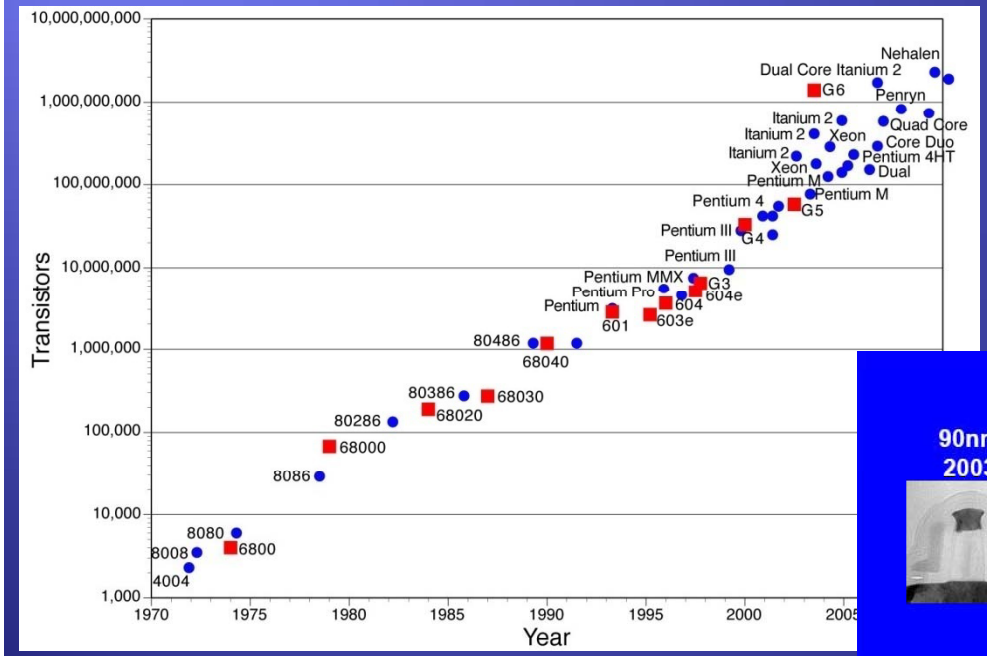
MaCGO: esperienze CUDA, OpenCL e MultiGPU realizzazione libreria di data analisi per detectors GW di 3° generazione.

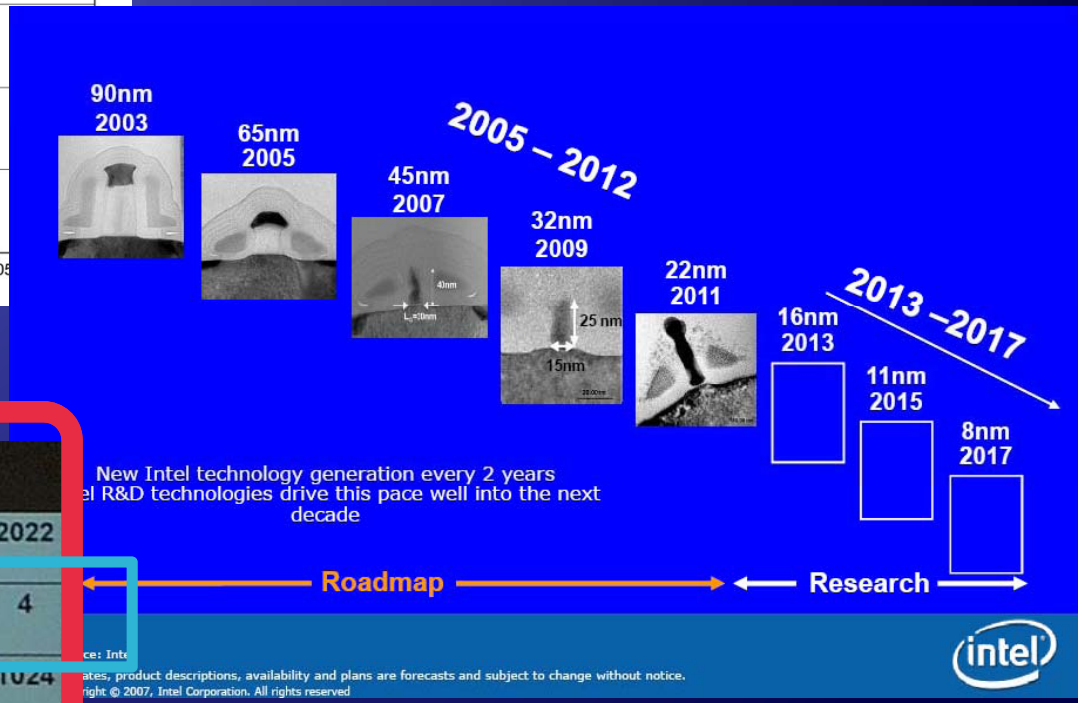
Dr. Leone B. Bosi – INFN Perugia
on behalf of MaCGO team

Dr. Leone B. Bosi coordinatore (INFN) - Dr. Michele Punturo (INFN) - Dr. Leonello Servoli (INFN) - Dr. O. Gervasi , Dip. Informatica, Univ. Perugia – Prof. Laganà Antonio, Dip. Chimica, Univ. Perugia

Workshop CCR– Napoli Italy
26 Gennaio 2010

Technological outlook:





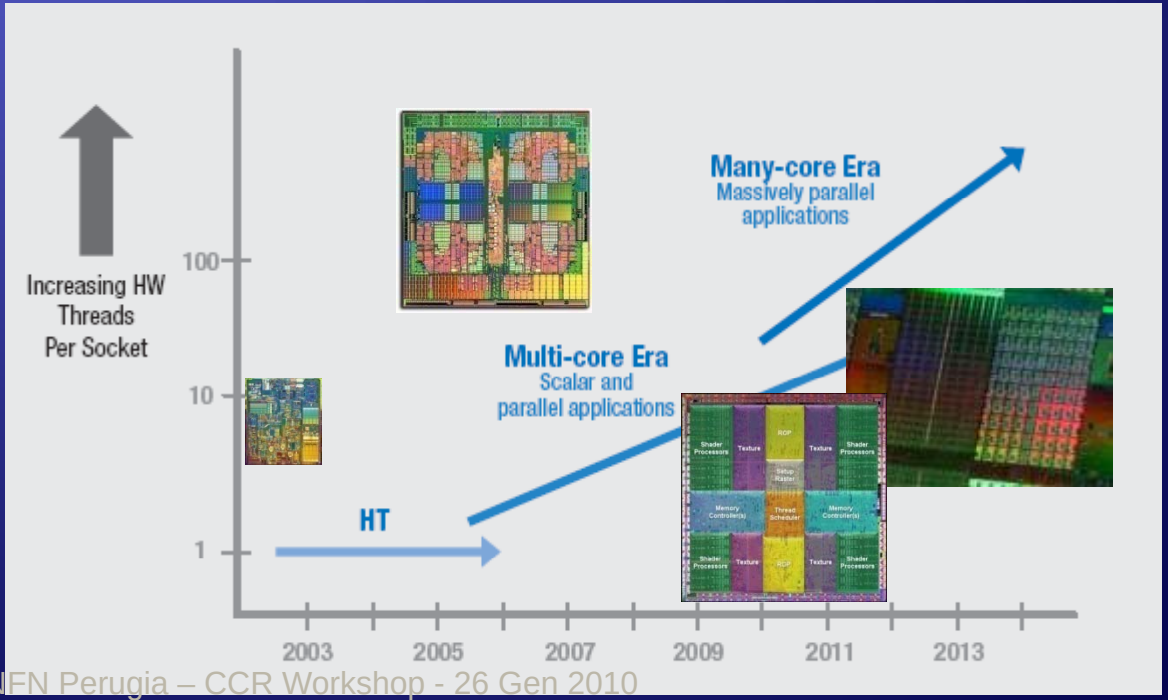
Technology Outlook

High Volume	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

Intel ref.

Technological outlook:

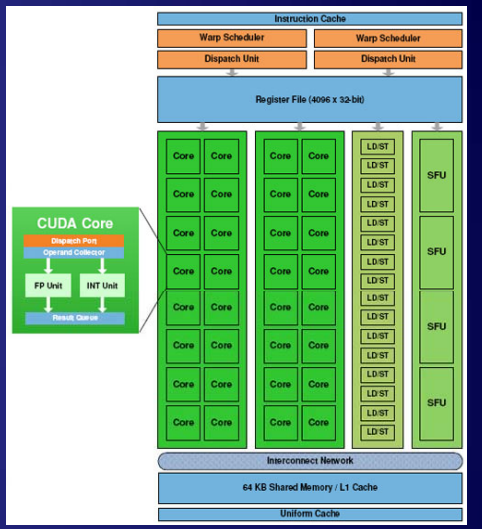
- Most important chip semiconductor maker are working in order to limit problems due to integration scale reduction.
- In fact last 10 years the processors architectures are changed a lot, introducing parallelization at several architectural levels.
- That evolutive process will continue in a deeper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

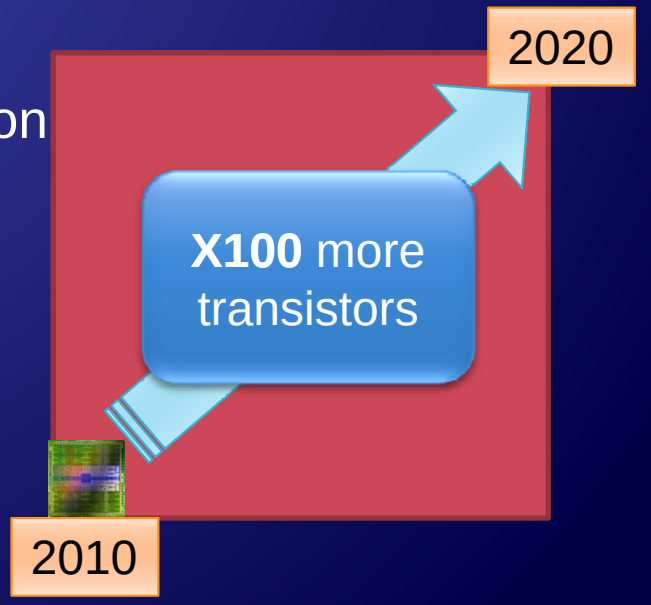


Intel ref.

Some performance considerations:

- These new architectures require a complete different programming models.
- In the manycore era, computing power will be distributed across thousands cores on a single die, and many processors on a single board.
- Performance achievable from these architecture is not predictable, depends on algorithms and:
  - Memory/registries architecture model
  - Inter-communication
  - Serial portion of the algorithm

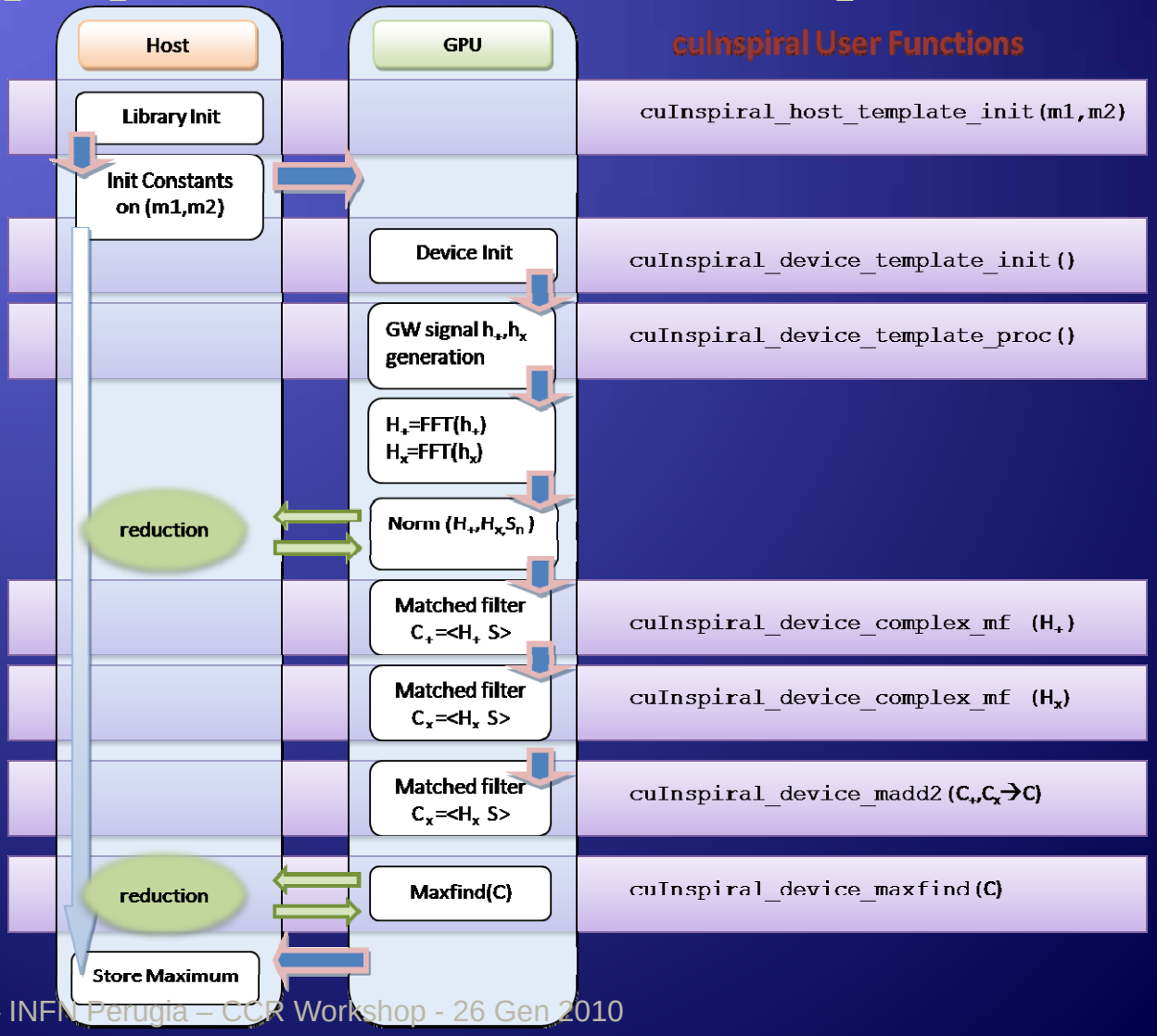




cuInspiral experience:
GPU CB library prototype

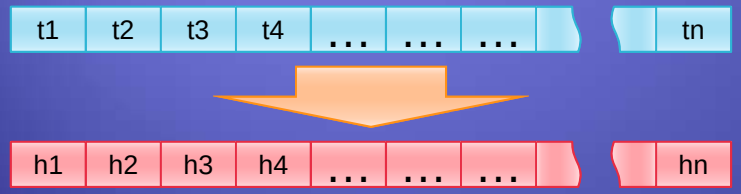
- cuInspiral was under develop in Virgo/ET INFN Perugia, using CUDA framework, now merged in **MaCGO** (***Manycore Computing for future Gravitational Observatories***) experiment.
- The first library implementing a high arithmetic intensity strategy on GPU for coalescing binaries DA. All computation is made completely inside GPU space.
- Functions :
  - Taylor PN2 generator
  - Normalization
  - Matched filtering
  - Maximum identification
  - Other complex vector operations

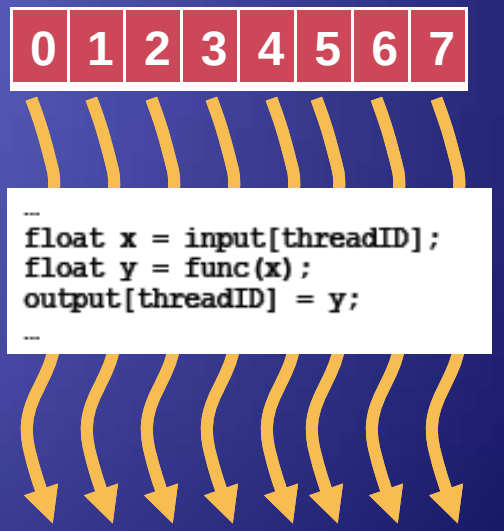
cuInspiral: CB pipeline description



Templates 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

Un kernel è definito tramite **__global__**

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

Ad ogni thread che esegue un kernel è assegnato un id che è accessibile tramite threadIdx

In questo esempio ogni thread elabora dati referenziati tramite il proprio threadIdx, che il sistema assegna in maniera sequenziale partendo da 0

Dove N è la dimensione del nostro vettore

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

The numero di Thread CUDA è definito con la sintassi: **<<< … >>>**

Ref: CUDA Programming guide

Cuda Example

Host memory allocation

Device memory allocation

Host to device memory copy

Def: blocksPerGrid & ThreadsPerBlock

Start kernel

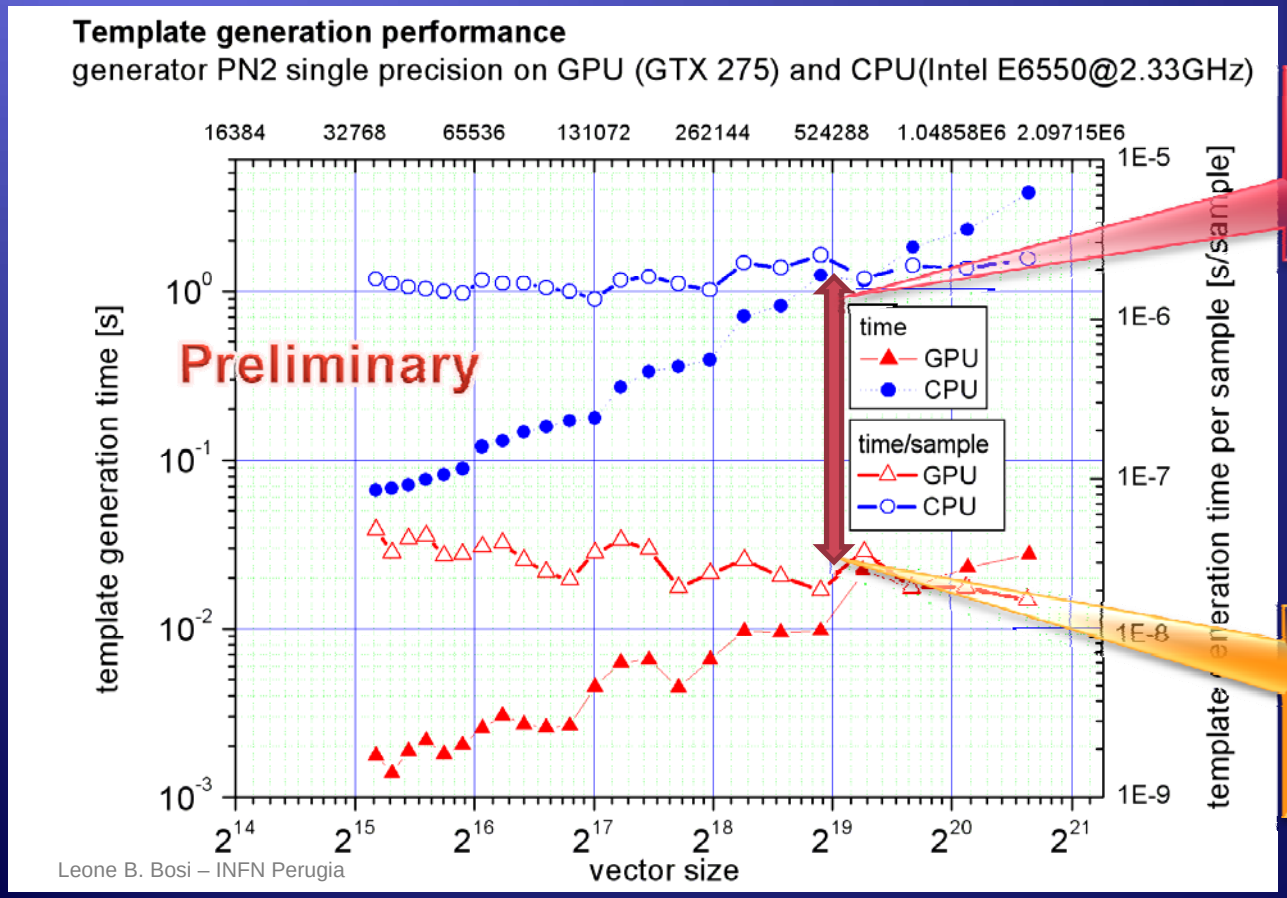
Copy results Device to Host

```
// 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

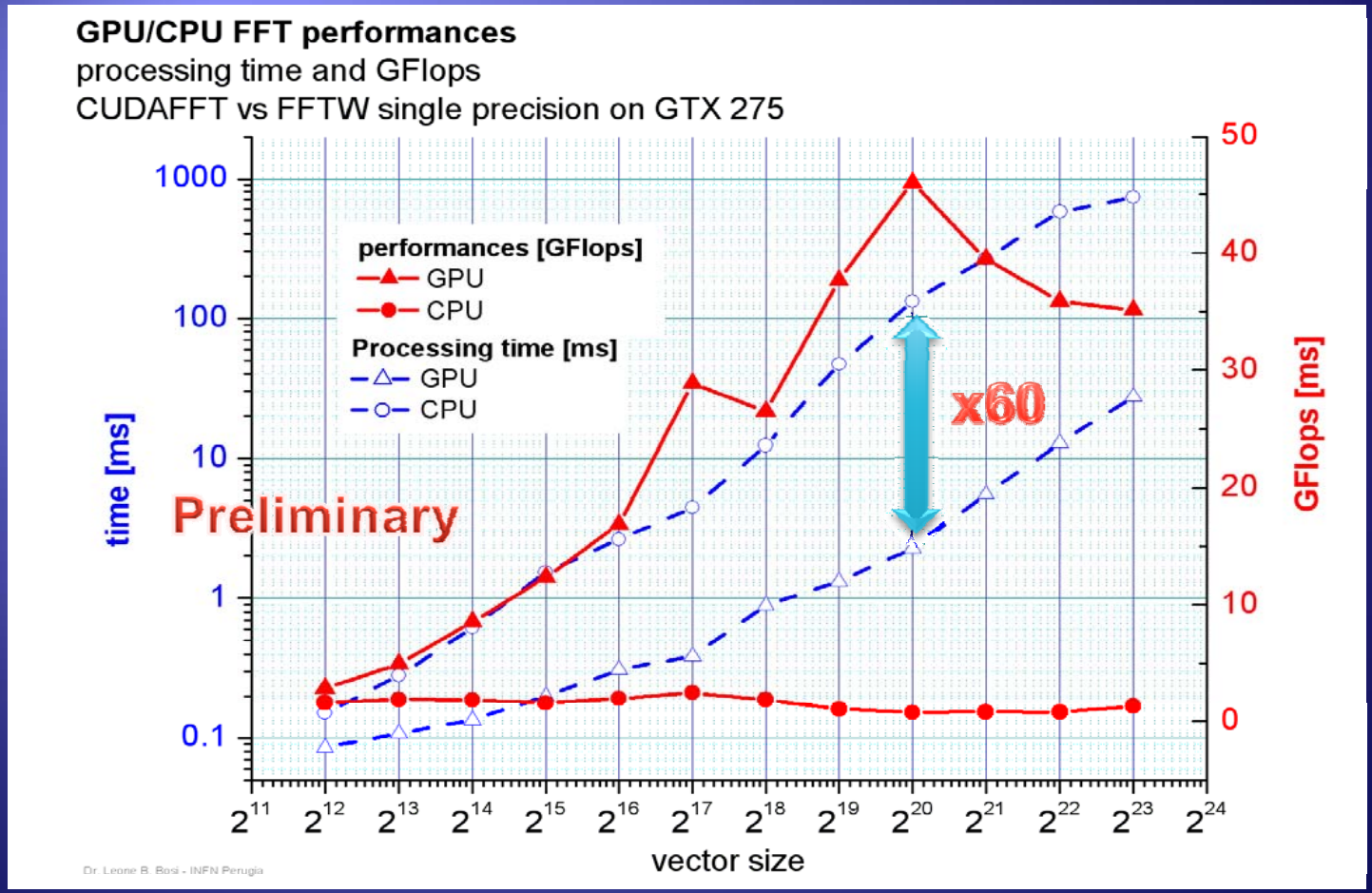
PN2 template generation [performance]



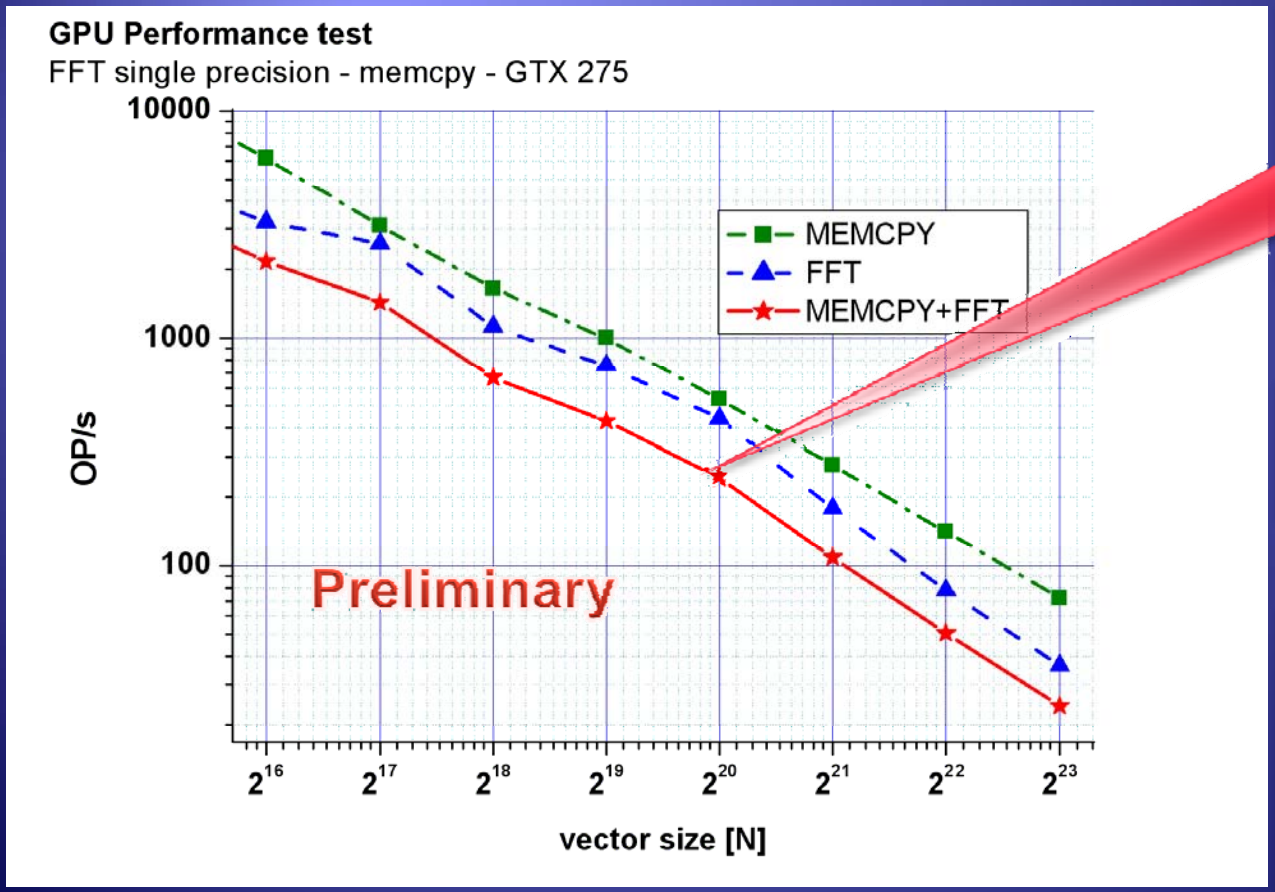
gain = X100

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.

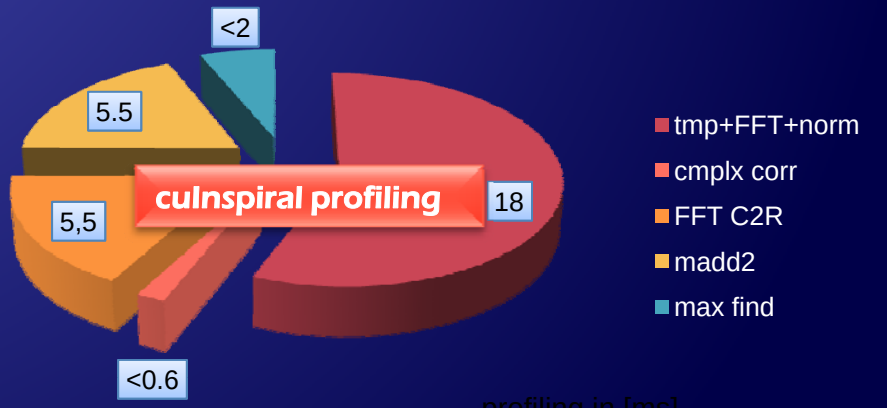
Pipeline profiling:

- If we consider analysis parameters of :
  - low.cutof.freq:24Hz,
  - vector length 2^20, fs=4kHz the
- The cuInspiral processing rate on GTX 275 is roughly of:

  30 TEMPLATES/SEC (lower limit)
- If we consider the online constrain processing with 4000 templates, we can estimate that analysis (detection only) can be performed with a couple of GPU 275 (=500 Euro), instead of 80 CPU

Pipeline Gain (lower limit):
>x50 with GTX 275
→ expected with Fermi GPU: **x150**

Multi-GPU configuration: x DevN



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

Multi GPU example (and OpenMP)

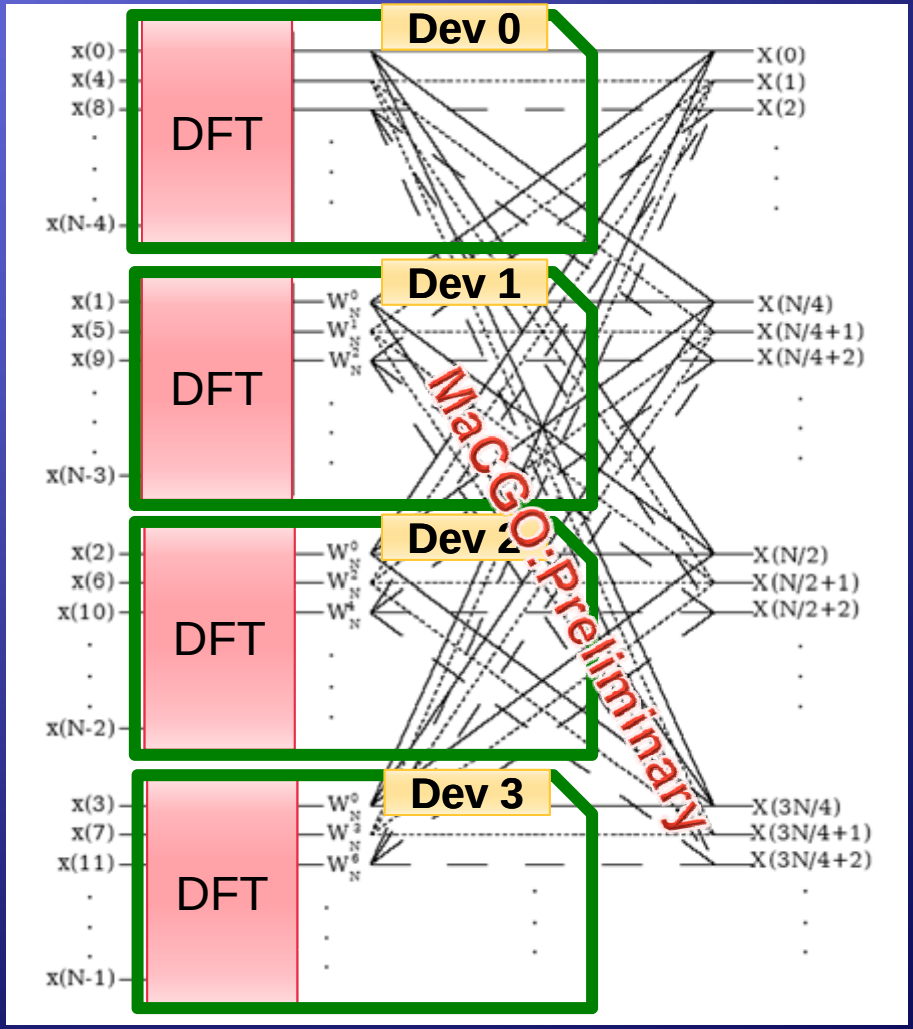
```
omp_set_num_threads(data->deviceCount);
#pragma omp parallel
{
    unsigned int cpu_thread_id = omp_get_thread_num();
    unsigned int num_cpu_threads = omp_get_num_threads();
    int d = cpu_thread_id % data->deviceCount;
    cudaSetDevice(d);
    Some code, containing kernels
}
```

Si seleziona il device tramite: **__cudaSetDevice__**

Le istruzioni CUDA sono dirette verso il device selezionato

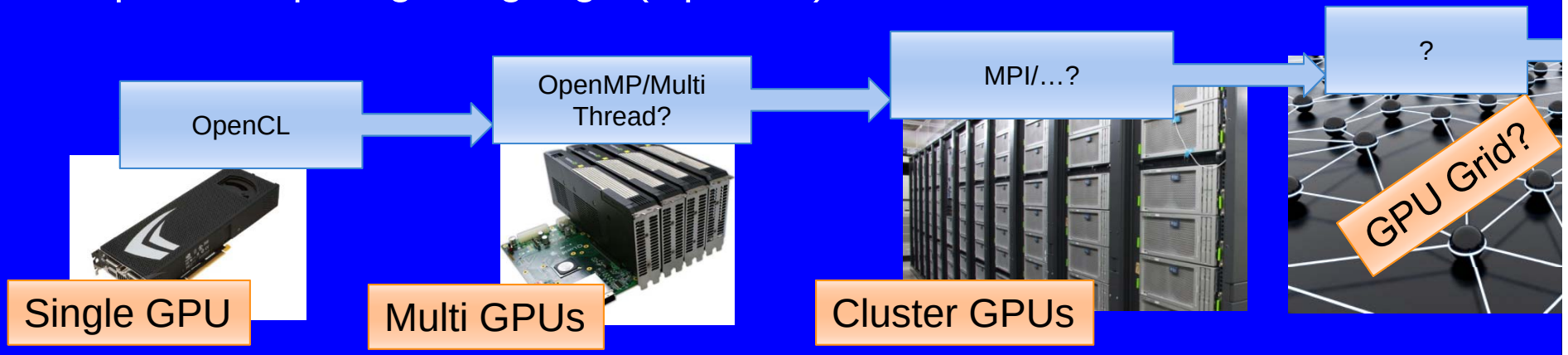
openMP note: overhead nella creazione dei thread

MaCGO:
Split radix 4 FFT on devices



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 simple example: kernels

Un kernel è definito tramite **__global__**

Ad ogni thread che esegue un kernel è assegnato un id che è accessibile tramite threadIdx

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

In questo esempio ogni thread elabora dati referenziati tramite il proprio threadIdx, che il sistema assegna in maniera sequenziale partendo da 0

Dove N è la dimensione del nostro vettore

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

The numero di Thread CUDA è definito con la sintassi: **<<< … >>>**

Ref: CUDA Programming guide

Cuda vs OpenCL code

C for CUDA Kernel Code:

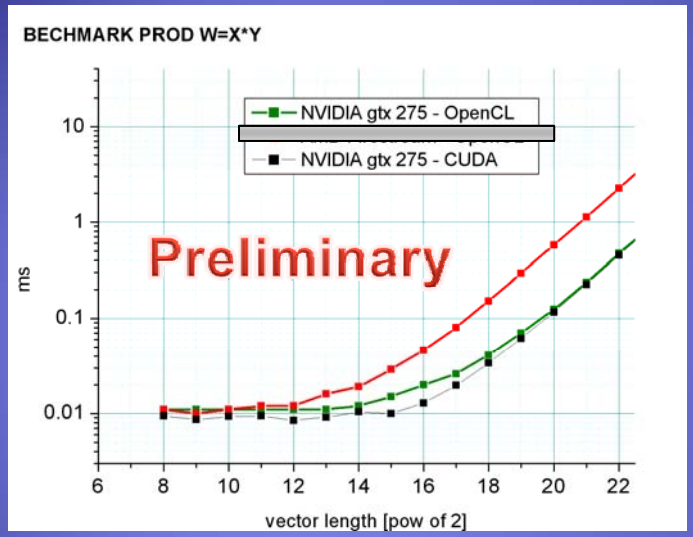
```
__global__ void
vectorAdd(const float * a, const float * b, float * c)
{
    // Vector element index
    int nIndex = blockIdx.x * blockDim.x + threadIdx.x;

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

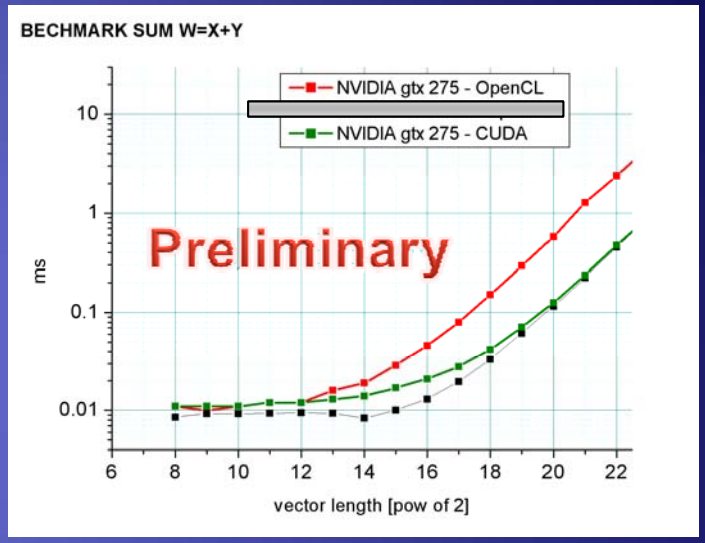
OpenCL Kernel Code

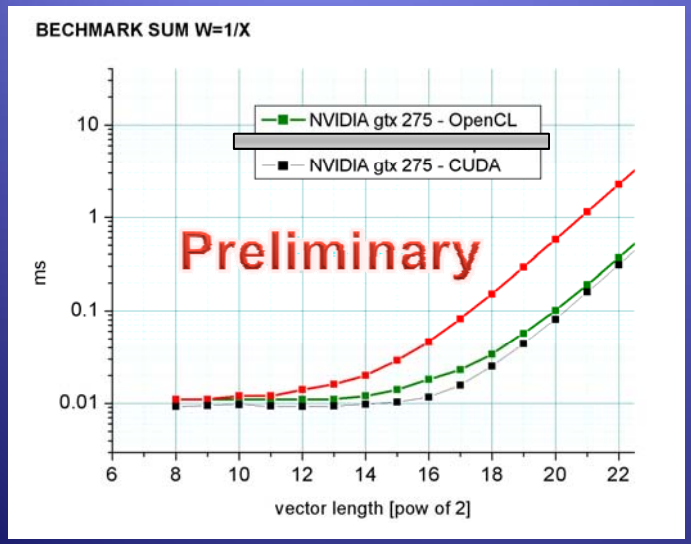
```
__kernel void
vectorAdd(__global const float * a,
          __global const float * b,
          __global       float * c)
{
    // Vector element index
    int nIndex = get_global_id(0);

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

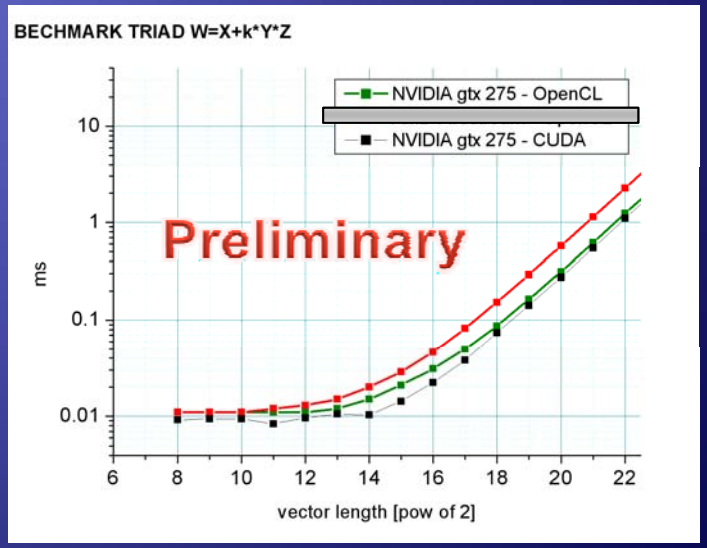
Ref: CUDA OpenCL JumpStart guide

CUDA vs OpenCL (GTX 275/FireStrem 9270)



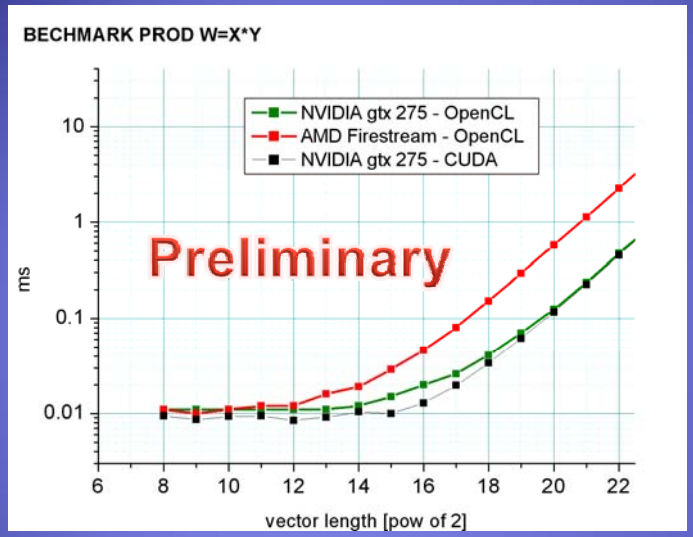


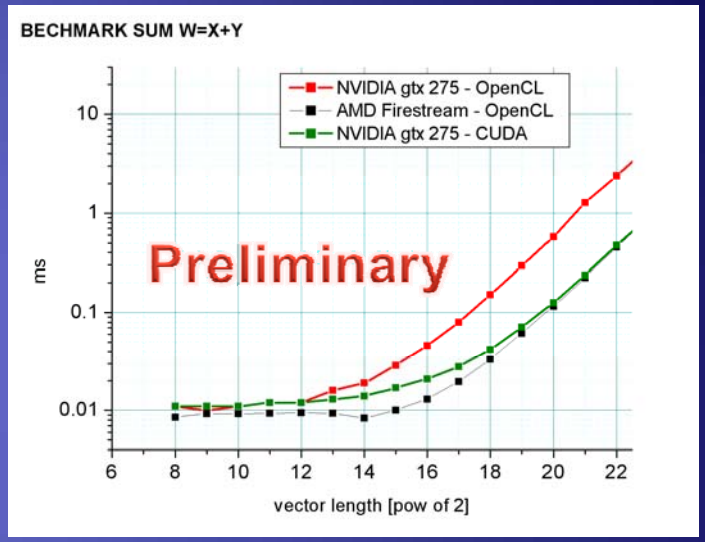


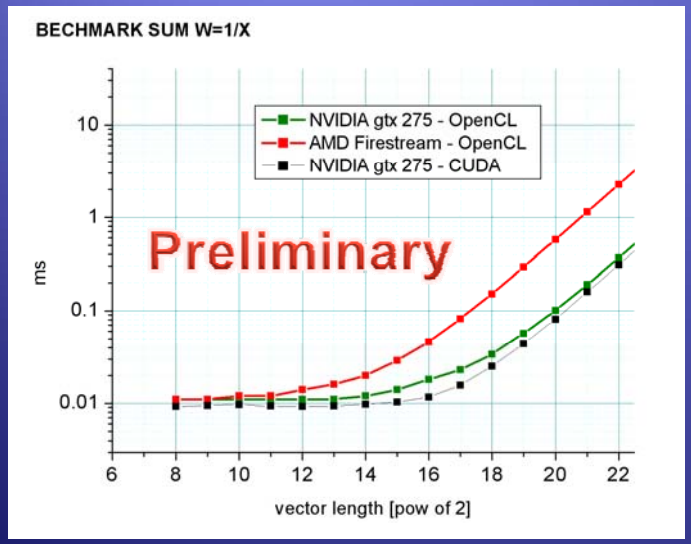


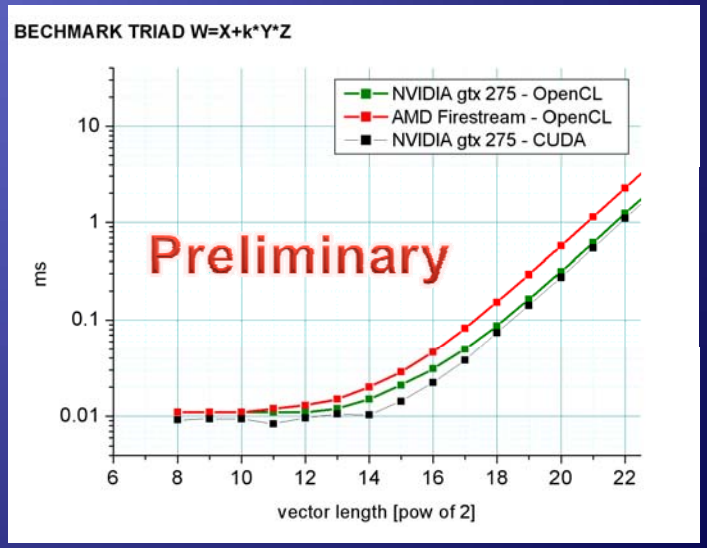
Benchmark made by Dr. Flavio Vella – INFN Perugia@MaGGO

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









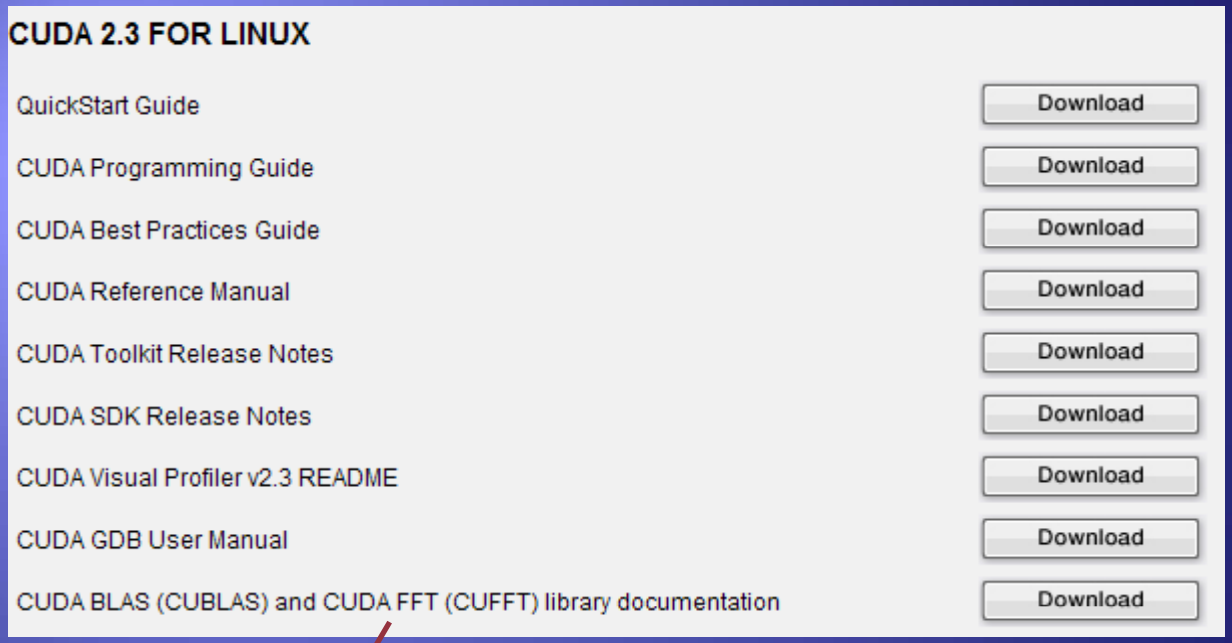
Benchmark made by Dr. Flavio Vella – INFN Perugia@MaGGO

Conclusions:

- Some important items of MaCGO experiment have been reported.
- This work is in the perspective of coding in the so called “manycore era”. That is important for ET telescope purposes and not only.
- We have reported some preliminary results about performances gain for CB detection pipeline (@ nvidia GTX275):
  - Gain factor **X100** about templates generation respect to identical CPU implementation. (This factor is expected to be roughly constant also for higher PN approximation or others generators).
  - Gain factor **X60** about FFT, using cuFFT library, but in the close future, new cuFFT versions promise to have **X120-180** or more.
  - Number of **X30-35** templates processed per seconds with vector size= 2^20
- An OpenCL and CUDA preliminary performances comparison has been reported
- ATI Firestream and NVIDIA gtx 275 performances comparison has been reported

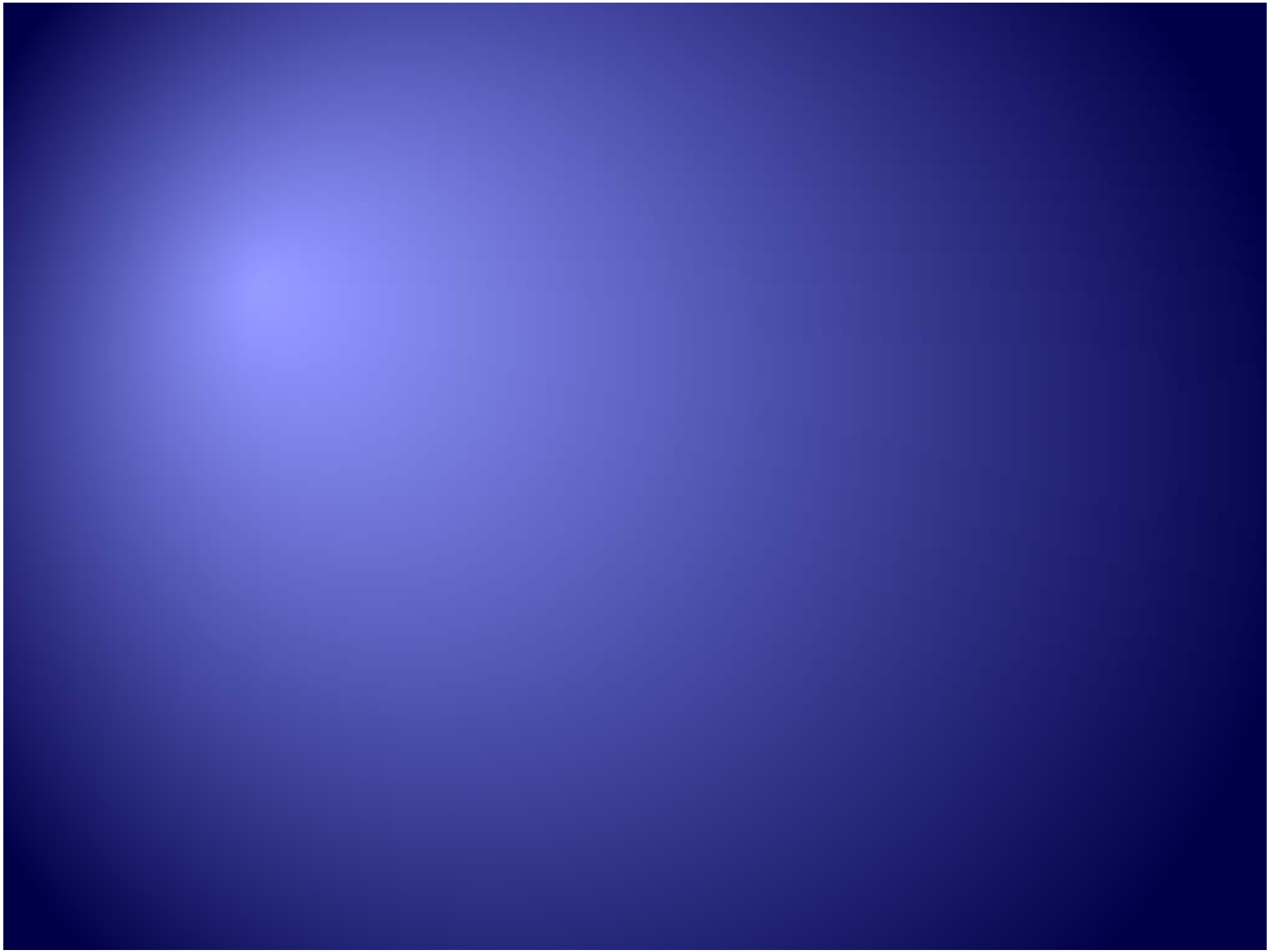
Some links:

- CUDA DOCUMENTATION: http://www.nvidia.com/object/cuda_develop.html



CUBLAS & CUFFT

- MaCGO experiment http://macgo.pg.infn.it. You can get doc and news on the GPUs world



Extra 1 forecast

We could try to make a projection of the available computing power by 2020, in the context of CB like algorithms, making some assumptions:

1. We can start considering that the actual firsts attempts of manycore architecture provides a factor **x150** in single precision respect CPU implementation.
2. We can consider a Moore's law factor at that time of **x100**
3. From the experiences coming from massive parallel architecture, usually performances are reduced significantly by communication overhead, thus we take **x0.4** (it could be even worse)
4. We obtain :
   - a gain of a factor **6000** respect the actual CPU implementation.
   - Equivalent to **5 TFLOPs or higher** on a single manycore processor by 2020.