Ukeoppgaver i uke 4, INF2440 – v2014

Denne uka skal vi også se på det å multiplisere to nxn matriser A og B sammen og regne ut resultatmatrisen C, altså C = AxB , først sekvensielt (koden for en rett fram implementasjon av den sekvensielle versjonen finner du nederst i tips) og så parallelliser dette. Denne gangen skal vi dele opp data på en annen måte for trådene enn i Uke3. Anta at du har n rader og n kolonner i matrisen din og k tråder på din maskin. I uke3 skulle hver tråd beregne n/k ***kolonner*** i C, nå i uke4 skal vi se hva som skjer når trådene skal beregne n/k ***rader*** i C (husk at den siste tråden må regne ut alle de siste radene hvis n ikke er delbar med k).

Resultatet av AxB er selv en nxn matrise C, og vi har at element c[i][j] i matrisen C er da definert som:

$$\mathrm{c[i][j]} = \sum_{k=0}^{n} \mathrm{a[i][k]} * \mathrm{b[k][j]}$$

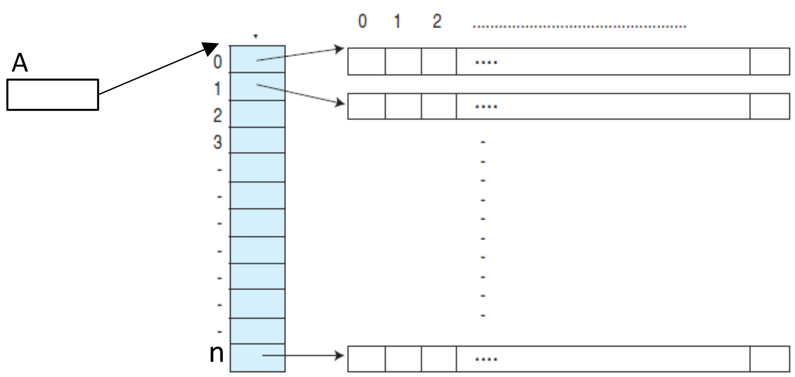
Vi multipliserer altså hvert element i rad i fra A parvis med hvert element fra kolonne j fra B og summerer de n multiplikasjonene for å få c[i][j] (se koden).

Sekvensiell implementasjon – som i Uke3 – du kan bruke samme kode.

a) Du skal før initiere matrisene A og b med tilfeldige double-tall fra klassen Random med metoden nextDouble(). Du får da et tilfeldig tall mellom 0,0 og 1.0. (Hvis du først vil teste koden din, kan du initiere A og B med 1.0, og da vil alle elementene i C være n.)

   Test kjøretiden for n=100, 500 og 1000 og lag en liten tabell om kjøretiden som funksjon av n. Forklar hvorfor kjøretiden 8-dobles når n dobles.

Her du ser på hvordan to-dimensjonale arrayer er lagret i Java:



Da ser du at når vi beregner c[i][j], så leser vi fra A tall fra en rad som ligger etter hverandre i lageret (raskt), mens elementene fra B hentes fra hver sin rad, og vi hopper rundt i lageret mellom alle radene. Det er forventet at du vil få problemer med cache-ene, og meget langsommere lesing av B-elementene. Du skal altså prøve ut en idé at du før du multipliserer sammen A og B, så bytter du om på elementene i B, slik at kolonnene i B ligger lagret radvis som rader (på matematisk kalles dette å transponere B). Dette oppnår du hvis du bytter alle elementer B[i][j] med B[j][i] for alle i <n og j<i (grunnen til dette siste j < i er å unngå å bytte om hvert element to ganger og dermed ende opp med at B ikke er endret). Du skal altså nå forsøke å multiplisere AxB med først å transponere B. Kall den B’, og da er :

$$c[i][j] = \sum_{k=0}^{n} a[i][k] * b'[j][k]$$

Ta også tider her (hvor tidene nå inkluderer først transponering av B og så multiplisering AxB’ og se hvor mye får raskere tider enn rett fram multiplisering som definert i pkt a).

Parallell implementasjon – nå radvis

Dette er ett av de problemene hvor parallellisering er meget lett, en av de pinlig parallelle problemene. La bare tråd 0 beregne de n/k første ***radene*** i C, tråd 1 de neste n/k radene,,…, og den siste tråden alle de siste radene i C.

Siden dette ikke består i å skrive samtidig på felles variable, bare lese A og B (eller B’) , trenger vi ikke noen synkronisering, bare en skikkelig avslutning av trådene – for eksempel med join().

Inkluder din parallelle løsning med den beste av løsningene ovenfor og parallelliser denne. Siden løsningen med transponering var raskest, er det tilsvarende enkelt å parallellisere transponeringen av B. Begrunn hvorfor en transponering tar mye kortere tid enn selve multipliseringen. Det holder derfor bare i begge tilfellene ikke multipliserer transponeringen. Skriv en liten tabell over kjøretidene og speedup for din parallelle løsning og kommenter den. ***Kommenter også ved å sammenligne resultatene fra Uke3 med disse resultatene:*** Hva var best: Dele opp C radvis (uke4) eller kolonnevis(uke3)? Forklar forskjellene du fant.

Tips

Koden for sekvensiell beregning av C = AxB:

```
double  elem ;

for (int i=0;i < n;i++) {
  // for hver rad  i C
        for (int j=0;j < n;j++) {
                // for hver kolonne i C
                elem =0.0;
                for (int k=0;k < n;k++){
                        elem += a[i][k]*b[k][j];
                }
               c[i][j] = elem;
        } // end j
} // end i
```

Ukeoppgaver uke 3 INF2440 – v2014

Denne uka skal vi se på det å multiplisere to nxn matriser A og B sammen og regne ut resultatmatrisen C, altså C = AxB , først sekvensielt (koden for en rett fram implementasjon av den sekvensielle versjonen finner du nederst i tips) og så parallelliser dette. Dette kan se ut som en veldig matematisk oppgave, og matriser brukes mye i løsning av store ingeniør-beregninger, men for å løse denne oppgaven trenger men ikke vite hva matriser brukes til. Det holder å vite at en matrise er simpelthen en todimensjonal double-array (eks. double [][] a = new double[n][n]; hvor n er et passe heltall – i vårt tilfelle mindre enn 1000).

Resultatet av AxB er selv en nxn matrise C, og vi har at element c[i][j] i matrisen C er da definert som:

$$\mathrm{c[i][j]} = \sum_{k=0}^{n} \mathrm{a[i][k]} * \mathrm{b[k][j]}$$

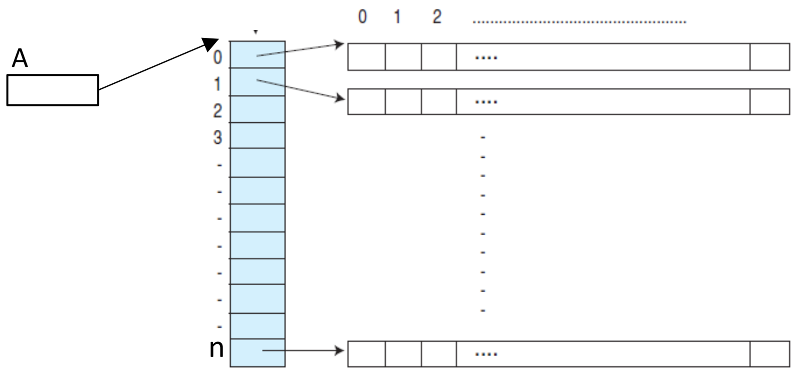
Vi multipliserer altså hvert element i rad i fra A parvis med hvert element fra kolonne j fra B og summerer de n muliplikasjonene for å få c[i][j] (se koden).

Sekvensiell implementasjon

a) Du skal før initiere matrisene A og b med tilfeldige double-tall fra klassen Random med metoden nextDouble(). Du får da et tilfeldig tall mellom 0,0 og 1.0. (Hvis du først vil teste koden din, kan du initiere A og B med 1.0, og da vil alle elementene i C være n.)

   Test kjøretiden for n=100, 500 og 1000 og lag en liten tabell om kjøretiden som funksjon av n. Forklar hvorfor kjøretiden 8-dobles når n dobles.

Her du ser på hvordan to-dimensjonale arrayer er lagret i Java:



Da ser du at når vi beregner c[i][j], så leser vi fra A tall fra en rad som ligger etter hverandre i lageret (raskt), mens elementene fra B hentes fra hver sin rad, og vi hopper rundt i lageret. Det er forventet at du vil få problemer med cache-ene, og da langsommere lesing av B-elementene. Du skal altså prøve ut en ide at før du multipliserer sammen A og B, så bytter du om på elementene i B, slik at kolonnene i B ligger lagret radvis som rader (på matematisk kalles dette å transponere B). Dette oppnår du hvis du bytter alle elementer B[i][j] med B[j][i] for alle i <n og j<i (grunnen til dette siste j < i er å unngå å bytte om hvert element to ganger og dermed ende opp med at B ikke er endret) Du skal altså nå forsøke å multiplisere AxB med først å transponere B. Kall den B', og da er :

$$\mathrm{c[i][j]} = \sum_{k=0}^{n} \mathrm{a[i][k]} * \mathrm{b'[j][k]}$$

Ta også tider her (hvor tidene nå inkluderer først transponering av B og så multiplisering AxB' og se om du får raskere tider enn rett fram multiplisering som definert i pkt a).

Parallell implementasjon

Dette er ett av de problemene hvor parallellisering er meget lett, en av de pinbart parallelle problemene. La bare tråd 0 beregne de n/k første kolonnene i C, tråd 1 de neste n/k trådene,,..., og den siste tråden alle de siste kolonnene i C.

Siden dette ikke består i å skrive samtidig på felles variable, bare lese A og B (eller B') , trenger vi ikke noen synkronisering, bare en skikkelig avslutning av trådene – for eksempel med join().

Inkluder din parallelle løsning med den beste av løsningene a) eller b) ovenfor og parallelliser denne. Hvis løsning b) med transponering var raskest, er det tilsvarende enkelt å parallellisere transponeringen av B. Skriv en liten tabell over kjøretidene og speedup for din parallelle løsning og kommenter den.

Tips

Koden for beregning av C = AxB:

```
double  elem ;

for (int i=0;i < n;i++) {
  // for each row i C
         for (int j=0;j < n;j++) {
                 // for each collumn in C
                 elem =0.0;
                 for (int k=0;k < n;k++){
                         elem += a[i][k]*b[k][j];
                 }
                c[i][j] = elem;
         } // end j
} // end i
```