UNIVERSITÀ DELLA CALABRIA

Il Calcolo Parallelo nelle Scienze Computazionali: un Overview

William Spataro
Dipartimento di Matematica e Informatica
Università della Calabria
spataro@unical.it

Computazione Sequenziale

- Tradizionalmente, il software è stato scritto per il calcolo seriale:
  - Per essere eseguito su un singolo computer con una singola Unità Centrale di Elaborazione (CPU);
  - Un problema è suddiviso in una serie discreta di istruzioni.
  - Le istruzioni vengono eseguite una dopo l'altra.
  - Solo una istruzione può eseguire in qualsiasi momento nel tempo
  - Teorema di Böhm-Jacopini

Le ultime parole famose …

- "Penso che ci sia un mercato mondiale per forse (al più) cinque computer."

Thomas Watson, presidente della IBM 1943.

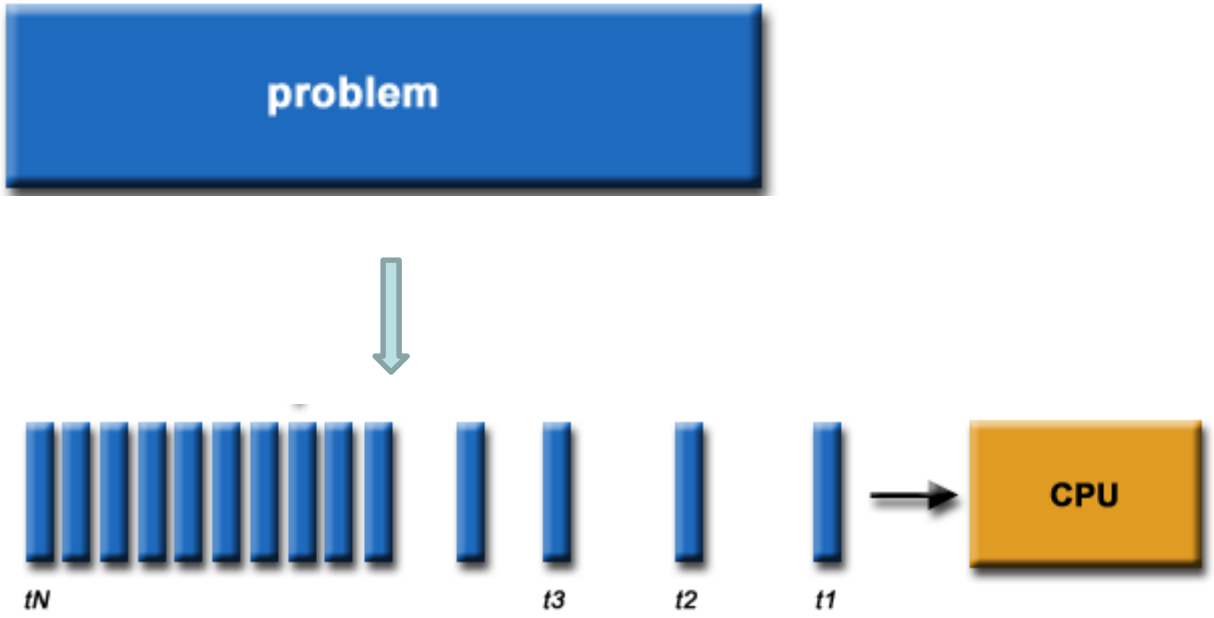
- "Non c'è alcuna ragione per ogni individuo di avere un computer in casa"

Ken Olson, presidente e fondatore di Digital Equipment Corporation (successivamente HP), 1977.

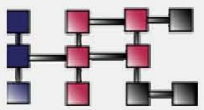
- "640K [di memoria] dovrebbe essere abbastanza per chiunque."

Bill Gates, presidente di Microsoft, 1981.

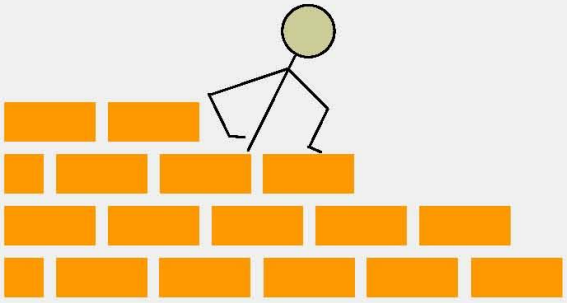
Computazione Sequenziale



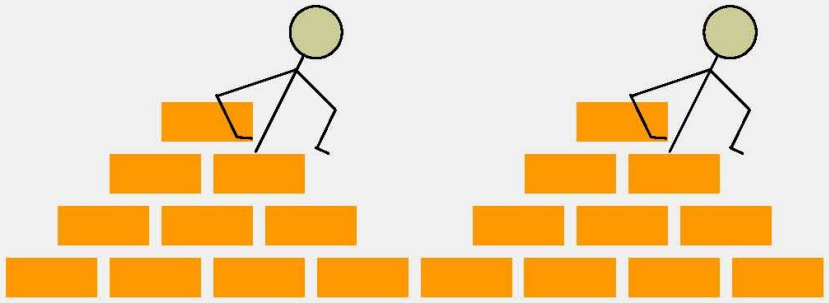
Cos'è il calcolo parallelo



- esempio di processi paralleli
  - dividere il lavoro (job) in processi più piccoli (task)
  - assegnare questi piccoli task a molti processi che lavorano simultaneamente
  - coordinare, controllare e monitorare questi processi
- Un esempo della vita quotidiana:
  - la costruzione di un muro



elaborazione sequenziale



elaborazione parallela

Definizione semplicista di Calcolo Parallelo

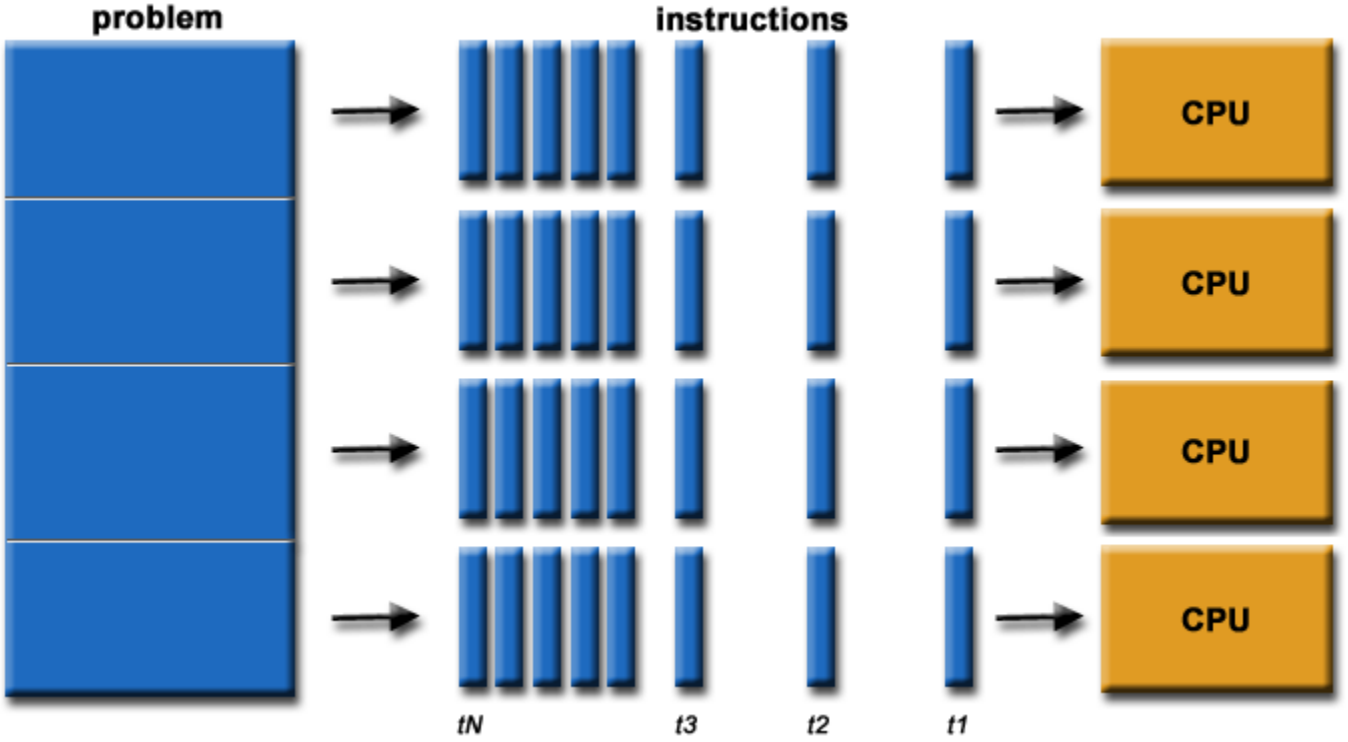
Nel senso piu' semplice, il **Calcolo Parallelo** è l'uso simultaneo di risorse di calcolo multiple per risolvere un problema computazionale:

1 – Tramite CPU multipli

2 – Un problema è spezzato in parti discrete che possono essere risolte in modo concorrente

3 - Ogni parte è ulteriormente spezzato in una serie di istruzioni

4 - Le istruzioni di ogni parte sono eseguite simultaneamente su CPU differenti



Seriale vs. Parallelo: Esempio

Esempio: Preparare la cena

Attività seriali: fare la salsa, assemblare la lasagna, cottura lasagne; lavaggio lattuga, tagliare le verdure, assemblaggio insalata

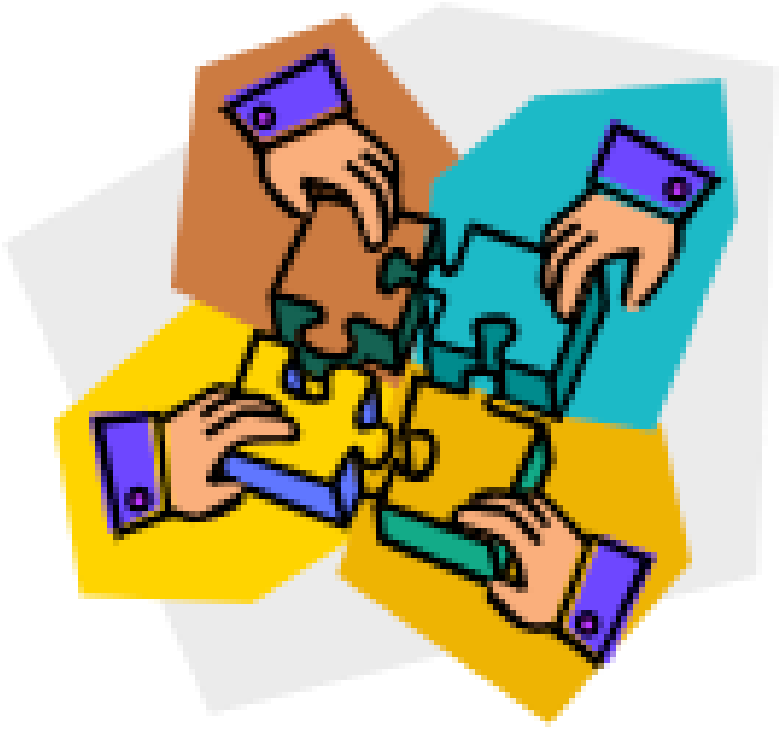
Attività parallele: preparare le lasagne, preparare l'insalata, apparecchiare la tavola

Seriale vs. Parallelo: Esempio

- Possiamo avere diversi cuochi, ognuno che esegue di un compito parallelo

Questo è il concetto dietro il calcolo parallelo!

Jigsaw Puzzle: Domande da porsi



- Aggiungere un'altra persona al tavolo
  - Effetto sul tempo
  - comunicazione
  - contesa delle risorse
- Aggiungere p persone al tavolo
  - Effetto sul tempo
  - comunicazione
  - contesa delle risorse

Dal Calcolo Sequenziale al Calcolo Parallelo

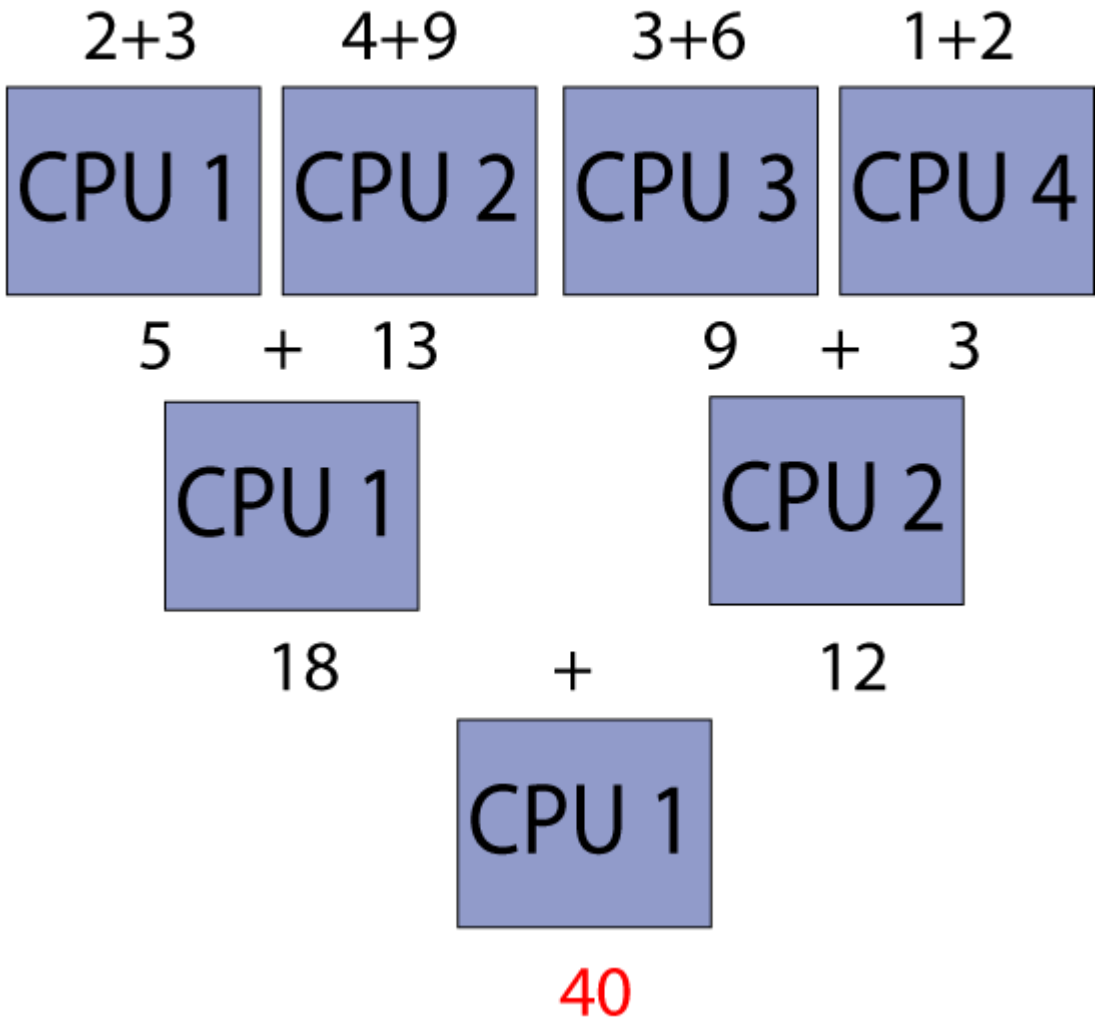
- Risolvere lo stesso problema tramite un algoritmo le cui istruzioni sono eseguite in modo parallelo

- Modello computazionale che prevede processori multipli e relativi meccanismi di cooperazione

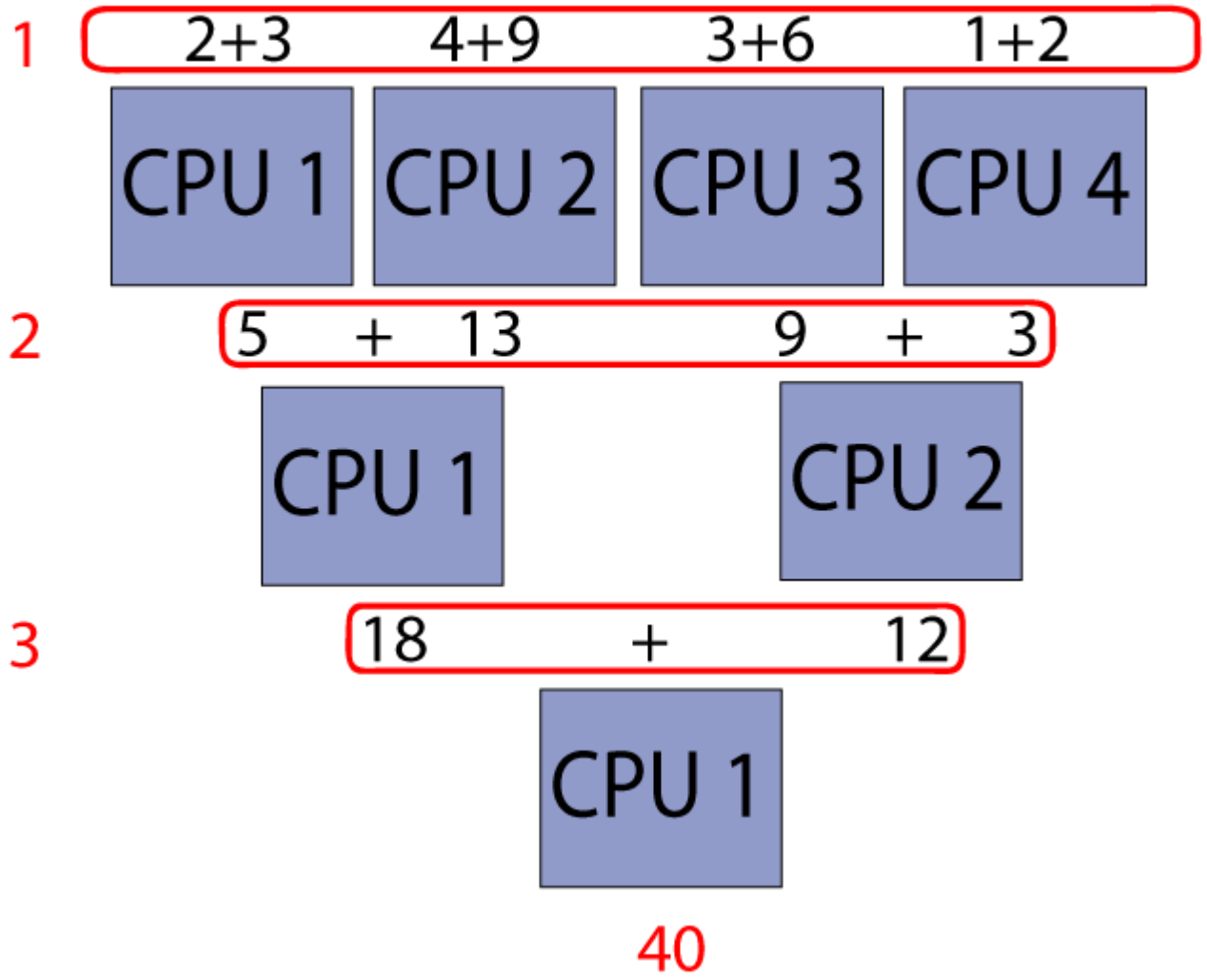
- Es: Supponiamo di dover effettuare la somma di 8 numeri con 4 CPU, ognuna delle quali effettua la somma di due variabili. In seguito 2 CPU effettuano la somma dei due risultati precedenti e 1 CPU somma alla fine i due risultati ottenuti.

Bastano 3 passi invece dei 7 precedenti ($\log_2$ n passi)

data la sequenza (2,3,4,9,3,6,1,2)



data la sequenza (2,3,4,9,3,6,1,2)



- Parallelism and Amdahl's Law
- Finding and exploiting granularity
- Preserving data locality
- Load balancing
- Coordination and synchronization
- Performance modeling

All of these issues makes parallel programming harder than sequential programming

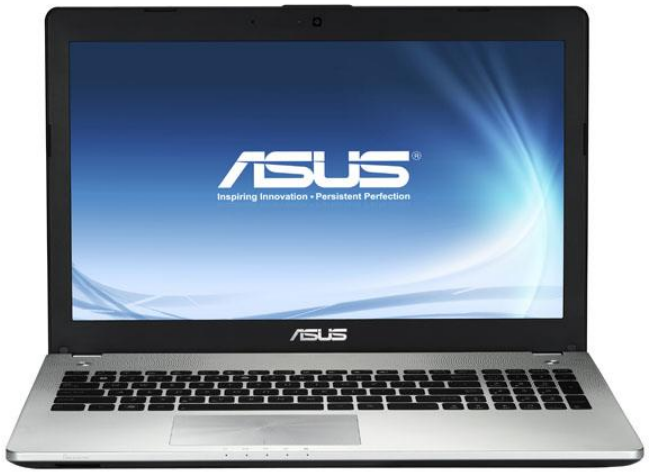
Non è per nulla scontato che l'utilizzo di più processori ci aiutino a raggiungere questi obiettivi:

Se un uomo può scavare un buco di 1 m^3 in 1 ora, 60 uomini possono scavare lo stesso buco in 1 minuto (!)? 3600 uomini possono farlo in 1 secondo (!!)?

So come «organizzare» 4 cavalli che tirano un carro, ma non so come fare con 1024 galline !

Esistono Macchine Parallele a basso costo ?

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4

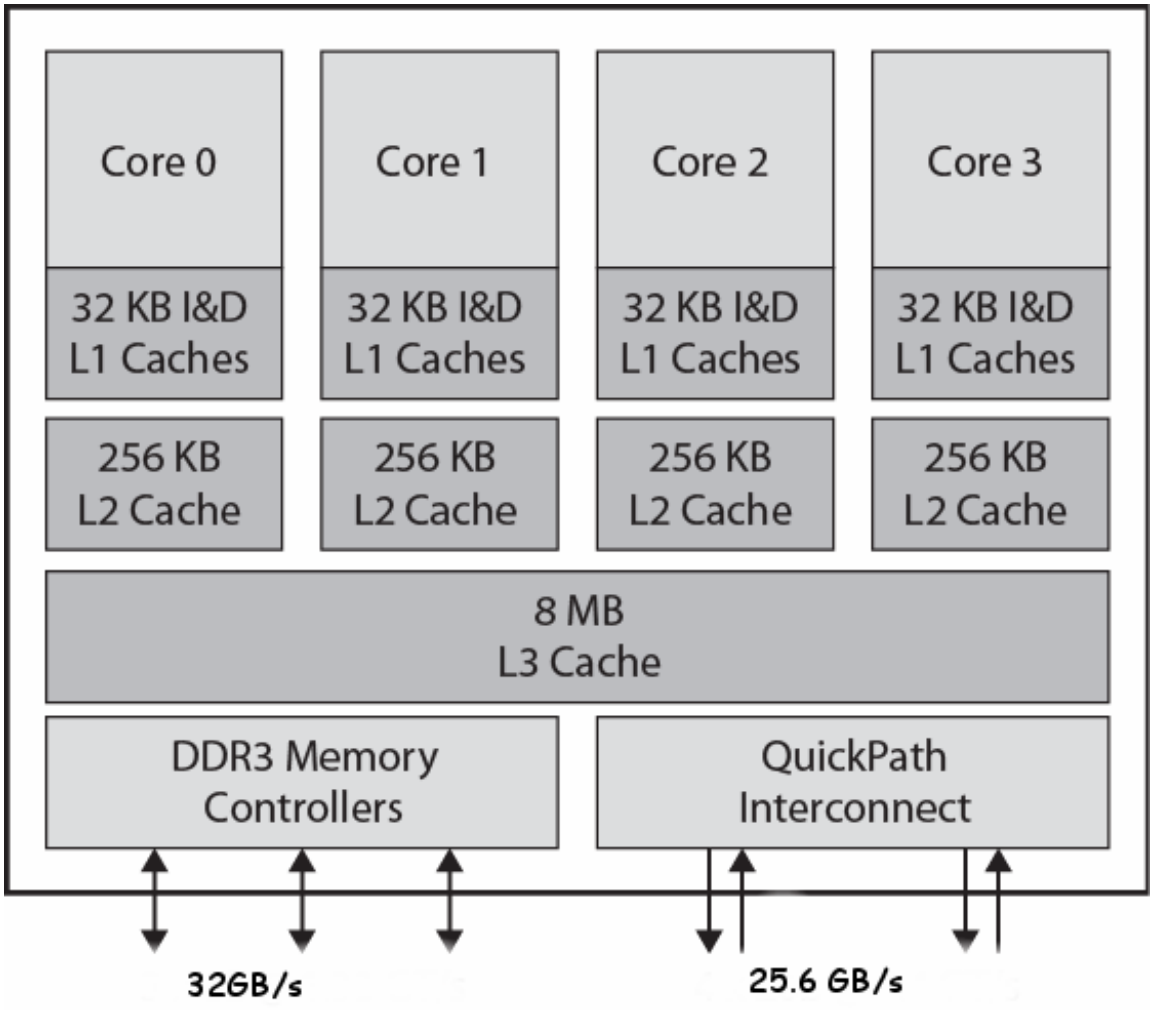


Il Mio Notebook - 4 cores
250 GFlops

Unità di misura adottata per valutare la “velocità” di una macchina parallela:

Flop/s: floating point operations per second

Intel Core i7 Block Diagram



.3 ns/B !

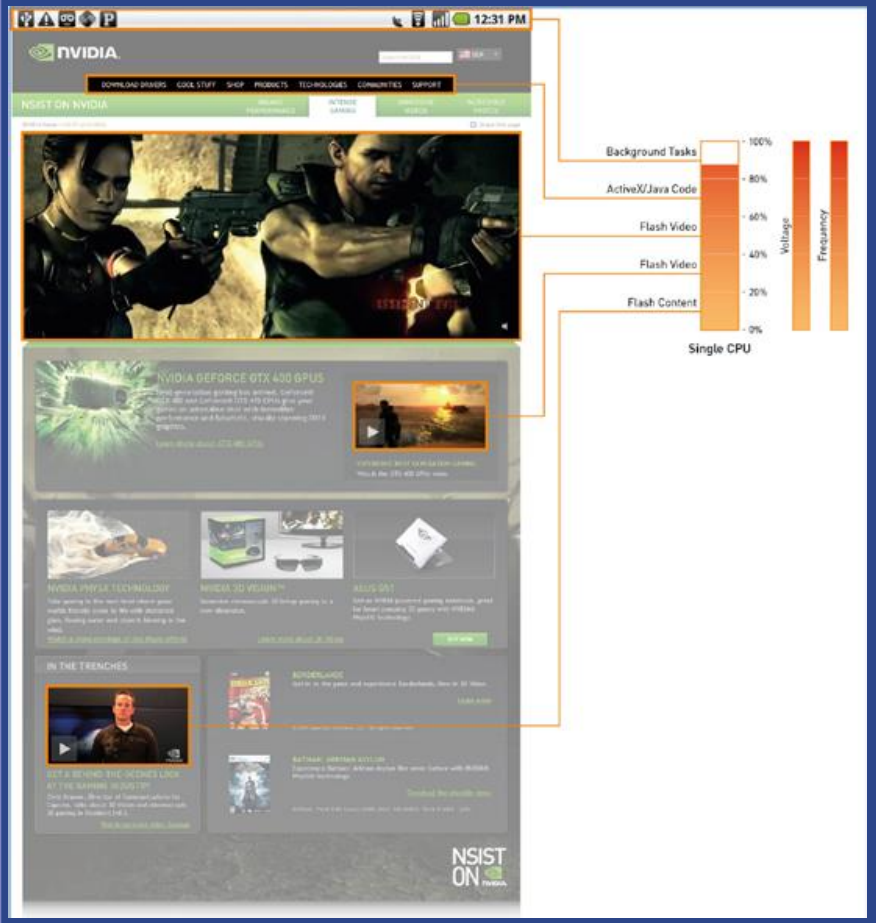
CPU – multi-core: Perchè?

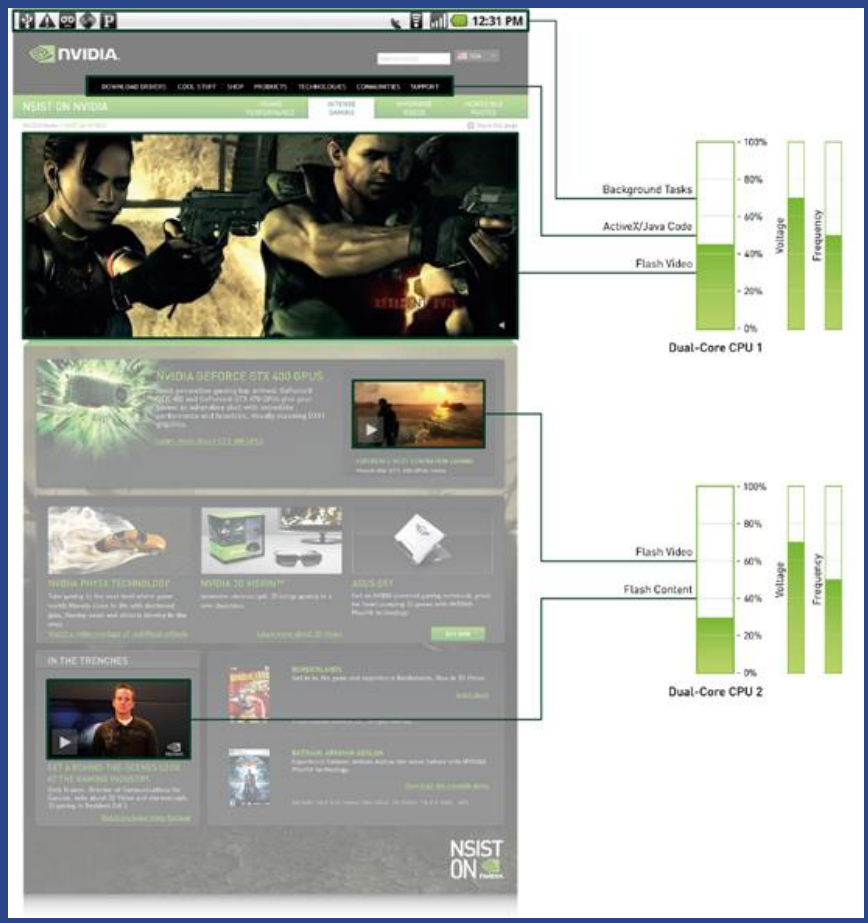
- Aumentare le prestazioni solo aumentando le prestazioni del clock porta a intasamenti
- Aumentano inoltre i consumi energetici
- Diventa difficile raffreddare le macchine specie nelle versioni portatili

CPU – multi-core

1core

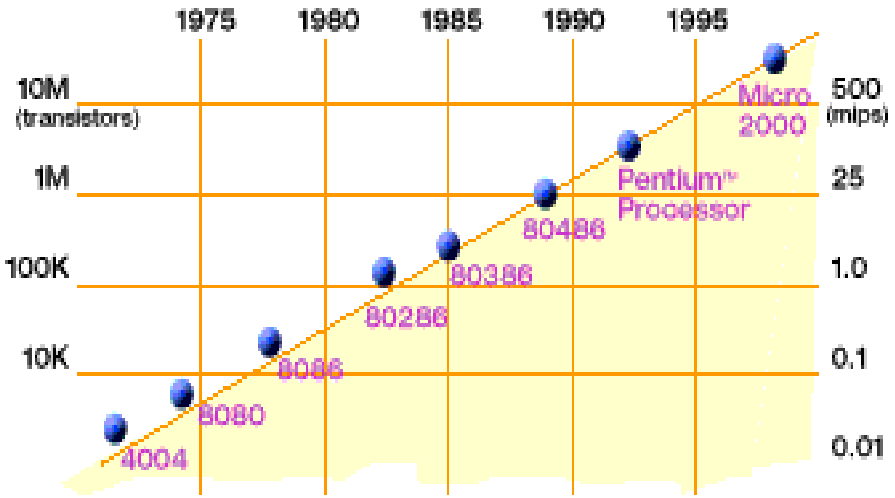
2 core





Per ora siamo arrivati a 16 core per macchine commerciali

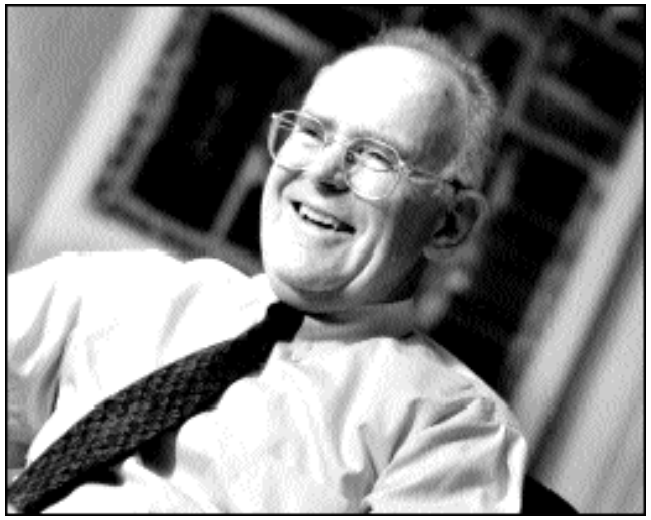
Technology Trends: Microprocessor Capacity



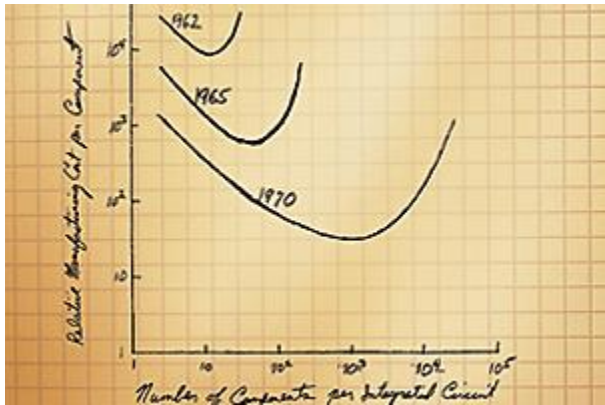
2X transistors/Chip Every 1.5 - 2 years

Called "Moore's Law"

Microprocessors have become smaller, denser, and more powerful.

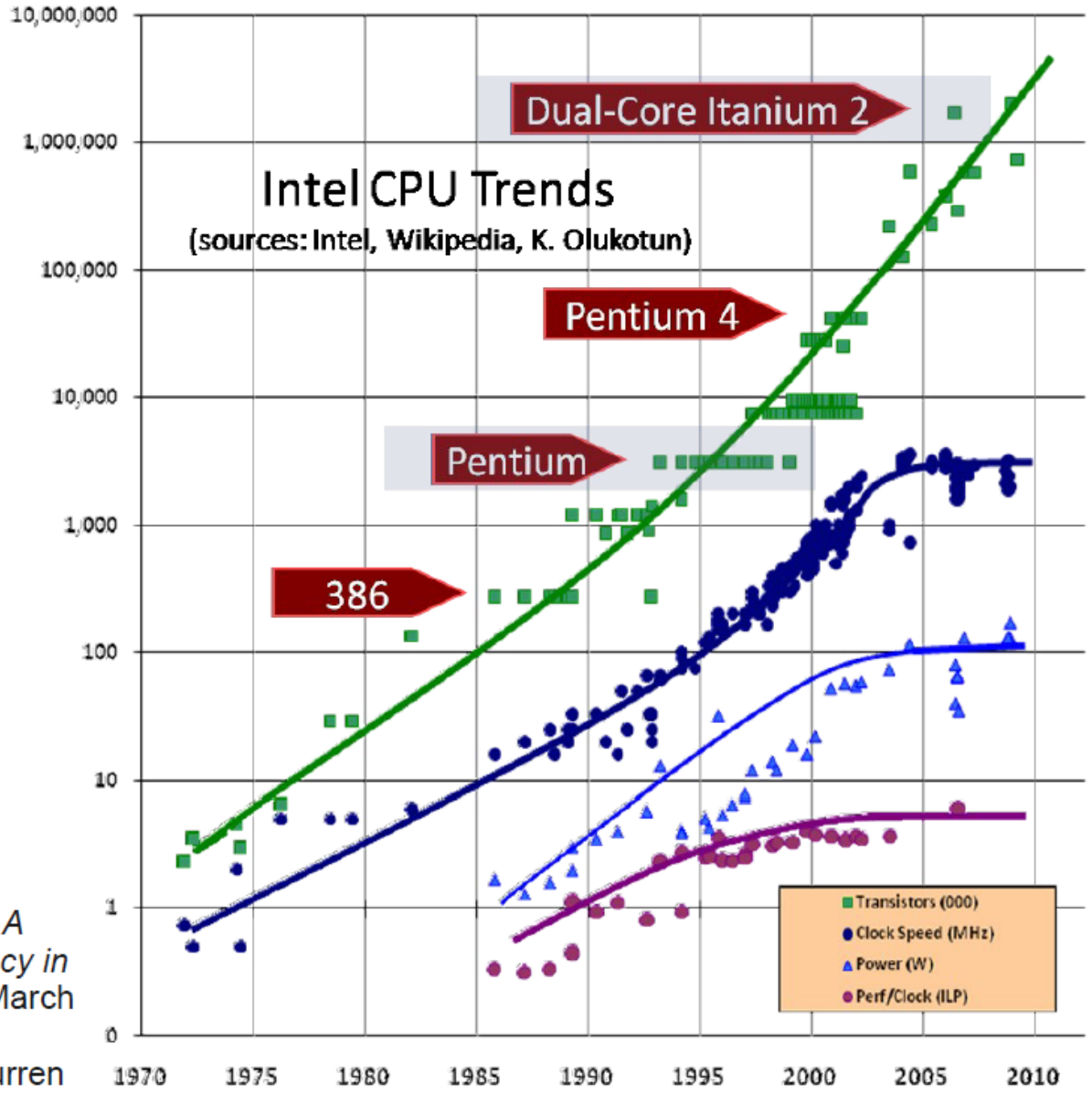


Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.



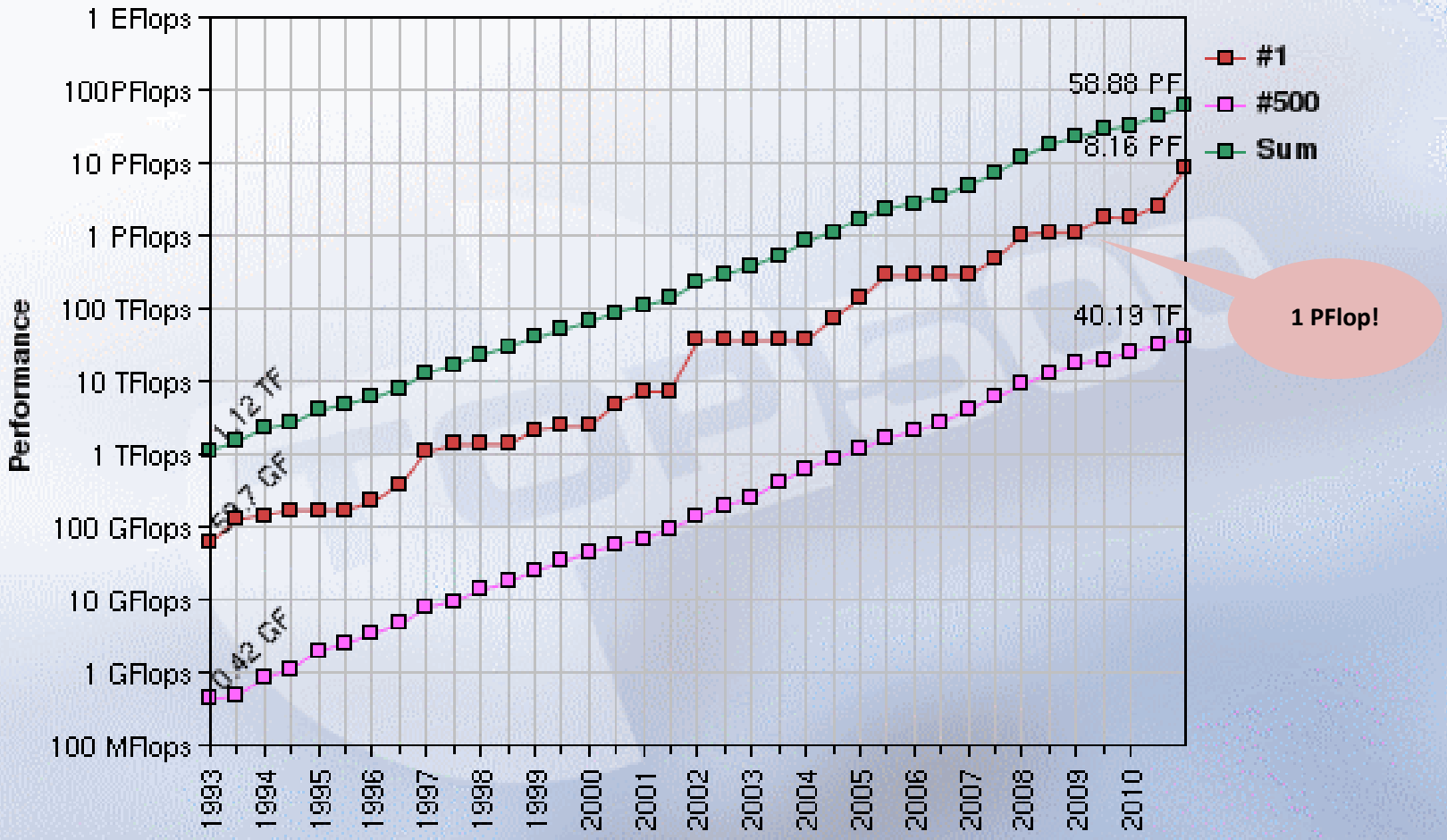
The free lunch is over

- Il software diventa sempre più complesso
- Soluzione comoda: aspettare la "prossima" generazione di CPU con una frequenza di clock maggiore
- Purtroppo, la pacchia è finita



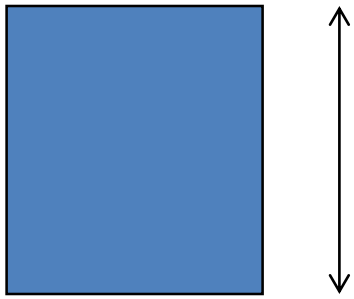
Herb Sutter, *The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software*, Dr. Dobb's Journal, 30(3), March 2005, http://www.gotw.ca/publications/concurrency-ddj.htm

Performance Development



Physical limits: how fast can a serial computer be?

1 Tflop/s, 1 Tbyte sequential machine



r = 0.3 mm

- Consider the 1 Tflop/s sequential machine:
  - Data must travel some distance, r, to get from memory to CPU.
  - Go get 1 data element per cycle, this means 10^{12} times per second at the speed of light, $c = 3x10^8$ m/s. Thus $r < c/10^{12} = 0.3$ mm.
- Now put 1 Tbyte of storage in a 0.3 mm 0.3 mm area:
  in fact 0.3^2 mm$^2/10^{12} = 9\ 10^{-2}\ 10^{-6}$ m$^2/10^{12} =$
  $9\ 10^{-20}$ m$^2 = (3\ 10^{-10})^2$ m$^2 = 3^2$ A^2 →
  - Each byte occupies less than 3 square Angstroms, or the size of a small atom! (1 Angstrom = 10^{-10} m = 0.1 nanometer)

No choice but parallelism !!!

TOP 10 – November 2014 (!!!)

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325
7	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462	5,168.1	8,520.1	4,510
8	Forschungszentrum Juelich (FZJ) Germany	**JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458,752	5,008.9	5,872.0	2,301
9	DOE/NNSA/LLNL United States	**Vulcan** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393,216	4,293.3	5,033.2	1,972
10	Government United States	Cray CS-Storm, Intel Xeon E5-2660v2 10C 2.2GHz, Infiniband FDR, Nvidia K40 Cray Inc.	72,800	3,577.0	6,131.8	1,499

Speedup

Measure of how much faster the computation executes versus the best serial code

- Serial time divided by parallel time:

$$S = Ts/Tp$$

Example: *Painting a picket fence*

- 30 minutes of preparation (serial)
- One minute to paint a single picket
- 30 minutes of cleanup (serial)

Thus, 300 pickets takes 360 minutes (serial time)

Computing Speedup

Number of painters	Time	Speedup
1	30 + 300 + 30 = 360	1.0X
2	30 + 150 + 30 = 210	1.7X
10	30 + 30 + 30 = 90	4.0X
100	30 + 3 + 30 = 63	5.7X
Infinite	30 + 0 + 30 = 60	6.0X

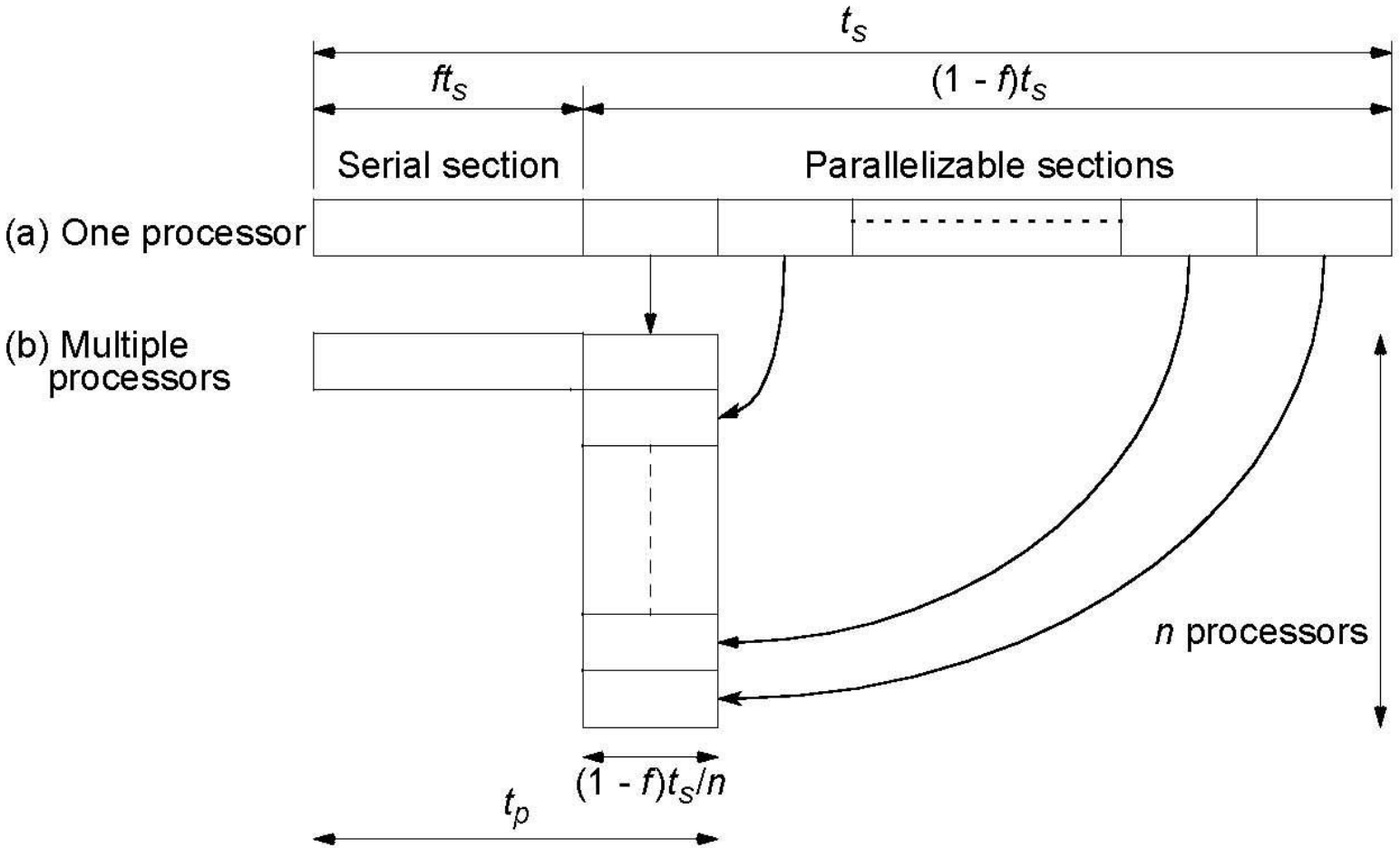
Illustrates Amdahl's Law

Potential speedup is restricted by serial portion

What if fence owner uses spray gun to paint 300 pickets in one hour?

- Better serial algorithm
- If no spray guns are available for multiple workers, what is maximum parallel speedup?

Massimo Speed-up – Legge di Amdahl



Il fattore di Speed-up è dato da:

$$S(n) = \frac{t_s}{ft_s + (1-f)t_s/n} = \frac{n}{1+(n-1)f}$$

Questa equazione è nota come Legge di Amdahl

Legge di Amdahl

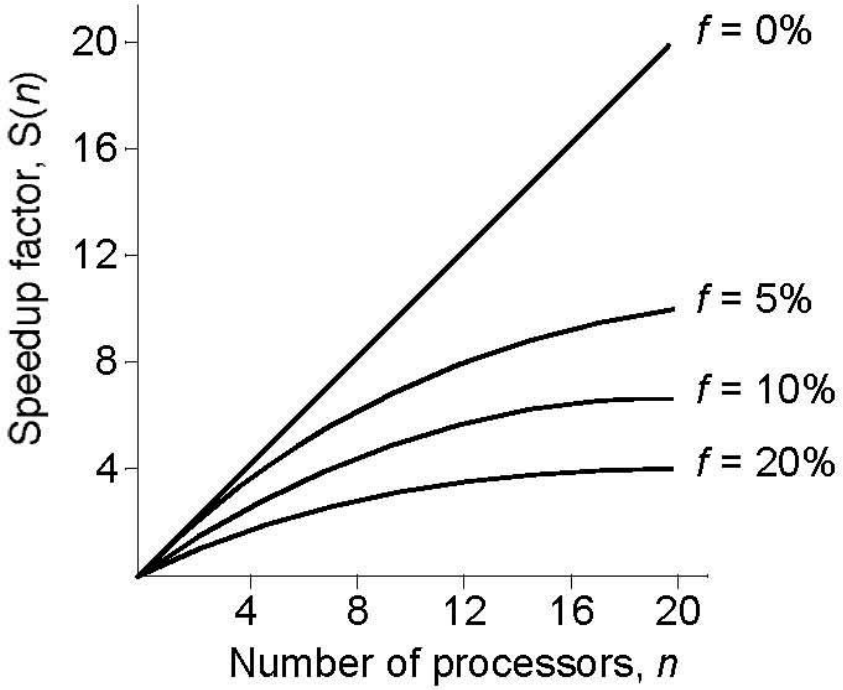
- Quindi, per n $\rightarrow \infty$:

$$S(n) \rightarrow \frac{1}{f}$$

- Ad esempio, se la **frazione seriale** f = 5% (plausibilissimo!) il massimo speedup è 20!

- NB La **frazione seriale** rappresenta quella “frazione” di codice che **non** può essere **parallelizzato** (es. I/O, sezioni critiche, etc)

Massimo Speed-up – Legge di Amdahl



Anche con un numero infinito di processori, il massimo speed-up è limitato a 1/f.
Esempio: Anche con solo il 5% di computazione “seriale”, il massimo speed-up e 20!

Speedup : Maggiore di p?

Una possibile ragione di **speedup superlineare** è che presumibilmente la versione parallela fa "meno" lavoro della corrispondente versione sequenziale

Algoritmo seriale = 14 t_c

Algoritmo parallelo = 5 t_c

Speedup = 14 t_c / 5 t_c = 2.8

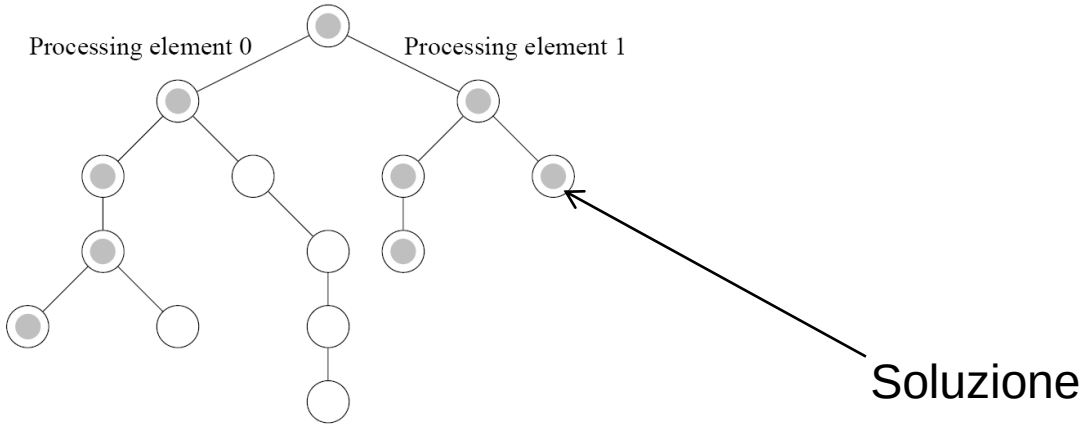
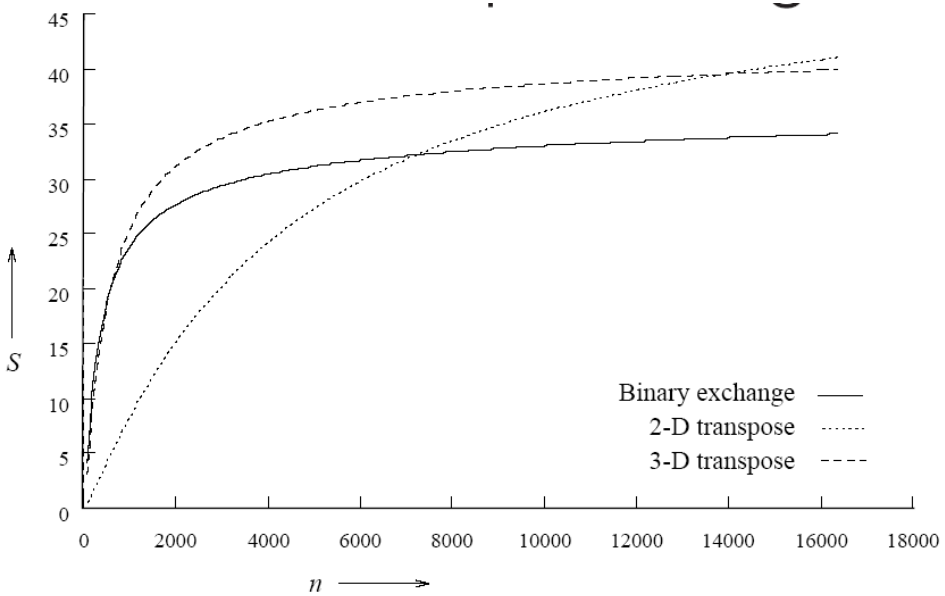


Figure 5.3 Searching an unstructured tree for a node with a given label, 'S', on two processing elements using depth-first traversal. The two-processor version with processor 0 searching the left subtree and processor 1 searching the right subtree expands only the shaded nodes before the solution is found. The corresponding serial formulation expands the entire tree. It is clear that the serial algorithm does more work than the parallel algorithm.

Scalabilità

Come facciamo ad **estrapolare** le **performance** da **piccoli problemi** e **piccoli sistemi** a problemi **piu' grandi** su configurazioni **maggiori**?

Il seguente esempio mostra l'andamento dello speed-up di tre algoritmi per di un algoritmo **Fast Fourier Transform (FFT)** ad ***n***-punti su 64 processori



Per piccoli valori di ***n***, sembrerebbe che gli algoritmi **Binary-exchange** e **3-D transpose** siano i **migliori**, ma per ***n*** >18000, l'algoritmo 2-D transpose permette uno speedup migliore.

Morale: Risulta **difficile inferire** caratteristiche di scalabilità dalle osservazioni su dati e macchine "**piccole"**

Scalable problem

- Problems that increase the percentage of parallel time with their size are more ***scalable*** than problems with a fixed percentage of parallel time
- Example:
  - Increase the problem size by doubling the grid dimensions and halving the time step.
    - four times the number of grid points
    - twice the number of time steps
  - Four times the number of operations
  - Expect the parallel portion of the program to perform better than the serial portion
  - Increase performance by increasing the problem size

2D Grid Calculations	85 sec	85%
Serial fraction	15 sec	15%
2D Grid Calculations	680 sec	97.84%
Serial fraction	15 sec	2.16%

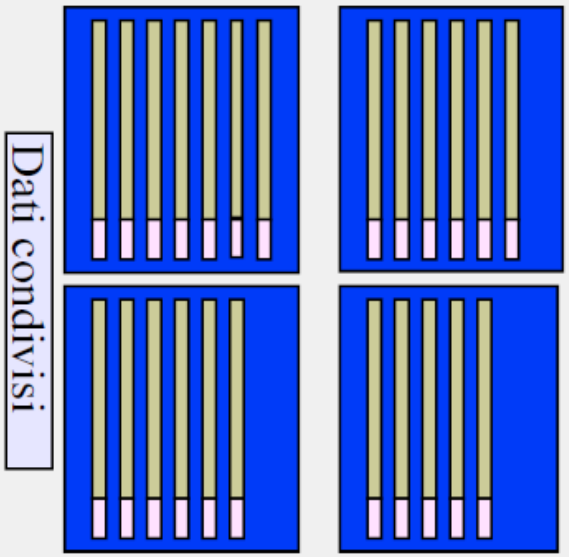
HPCC

Parallel Computing:

I passi per parallizzare una applicazione.

Tasks + dati condivisi e locali

Raggrupamento...



Esecuzione mappata su più cpu

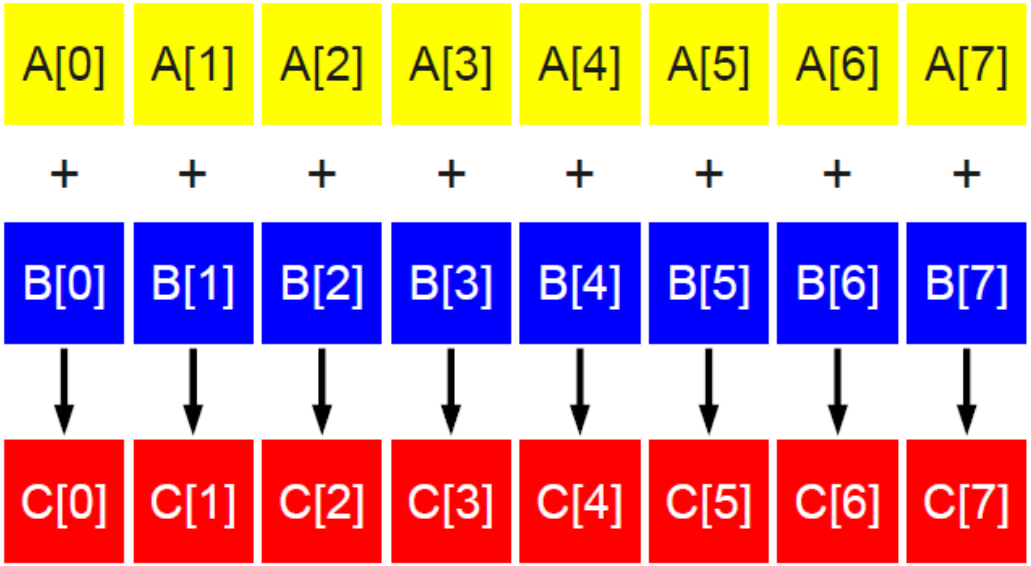
Codifica in un
linguaggio parallelo

```
Program SPMD_Emb_Par ()
{
  Program SPMD_Emb_Par ()
  {
    Program SPMD_Emb_Par ()
    {
      Program SPMD_Emb_Par ()
      {
        TYPE *tmp, *func();
        global_array Data(TYPE);
        global_array Res(TYPE);
        int Num = get_num_procs();
        int id = get_proc_id();
        if (id==0) setup_problem(N, Data);
        for (int I= ID; I<N;I=I+Num){
          tmp = func(I, Data);
          Res.accumulate( tmp);
        }
      }
    }
  }
}
```

Il nuovo codice sorgente

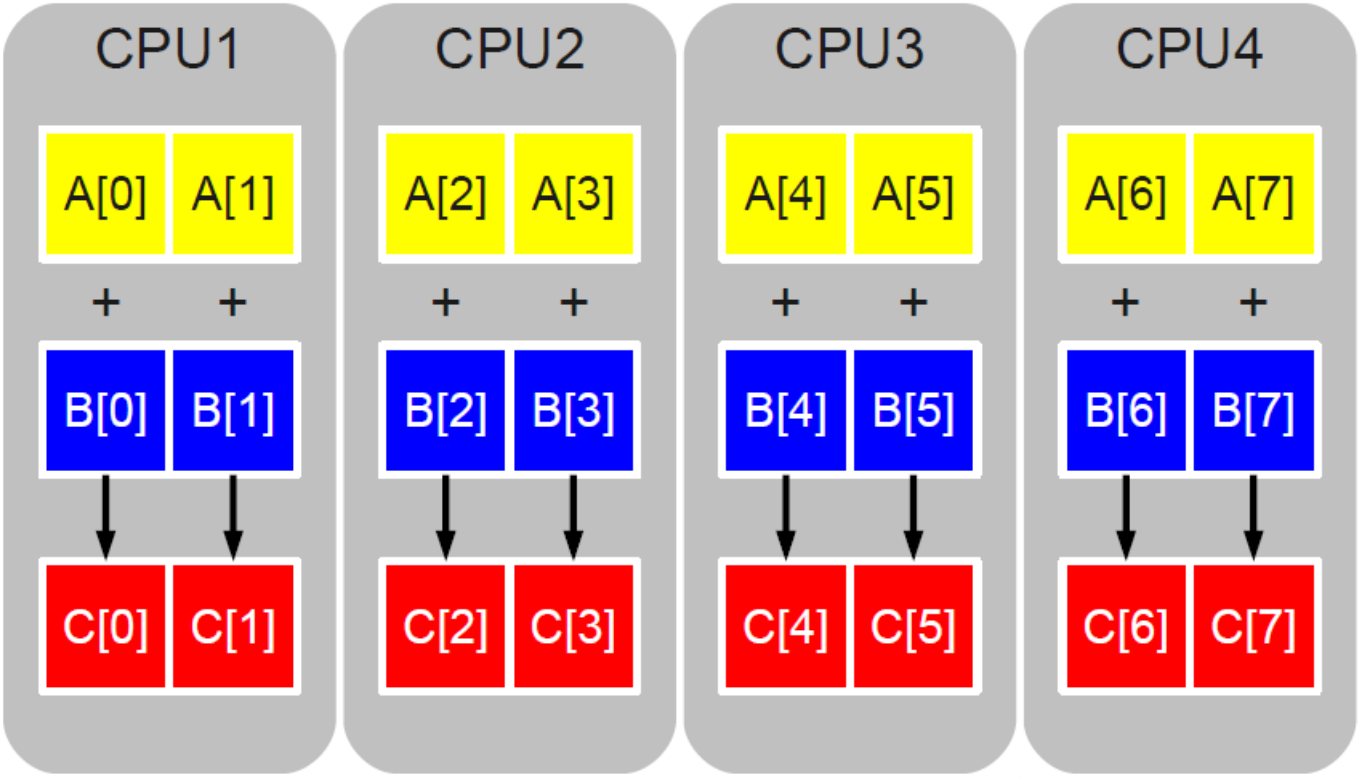
Esempio

- Somma elemento per elemento di due vettori




```
void somma_seq(float* A, float* B, float* C, size_t n)
{
    size_t i;
    for (i=0; i<n; ++i)
        C[i] = A[i] + B[i];
}
```

Somma Parallela



Un esempio di codice parallelo

For-loop with independent iterations

```
for (int i=0; i<n; i++)
  c[i] = a[i] + b[i];
```

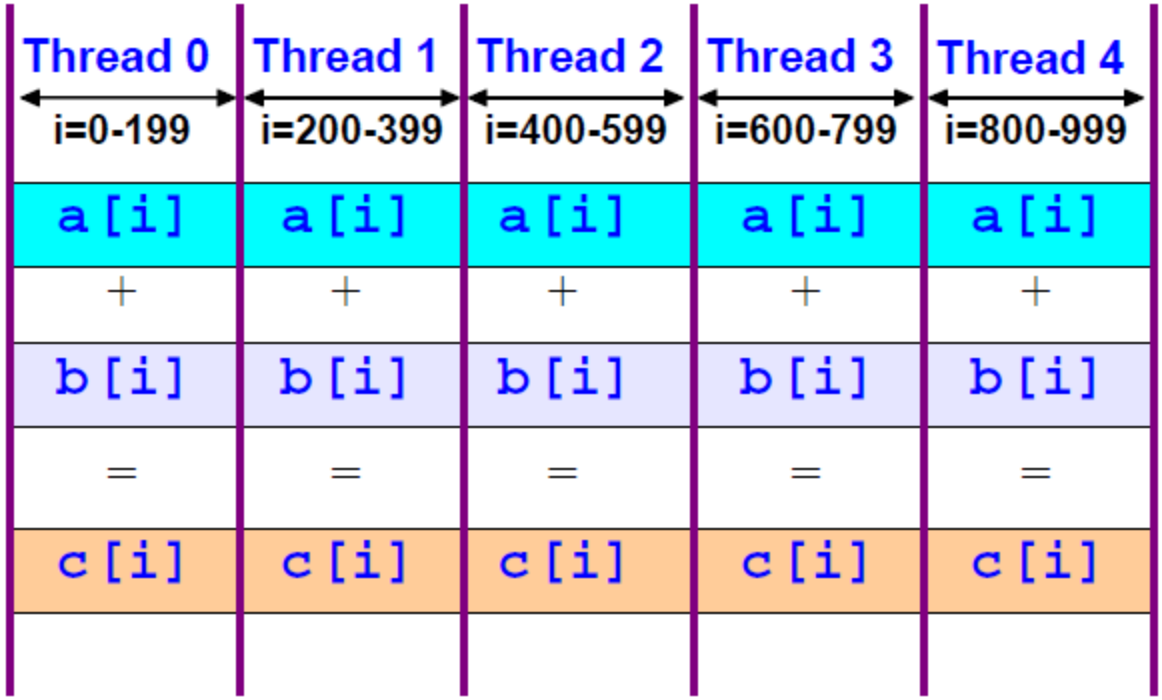
Versione Seriale

For-loop parallelized using an OpenMP pragma

```
#pragma omp parallel for
for (int i=0; i<n; i++)
    c[i] = a[i] + b[i];
```

Versione Parallela usando la libreria OpenMP

… diventa

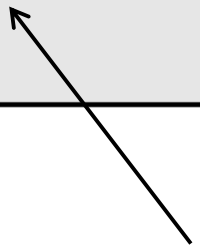

Thread 0
i=0-199
Thread 1
i=200-399
Thread 2
i=400-599
Thread 3
i=600-799
Thread 4
i=800-999
a[i]
+
b[i]
=
c[i]

Non è sempre così semplice!

- Somma di tutti gli elementi di un vettore

Soluzione SBAGLIATA!!

```
float somma_vec(float* A, size_t n)
{
    size_t i;
    float s=0;
#pragma omp parallel for
    for (i=0; i<n; ++i)
        s = s + A[i];
    return s;
}
```



Attenzione! La variabile s è acceduta contemporaneamente da piu processori!
Torneremo su questo concetto tra poco….

Le sezioni Critiche

If sum is a shared variable, this loop can not run in parallel

```
for (i=0; i < n; i++){
      .....
   sum += a[i];
      .....
}
```

Sezione Critica:
Un processore alla volta esegue il calcolo

We can use a critical region for this:

```
for (i=0; i < n; i++){
      .....                    one at a time can proceed
   sum += a[i];
      .....                    next in line, please
}
```

Versione con sezione critica

- Considerate questo segmento di programma in linguaggio C (sequenziale) per calcolare π usando la regola di rettangolo:

```
double area, pi, x;
int i, n;
...
area = 0.0;
for (i = 0; i < n; i++) {
   x = (i+0.5)/n;
   area = area + 4.0/(1.0 + x*x);
}
pi = area / n;
```

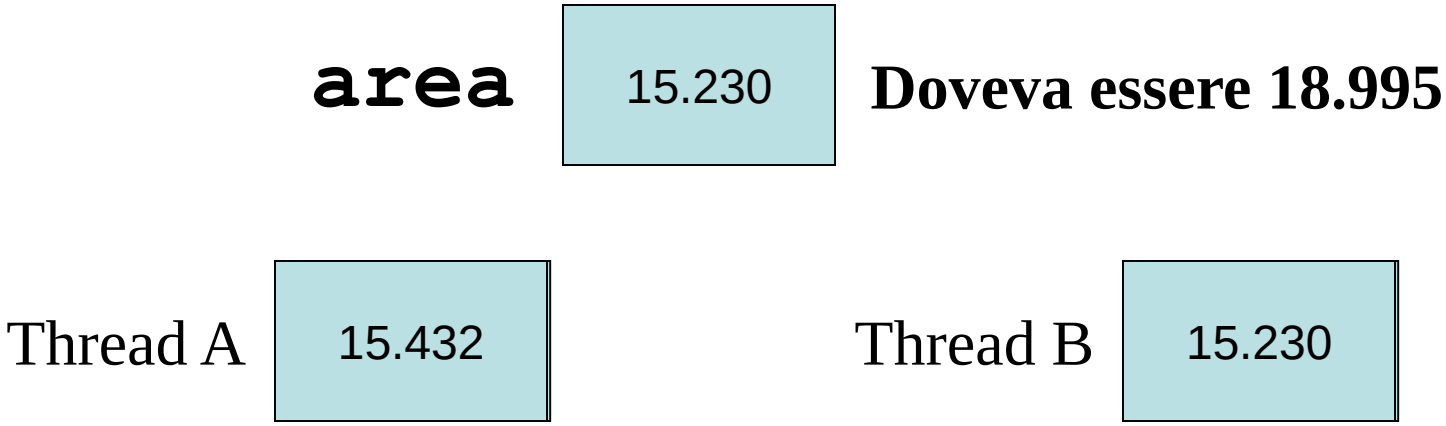
Versione con sezione critica

- Se parallelizziamo il ciclo con OpenMP...

```
double area, pi, x;
int i, n;
...
area = 0.0;
#pragma omp parallel for private(x)
for (i = 0; i < n; i++) {
   x = (i+0.5)/n;
   area = area + 4.0/(1.0 + x*x);
}
pi = area / n;
```

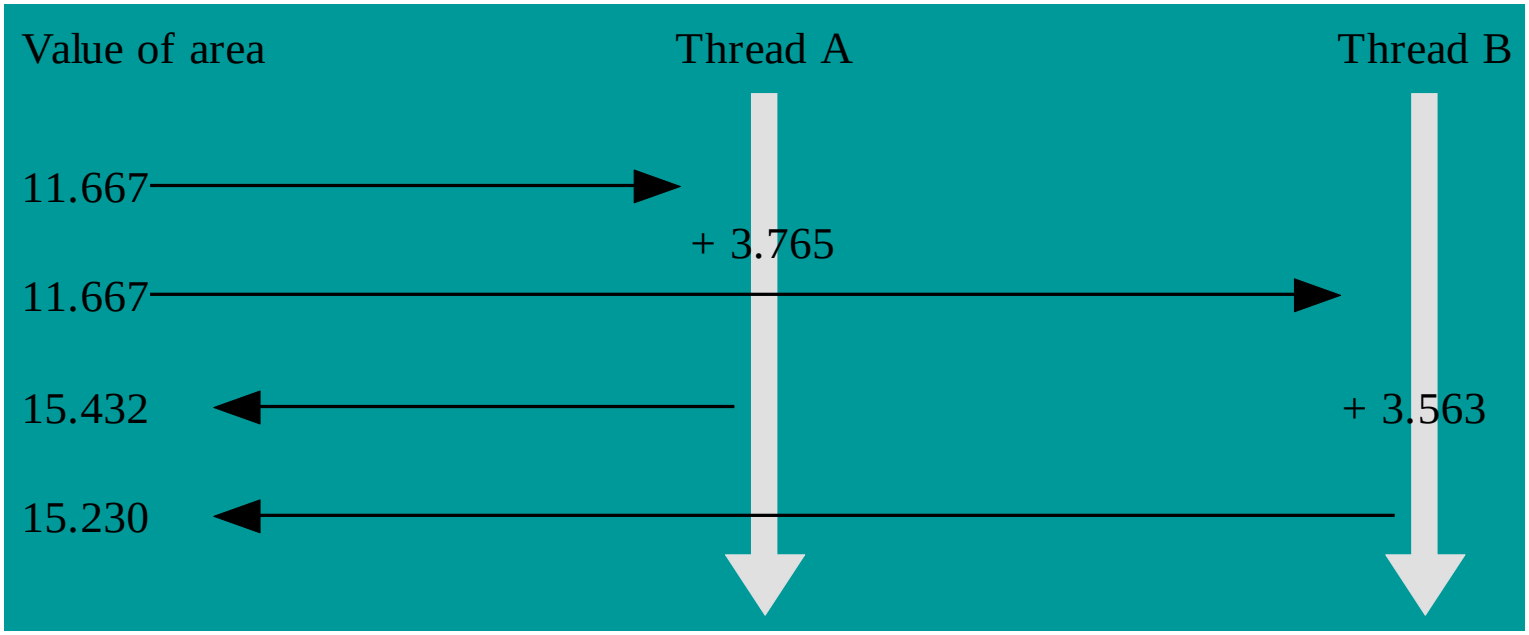
Race Condition

- Problema: abbiamo istituito una «condizione di competizione» (race condition) in cui un processo può "correre avanti" di un altro e non vedere il cambiamento di una variabile condivisa



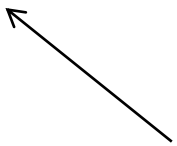

```
area = area + 4.0/(1.0 + x*x)
```

Race Condition Time Line



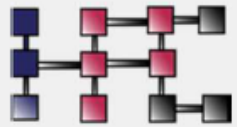
Corretto

```
double area, pi, x;
int i, n;
...
area = 0.0;
#pragma omp parallel for private(x)
for (i = 0; i < n; i++) {
   x = (i+0.5)/n;
#pragma omp critical
   area = area + 4.0/(1.0 + x*x);
}
pi = area / n;
```



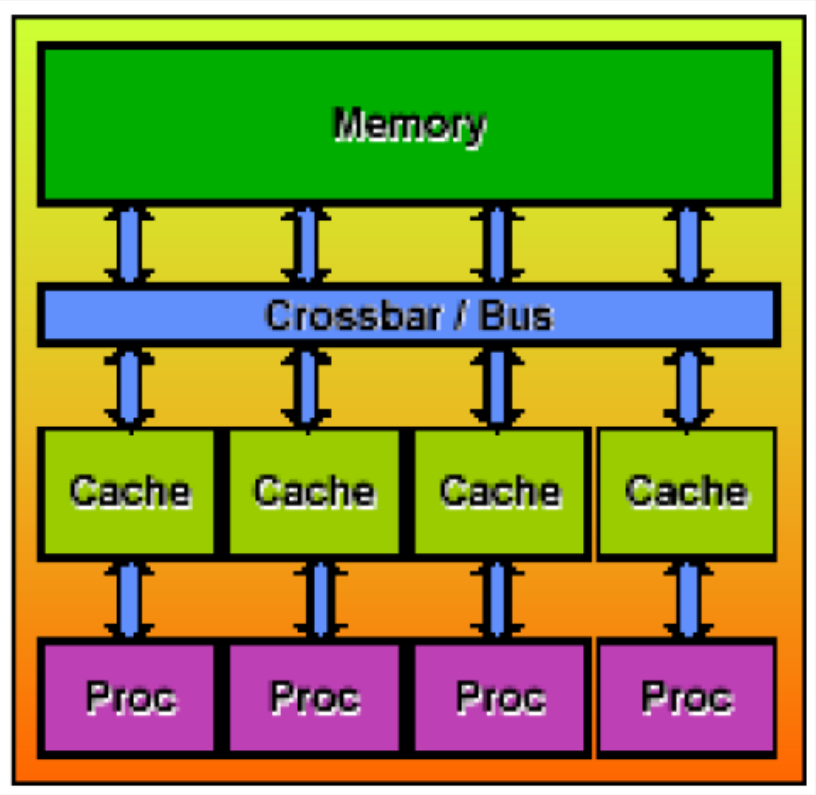
Sezione Critica:
Un processore alla volta esegue il calcolo

OpenMP: Il primo linguaggio parallelo «automatico»

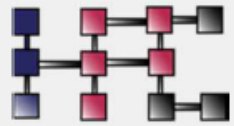


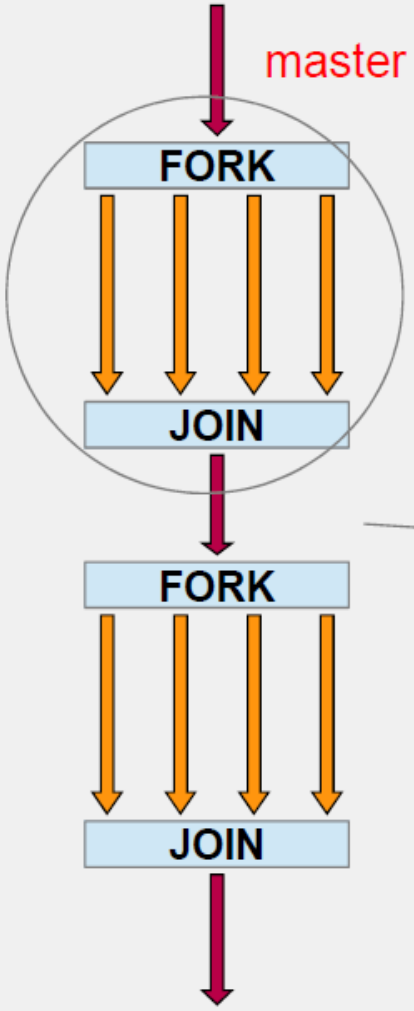
- OpenMP:
  Open specifications for Multi Processing
- Impiegato per sviluppare programmi in ambienti a memoria condivisa
- Consiste principalmente di un insieme di direttive di compilazione
- Richiede uno sforzo minimo per essere implementato

É UNO STANDARD DAL 1997



Metodo Fork-Join





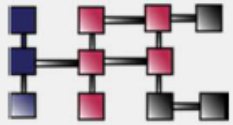
Esecuzione in sequenziale

oppure Parallel Region

Regione PARALLELA
Si genera un **pool di thread**
che eseguono il task

Esecuzione in seriale
tra due **Regioni PARALLELE**
ATTENZIONE

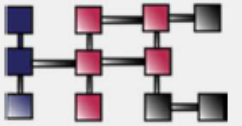
Concetti base



- Il parallelismo in OpenMP si esplica tramite thread
  - concetto di **thread**:
    - scomposizione del processo in blocchi di codice indipendente
    - ogni thread ha program counter, stack e register set propri
  - la parte dati del processo e lo spazio degli indirizzi sono condivisi fra le thread
  - anche i file aperti dal processo sono condivisi

Worksharing

OpenMP ed i loop



- L' OpenMP è usato generalmente per parallelizzare i loop
  - si trova il loop da parallelizzare
  - Si inserisce la direttiva per il worksharing

```
for(int i=1;i<100;i++) {

    a[i] = b[i] + c[i];

}
```

Programma sequenziale

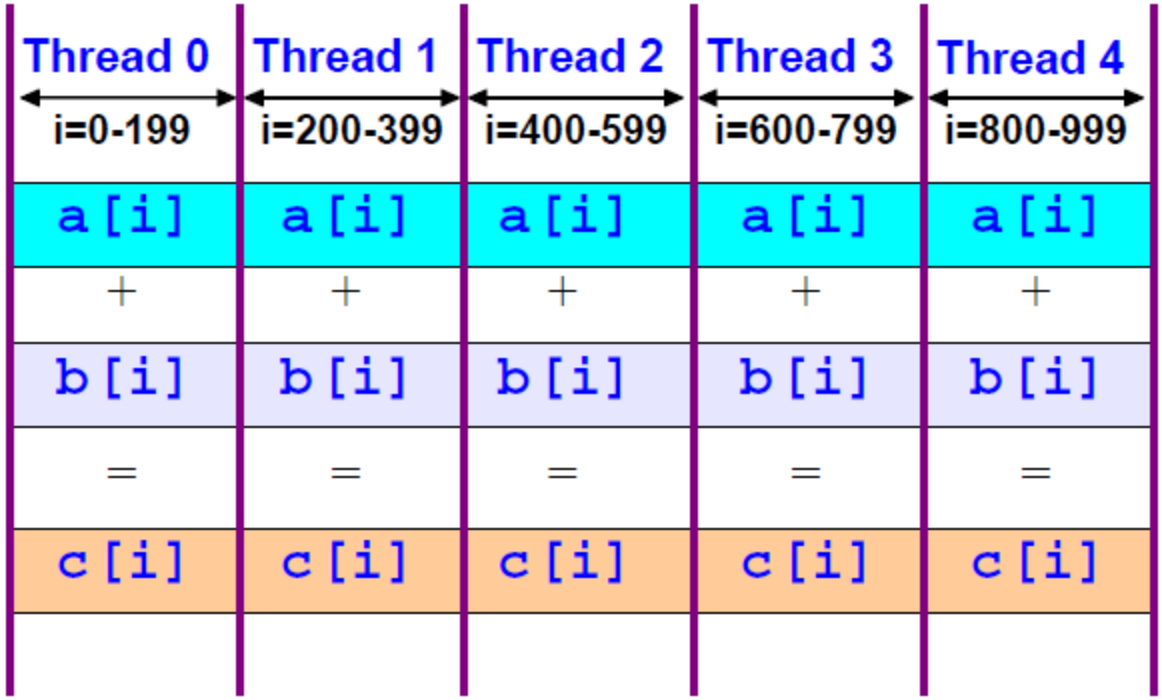
```
#pragma omp parallel for
for(int i=1;i<100;i++) {

    a[i] = b[i] + c[i];

}
```

Programma parallelo

Da Prima


Thread 0
i=0-199
Thread 1
i=200-399
Thread 2
i=400-599
Thread 3
i=600-799
Thread 4
i=800-999
a[i]
a[i]
a[i]
a[i]
a[i]
+
+
+
+
+
b[i]
b[i]
b[i]
b[i]
b[i]
=
=
=
=
=
c[i]
c[i]
c[i]
c[i]
c[i]

pthreads? no, thanks!

```
int a, b;
main() {
  // serial segment
  #pragma omp parallel num_threads (8) private (a) shared (b)
  {
    // parallel segment
  }
  // rest of serial segment
}
```

Sample OpenMP program

```
int a, b;
main() {
  // serial segment

  for (i = 0; i < 8; i++)
      pthread_create (......., internal_thread_fn_name, ...);

  for (i = 0; i < 8; i++)
      pthread_join (.......);

  // rest of serial segment

}

void *internal_thread_fn_name (void *packaged_argument) [
    int a;

  // parallel segment

}
```

Code inserted by the OpenMP compiler

Corresponding Pthreads translation

Comparison between OpenMP and the corresponding program written with Pthreads

Ciao Mondo! (Hello World!)

```
#include <omp.h>
main () {
int nthreads, tid;

/* Fork a team of threads with each thread having a private tid
variable */
#pragma omp parallel private(tid)
  { /* Obtain and print thread id */
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);

  /* Only master thread does this */
  if (tid == 0)
  {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
  }
} /* All threads join master thread and terminate */ }
```

To compile:

`gcc -fopenmp hello.c -o hello`

Example

```
#pragma omp parallel
printf("Hello from %d\n", omp_get_thread_num() );
```

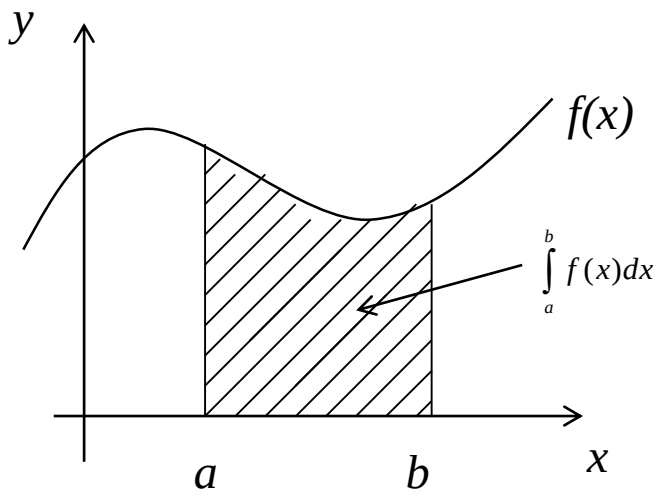
With 5 threads

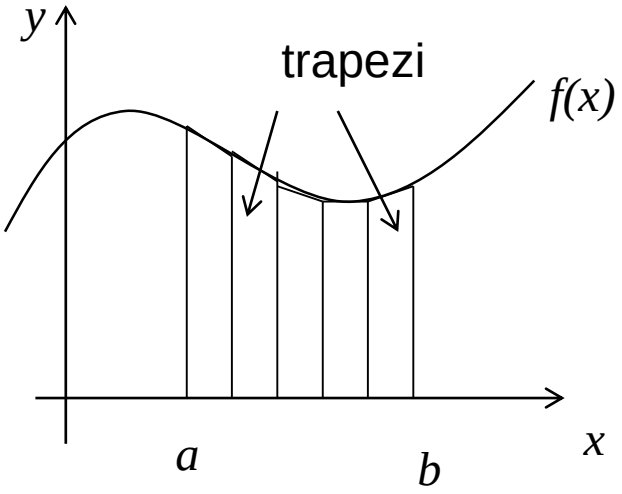
```
<bonaccor@poseidon ~/OMP-EX>./WriteThrNumC
Hello from 1
Hello from 2
Hello from 0
Hello from 3
Hello from 4
```

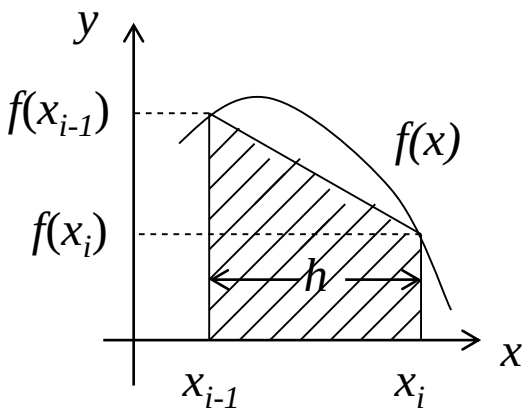
With 2 threads

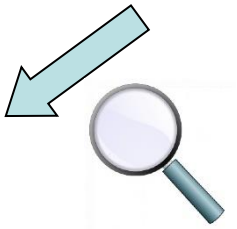
```
<bonaccor@poseidon ~/OMP-EX>./WriteThrNumF
Hello from         0
Hello from         1
```

Esempio: Integrazione Numerica

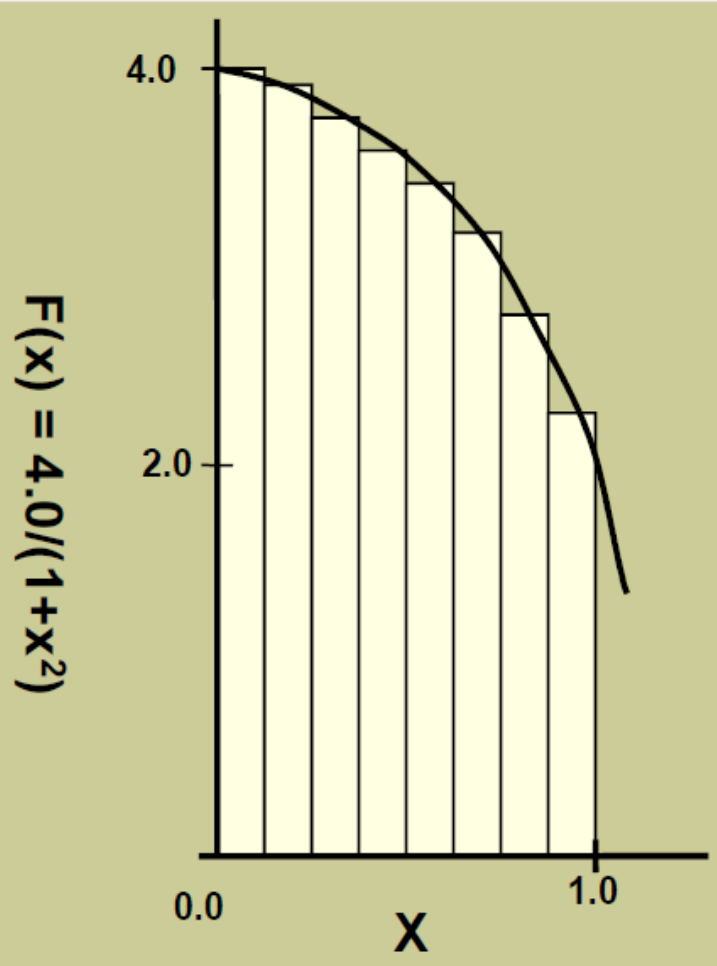








Un esempio:
Calcolo del valore di π



Sappiamo che:

$$\int \frac{4}{1+x^2} dx = \pi$$

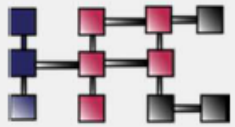
e si può approssimare l'integrale con una somma di rettangoli:

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

ogni rettangolo ha base Δx e altezza $F(x_i)$ in mezzo all'intervallo *i*esimo

Calcolo del valore di π

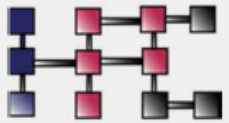
Integrazione numerica



- Usiamo un algoritmo di integrazione numerica per calcolare il valore di π
- Per parallelizzare il programma usando OpenMP si possono seguire 2 strade:
  - usare il costrutto parallel region
  - usare un costrutto work-sharing

Calcolo del valore di π

il programma sequenziale



```
static long num_steps = 100000;
double step;
void main ()
{      int i;
  double x, pi, sum = 0.0;
  step = 1.0/(double) num_steps;

  for (i=1;i<= num_steps; i++){
        x = (i-0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
  }
  pi = step * sum;
}
```

N

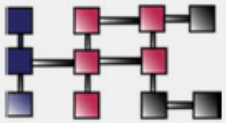
Δx

x_i

$$sum = \sum_{i=0}^{N} \frac{4}{1+x_i^2}$$

Calcolo del valore di π

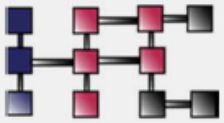
esemio d'uso di parallel region



```
#include <omp.h>
static long num_steps = 8;
double step;
#define NUM_THREADS 2
void main ()
{       int i;        double x, pi, sum[NUM_THREADS] ={0};
  step = 1.0/(double) num_steps;
  omp_set_num_threads(NUM_THREADS);
#pragma omp parallel private(x, id, i)
{       double x;     int id, i;
  id = omp_get_thread_num();
                int nthreads = omp_get_num_threads();
  for (i=id;i< num_steps; i=i+nthreads){
        x = (i+0.5)*step;
        sum[id] += 4.0/(1.0+x*x);
  }
}
  for(i=0, pi=0.0;i<NUM_THREADS;i++)pi += sum[i] * step;
}
```

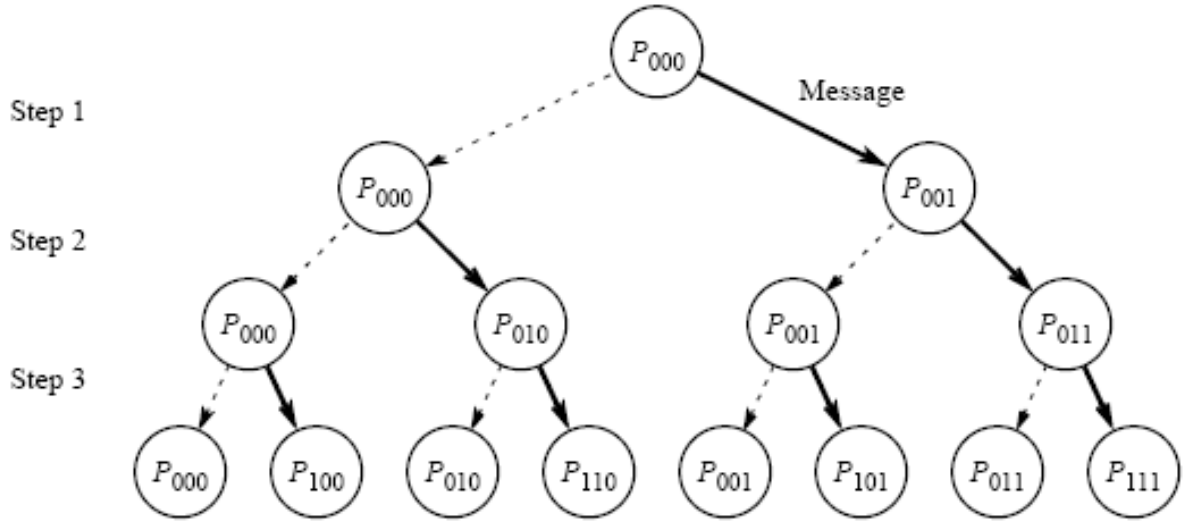
Calcolo del valore di π

esempio d'uso di un costrutto worksharing



```
#include <omp.h>
static long num_steps = 8;        double step;
#define NUM_THREADS 2
void main ()
{       int i, id;
       double x, pi, sum[NUM_THREADS] ={0.0};
  step = 1.0/(double) num_steps;
  omp_set_num_threads(NUM_THREADS);
#pragma omp parallel private(i, id, x)
{
  id = omp_get_thread_num();
#pragma omp for
  for (i=0;i< num_steps; i++){
        x = (i+0.5)*step;
        sum[id] += 4.0/(1.0+x*x);
  }
}       for(i=0, pi=0.0;i<NUM_THREADS;i++)pi += sum[i] * step;
}
```

Ottimizzazione delle sezioni critiche

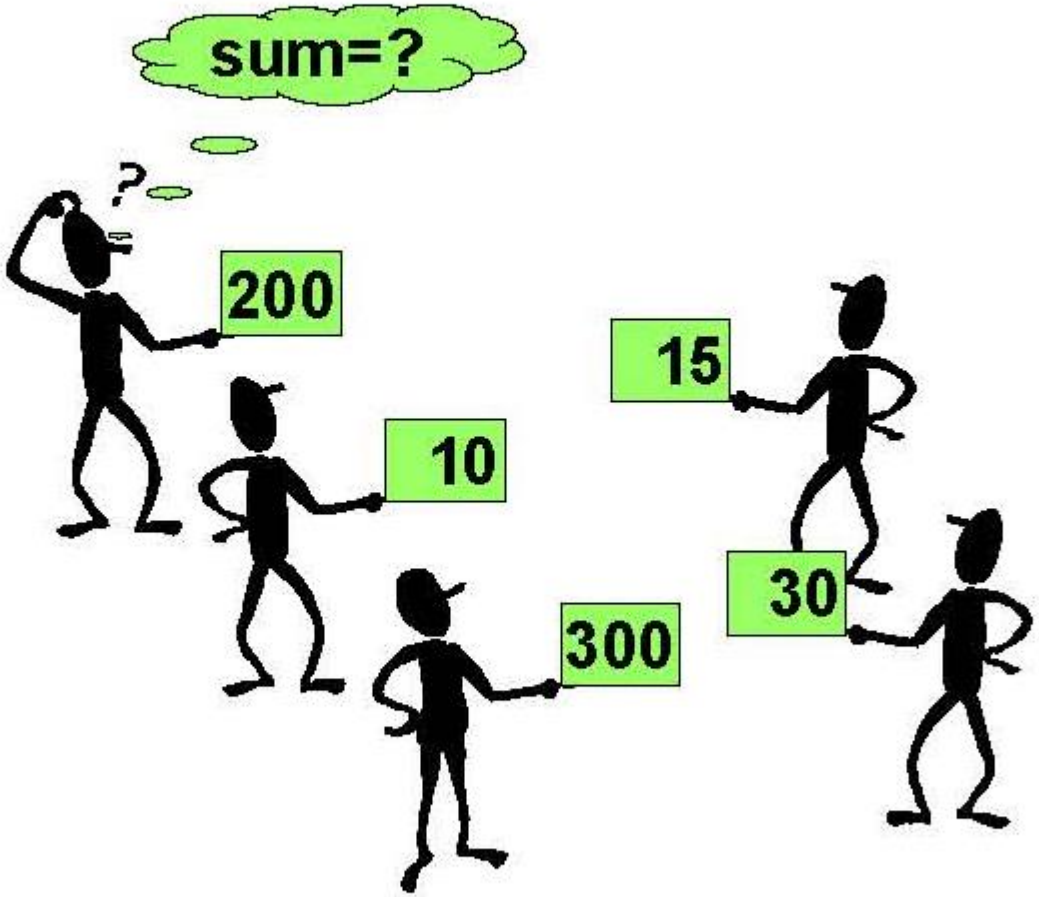


Se molti processori sono coinvolti, la sezione critica puo' diventare un collo di bottiglia!

Solitamente, si utilizzano tecniche di ottimizzazione, come le **RIDUZIONI**, che impiega in questo caso $O(\log n)$ passi

Reduction Operations

- Combiniamo dati da diversi processori per produrre un singolo risultato



Riduzioni con OpenMP

- Somma di tutti gli elementi di un vettore

Soluzione corretta

```
float somma_vec(float* A, size_t n)
{
    size_t i;
    float s=0;
#pragma omp parallel for reduction(+:s)
    for (i=0; i<n; ++i)
        s = s + A[i];
    return s;
}
```

Esempio: Calcolare π con le Riduzioni

Monte Carlo Calculations:

L'utilizzo di numeri random per stimare aree, calcolare probabilità, trovare valori ottimali, etc

Esempio: Calcolo di π con il metodo delle «freccette»

- Tirare freccette nel cerchio/quadrato
- La probabilità che una freccetta cada nel cerchio e proporzionale al rapporto tra le aree:

$A_c = r^2 * \pi$

$A_s = (2*r) * (2*r) = 4 * r^2$

$P = A_c/A_s = \pi /4$

- Calcolare π tramite la scelta random di punti, contare la frazione che cade nel cerchio, computare π

N= 10	π = 2.8
N=100	π = 3.16
N= 1000	π = 3.148

Parallel Programmers love Monte Carlo algorithms

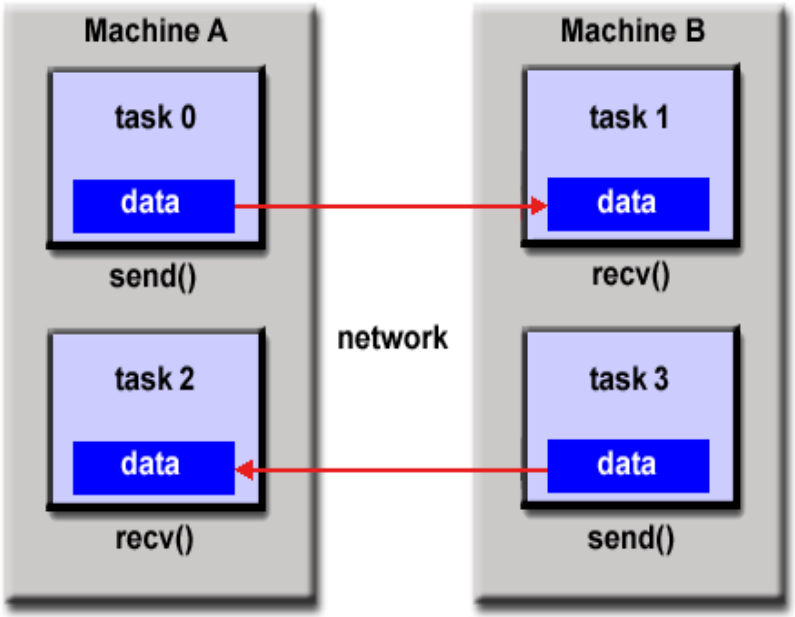
```
#include “omp.h”
static long num_trials = 10000;
int main ()
{
  long i;     long Ncirc = 0;      double pi, x, y;
  double r = 1.0;   // radius of circle. Side of squrare is 2*r
  seed(0,-r, r);  // The circle and square are centered at the origin
  #pragma omp parallel for private (x, y) reduction (+:Ncirc)
  for(i=0;i<num_trials; i++)
  {
    x = random();        y = random();
    if ( x*x + y*y) <= r*r)   Ncirc++;
   }

   pi = 4.0 * ((double)Ncirc/(double)num_trials);
   printf("\n %d trials, pi is %f \n",num_trials, pi);
}
```

Parallelismo perfetto: Basta aggiungere una riga di codice e si ottiene un programma parallelo!

Modello Message Passing

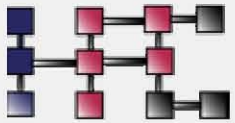
1. Un insieme di task che utilizzano la propria memoria durante la computazione. Task multipli possono risiedere sulla stessa macchina fisica e su differenti macchine
2. I tasks si scambiano data attraverso comunicazioni, spedendo e ricevendo messaggi.
3. Il trasferimento di dati solitamente richiede operazioni cooperative per ogni processo. Ad esempio, una operazione send deve avere una receive corrispondente



Da un punto di vista del programmatore, le implementazioni message passing solitamente includono una libreria di funzioni che sono inglobati nel codice. Il programmatore è responsabile nella determinazione del parallelismo.

MPI (Message Passing Interface), standard de-facto per il modello the Message Passing

Le basi di MPI
SPMD



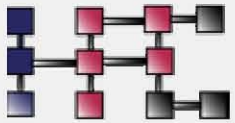
- E' raro che si scriva un programma differente per ogni processo di una applicazione parallela
- In molti casi si usa scrivere un SPMD (Single Program on Multiple Data)
  - lo stesso programma gira su tutti i processori che partecipano all'esecuzione dell'applicazione
  - ogni processo ha un attributo di identificazione chiamato *rank*

Message Passing Model

- Approccio Single program Multiple data (SMPD)

```
if (my_process_rank ==0)
   MPI_Send(&x, 1, MPI_FLOAT, 1, 0, MPI_COMM_WORLD);
else
   if (my_process_rank ==1)
     MPI_Recv(&x, 1, MPI_FLOAT, 0, 0, MPI_COMM_WORLD,
           &status);
```

MPI program in C



```
#include <unistd.h>
#include <mpi.h>
int main(int argc, char **argv) {
  int my_rank, n;
  char hostname[128];
  MPI_init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
  MPI_Comm_size(MPI_COMM_WORLD,&n);
  gethostname(hostname,128);
  if (my_rank == 0) { /* master */
    printf("Sono il master: %s\n",hostname);
  } else { /* worker */
    printf("Sono il worker: %s (rank=%d/%d)\n",
          hostname,my_rank,n-1);
  }
  MPI_Finalize();
  exit(0);
}
```

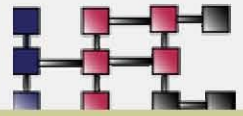
always include

Called first, only once!

Called last, only once!

Standard SEND e RECEIVE:

un esempio in C



```
#include <mpi.h>
int main(int argc, char **argv) {
  int i, my_rank, nprocs, x[4];
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
  if (my_rank == 0) { /* master */
    x[0]=42; x[1]=43; x[2]=44; x[3]=45;
    MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
    for (i=1;i<nprocs;i++)
          MPI_Send(x,4,MPI_INT, i ,0 ,MPI_COMM_WORLD);
  } else { /* worker */
    MPI_Recv(x,4,MPI_INT,0 ,0 ,MPI_COMM_WORLD,&status);
  }
  MPI_Finalize();
  exit(0);
}
```

numero di elementi da trasmettere

destinazione e sorgente

user-defined tag

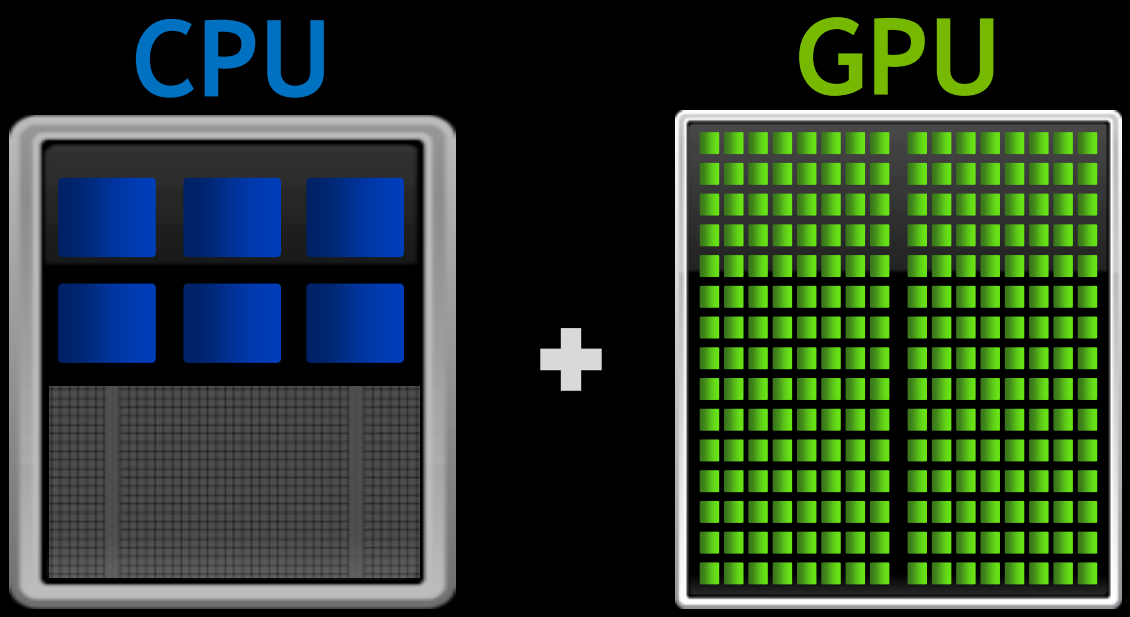
numero di elementi da recevere

Shared memory vs Distributed memory

- Memoria condivisa
- Vantaggi:
  - In generale, più “facile” da programmare
  - Vantaggioso per applicazioni che prevedono un accesso “irregolare” ai dati (esempio, algoritmi di esplorazione di grafi)
- Svantaggi:
  - Problemi di concorrenza/mutua esclusione
  - Banda di memoria limitata

- Memoria distribuita
- Vantaggi:
  - Accesso a potenze di calcolo molto elevate
  - Vantaggioso per applicazioni che esibiscono forte località di accesso ai dati, con elevato rapporto computazione / comunicazione
- Svantaggi:
  - Latenza della rete di comunicazione
  - In generale, più complesso da programmare

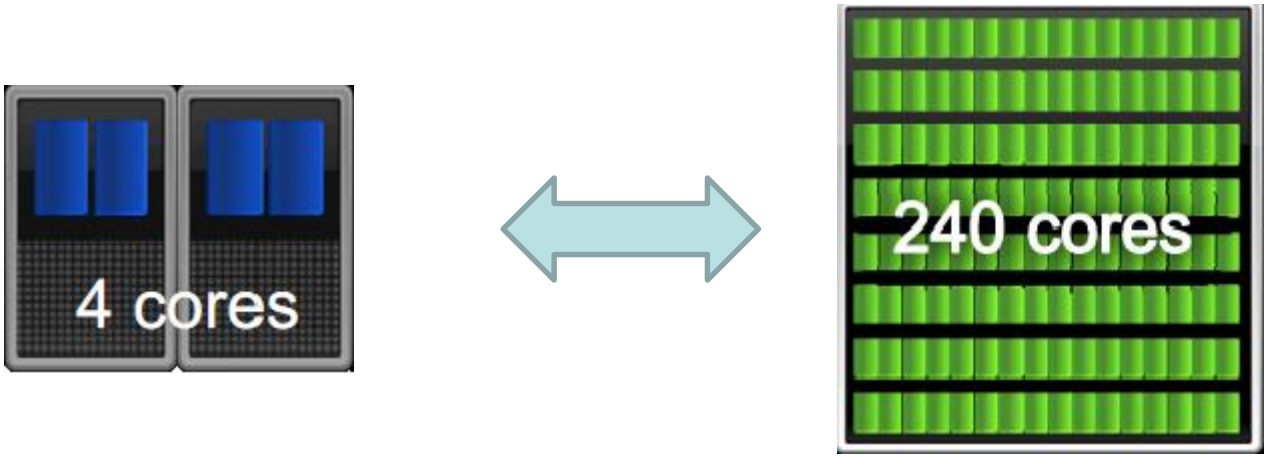
Heterogeneous computing

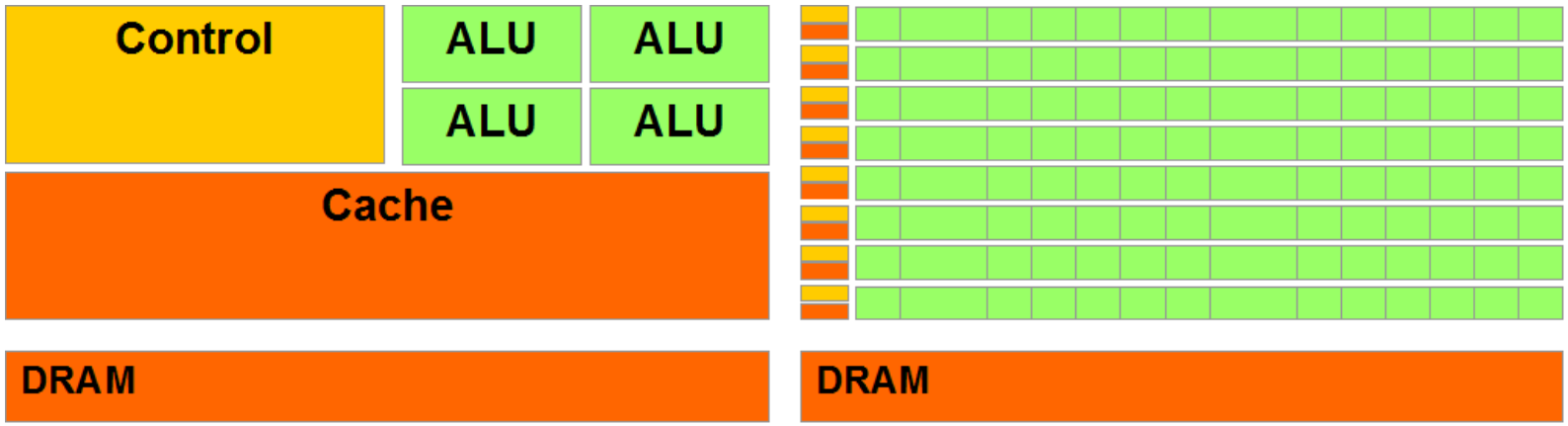


GPGPU and CUDA

- **GPGPU** (**General Purpose programing on GPUs**): Use of graphics processor units (GPU) for purposes other than the traditional three-dimensional creating processes
- GPUs are **high performance multi-core** processors
- The first programmable solutions date back to 2006, were previously dedicated only to the development of graphics and video games.
- GPUs are now considered as parallel processors with general-purpose programming interfaces with support for programming languages such as **CUDA-C** (nVidia) or **Stream** (ATI).
- **OpenCL** standard, framework for writing programs that execute across **heterogeneous platforms** consisting of central processing units (CPUs) and graphics processing units (GPUs),

Macroscopic differences
CPU / GPU





DISCOVER WITH PLYMOUTH UNIVERSITY

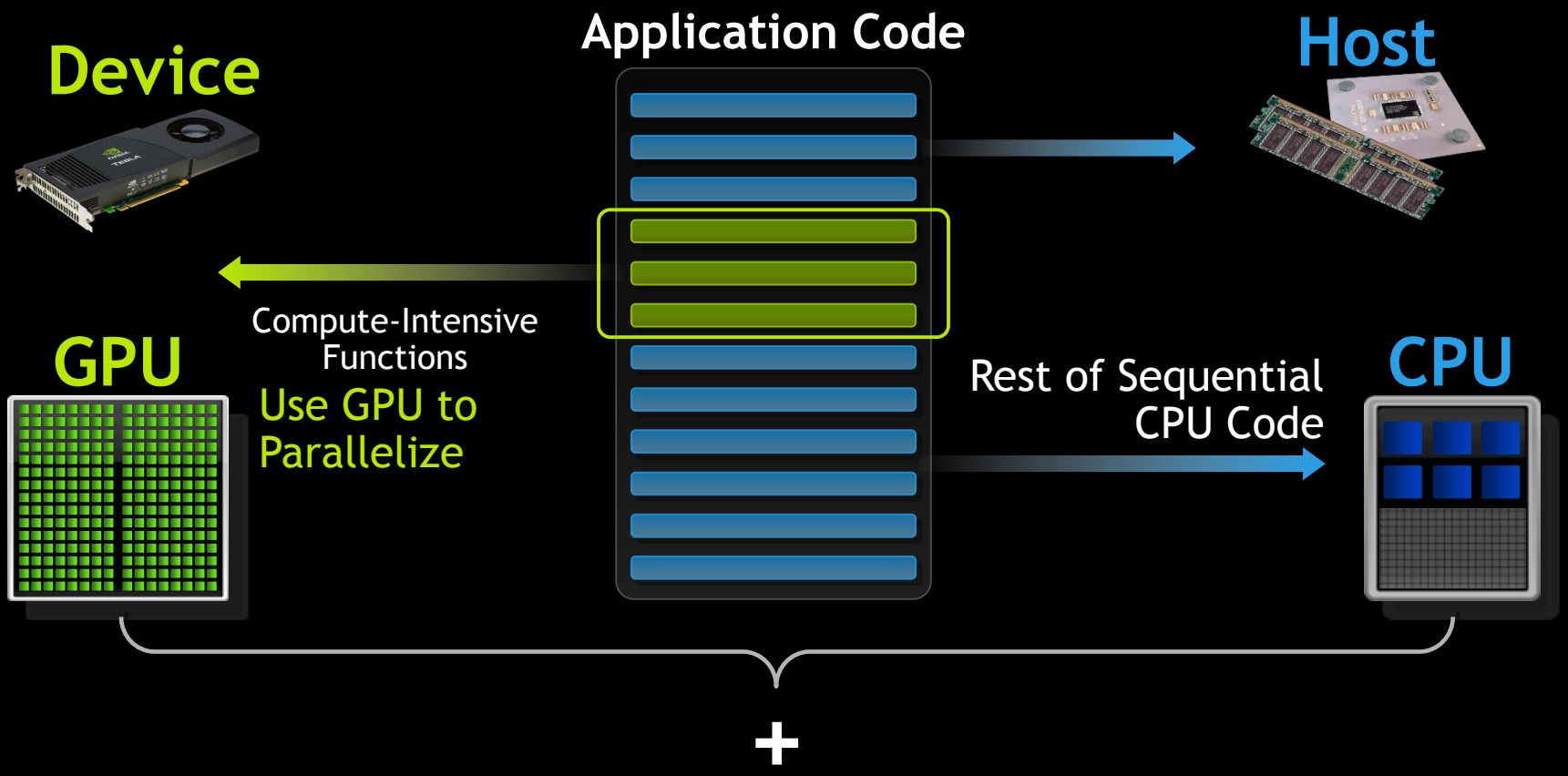
SM Multiprocessor

- 32 CUDA Cores per SM (512 total)
- Direct load/store to memory
  - Usual linear sequence of bytes
  - High bandwidth (Hundreds GB/sec)
- 64KB of fast, on-chip RAM
  - Software or hardware-managed
  - Shared amongst CUDA cores
  - Enables thread communication

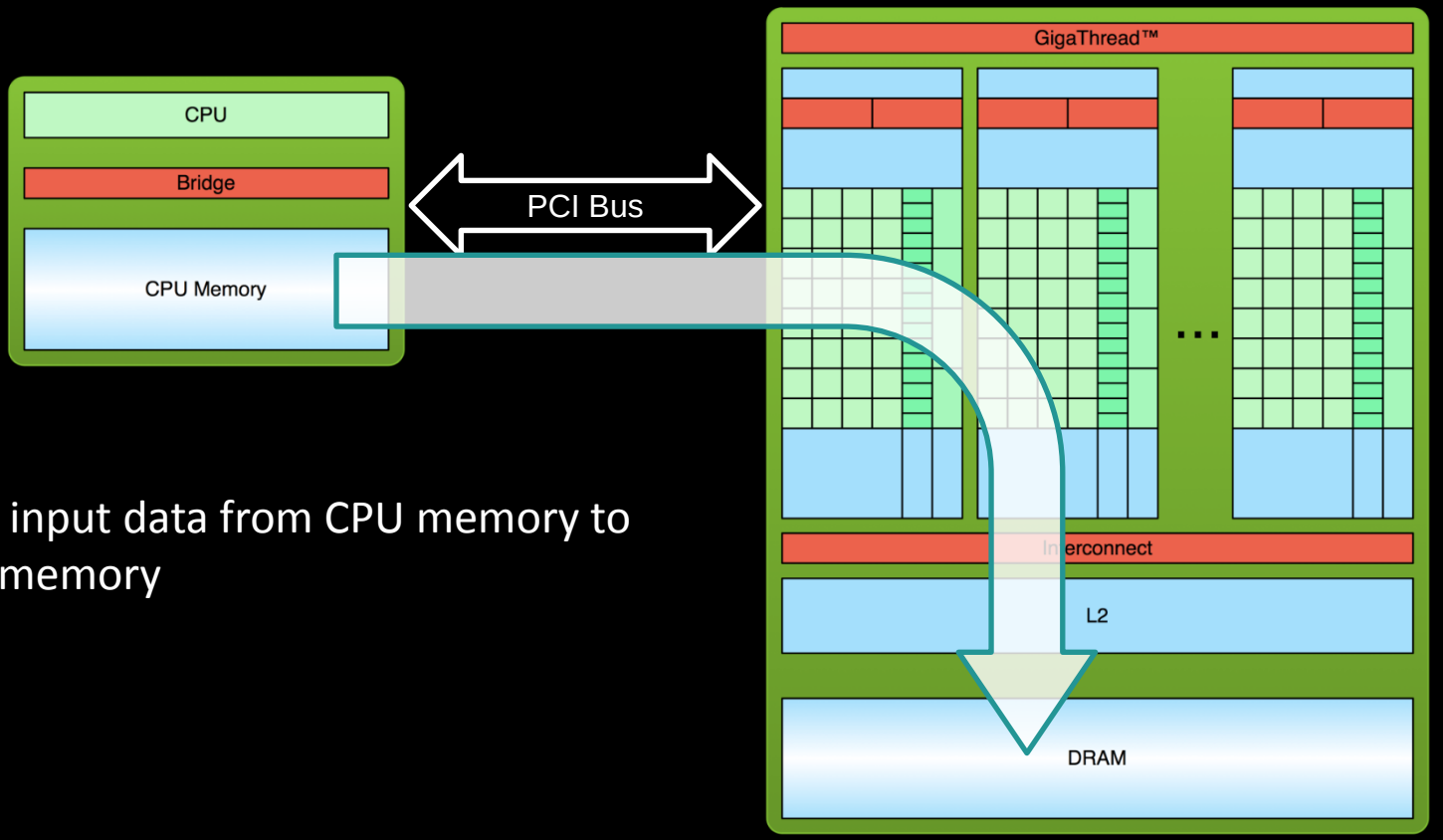
SIMT (Single Instruction Multiple Thread) execution

DISCOVER WITH PLYMOUTH UNIVERSITY

Heterogeneous computing



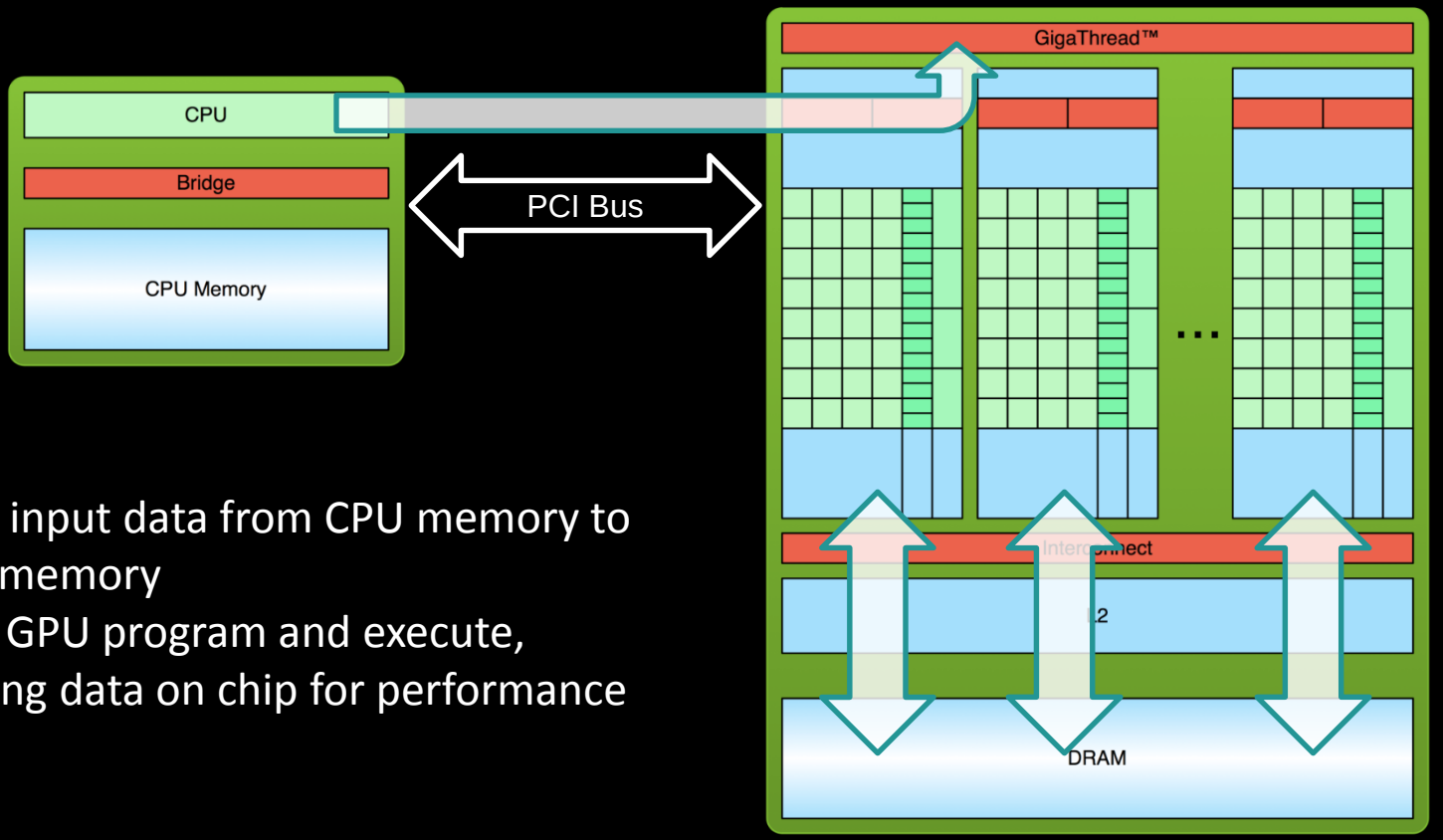
Simple Processing Flow



1. Copy input data from CPU memory to GPU memory

Simple Processing Flow

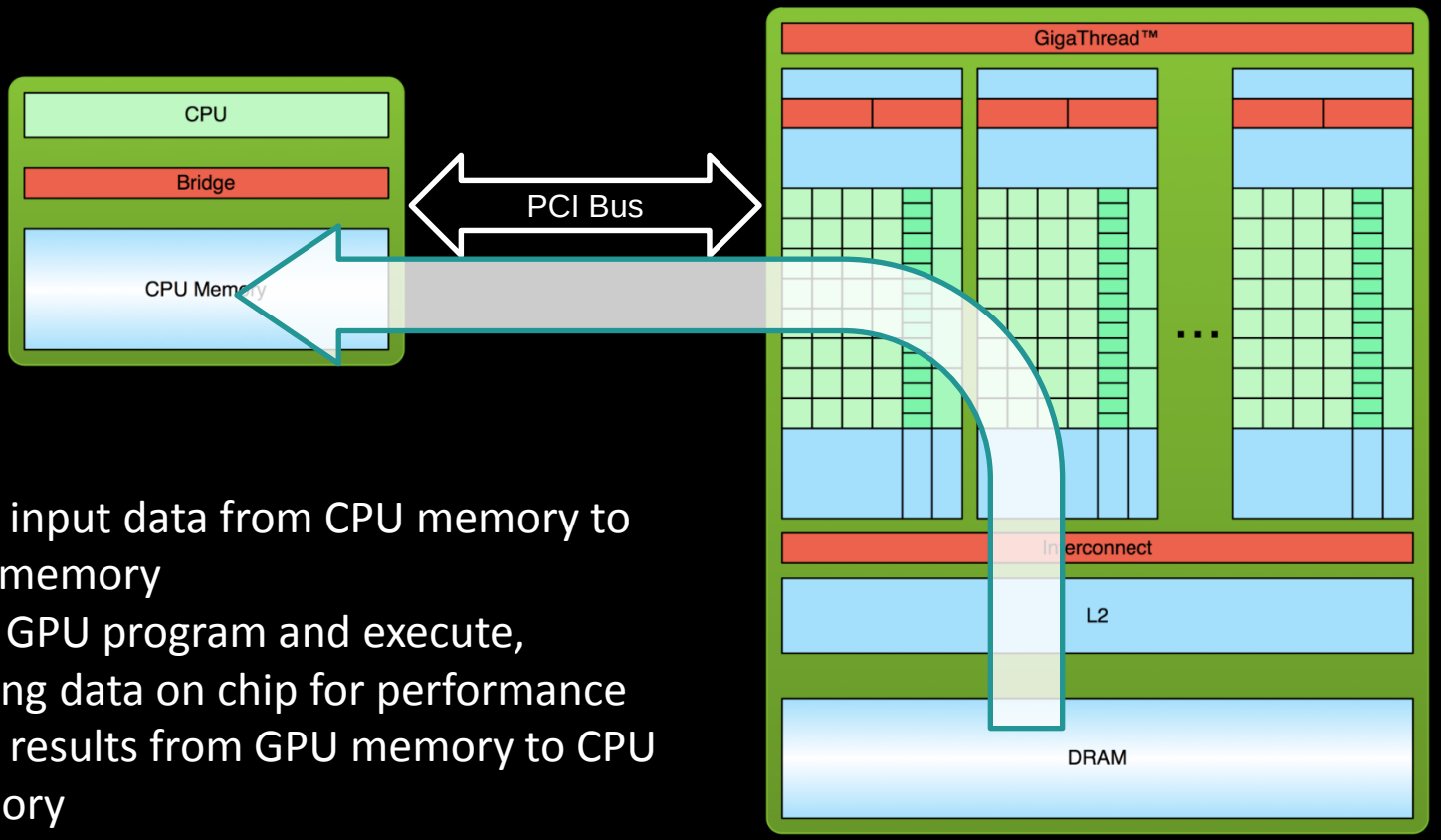
DISCOVER WITH PLYMOUTH UNIVERSITY



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance

Simple Processing Flow

DISCOVER WITH PLYMOUTH UNIVERSITY



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

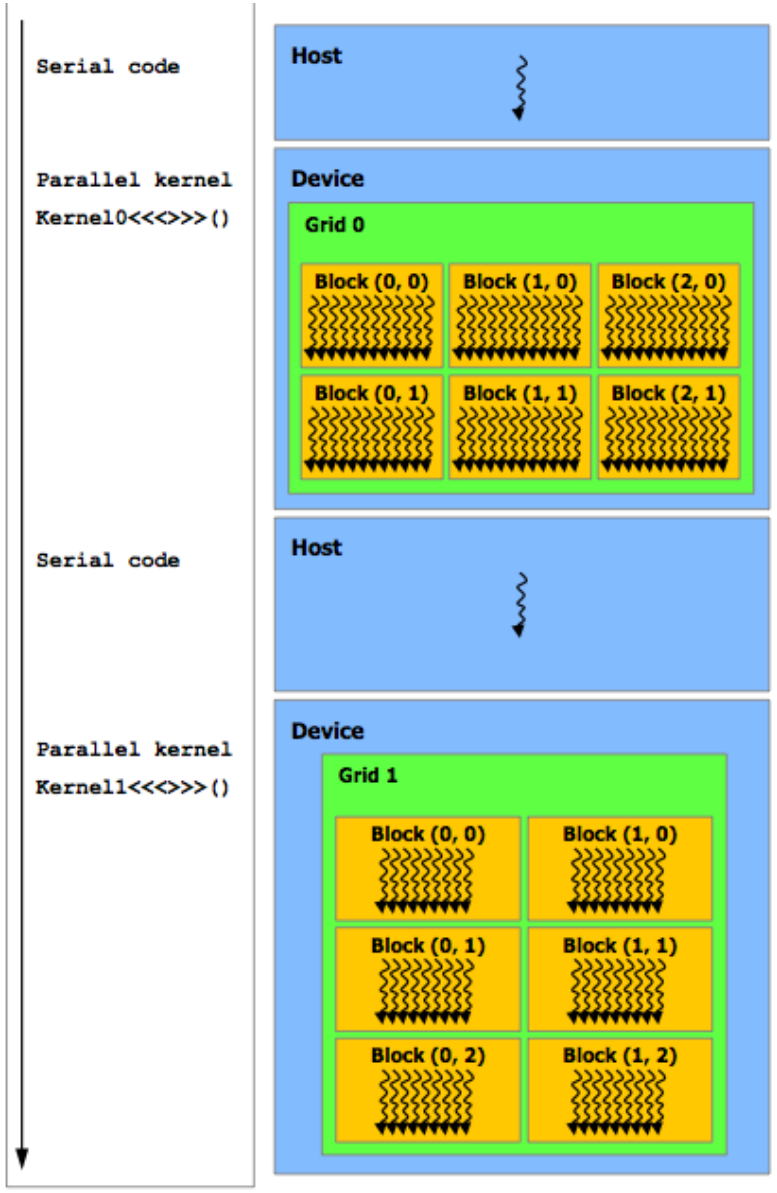
Modello di esecuzione

- Un codice CUDA alterna porzioni di codice seriale, eseguito dalla CPU,e di codice parallelo, eseguito dalla GPU.

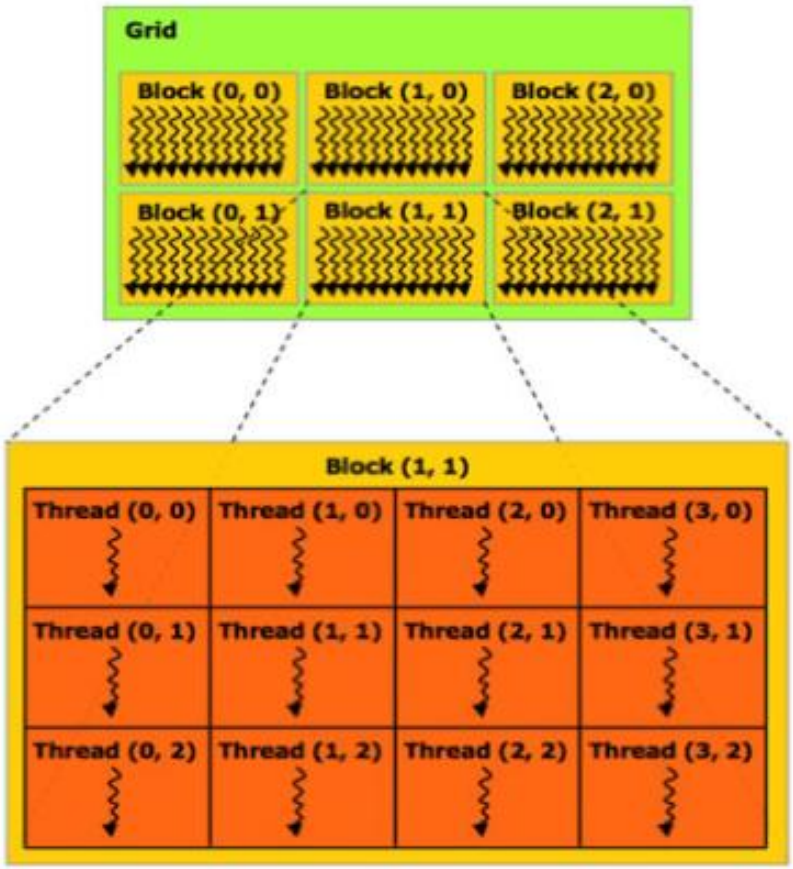
- Il codice parallelo viene lanciato, ad opera della CPU, sulla GPU come **kernel.**
  - ➢La GPU esegue un solo kernel alla volta.

- Un kernel è organizzato in **grids di blocks.**
  - ➢Ogni block contiene lo stesso numero di threads.

- Ogni block viene eseguito da un solo multiprocessore: non può essere spezzato su più SM (symmetric multiprocessors), mentre più blocks possono risiedere ed essere eseguiti in parallelo dallo stesso multiprocessore.



Gerarchia dei thread



Host = CPU
Device = GPU

Thread: codice concorrente, eseguibile in parallelo ad altri threads su un device CUDA.

Warp: un gruppo di threads che possono essere eseguiti **fisicamente** in parallelo.

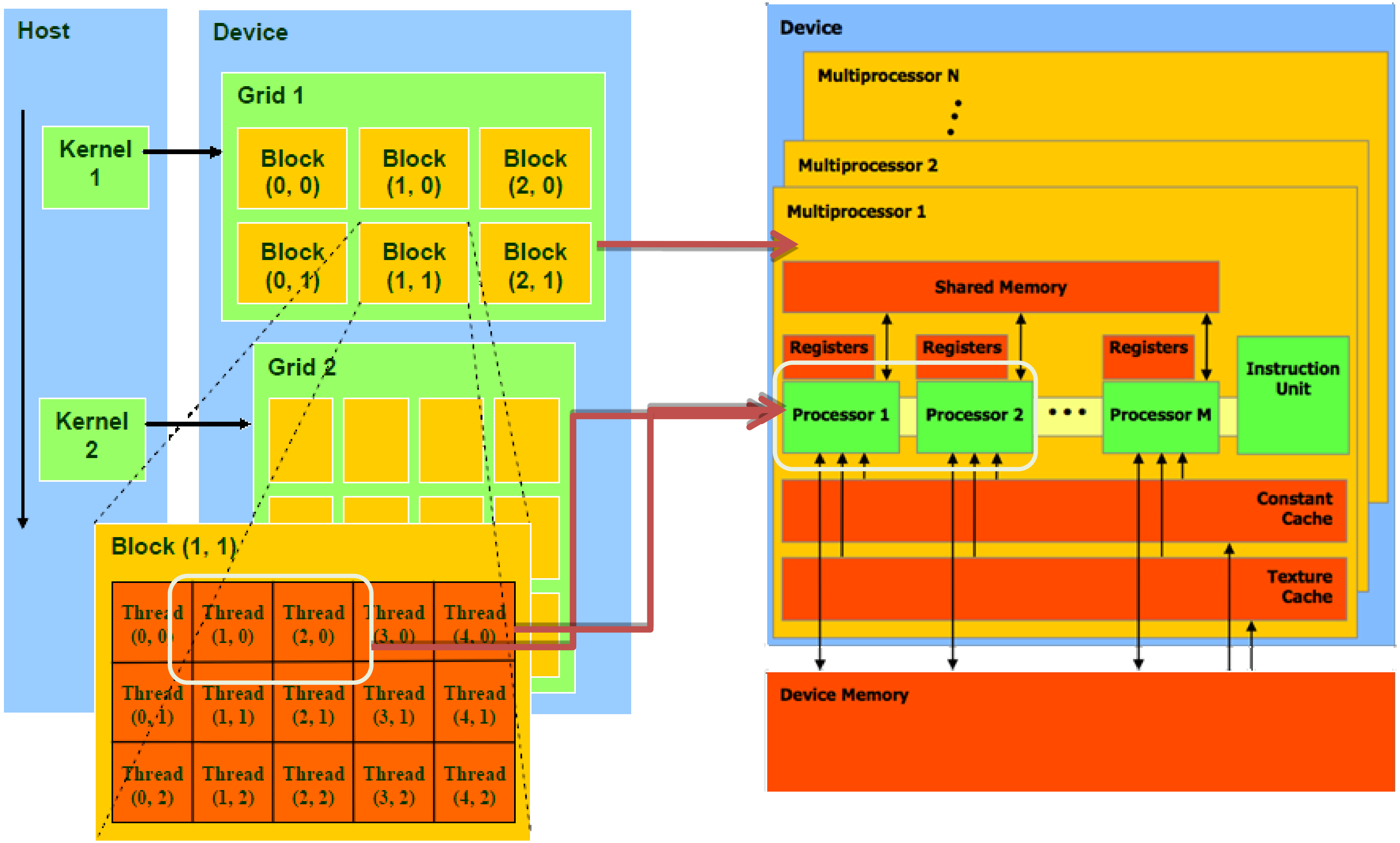
Half-warp: una delle 2 metà di un warp spesso eseguiti sullo stesso multiprocessore.

Block: un insieme di threads eseguiti sullo stesso Multiprocessore, e che quindi possono condividere memoria (stessa shared memory).

Grid: un insieme di thread blocks che eseguono un singolo kernel CUDA, in parallelismo logico, su una singola GPU.

Kernel: il codice CUDA che viene lanciato dalla CPU su una o più GPU.

Esecuzione del codice



Ogni blocco di Thread è caricato in un "Multiprocessore"

In ogni "momento" di calcolo è processato un "warp" di threads per volta

Multidimensionalità degli IDs

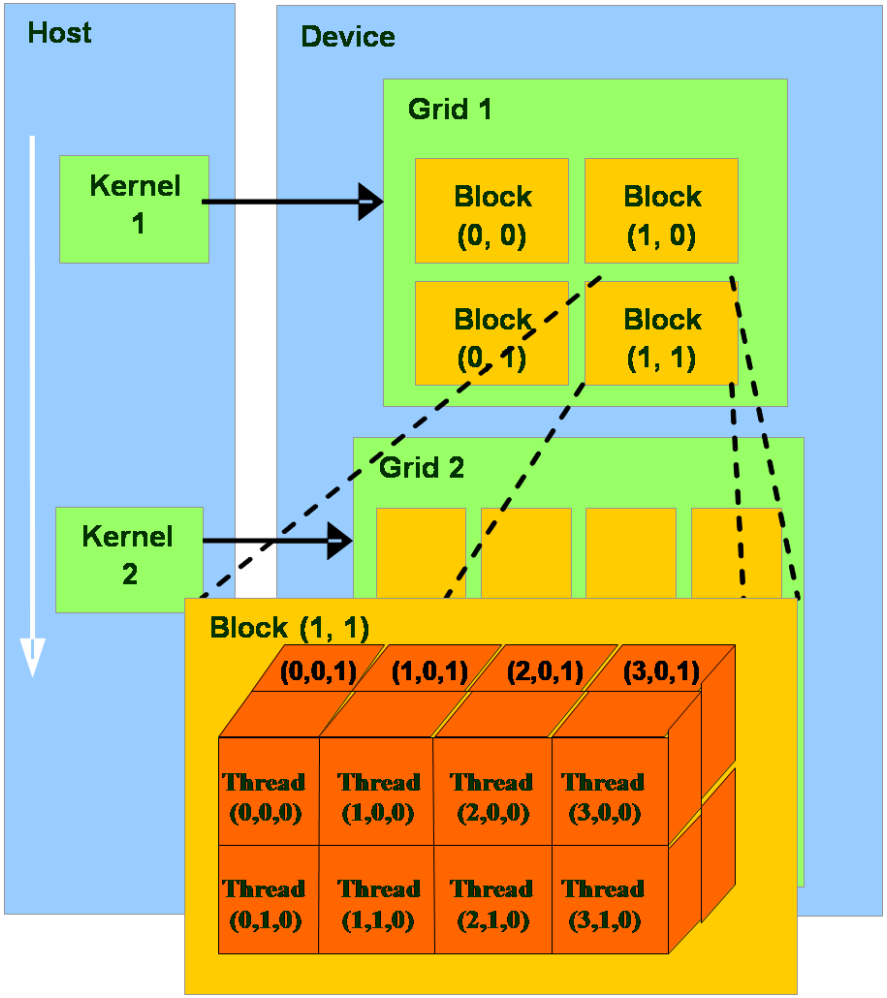
Il codice parallelo viene lanciato, dalla CPU sulla GPU, e questa esegue un solo kernel alla volta.

La dimensione della griglia si misura in blocchi questi possono essere:

Block: 1-D o 2-D (3D da comp. capability 2.0 in poi)

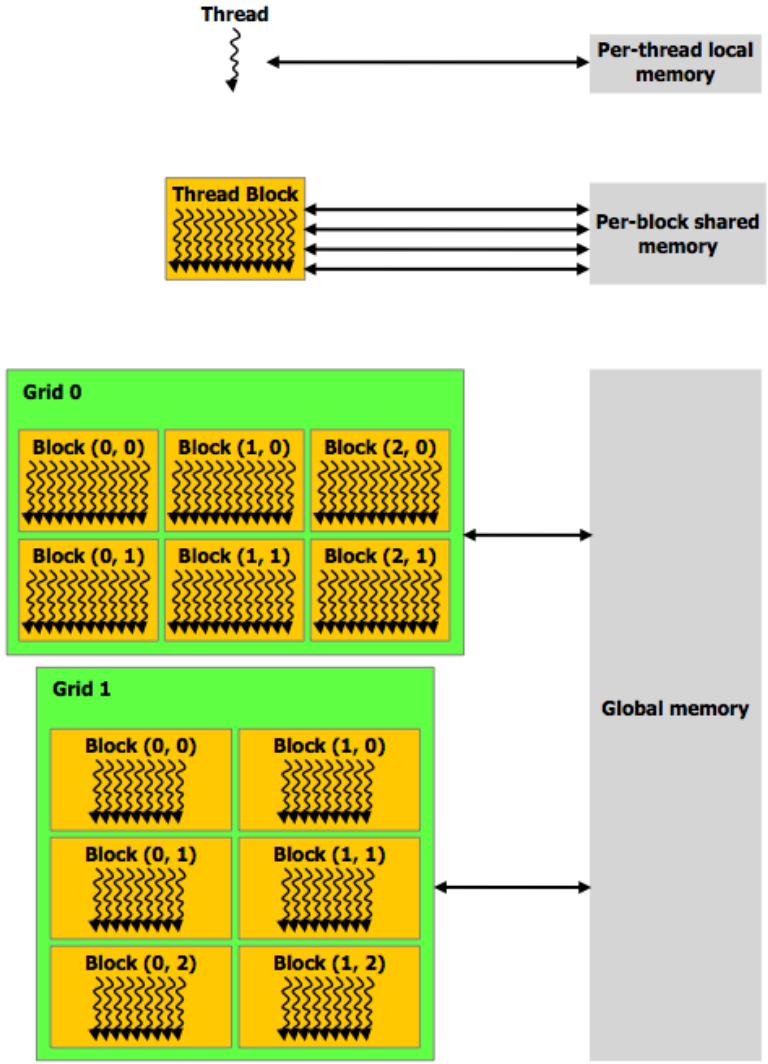
La dimensione dei blocchi si misura in thread

Thread 1-D,2-D,3-D



Organizzazione gerarchica della memoria

- **Register file**: area di memoria privata di ciascun thread (var. locali).
- **Shared memory**: accessibile a tutti i threads dello stesso block. Può essere usata sia come spazio privato che come spazio condiviso.
- Tutti i threads accedono alla medesima **global memory** (off-chip DRAM).
- Memorie read-only accessibili da tutti i threads: **constant** e **texture memory**.
  - ➢dotate di cache locale in ogni SM
- ***Global, constant*** *e* ***texture memory*** *sono memorie persistenti tra differenti lanci di kernel della stessa applicazione.*
- ***Global memory bandwidth****: 2 ordini di grandezza superiori della shared memory!*



Calling a Kernel Function – Thread Creation

- A kernel function must be called with an execution configuration:

```
__global__ void KernelFunc(...);
dim3   DimGrid(100, 50);    // 5000 thread blocks
dim3   DimBlock(4, 8, 8);   // 256 threads per block
size_t SharedMemBytes = 64; // 64 bytes of shared memory

KernelFunc<<< DimGrid, DimBlock, SharedMemBytes >>>(...);
```

- Any call to a kernel function is **asynchronous**

from CUDA 1.0 on, explicit synch needed for blocking

Device Runtime Component: Synchronization Function

- **`void __syncthreads();`**
- Synchronizes all threads in a block
- Once all threads have reached this point, execution resumes normally
- Used to avoid RAW / WAR / WAW hazards when accessing shared or global memory
- Allowed in conditional constructs only if the conditional is uniform across the entire thread block

Confronto codice seriale e parallelo

Programma CPU

```
void add_matrix
( float* a, float* b, float* c, int N ) {
int index;
for ( int i = 0; i < N; ++i )
  for ( int j = 0; j < N; ++j ) {
    index = i + j*N;
    c[index] = a[index] + b[index];
 }

}
int main() {
  add_matrix( a, b, c, N );
}
```

Programma CUDA

```
__global__ add_matrix
( float* a, float* b, float* c, int N ) {
int i = blockIdx.x * blockDim.x + threadIdx.x;
int j = blockIdx.y * blockDim.y + threadIdx.y;
int index = i + j*N;
if ( i < N && j < N )
  c[index] = a[index] + b[index];
}

int main() {
dim3 dimBlock( blocksize, blocksize );
dim3 dimGrid( N/dimBlock.x, N/dimBlock.y );
 add_matrix<<<dimGrid, dimBlock>>>( a, b, c, N );
}
```

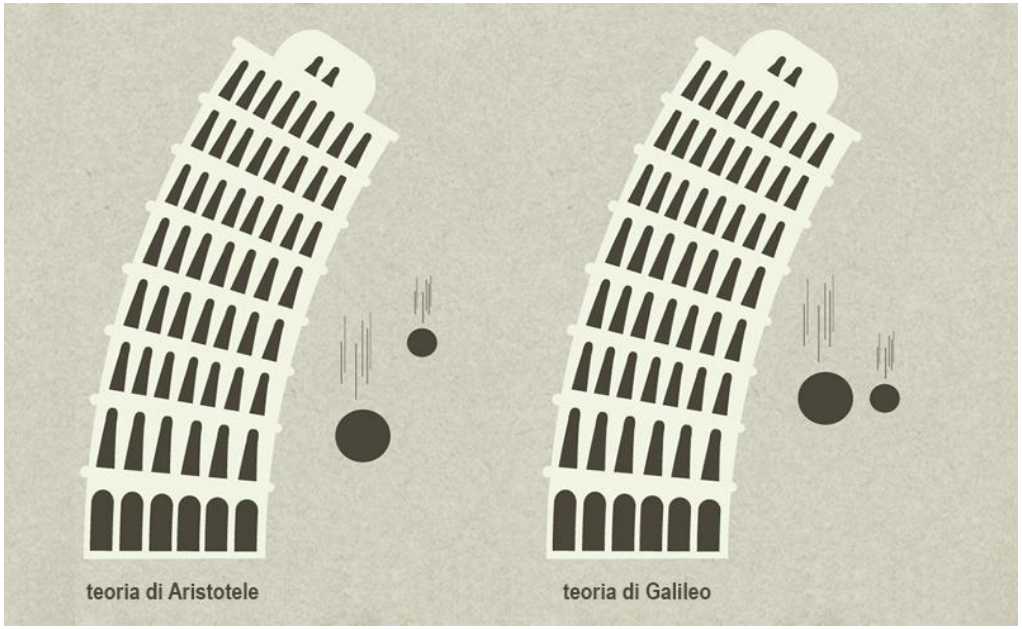
Il ciclo for è sostituito da una griglia implicita

Modelli Paralleli
(… basta codici!)

La Simulazione: il Terzo Pilastro della Scienza

I due **Paradigmi** Tradizionali **Scientifici** e di **Ingegneria** sono:

1) **Fai** la Teoria o Disegno "su carta" (***teorizzazione***)
2) **Esegui** Esperimenti o Costruisci un Sistema (***sperimentazione***)

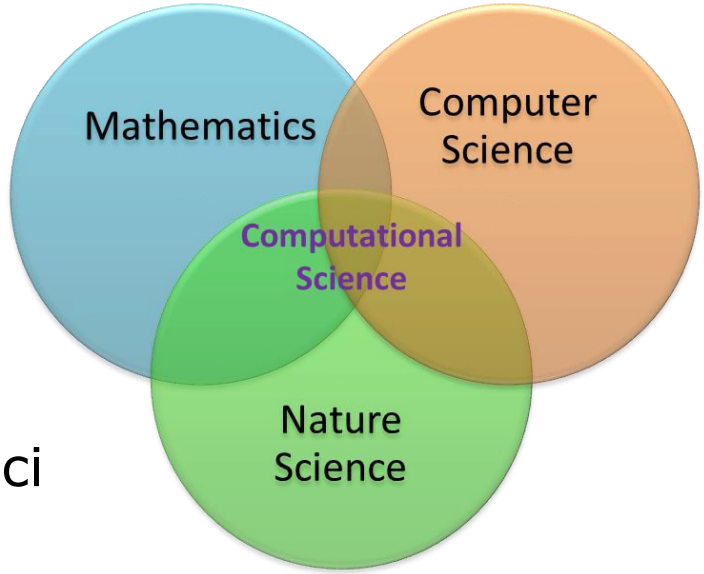


Nel 1590 per il famoso esperimento di **Galileo Galilei** sulla **caduta dei gravi**. Lanciando dalla cima delle torre delle **sfere di diverso peso** dimostrò che "*le velocità de' mobili dell'istessa materia, disegualmente gravi, movendosi per un istesso mezzo, non conservano altrimenti la proporzione delle gravità loro, assegnatagli da Aristotele, anzi che si muovon tutti con pari velocità*".

La Simulazione: il Terzo Pilastro della Scienza

Limitazioni del metodo tradizionale:

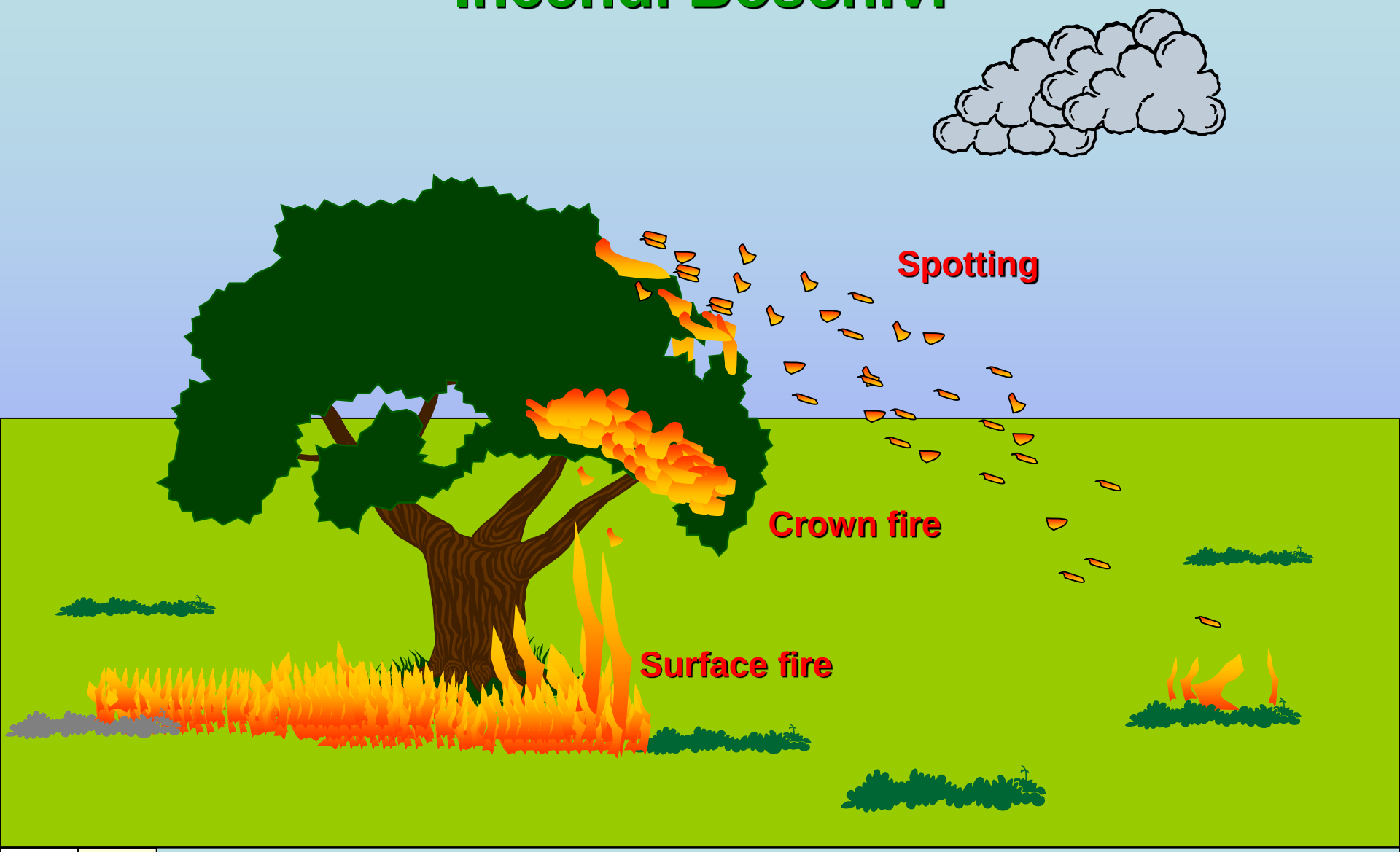
- Troppo Difficile (es: Costruire Gallerie a Vento)
- Troppo Costoso (es: Costruire un aereo usa e getta)
- Troppo lento (es: attendere l'evoluzione del clima, etc)
- Troppo Pericoloso (es: armi nucleari, disegno di farmaci nuovi, etc)



Il Paradigma delle Scienze Computazionali:

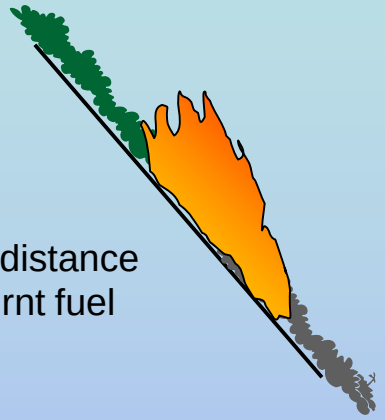
3) **Usa** sistemi di computers ad alte prestazioni per simulare i fenomeni, basati su leggi fisiche e metodi numerici efficienti (***simulazione***)

Esempio di un modello di Simulazione: Incendi Boschivi



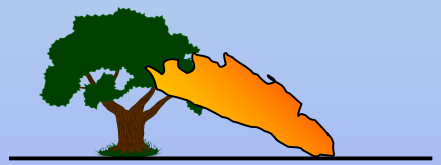
Considerazioni

- upward **slope**: **higher propagation velocity**
- downward **slope**: **smaller propagation velocity**

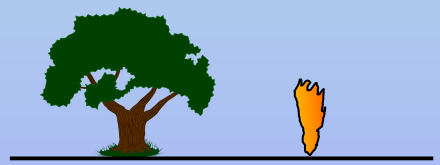


The slope decrease the distance from flames to unburnt fuel

- Also **wind speed** and direction greatly affect fire propagation



wind



no wind

According to Rothermel (1972), maximum spread rate occurs in the direction of the resultant wind-slope effect vector:

$$\phi = \phi_w + \phi_s$$

Slope effect

Wind effect

Equazioni Analitiche

The **maximum fire spread rate** (m min^{-1}) is computed as (Rothermel, 1972):

$$R_{max} = R_0(1+|\phi|) \quad \text{where} \quad R_0 = I_R\ \xi/(\rho_b\ h\ Q_{ig})$$

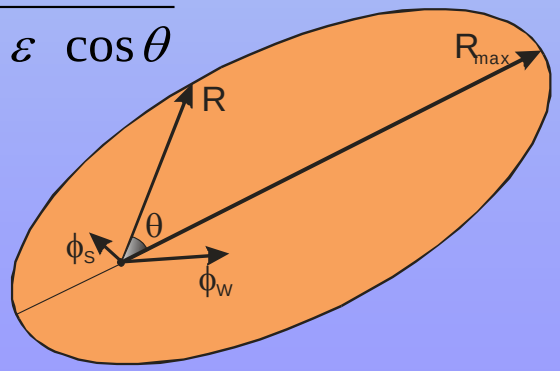
from fuel characteristics

spread rate on flat terrain and without wind

From 1-D to 2-D

The **spread rate in an arbitrary direction** is obtained assuming an elliptical shaped local spread (Anderson, 1982):

$$R = R_{max}\frac{1-\varepsilon}{1-\varepsilon\ \cos\theta}$$



the ellipse eccentricity increases as a function of the midflame windspeed

midflame windspeed

$$\varepsilon = \frac{\sqrt{l_w^{\,2}-1}}{l_w}, \qquad l_w = 0.936\ e^{0.2566 v_e} + 0.461\ e^{-0.1548 v_e} - 0.397$$

Modello (alternativo) di supporto

$\sigma_{I2}: (S_A)^{19} \times S_V \times S_C \times S_H \times S_T \times S_D \times S_{WD} \times S_{WR} \rightarrow (S_{FS})^{18}$

Local interaction I2:

If the substate combustion is “burning”

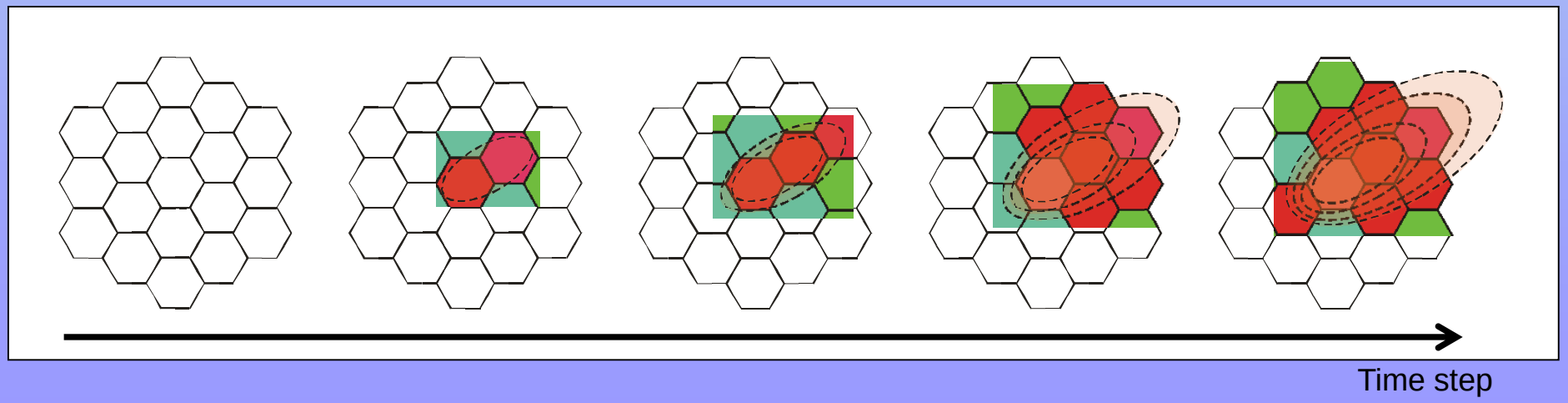
a) First the combined effect is determined as the sum of slope effect and wind effect;
b) then the maximum spread rate R_{max} is computed;
c) the time step dependent ellipse is calculated;
d) The fire can propagate towards the neighbouring cells inside the ellipse.

$\sigma_{I3}: (S_{FS})^{18} \times S_C \rightarrow S_C$

Local interaction I3:

This function tests if the fire is spreading from any neighbouring cell towards the cell itself.

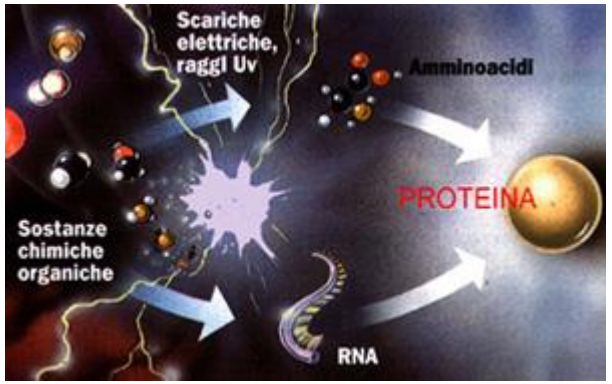
If yes and the combustion substate S_C is “inflammable”, then it changes to “burning”.

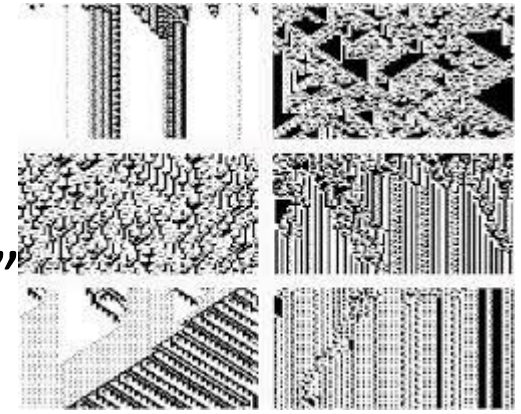


Visualizzazione Risultati

Automi Cellulari (AC)

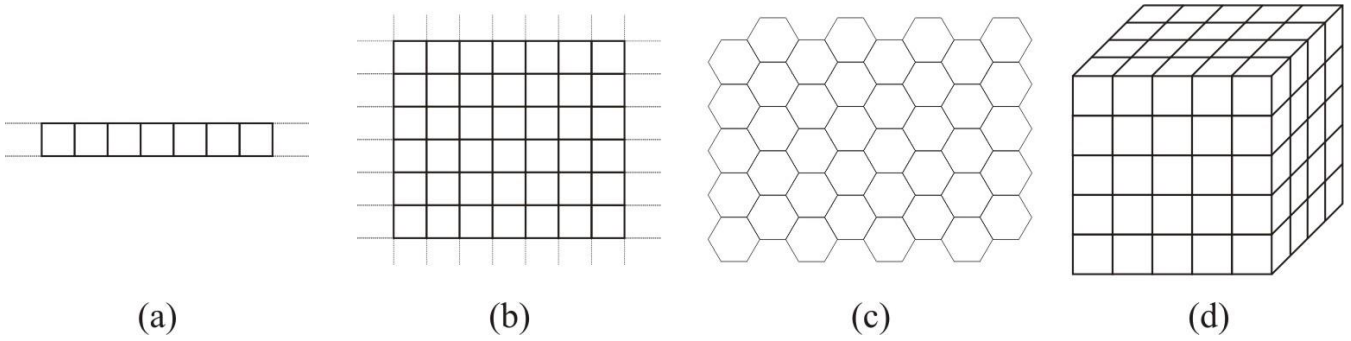
- Sono stati proposti da **John von Neumann** per lo studio dell'autoriproduzione (von Neumann, 1966)
- Sono modelli di **calcolo parallelo discreti nello spazio e nel tempo**
- Gli AC possono essere descritti come una matrice di semplici unità di calcolo, le **celle**, ognuna delle quali "interagisce" con le proprie **vicine**



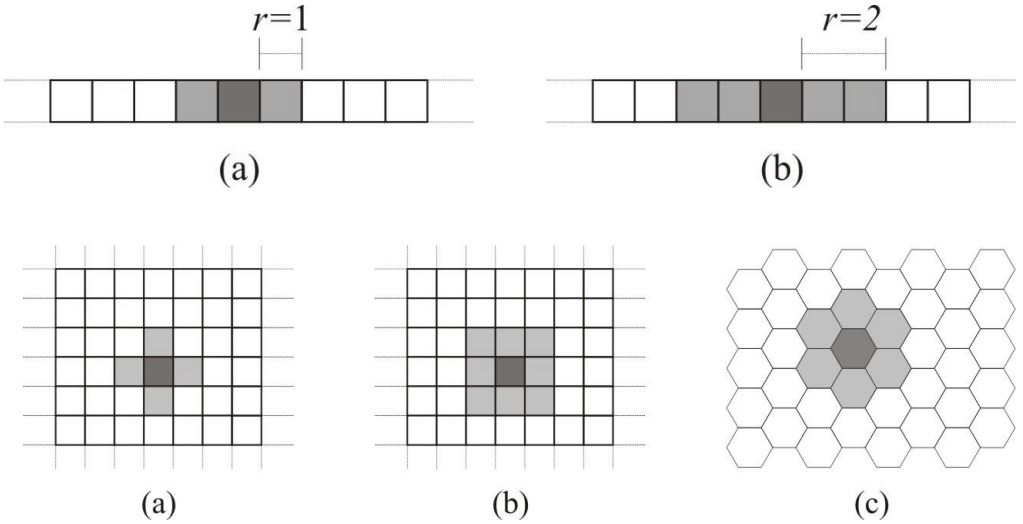


Automi Cellulari (AC)

Esempi di Spazi Cellulari



Esempi di Vicinato



Automi Cellulari (AC)

- **Studi teorici**
  - Teoria della computabilità,
  - Artificial Life,
  - Sistemica dell'emergenza
- **Simulazione di fenomeni complessi:**
  - Processi fluidodinamici
  - Incendi
  - Traffico pedonale/autostradale/ferroviario,
  - Fenomeni geologici

La Simulazione di Processi Geologici

- **Processi Geologici Pericolosi**
  - Flussi di detrito (mudflow, etc)
  - Flussi di lava (basaltiche, piroclastici)
  - Alluvioni
  - Terremoti
- **Approcci per Simulazione**
  - Systems of differential equations (eg. Navier-Stokes, de Saint Venant)
  - FEM, FDM, FVM
  - Cellular Automata (CA)

Flussi Lavici

Navier Stokes ?

Navier–Stokes Equations

3 – dimensional – unsteady

Glenn Research Center

Coordinates: (x,y,z)

Velocity Components: (u,v,w)

Time : t Pressure: p Heat Flux: q

Density: ρ Stress: τ Reynolds Number: Re

Total Energy: Et Prandtl Number: Pr

Continuity:

$$\frac{\partial \rho}{\partial t}+\frac{\partial(\rho u)}{\partial x}+\frac{\partial(\rho v)}{\partial y}+\frac{\partial(\rho w)}{\partial z}=0$$

X – Momentum:

$$\frac{\partial(\rho u)}{\partial t}+\frac{\partial(\rho u^2)}{\partial x}+\frac{\partial(\rho uv)}{\partial y}+\frac{\partial(\rho uw)}{\partial z}=-\frac{\partial p}{\partial x}+\frac{1}{Re_r}\left[\frac{\partial \tau_{xx}}{\partial x}+\frac{\partial \tau_{xy}}{\partial y}+\frac{\partial \tau_{xz}}{\partial z}\right]$$

Y – Momentum:

$$\frac{\partial(\rho v)}{\partial t}+\frac{\partial(\rho uv)}{\partial x}+\frac{\partial(\rho v^2)}{\partial y}+\frac{\partial(\rho vw)}{\partial z}=-\frac{\partial p}{\partial y}+\frac{1}{Re_r}\left[\frac{\partial \tau_{xy}}{\partial x}+\frac{\partial \tau_{yy}}{\partial y}+\frac{\partial \tau_{yz}}{\partial z}\right]$$

Z – Momentum

$$\frac{\partial(\rho w)}{\partial t}+\frac{\partial(\rho uw)}{\partial x}+\frac{\partial(\rho vw)}{\partial y}+\frac{\partial(\rho w^2)}{\partial z}=-\frac{\partial p}{\partial z}+\frac{1}{Re_r}\left[\frac{\partial \tau_{xz}}{\partial x}+\frac{\partial \tau_{yz}}{\partial y}+\frac{\partial \tau_{zz}}{\partial z}\right]$$

Energy:

$$\frac{\partial(E_T)}{\partial t}+\frac{\partial(uE_T)}{\partial x}+\frac{\partial(vE_T)}{\partial y}+\frac{\partial(wE_T)}{\partial z}=-\frac{\partial(up)}{\partial x}-\frac{\partial(vp)}{\partial y}-\frac{\partial(wp)}{\partial z}-\frac{1}{Re_rPr_r}\left[\frac{\partial q_x}{\partial x}+\frac{\partial q_y}{\partial y}+\frac{\partial q_z}{\partial z}\right]$$

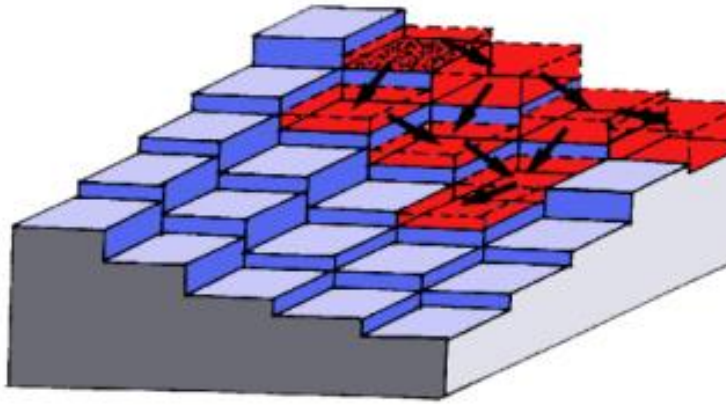
$$+\frac{1}{Re_r}\left[\frac{\partial}{\partial x}(u\tau_{xx}+v\tau_{xy}+w\tau_{xz})+\frac{\partial}{\partial y}(u\tau_{xy}+v\tau_{yy}+w\tau_{yz})+\frac{\partial}{\partial z}(u\tau_{xz}+v\tau_{yz}+w\tau_{zz})\right]$$

Simulazione di Flussi Lavici con AC

$$SCIARA = <R, L, X, Q, P, \tau, \gamma>$$

- R è l'insieme di celle quadrate che ricoprono la regione deve il fenomeno si evolve
- $L \subset R$ è l'insieme delle celle sorgenti (i.e. i crateri);
- $X = \{Center, NW, NE, E, SE, SW, W, N, S\}$ identifica il pattern di celle appartenenti al vicinato
- $Q = Q_a \times Q_t \times Q_T \times Q_f^8$ è l'insieme finito degli stati:

altitudine, spessore di lava, temperatura della lava, flussi uscenti di lava (dalla centrale verso gli otto celle adiacenti)

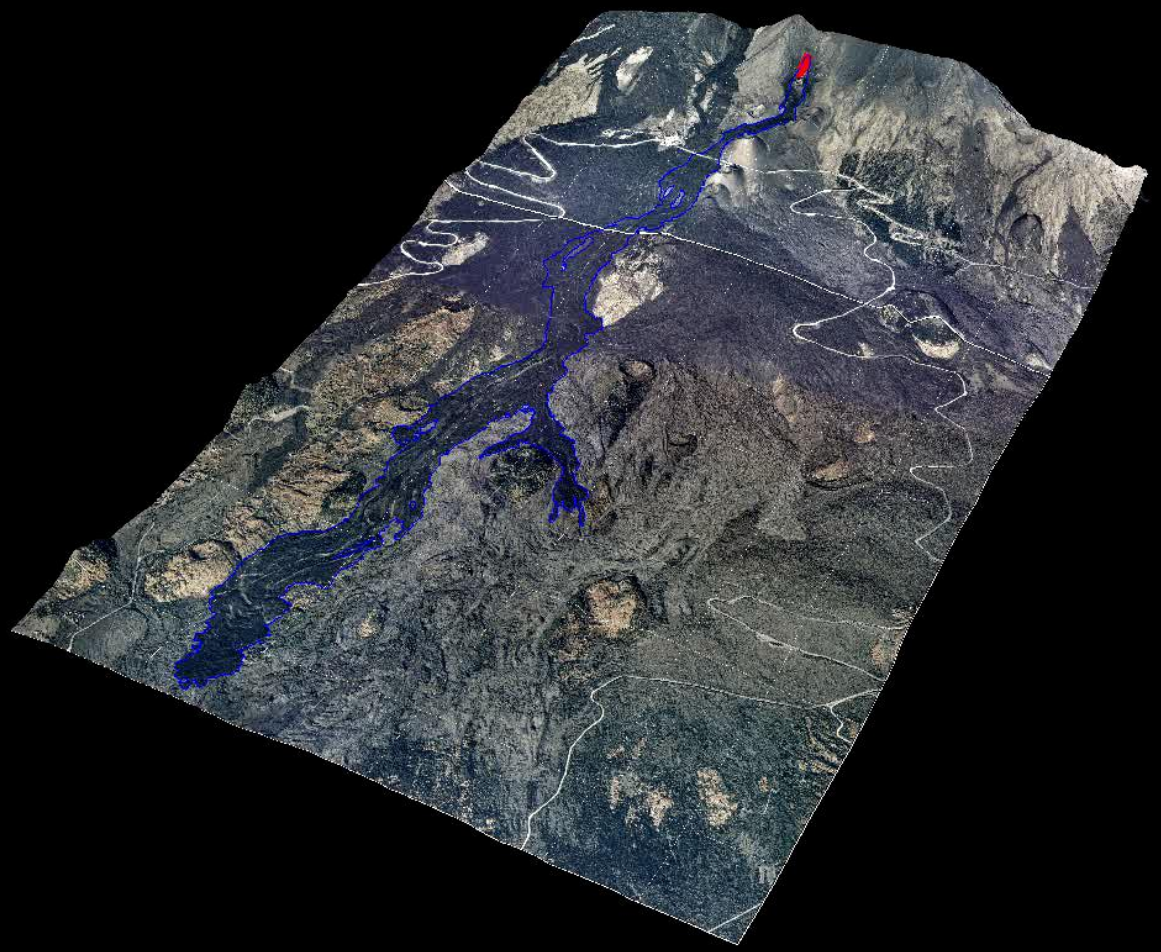


Examples of Applications: lava flows

Mt Etna (Sicily)

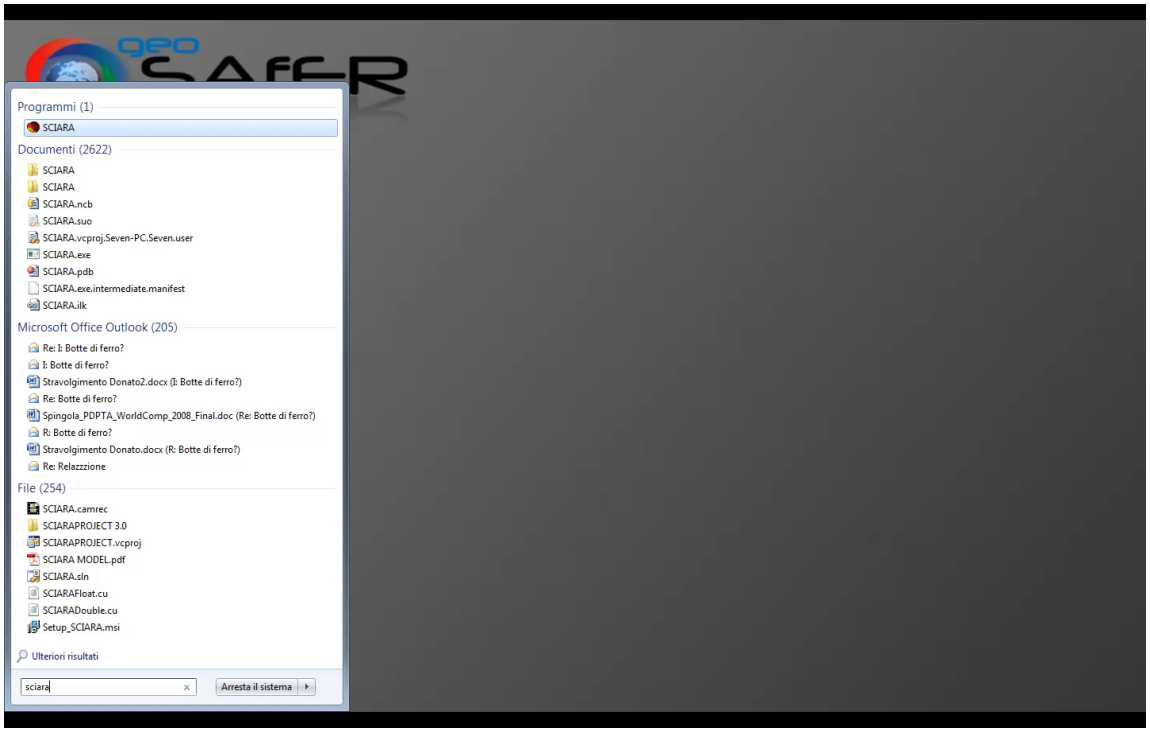
Nicolosi event

July 2001



SCIARA Software 1.0

Analisi e visualizzazione Interattiva di simulazioni di flussi di lava

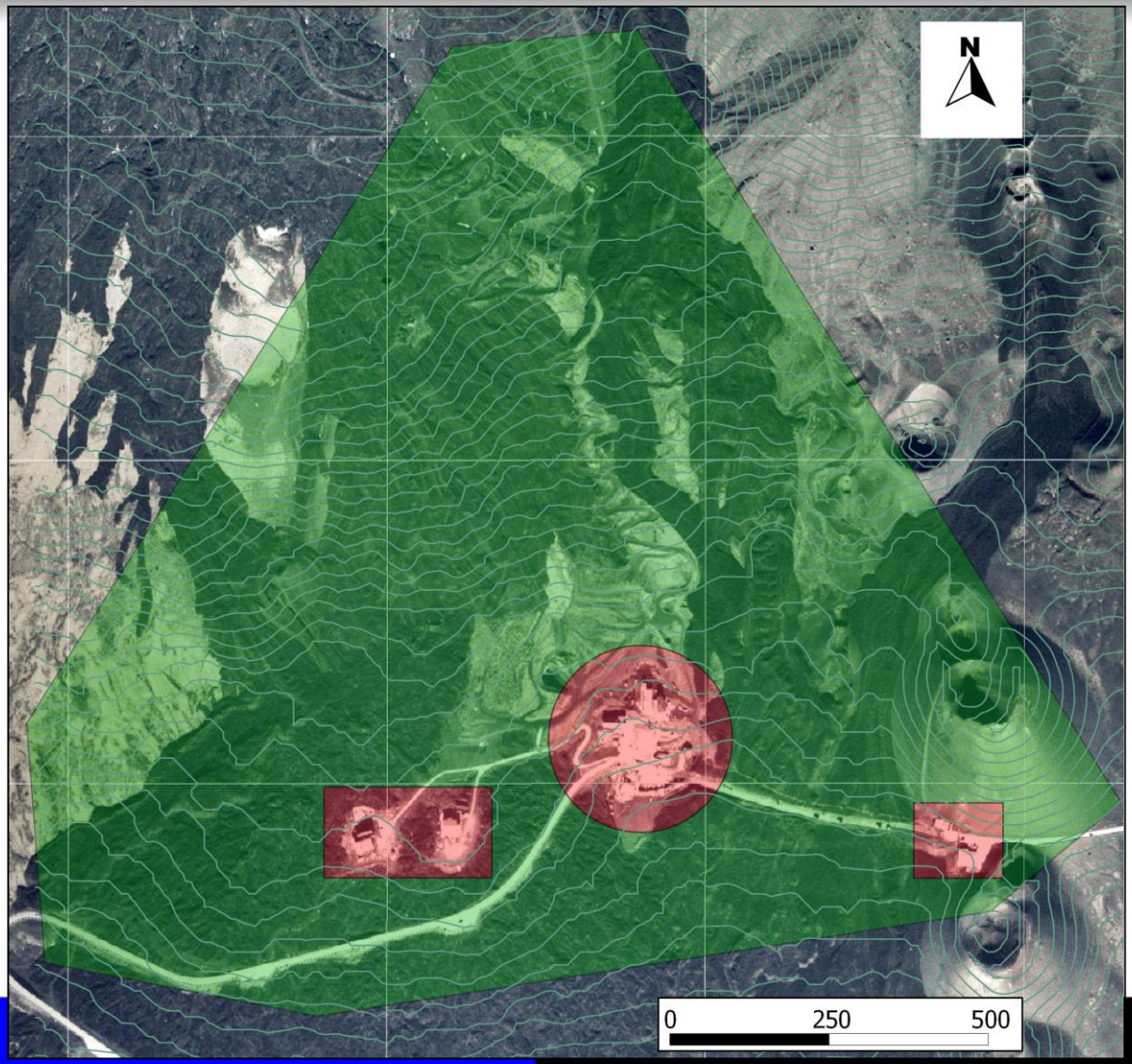


- 3D Graphics engine implemented in **C++** and **OpenGL**
- Visualization System integrated in **Qt GUI**
- CA numerical model resides in a remote HPC and establishes a connection with SCIARA Software **via sockets**

SOFTWARE TOOLS

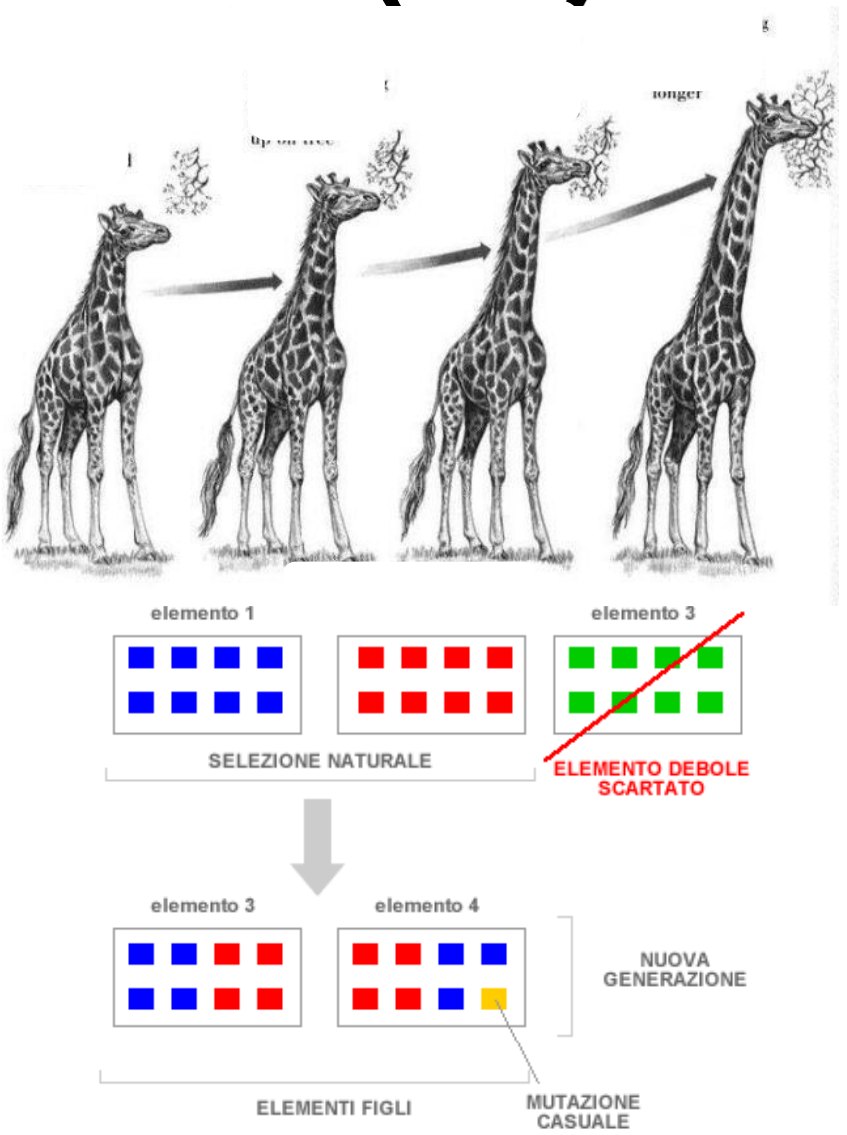
Modeling:3D Models (Triangle Strip Meshing) 2D Models (Color Mapping)
Visualisation: Prospective/orthogonal visualisation
Real-time interaction: Camera management (rotation, zoom, pan), Cell picking, Execution Control

Evoluzione morfologica delle opere di protezione tramite Algoritmi Genetici



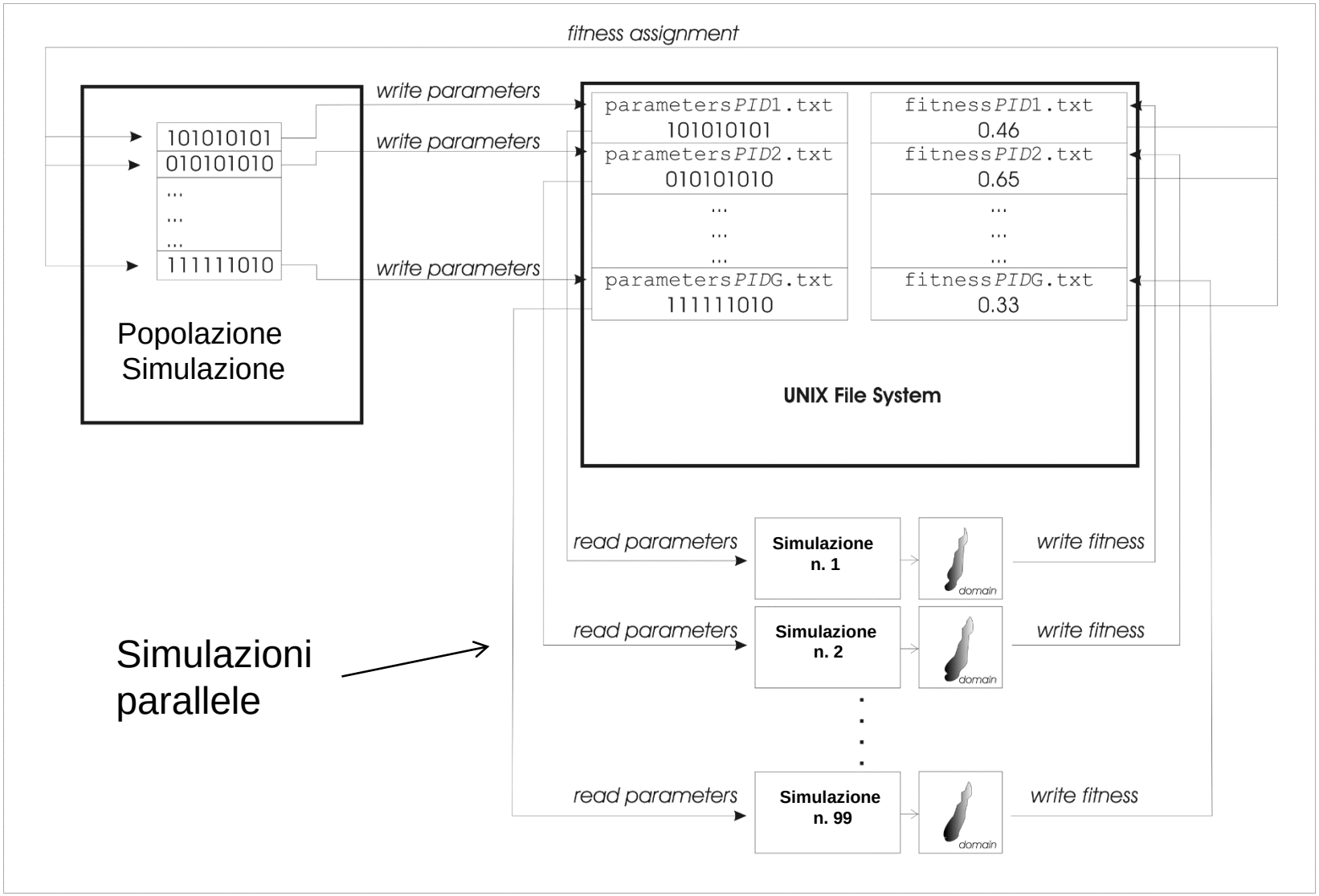
Ottimizzazione dei modelli con gli Algoritmi Genetici (AG)

- Gli AG (Holland, 1975, Goldberg, 1989), utilizzati anche nel campo dell'**Intelligenza Artificiale**, sono **algoritmi di ricerca** che si ispirano ai meccanismi della **selezione naturale** e della **riproduzione sessuale**

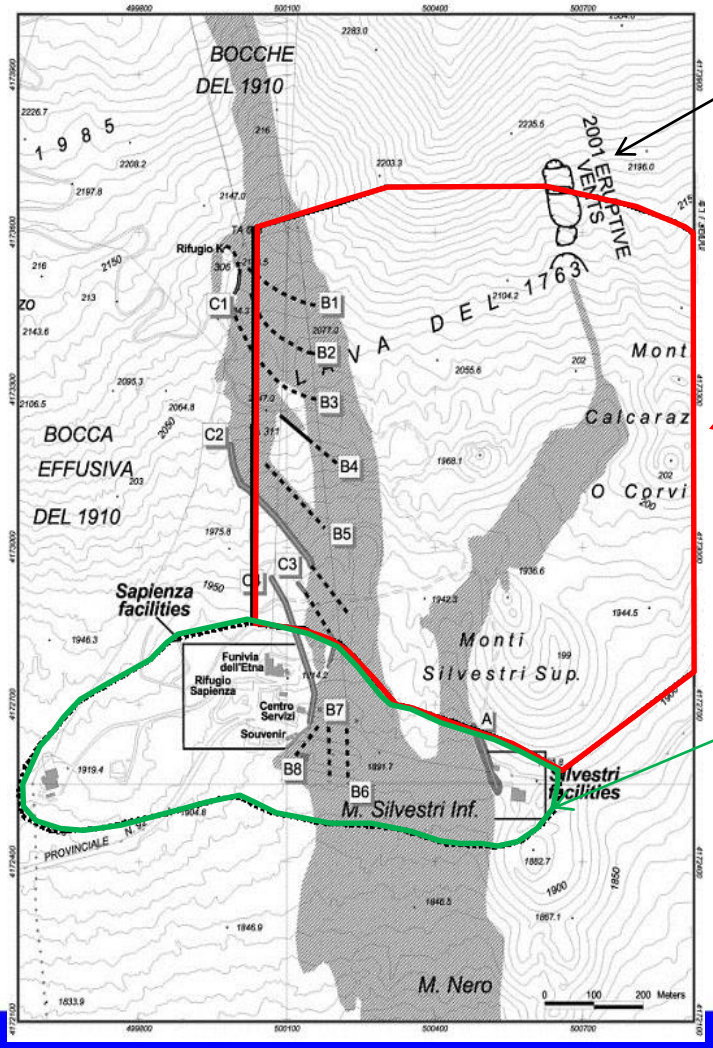


Gli AG simulano **l'evoluzione in modo parallelo (simultaneamente)** di una **popolazione** di **individui**, che rappresentano soluzioni candidate di uno specifico problema, favorendo la sopravvivenza e la riproduzione dei migliori

«Evoluzione» di simulazioni



Evoluzione morfologica delle opere di protezione tramite Algoritmi Genetici



Vent

P Work Perimeter (Mts. Silvestri Zone)

Z Safety perimeter

List of GA parameters

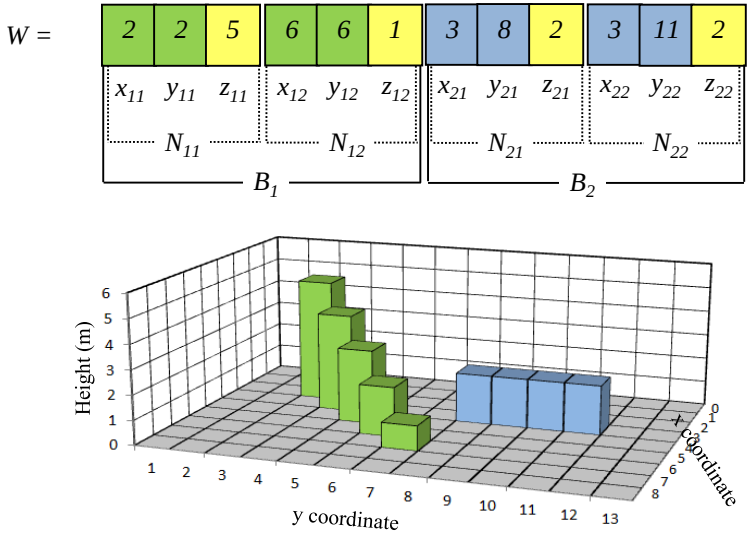
GA parameters	Specification	Value
g_l	Genotype' length	6
p_s	Population size	100
n_g	Number of generations	100
p_{mc}	Coord gene mutation probability	0.5
p_{mh}	Height gene mutation probability	0.5
x_{max}	Gene x position variation radius	10
y_{max}	Gene y position variation radius	10
h_{min}	height min variation range	-5
h_{max}	height max variation range	10
c_{h+}	Cost to build	1
c_{h-}	Cost to dig	1
$\omega 1$	f_1 weight parameter	0.90
$\omega 2$	f_2 weight parameter	0.10

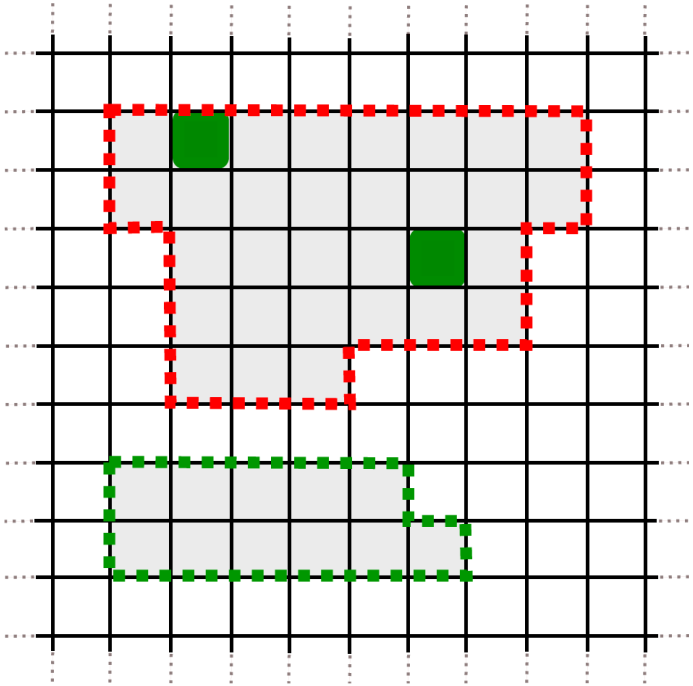
10 seed tests

MEGA system

Genotype Coding

- Opere di protezione considerate come serie di muri





- $x_{ij}, y_{ij}, z_{ij} \in N$
- Search Space depends on $|W|, P$

$$S_r = \{[P_{x_{min}}, P_{x_{max}}] \times [P_{y_{min}}, P_{y_{max}}] \times [(h_{min} * NoG), (h_{max} * NoG)]\}^{2|w|}$$

MEGA Model: the approach

Fitness evaluation 1/2

Three different fitness function were considered for the genotypes evaluation:

$$f_1 = \frac{\mu(S \cap Z)}{\mu(S \cup Z)}$$

where S and Z respectively identify the areal extent of the simulated lava event and the Safety Zone area, with $\mu(S \cap Z)$ e $\mu(S \cup Z)$ as the measures of their intersection and union.

$$f_2 = \frac{\sum_{i=1}^{|w|-1} p_c \cdot d(N_i, N_{i+1}) \cdot h(N_i, N_{i+1})}{V_{max}}$$

$$h(N_i, N_{i+1}) = \frac{|z_i + z_{i+1}|}{2}$$

$$d(N_i, N_{i+1}) = \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$$

where $h(N_i, N_{i+1})$ represent the average height value between two different nodes and $d(N_i, N_{i+1})$ identifies the distance (in m). The parameter p_c is the cell side and $V_{max} \in R$ is a threshold parameter, given by experts, for the function normalization

$$f_3 = f_1 \omega_1 + f_2 \omega_2$$

where $\omega_1, \omega_2 \in R$, $(\omega_1 + \omega_2) = 1$ represent weight parameters associated to f_1 and f_2;

MEGA system

Evoluzione dell'Algortimo Genetico alla ricerca della migliore soluzione

C:\Users\Giuseppe\Desktop\Lavoro\Dottorato\PLYMOUTH\MCA and GA (Progetto di ricerca)\MEGA V 2.2 Tests\GTX680\7890

File

GA Parameters

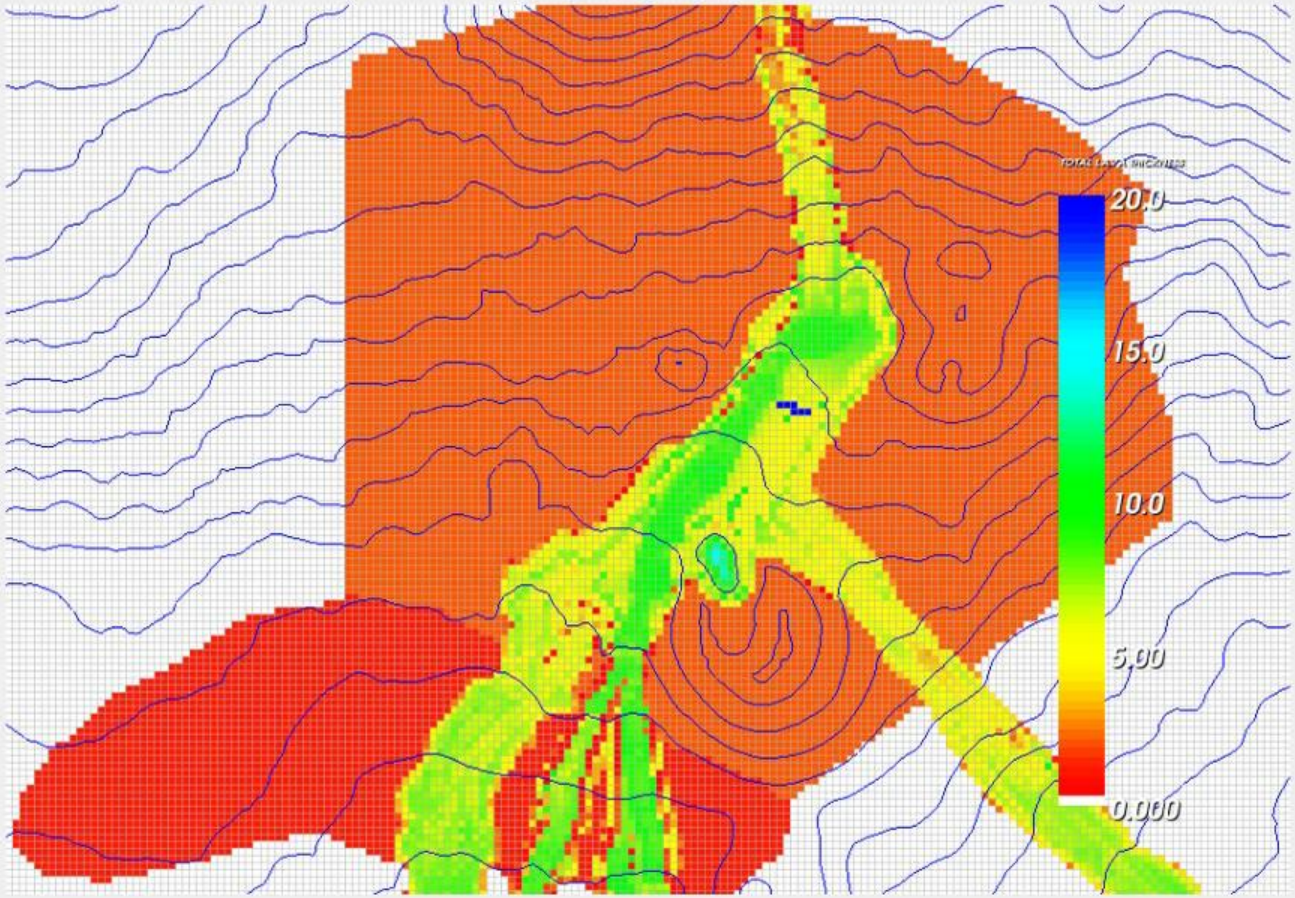
Problem Path:

	Type	Value
1	Population_len...	100
2	Number_of_ge...	100
3	Fenotype_length	2
4	Max_radius_x	10
5	Max_radius_y	10
6	Hmin	-5
7	Hmax	10
8	Ch+	0.5
9	Ch-	0.5
10	Pmc	0.5
11	Pmh	0.5
12	w1	0.99
13	w2	0.01

Generations

Generation 1

	X0	Y0	Z0	X1	Y1	Z1	F1	F2	F3
1	112	173	2	113	177	10	0.278102	0.000259	0.275324
2	150	165	5	153	155	-5	0.280853	0.000259	0.278047
3	98	175	3	98	170	4	0.286353	0.000181	0.283491
4	140	137	0	150	139	4	0.287728	0.000190	0.284852
5	154	145	10	151	145	-2	0.288072	0.000172	0.285193
6	146	158	-2	138	150	-2	0.288415	0.000155	0.285533
7	69	179	-4	76	181	3	0.289790	0.000138	0.286894
8	151	156	3	156	152	1	0.290134	0.000103	0.287234
9	126	156	-5	121	155	0	0.291165	0.000129	0.288255
10	100	164	10	101	169	-5	0.291165	0.000250	0.288256




"That's all Folks!"