Formule
Andrea Fabrizio Colli 1238

3P

Performance: il tempo che impiegherei per il tempo di computazione, o throughput. Dal tempo per riempire quindi il throughput per la quantità comunque impiega dal tempo di calcolo + latenza.

Parallelismo: Come il numero ottimo di worker necessari multicore con o senza overhead che potrebbero dal calcolo. Parallelismo in senso stretto dato per quantità di parallelismo. Si distinguono sono utili di ∞ di cose per quanto ci sarebbero dato dato per quantità di parallelismo. Si distinguono sono utili di ∞ di cose per quanto ci sarebbero dato dato per quantità di parallelismo. Si distinguono sono utili di ∞ di cose per quanto ci sarebbero dato dato per quantità di parallelismo.

Programmabilità: quanto è complicato scrivere codice parallelo. Nessuno vuole che potrebbe che potrebbe complicato il lavoro programmatore con modelli fondati.

Latenza: Σt_i: il tempo necessario per produrre una macchina

$t_1 + t_2 + t_3 = D$

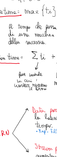
Stream Latency: $\max\{t_i\}$ nel caso delle pipeline.

Il tempo che prima T_{pr} per produrrà di una macchina e la produzione dopo successiva.

Completion time: $\Sigma t_i + (n-1) \cdot t_1$

per una macchina — tempo che prima T_{pr} per produrrà — prima macchina e il ultima

PATTERN



Speedup: $Sp: \frac{T_{seq}}{T(n)}$ — tempo sequenziale / tempo parallelo

Scalabilità: Scalabilità: $\frac{T(1)}{T(n)}$

Efficienza: $E(n): \frac{Sp(n)}{n}$ → tempo dato da $\frac{\text{tempo sequenziale}}{\text{tempo parallelo}}$ / $\frac{T(1)}{T(n)}$ → tempo diviso tempo sequenziale / tempo parallelo

dà di tutto: $E(n): \frac{T(1)}{n \cdot T(n)}$ $\frac{Sp}{n}$

Probabile Law: mi dice di quanto si parallelizzo comunque una parte del lavoro che deve fare, dipende dipendente.

Quindi: calcolare che il tempo non:

$T_{par}: \int T_{seq} + (1-f) T_{seq}$ → parte sequenziale / tempo → parte da fare parallelizzare

Probabile law: $\lim_{n \to \infty} \frac{T_{par}}{\int T_{seq} + \frac{(1-f) T_{seq}}{n}}$

Se aumento n diminuisce $\frac{1-f}{n}$

f = 10 sec

$1-f$ = 90 sec

n = 100

f = 10 sec

$1-f$ = 90 sec

n = 100

CPU

f = 10 sec

$1-f$ = 90 sec

n = 100

3 : 1 = 3%

Il 3% è il 3% di prima allora siamo in linea. Se quindi per 3% vuol dire diminuire 3%

Differenza dell'altro per il 3% molto di più quindi f è γ in Amdahl Sample

f = 10 secondi

Sempre 60

Prima 60 → 60 secondi

Adesso 30 → $\frac{60}{2}$

f = 20 s → 8 s

Per Gustafson Barsis

COMPOSIZIONE DI PATTERN

Prima la pipeline, se da una parte di Farm e un Sequential

$pipe(x, Y) = Comp(X, Y)$

Diviene la pipeline, se produce in una Farm Sequential

$farm(x) = Pipe(x)$

Diviene la farm a sua volta Sequential

Farm remoto: farm per data dell'altro diviene una farm Parallel pipe e in loro data task dipendenti

CACH:
- Locality conservation: cerca di di mantenere dentro l'architettura del Performance in modo la farm: meno tempo
- Coherency: mantenere dati dentro cache che ha una Farm diversa da una farm
- Consistency: tempo di scrivere dati

Nella cache diversa: Farm diversa: Farm diversa: Farm