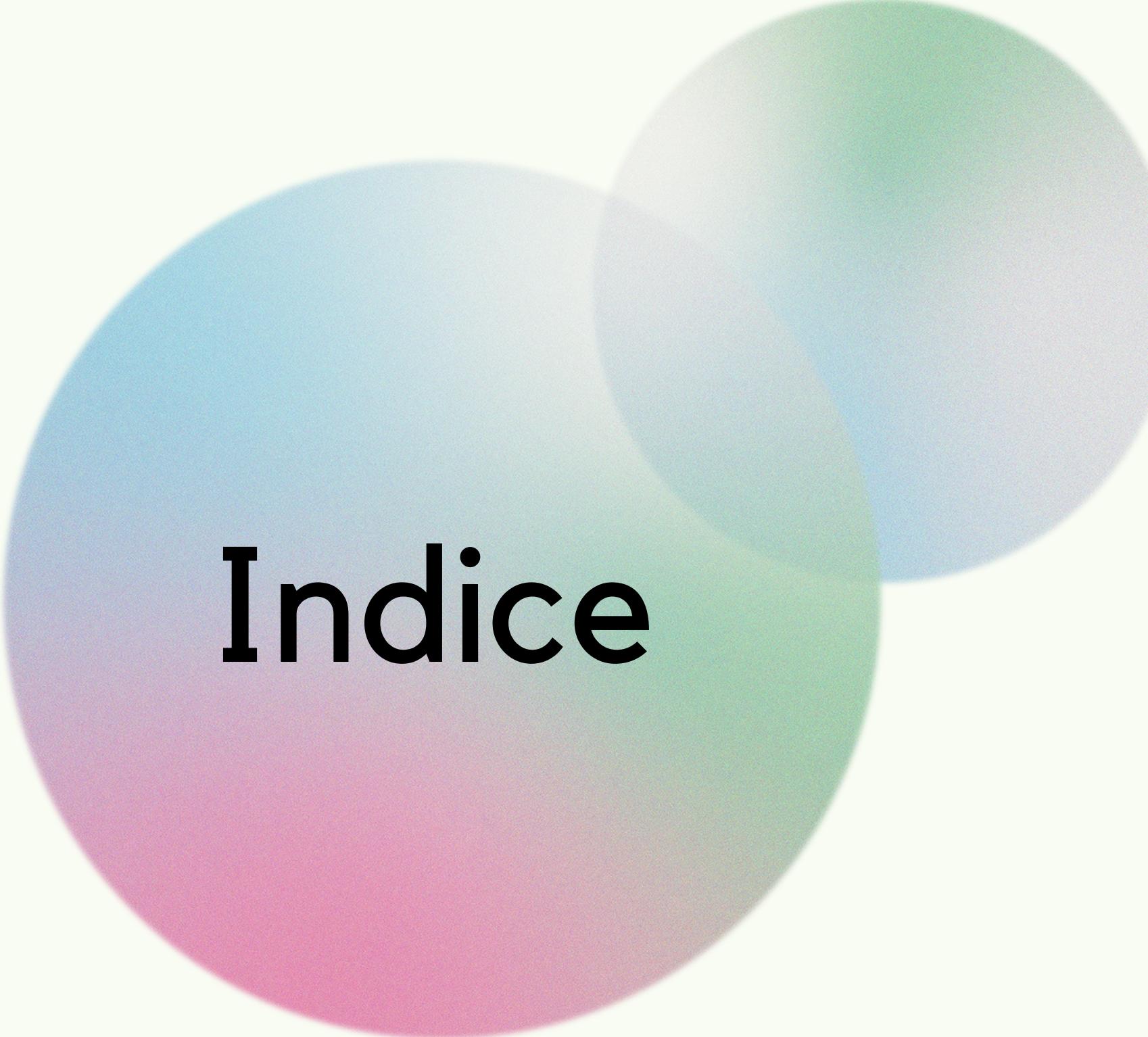


# Calcolo parallelo su GPU: C++ e OpenCL

Un playground per la sperimentazione

Giovanni Spera  
M63001698



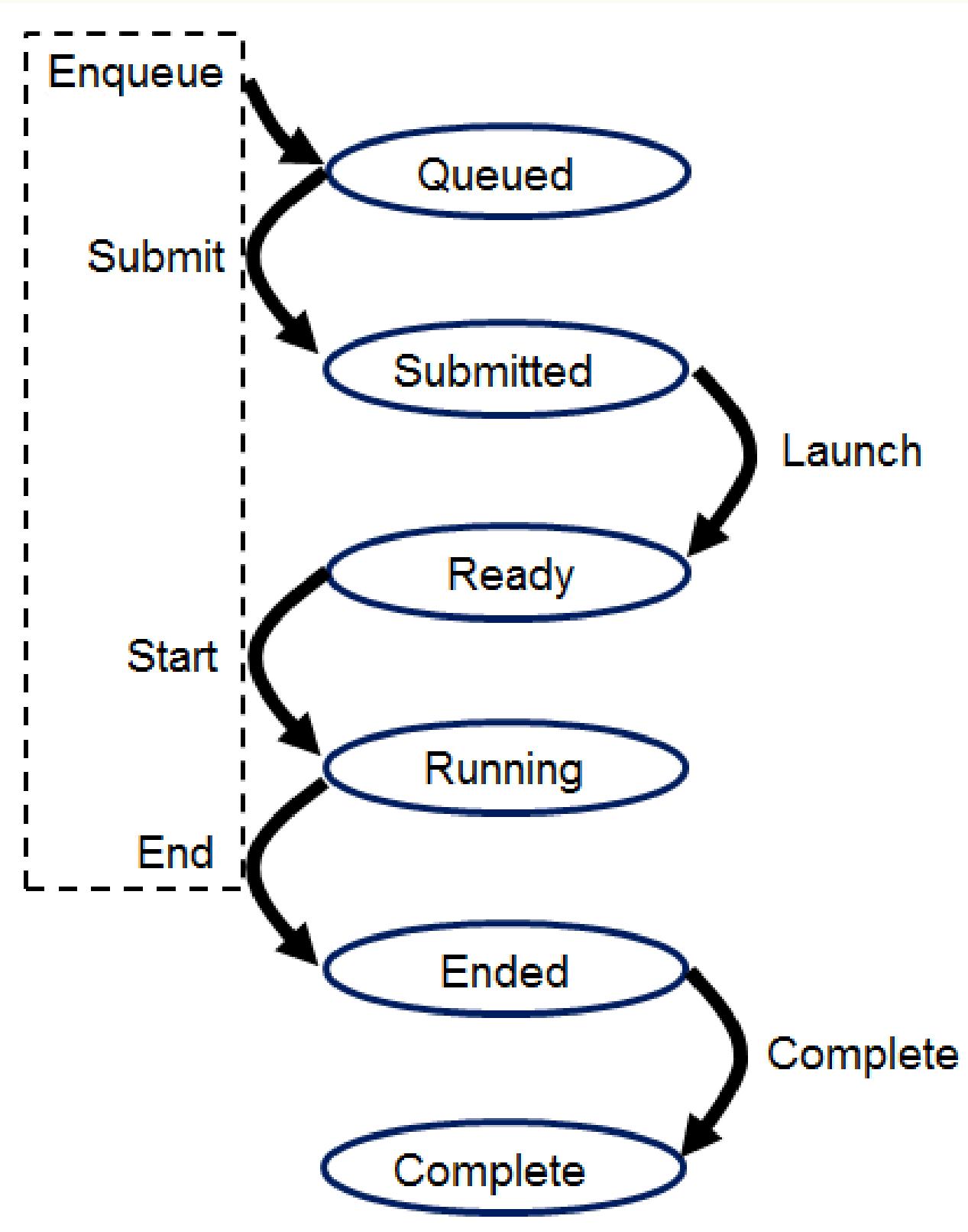
# Indice

- OpenCL
  - Modello di esecuzione
  - Gestione della Memoria
  - Alternative
- Progetto
  - Parametri
  - Implementazione
- DEMO



- E' uno standard **aperto** per la programmazione su sistemi eterogenei, non solo per le GPU, ma anche NPU, DSP, FPGA.
- Basato su un architettura host - devices
- Definisce un interfaccia C per la scoperta e l'interazione delle periferiche
- Permette un accesso granulare e a basso livello delle risorse disponibili

# Modello di Esecuzione

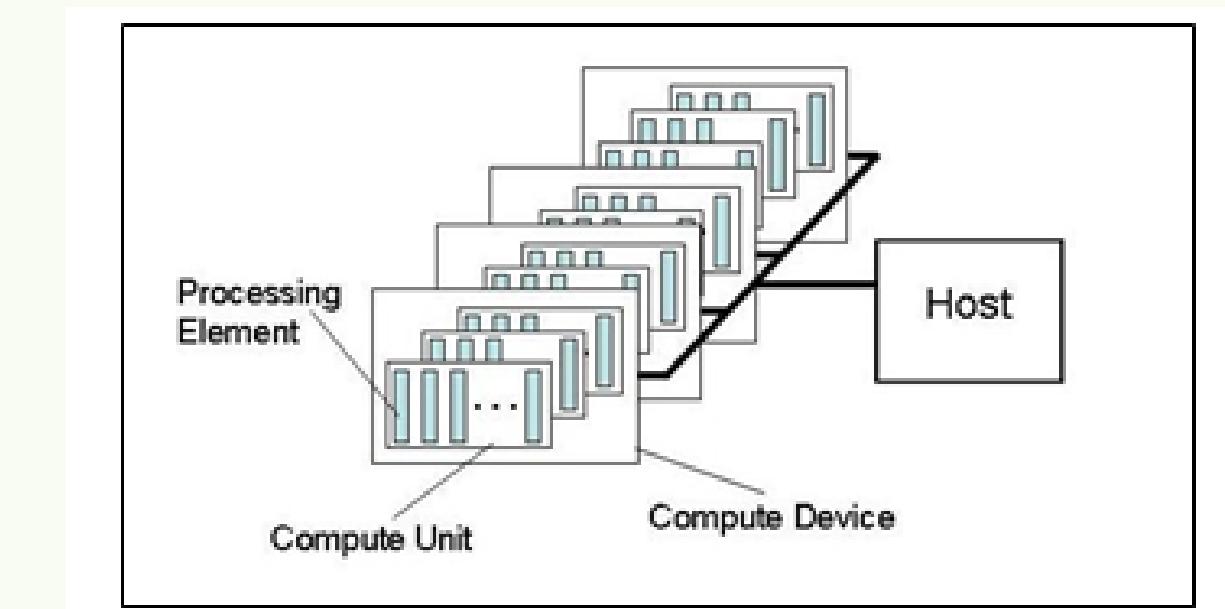


Il flusso di esecuzione di un kernel

OpenCL definisce **kernel** il programma che viene eseguito sui dispositivi, questo è distinto dal programma host, eseguito normalmente sulla CPU.

Il programma host può collegarsi a più dispositivi, ognuno di questi è poi diviso in **compute unit** e, a loro volta, in **processing element**.

La comunicazione con i dispositivi avviene tramite una **coda di comandi**.

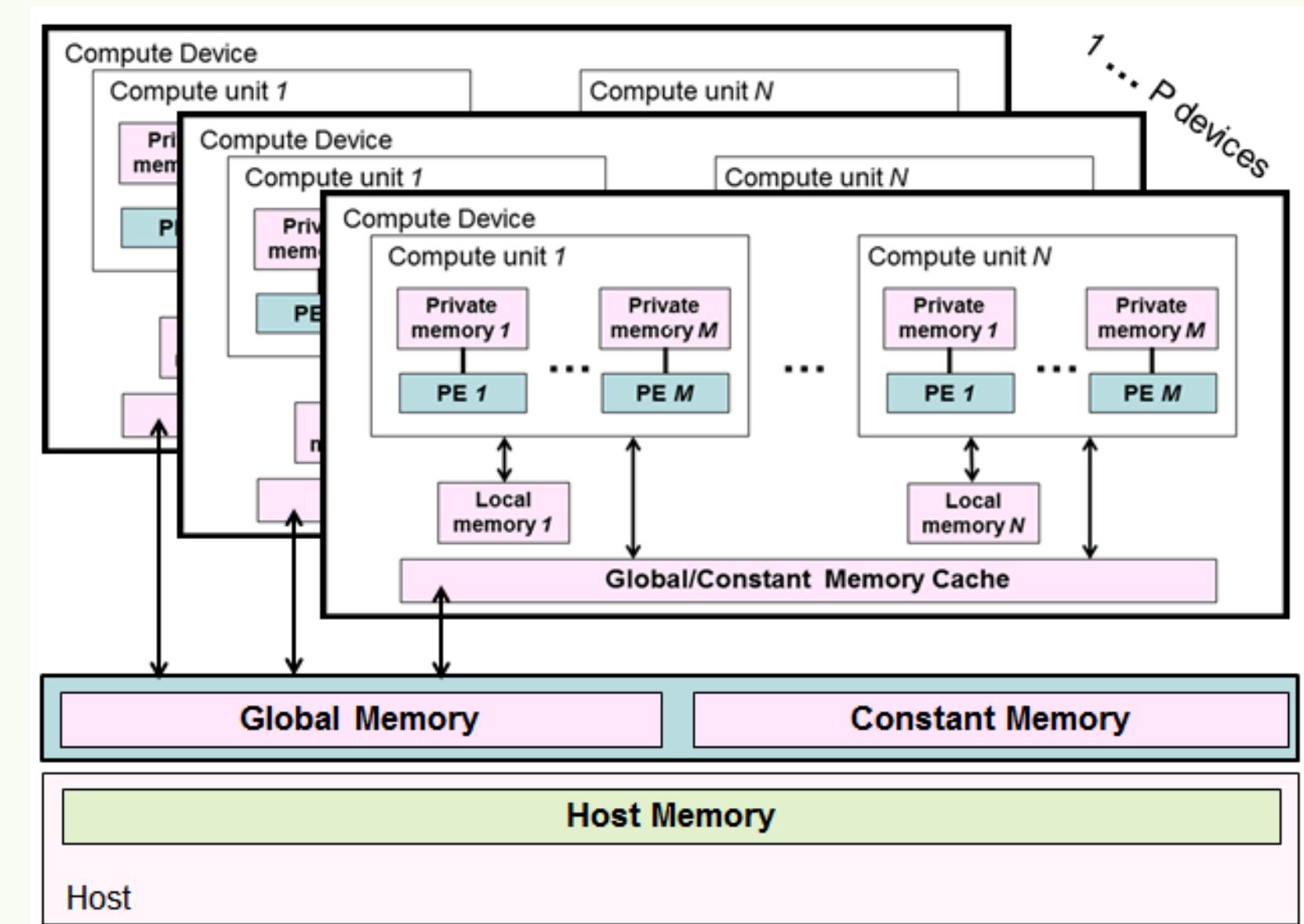


# Gestione della memoria

La memoria dei dispositivi è **distinta** dalla memoria host. In ogni caso deve essere gestita manualmente dal programma host

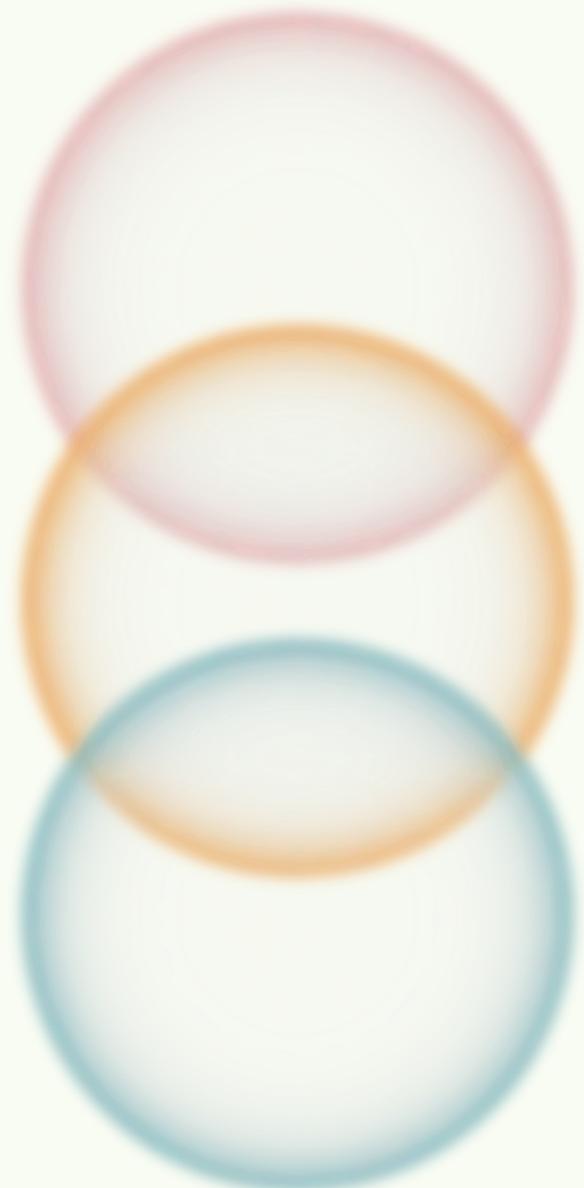
OpenCL definisce diverse zone di memoria, permettendo migliori ottimizzazioni.

- La memoria **globale** è condivisa tra tutti i work-group, ma può avere una cache.
- La memoria **costante**, come la memoria globale, è condivisa tra tutti i work-group, ma può essere solo letta.
- La memoria **locale** è privata per work-group, condivisa tra i work-item all'interno.
- La memoria **privata** è allocata per il singolo work-item.



# Differenze con CUDA

<p>CUDA è specifico per schede grafiche NVIDIA</p>	<p>OpenCL è uno standard aperto, inoltre supporta acceleratori di diverso tipo, come FPGA e NPU.</p>
<p>CUDA usa un linguaggio basato su C++, più avanzato ed espressivo</p>	<p>OpenCL usa un linguaggio basato su C</p>



# Playground

Il progetto è basato sullo sviluppo di un playground in C++ per la sperimentazione di kernel OpenCL.

- E' possibile scrivere ed eseguire velocemente nuovi kernel.
- Supporta nativamente vettori, matrici e immagini.
- Permette di analizzare i tempi di esecuzione e comprendere come il parallelismo agisce sulle prestazioni
- Esecuzione su intervalli sia mono che bi-dimensionali.

Tutto ancora in sviluppo, ci si aspettino crash 😊

The screenshot shows the 'Editor' and 'Argomento: C' sections of the application. In the Editor, a kernel named 'esempio' is displayed:

```
__kernel void esempio(
    __global float *a,
    __global float *b,
    __global float *c
) {
    int id = get_global_id(0);
    c[id] = sin(a[id]) * b[id]*b[id];
}
```

The 'Argomento: C' section displays a table of results for 32 positions. The first few rows are:

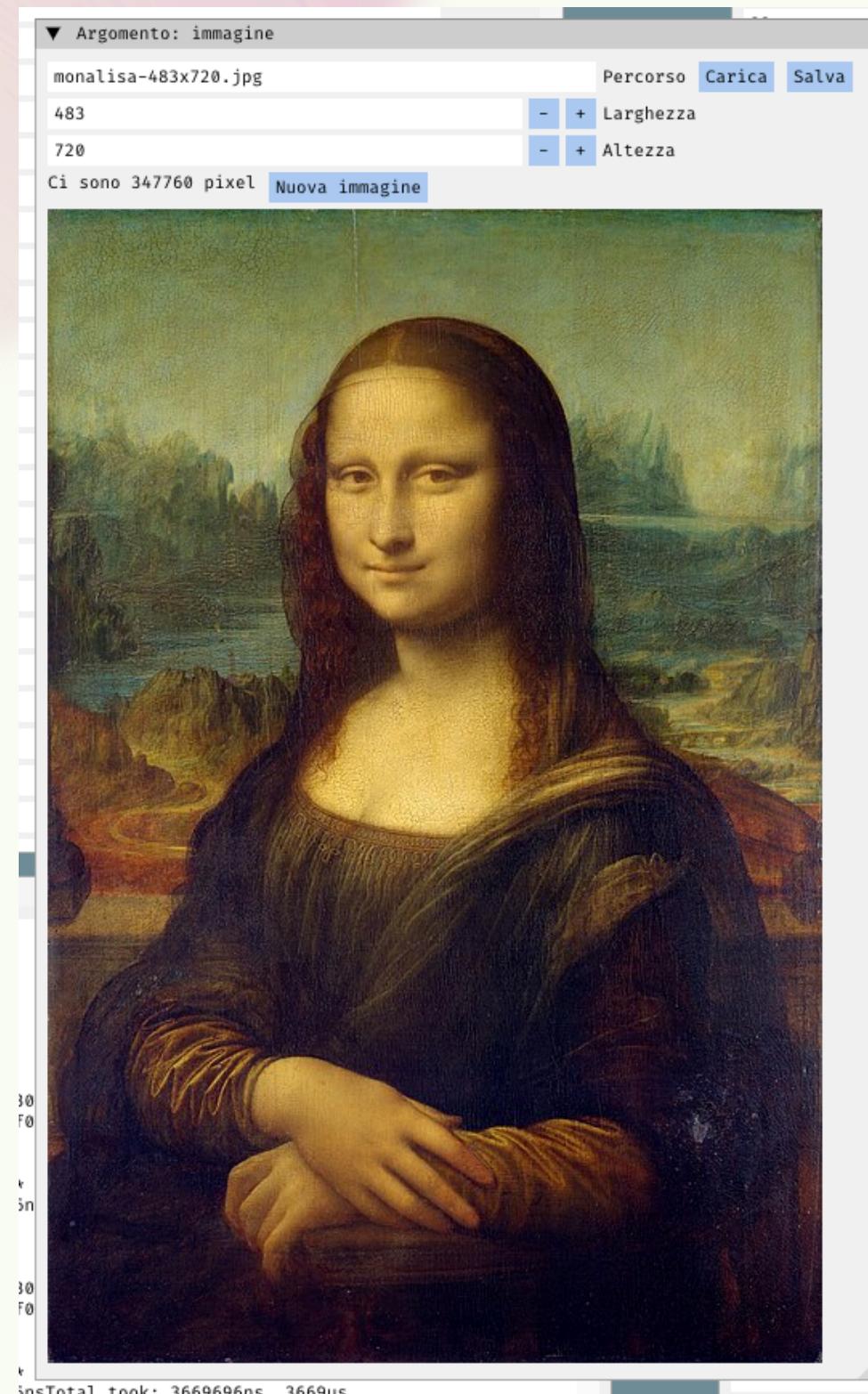
Posizione	Valore
0	6063.929
1	1001.780
2	-94.738
3	-3135.214
4	3433.423
5	544.848
6	-6987.466
7	-5391.070
8	-522.577
9	-1129.335
10	8555.410
11	-5311.300
12	3761.383
13	-53.449
14	1992.488
15	-42.318
16	-116.043
17	-2528.079
18	8922.644
19	820.659
20	-2254.373
21	1252.929
22	-2418.657
23	-357.875
24	3378.741
25	-4566.586
26	-307.243
27	3766.499
28	-2284.562
29	3507.935
30	-4895.645
31	-84.087
32	-1522.100

The 'Argomenti' section shows three vectors (A, B, C) with their respective elimination buttons and ranges.

# Parametri in ingresso e uscita

Argomento: a	
10000	- + Dimensione
<a href="#">Randomizza</a>	<a href="#">Carica CSV</a>
<a href="#">Salva CSV</a>	
Posizione 0:	94.400
Posizione 1:	45.900
Posizione 2:	44.400
Posizione 3:	19.500
Posizione 4:	42.300
Posizione 5:	94.900
Posizione 6:	64.400
Posizione 7:	90.000
Posizione 8:	81.600
Posizione 9:	21.700
Posizione 10:	81.300
Posizione 11:	5.400
Posizione 12:	46.700
Posizione 13:	3.900
Posizione 14:	12.100
Posizione 15:	95.300
Posizione 16:	69.400
Posizione 17:	45.600
Posizione 18:	72.200
Posizione 19:	9.000
Posizione 20:	91.400
Posizione 21:	6.000
Posizione 22:	89.400
Posizione 23:	72.400
Posizione 24:	31.000

Vettore



Immagine

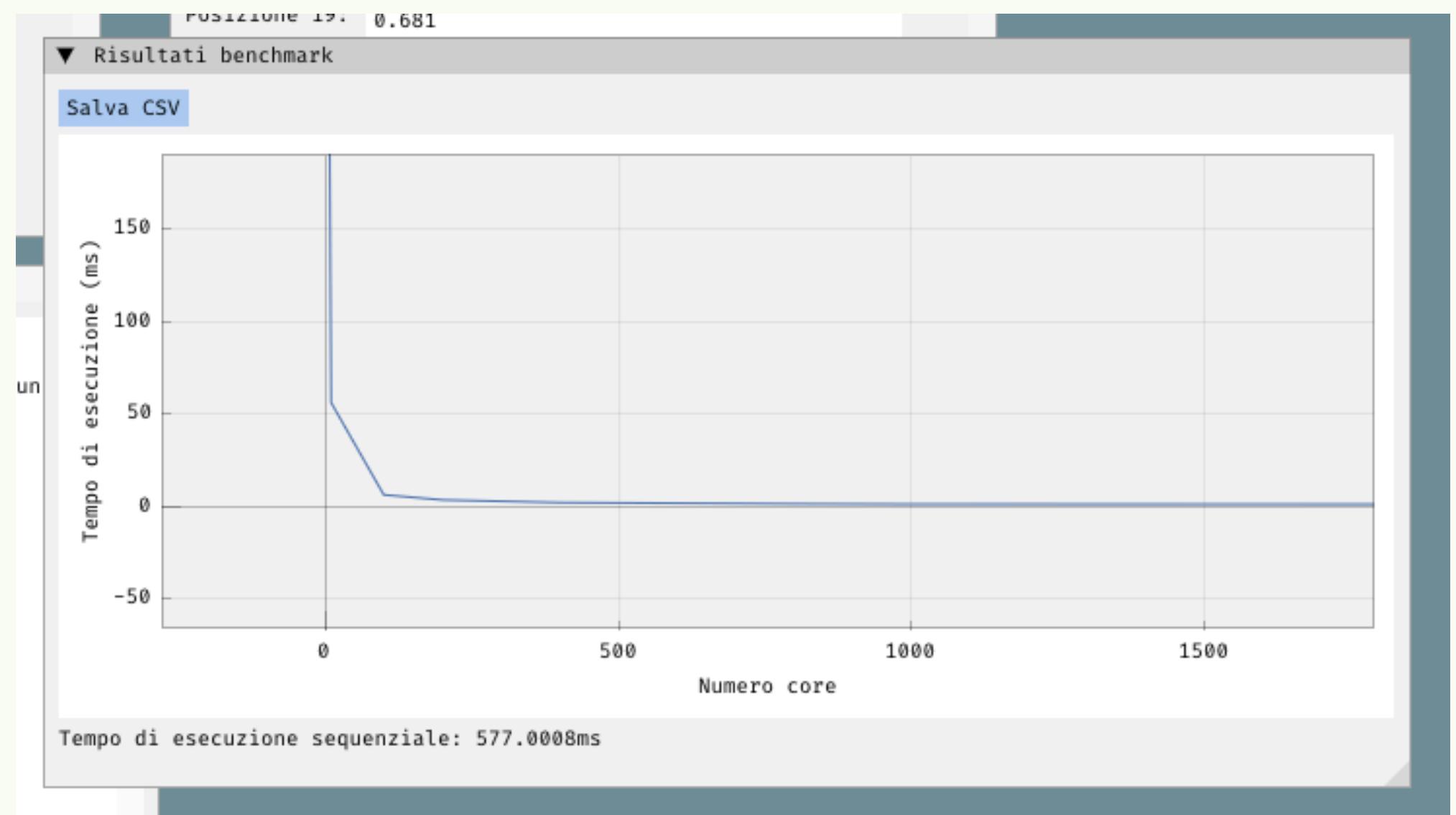
Argomento: matrice									
32	32	- + Righe							
<a href="#">Randomizza</a> <a href="#">Carica CSV</a> <a href="#">Salva CSV</a>									
39.800	4.300	66.000	30.700	77.700	19.700	96.400	59.800	45.600	
89.900	74.100	59.100	2.900	29.000	55.500	97.900	9.800	93.100	
52.400	46.000	47.200	16.600	36.700	80.300	26.500	29.800	85.300	
7.200	67.100	99.600	43.900	47.400	61.300	8.900	67.900	97.400	
92.700	96.800	88.900	75.300	58.100	97.800	43.200	90.700	12.900	
82.900	50.800	60.700	41.000	83.900	39.200	31.800	32.000	55.200	
53.400	42.300	49.800	72.500	16.700	16.800	4.500	7.200	57.100	
47.300	63.100	74.100	64.000	79.900	78.600	71.200	72.300	27.400	
59.700	69.500	58.100	74.800	83.400	64.600	47.100	46.000	35.600	
91.300	75.700	19.900	9.900	40.300	67.100	56.000	11.100	98.200	
32.400	16.800	29.900	7.900	19.100	85.900	19.000	52.600	54.000	
68.300	63.200	43.400	22.600	84.300	97.700	75.200	38.300	23.100	
53.500	13.500	38.100	37.800	11.200	13.300	11.300	69.500	91.300	
34.300	7.300	40.600	80.700	55.900	52.000	50.300	47.200	49.300	
31.800	36.400	74.000	22.900	23.600	24.300	5.400	72.900	31.700	
69.000	93.000	22.600	27.800	52.500	28.000	35.900	84.200	51.900	
59.000	18.000	20.100	11.500	46.100	56.100	31.000	98.000	54.800	

Matrice

# Benchmark

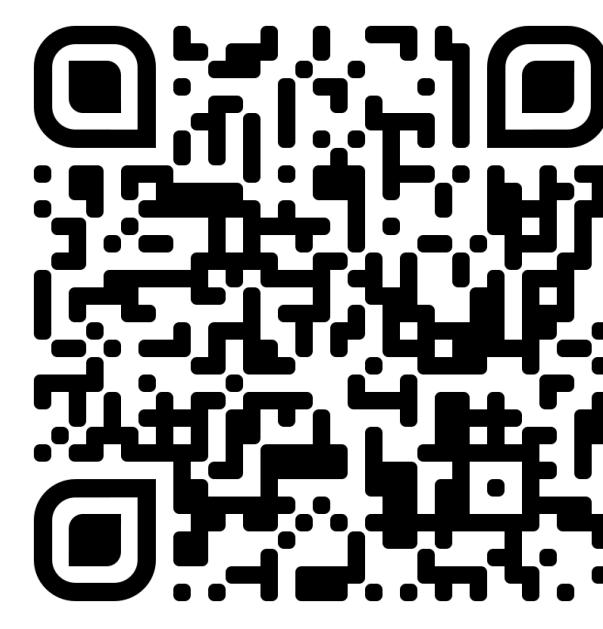
E' poi possibile visualizzare come il tempo di esecuzione vari in base al numero di core disponibili per l'elaborazione.

- I risultati possono essere esportati per ulteriori elaborazioni



# Implementazione

Il codice è scritto in C++ 20 e utilizza le librerie **ImGUI** e **SDL2** per il rendering grafico. Il codice è disponibile su github per chi fosse interessato



```
#include "memory.h"

#include <format>

Error<Memory> Memory::init(Context ctx, size_t size) {
    cl_int err;
    this->mem = clCreateBuffer(
        ctx.get_context(),
        CL_MEM_READ_WRITE | CL_MEM_ALLOC_HOST_PTR,
        size, NULL, &err);
    if (err != CL_SUCCESS) {
        return Error<Memory>();
    }
    this->command_queue = ctx.get_queue();
    this->size = size;
    return Error<Memory>().set_value(*this);
}

Error<Unit> Memory::write(void *data) {
    if (data == nullptr)
        return Error<Unit>().set_error("nullptr");
    cl_int err = clEnqueueWriteBuffer(
        command_queue, mem, CL_TRUE, 0,
        size, data, 0, NULL, NULL);
    if (err != CL_SUCCESS) {
        return Error<Unit>();
    }
    return Error<Unit>().set_value(unit_value);
}

Error<uint8_t *> Memory::read() {
    uint8_t *data = new uint8_t[size];
    cl_int err = clEnqueueReadBuffer(
        command_queue, mem, CL_TRUE, 0,
        size, data, 0, NULL, NULL);
    if (err != CL_SUCCESS) {
        return Error<uint8_t *>();
    }
    return Error<uint8_t *>().set_value(data);
}
```



# DEMO

Avete delle domande?