# Calcolo Parallelo e Distribuito

I parametri di valutazione di un algoritmo parallelo:
legge di Ware-Amdhal

Docente: Prof. L. Marcellino

Tutor: Prof. P. De Luca

# Per gli algoritmi sequenziali

L'efficienza dipende dalla complessità computazionale

$T(N)$ = numero operazioni

# Tempo di esecuzione di un software

$T_1(N)$

complessità computazionale 1 core

$$\tau_1 = k \cdot T_1(N) \cdot \mu$$

$\mu$ = tempo di esecuzione di 1 op. f.p.

Tempo di esecuzione software parallelo

$$\tau_p = T_p(N) \cdot K \cdot \mu$$

$T_p(N)$

complessità computazionale p core

N dimensione del problema iniziale

# Per gli algoritmi paralleli

## Speed-up

Si definisce il rapporto $T_1$ su $T_p$

$$S_p = \frac{T_1}{T_p}$$

Lo speed up misura la riduzione del tempo di esecuzione rispetto all'algoritmo su 1 core

$$S_p < p$$

SPEEDUP IDEALE

$$S_p^{ideale} = p$$

# OVERHEAD totale

$$S_p^{ideale} = \frac{T_1}{T_p} = p$$

$$\boxed{O_h = (pT_p - T_1)} \longrightarrow T_p = (O_h + T_1)/p$$

$$S_p = \frac{T_1}{T_p} = \frac{T_1}{(O_h + T_1)/p} = \frac{pT_1}{O_h + T_1} = \frac{p}{\frac{O_h}{T_1} + 1}$$

L'OVERHEAD totale misura

quanto lo speed up differisce da quello ideale

# Efficienza

Si definisce il rapporto $E_p$ su p

$$E_p = \frac{S_p}{p}$$

misura quanto l'algoritmo sfrutta il parallelismo del calcolatore

**EFFICIENZA IDEALE**

$$E_p^{ideale} = \frac{S_p^{ideale}}{p} = 1$$

# Domanda

Lo speed-up $S_p = T_1 / T_p$ misura la riduzione del tempo di

esecuzione dell'algoritmo sequenziale rispetto al tempo di esecuzione dell' algoritmo parallelo

**Quale algoritmo scegliere per misurare $T_1$?**

# PRIMA SCELTA

$T_{p=1}$ = tempo di **esecuzione** dell'algoritmo parallelo **su 1 processore**

⇓

$S_p$ dà informazioni su quanto l'algoritmo **si presta all'implementazione** su un'architettura parallela

Svantaggio:

l'algoritmo parallelo su 1 processore potrebbe eseguire più operazioni del necessario

# SECONDA SCELTA

$T_1$ = tempo di esecuzione del migliore algoritmo sequenziale

⇓

$S_p$ dà informazioni sulla
riduzione effettiva del tempo
nella risoluzione di un problema con p processori

Difficoltà:

- individuazione del "miglior" algoritmo sequenziale
- disponibilità di software che implementi tale algoritmo

# Convenzione: prima scelta

$T_p$ = tempo di esecuzione dell'algoritmo parallelo su 1 processore

⇓

$S_p$ e $E_p$ danno informazioni su quanto l'algoritmo si presta all'implementazione su un'architettura parallela

# Domanda

Quando è possibile ottenere

speed-up prossimi allo speed-up ideale ?

- algoritmi full parallel: SEMPRE!
- algoritmi con collezione dei risultati

    (REDUCTION): ALL'INFINITO(?)

    mi posso accontentare un po' prima?

# Analisi asintotica dello speedup

Ripartiamo dall'inizio...

Il punto di partenza per scrivere un buon algoritmo parallelo è aver scritto un ottimo algoritmo sequenziale

ATTENZIONE:
differenze fra algoritmo – codice – software?

# Complessità computazionale $T_1$ si può decomporre in 2 parti:

una parte relativa alle operazioni che devono essere eseguite esclusivamente in sequenziale

una parte relativa alle operazioni che potrebbero essere eseguite concorrentemente

$T_s$ $T_c$

$$T_1 = T_s + T_c$$

# Esempio:

**Somma di 2 vettori di dimensione n=4**, $T_1(4)= n =4$

**In fase di progettazione** dell'algoritmo parallelo

**con 2 core**

2 addizioni potrebbero essere eseguite concorrentemente ($T_c$) da 2 core (p)

NON SERVE ALTRO

$$T_1(4) = T_s + T_c =$$

$$= 0 + (2+2)$$

$T_s = 0$

nessuna operazione sequenziale

$T_c = 2+2$

operazioni che possono essere eseguite in parallelo

# Esempio:

**Somma di n=4 numeri**, $T_1(4)= n-1 = 3$

2 addizioni potrebbero essere eseguite concorrentemente ($T_c$) da 2 processori (p) al passo 1

1 addizione eseguita in sequenziale ($T_s$) da 1 processore al passo 2

**In fase di progettazione** dell'algoritmo parallelo

con 2 core

$$T_1(4) = T_s + T_c =$$

$$= 1 + 2 = 3$$

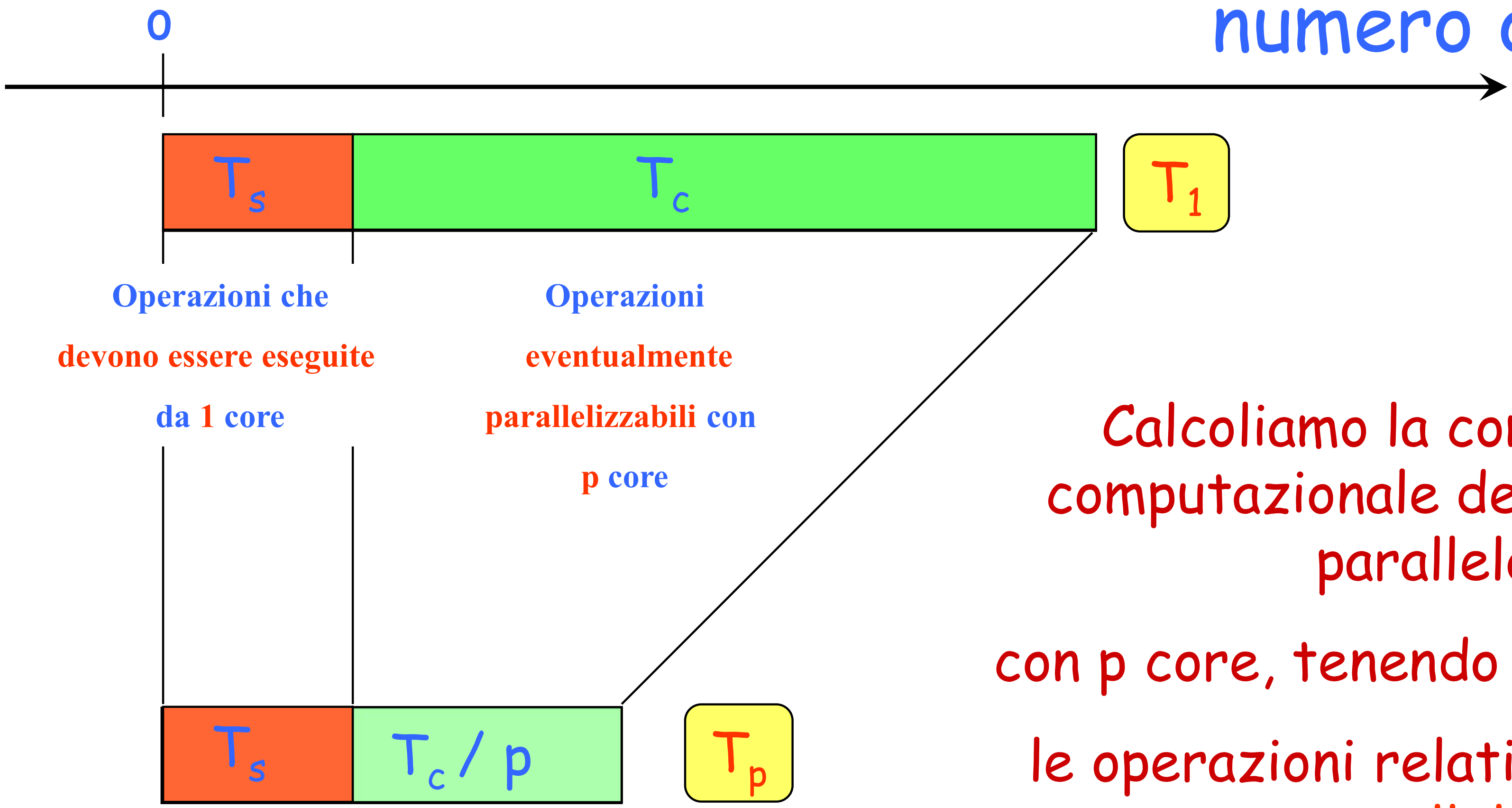
$T_s = 1$

Operazione che deve essere eseguita in sequenziale

$T_c = 2$

Operazioni che possono essere eseguite in parallelo

$$T_1 = T_s + T_c$$




Calcoliamo la complessità computazionale dell'algoritmo parallelo

con p core, tenendo presente che

le operazioni relative alla parte parallela

sono ora seguite concorrentemente

dai p core

$$T_p = T_s + T_c / p$$

# Esempio:

**Somma di 2 vettori di dimensione n=4**, p=2

$T_1(4) = n = 4$

2 addizioni eseguite concorrentemente ($T_c$) da 2 processori (p)

NON SERVE ALTRO

$T_2(4) = T_s + T_c / p =$

$= 0 + (2+2)/2$

$= 2$

$T_s = 0$

nessuna operazione sequenziale

$T_c / 2 = (2+2)/2 = 2$

operazioni che possono essere eseguite in parallelo

# Esempio:

Somma di $n$ =4 numeri con p=2,

$T_1(4)= n -1 =3$

2 addizioni eseguite concorrentemente ($T_c$) da 2 processori (p) al passo 1

1 addizione eseguita in sequenziale ($T_s$) da 1 processore al passo 2

$$T_p = T_s + T_c / p =$$

$$T_p = 1 + 1 = 2$$

$T_s = 1$

Operazione che viene eseguita in sequenziale

$T_c/p = 2/2 = 1$

Operazioni eseguite in parallelo

# In generale

La complessità computazionale di un algoritmo parallelo, su *p* processori, comprende 2 componenti:

- **$T_s$** operazioni per eseguire la parte seriale
- **$T_c/p$** operazioni per eseguire la parte parallela

# Smaltiamo un po' le notazioni

Ricordiamocelo questo!
Lo speedup riscritto in funzione dell'overhead

$$S_p = \frac{p}{\frac{O_h}{T_1} + 1}$$

$T_s = \alpha$ , $T_c = 1 - \alpha$

$T_p = T_s + T_c / p$ ⇒ $T_p = \alpha + (1 - \alpha) / p$

$$S_p = \frac{T_1}{T_p} = \frac{1}{\alpha + (1 - \alpha)/p}$$

***Legge di Ware (Amdahl)***

# Analizziamo questa formulazione…

Somma di due vettori di dimensione N

α=0, 1- α=1, (1-α)/p

$$S_p = \frac{1}{1/p} = p \xrightarrow[p \to \infty]{} \infty$$

Somma di N numeri

$$S_p = \frac{1}{\alpha + (1-\alpha)/p} \xrightarrow[p \to \infty]{} \frac{1}{\alpha}$$

# Esempio 1 (n fissato e p variabile)

Algoritmo della somma N numeri:

applichiamo la legge di W-A con $n$ =32 e p =2, 4, 8, 16

Al crescere del numero p di processori…

| p | $\alpha$ | $(1-\alpha)/p$ | $S_p$ | $E_p$ |
|---|---|---|---|---|
| 2 | 0,032 | 0,968 | 1,9 | 0,95 |
| 4 | 0,032 | 0,903 | 3,4 | 0,85 |
| 8 | 0,032 | 0,775 | 5,1 | 0,6 |
| 16 | 0,032 | 0,516 | 6,2 | 0,3 |

La parte *sequenziale*

è costante!

# Esempio 1 (n fissato e p variabile)

Algoritmo somma di N numeri:

applichiamo la legge di W-A con $n$ =32 e p =2, 4, 8, 16

Al crescere del numero p di processori…

| p | $\alpha$ | $(1-\alpha)/p$ | $S_p$ | $E_p$ |
|---|---|---|---|---|
| 2 | 0,032 | 0,968 | 1,9 | 0,95 |
| 4 | 0,032 | 0,903 | 3,4 | 0,85 |
| 8 | 0,032 | 0,775 | 5,1 | 0,6 |
| 16 | 0,032 | 0,516 | 6,2 | 0,3 |

Speed up ed efficienza degradano perché la parte parallela diminuisce!

# Per l'algoritmo della somma di N numeri…

Se la dimensione *n* del problema è fissata,

al crescere del numero p di core,

non solo non si riescono ad ottenere

speed up vicini a quello ideale

**MA**

## Le prestazioni peggiorano!

(non conviene utilizzare un elevato numero di core!!!)

# Caso migliore p=2 CPU

| p | $\alpha$ | $(1-\alpha)/p$ | $S_p$ | $E_p$ |
|---|---|---|---|---|
| 2 | 0,032 | 0,968 | 1,9 | 0,95 |
| 4 | 0,032 | 0,903 | 3,4 | 0,85 |
| 8 | 0,032 | 0,775 | 5,1 | 0,6 |
| 16 | 0,032 | 0,516 | 6,2 | 0,3 |

Facciamo delle osservazioni su questo caso migliore!!!

# Esempio CPU=2 e aumentiamo il size…

Applichiamo alla somma di n numeri

la legge di Amdahl con p =2 e $n$ = 8, 16, 32, 64

## Al crescere della dimensione $n$…

| n | $\alpha$ | $1-\alpha$ | $S_2(n)$ | $E_2(n)$ |
|---|---|---|---|---|
| 8 | 0,14 | 0,86 | 1,75 | 0,875 |
| 16 | 0,06 | 0,94 | 1,8 | 0,9 |
| 32 | 0,03 | 0,97 | 1,9 | 0,96 |
| 64 | 0,01 | 0,99 | 1,96 | 0,99 |

La parte *sequenziale* tende a zero!

# Esempio CPU=2 e aumentiamo il size…

Applichiamo alla somma di n numeri

la legge di Amdahl con p =2 e *n* = 8, 16, 32, 64

## Al crescere della dimensione *n*…

| n | $\alpha$ | $1-\alpha$ | $S_2(n)$ | $E_2(n)$ |
|---|---|---|---|---|
| 8 | 0,14 | 0,86 | 1,75 | 0,875 |
| 16 | 0,06 | 0,94 | 1,8 | 0,9 |
| 32 | 0,03 | 0,97 | 1,9 | 0,96 |
| 64 | 0,01 | 0,99 | 1,96 | 0,99 |

La parte *parallela* è costante

# Esempio CPU=2 e aumentiamo il size…

Applichiamo alla somma di n numeri

la legge di Amdahl con $p = 2$ e $n = 8, 16, 32, 64$

Al crescere della dimensione *n*…

| n | $\alpha$ | $1-\alpha$ | $S_2(n)$ | $E_2(n)$ |
|---|---|---|---|---|
| 8 | 0,14 | 0,86 | 1,75 | 0,875 |
| 16 | 0,06 | 0,94 | 1,8 | 0,9 |
| 32 | 0,03 | 0,97 | 1,9 | 0,96 |
| 64 | 0,01 | 0,99 | 1,96 | 0,99 |

Speed up ed efficienza sono "costanti"!

E soprattutto l'efficienza tende all'efficienza IDEALE!

# Che sta succedendo?

Se p è fissato, al crescere della dimensione *n* del problema, la parte sequenziale $\alpha \rightarrow 0$

| n | $\alpha$ | $1-\alpha$ | $S_2(n)$ | $E_2(n)$ |
|---|---|---|---|---|
| 8 | 0,14 | 0,86 | 1,75 | 0,875 |
| 16 | 0,06 | 0,94 | 1,8 | 0,9 |
| 32 | 0,03 | 0,97 | 1,9 | 0,96 |
| 64 | 0,01 | 0,99 | 1,96 | 0,99 |

# In W-A

$$S_p = \frac{1}{\underbrace{\boxed{\alpha}}_{\downarrow\, 0} + \frac{(1-\underbrace{\boxed{\alpha}}_{\downarrow\, 0})}{p}} \xrightarrow[\alpha \to 0]{} p$$

$S_p \to \; p$

# Per incrementare le performance

Fissare il miglior numero p di CPU

e

aumentare la dimensione del problema

# Attenzione…

**MA**

non è possibile aumentare in maniera indefinita la dimensione *n* del problema:
le risorse (hardware) sono limitate!

Cosa abbiamo imparato dalla legge di W-A per l'algoritmo parallelo

della somma di N numeri?

1.

Fissato il size n del problema e aumentando il numero p di CPU

…esiste $p_{max}$ miglior numero di processori per risolvere il problema con l'algoritmo in esame

superato questo valore le prestazioni peggiorano!!!

2.

Fissato il numero p delle CPU e aumentando il size n del problema

…esiste $n_{max}$ la massima dimensione che la macchina può memorizzare/elaborare

superato questo valore potrei avere problemi con la memoria hardware!!!