Obligatorisk oppgave 1: Regneklynge

INF1010

Frist: mandag 6. februar 2017 kl. 12:00

Versjon 1.0 (`aacecc0`)

Innhold

1 Innledning

I denne oppgaven skal du lage et program for å holde oversikt over alle komponentene i en regneklynge (eng: computer cluster). En slik regneklynge kan brukes til å fordele tunge beregninger på mange maskiner slik at de kan jobbe i parallell. På den måten kan en simulering som ville tatt en måned å kjøre på en vanlig maskin, kjøres på regneklyngen på noen timer i stedet. Det finnes flere regneklynger på UiO, hvorav Abel er den største.

Figur 1: Et rack fra Abel. Foto: Benjamin Sørlie Ormset

2 Regneklyngens bestanddeler

En regneklynge består av ett eller flere *rack* (et kabinett med skinner) hvor mange *noder* kan monteres over hverandre. En *node* er en selvstendig maskin med et hovedkort med én eller flere prosessorer og minne, i tillegg til en del andre ting. I denne oppgaven skal vi bare se på prosessoren(e) og minnet. Du kan derfor anta at en node består av inntil to prosessorer og har et heltallig antall GB med minne. For prosessoren trenger vi bare å vite hvor mange kjerner den har og hvilken klokkehastighet den maksimalt kan kjøre med. Klokkehastigheten er antall klokkesykler per sekund. Dette kalles også frekvensen til prosessoren og måles i hertz (Hz). Vi skal anta at alle kjernene i prosessoren kjører ved samme klokkehastighet selv om moderne prosessorer kan regulere klokkehastigheten til hver kjerne individuelt.

Skriv klassene `Regneklynge`, `Rack`, `Node` og `Prosessor`. Siden vi bare trenger å vite hvor mange GB minne en node har, trenger vi ikke en egen klasse for å representere minnet.

Klassen `Regneklynge` skal ha en `ArrayList<Rack>` og en metode `settInnNode(Node node)`. Denne metoden skal sette inn noden inn i et ledig `Rack`-objekt fra listen hvis det er plass. Hvis alle rackene er fulle, skal det lages et nytt `Rack`-objekt som legges inn i listen, og noden plasseres i det nye racket. For enkelhets skyld skal vi anta at alle noder tar opp like stor plass i racket, og dermed har alle rackene i regneklyngen plass til like mange noder.

Tips. Det kan være lurt å ta inn antall noder per rack i konstruktøren til `Regneklynge`.

Klassen `Rack` skal lagre `Node`-objektene i et array. Vi skal kunne legge til noder i racket.

Tips. Det kan være lurt å la `Node` ha to konstruktører som i tillegg til minnestørrelsen tar inn klokkehastigheten og antall kjerner for både én og to prosessorer, så kan konstruktørene til `Node` opprette `Prosessor`-objektene istedenfor at de sendes inn som parametre. Dette vil gjøre det enklere for oss å skrive et kort hovedprogram senere.

3 Datastrukturtegning

Tegn datastrukturen slik den ville sett ut etter at vi har satt inn følgende noder i et nytt objekt av `Regneklynge`. La antall noder per rack være 2.

- Node 1: 16 GB minne, 1 prosessor med 4 kjerner og en klokkehastighet på 3.2 GHz
- Node 2: 16 GB minne, 1 prosessor med 4 kjerner og en klokkehastighet på 3.2 GHz
- Node 3: 128 GB minne, 2 prosessorer med 16 kjerner hver og en klokkehastighet på 2.1 GHz

Merk.

Klasser vi ikke kjenner implementasjonen til, som f.eks. `ArrayList`, kan tegnes som en slags black box. Se notatet om datastrukturtegning hvis du er usikker på hvordan noe skal tegnes.

4 Maksimal teoretisk ytelse

Til sammen vil en regneklynge ha en samlet regnekapasitet som vanligvis måles i hvor mange operasjoner på (64-bits) flyttall vi kan gjøre på alle nodene per sekund. Dette målet kalles FLOPS (FLyttallsOPerasjoner per Sekund). Vi kan beregne hvor mange FLOPS en prosessor yter ved å ta produktet av antall kjerner, klokkehastigheten til kjernene (vi har allerede antatt at alle kjernene kjører på samme klokkehastighet) og antall flyttallsoperasjoner vi kan gjøre per klokkesykel. Du kan la antall flyttallsoperasjoner per klokkesykel være 8 for alle prosessorer (moderne prosessorer vil typisk klare å gjøre 4 multiplikasjoner og 4 addisjoner samtidig). Formelen for å regne ut maksimal teoretisk ytelse for én node blir altså

$$\text{FLOPS} = \text{antall kjerner} \cdot \text{klokkehastighet} \cdot \text{flyttallsoperasjoner per klokkesykel} \tag{1}$$

Vi antar at regnekapasiteten for hele regneklyngen er summen av regnekapasiteten til alle nodene.

Lag en metode `flops()` i `Regneklynge` som gir den samlede regnekapasiteten til regneklyngen.

5 Minnekrav

Noen programmer trenger mye minne. Typisk vil vi trenge et gitt antall GB med minne på hver node vi bruker. Vi er derfor interessert i å vite hvor mange noder som har nok minne til at vi kan bruke dem. Lag en metode `noderMedNokMinne(int paakrevdMinne)` som returnerer antall noder med minst `paakrevdMinne` GB minne.

6 Hovedprogram

Lag et hovedprogram for å teste at klassene virker som de skal. Lag en regneklynge, `abel`, og la det være plass til 12 noder i hvert rack. Legg inn 650 noder med 64 GB minne og 2 prosessorer med 8 kjerner hver og en klokkehastighet på 2.6 GHz (1 GHz = 10^9 Hz) på begge prosessorene. Legg også inn 16 noder med 1024 GB minne og to prosessorer med 8 kjerner hver og en klokkehastighet på 2.3 GHz på begge prosessorene.

Finn regneklyngens maksimale teoretiske ytelse og sjekk hvor mange noder som har minst 32 GB, 64 GB og 128 GB minne. Sjekk også hvor mange rack som brukes. Skriv ut svarene i terminalen. Utskriften kan f.eks. se slik ut:

```
Samlet FLOPS: 2.21030e+14
Noder med minst 32 GB: 666
Noder med minst 64 GB: 666
Noder med minst 128 GB: 16
Antall rack: 56
```

6.1 Ekstraoppgave

Hvis du vil, kan du også legge inn støtte for å lese fra fil. Filformatet vil da se slik ut for eksemplet over:

```
# Noder per rack
12
# Antall Minne P1_Kjerner P1_Klok. [P2_Kjerner P2_Klok.]
650 64 8 2.6E9 8 2.6E9
16 1024 8 2.3E9 8 2.3E9
```

Lykke til!
Stein Gjessing, Stein Michael og Kristian