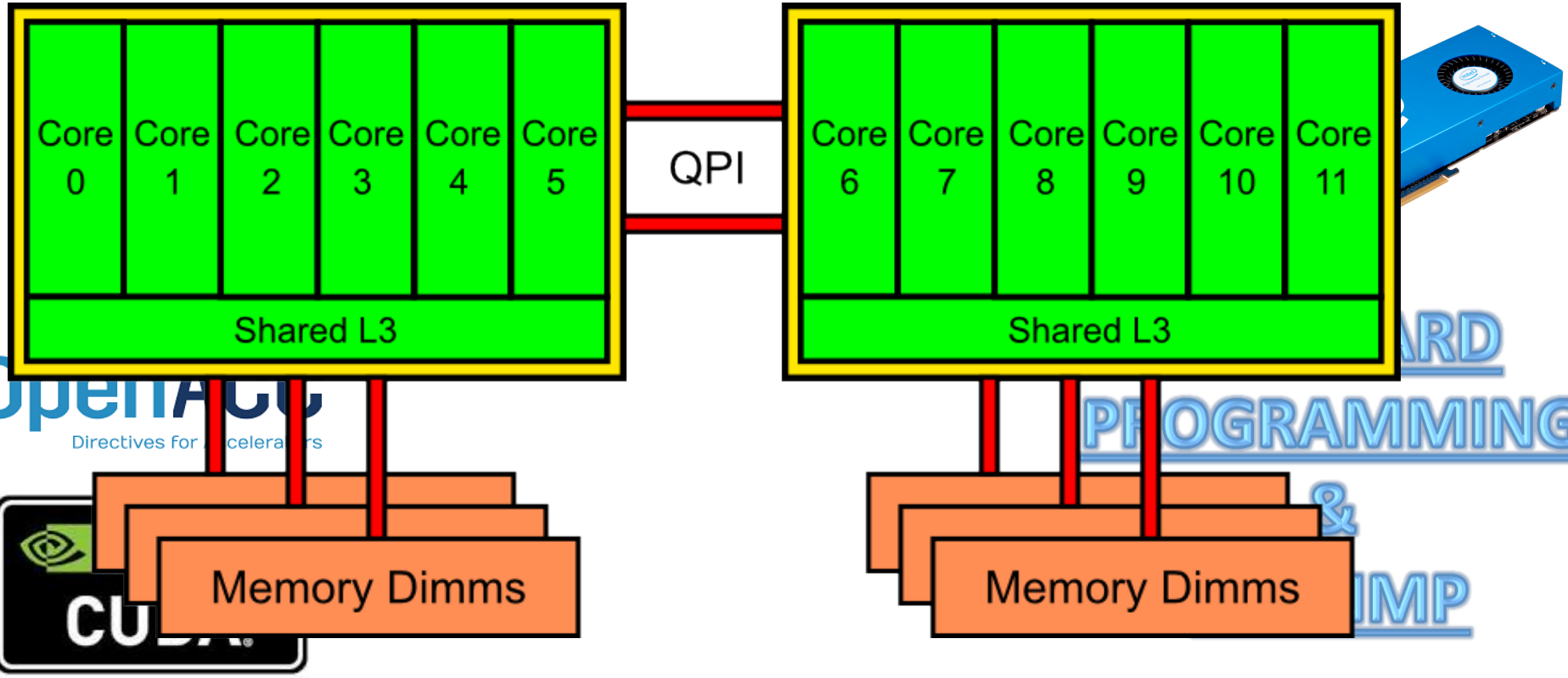
Overview on Modern Accelerators and Programming Paradigms

Ivan Girotto – igirotto@ictp.it

Information & Communication Technology Section (ICTS)

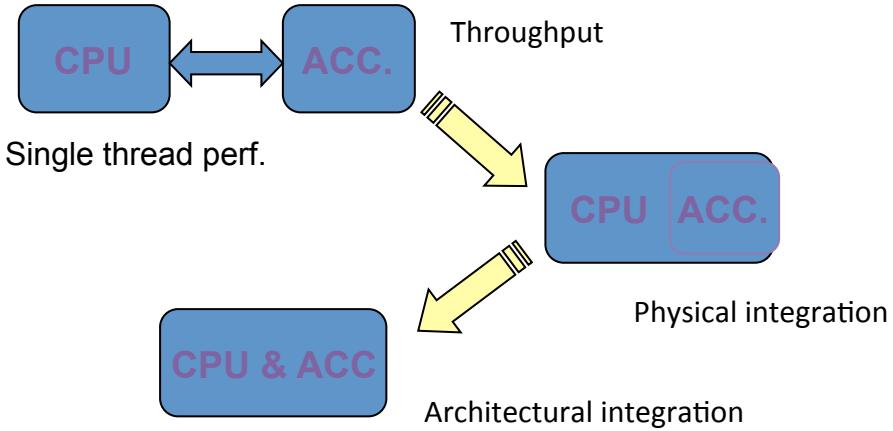
International Centre for Theoretical Physics (ICTP)

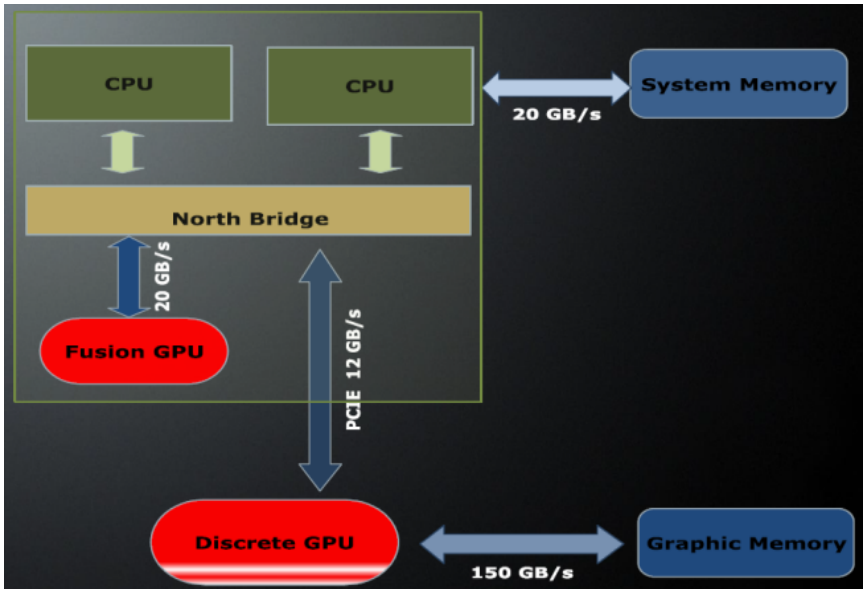
Multiple Socket CPUs + Acceleretors



Accelerated co-Processors

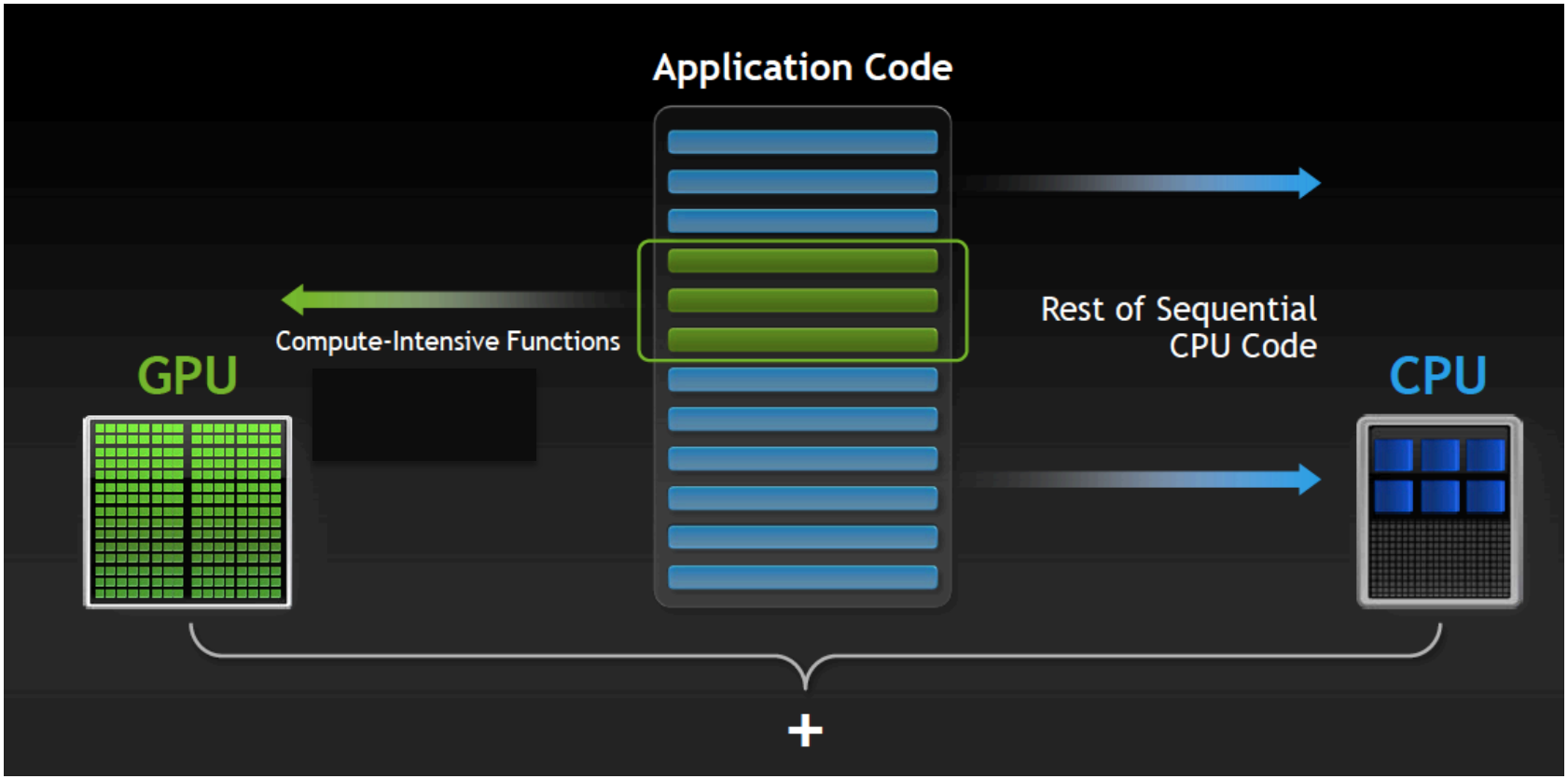
- A set of simplified execution units that can perform few operations (with respect to standard CPU) with very high efficiency. When combined with full featured CPU can accelerate the “nominal” speed of a system.

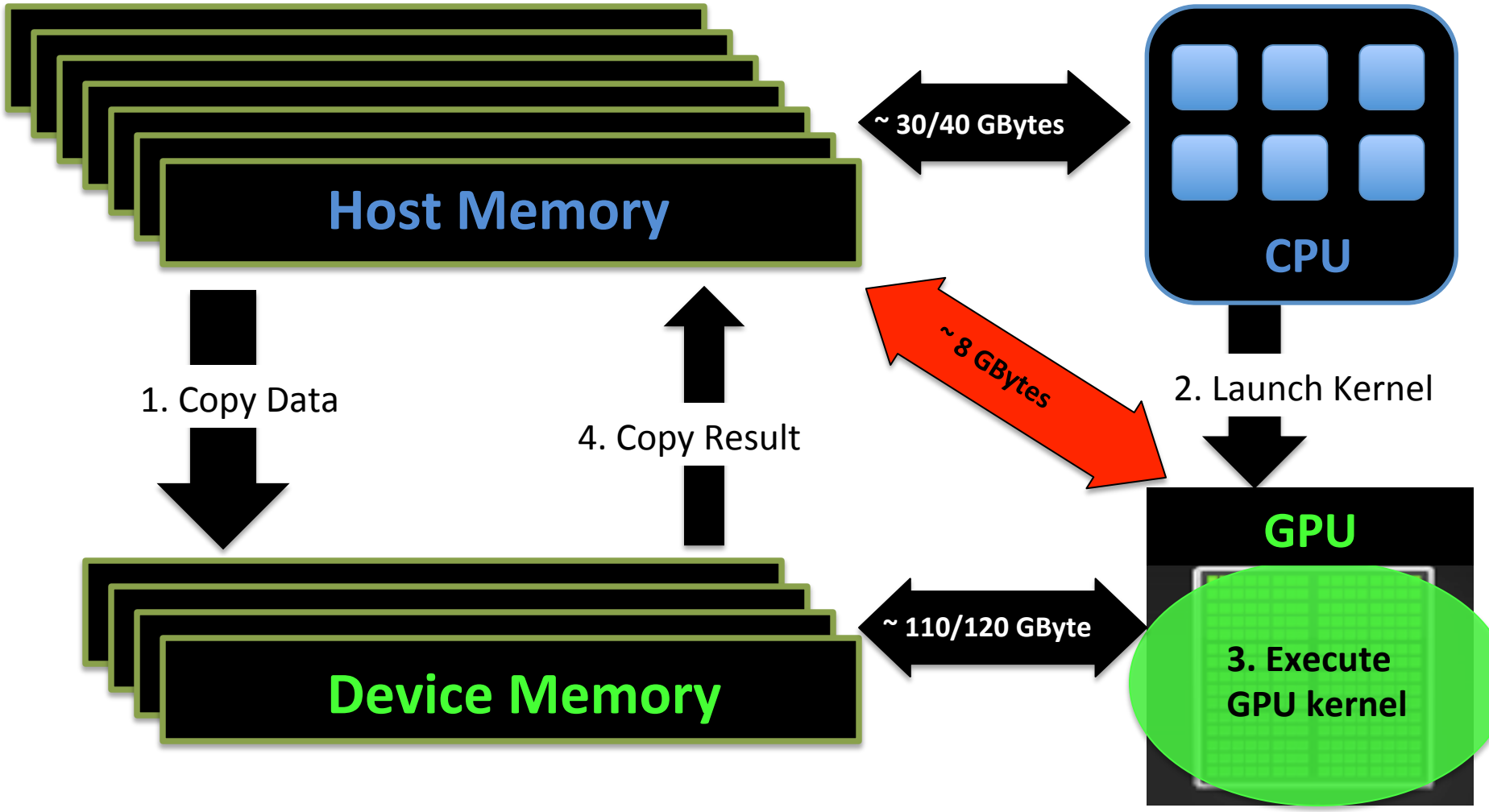




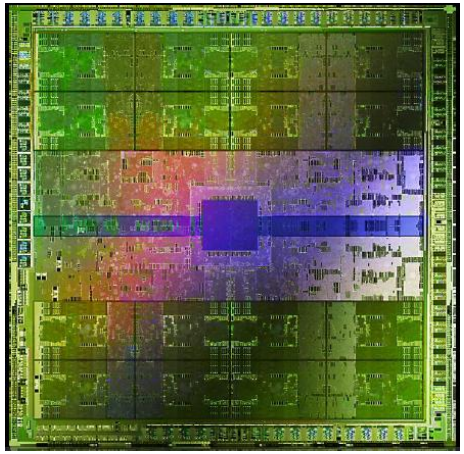
- **Main approaches to accelerators:**
  - **Task Parallelism (MIMD) → MIC**
  - **Data Parallelism (SIMD) → GPU**

The General Concept of Accelerated Computing




Host Memory
~ 30/40 GBytes
CPU
~ 8 GBytes
1. Copy Data
4. Copy Result
2. Launch Kernel
GPU
Device Memory
~ 110/120 GByte
3. Execute
GPU kernel

NVIDIA GPU

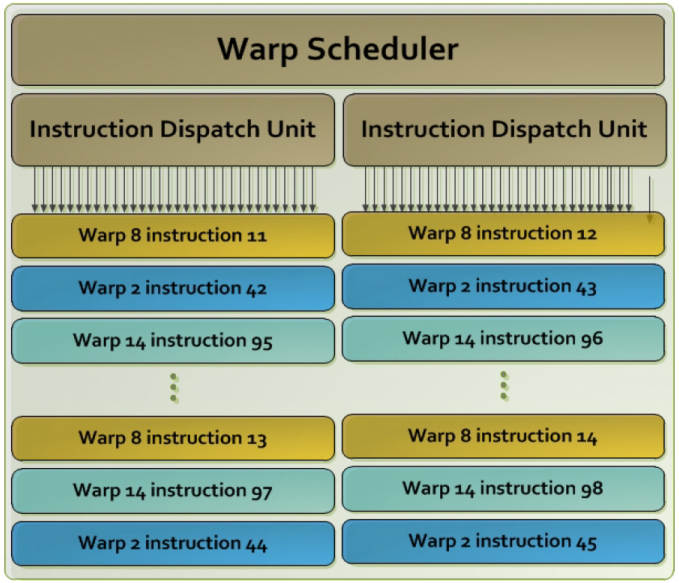


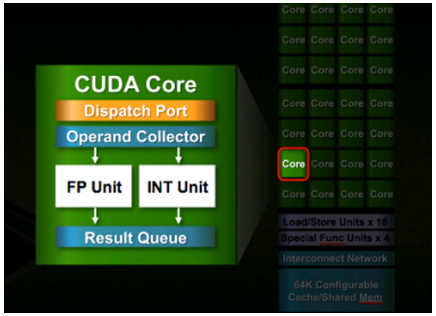
Why Does GPU Accelerate Computing?

- Highly scalable design
- Higher aggregate memory bandwidth
- Huge number of low frequency cores
- Higher aggregate computational power
- Massively parallel processors for data processing


PCI Express 3.0 Host Interface
GigaThread Engine
SMX
SMX
SMX
SMX
SMX
SMX
SMX
SMX
Memory Controller
Memory Controller
Memory Controller
Memory Controller
Memory Controller
Memory Controller
L2 Cache
SMX
SMX
SMX
SMX
SMX
SMX
SMX

SMX Processor & Warp Scheduler & Core





Why Does GPU Not Accelerate Computing?

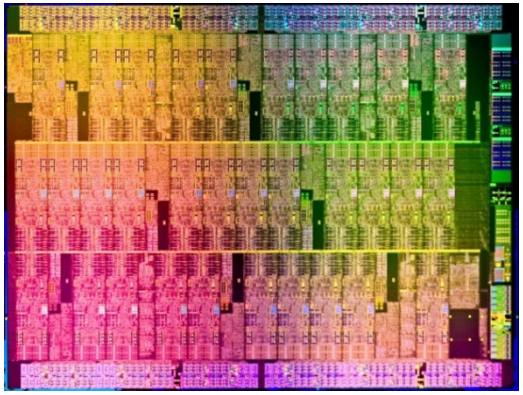
- PCI Bus bottleneck
- Synchronization weakness
- Extremely slow serialized execution
- High complexity
  - SPMD(T) + SIMD & Memory Model
- People forget about the Amdahl's law
  - accelerating only the 50% of the original code, the expected speedup can get at most a value of 2!!

What is CUDA?

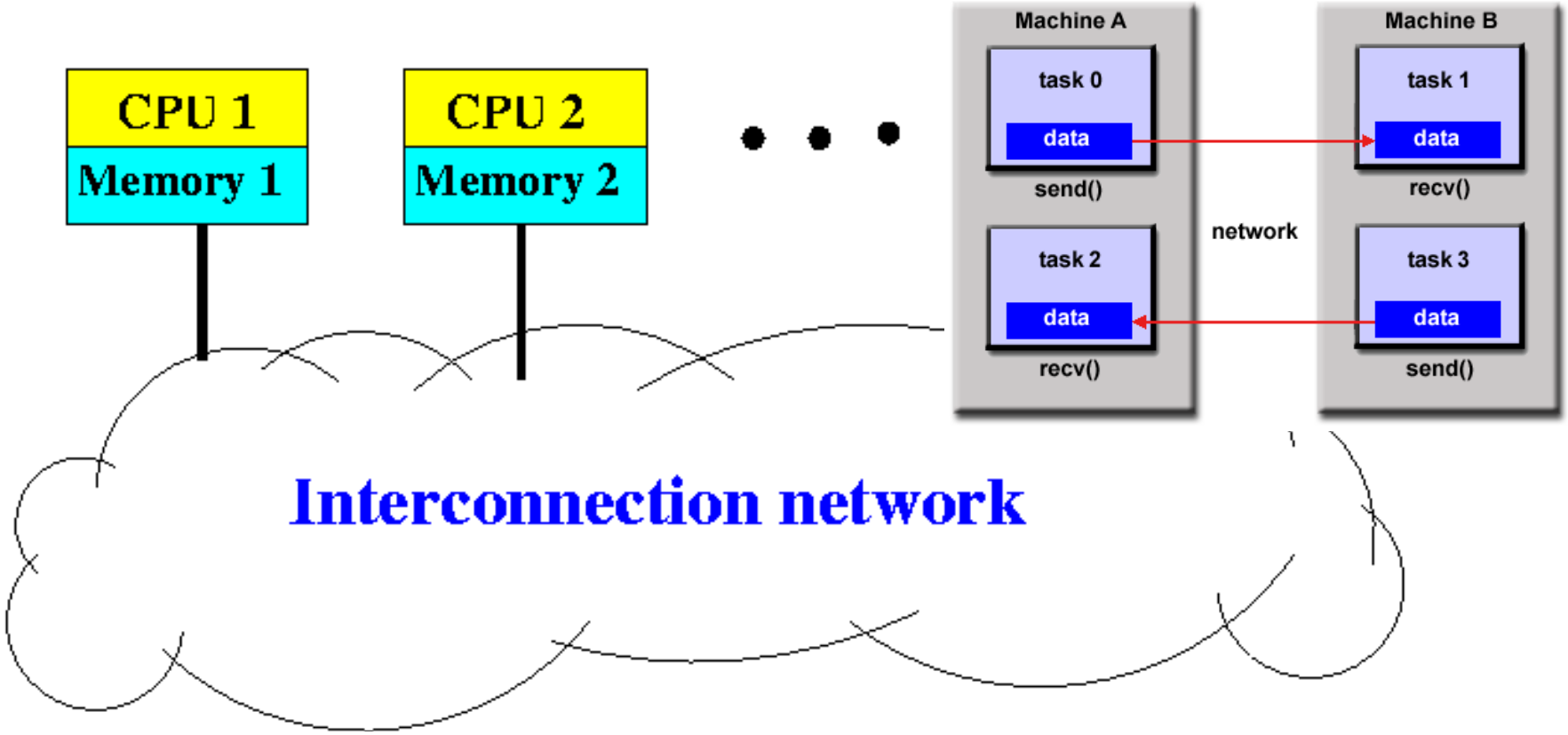
- **NVIDIA** compute architecture
- Quickly maturing software development capability provided free of charge by NVIDIA
- C and C++ programming language extension that simplifies creation of efficient applications for CUDA-enabled GPGPUs
- Available for Linux, Windows and Mac OS X


PCI Express 3.0 Host Interface
GigaThread Engine
SMX
SMX
SMX
SMX
SMX
SMX
SMX
SMX
Memory Controller
Memory Controller
Memory Controller
Memory Controller
Memory Controller
Memory Controller
L2 Cache
SMX
SMX
SMX
SMX
SMX
SMX
SMX

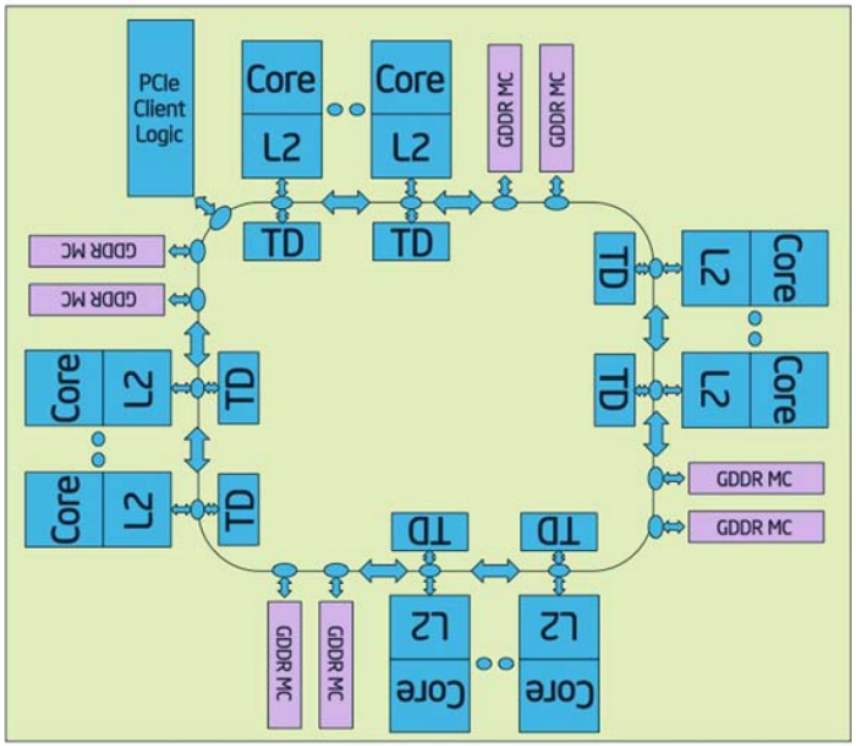
INTEL MIC

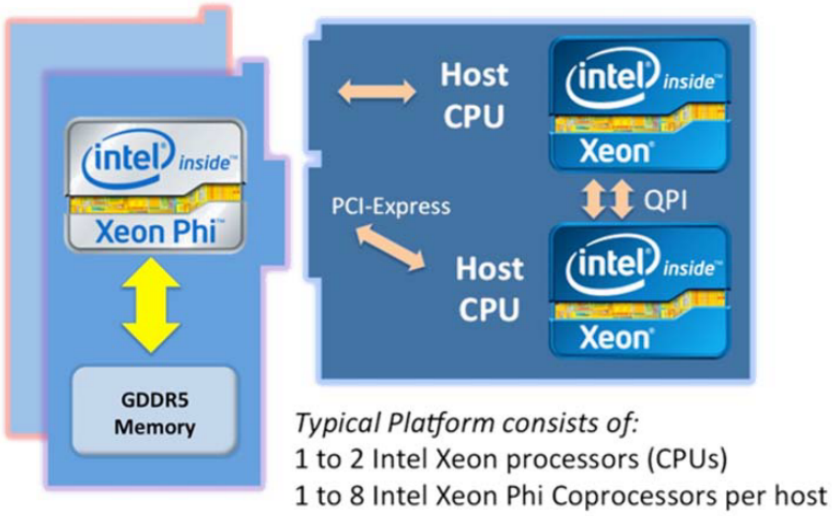


TASK Parallelism (MIMD)



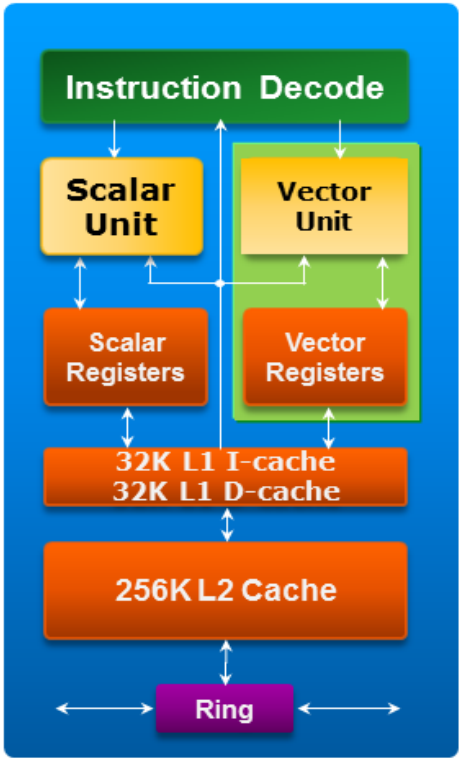
Xeon PHI Architecture





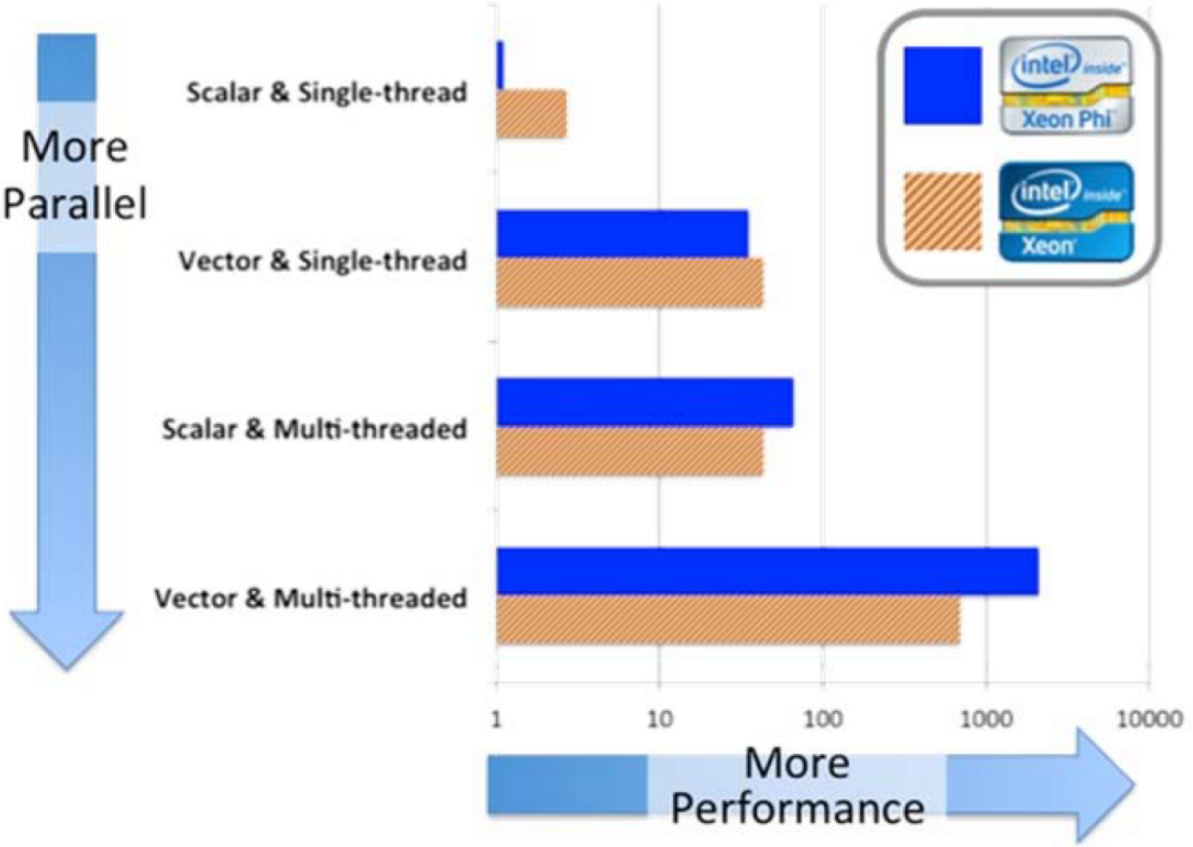
Typical Platform consists of:
1 to 2 Intel Xeon processors (CPUs)
1 to 8 Intel Xeon Phi Coprocessors per host

Core Architecture



- Up to 32 in-order cores
- 4 hardware threads per core
- Two pipelines
  - Pentium® processor family-based scalar units
  - Fully-coherent L1 and L2 caches
  - 64-bit addressing
- All new vector unit
  - 512-bit SIMD Instructions – not Intel® SSE, MMX™, or Intel® AVX
  - 32 512-bit wide vector registers
    - Hold 16 singles or 8 doubles per register
  - Pipelined one-per-clock throughput
    - 4 clock latency, hidden by round-robin scheduling of threads
  - Dual issue with scalar instructions

The Increasing Parallelism



Execution Models: Offload Execution

- Host system offloads part or all of the computation from one or multiple processes or threads running on host
- The application starts execution on the host
- As the computation proceeds it can decide to send data to the coprocessor and let that work on it and the host and the coprocessor may or may not work in parallel.

OpenMP 4.0 TR being proposed and implemented in Intel® Composer XE provides directives to perform offload computations. Composer XE also provides some custom directives to perform offload operations.

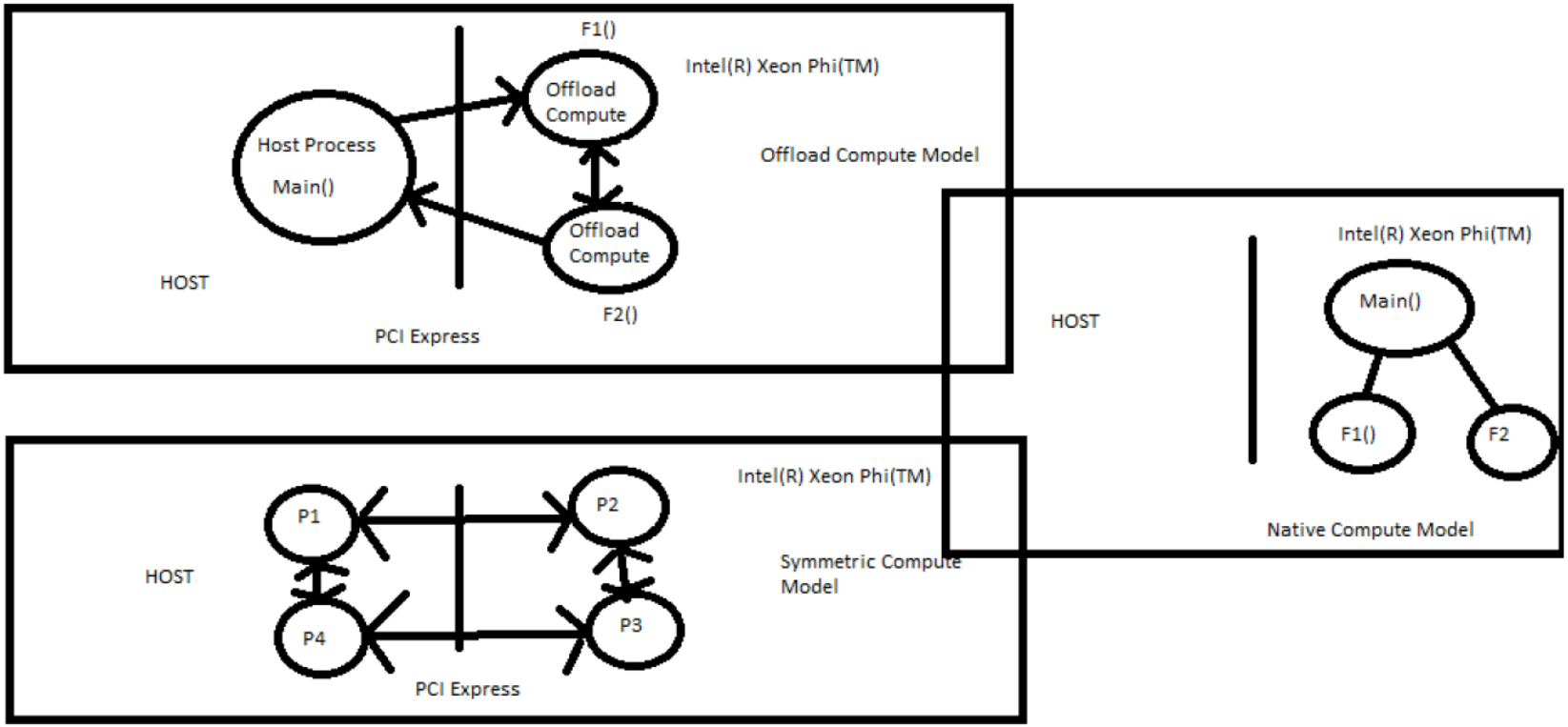
Execution Models: Native Execution

- An Xeon Phi hosts a Linux micro OS in it and can appear as another machine connected to the host like another node in a cluster.
- This execution environment allows the users to view the coprocessor as another compute node.
- In order to run natively, an application has to be cross compiled for Xeon Phi operating environment. Intel® Composer XE provides simple switch to generate cross compiled code.

Execution Models: Symmetric Execution

- The application processes run on both the host and the Phi coprocessor and communicate through some sort of message passing interface like MPI.
- This execution environment treats Xeon Phi card as another node in a cluster in a heterogeneous cluster environment.

Execution Models: Summary



1. Offloading a function call

```
#pragma offload target (mic)
    foo();
```

```
foo() { .... } // Compiled for mic
```

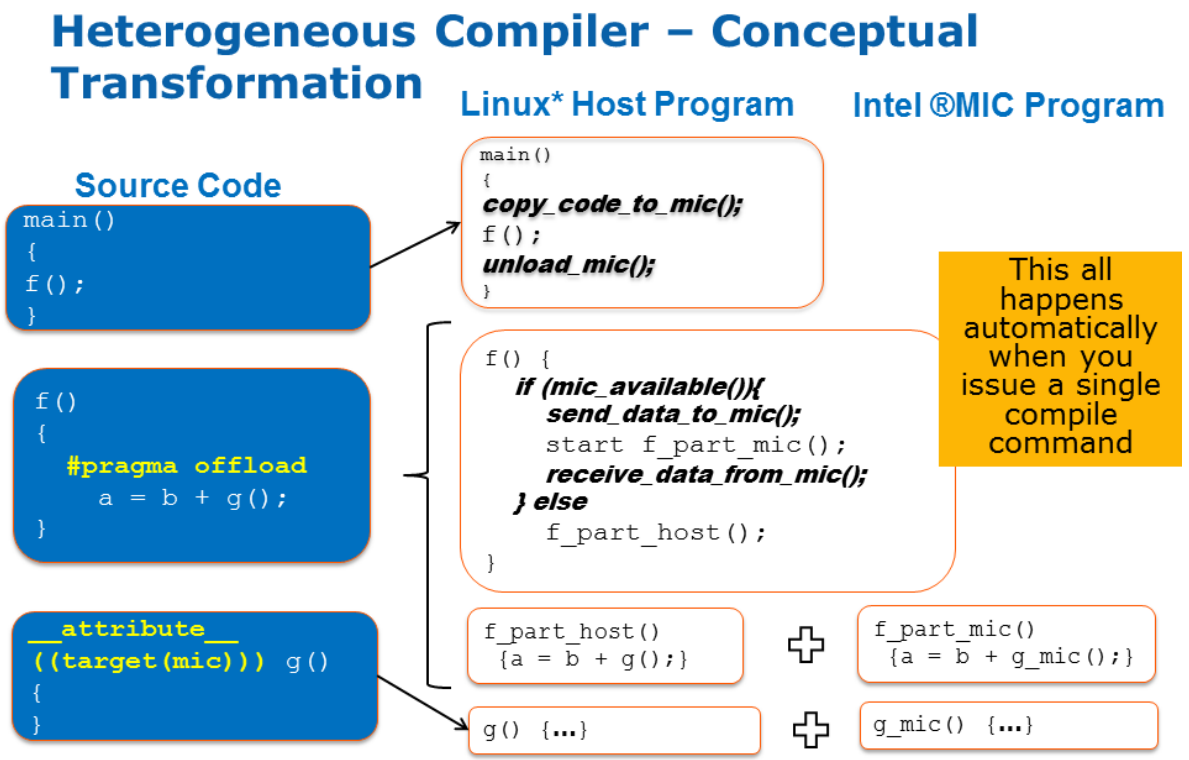
2. Calculating Pi with automatic offload

```
#pragma offload target (mic)
#pragma omp parallel for reduction(+:pi)
 for (i=0; i<count; i++)
 {
       float t = (float)((i+0.5)/count);
       pi += 4.0/(1.0+t*t);
 }
 pi /= count
```

3. Using MKL with offload

```
void your_hook()
{
     float *A, *B, *C; /* Matrices */
     #pragma offload target(mic)
      in(transa, transb, N, alpha, beta) \
      in(A:length(matrix_elements)) \
      in(B:length(matrix_elements)) \
      in(C:length(matrix_elements)) \
     out(C:length(matrix_elements)alloc_if(0))
      sgemm(&transa, &transb, &N, &N,
                &N, &alpha, A, &N, B, &N, &beta, C,
     &N);
}
```

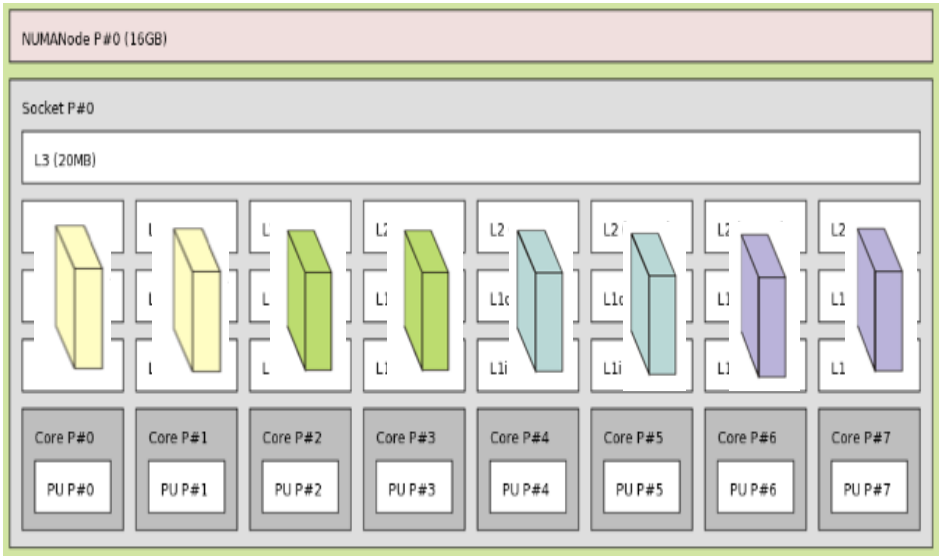
Heterogeneous Compiler



OpenCL

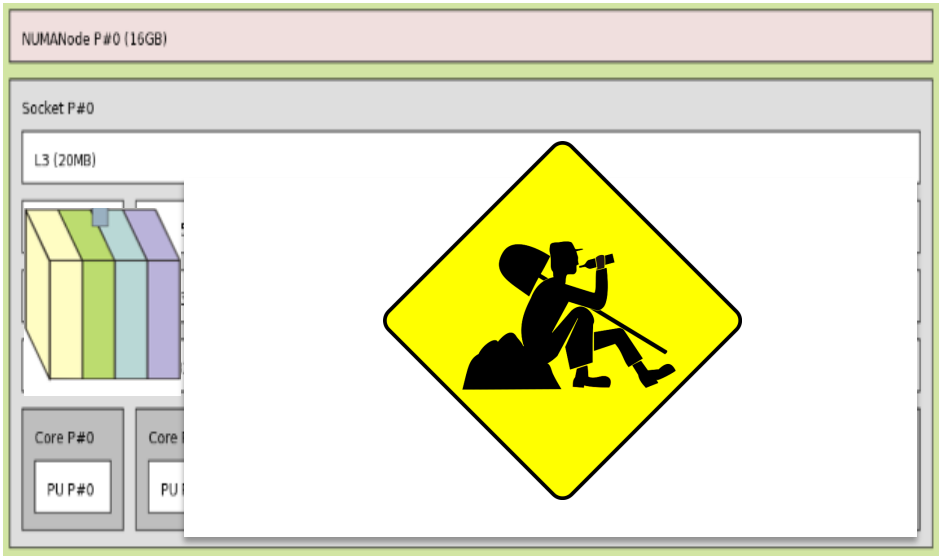
- *O*pen *C*ompute *L*anguage
- Open, royalty-free standard for cross-platform,
- For heterogeneous parallel-computing systems
- Cross-platform. Implementations for
  - ATI GPUs
  - NVIDIA GPUs
  - x86 CPUs

CPU & GPU



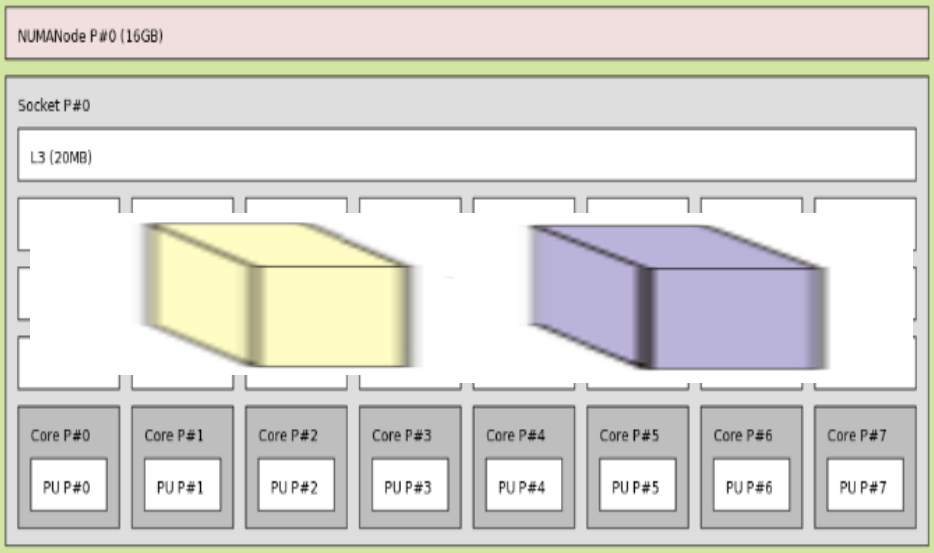
The Intel Xeon E5-2665
Sandy Bridge-EP 2.4GHz

CPU & GPU



The Intel Xeon E5-2665
Sandy Bridge-EP 2.4GHz

CPU & GPU



The Intel Xeon E5-2665
Sandy Bridge-EP 2.4GHz

Higher aggregate computational power

- Do we really ... need it? ... have it available?
- Can we really exploit it?
- Remember the key-factors for performance
  - #operations per clock cycle x frequency x #cores
  - the DP power is drastically reduced if the compute capability is only partially exploited
- How much is my GPU better than my CPU?
- Can data move from CPU2GPU and from GPU2CPU be reduced?
- For general purpose and scalable applications, both CPU and GPU must usually be exploited

Conclusions

- A low number of applications and scientific codes are enabled for accelerators: some for GPU, few for Intel Xeon Phi
- For general DP intensive applications the average speedup is of a factor between 2x and 3x using two accelerators on top of the CPU platform
- Fast GPU computing requires the technological background for exploiting the compute power available, manage the balance between CPU and GPU along with the effort for the system management

25/05/2015 - 05/06/2015

WORKSHOP ON ACCELERATED HIGH-PERFORMANCE COMPUTING IN COMPUTATIONAL SCIENCES (SMR 2760)