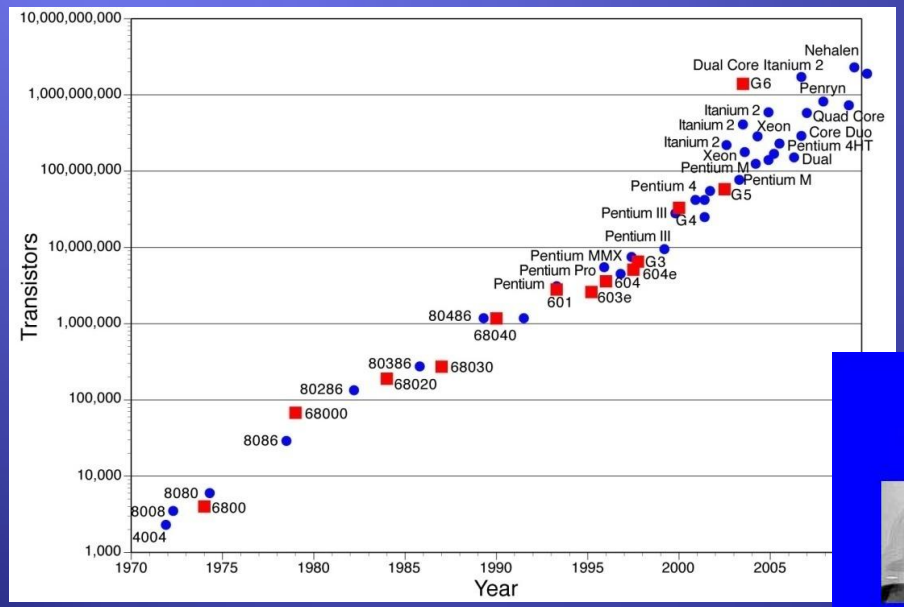
# Coding for the GPU: highlights and application examples

SuperB Computing R&D Workshop
Ferrara 10 Mar 2010

Dr. Leone B. Bosi
INFN Perugia
MACGO/ET Project

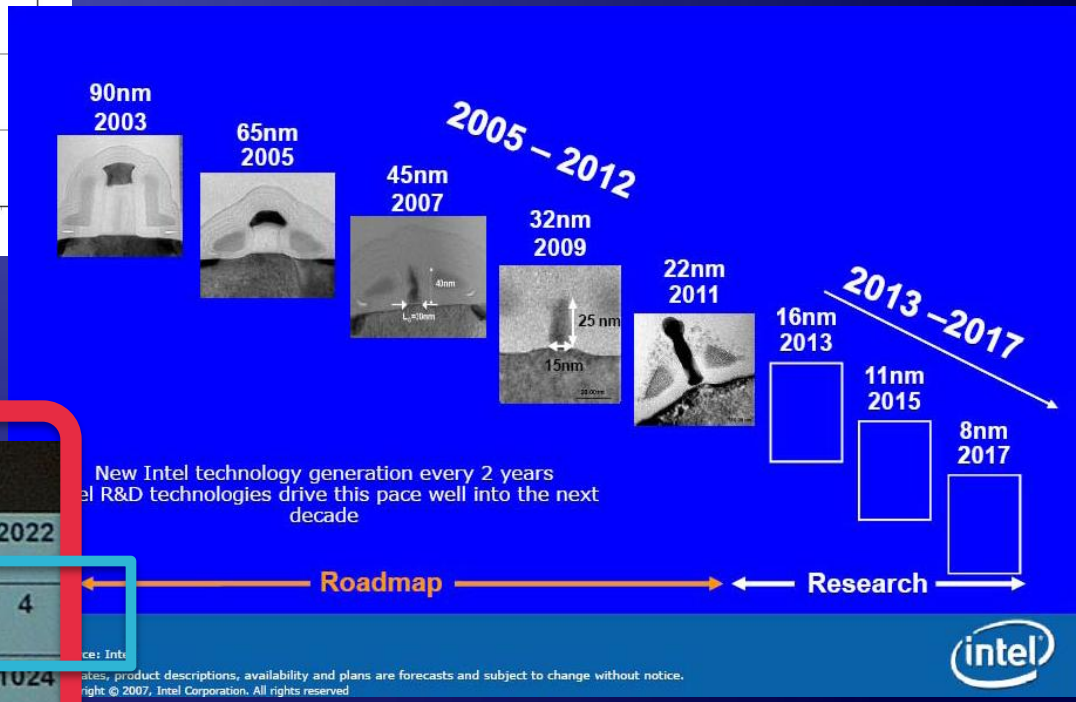
# Technological outlook:




**Is the Moore's Law close to its limit?**

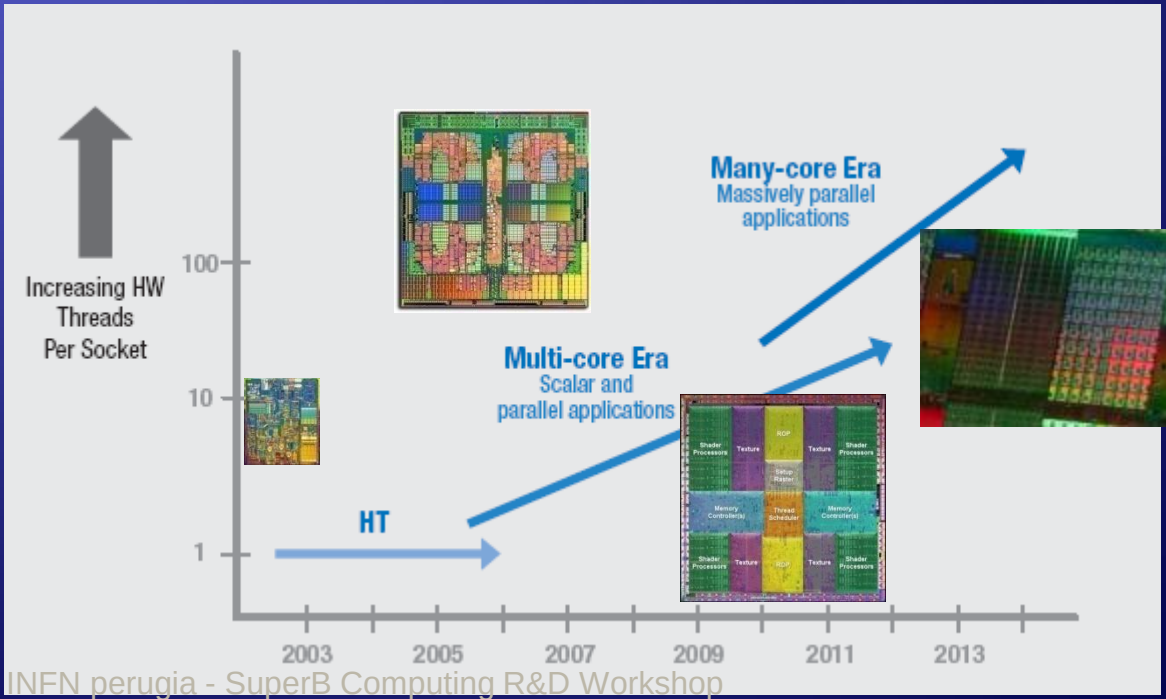
Intel ref.

| Technology Outlook | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| High Volume Manufacturing | 2008 | 2010 | 2012 | 2014 | 2016 | 2018 | 2020 | 2022 |
| Technology Node (nm) | 45 | 32 | 22 | 16 | 11 | 8 | 6 | 4 |
| Integration Capacity (BT) | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |



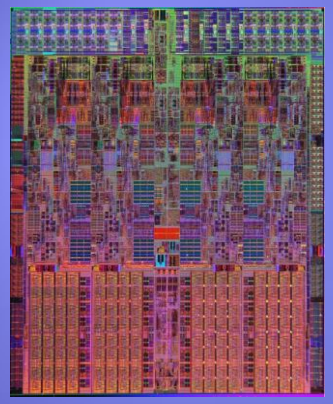
# Technological outlook:

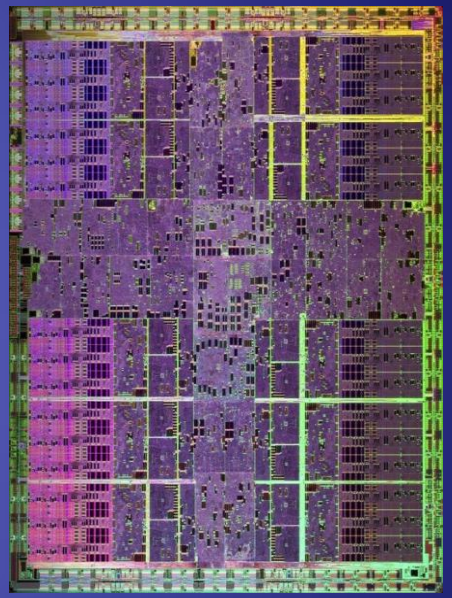
- Most important chip semiconductor maker are working in order to limit the problems due to integration scale reduction.
- In the last 10 years  processor architectures  are changed a lot, introducing parallelization at several architectural levels.
- That evolutive process will continue in a deeper way,  moving to the so called “many-core” era.




Intel ref.

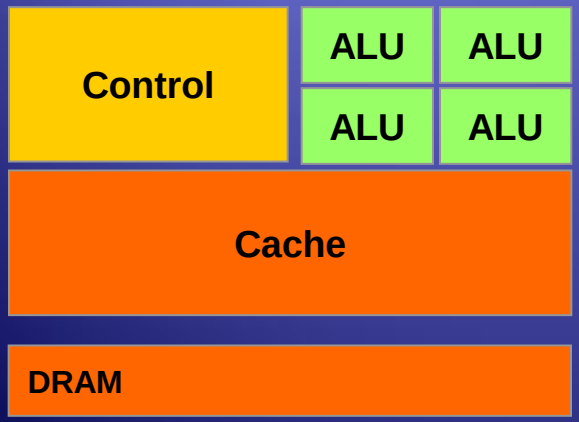
# What is GPU Computing?



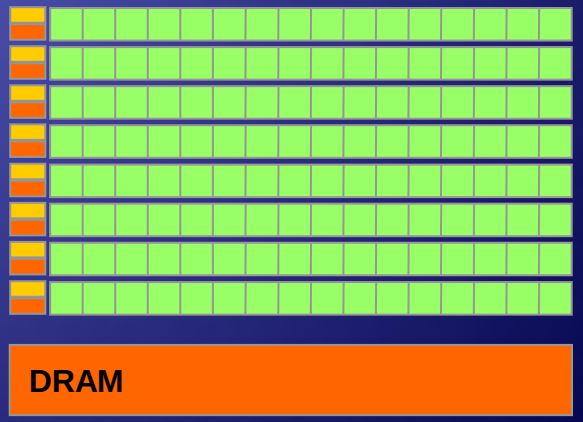


CPU

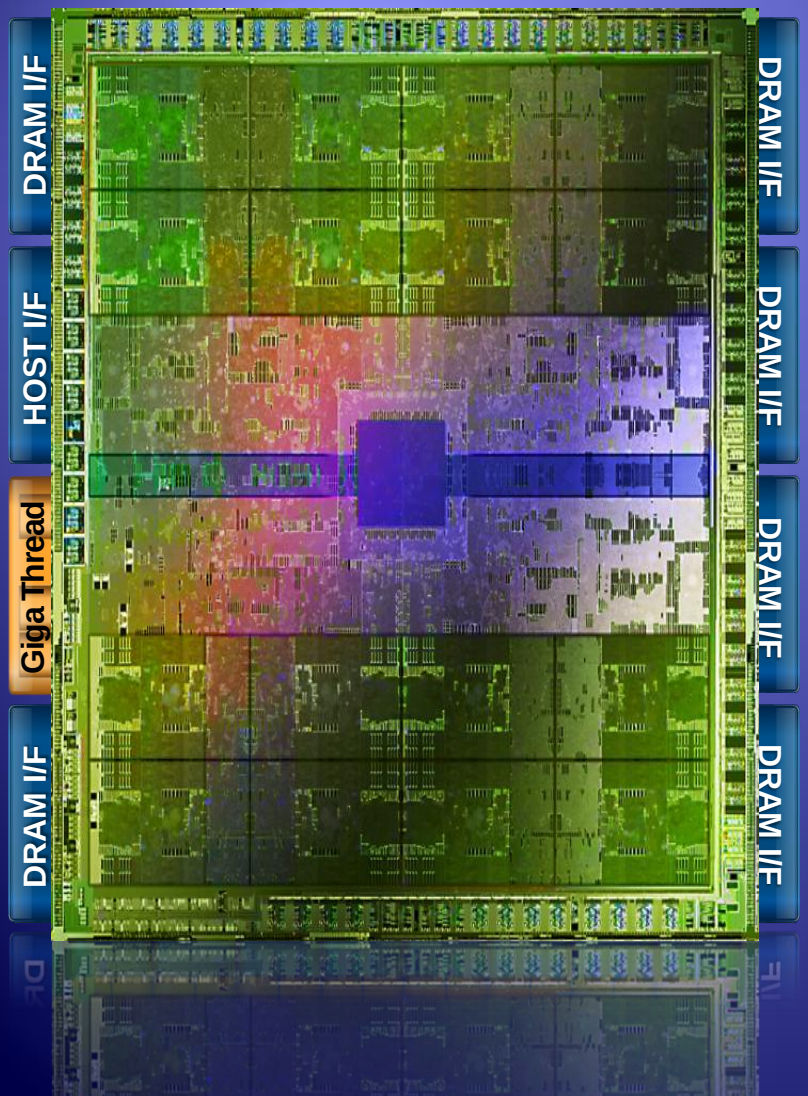
GPU




PCIe bus

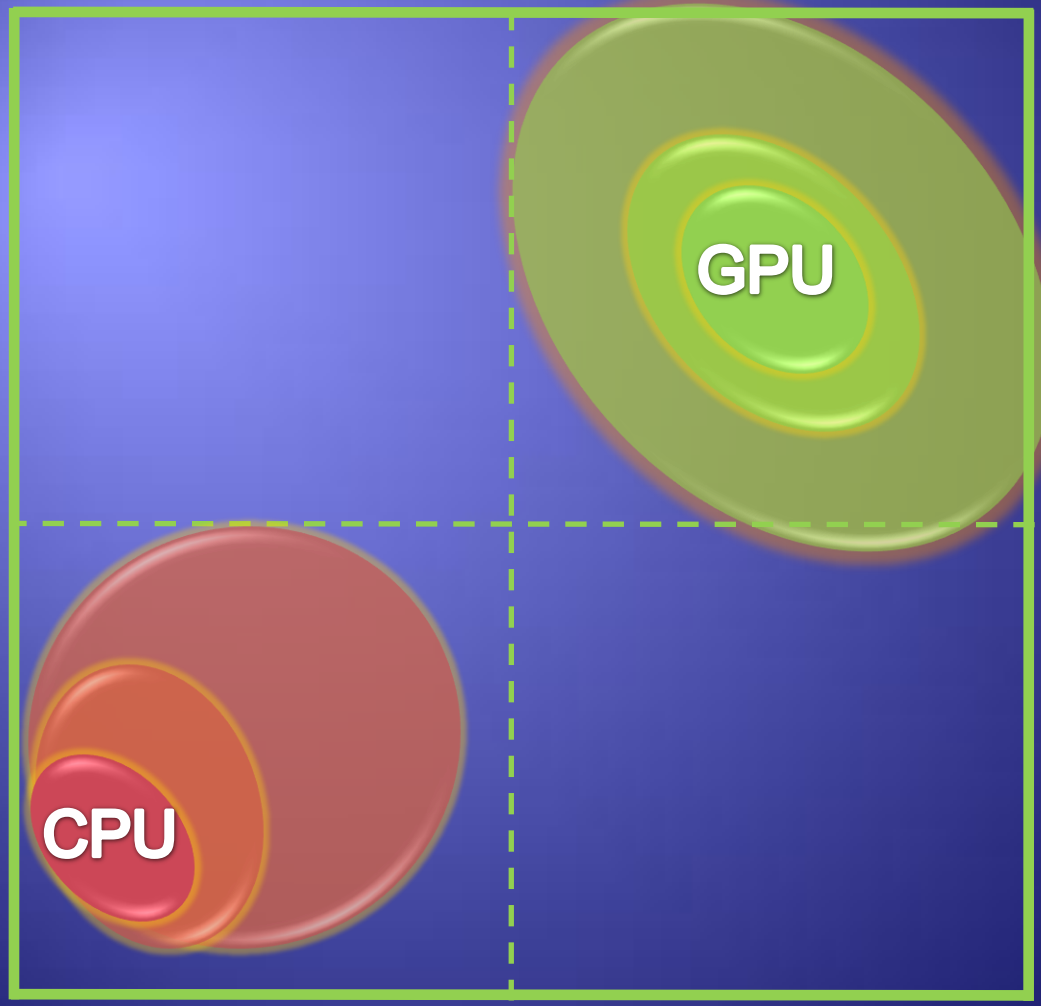


# NVIDIA Status of art: FERMI Processor



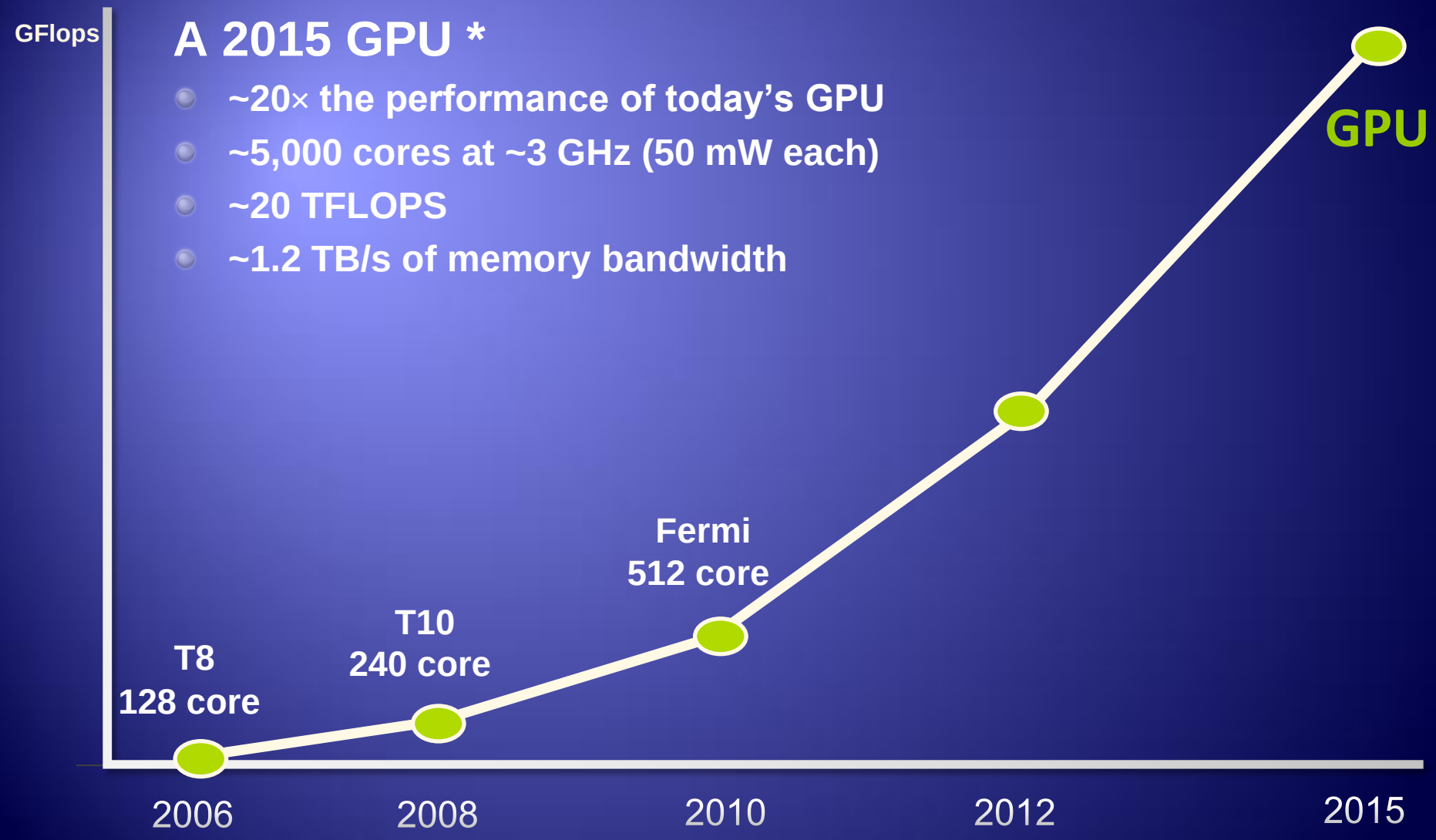

- 3 billion transistors
- Over 2× the cores (512 total)
- 8× the peak DP performance
- ECC
- L1 and L2 caches
- ~2× memory bandwidth (GDDR5)
- Up to 1 Terabyte of GPU memory
- Concurrent kernels
- Hardware support for C++

# NVIDIA: Design Goal of Fermi



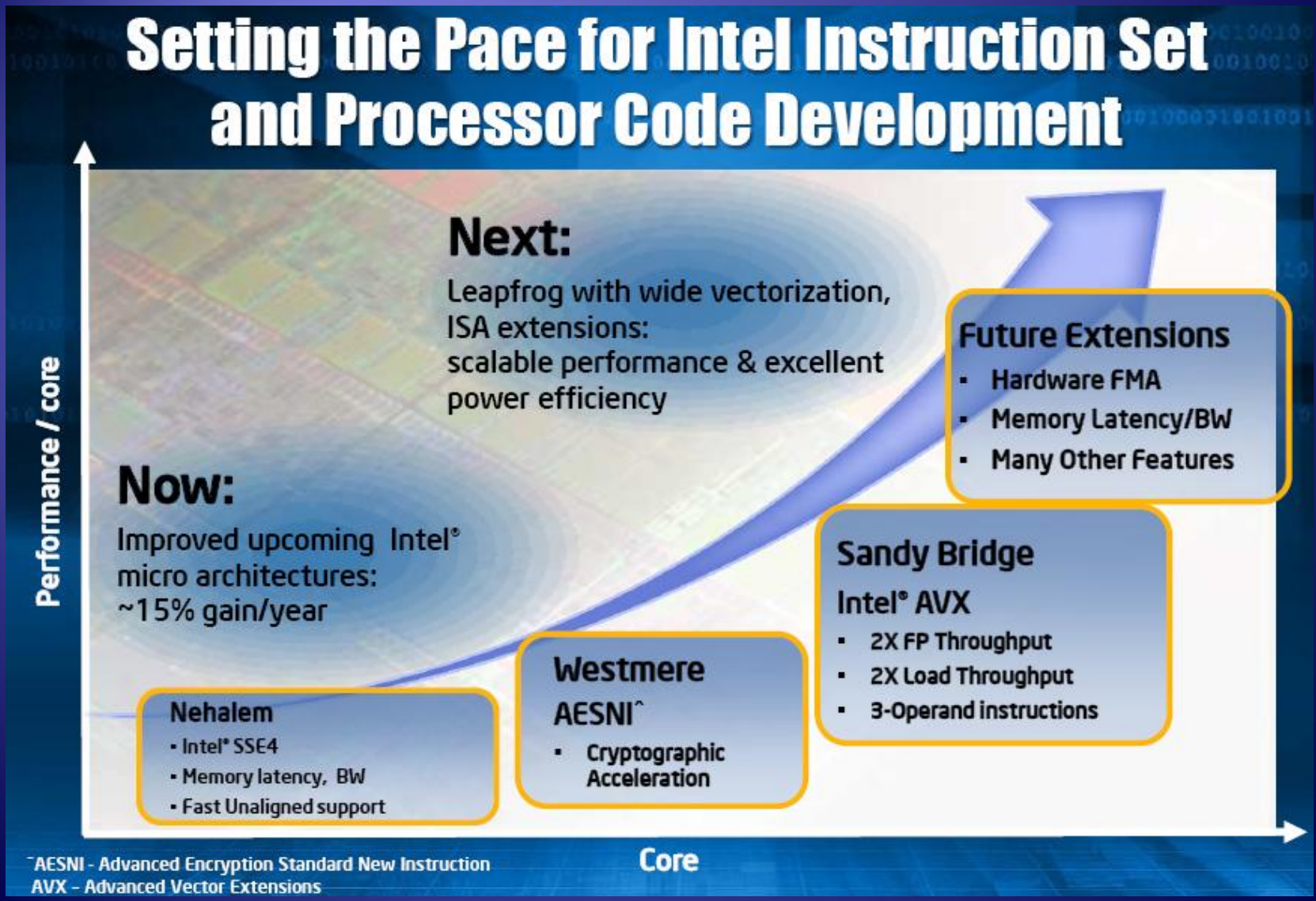

- Expand performance sweet spot of the GPU
- Bring more users, more applications to the GPU

# NVIDIA Investments next 5 years…

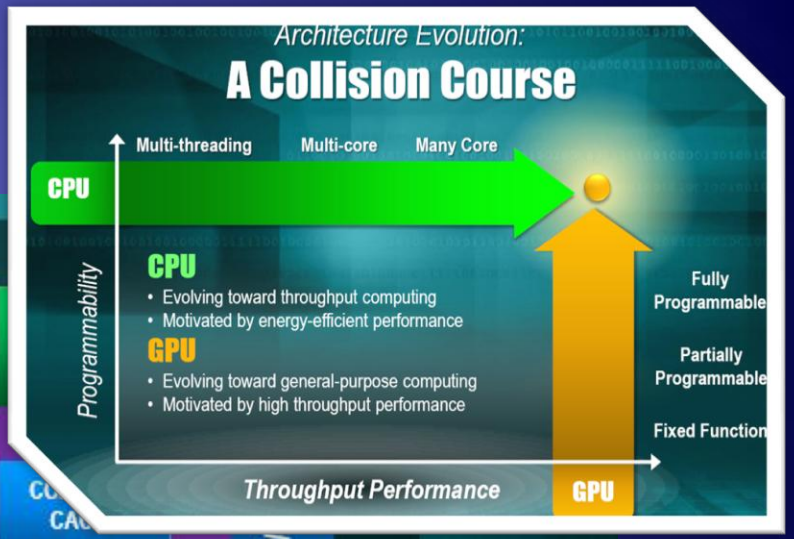



* This is a sketch of a what a GPU in 2015 might look like; it does not reflect any actual product plans.

# Intel point of view

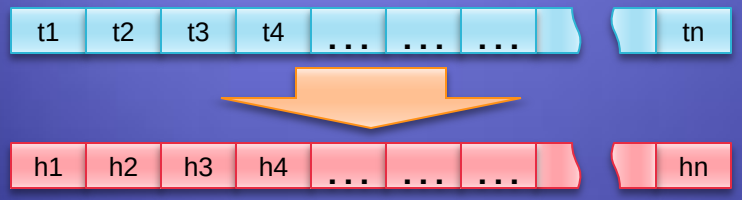


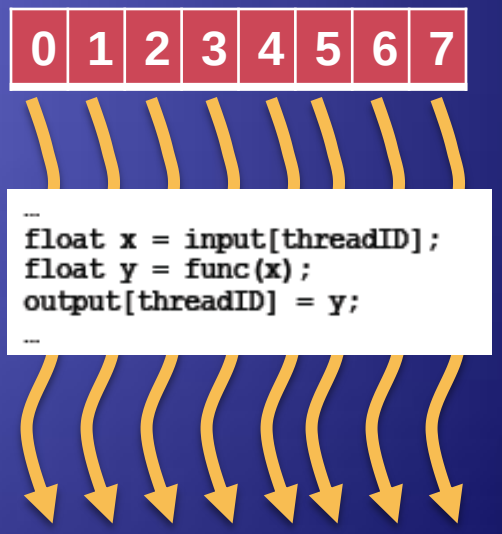
# Intel point of view: Larrabee Processor
## A computational Co-processor for the Intel Xeon and Core Families



# Signal generation on GPU

- Signals generation is a typical computing problem that maps very well  the GPU architectural model.




- Each GPU thread computes a time sample

# Cuda simple example: kernels

A kernel is defined via **__global__**

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```

```
int main()
{
    ...
    // Kernel invocation
    VecAdd<<<1, N>>>(A, B, C);
}
```

N CUDA Threads are defined with **<<< … >>> syntax**

Ref: CUDA Programming guide

# Cuda Example

Host memory allocation

Device memory allocation

Host to device memory copy

Def: blocksPerGrid & ThreadsPerBlock

Start kernel

Copy results Device to Host

```
// Device code
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}

// Host code
int main()
{
    int N = ...;
    size_t size = N * sizeof(float);

    // Allocate input vectors h_A and h_B in host memory
    float* h_A = malloc(size);
    float* h_B = malloc(size);

    // Allocate vectors in device memory
    float* d_A;
    cudaMalloc((void**)&d_A, size);
    float* d_B;
    cudaMalloc((void**)&d_B, size);
    float* d_C;
    cudaMalloc((void**)&d_C, size);

    // Copy vectors from host memory to device memory
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    // Invoke kernel
    int threadsPerBlock = 256;
    int blocksPerGrid =
            (N + threadsPerBlock - 1) / threadsPerBlock;
    VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C);

    // Copy result from device memory to host memory
    // h_C contains the result in host memory
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
```

Kernel definition

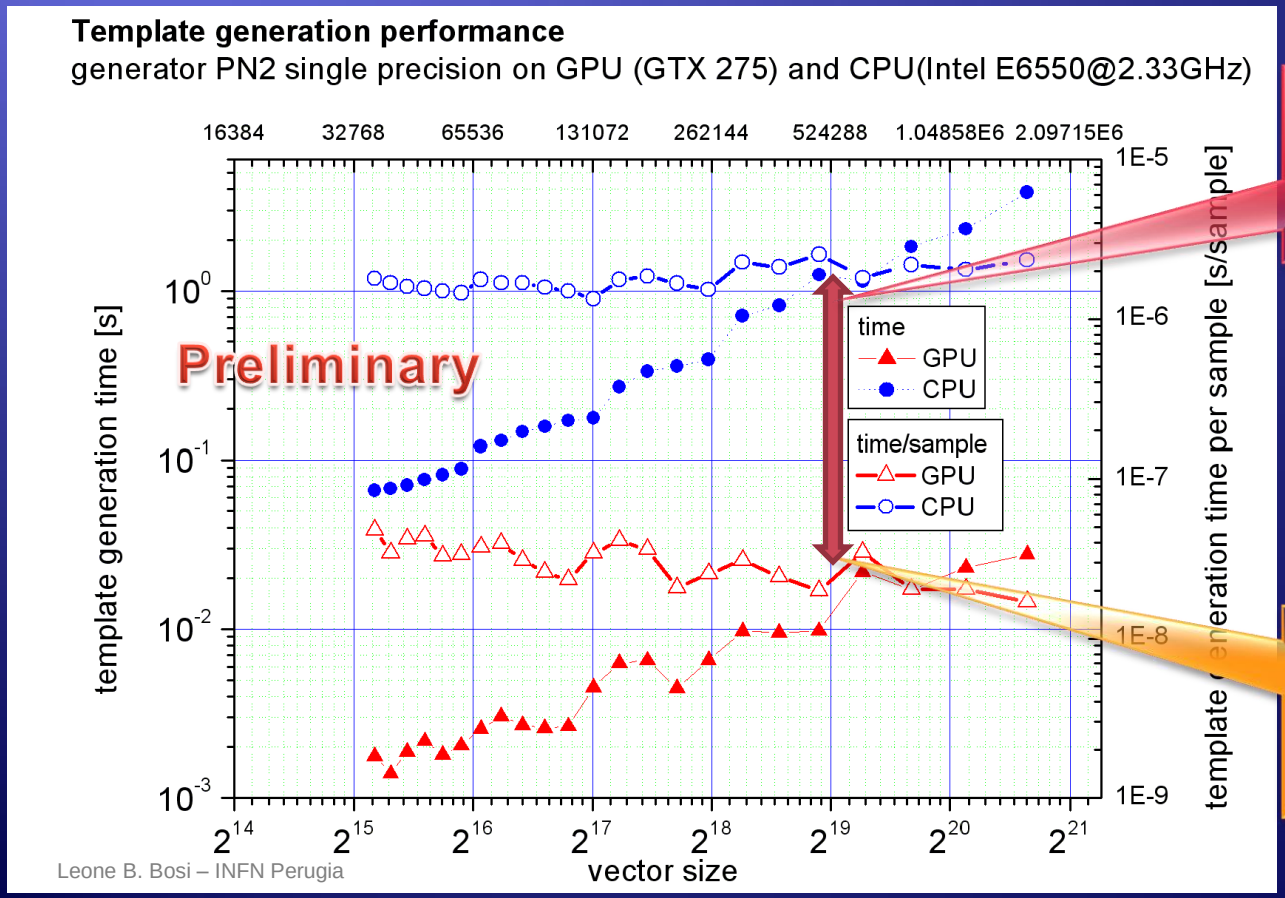
Ref: CUDA Programming guide

# MaCGO: *Manycore Computing for future Gravitational Observatories*

- MaCGO is an INFN V:
  - Dr. Leone B. Bosi coordinatore (INFN Perugia) - Dr. Michele Punturo (INFN) - Dr. Leonello Servoli (INFN) - Dr. O. Gervasi – Dip. Informatica, Univ. Perugia – Prof. Laganà Antonio - Dip. Chimica, Univ. Perugia
- Some project items:
  - Explore and Develop of dedicated algorithms for Multi-GPU/devices configuration
  - Explore and use a general programming language OpenCL
  - Production of a first release of a numerical library for GW-DA on manycore architecture.

More Info: http://macgo.pg.infn.it

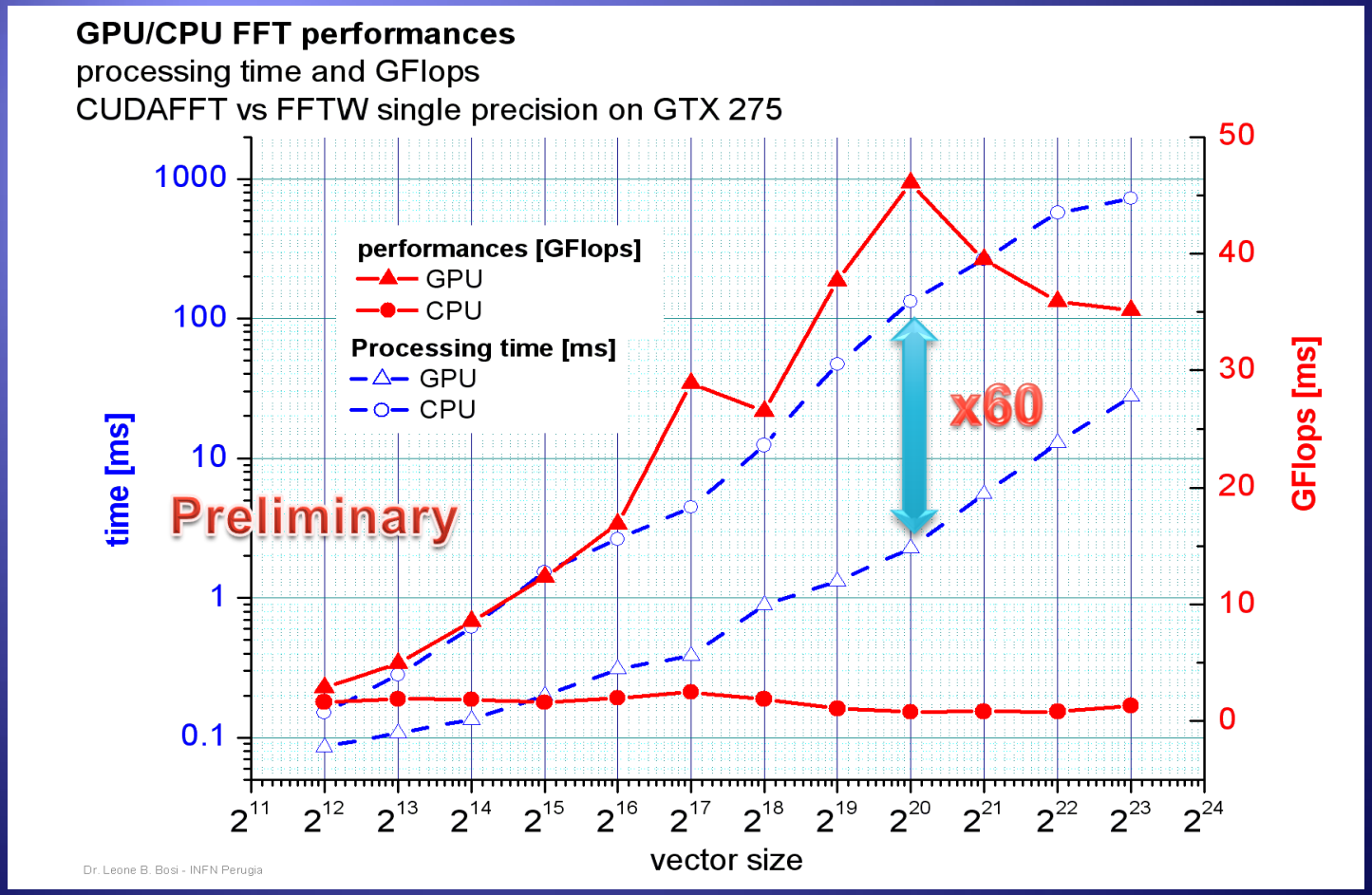
# Signal generation [ performance ]



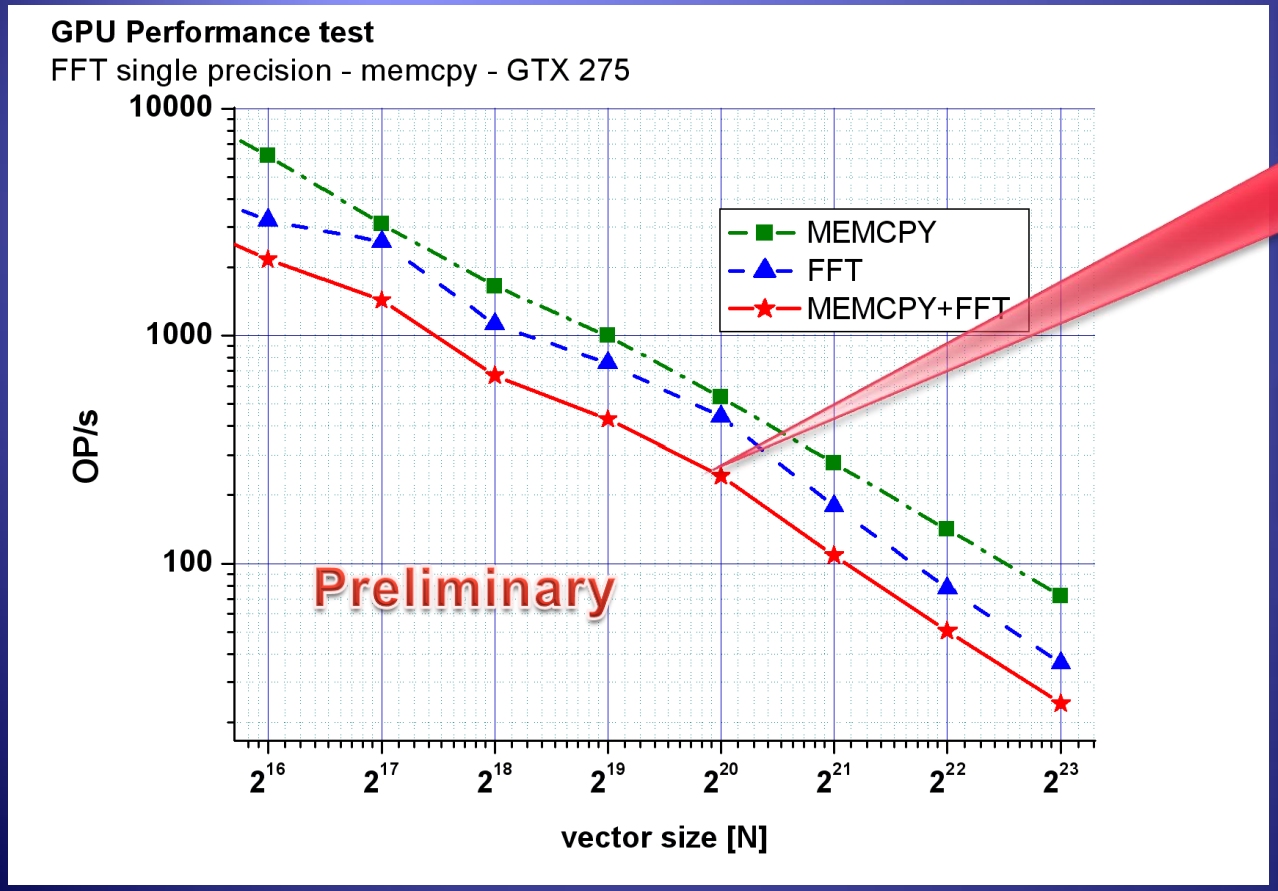

gain = X100

2-3 $10^{-8}$ s/sample

# CUDAFFT vs FFTW: proc time | GFlops (single precision)



# Host←→Device Memory I/O overhead




cuFFT wrapping approach looses performances due to memory IO operations

**Host-Device memory IO kills performances**

**GPU express best performances in high arithmetic intensity conditions.**

# Multi GPU example (and OpenMP)

```
omp_set_num_threads(data->deviceCount);
#pragma omp parallel
{
    unsigned int cpu_thread_id = omp_get_thread_num();
    unsigned int num_cpu_threads = omp_get_num_threads();
    int d = cpu_thread_id % data->deviceCount;
    cudaSetDevice(d);
    Some code, containing kernels
}
```

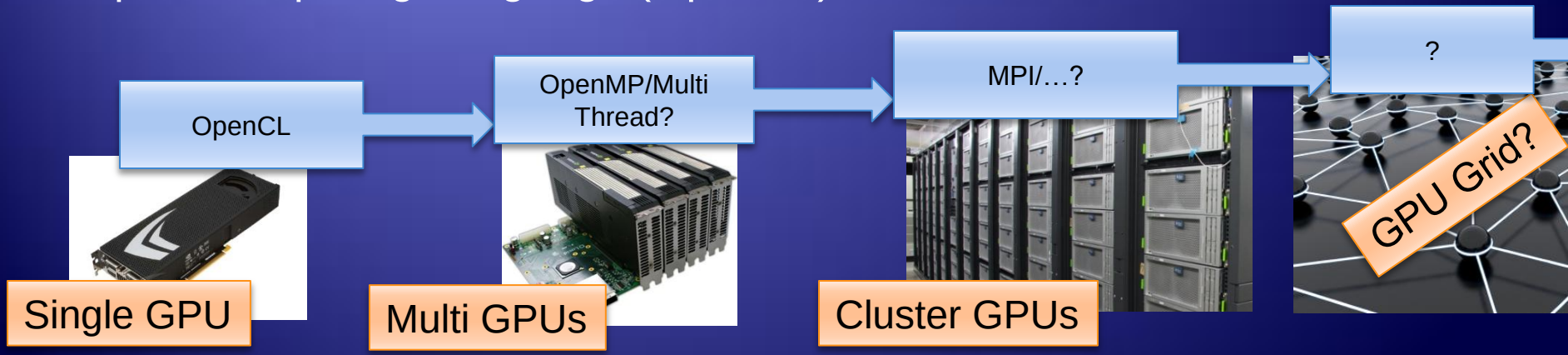
Si seleziona il device tramite: **__cudaSetDevice__**

Le istruzioni CUDA sono dirette verso il device selezionato

openMP note: overhead nella creazione dei thread

# GPU computing and programming paradigms

- The architectural differences between GPU and CPU are evident, in particular the way how the relations between cores, memory, shared memory and IO subsystem are organized
- Moreover different chip producers implement different solutions with different characteristics and instructions sets
- Recently, several important efforts have been done by Apple, Intel, NVIDIA , AMD-Ati, Sony, … in the direction of programming standardization for parallel architecture: The Khronos Group, and the Open Computing Language (OpenCL) definition.



# Cuda vs OpenCL code

C for CUDA Kernel Code:

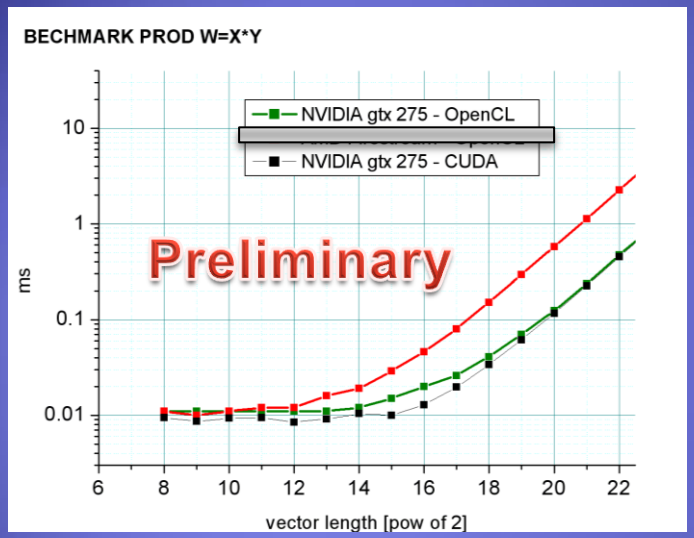
```
__global__ void
vectorAdd(const float * a, const float * b, float * c)
{
    // Vector element index
    int nIndex = blockIdx.x * blockDim.x + threadIdx.x;

    c[nIndex] = a[nIndex] + b[nIndex];
}
```

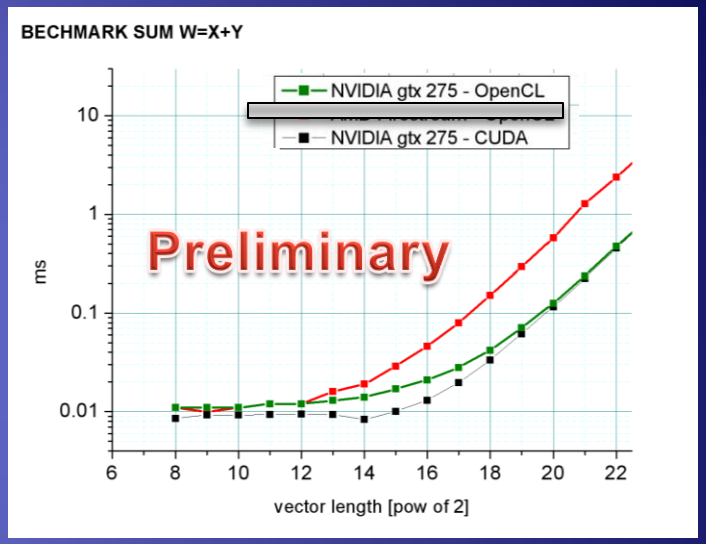
OpenCL Kernel Code

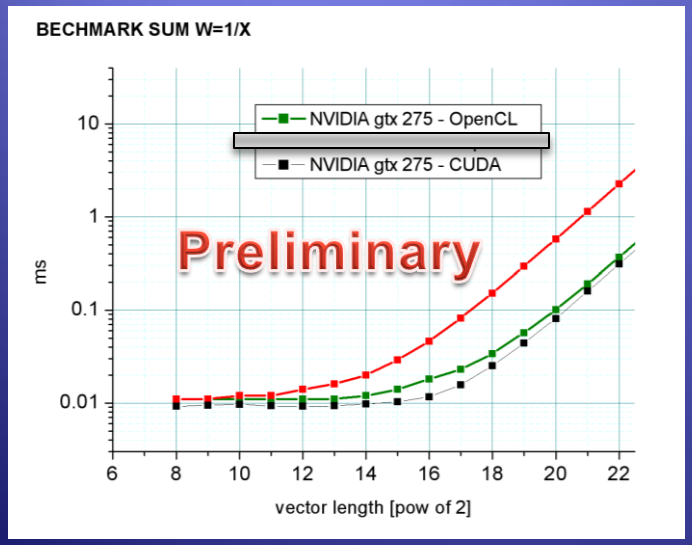
```
__kernel void
vectorAdd(__global const float * a,
          __global const float * b,
          __global       float * c)
{
    // Vector element index
    int nIndex = get_global_id(0);

    c[nIndex] = a[nIndex] + b[nIndex];
}
```

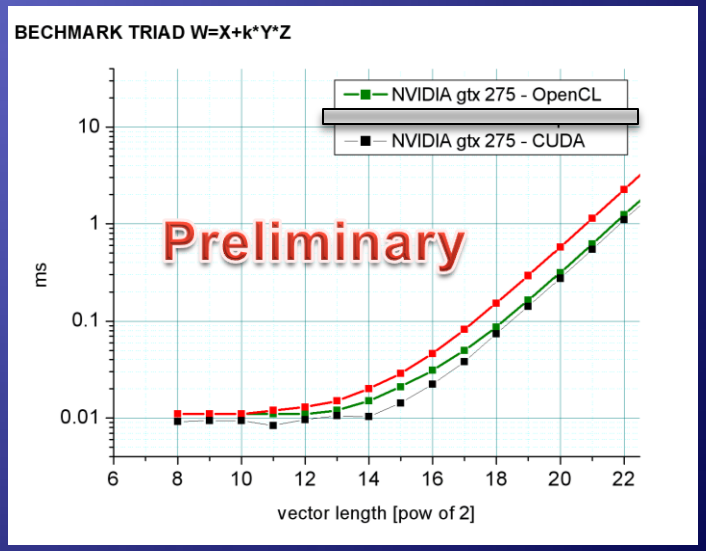
Ref: CUDA OpenCL JumpStart guide

# CUDA vs OpenCL (GTX 275/FireStrem 9270)



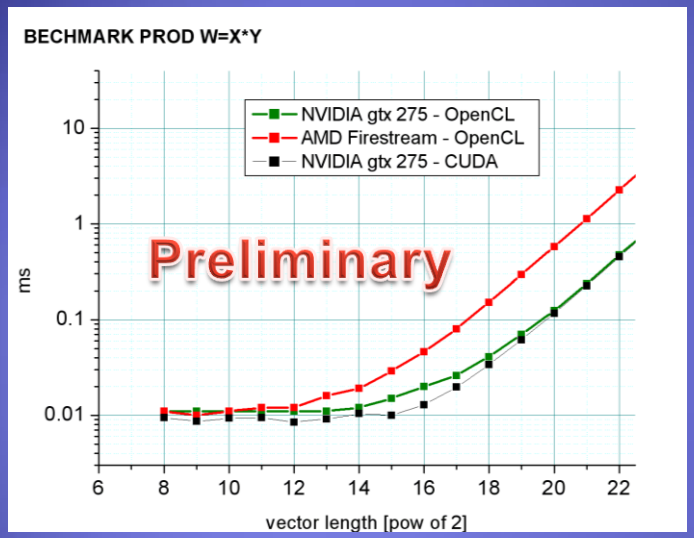



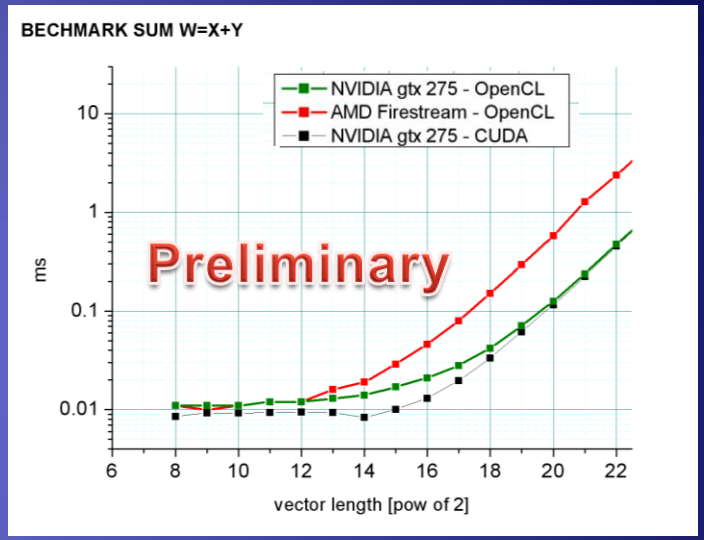



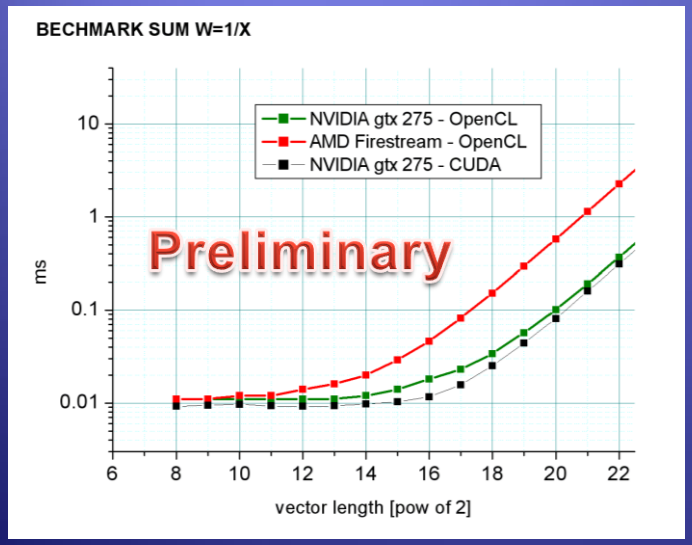



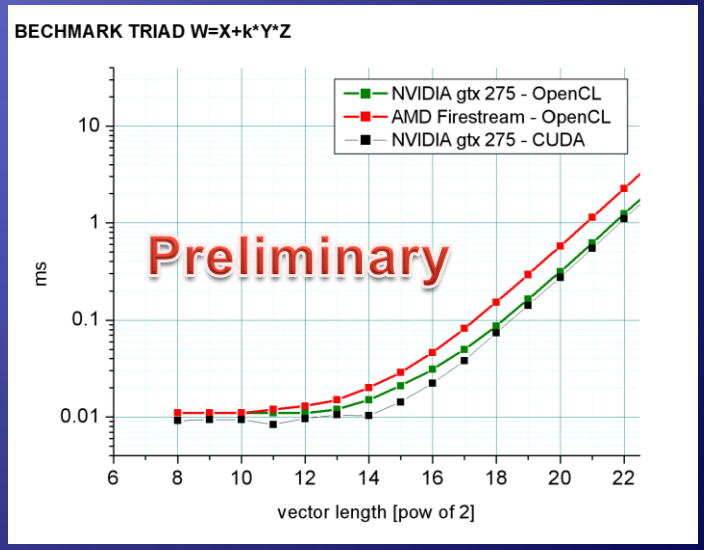

Benchmark made by Dr. Flavio Vella – INFN Perugia@MaGGO

# NVIDIA vs ATI-AMD (GTX 275/FireStrem 9270)




Benchmark INFN Perugia@MaGGO

# INFN on GPU: Some results

- **GPU computing for 2-d spin systems: CUDA vs OpenGL**. (**Viola Anselmi,** *university of Parma ,* **Giovanni Conti** *University of Parma,* **Francesco Di Renzo** *INFN)* → **X30-40**
- **Simulazioni Monte Carlo Simulations di Sistemi di Spin Glass Systems su architetture Multi-core** - S.F.Schifano (fe)
  - Tested Intel, Cell, Nvidia solutions.
  - Significative speed up on "3D Heisenberg Model SUT", **T1060 vs I7 1.6GHz with → X10**.
- **Lattice QCD on GPUs** - M. D'Elia et al.
  - sustained 60 GFLOPs @ T1060
  - **1 T1060 = 20-30 cores**
- **GPU trigger in HEP** (NA62) - G.Lamanna (INFN Pi) at al.
  - **3.1 us/evento** -
  - trigger di L0 xNA62 → **30GPU**
- **Manycore computing for future gravitational observatories** - L.Bosi et al. (INFN Perugia)
  - CB GW Pipeline Detection → **X50**
  - OpenCL/CUDA and ATI/NVIDIA hardware tests

# Others HEP on GPU results:

- **Fast calculation of HELAS amplitudes using graphics processing unit (GPU) -** Hagiwara, K et al. → **X 40-150**
- **Development of a GPU-based Monte Carlo dose calculation code for coupled electron-photon transport (Xun Jia1 et al.)) → X5**
- **GPU-accelerated Monte Carlo simulation for photodynamic therapy treatment planning. (William Chun Yip Loa et al.) → X40-250**
- **High performance stream computing for particle beam transport simulations (R. Appleby, D. Bailey, J. Higham, M. Salt) → X10**

**Geant4 porting on GPU Nvidia platform -** Caccia(INFN)

- **Multi-level Parallel fit algorithms using MPI and CUDA** - K.Tomko et al.
- http://gpgpu.org/tag/monte-carlo-simulation