Øving 2
TDT4200

Anette Fossum Morken

Del 1, oppgave 1

a

skrives om lockblock?? pipeline minnehierarki hetero/homo? numa/cluster?

i)

Nivida Maxwell Nvidia Maxwell har homogene kjerner med tråder og er NUMA .

ii)

ARM big.LITTLE er en samling av to forskjelige prosessorer. Den ene prosessoren er energikrevende og den andre er lite energikrevende slik at når enkle oppgaver, som tekstbehandling, skal gjøres gjøres det på den mindre energikrevende prosessoren og når tyngre oppgaver skal gjøres gjøres de på den mer energikrevende prosessoren. Siden den tydelig består av to forskjellige prosessorer har denne heterogenious kjerner

iii)

vilje@NTNU er en superdatamaskin som består av prosessorere i kluster. Hver prosessor igjen bastår av et vist antall tråder. Prosessorene og trådene er organisert slik at tingangen til alle trådene ikke er lik og dermed er vilje NUMA.

iv)

En vanlig CPU i dag består av flere like kjerner hvor hver kjerne består av et vist antall tråder som kan brukes samtidig.

b

SIMT passer inn i Flynns taksonomi i SIMD, der det på Nivida er en instruksjon flere tråder vil det i Flynns taksonomi bli en instruksjon multiple data.

c

i)

Nivida Maxwell passer inn i Flynns både som SIMD og MIMD siden den har mulighet til å gjøre plere ting på en gang.

ii)

ARM big.LITTLE blir i dag mest sett på som SIMD, men siden den består av to forkjellige prosessorere kan den i teorien gjøre to forskjellige oppgaver samtidig.

iii)

vilje@NTNU MIMD

iv)

En vanlig CPU SIMD

Del 1, oppgave 2

a

Tråder er organisert i blokker, bokker er organisert i gridd. Blokkene er helt uangengige av hverandre og kan ikke kommunisere, mens trådene i en blokk kan komunisere med hverandre.

b

Setter opp ligninger for tiden GPUen vil bruke og CPUen vil bruke:

$$\begin{aligned} GPU &= \frac{2n}{r} + \frac{5n}{r} + 5h_{GPU}n7h_{GPU}log_2(n) \\ CPU &= 5h_{CPU}n7h_{CPU}log_2(n) \end{aligned} \tag{1}$$

der $h_{GPU} = 1$, $h_{CPU} = 10$, r er båndbredden og n er mengde data som overføres. Det lønner seg å bruke GPUen når $GPU > CPU$ det finnes ved å regne ut GPU og CPU for mange forskjellige n og plotte dem og se hvor de krysser hverandre slik at $GPU > CPU$. Med den oppgitte algoritmen vil det ta kortest til å kjøre på GPUen hvis $n \geqslant 1$, dette kan sees ved å studere plottet i figur 1. For de fleste andre programmer er ikke dette riktig. Vanligvis vil det være raskest å kjøre programmer der n er liten på CPUen siden minneoverføringen til og fra GPUen er tidkrevende.

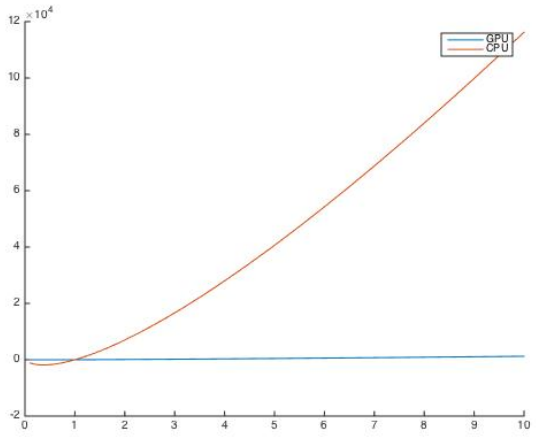

Figure 1: Plottet viser kjøretiden for et program gitt med kjøretidene i (1)

c

Kernel2 vil være raskest siden her blir alle trådene satt i arbeid. I kernel1 looper man gjennom trådene og gir bare et utvalg av trådene arbeid. Hvis trådene i en warp har fått forskjellige instruksjoner vil de bli kjørt serielt og siden alle trådene ikke har samme instruksjon i kernel1 vil denne kjøres serielt. I kernel2 får alle trådene i hver blokk samme instruksjon og vil derfor kjøres parallelt.

d

i)

Warps er samlinger med tråder. Hvis trådene utfører samme operasjon kan de kjøres paralellt, men har de forskjellige oppgaver må de kjøres serielt.

ii)

Occupancy gir hvor stor andel av tilgjengelige warps som er i bruk. Man kan se også se på det som hvor mange av trådene som står og venter på å få arbeid.

iii)

Minne fortetting(coalescing) er å slå sammen flere minneoverføringer mellom minet og tråder til en overføring. Det gjør at et gitt antall tråder i en warp kan hente ut informasjon fra minnet på en gang og tiden det tart for alle å hente ut informasjonen er samme som en trå hadde brukt om man ikke benyttet seg av coalescing. På denne måten kan man redusere kjøretiden til programmet.

iv)

Lokalt minne er minne som en tråd kun har tilgang til skal en tråd ha informasjon fra en annen må dette deles gjennom message passing.

v)

Delt minne er at alle trådene i en blokk har tilgang til informasjonen som ligger her.

Del 2, oppgave 1

c

TIden ble tatt på forskjellige steder i koden, hele programmet, minneoverforingen fra CPUen til GPUen og for minneoverføringen CPU-GPU og tilbake og invertringen av bildet. Hele programmet bruker rundt $0,19 \cdot 10^9$ns, minneoverføringen CPU-GPU og tilbake og invertringen av bildet bruker rundt $0,63 \cdot 10^6$ns og minneoverføringen bruker $0,23 \cdot 10^6$ns

antall prosent tiden brukt til minneoverføringen er av hele kjøretiden: antall prosent tiden brukt til minneoverføringen er av selve oppaven til programmet:

DEt første man kan merke seg at I/O tar mye tid, så monneoverføringen er ikke den største tidstyven i dette programmet, men når man ser på CPU-GPU og tilbake og invertringen av bildet ser man at minneoverføring er en prosess som tar tid.