Valutazione delle prestazioni di programmi paralleli

Moreno Marzolla
Dip. di Informatica—Scienza e Ingegneria (DISI)
Università di Bologna

moreno.marzolla@unibo.it

Speedup

- Definiamo
  - p = Numero di processori / core
  - T_{serial} = Tempo di esecuzione del programma sequenziale
  - $T_{parallel}(p)$ = Tempo di esecuzione del programma parallelo su p processori / core
- Nel caso ideale, il programma parallelo impiega 1/p del tempo del programma sequenziale

$$T_{parallel}(p) = T_{serial} / p$$

Speedup ed Efficienza

Spesso si definisce
Tserial = Tparallel(1)

- Speedup S(p)

$$S(p) = \frac{T_{\text{serial}}}{T_{\text{parallel}}(p)}$$

- Efficienza E(p)

$$E(p) = \frac{S(p)}{p} = \frac{T_{\text{serial}}}{p \times T_{\text{parallel}}(p)}$$

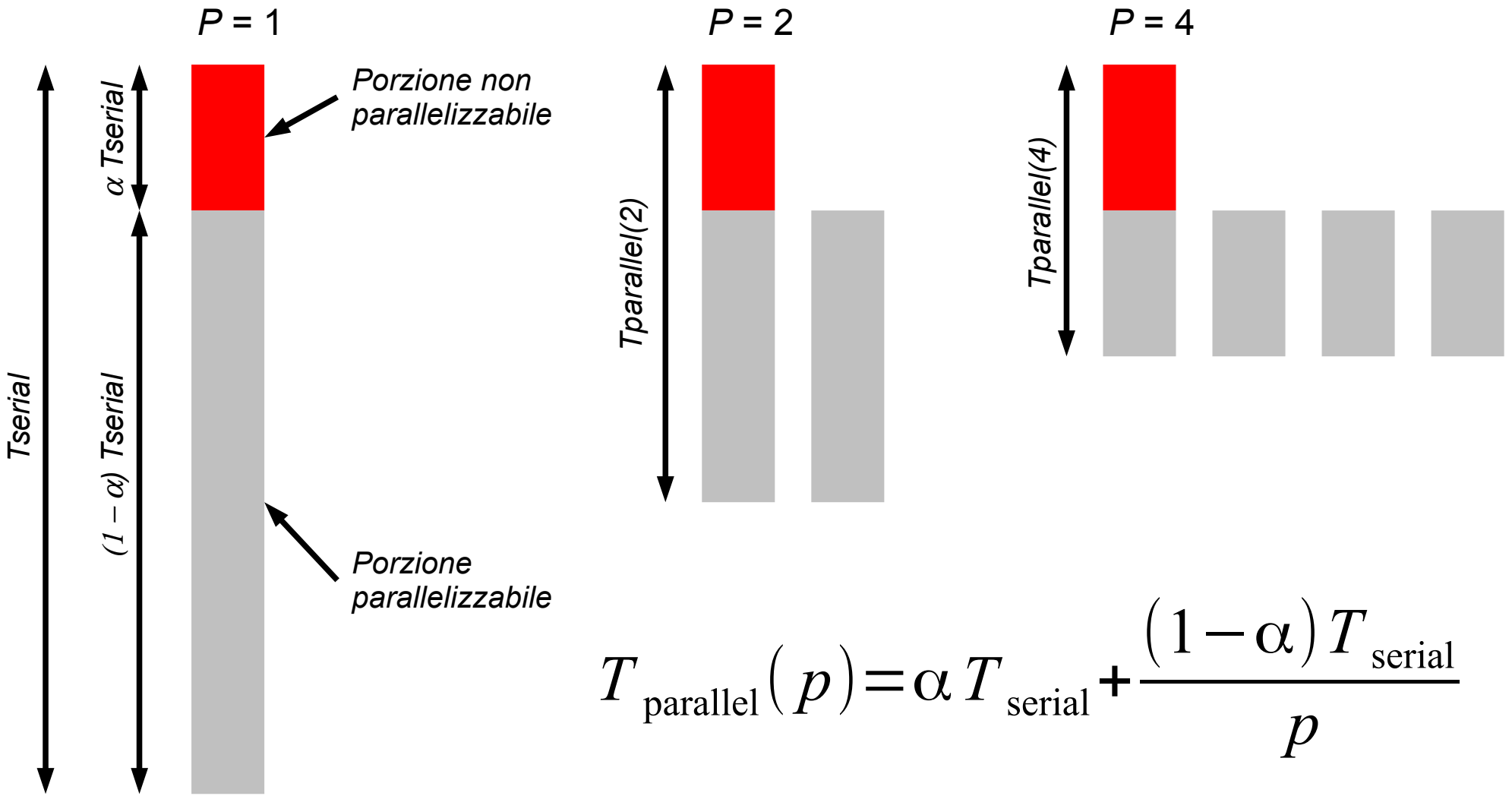
- Solitamente S(p) ≤ p
  - S(p) = p è il caso ottimo di speedup lineare
  - Quando può verificarsi S(p) > p ?

Porzioni non parallelizzabili

- Supponiamo che una frazione α del tempo di esecuzione sia dovuto ad una componente intrinsecamente seriale del programma
- Supponiamo che la rimanente frazione (1 - α) del tempo di esecuzione sia relativo ad una componente che puo' essere parallelizzata perfettamente
- Quindi avremo

$$T_{\text{parallel}}(p) = \alpha T_{\text{serial}} + \frac{(1-\alpha) T_{\text{serial}}}{p}$$

Esempio



$$T_{\text{parallel}}(p) = \alpha T_{\text{serial}} + \frac{(1-\alpha) T_{\text{serial}}}{p}$$

Legge di Amdahl

- Quale è lo speedup ottenibile?

$$S(p) = \frac{T_{\text{serial}}}{T_{\text{parallel}}(p)} = \frac{T_{\text{serial}}}{\alpha T_{\text{serial}} + \frac{(1-\alpha) T_{\text{serial}}}{p}}$$

$$= \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

Legge di Amdahl

Gene Myron Amdahl (1922—)

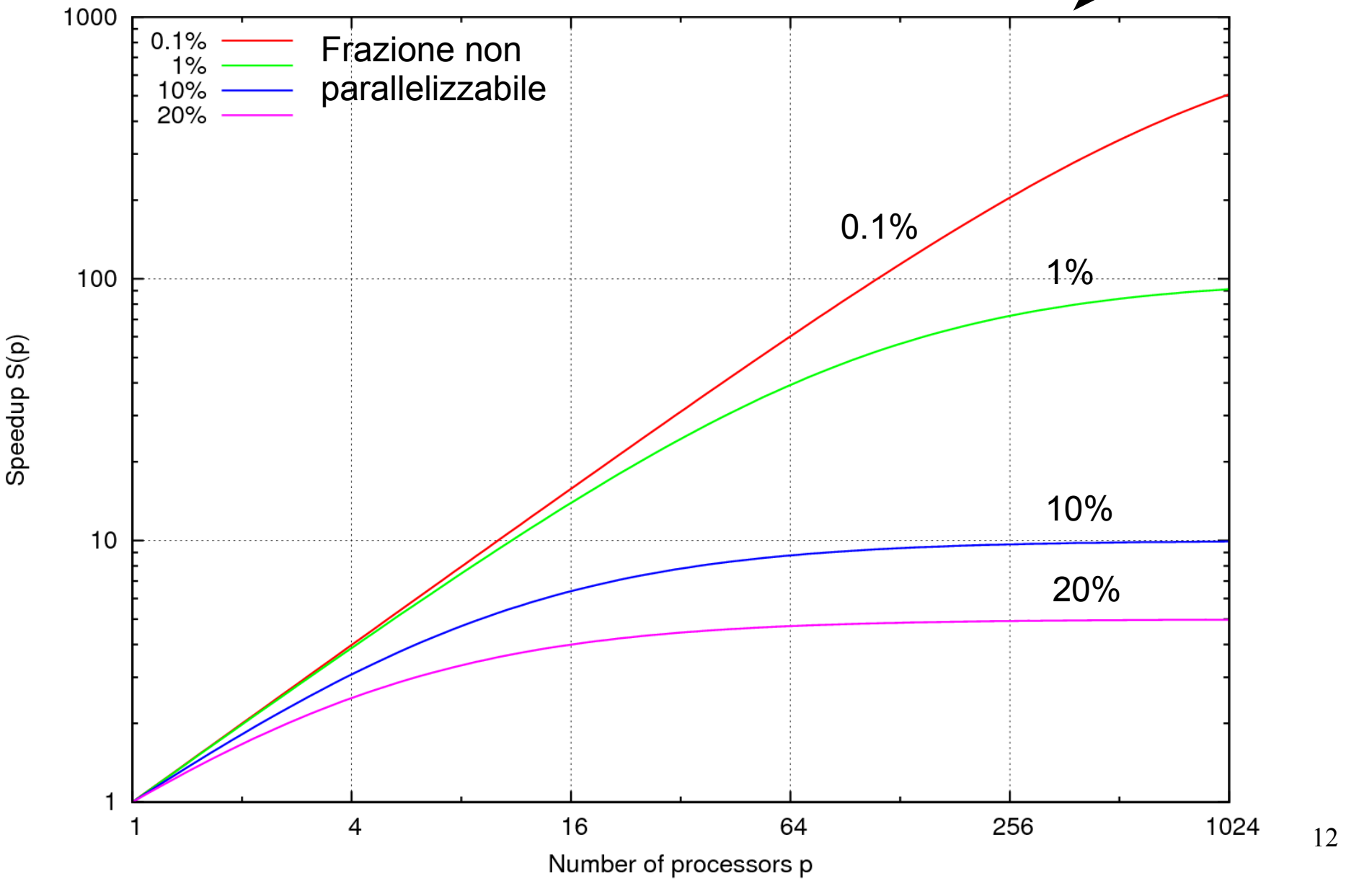
Legge di Amdahl

- Da $$S(p) = \frac{1}{\alpha + \frac{(1-\alpha)}{p}}$$

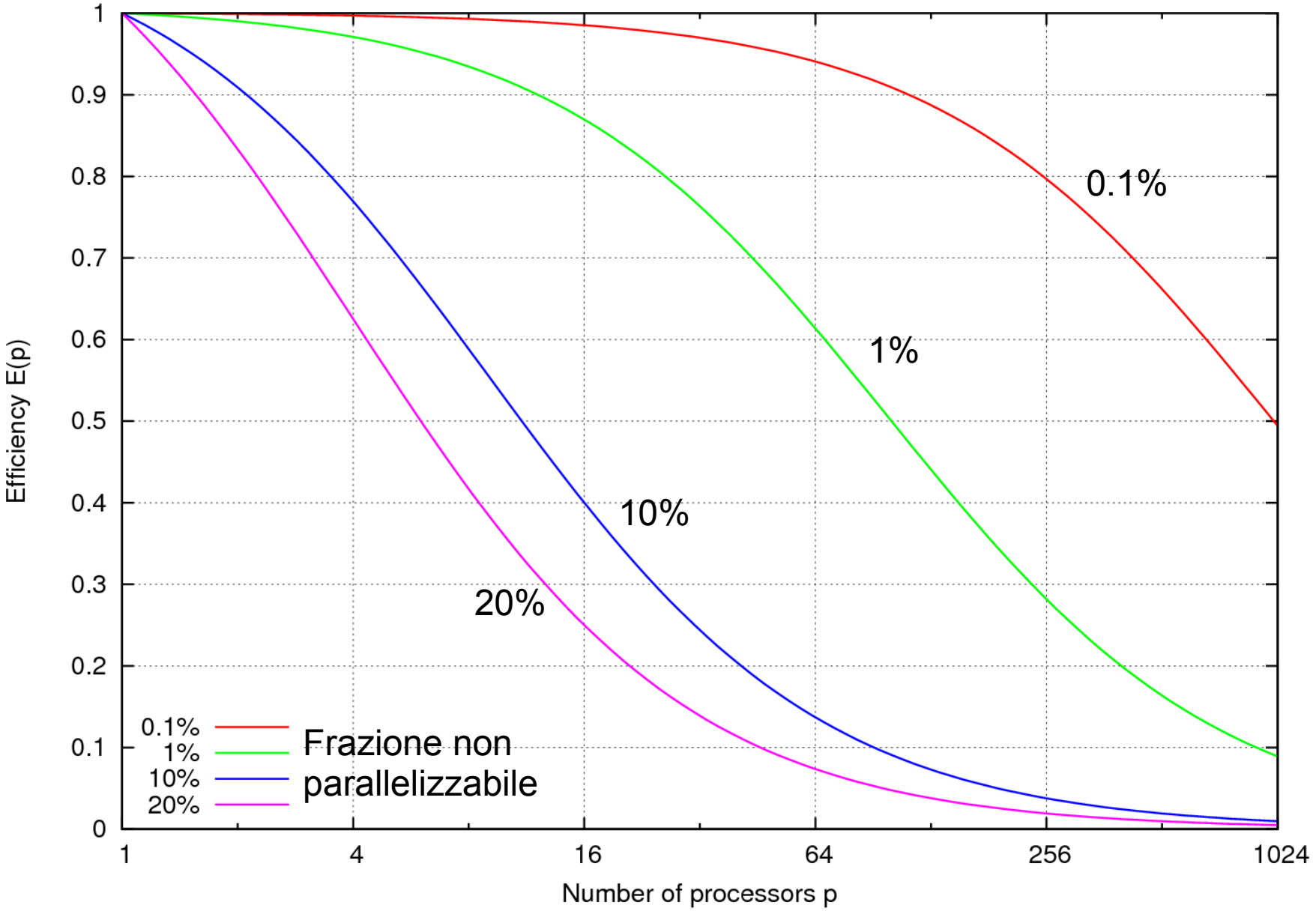
  otteniamo che, per *p* tendente a +infinito, lo speedup asintotico è 1 / α

Se una frazione α di un programma seriale non è parallelizzabile, allora lo speedup è limitato superiormente da 1/α, indipendentemente dal numero di core a disposizione

Speedup


Note: scala log-log
Frazione non parallelizzabile
0.1%
1%
10%
20%
1000
100
10
1
Speedup S(p)
1
4
16
64
256
1024
Number of processors p

Efficiency

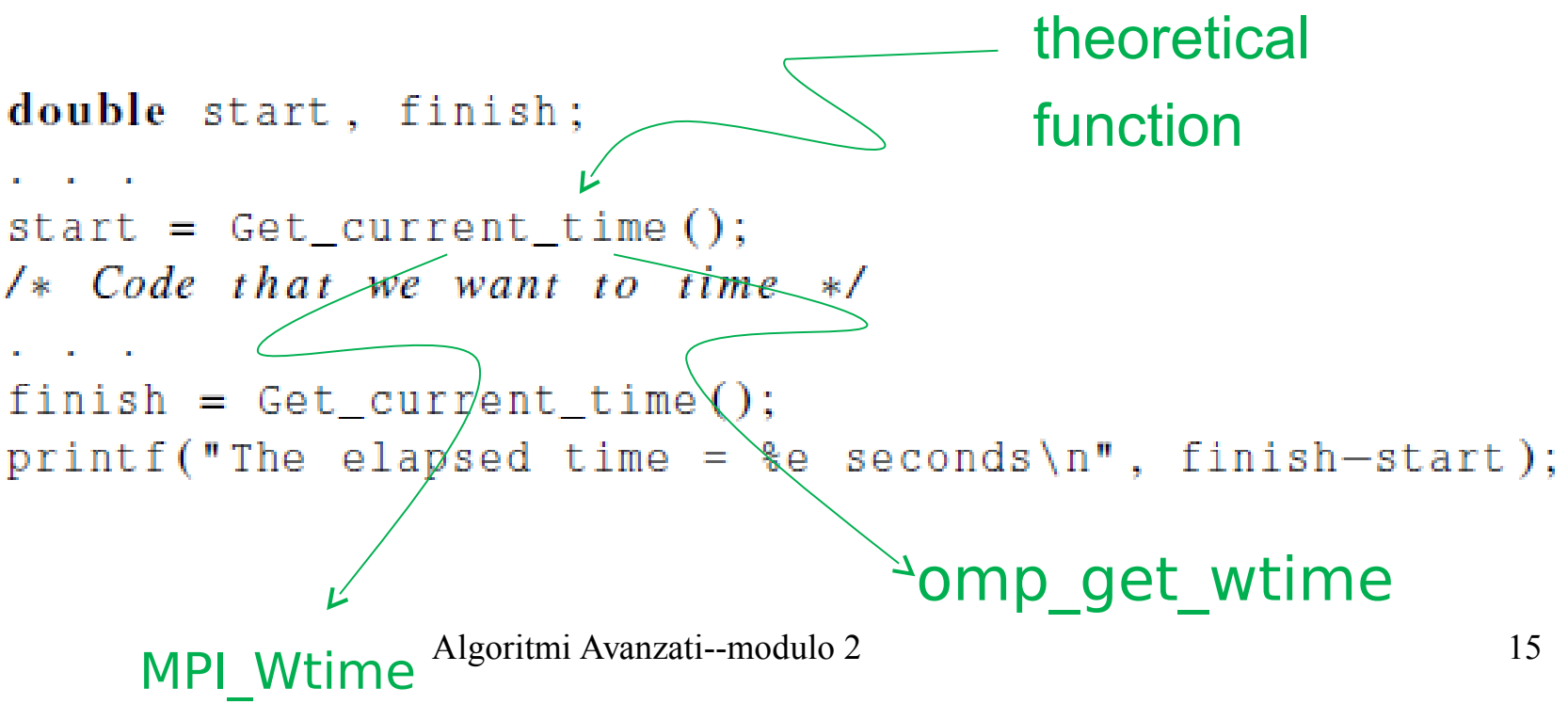

Efficiency E(p)
Number of processors p
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
4
16
64
256
1024
0.1%
1%
10%
20%
0.1%
1%
10%
20%
Frazione non
parallelizzabile

Scalabilità

- Un algoritmo è scalable se riesce a gestire problemi di dimensioni crescenti
  - Se riusciamo a incrementare il numero di processiori mantenendo l'efficienza e la dimensione del problema costanti, l'algoritmo è strongly scalable.
  - Se riusciamo a mantenere l'efficienza costante incrementando contemporaneamente la dimensione del problema e il numero di core, l'algoritmo è weakly scalable.

Misurazione dei tempi

- Per valutare sperimentalmente speedup ed efficienza, il tempo di esecuzione che viene misurato è il wall clock time del programma, ESCLUDENDO il tempo necessario a leggere l'input e a scrivere l'output.



Misurazione dei tempi

- Per valutare sperimentalmente speedup ed efficienza, il tempo di esecuzione che viene misurato è il wall clock time del programma, ESCLUDENDO il tempo necessario a leggere l'input e a scrivere l'output.

```
double start, finish;
/* lettura dei dati di input da file; questo NON va considerato */

start = Get_current_time(); /* omp_get_wtime() oppure MPI_Wtime() */
/* parte di codice di cui vogliamo misurare il wall clock time */
finish = Get_current_time();

printf("The elapsed time = %e seconds\n", finish - start);

/* scrittura dell'output; questo NON va considerato */
```

Questa è la parte in cui sono svolte le computazioni,

Misurazione del wall-clock time

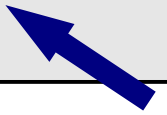
- Per i programmi OpenMP: omp_get_wtime()
- Per i programmi MPI: MPI_Wtime()
- Soluzione “generica”: clock_gettime()
- Soluzione **SBAGLIATA**: clock()

```
NAME
       clock - Determine processor time

SYNOPSIS
       #include <time.h>

       clock_t clock(void);

DESCRIPTION
       The clock() function returns an
approximation of processor time used by the
program.
```



Esempio

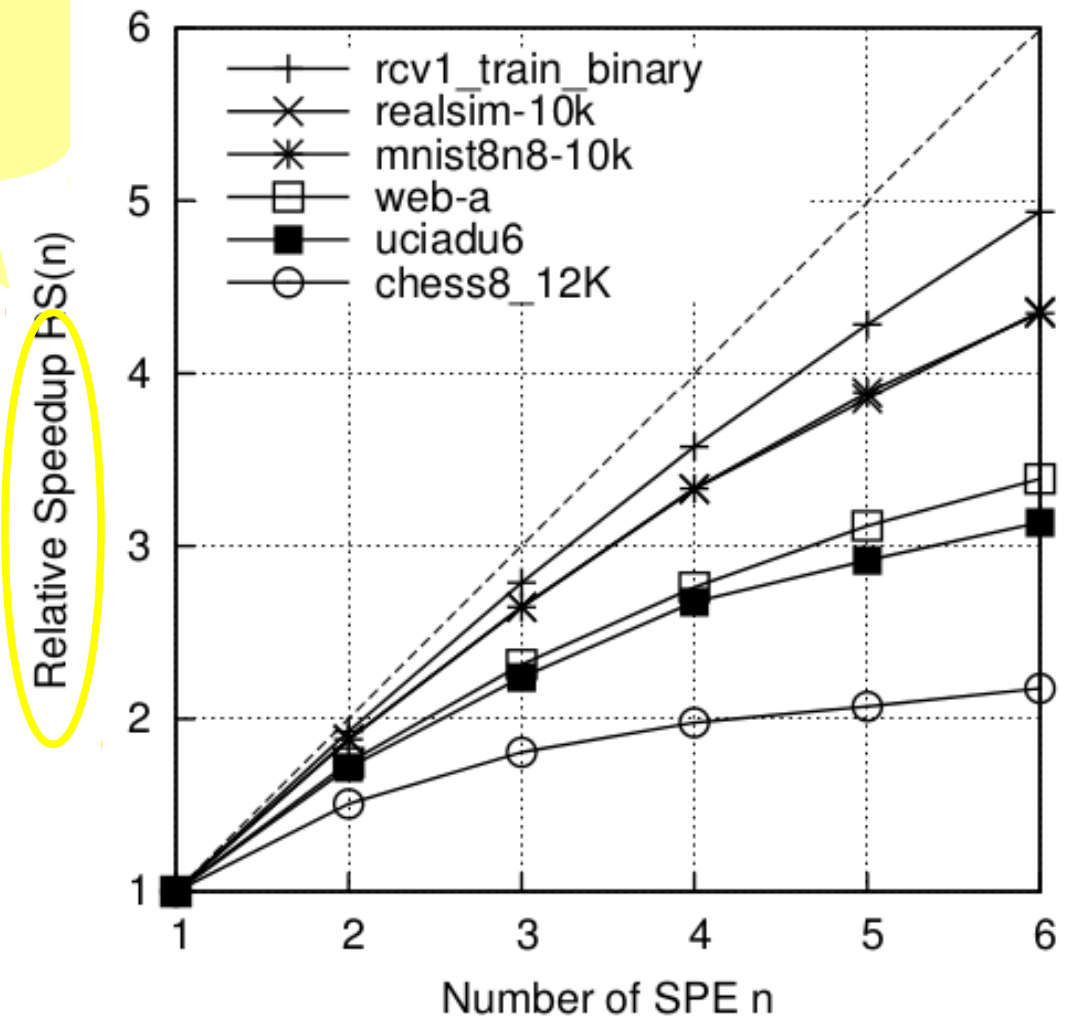
	T_{CPU}	T_{PPE}	$T_{SPE,1}$	$T_{SPE,2}$	$T_{SPE,3}$	$T_{SPE,4}$	$T_{SPE,5}$	$T_{SPE,6}$	**Speedup** $(T_{CPU}/T_{SPE,6})$
chess8_12K	32.82	62.53	41.64	27.61	23.05	21.05	20.10	19.13	1.72
mnist8n8-10k	253.46	334.33	173.64	92.30	65.62	52.09	44.65	39.93	6.35
uciadu6	24.96	52.72	30.20	17.57	13.48	11.28	10.35	9.63	2.59
web-a	70.22	149.92	98.74	56.54	42.68	35.74	31.68	29.13	2.41
rcv1_train_binary	617.17	1674.64	1079.19	560.00	387.19	301.79	251.96	218.71	2.82
realsim-10k	107.76	279.62	185.07	98.38	69.91	55.57	48.02	42.46	2.54

Table 2: Wall-clock training time, in seconds (average of 5 runs, lower is better)

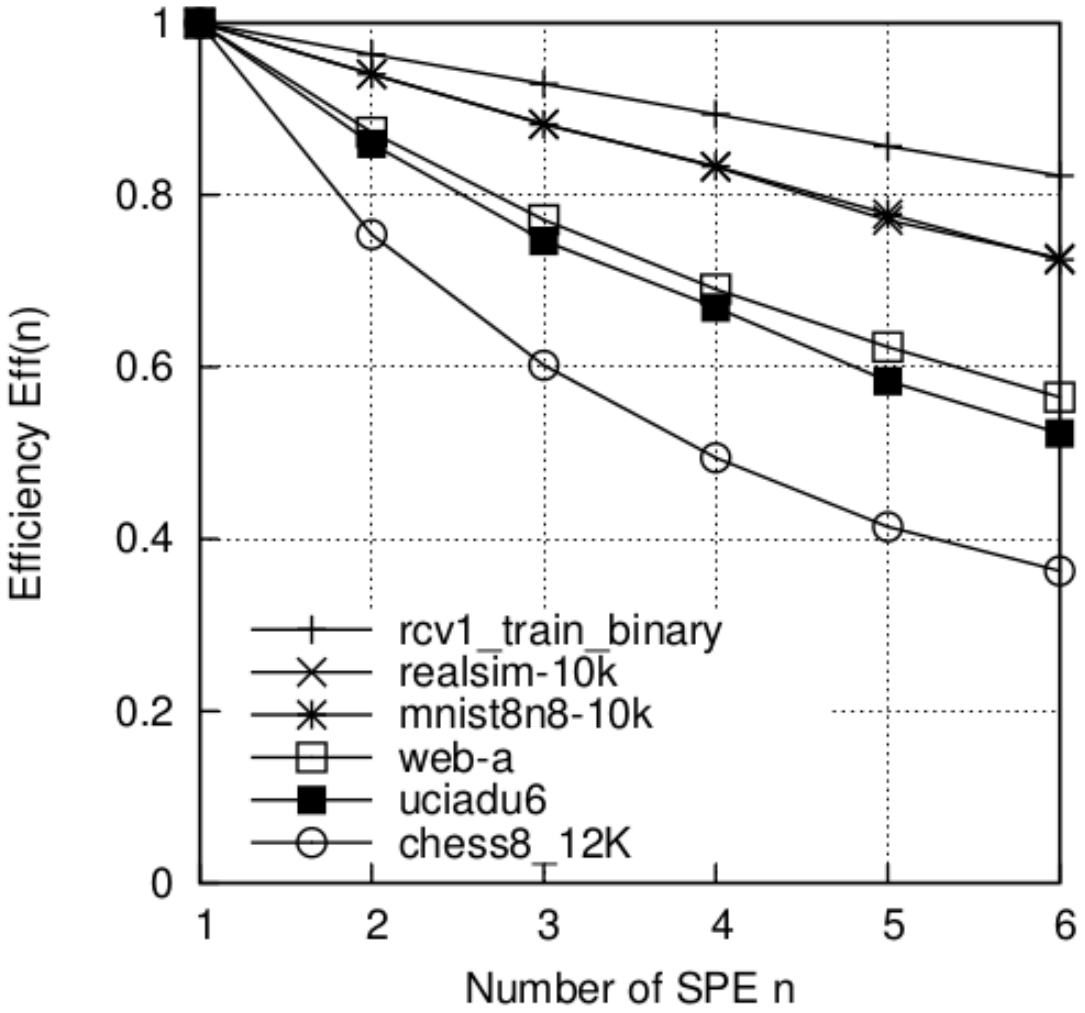
M. Marzolla, *Fast Training of Support Vector Machines on the Cell Processor,* Neurocomputing, Volume 74, Issue 17, October 2011, pp. 3700–3707, ISSN 0925-2312; disponibile all'indirizzo http://www.moreno.marzolla.name/publications/

Esempio

$RS(n) = T_{SPE,n} / T_{SPE,1}$



Esempio


Efficiency Eff(n)
1
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6
Number of SPE n
rcv1_train_binary
realsim-10k
mnist8n8-10k
web-a
uciadu6
chess8_12K