Kap. 3: Prinsipper for skalerbar ytelse

- **Mål for ytelse**
  - parallellitets-grad
  - parallellitets-profil
  - aritmetisk, geometrisk og harmonisk midlere ytelse
  - effektivitet, redundans, utnyttelse og kvalitet
  - "benchmarks"
  - MIPS, Mflops, TPS, KLIPS
- **Applikasjoner for parallelle maskiner**
  - "Grand Challenges"
  - hovedklasser av applikasjoner
- **Lover for hastighetsøkning**
  - Amdahl's lov
  - Gustafson's lov for skalerte problemer
  - Sun og Ni's lov
- **Skalerbarhet**
  - skalerbarhets-mål
  - ulike typer skalerbarhet

Parallellitets-grad

- **Parallellitets-grad**
  - antall prosessorer som benyttes i utføring av en pågående del av en beregning
  - varierer med tiden
  - Eng. Degree of parallelism (DOP)
  - *DOP(t)*
- **Parallellitets-profil**
  - parallellitetsgrad plottet som funksjon av tid
  - eks. fig. 3.1
    - » antar et antall homogene prosessorer, det totale arbeid er da lik arealet under parallellitets-profilen
- **Gjennomsnittlig parallellitet**
  - Arbeid / tid brukt

$$A = \frac{1}{t_2 - t_1}\int_{t_1}^{t_2} DOP(t)dt \qquad (3.3)$$

Aritmetisk, geometrisk og harmonisk midlere ytelse

- **Kilde: James E. Smith, *Characterizing computer performance with a single number,* CACM okt. 88, pp. 1202-1206**
- **Midlere ytelse**
  - Aritmetisk midlere ytelse, R_a
  - Geometrisk midlere ytelse, R_g
  - Harmonisk midlere ytelse, R_h
- **Antakelser**
  - en maskin som utfører *m* program med ulik ytelse
  - lik vekt (sannsynlighet) for alle programmer
    - » ("veiet" midlere ytelse, se i boka s. 109-110)

R_i = kjørehastighet for program i

$$R_a = \sum_{i=1}^{m} R_i / m$$

$$R_g = \prod_{i=1}^{m} R_i^{(1/m)}$$

$$R_h = \frac{m}{\sum_{(i=1)}^{(m)} (1 / R_i)}$$

Aritmetisk, geometrisk og harmonisk midlere ytelse, **eksempel**

- Kilde: James E. Smith, CACM okt. 88, pp. 1202-1206

Ytelse av tre maskiner på to u like program:

Benchmark	# Mill. flops i programmet	Maskin1	Maskin2	Maskin3
Program - 1	100	1	10	20
Program - 2	100	1000	100	20
Total tid		1001	110	40

5 ulike ytelses - mål basert på disse to programmer i Mflops:

Benchmark	Maskin1	Maskin2	Maskin3
Program - 1	100.0	10.0	5.0
Program - 2	0.1	1.0	5.0
Aritmetisk middel	50.1	5.5	5.0
Geometrisk middel	3.2	3.2	5.0
Harmonisk middel	0.2	1.8	5.0

Effektivitet, redundans, utnyttelse og kvalitet

- **Ruby Lee (1980)**
- **Parametre for evaluering av parallelle beregninger**
  - antar maskin med *n* prosessorer
  - skriver *W(n)* istf. *O(n)* (for å unngå misforståelse)
  - *W(n)* er totalt # operasjoner utført av *n*-prosessor maskin
  - *T(n)* er kjøretid for beregning på *n*-prosessor maskin
- **Hastighetsøkning**
  S(n) = T(1) / T(n)
- **Effektivitet**
  E(n) = S(n) / n
- **Redundans**
  R(n) = W(n) / W(1)
- **Utnyttelse**
  U(n) = R(n) E(n)
- **Kvalitet**
  Q(n) = [S(n) E(n)] / R(n)

Ytelses-mål, "benchmarks"

- **MIPS, Mflops,**
  - topphastighet (Eng. peak)
  - "midlere hastighet" (Eng. sustained)
  - sammenlikning av MIPS (eller Mflops) -rater for maskiner med ulike instruksjons-sett kan være misvisende
    - » **relative MIPS**, dvs. målt i forhold til en gitt maskin
    - » "benchmarks", dvs. tid brukt på et knippe av program-eksempler
  - Mflops; må spesifisere *hvilken type* flyt-tallsoperasjoner
- **Whetstones/s**
  - FORTRAN basert syntetisk benchmark
  - heltall, flyttall, matte-bibliotek, m.m. brukes
  - påvirkes av kompilator
- **Dhrystones/s**
  - CPU intensiv heltalls-benchmark, inspirert av Whetstone
  - finnes i ulike versjoner
  - påvirkes av kompilator
- **Andre kjente benchmarks**
  - SPEC, Perfect Club, m.fl.

Ytelses-mål, forts.

- **TPS**
  - transaksjoner pr. sekund
    - » **transaksjon** = logisk samling av enkelt-operasjoner mot database for å få utført en arbeidsoppgave
      - krever typisk 0.5-1.0 Millioner Instruksjoner (MI)
    - » 100 TPS er bra i dag [TPC, se under]
      - TF-Tyholt utvikler system med 1000 TPS basert på 16 stk. HP-maskiner (arbeidsstasjoner)
      - Rekord i området 2000-3000 TPS
  - administrativ databehandling
  - TPC-A, TPC-B, TPC-C benchmark
- **KLIPS**
  - Kilo Logiske Inferenser pr. sekund
  - AI og prolog-maskiner
  - Japan's FGCS prosjekt;
    - » Fifth Generation Computer System
    - » Bygget en parallell prolog-maskin med en ytelse på ca. 100 MegaLips
  - Sparc20 med Sicstus Prolog: ca. 400 KLIPS

Applikasjoner for parallelle maskiner

- **Massiv Parallell Prosessering (MPP)**
  - 1000 prosessorer el. flere
  - oppblomstring på 90-tallet
  - en rekke maskiner på vei nå (Tabell 3.5)
  - usikkerhet høsten 94
- **3T ytelse**
  - 1 TeraFlops / 1 TeraByte primær-lager
  - 1 TeraByte/s I/U-båndbredde
- **HPCC programmet**
  - High Performance Computing and Communication
  - "Grand Challenges"
  - HPCN i Europa
- **Hovedklasser av applikasjoner**
  - fast oppgavestørrelse (Eng. fixed-load)
    - » løs en gitt oppgave hurtigst mulig
  - fast løsningstid (Eng. fixed-time)
    - » løs et størst mulig problem, eller løs et problem mest mulig nøyaktig innenfor en tidsramme
  - lager-begrenset
    - » hoved-lager og disk-kapasitet setter en stopp for hvor store problemer som kan løses
  - kommunikasjons-begrenset
    - » IPC- og IO-begrensning

Lover for hastighetsøkning, bakgrunn

- **Amdahl's lov**
  - fast problemstørrelse
  - hvilken hastighetsøkning kan oppnås ved å bruke flere prosessorer?
  - beregningen består av en seriell og en parallell del
  - den parallelle delen kan fordeles likt på *n* prosessorer
  - den serielle delen er fast, og begrenser sterkt den hastighetsøkning en kan oppnå ved parallell prosessering
  - pessimistisk syn for de fleste problemer
- **Gustafson's lov**
  - utvider problemet sammen med maskinen
  - fast utførelses-tid
  - antar seriell del og parallell del som for Amdahl's lov
  - problemet skaleres ved å utvide den parallell delen
  - realistisk syn for mange problemer
- **Sun og Ni**
  - Generalisering av Amdahl's lov og Gustafson's lov
  - Når maskinen blir større, skaler problemet så mye som mulig innenfor tilgjengelig lagerplass
  - Økt kjøretid kan være akseptabelt når beregningen utføres nøyaktigere/"bedre"
- **Kap. 3.2.3 utgår**

Lover for hastighetsøkning - 1 (1 av 2)
Amdahl's lov

S_∞ er asymptotisk hastighetsøkning

W_i = mengden av arbeid som utføres med i prosessorer

$$S_\infty = \frac{T(1)}{T(\infty)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} W_i / i} \qquad (3.7)$$

m = max . parallellitetsgrad

her $S_\infty = S_m$, $W_k = 0$, $k = m+1, \ldots$

Antar at alt arbeid utføres enten på en prosessor , eller på n prosessorer

$$S_n = \frac{T(1)}{T(n)} = \frac{W_1 + W_n}{W_1 + W_n / n}$$

Lover for hastighetsøkning - 1 (2 av 2)
Amdahl's lov

$$S_n = \frac{T(1)}{T(n)} = \frac{W_1 + W_n}{W_1 + W_n / n}$$

Anta en normalisert situasjon der

$W_1 + W_n = \alpha + (1-\alpha) = 1$ med

$W_1 = \alpha$ gir det

$$S_n = \frac{T(1)}{T(n)} = \frac{1}{\alpha + (1-\alpha)/n} = \frac{n}{n\alpha + 1 - \alpha} = \frac{n}{1 + (n-1)\alpha} \qquad (3.16)$$

$$S_n = \frac{n}{1 + (n-1)\alpha}$$ er kjent som Amdahls lov

Lover for hastighetsøkning - 2 (1 av 2)
Skalerbart problem
Gustafson's lov

W'_i er mengden av arbeid som utføres med i prosessorer i det skalerte problemet

m' er maksimal parallellitetsgrad i det nye problemet

$T(1) = T'(n)$ fordi vi antar konstant tid skalering

$W_1 = W'_1$ fordi vi antar at det serielle arbeidet ikke øker

$$S'_n = \frac{T'(1)}{T'(n)} = \frac{T'(1)}{T(1)} = \frac{\sum_{i=1}^{m'} W'_i}{\sum_{i=1}^{m} W_i} \qquad ("3.31")$$

Antar at alt arbeid utføres enten på en prosessor eller på n prosessorer

$W_1 + W_n = W'_1 + W'_n / n$ pga konstant tid

$W'_n = nW_n$ siden det serielle arbeidet forblir uendret

Det gir:

$$S'_n = \frac{\sum_{i=1}^{m'} W'_i}{\sum_{i=1}^{m} W_i} = \frac{W'_1 + W'_n}{W_1 + W_n} = \frac{W_1 + nW_n}{W_1 + W_n} \qquad (3.32)$$

Lover for hastighetsøkning - 2 (2 av 2)
Skalerbart problem
Gustafson's lov

Vi har:

$$S'_n = \frac{W_1 + nW_n}{W_1 + W_n} \qquad (3.32)$$

Anta en normalisert situasjon der

$W_1 + W_n = \alpha + (1-\alpha) = 1$ med

$W_1 = \alpha$ det gir

$$S'_n = \frac{W_1 + nW_n}{W_1 + W_n} = \frac{\alpha + n(1-\alpha)}{\alpha + (1-\alpha)} = n - \alpha(n-1) \qquad (3.33)$$

$S'_n = n - \alpha(n-1)$ er kjent som Gustafsons lov

Lover for hastighetsøkning - 3
Lagerplass -begrenset problem,
Sun & Ni's lov

- **S^*_n er hastighetsøkning når vi går fra 1 til *n* prosessorer, og antar at problemet skaleres maksimalt innenfor rammen av tilgjengelig plass**
- **Hvis *M* er lagerplass behovet til et gitt problem vil arbeidsmengden knyttet til løsning av problemet, *W*, være relatert til *M***
- **Anta at det serielle arbeidet forblir uendret mens det parallelle arbeidet i det skalerte problemet kan uttrykkes som *G(n)W*$_n$**

Vi har da

$$W^*_n = G(n)W_n$$

$$S^*_n = \frac{W^*_1 + W^*_n}{W^*_1 + W^*_n / n} = \frac{W_1 + G(n)W_n}{W_1 + G(n)W_n / n}$$

Formelen til Sun og Ni kan betraktes for tre hovedtilfeller;

Sun & Ni's lov:

$$W^*_n = G(n)W_n$$

$$S^*_n = \frac{W_1 + G(n)W_n}{W_1 + G(n)W_n / n} \qquad (3.35)$$

- ***G(n) = 1***
  - Problemstørrelsen er fast, dvs. Amdahl's lov
- ***G(n) = n***
  - Arbeidsmengden øker n ganger når lagerplassen øker n ganger, dvs. Gustafson's lov
- ***G(n) > n***
  - Arbeidsmengden øker raskere enn lagerplassen. Vi har da lagerplass til å la arbeidsmengden øke mer enn maskinstørrelsen n, og kan da få en hastighetsøkning som er større enn ved Gustafson's lov

Skalerbarhet

- **Nytt begrep, moteord, mange typer og tolkninger**
- **Den enkleste og vanligste definisjon: En maskins ytelse øker lineært med antall prosessorer**
- **Arkitektur-algoritme-par**
  - Skalerbarhet avhenger av (maskin)-arkitektur og hvordan denne passer til en gitt algoritme ("applikasjons-arkitektur")
- **En arkitektur kan gi god skalerbarhet for en algoritme men dårlig for en annen.**
  - Eks.; for noen beregninger er nettverks-diameter det kritiske, for andre kan det være kommunikasjons-kapasitet

Skalerbarhet, hastighetsøkning og effektivitet

$s =$ problemstørrelse

$n =$ antall prosessorer

$h(s,n) =$ kommunikasjon og IUoverhead

$$S(s,n) = \frac{T(s,1)}{T(s,n) + h(s,n)}$$

$$E(s,n) = \frac{S(s,n)}{n}$$

- **Et system er skalerbart hvis E(s,n) = 1, for alle algoritmer, alle n, og alle s ..... UREALISTISK !**
- **Skalerbarheten til en gitt maskin for en gitt algoritme er definert som forholdet mellom den asymptotiske hastighetsøkning på den gitte virkelige maskin og den asymptotiske hastighetsøkning på en ideel maskin (EREW PRAM). (Nussbaum og Agarwal)**

$$\Phi(s,n) = \frac{S(s,n)}{S_I(s,n)} = \frac{T(s,1)/T(s,n)}{T_I(s,1)/T_I(s,n)} = \frac{T_I(s,n)}{T(s,n)}$$

- **ordens-uttrykk, kan være misvisende, spesielt for små verdier**

Gordon Bell's tre hovedtyper av skalerbarhet

- **Størrelses-skalerbarhet**
  - det vi i hovedsak har behandlet over
  - alle komponenter må utvides i takt med økte ressurskrav
  - lineær sammenheng mellom kostnader og ytelse
    - » sjeldent mulig
  - Skalerbarhetsområde, eksempler:
    - » Cray Y-MP: 1--16
    - » CM-5: 1024--16K prosessorer (maskiner)
    - » KSR-1: 8--1088
- **Teknologi-skalerbarhet (Generasjons (tid) -skalerbarhet)**
  - Arkitekturen bør gjøre det mulig å ta i bruk ny teknologi for prosessorer, lager osv. etterhvert som den blir tilgjengelig
  - Generasjoner av mikroprosessorer (3 år)
  - Viktig å skille mellom implementasjon og arkitektur (F.eks. DEC's Alpha prosjekt påpeker dette)
  - Nye maskiner (og varianter) som bruker ny og mer effektiv teknologi må ikke føre til at (for mye) programvare må kastes
- **Problem-skalerbarhet**
  - Diskutert tidligere
  - Noen anvendelser er "perfekt parallelle", dvs. består av stort antall uavhengige beregninger

"Balansert ytelse i multimaskiner"

- **Problem-granularitet**
  - ⇨"antall operasjoner i ett "rutenett-punkt"" dividert med "nødvendig datamengde fra nabo-punkter"
- **Maskin-granularitet**
  - ⇨Operasjonshastighet i node dividert med kommunikasjonshastighet mellom nabo-noder
- **Jo lavere maskin-granularitet jo mer finkornet kan vi parallellisere et problem**
  - uten å miste effektiv prosessor-utnyttelse
  - forutsatt at maskinen er stor nok
  - Eks.
    - » CM5: 102 operasjoner/ord overført
    - » Paragon: 3 operasjoner/ord overført
- **eks. 3.8**

Realisering av gode MPP systemer, sentrale forskningstemaer

- **Problemer (når antall prosessorer øker)**
  - økt lager-aksess tid, økt ikke-uniformitet
  - økt IPC og synkroniserings-kompleksitet
  - økte cache konsistens problemer
  - svekket prosessor-utnyttelse
  - vanskeligere ressurs-deling
- **Noen nyere løsningsteknikker:**
  - større og forbedret lagerhierarki
  - nye synkroniseringsteknikker og teknologi
  - svakere krav til konsistens i lageret
  - nye (distribuerte) teknikker for cache-koherens
  - delt virtuelt lager (SVM)
  - "multithread" arkitekturer
  - programvare portabilitet og standardisering

Diverse

- **S(n) = T(1)/T(n), hva er T(1) ?**
  - For alle hastighetsmål er det alltid riktig og viktig å benytte den best kjente sekvensielle algoritme, ikke den aktuelle parallelle algoritme kjørt på en prosessor
- **Lokalitet i nettverk**
  - Et nettverk utnytter lokalitet hvis kommunikasjon til nære prosessorer går hurtigere enn gjennomsnittlig kommunikasjon. Eks. hyperkube, ikke Omega-nettverk.
- **Den ideelle parallelle maskin**
  - lett å programmere (delt lager)
  - skalerbar (distribuert lager)
  - Fig 3.12 b)
  - skill mellom hva programmereren ser og implementasjon
  - stor forskningsaktivitet
    - » Delt virtuelt lager
    - » Valiant's BSP-modell (?)