Co-funded by the
Erasmus+ Programme
of the European Union

Overview of High Performance Computing

S. F. Schifano

University of Ferrara and INFN-Ferrara

Distributed Computing Architectures and Environmental Science Applications

June 6-10, 2016

Ferrara, Italy

Outline

Co-funded by the Erasmus+ Programme of the European Union

Overview of high-performance architectures

1. Evolution of processor architectures
2. Multi-core architectures
3. Many-core architectures

the one million dollar question

. . . which is the best computing system to use today ?

Background: Let me introduce myself

Co-funded by the Erasmus+ Programme of the European Union

- I'm a computer scientist
- I have been involved in several projects to develop computing systems optimized for computational physics:
  - APEmille and apeNEXT: LQCD-machines
  - AMchip: pattern matching processor, installed at CDF
  - Janus: FPGA-based system for spin-glass simulations
  - QPACE: Cell-based machine, mainly LQCD, 1^{st} TOP-GREEN 500 in Nov.'09 and July'10
  - AuroraScience: multi-core based machine
  - Janus2: 2^{nd} generation of FPGA-based system for spin-glass simulations
  - COKA: Computing on Knights Architectures

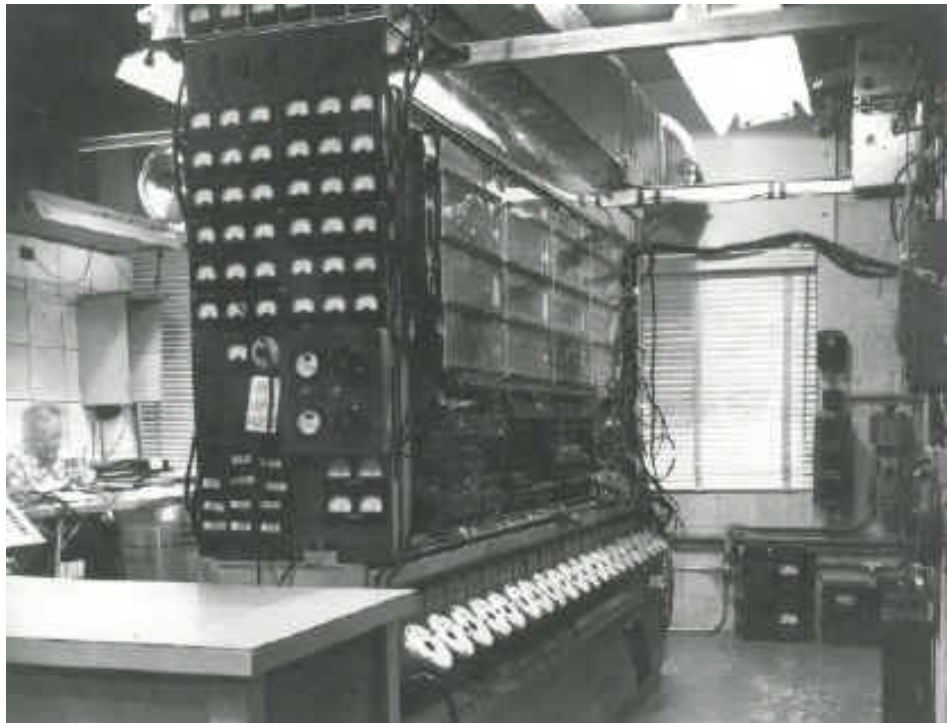
APE, Janus and QPACE

Co-funded by the Erasmus+ Programme of the European Union

APEmille, apeNEXT Janus, Janus2 QPACE

A “modern” CPU architecture: my point of view !

Co-funded by the Erasmus+ Programme of the European Union



. . . YES . . . (the core of) a modern CPU is **still** based on the 1950 Von Neumann model !!

J. Backus

. . . thus programming is basically planning and detailing the enormous traffic of words through the von Neumann bottleneck, and much of that traffic concerns not significant data itself, but where to find it.

CPU performances

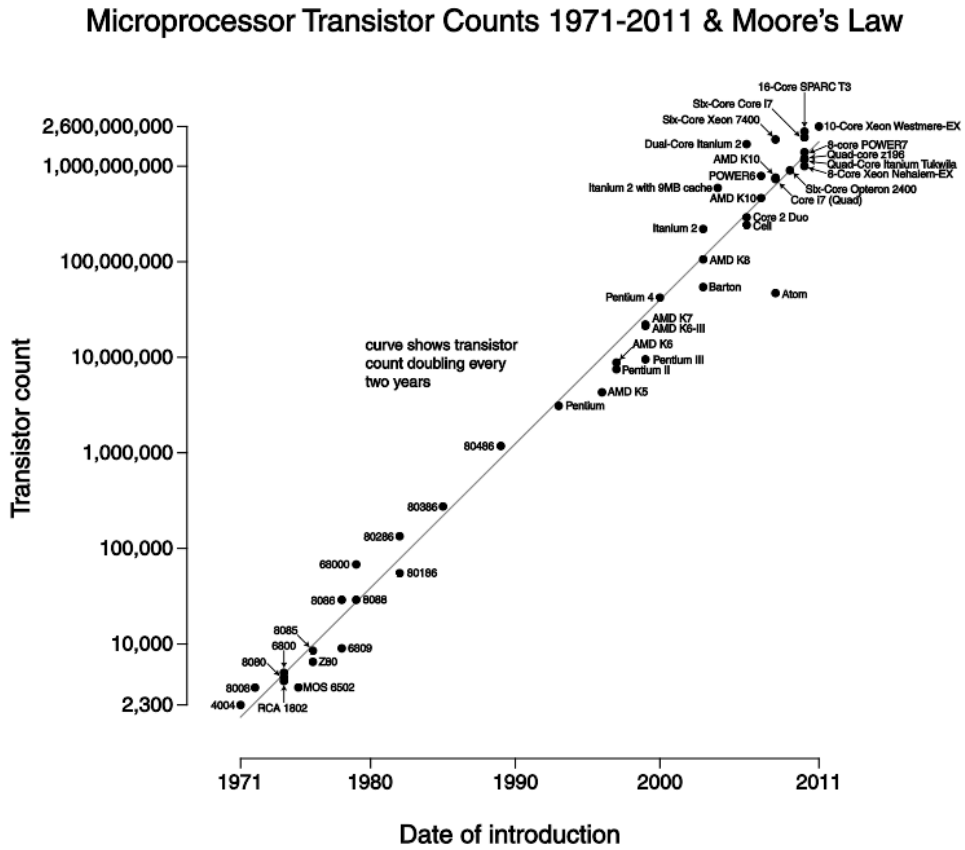
Co-funded by the Erasmus+ Programme of the European Union

At beginning, CPU performances have heavily relied on hardware:

- clock frequency
- hw supports to optimized memory time access:
  - one or more levels of caches,
  - reorder-buffer (ROB)
  - ...
- hw supports to increase ILP:
  - brach-predictors,
  - out-of-order execution,
  - ...

Hardware Evolution: Moore's Law

Co-funded by the Erasmus+ Programme of the European Union



Gordon Moore – co-founder of Intel – *Electronics Magazine 1965*:

Moore's Law

Number of devices/chip doubles every 18 months !

Hardware Evolution: Dennard Scaling

Moore's Law

... 2X number of transistors on a chip every 1.5 years ...

but it's Deannard's law[1] that made them useful:

Dennard's Law

as transistors get smaller their power density stays constant, so that the power use stays in proportion with area: both voltage and current scale (downward) with length.

Roughly, . . . decreasing transistor feature-size by λ:

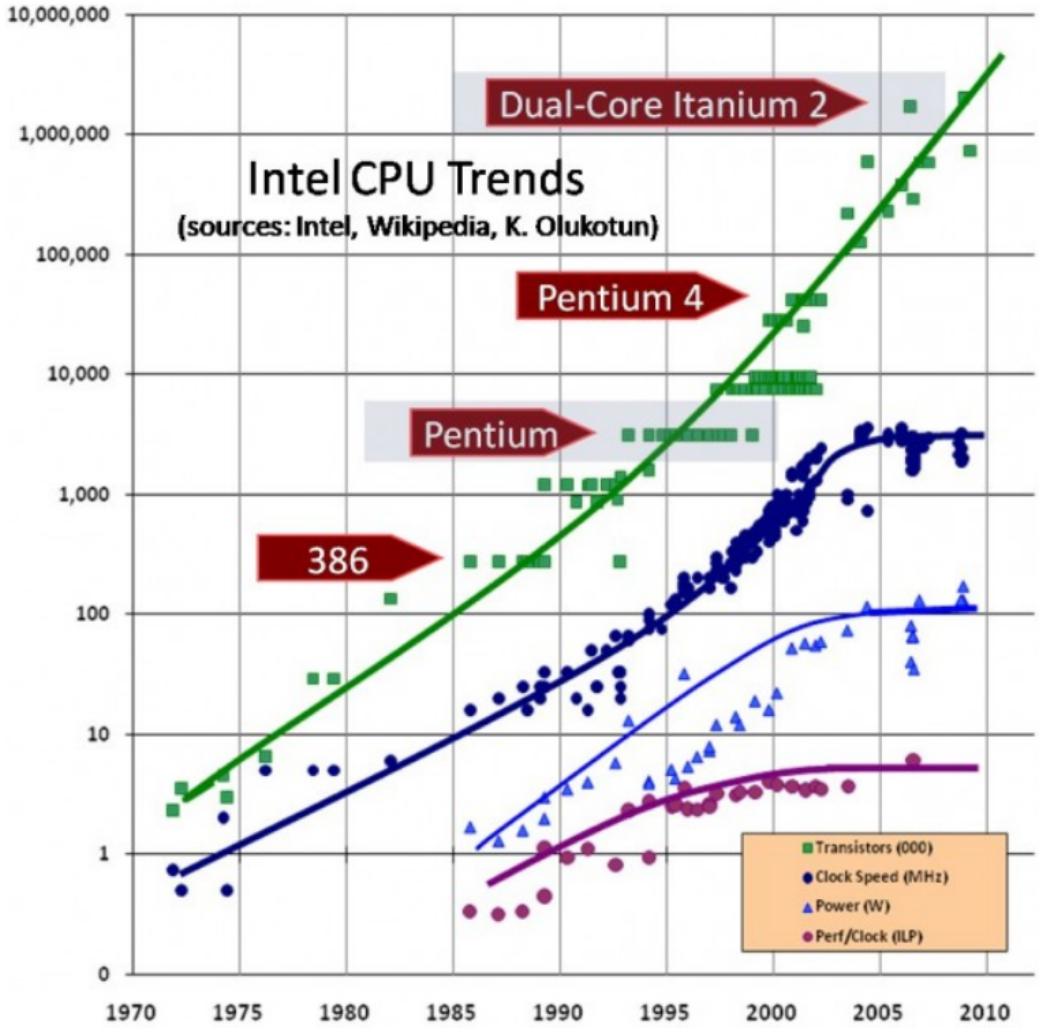
- number of transistors increase by λ^2
- clock-speed increase by λ
- **Energy comsumption does not change** !!

[1] Dennard et. al IEEE JSSC 1974

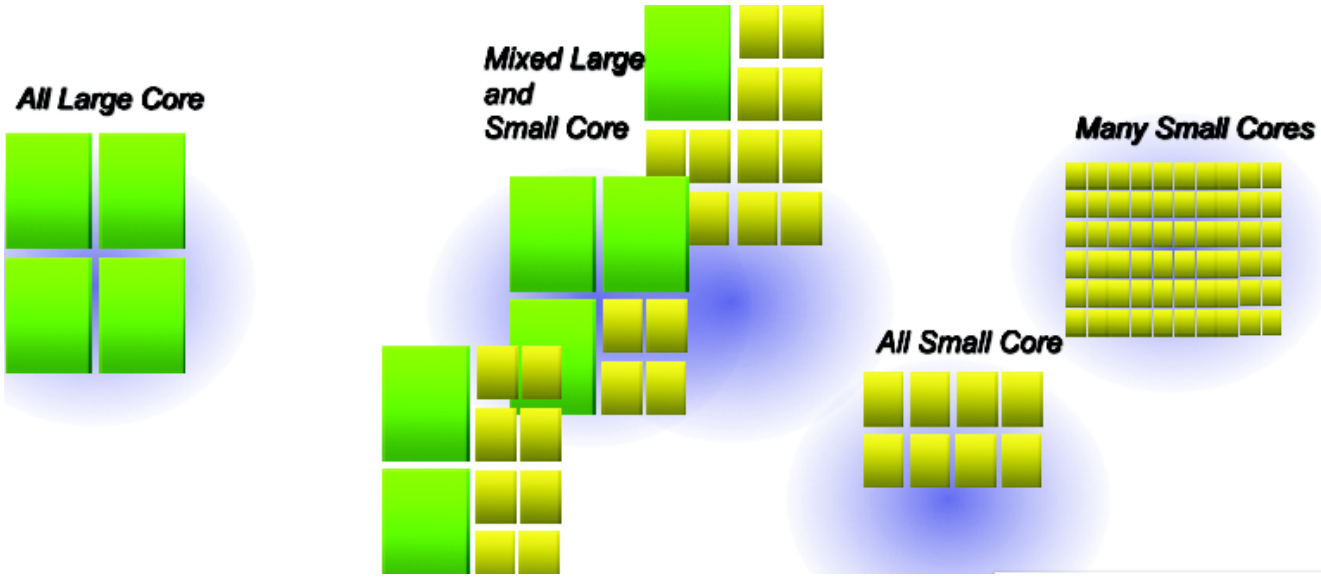
Hardware Evolution: Dennard's Law is Over !

Co-funded by the Erasmus+ Programme of the European Union

- Dennard's scaling ignore the *leakage current* and *threshold voltage*
- these created a *Power Wall* limiting processor frequency to $\approx$ 4 GHz since 2006.



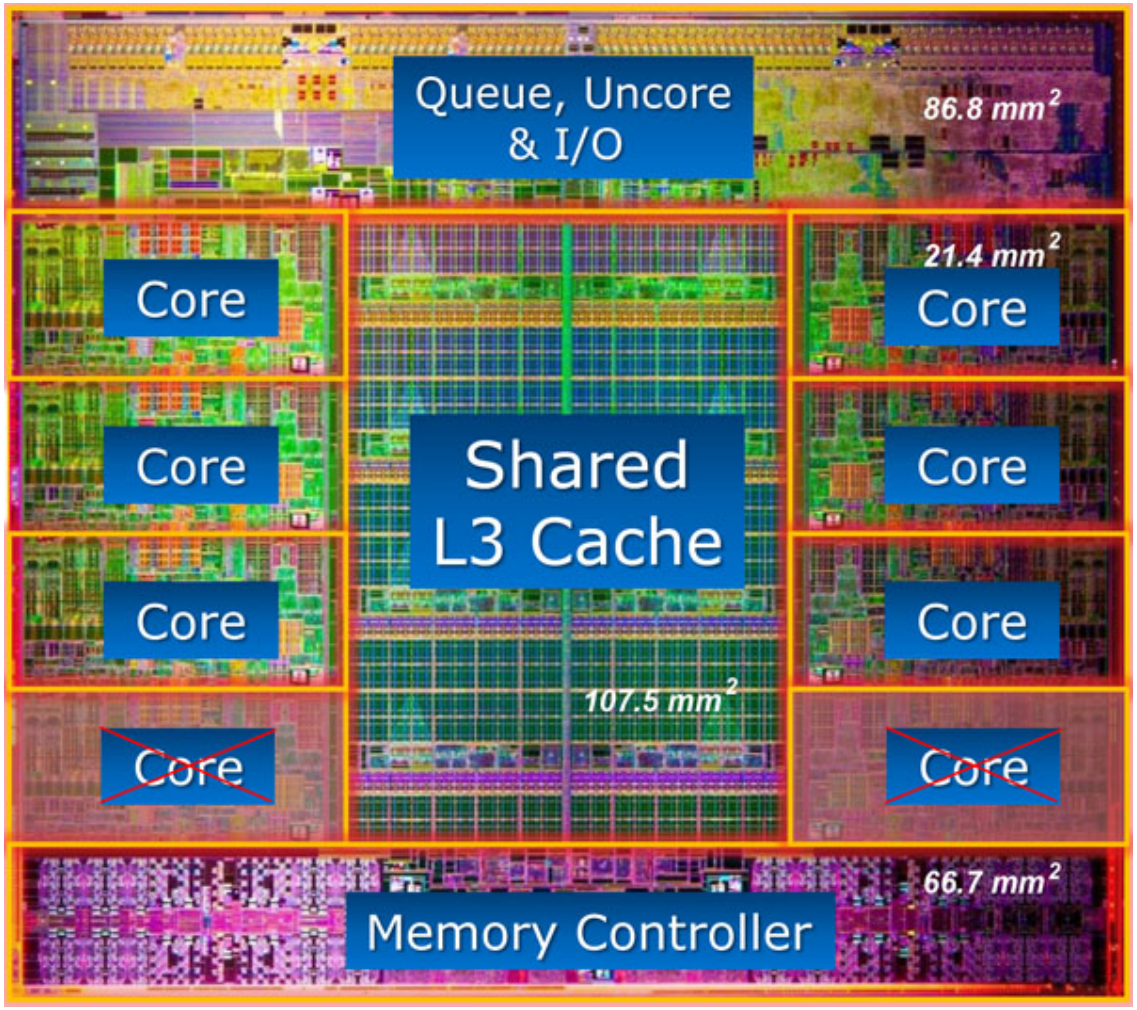
The Multi-core processors era begins !



- all large core: multi-core Intel x86 CPUs
- many small core: NVIDIA GPUs accelerators
- all small cores: MIC architectures, Intel Xeon Phi accellerator
- mixed large and small cores: Cell, AMD-Fusion, NVIDIA-Denver
- assembly more CPUs in a single silicon device ✔
- great impact on application performance and design ✘
- move challenge to exploit high-performance computing from HW to SW ✘

“Conventional” Multi-Core CPU Architectures

Co-funded by the Erasmus+ Programme of the European Union



4-8 . . . 22 cores, 1 shared L3-cache

“Conventional” CPU Architectures

Co-funded by the Erasmus+ Programme of the European Union

Main features:

- 8-22 (and soon more) cores
- frequency $\approx$ 3 GHz
- 3 levels of caches, 2 within a core and 1 shared
- support for **SIMD** execution: AVX 256-bits
- e.g.: Xeon E5-2680 Sandybridge: 691.2/345.6 GFlops SP/DP

Programming issues:

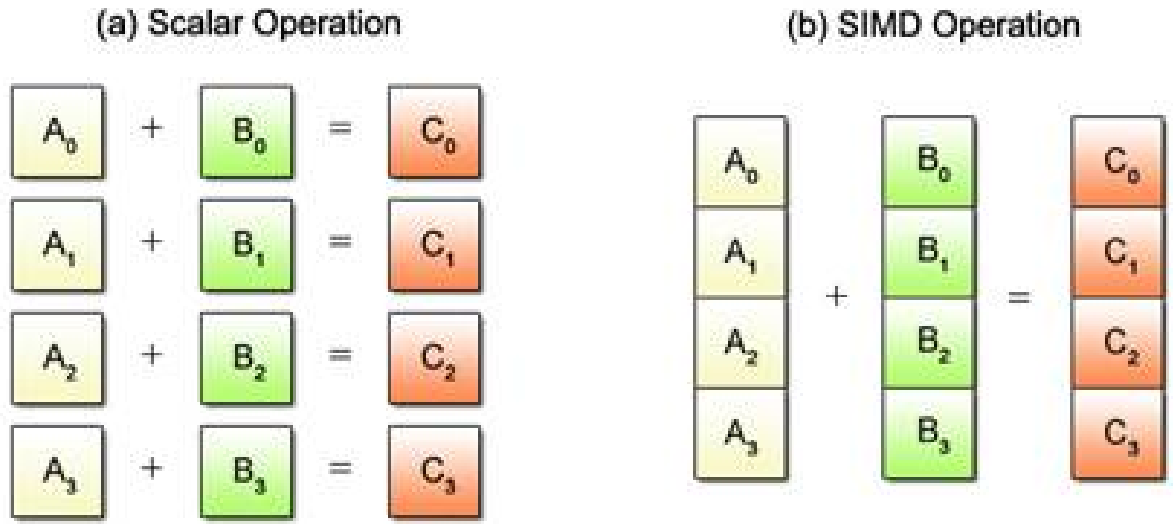
- **core parallelism**
- **data parallelism**
- **cache optimizations**
- **Non Uniform Memory Architecture** (NUMA)

Co-funded by the Erasmus+ Programme of the European Union

Performances Issues

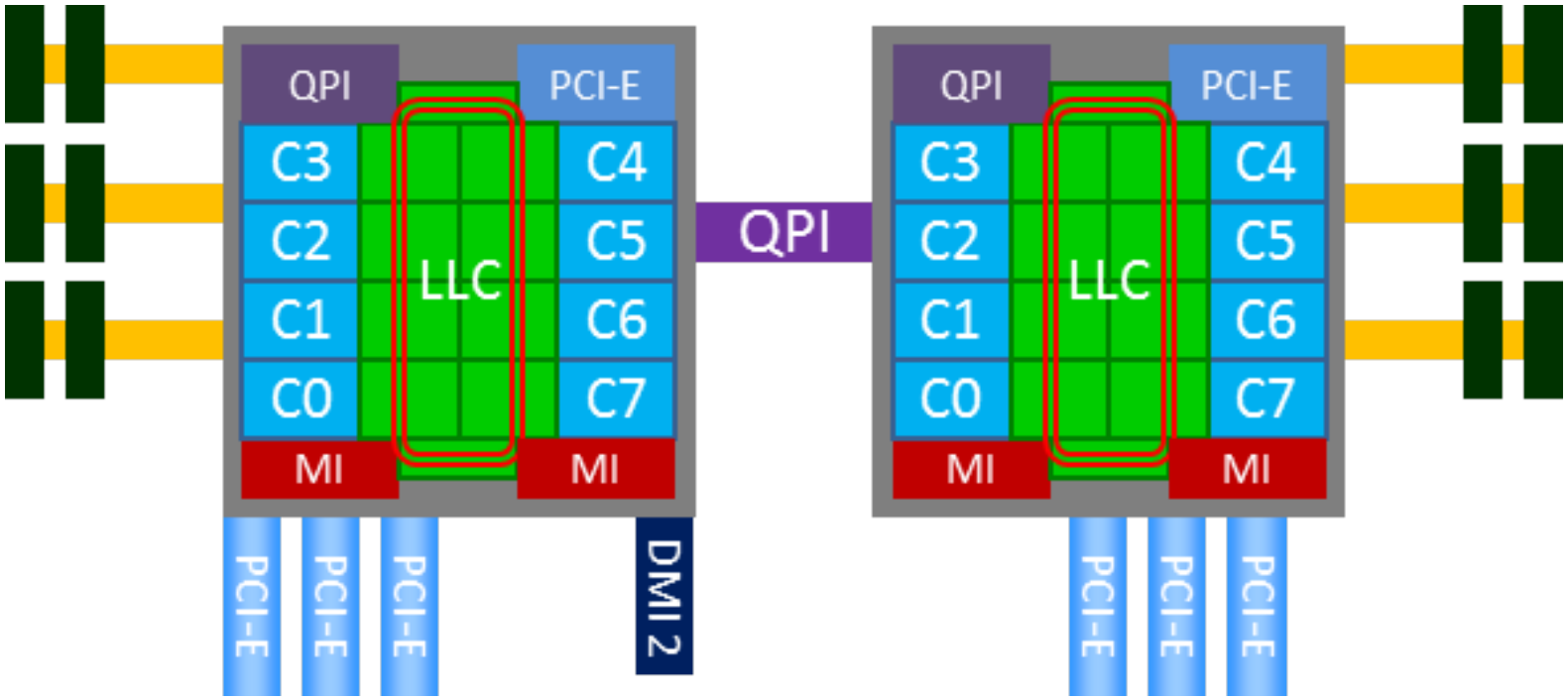
- c number of cores
- SIMD instructions on 256-bit operands:
  each vector register can pack $n = 4(8)$ double (single) precision numbers
- each core can execute two operations per clock-cycle:
  one *add* and one *mul*

$$P = f \times 2 \times n \times c$$



Numa SMP Multi-socket Multi-core Systems

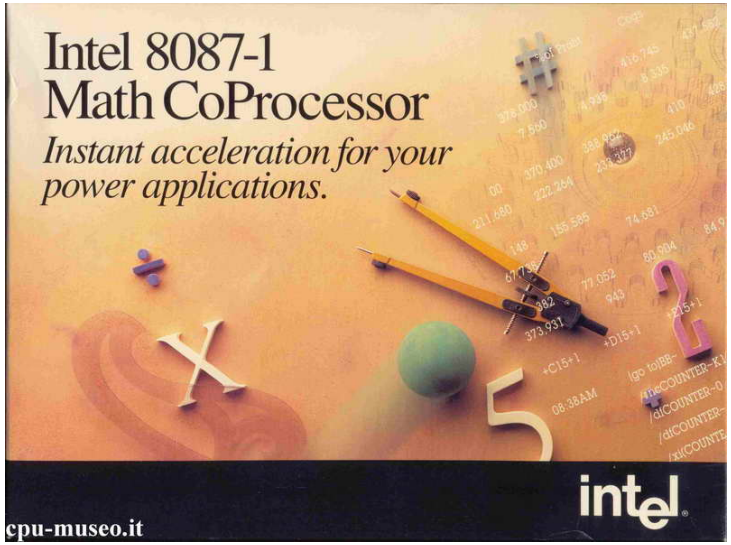
Co-funded by the Erasmus+ Programme of the European Union

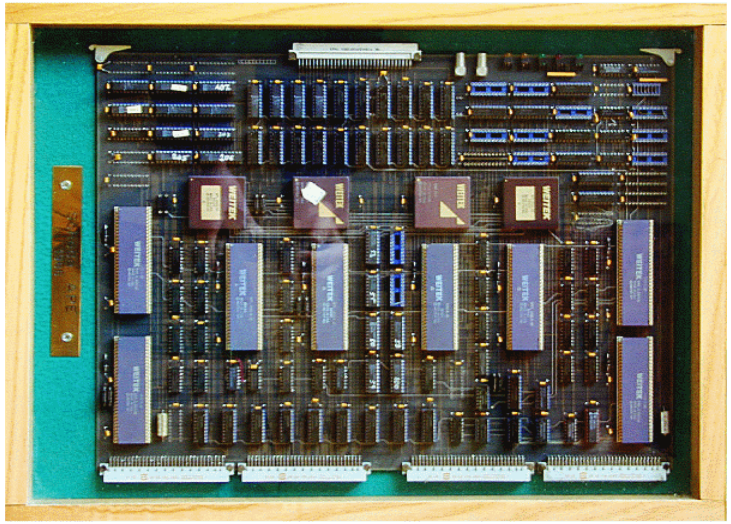


- *Symmetric Multi-processor Architecture* (SMP)
- *Non Uniform Memory Architecture* (NUMA)
- issue on allocation of data to memory

Accelerator: Is this a really new concept ?

Co-funded by the Erasmus+ Programme of the European Union





Accelerator: today it look likes much better !

Co-funded by the
Erasmus+ Programme
of the European Union

NVIDIA GPU Architecture Evolution

Co-funded by the Erasmus+ Programme of the European Union

	Intel Xeon	NVIDIA K80		AMD S9150
processor codename	E5-2630 v3	GK210		Hawaii XT
#physical-cores	8	13	x 2	44
#logical-cores	16	2496	x 2	2816
nominal clock Freq. (GHz)	2.1	0.562		0.900
Nominal GFLOPS (DP)	268.625	935	x 2	**2530**
Max Boosted clock Freq. (GHz)	2.6	0.875		N/A
Boosted GFLOPS (DP)	**331.56**	**1455**	x 2	N/A
Max Memory (GB)	768	12	x 2	16
Mem Bandwidth (GB/s)	59	240	x 2	320
ECC	YES	YES		YES

The Sausage Machine Model

A GPU is like a sausage machine:

- . . . no input-meat . . . no output-sausage !!
- . . . it produces output-results if you provide enough input-data !!

Are accellerators good sausage machines ?

Co-funded by the Erasmus+ Programme of the European Union

FPS-164 and VAX (1976):

- Floating Point: F = 11 Mflop/s, IO Rate: B = 44 MB/s
- Ratio of flops to bytes of data movement: R = **0.25** Flops / Byte
- Host-device latency: $\mathcal{O}(1)$ clock-cycle

Nvidia Kepler K20 and PciE (2012):

- Floating Point: F = 1170 Gflop/s (DP), IO Rate: B = 8 GB/s
- Ratio of flops to bytes of data movement: R = **146.25** Flops / Byte
- Host-device latency: $\mathcal{O}(10 - 100)$ clock-cycles

- Flop/s are cheap, so are provisioned in excess,
- data needs to be re-used and processed several times by the FPUs,
- smart programming techniques to hide data movement latency, e.g. recompute data instead of access memory.

Performance Evaluation: Amdhal's Law

Co-funded by the Erasmus+ Programme of the European Union

How much can I accelerate my application ?

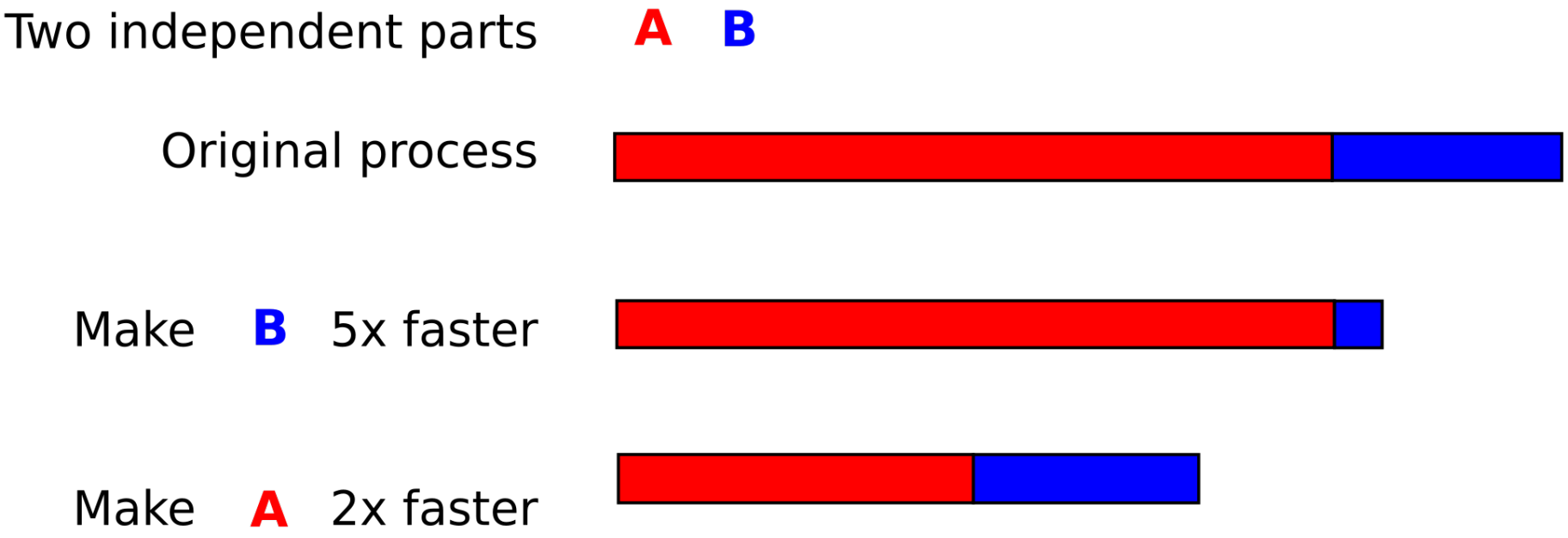
Amdahl's Law roughly states:

> *Suppose a car is traveling between two cities 60 miles apart, and has already spent one hour traveling half the distance at 30 mph.*
>
> *No matter how fast you drive the last half, it is impossible to achieve 90 mph average before reaching the second city.*
>
> *Since it has already taken you 1 hour and you only have a distance of 60 miles total,*
>
> *going infinitely fast you would only achieve 60 mph.*

Accelerator and the Amdahl's Law

Amdahl's Law

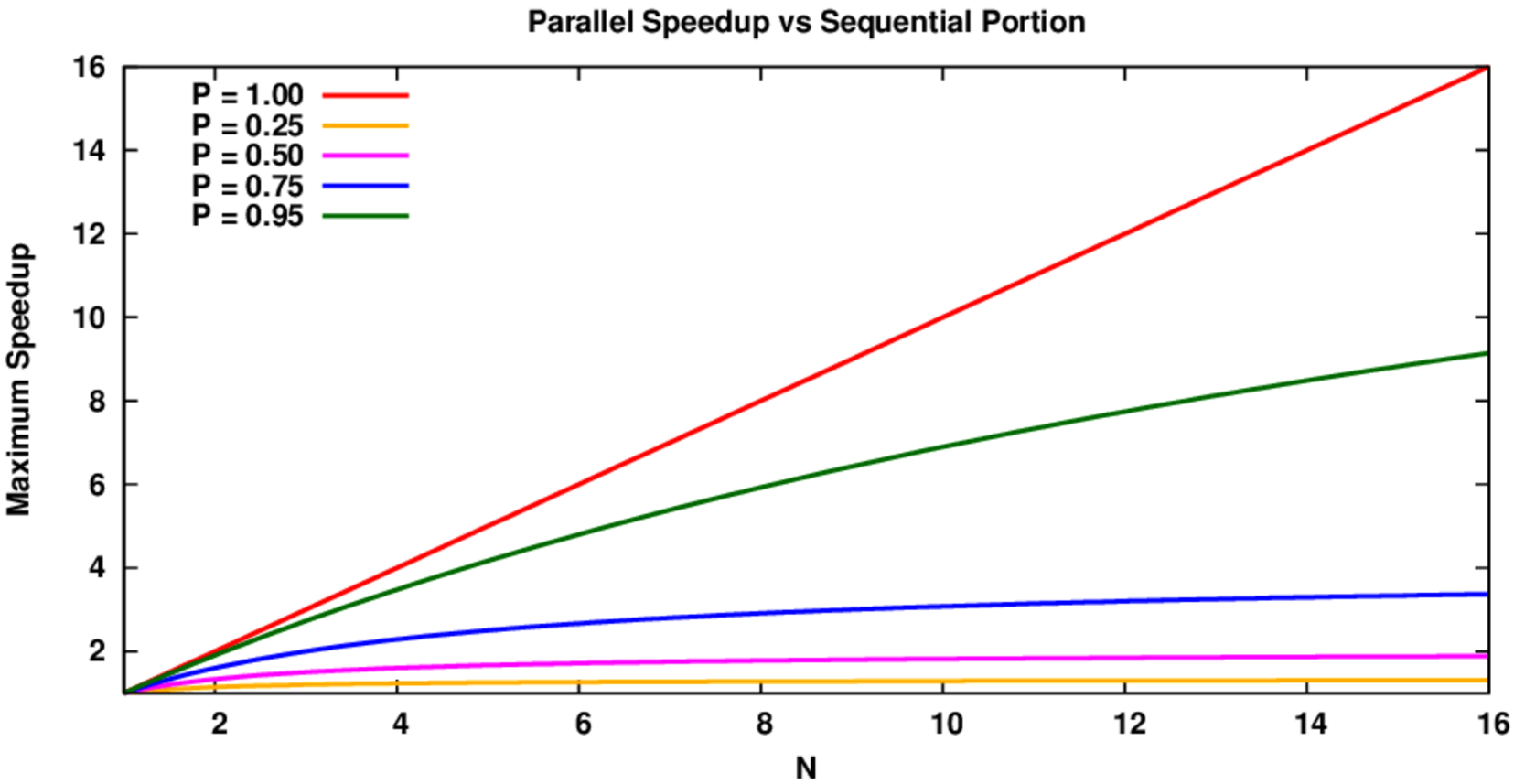
The effective speedup of an accelerated program is limited by the time needed for the host fraction of the program.



Co-funded by the Erasmus+ Programme of the European Union

Accelerator Issues: the Amdahl's law

Let assume that P is the fraction of code accelerated, and N is the improving factor, plotting the speed-up as function of N:



even if we accelerate the 3/4 of our code, by large values of N the maximum speedup we can achieve is limited to 4 !!!

Accelerator Issues: Host-Device Latency

Co-funded by the Erasmus+ Programme of the European Union

Anonymous

. . . bandwidth problems can be cured with money.

Latency problems are harder because the speed of light is fixed - you can't bribe Nature.

Moving data between Host and GPU is limited by **bandwidth** and **latency**:

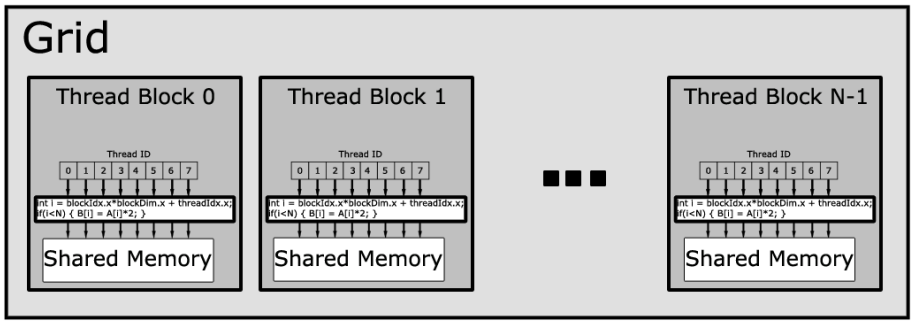
$$T(n) = l + n/B$$

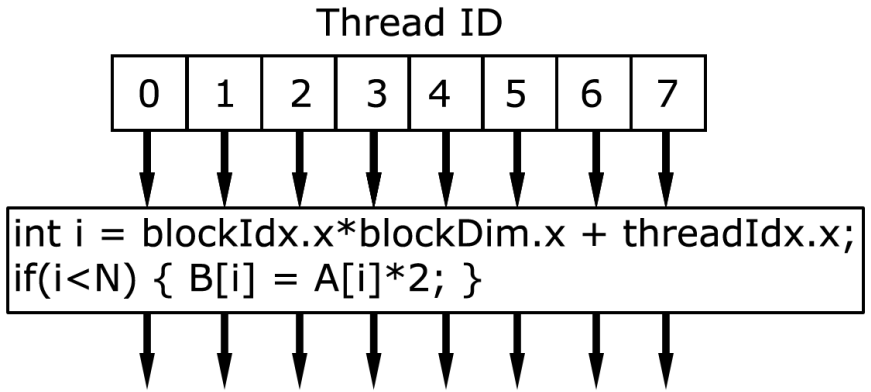
- accelerator processor clock period is $\mathcal{O}(1)$ns
- PciE latency is $\mathcal{O}(1)\mu$s

GPU Programming Model

Co-funded by the Erasmus+ Programme of the European Union

- execution has an hierarchical structure:
  - a grid of blocks
  - each block is a 1-2-3 D array of threads
- host launches a **grid** of thread-blocks
- a CUDA kernel (program executed on the device) is executed by an array of **threads**





Vector sum example

```
// device code
__global__ void vadd ( double * A, double * B, double * C ) {
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  C[i] = A[i] + B[i];
}

int main () {
  double A_h[N], B_h[N], C_h[N];
  double * A_d, * B_d, * C_d;

  srand48();
  vinit(double *A, double *B, double *C);

  // allocate and copy data on the device
  cudaMalloc((void**) &A_d); cudaMalloc((void**) &B_d); cudaMalloc((void**) &C_d);
  cudaMemcpy(A_d, A_h, N, H2D); cudaMemcpy(B_d, B_h, N, H2D); cudaMemcpy(C_d, C_h, N, H2D);

  dim3 dimBlock(64,  1) ; // size of thread-block
  dim3 dimGrid(N/64, 1) ; // size of block-grid

  // run kernel
  vadd <<< dimGrid, dimBlock >>> (A,B,C);

  cudaThreadSynchronize(); // wait until kernel terminates !!!!

  // copy results back to host
  cudaMemcpy(C_h, C_d, N, D2H);

  // feee memory device
  cudaFree(A_d); cudaFree(B_d); cudaFree(C_d);
}
```

GPU Programming Issues

Co-funded by the Erasmus+ Programme of the European Union

- host-to-device latency:
  Amdhal's law
- memory access latency:
  $\mathcal{O}(10^3)$ processor cycles, run many threads to hide memory-latency
- high-data parallelism:
  many threads-per-block and many blocks-per-grid

Co-funded by the Erasmus+ Programme of the European Union

So ... what's better ? Multi-core CPUs or Accelerators

in other words ... what's better to plow a ground ?

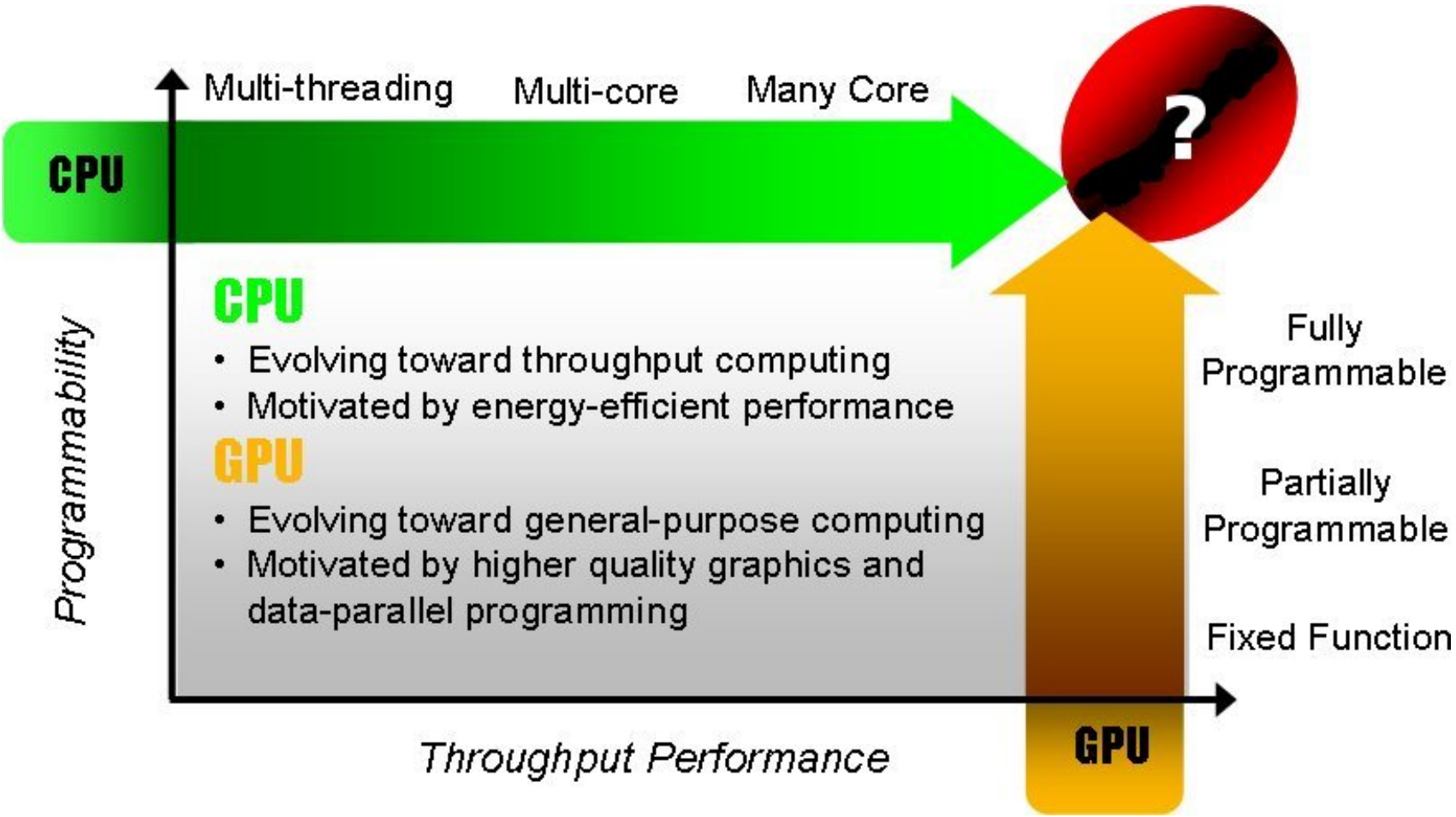
It depends on what do we need. As rule of thumb:

- low-latency and reasonable throughput: left
- high-througput and reasonable-latency: right

Better if you can use both !!! May be hard to program and get good efficiency !

Where we are going ?

Co-funded by the Erasmus+ Programme of the European Union



. . . towards a convergence between CPU and GPU architectures

First attempt to merge GPU and CPU concepts

Co-funded by the Erasmus+ Programme of the European Union

MIC: Many Integrated Core Architecture

- Knights Ferry: development board
- Knights Corners: production board
- Intel Xeon-Phi: commercial board
- Knights Landing next generation

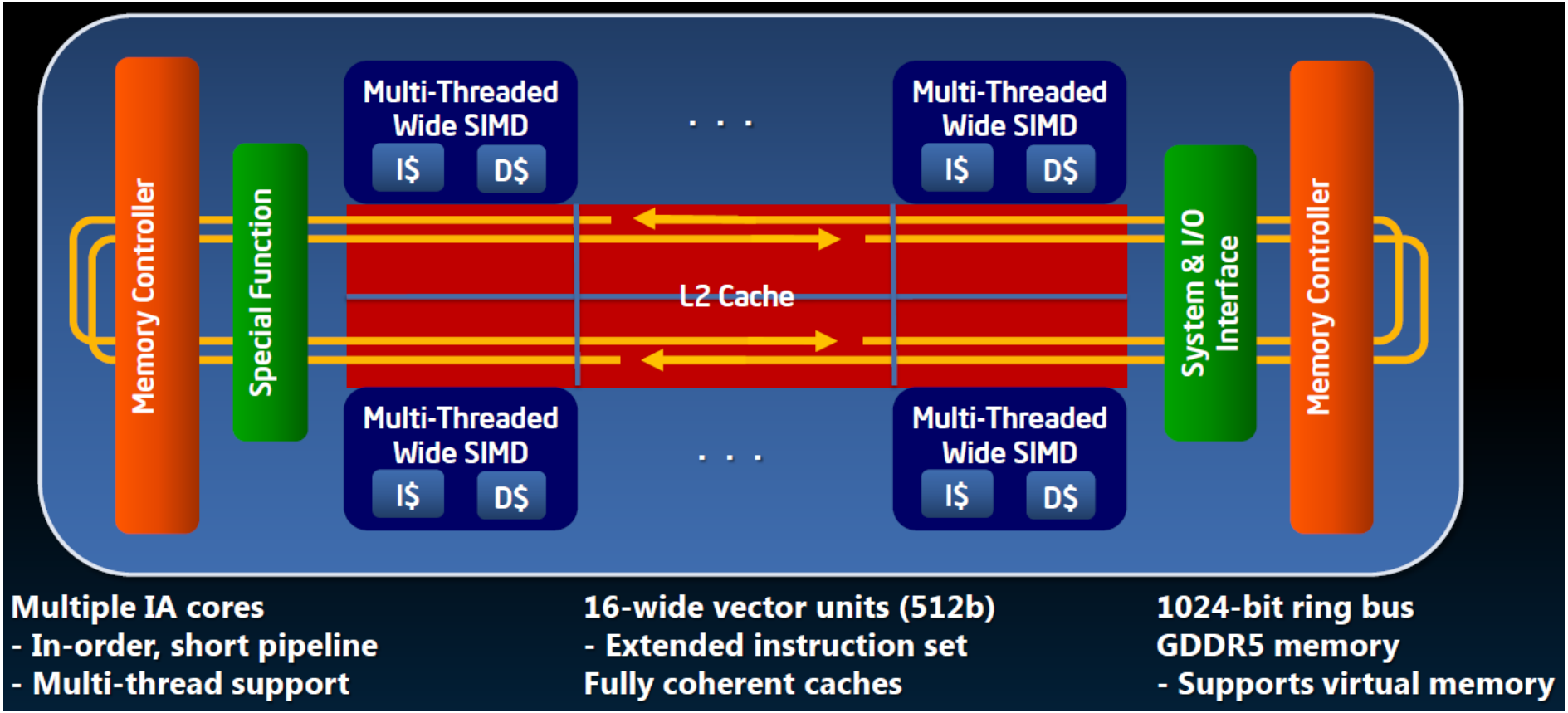
Intel MIC Systems: Knights Corners

Co-funded by the Erasmus+ Programme of the European Union

- PCIe interface
- Knights Corners: 61 x86 core @ 1.2 GHz
- each core has 32KB L1 instruction cache, 32KB L1 data cache, and 256KB L2 cache
- 512-bit SIMD unit: 16 SP, 8 DP
- multithreading: 4 threads / core
- 8 MB L3 shared coherent cache
- 4-6 GB GDDR5

MIC Architectures

Co-funded by the Erasmus+ Programme of the European Union



- cores based on Pentium architecures
- $\approx$ 60 cores
- in-order architecture
- 512-bit SIMD instructions

MIC Programming Model

Co-funded by the Erasmus+ Programme of the European Union

- native:
  `icc -mmic pippo.c -o pippo`
- offload:
  using approriate `pragmas` to mark code that will be *transparently* executed onto the MIC board

Programming is well integrated with many languages:

- openMP
- TBB
- Cilk
- ...

Parallelism management

- offload a code that spanws threads
- use openMP

```
for(t = 0; t < NTHREAD; t++) {
  pthread_create(&threads[t], NULL, threadFunc, (void *) &tData[t]);
}

for(t = 0; t < NTHREAD; t++) {
  pthread_join(threads[t], NULL);
}
```

```
#pragma omp parallel private(tid)
{
  tid = omp_get_thread_num();
  theadFunc((void *) &targv[tid]);
}
```

Example: vector sum

```
#define N 1717

void __attribute__((target(mic))) vadd ( double *A , double *B , double *C )

void vinit (double *A, double *B, double *C) {
  int i;
  for (i=0; i<N; i++){
    A[i] = drand48(); B[i] = drand48(); C[i]= 0.0;
  }
}

int main () {
  double A[N], B[N], C[N];
  srand48();
  vinit(double *A, double *B, double *C);
  ....
  #pragma offload target(mic:0) in(A,B:lenght(N)) inout(C:lenght(N))
  {
    vadd (A,B,C);
  }
  ....
}

void vadd (double *A, double *B, double *C) {
#ifdef __MIC__
  int i;
  for (i=0; i<N; i++)
    C[i] = A[i] + B[i];
#else
  fprint(stderr, "This code is running on the host\n");
#endif
}
```

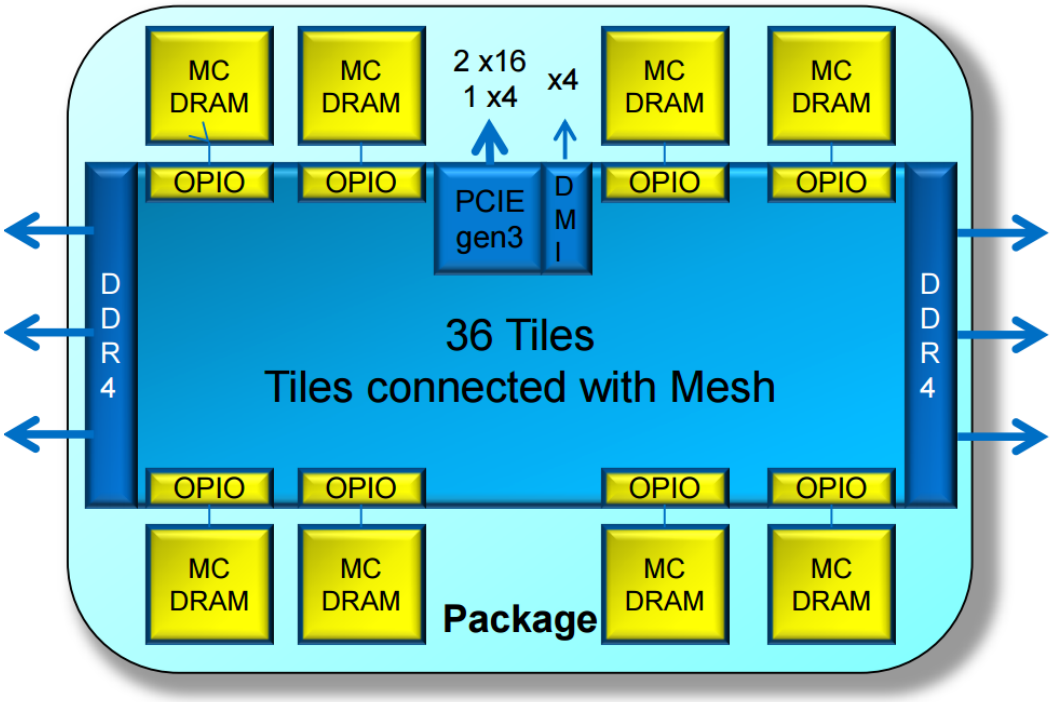
MIC Programming Issues

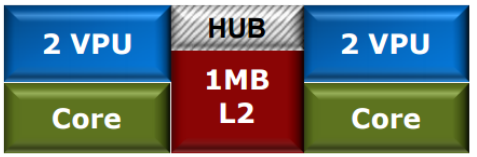
Co-funded by the Erasmus+ Programme of the European Union

- **core parallelism**:
  - keep all 60 cores (1 reserver for OS) busy
  - runs 2-3 (up-to) 4 threads/core is necessary to hide memory latency
- **vector parallelism**:
  - enable data-parallelism
  - enable use of 512-bit vector instructions
- **Amdhal's law**:
  - transfer time between host and MIC-board not negligible
  - hide transfer time overlapping computation and processing

Intel Knights Landing

Knights Landing Overview





- Stand-alone, Self-boot CPU
- Up to 72 new Silvermont-based cores
- 4 Threads per core. 2 AVX 512 vector units
- Binary Compatible[1] with Intel® Xeon® processor
- 2-dimensional Mesh on-die interconnect

- MCDRAM: On-Package memory: 400+ GB/s of BW[2]
- DDR memory
- Intel® Omni-path Fabric

- 3+ TFLops (DP) peak per package
- ~3x ST performance over KNC

Conclusions

Multi-core architectures have a big inpact on programming.

- Efficient programming requires to exploit all features of hardware systems:
  - core parallelism
  - data parallelism
  - cache optimizations
  - NUMA (Non Uniform Memory Architecture) system
- Accelerators are not a *panacea*:
  - good for desktop-applications
  - hard to scale on large clusters

the one million dollar question

So . . . which is the best computing system to use ?