QE and many-core architectures

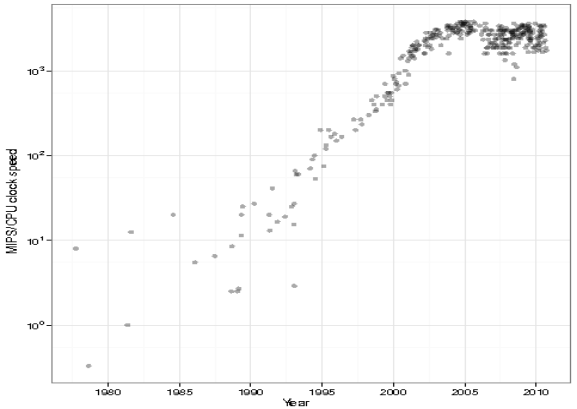
Fabio AFFINITO

SCAI - Cineca

«*What I cannot compute, I do not understand.*» (adapted from Richard P. Feynman)

HPC trends

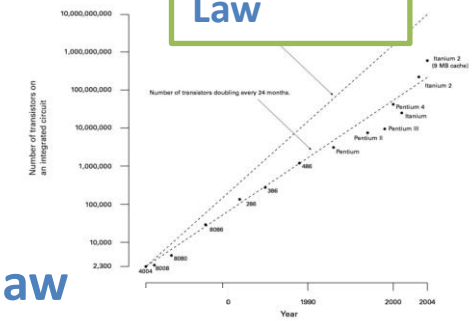
Dennard scaling law (downscaling)



The core frequency and performance do not grow following the Moore's law any longer

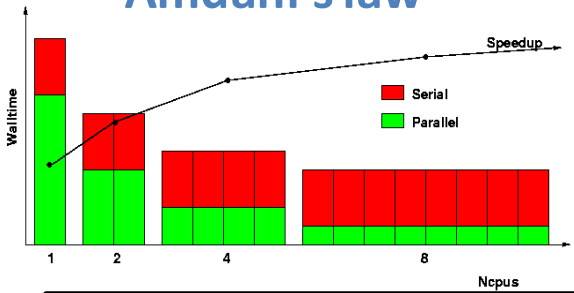
Increase the number of cores to maintain the architectures evolution on the Moore's law

Moore's Law



Number of transistors per chip double every 24 month

Amdahl's law



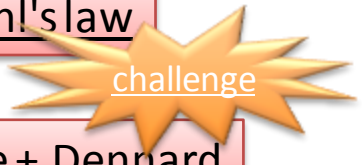
maximum speedup tends to $1/(1-P)$
P= parallel fraction

The upper limit for the scalability of parallel applications is determined by the fraction of the overall execution time spent in non-parallel operations.

QUANTUM ESPRESSO FOUNDATION

HPC trends

Peak Performance		Exaflops 10^18	Moore law
FPU Performance		Gigaflops 10^9	Dennard law
Number of FPUs		10^9	Moore + Dennard
App. Parallelism		serial fraction 1/10^9	Amdahl's law
System TCO		Power >10^6 Watt	Moore + Dennard



The energy problem

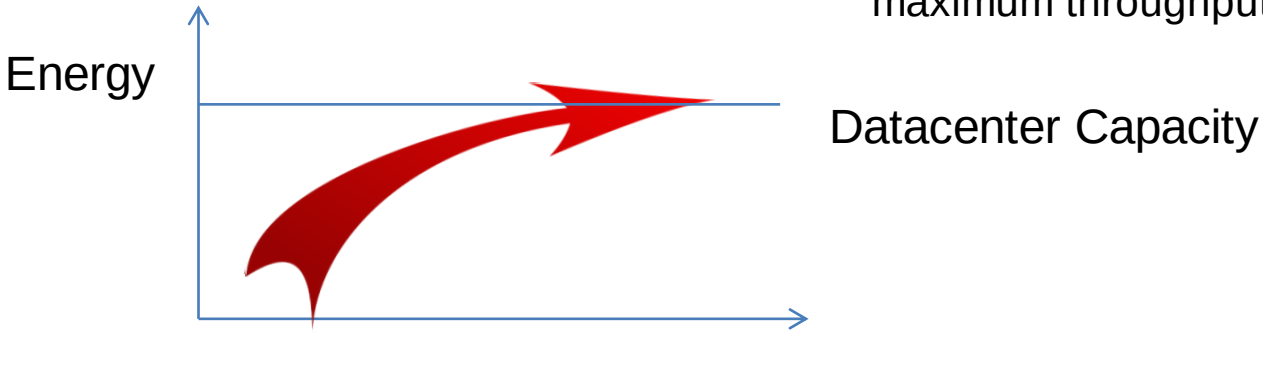
"traditional" RISC and CISC chips are designed for maximum performance for all possible workloads

A lot of silicon to maximize single thread performace

A change of paradigm

New chips designed for maximum performance in a small set of workloads

Simple functional units, poor single thread performance, but maximum throughput

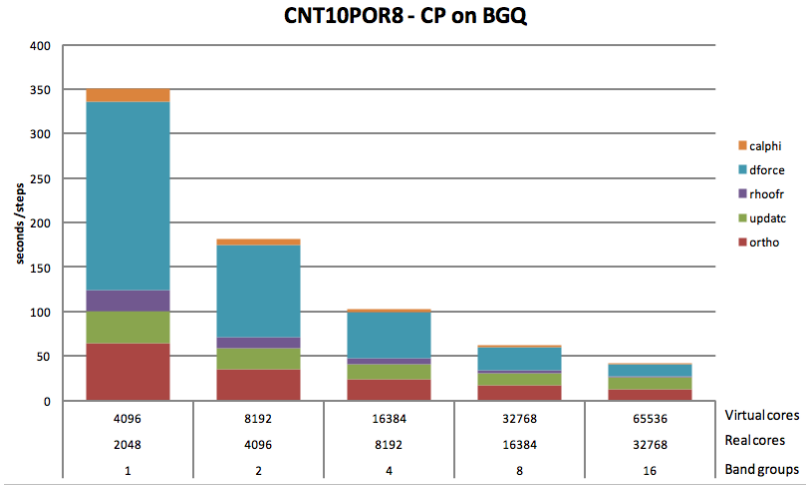


Key factors

1. Increase the parallelism
   - MPI communicators
   - OpenMP
2. Reduce communications
   - increase data locality
   - introduce communication-avoiding algorithm (trade-off with data replication)
3. Use asynchronous mechanisms (OpenMP tasks or NB MPI comms)
   - hide the latency
   - increase the pipelining

IBM BlueGene/Q

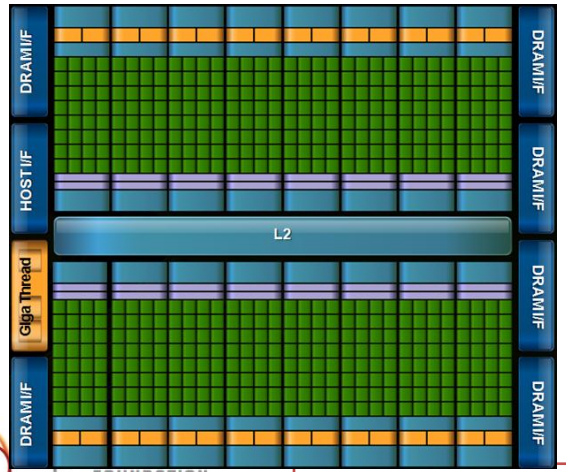
- Using all the MPI levels of parallelization, permitted to scale up nicely on BGQ.

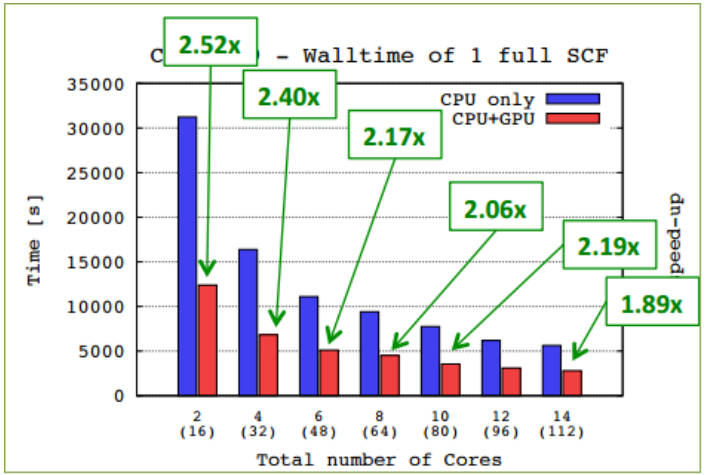


10.240 nodes with 16 cores each
16 GB per node

GP-GPUs

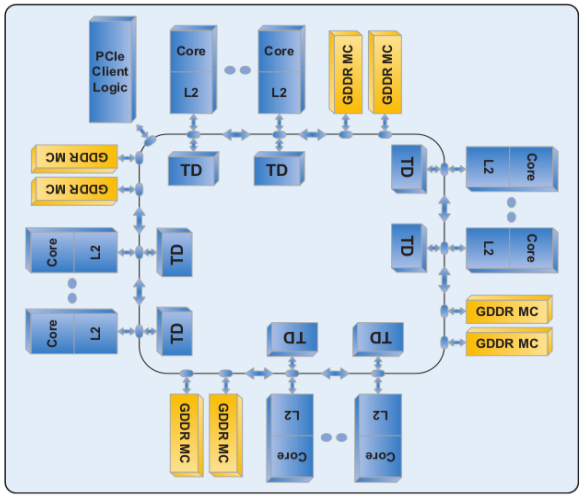
- GP-GPUs are devices, that are connected to a host CPU, with a huge number of (special) cores. The parallelism on GPUs can be exploited only with ad-hoc programming paradigms (e.g. CUDA, OpenCL)
- A *parallel* project to enable QE to exploit GPUs was started several years ago





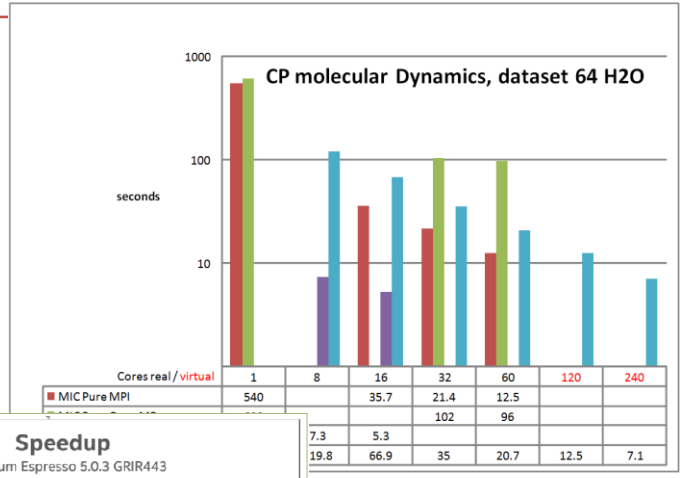
Intel MIC

- Intel MIC technology is based on a device coupled to a host cpu, similarly to a GP-GPU
- Differently from a GP-GPU, the Intel MIC cores are derived from a *x86* architecture. The programming paradigms usable for MICs cards are MPI and OpenMP
- The first generation of MICs, codenamed Knights Corner (KNC) consists in 60 cores, with 4 hardware threads each and with 512-bit SIMD vectorial registers. The internal network is a bidirectional ring

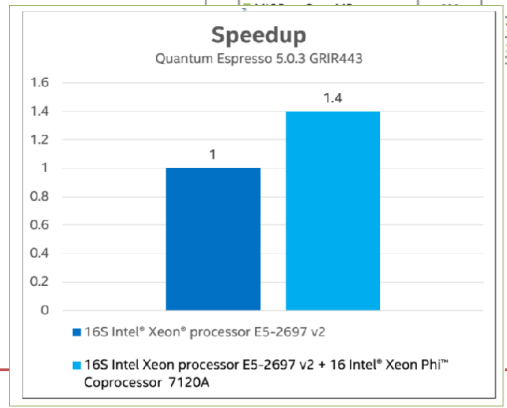


Intel KNC and QE

- Two big challenges
  - Latency between CPU and MIC
  - MPI performance on the card is quite poor (better OpenMP)
- Native compilation can be an option, but you are limited by the available memory per task
- A good solution is to offload to the MIC the linear algebra calculation: Xphilib library
  - Some (small) performance improvement



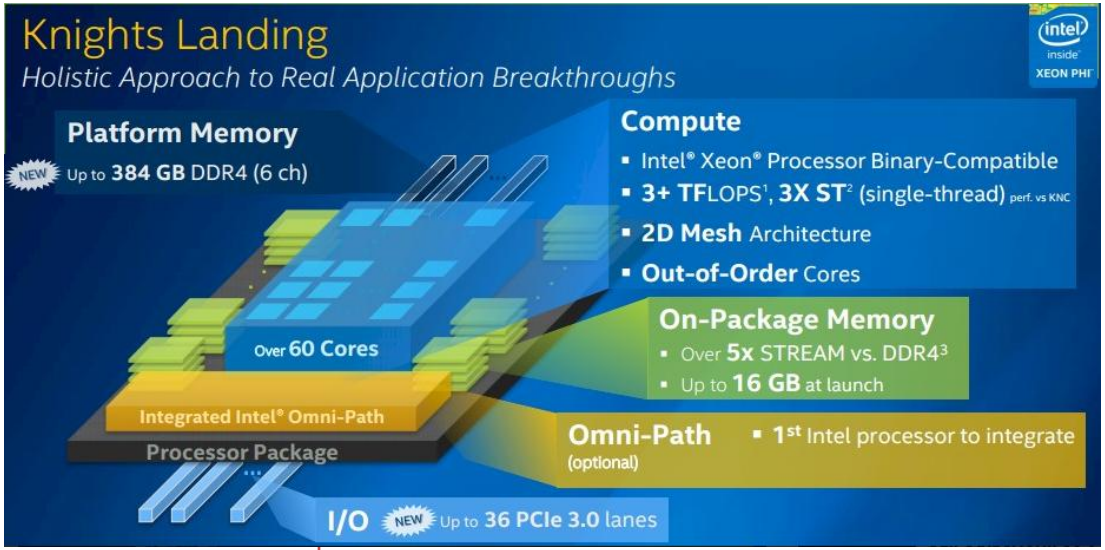
Cores real / virtual	1	8	16	32	60	120	240
MIC Pure MPI	540		35.7	21.4	12.5		
				102	96		
		7.3	5.3				
		19.8	66.9	35	20.7	12.5	7.1



QUANTUM ESPRESSO FOUNDATION

Intel KNL

- The next generation of Intel MIC, codenamed Knights Landing, will be more similar to a BG/Q node than to a GPU



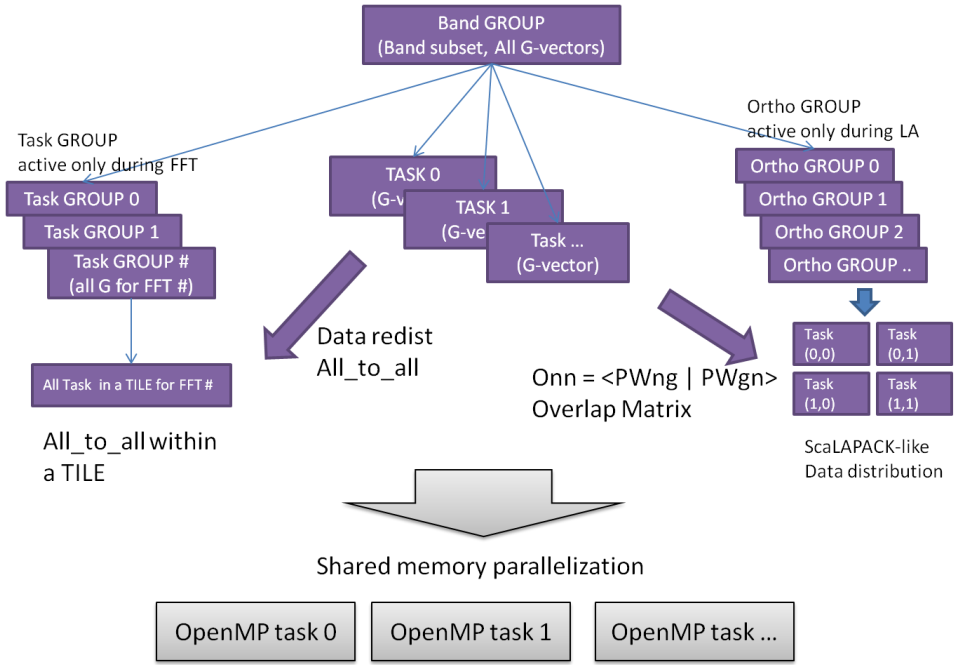
New memory hierarchy !!!

The next steps

- KNL will provide a double kind of memory: on package (fast) and standard DDR4 (larger, but slower).
- Again, the solution is: expose more parallelism and be flexible
- Tiling of data-structures to fit the memory will make algorithms faster (also on standard CPUs)
- FFT could be a good benchmark

Tiling

- A single TILE of processes, inside a given taskgroup, contains all the G-vectors and subset of bands to compute full 3D FFTs.
- If memory permits, thanks to a redistribution with a MPI_Alltoall of the bands, a whole FFT can be performed on a single MPI task locally.
- This data distribution permits, in principle, a hierarchical tiling increasing the data locality and reducing the amount of communications.



Another recipe: tasking

- Within the last implementation, FFTs for independent band indexes are pipelined. This permits to implement OpenMP tasks for each FFT, increasing the workload and the pipelining efficiency.

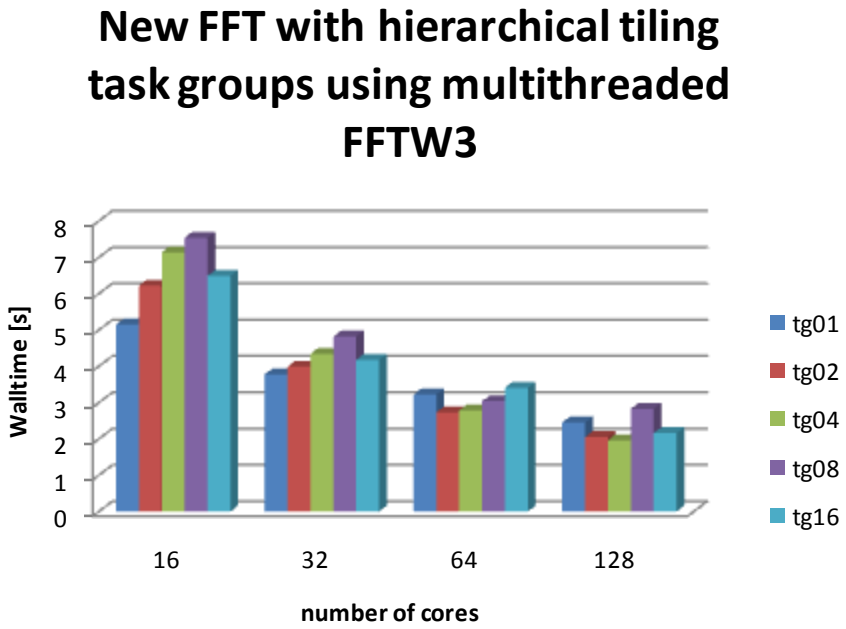
```
do itg = start_tg, end_tg

  distribute_gvectors_sticks

  compute_fft

end do
```

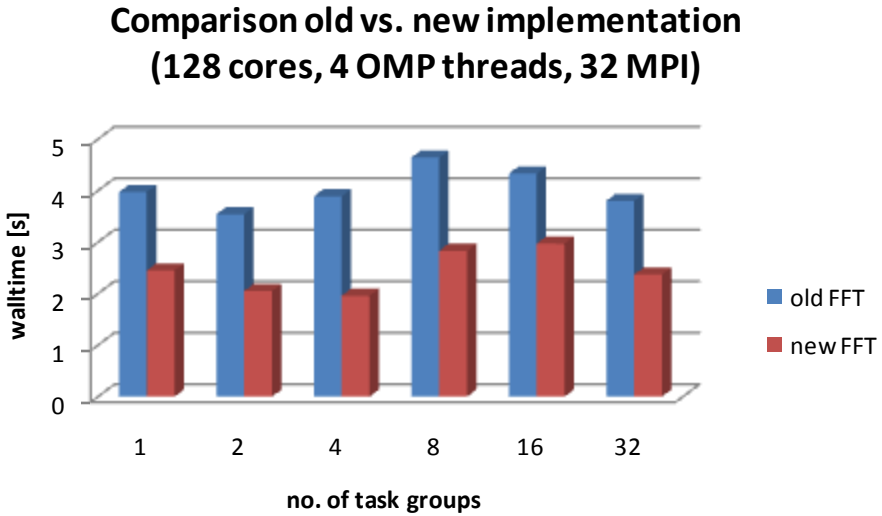
```
do itg = start_tg, end_tg

  distribute_bands_across_pes

  compute z_fft
  scatter
  compute xy_fft

end do
```

task

task

First tests





Now the problem is HOW to choose the right parameters?