήσουμε τον αλγόριθμο εύρεσης συντομότερων μονοπατιών μεταξύ κάθε ζεύγους κόμβων **Floyd-Warshall**. Για το σκοπό αυτό θα μελετήσουμε 3 διαφορετικές υλοποιήσεις του αλγόριθμου (FW, FW-Recursive και FW-Tiled) και θα εξετάσουμε την παραλληλοποίηση που προσφέρει κάθε υλοποίηση καθώς και την επίδοσή τους.

2 FW

Η πρώτη υλοποίηση του αλγορίθμου που εξετάζουμε είναι και η απλούστερη αποτελούμενη από 3 nested for loops. Η σειριακή υλοποίηση είναι η εξής:

Algorithm 1 FW

```
for k, 1 to n do
    for i, 1 to n do
        for j, 1 to n do
            A[i][j] = min(A[i][j], A[i][k] + A[k][j])
        end for
    end for
end for
```

Σε μια πρώτη ματιά το $A[i][j]$ εξαρτάται από τα $A[i][k]$ και $A[k][j]$ και η παραλληλοποίηση φαίνεται δύσκολη. Όμως ισχύει ότι:

$$A[i][k] = \min(A[i][k], A[i][k] + A[k][k]) = A[i][k]$$
$$A[k][j] = \min(A[k][j], A[k][k] + A[k][j]) = A[k][j]$$

Παρατηρούμε ότι τα 2 dependencies του $A[i][j]$ διατηρούν τις τιμές τους και έτσι τα 2 inner loops είναι πλήρως παραλληλοποιήσιμα. Υλοποιήσαμε την παραλληλοποίησή τους με OpenMP και εξήγαμε τις παρακάτω (αρχικές) μετρήσεις. Υπάρχει μια ανωμαλία στην μέτρηση μεγέθους 4096×4096 για 32 threads την οποία προσπαθούμε ακόμη να εξηγήσουμε (παρουσιάζεται κάθε φορά που κάνουμε νέες μετρήσεις). Πιθανολογούμε ότι οφείλεται στην αρχιτεκτονική του sandman (για 32 threads εισάγεται 2ος επεξεργαστής και πληρώνουμε την επικοινωνία μεταξύ τους) αν και για 64 threads ο χρόνος βελτιώνεται πάλι.

FW

Threads \ Size	1024×1024	2048×2048	4096×4096
1	2.1542	17.5780	144.2793
2	1.1182	8.5487	82.8490
4	0.6164	4.3748	37.4704
8	0.3696	2.3217	22.8955
16	0.2393	1.4516	26.5718
32	0.2486	1.3367	50.4221
64	0.3787	0.8835	19.7523

3 FW-Recursive

Η δεύτερη υλοποίηση του αλγορίθμου που εξετάζουμε είναι η αναδρομική. Σε αυτήν, ο κάθε υποπίνακας που εξετάζουμε χωρίζεται σε άλλους 4 υποπίνακες $A_{00}, A_{01}, A_{10}, A_{11}$ μέχρι να φτάσουμε σε κάποιο base case. Για λόγους ευκολίας της υλοποίησης δίνονται τρεις πίνακες A, B, C ώστε να μη χρειάζεται να κάνουμε overwrite στον πίνακα A και να μπορούμε να παραλληλοποιήσουμε ευκολότερα. Η σειριακή υλοποίηση είναι η εξής:

Algorithm 2 FW-Recursive | FWR(A,B,C)

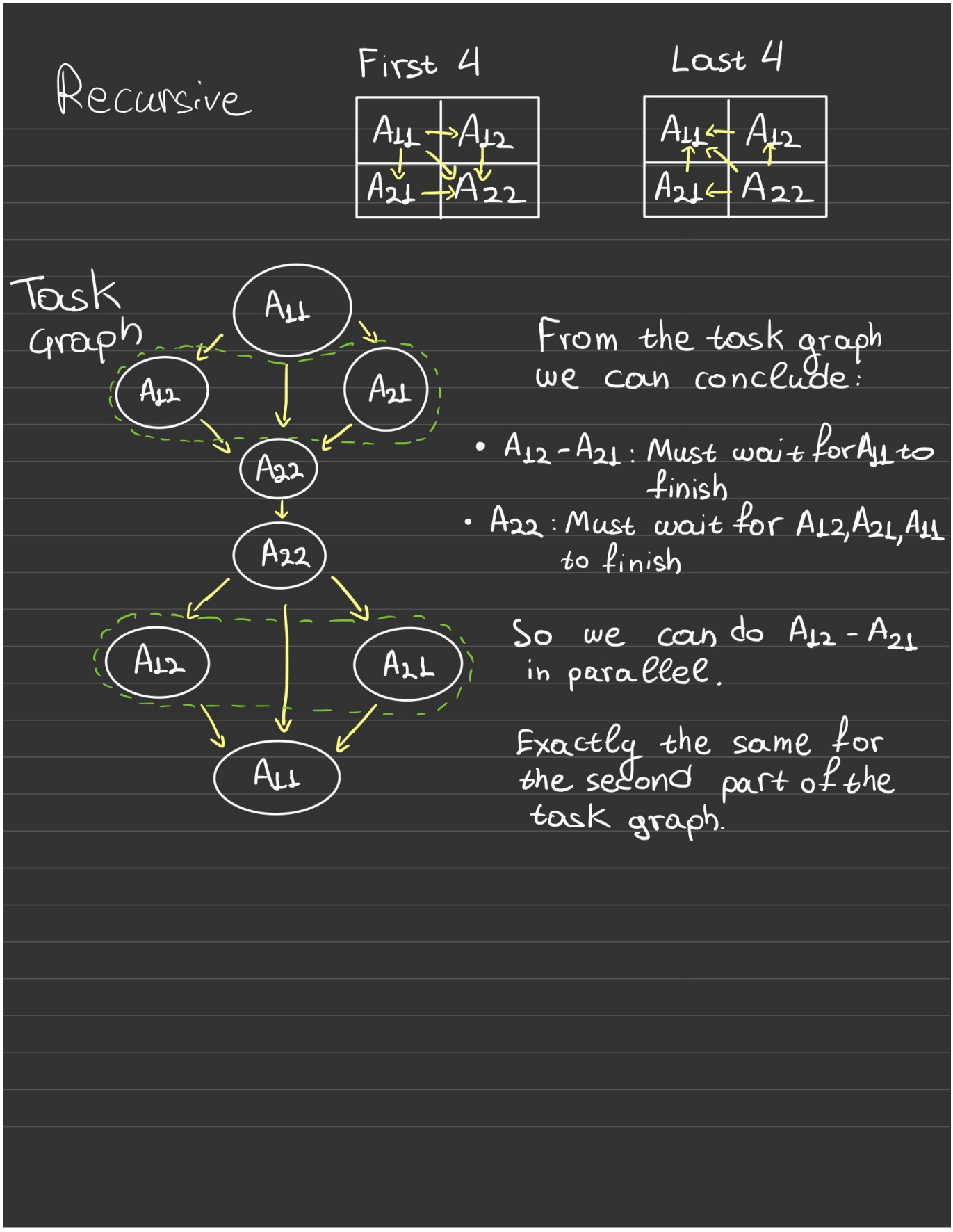
if base case **then**
  FWI(A,B,C)
else
  FWR(A_{00}, B_{00}, C_{00})
  FWR(A_{01}, B_{00}, C_{01})
  FWR(A_{10}, B_{10}, C_{00})
  FWR(A_{11}, B_{10}, C_{01})
  FWR(A_{11}, B_{11}, C_{11})
  FWR(A_{10}, B_{11}, C_{10})
  FWR(A_{01}, B_{01}, C_{11})
  FWR(A_{00}, B_{01}, C_{10})
end if

Στην επόμενη σελίδα φαίνεται το task graph (έχει γίνει 1-indexing στο σχήμα). Επίσης, υλοποιήσαμε την παραλληλοποίηση που δείχνουμε στο task graph σε OpenMP και εξήγαμε τις παρακάτω μετρήσεις. Η (αρχική) υλοποίησή μας είναι task-based με αρκετό περιττό overhead ορίζοντας πολλές parallel περιοχές και αυτός είναι ο λόγος που βλέπουμε να κλιμακώνει αρνητικά για πάνω απο 2 threads. Πιστεύουμε ότι για τη συγκεκριμένη task-based υλοποίηση μάλλον είναι καταλληλότερα τα TBBs και θα χρησιμοποιήσουμε αυτά για την τελική έκδοση.

FW-RECURSIVE

Threads \ Size	1024×1024	2048×2048	4096×4096
1	0.8371	6.6749	55.6581
2	0.5004	3.5985	29.1000
4	0.5095	3.7383	29.4581
8	0.5674	3.8284	29.7678
16	0.5913	4.1321	30.4837
32	0.5511	4.1361	32.0966
64	0.8031	4.8234	35.1201

Σχήμα 1: FW-recursive task graph



4 FW-Tiled

Η τρίτη υλοποίηση του αλγορίθμου που εξετάζουμε είναι η tiled. Σε αυτήν, χωρίζουμε τον πίνακα σε blocks $B \times B$. Σε κάθε timestep, υπολογίζουμε αρχικά το block A_{tt}. Έπειτα, υπολογίζουμε όλα τα blocks A_{it}, Atj και έπειτα όλο τον υπόλοιπο πίνακα. Παρακάτω φαίνεται το task graph. Στο task graph συμβολίζουμε τις περιοχές με βάση τη γεωγραφική τους θέση ως προς το block A_{tt} (N = north, NW = north-west κ.ο.κ.). Υλοποιήσαμε το παρακάτω task graph σε OpenMP με ένα parallel section, pragma omp for nowait και barriers όπου έπρεπε και εξήγαμε τις εξής (αρχικές) μετρήσεις. Οι μετρήσεις αυτές φαίνεται να είναι πολύ κοντα σε ένα καλό τελικό αποτέλεσμα.

FW-TILED

Threads \ Size	1024×1024	2048×2048	4096×4096
1	0.8357	6.3135	52.0147
2	0.4464	3.2310	26.4611
4	0.2533	1.7123	13.1537
8	0.1539	0.8969	6.8161
16	0.1340	0.7892	4.1138
32	0.1271	0.5084	3.7346
64	0.2722	0.6260	2.1961

Σχήμα 2: FW-tiled task graph

Tiled

Current Tile

E, S, N, W : These can be in parallel

NW, NE, SW, SE : These can be in parallel

CR_1	E_1	E_1	E_1
S_1	SE_1	SE_1	SE_1
S_1	SE_1	SE_1	SE_1
S_1	SE_1	SE_1	SE_1

NW_2	N_2	NE_2	NE_2
W_2	CR_2	E_2	E_2
SW_2	S_2	SE_2	SE_2
SW_2	S_2	SE_2	SE_2

NW_3	NW_3	N_3	NE_3
NW_3	NW_3	N_3	NE_3
W_3	W_3	CR_3	E_3
SW_3	SW_3	S_3	SE_3

NW_4	NW_4	NW_4	N_4
NW_4	NW_4	NW_4	N_4
NW_4	NW_4	NW_4	N_4
W_4	W_4	W_4	CR_4

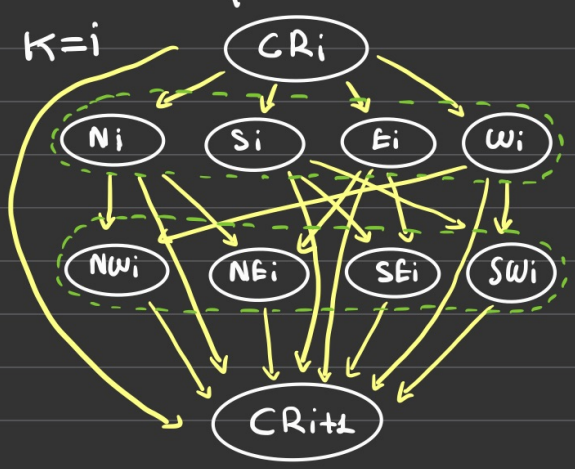
From the task graph we can conclude:

- A_{12} - A_{21} : Must wait for A_{11} to finish
- A_{22} : Must wait for A_{12}, A_{21}, A_{11} to finish

So we can do A_{12} - A_{21} in parallel.

Exactly the same for the second part of the task graph.

Task Graph



From the task graph we can conclude:

- N_i - S_i - E_i - W_i : Can be in parallel
- NW_i - NE_i - SE_i - SW_i : Can be in parallel