Eksamen i IN3200
High-Performance Computing and Numerical Projects

Kandidatnummer: 15722

30.03.21

read_graph_from_file1

Leser inn filen med `fopen`, skipper linjene som ikke er relevant, henter ut antall noder og kanter. Går deretter gjennom filen linje for linje og setter alle kanter. Gjør nødvendige sjekker før en kant plasseres, sjekker at det ikke er en kant til seg selv, og at begge nodene er innenfor antall noder. Hvis noe av dette ikke er tilfellet, ignoreres denne linjen.

Når vi setter inn verdiene i arrayet, hopper vi rundt i minnet, siden indeks [i][j] og [j][i] ikke ligger ved siden av hverandre. Dette vil ta noe tid, men jeg kunne ikke se noen måte å unngå dette på. Vi vet ingenting om hvilke kanter som kommer og i hvilken rekkefølge de kommer, så det er vanskelig å ta noen hensyn her for å unngå hopp i minnet.

read_graph_from_file2

Må lese gjennom filen to ganger, en gang for å sette `row_ptr`, og en gang for å sette `col_idx`. For å sette `row_ptr`, så går jeg først gjennom kantene, med de samme nødvendige sjekkene for i `file1`, og øker på riktig indeks. Nå får vi bare antall naboer til hver node, derfor går jeg gjennom `row_ptr` en gang til etter alle kantene er lest inn for å oppdatere den slik at indeksene stemmer. Kunne kanskje oppdatert dette underveis, men da ville vi gått mye mer frem og tilbake i arrayet under innlesningen, det ville blitt flere

operasjoner for å oppdatere arrayet, og mer frem og tilbake. Jeg syns det er ryddigere å finne antall naboer per node først, og deretter oppdatere hele arrayet en gang når vi vet hvor mye vi skal øke hver indeks med.

Alle kantene leses inn igjen for å sette `col_idx`, bruker et array `rowcopy` som er en kopi av `row_ptr` for å holde styr på hvilken nabo som skal settes. Oppdaterer `rowcopy` på den gjeldene indeksen når en nabo settes, slik at vi vet hvor neste nabo til denne noden skal. `rowcopy` fjernes etter denne bruken.

Sorterer til slutt naboene til hver node, slik at det skal være enklere å lage SNN til senere.

Trenger informasjon fra `row_ptr` for å sette opp `col_idx`, så det er nødvendig å gå gjennom filen to ganger.

create_SNN_graph1

Mistenker at det finnes en mer effektiv måte å løse denne på. Siden arrayet er symmetrisk om diagnoalen, burde det vært mulig å kun gjøre halve jobben, og oppdatere `x[i][j]` og `x[j][i]` samtidig. Eller en mer elegant og effektiv måte å sammenligne naboer til noder på, er ikke helt fornøyd med denne løseningen med tanke på effektiviteten.

Tanken her er en for-loop som går gjennom hver indeks i `table2D`, for hver indeks `(i, j)` som er en, så sammenlignes gjeldene `(i)` rad, med raden vi fant en på `(j)`, og sjekker hvor mange steder de begge har en. Dette gir antall felles naboer for node `i` og `j`.

Parallellisering: Holder variabelene i,j,k private for å unngå at flere tåder prøver å oppdatere samme sted i arrayet samtidig. Mulig det hadde holdt å ha j private her. Har gått for dynamisk schedulering siden det kan være veldig varierende antall naboer til en node, og dermed vil noen tråder få mer jobb enn andre. Mine tester har også gitt raskere kjøretid ved dynamisk shedulering enn ved statisk.

Får ca kjøretid på 2.2 sekunder ved sekvensiell kjøring, som forbedres til ca 0.17 sekunder ved openMP.

create_SNN_graph2

Går gjennom alle noder, og hver nodes naboer, for hver node i,j som er naboer, sammenlignes deres liste med naboer. Siden denne er sortert på størrelse, kan vi sammenligne disse listene på lineær tid.

Parallellisering: Viktig at ikke to tråder oppdaterer samme sted i arrayet samtidig, så holder v privat. Her også med dynamisk schedulering fordi antall naboer vil kunne variere veldig.

Her er kjøretiden ca 0.2 sekunder ved sekvensiell kjøring, og 0.01 sekunder ved openMP.

Testet på ifi maskin. Selve koden godkjennes av valgrind, men får noe minnefeil når jeg la til -fopenmp flagget og parallelliseringen. Fikk beskjed om at jeg kunne ignorere disse klagene.