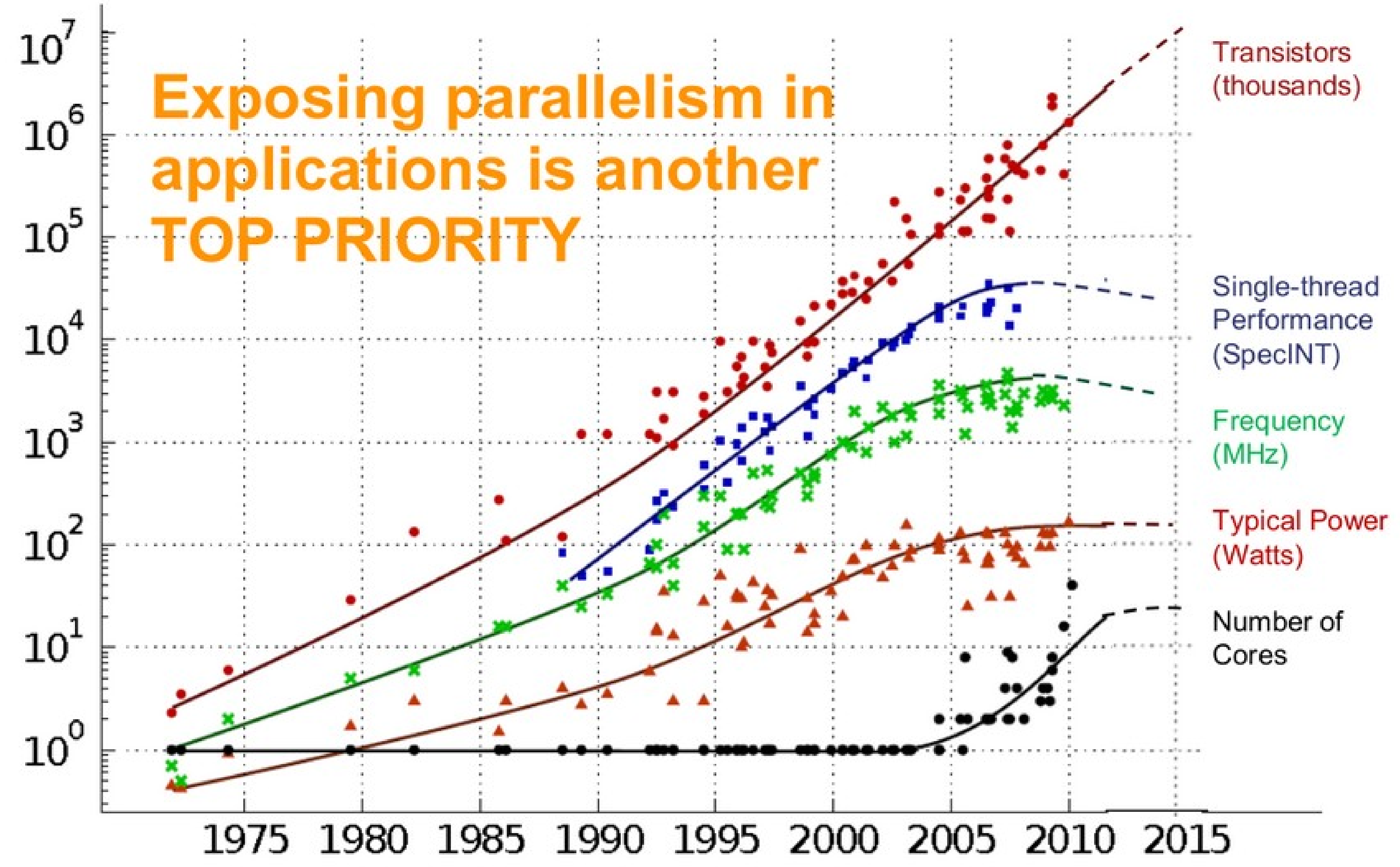
# Approaches to acceleration: GPUs vs Intel MIC

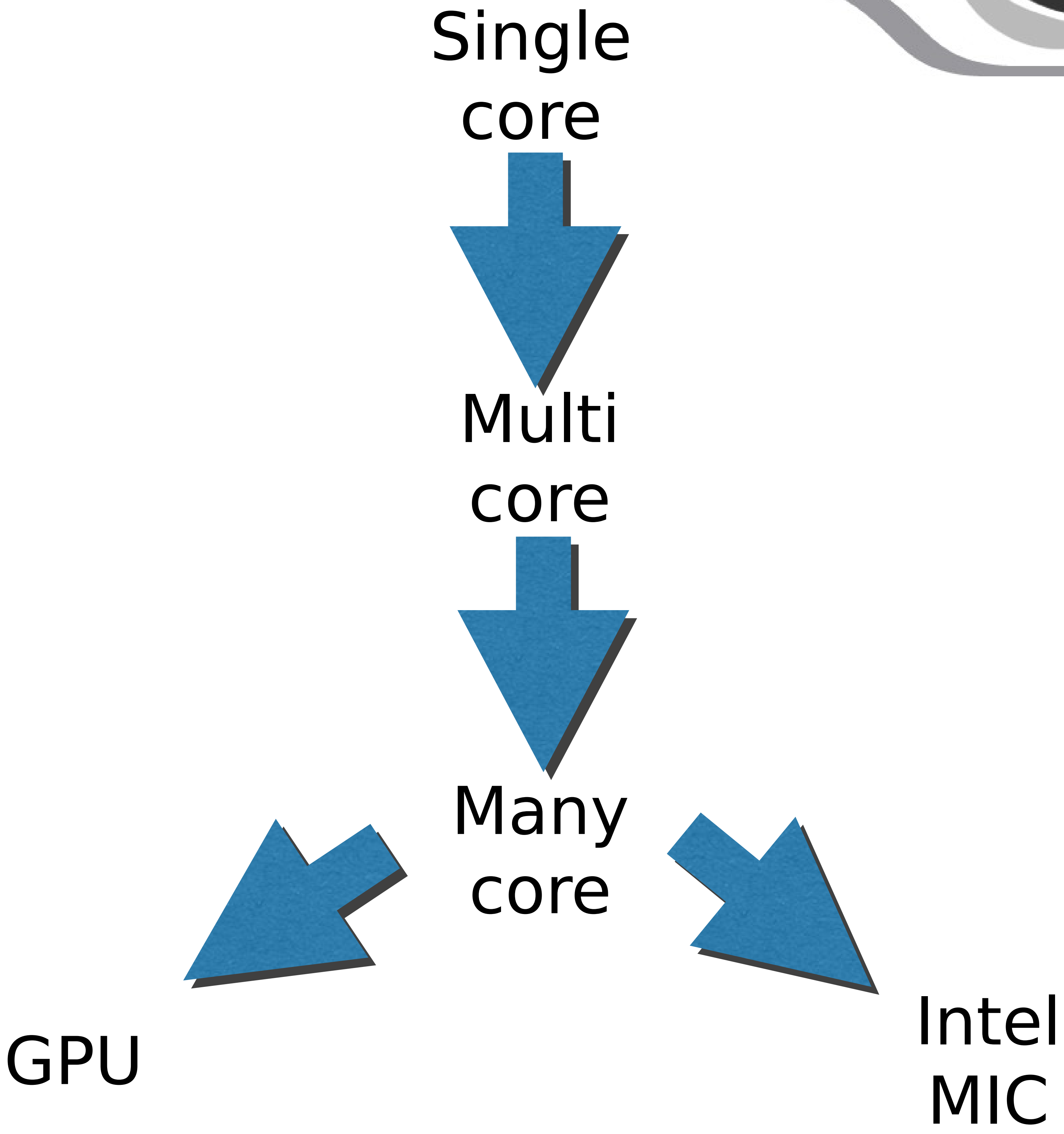
Fabio AFFINITO
SCAI department

## 35 Years of Microprocessor Trend Tata



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Source: C. Moore's talk at Salishan, April 2011


Single core
Multi core
Many core
GPU
Intel MIC

| Rank | Site | System | Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|---|---|---|---|---|---|---|
| 1 | National Super Computer Center in Guangzhou<br>China | **Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P<br>NUDT | 3120000 | 33862.7 | 54902.4 | 17808 |
| 2 | DOE/SC/Oak Ridge National Laboratory<br>United States | **Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x<br>Cray Inc. | 560640 | 17590.0 | 27112.5 | 8209 |
| 3 | DOE/NNSA/LLNL<br>United States | **Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom<br>IBM | 1572864 | 17173.2 | 20132.7 | 7890 |
| 4 | RIKEN Advanced Institute for Computational Science (AICS)<br>Japan | K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect<br>Fujitsu | 705024 | 10510.0 | 11280.4 | 12660 |
| 5 | DOE/SC/Argonne National Laboratory<br>United States | **Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom<br>IBM | 786432 | 8586.6 | 10066.3 | 3945 |
| 6 | Swiss National Supercomputing Centre (CSCS)<br>Switzerland | **Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x<br>Cray Inc. | 115984 | 6271.0 | 7788.9 | 2325 |
| 7 | Texas Advanced Computing Center/Univ. of Texas<br>United States | **Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P<br>Dell | 462462 | 5168.1 | 8520.1 | 4510 |
| 8 | Forschungszentrum Juelich (FZJ)<br>Germany | **JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect<br>IBM | 458752 | 5008.9 | 5872.0 | 2301 |
| 9 | DOE/NNSA/LLNL<br>United States | **Vulcan** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect<br>IBM | 393216 | 4293.3 | 5033.2 | 1972 |
| 10 | Leibniz Rechenzentrum<br>Germany | **SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR<br>IBM | 147456 | 2897.0 | 3185.1 | 3423 |

| Green500 Rank | MFLOPS/W | Site* | Computer* | Total Power (kW) |
|---|---|---|---|---|
| 1 | **4,503.17** | GSIC Center, Tokyo Institute of Technology | TSUBAME-KFC - LX 1U-4GPU/104Re-1G Cluster, Intel Xeon E5-2620v2 6C 2.100GHz, Infiniband FDR, NVIDIA K20x | 27.78 |
| 2 | **3,631.86** | Cambridge University | Wilkes - Dell T620 Cluster, Intel Xeon E5-2630v2 6C 2.600GHz, Infiniband FDR, NVIDIA K20 | 52.62 |
| 3 | **3,517.84** | Center for Computational Sciences, University of Tsukuba | HA-PACS TCA - Cray 3623G4-SM Cluster, Intel Xeon E5-2680v2 10C 2.800GHz, Infiniband QDR, NVIDIA K20x | 78.77 |
| 4 | **3,185.91** | Swiss National Supercomputing Centre (CSCS) | Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x<br>Level 3 measurement data available | 1,753.66 |
| 5 | **3,130.95** | ROMEO HPC Center - Champagne-Ardenne | romeo - Bull R421-E3 Cluster, Intel Xeon E5-2650v2 8C 2.600GHz, Infiniband FDR, NVIDIA K20x | 81.41 |
| 6 | **3,068.71** | GSIC Center, Tokyo Institute of Technology | TSUBAME 2.5 - Cluster Platform SL390s G7, Xeon X5670 6C 2.930GHz, Infiniband QDR, NVIDIA K20x | 922.54 |
| 7 | **2,702.16** | University of Arizona | iDataPlex DX360M4, Intel Xeon E5-2650v2 8C 2.600GHz, Infiniband FDR14, NVIDIA K20x | 53.62 |
| 8 | **2,629.10** | Max-Planck-Gesellschaft MPI/IPP | iDataPlex DX360M4, Intel Xeon E5-2680v2 10C 2.800GHz, Infiniband, NVIDIA K20x | 269.94 |
| 9 | **2,629.10** | Financial Institution | iDataPlex DX360M4, Intel Xeon E5-2680v2 10C 2.800GHz, Infiniband, NVIDIA K20x | 55.62 |
| 10 | **2,358.69** | CSIRO | CSIRO GPU Cluster - Nitro G16 3GPU, Xeon E5-2650 8C 2.000GHz, Infiniband FDR, Nvidia K20m | 71.01 |

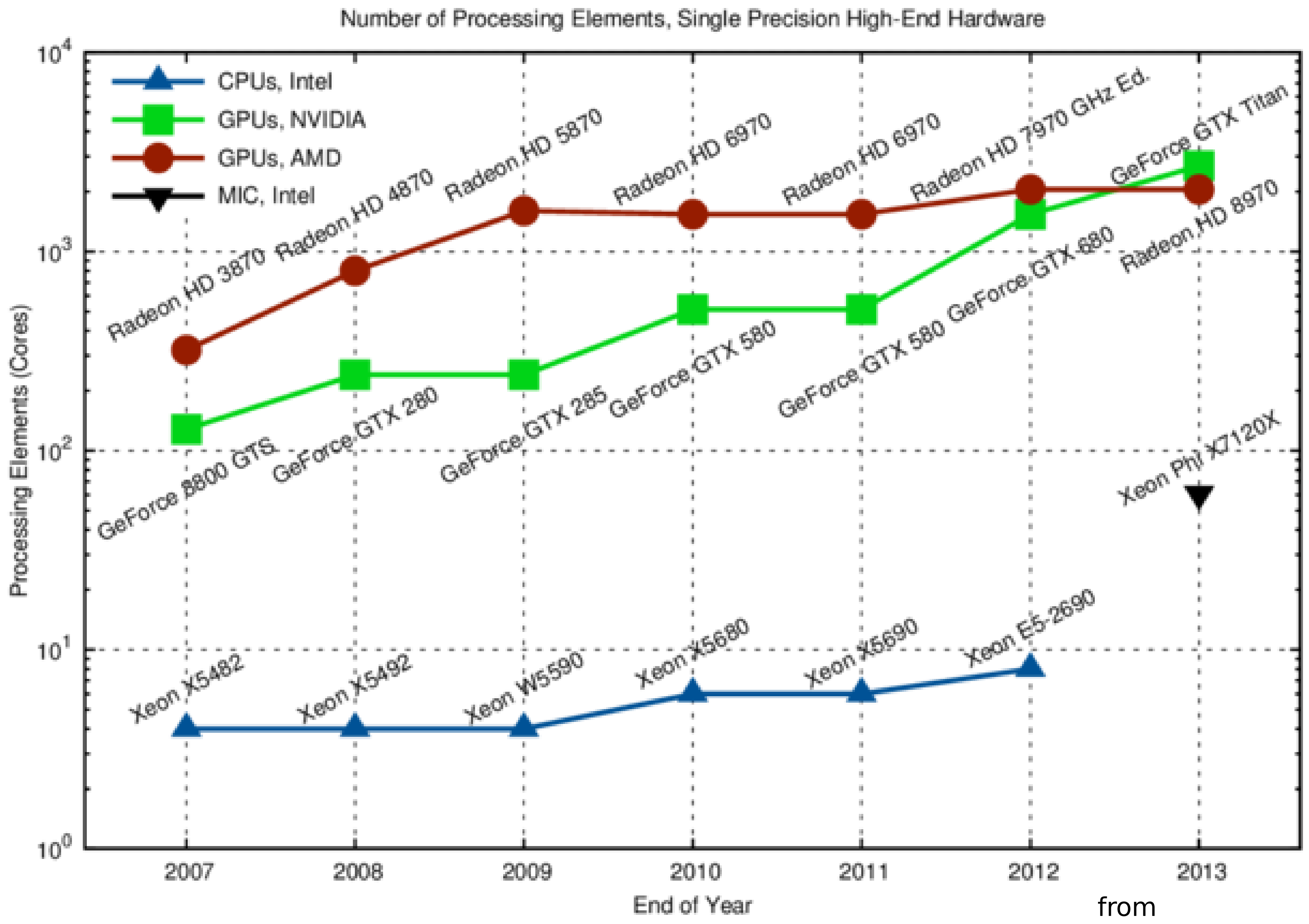
**NVIDIA Tesla K20**

- 13 Multiprocessors
- 2496 CUDA Cores
- 5 GB of global memory
- GPU clock rate 760MHz

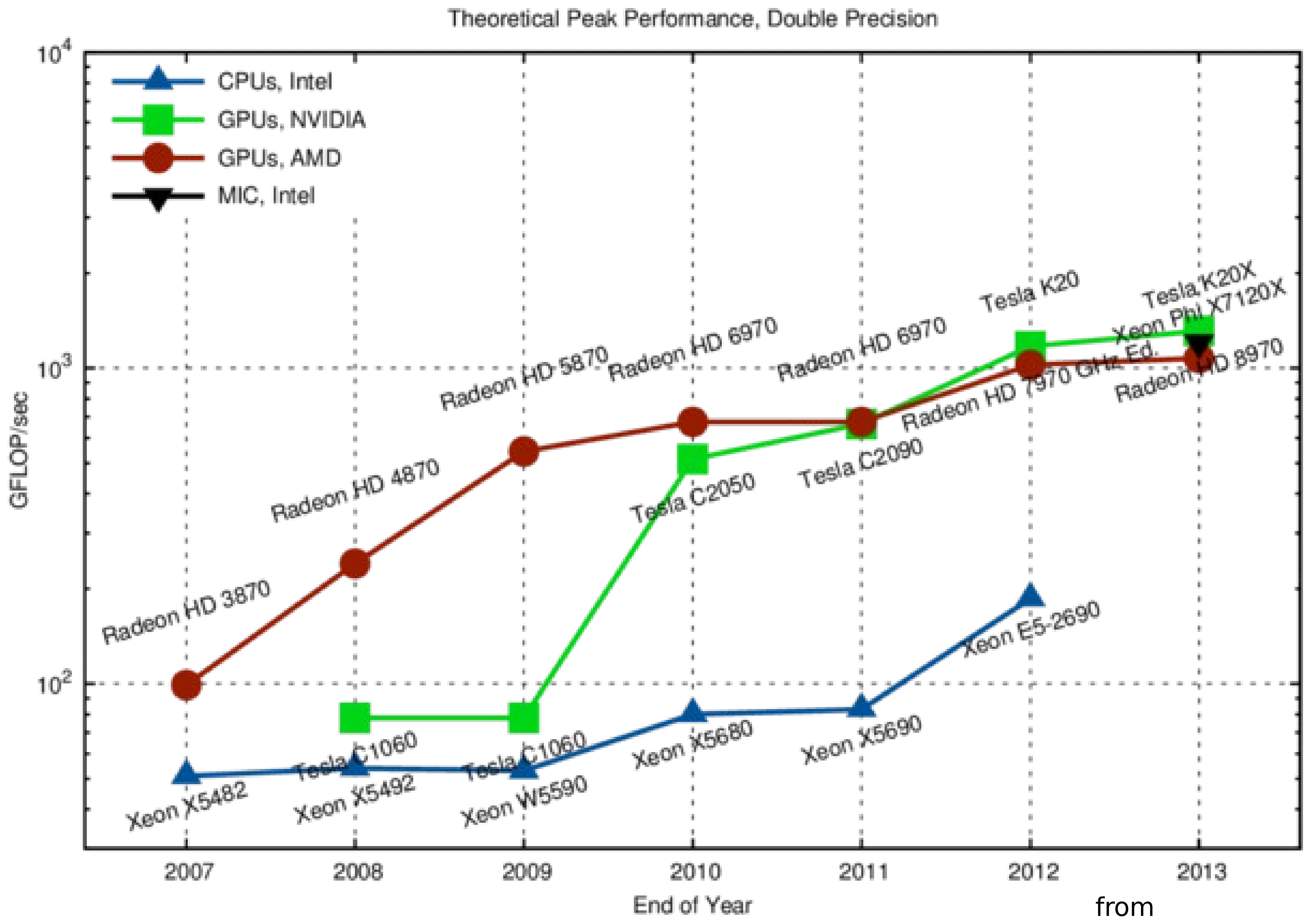
**Intel MIC Xeon Phi**

- 236 compute units
- 8 GB of global memory
- CPU clock rate 1052 MHz

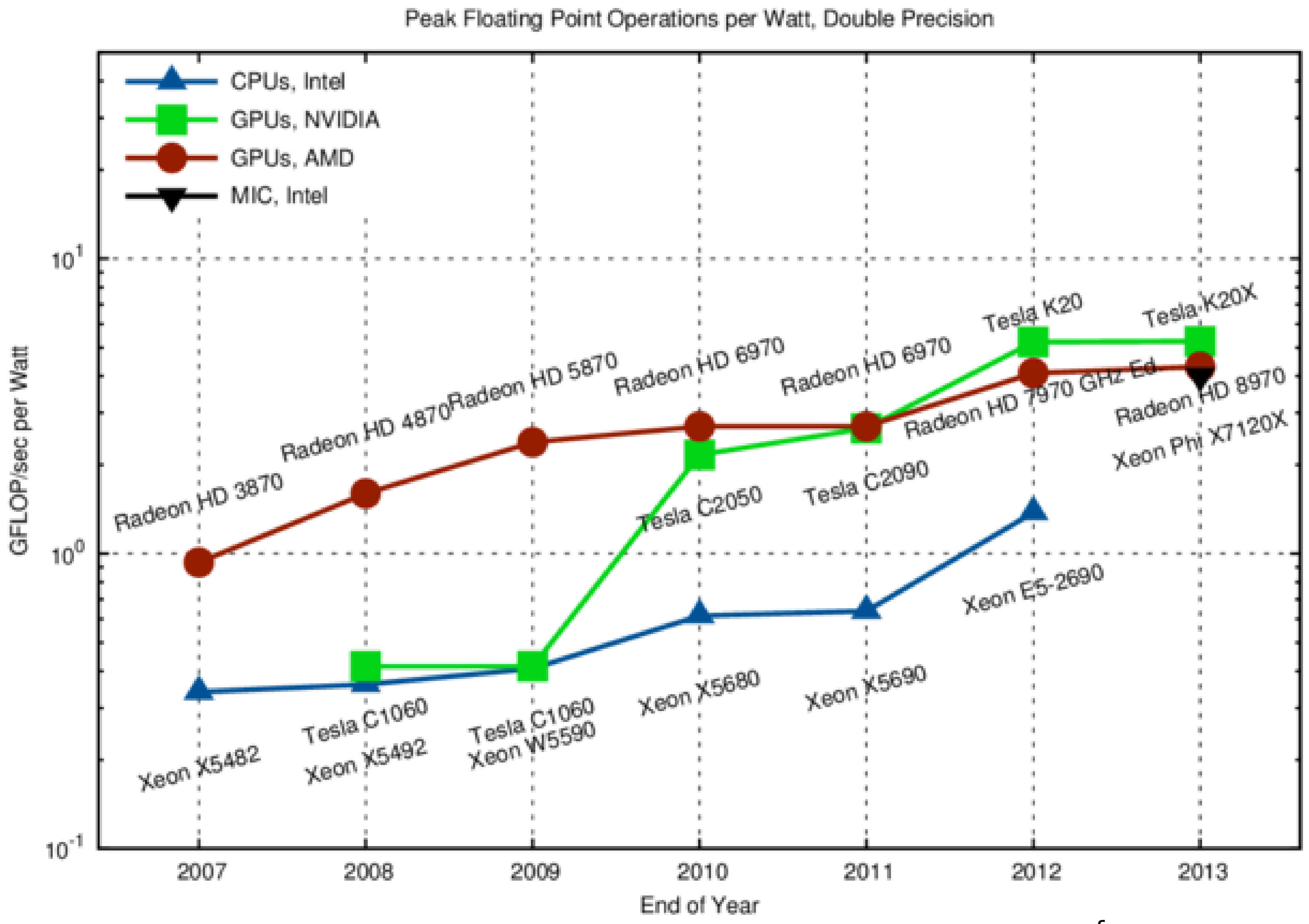
- 61 cores
- 512bit-SIMD units



from
http://www.karlrupp.net/



from
http://www.karlrupp.net/



from
http://www.karlrupp.net/

**GP U**

**MI C**

Both are devices connected through a PCIe to a CPU (=bottleneck in bandwith).
Both have a large number of computing units.

CUDA
OpenACC
OpenCL

MPI+OpenMP within offload directives

OpenCL

Quantum ESPRESSO is an integrated suite of Open-Source computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials.

Intensive computational kernes are based on:

*- linear algebra (diagonalization, matrix-matrix mult. etc.)*

*- Fourier transform*

## ΦGEMM

Project started in 2010 (with PRACE1IP) by Girotto and Spiga.

**Goals**:
- create a library to transparently accelerate part of a complex application
- rely on existing CUBLAS algebraic libraries
- preserve the maintainability of the application

## Profiling of Quantum ESPRESSO

- calculation of charge density
  - FFT + matrix-matrix multiplication

- calculation of potential
  - FFT + operations on real-space grid

- Davidson iterative diagonalization (SCF)
  - FFT + eigenproblem + matrix-matrix multiplications

Most CPU time is spent in linear-algebra operations implemented in BLAS and LAPACK libraries, and in FFT

## The 3 best rules for fast GPU codes

1. get the data on the GPU and keep it there

2. give the GPU enough work to do

3. reuse and locate data to avoid global memory bandwith bottlenecks

**caveat**: not always true... Not always possible..
What is the trasde-off between performance and effort?

## The 3 best rules for fast MIC codes

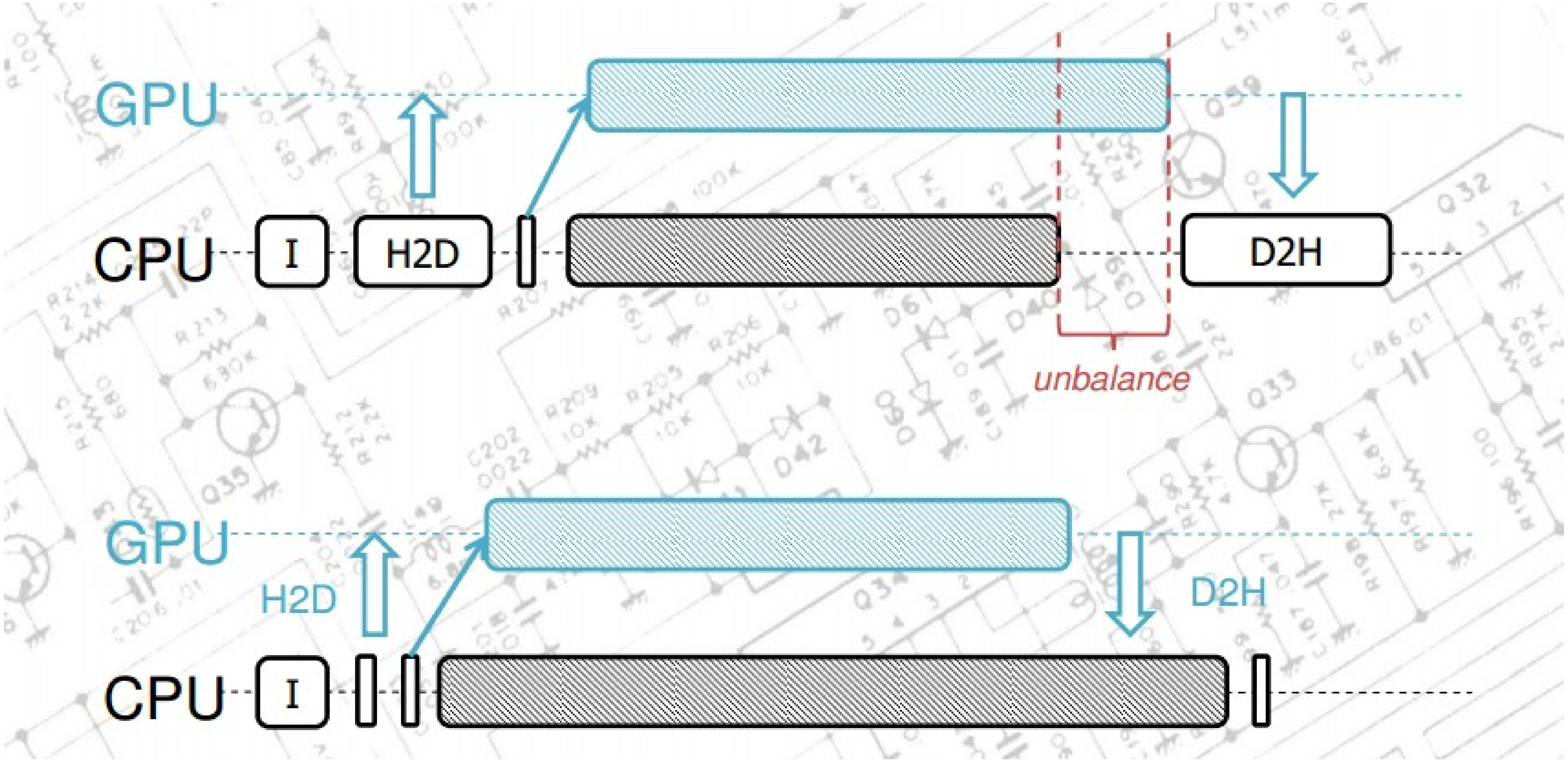
1. get the data on the MIC and keep it there

2. give the MIC enough work to do

3. reuse and locate data to avoid global memory bandwith bottlenecks

caveat: not always true...

## The phiGEMM design



## The split issue

Division of workload between multi-core CPUs and multiple GPUs is not easy because multiple factors can affect the performance.

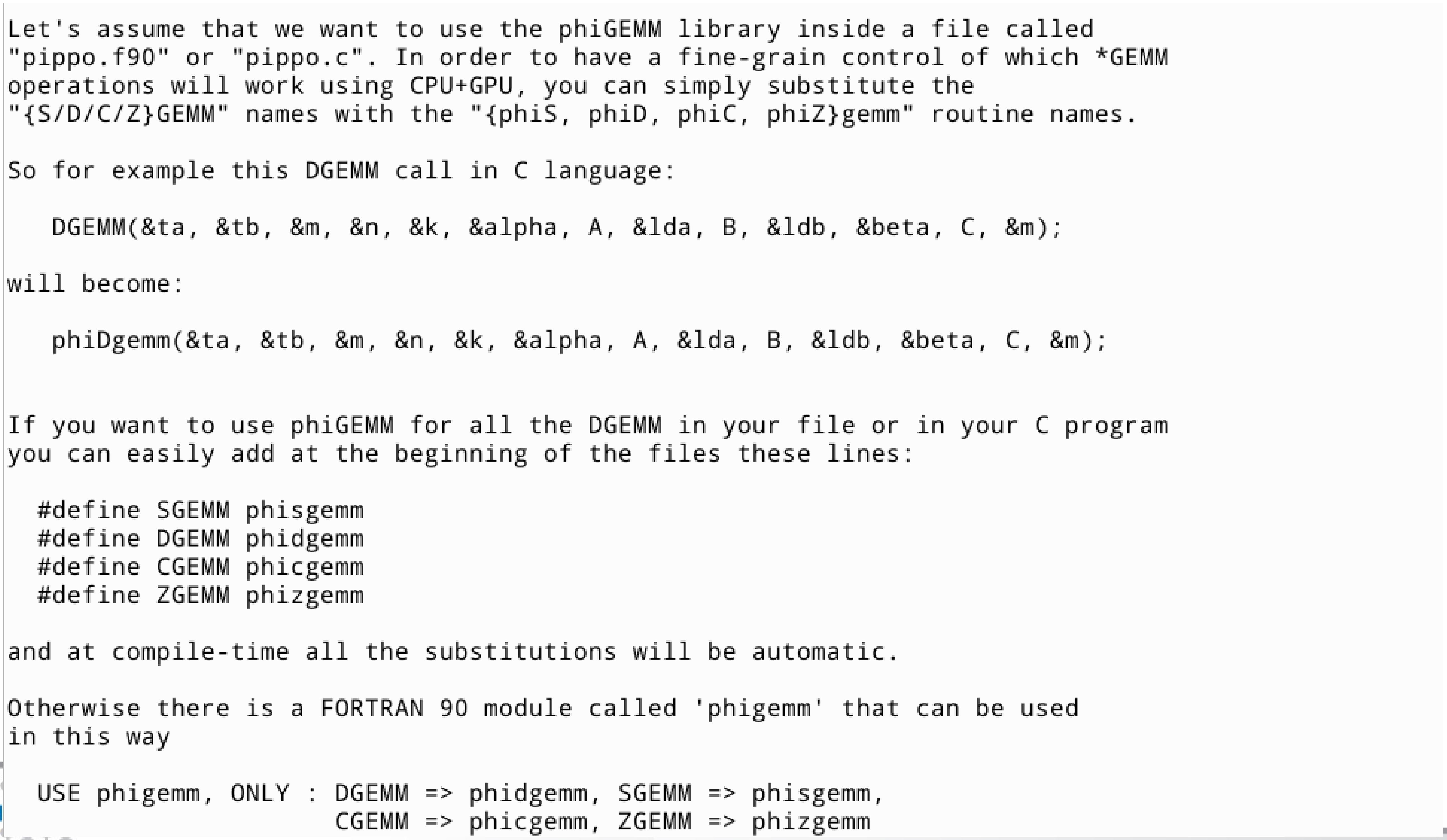
What is the best strategy?

Use a test-program to auto-tune possible splits (ATLAS style)

- matrices too small -> CPU only
- matrices small -> CPU+GPU using a pre-calculated split factor
- matrices "big enough" -> CPU+GPU using a performance-based split factor
- matrices "too rectangular" -> GPU only

## Intrusivity?

From the documentation...



```
Let's assume that we want to use the phiGEMM library inside a file called
"pippo.f90" or "pippo.c". In order to have a fine-grain control of which *GEMM
operations will work using CPU+GPU, you can simply substitute the
"{S/D/C/Z}GEMM" names with the "{phiS, phiD, phiC, phiZ}gemm" routine names.

So for example this DGEMM call in C language:

   DGEMM(&ta, &tb, &m, &n, &k, &alpha, A, &lda, B, &ldb, &beta, C, &m);

will become:

   phiDgemm(&ta, &tb, &m, &n, &k, &alpha, A, &lda, B, &ldb, &beta, C, &m);


If you want to use phiGEMM for all the DGEMM in your file or in your C program
you can easily add at the beginning of the files these lines:

  #define SGEMM phisgemm
  #define DGEMM phidgemm
  #define CGEMM phicgemm
  #define ZGEMM phizgemm

and at compile-time all the substitutions will be automatic.

Otherwise there is a FORTRAN 90 module called 'phigemm' that can be used
in this way

  USE phigemm, ONLY : DGEMM => phidgemm, SGEMM => phisgemm,
                      CGEMM => phicgemm, ZGEMM => phizgemm
```

## *xphizgemm*

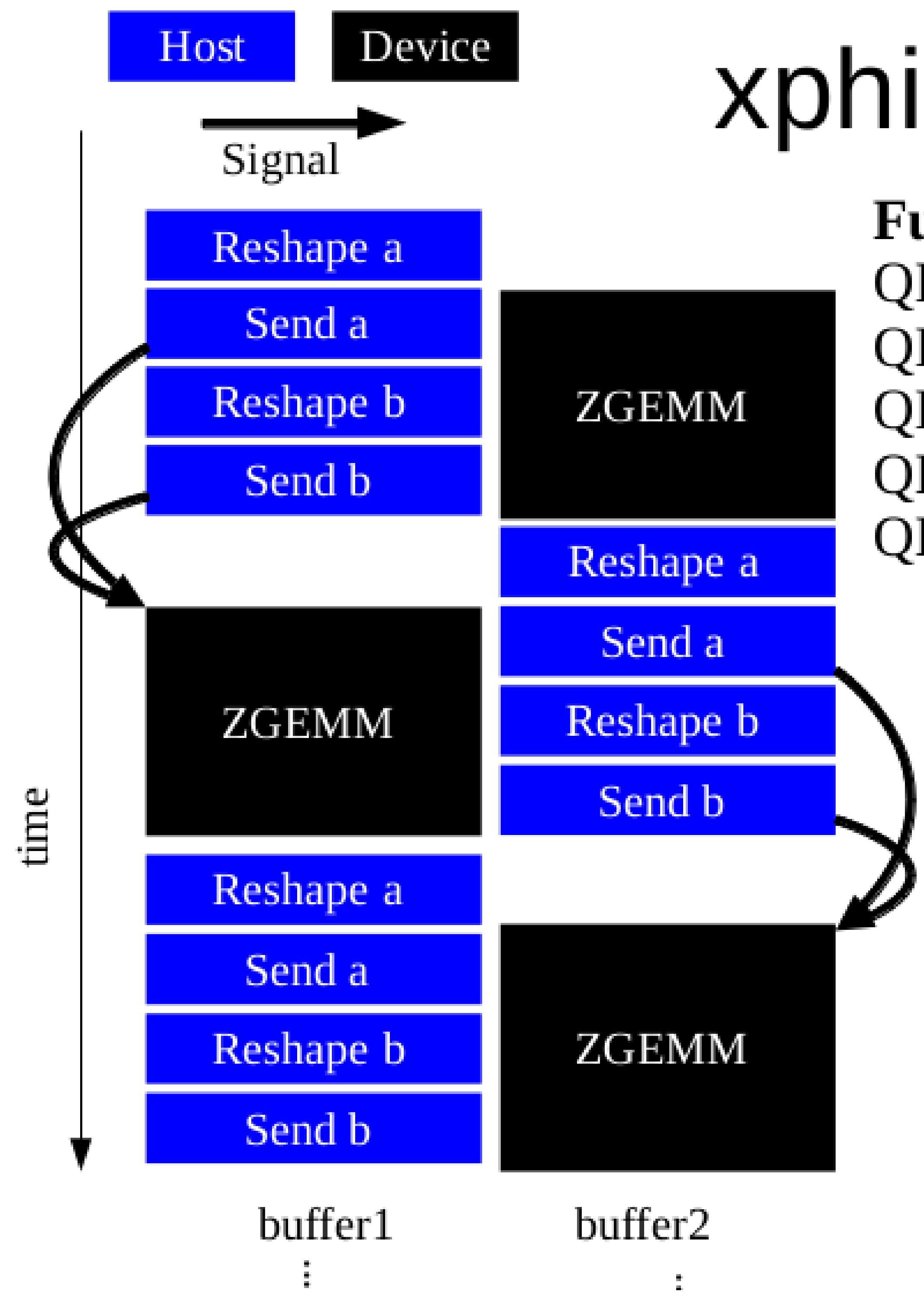
Use the offload model to run *gemm calculation on the MIC platform.

Project started in oct.2013 with the engagement of CINECA as Intel Parallel Computing Center.

**Goals:**

- same of phiGEMM
- try to get the best performance out of the MIC cards

## Double-buffer technique



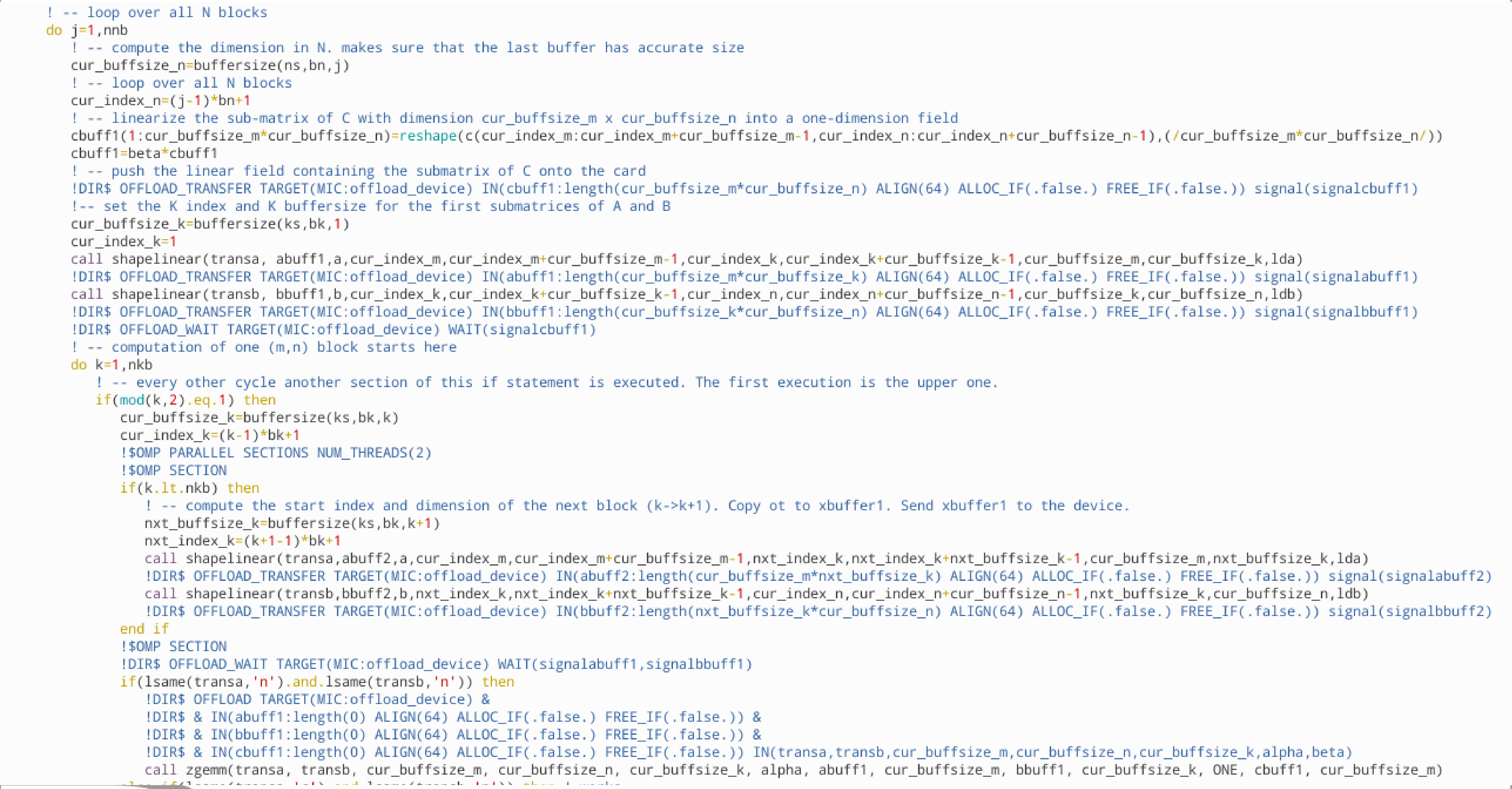
xphizgemm

**Fully configurable per MPI process**:
QE_MIC_BLOCKSIZE_M
QE_MIC_BLOCKSIZE_N
QE_MIC_BLOCKSIZE_K
QE_MIC_OFFLOAD_DEVICE
QE_MIC_OFFLOAD_THRESHOLD

Offload device:
Ensures optimal operation across systems with more than one KNC

Threshold:
ZGEMM to deliver host performance for small and KNC performance for larger matrices

# Directive based approach



```
! -- loop over all N blocks
do j=1,nnb
   ! -- compute the dimension in N. makes sure that the last buffer has accurate size
   cur_buffsize_n=buffersize(ns,bn,j)
   ! -- loop over all N blocks
   cur_index_n=(j-1)*bn+1
   ! -- linearize the sub-matrix of C with dimension cur_buffsize_m x cur_buffsize_n into a one-dimension field
   cbuff1(1:cur_buffsize_m*cur_buffsize_n)=reshape(c(cur_index_m:cur_index_m+cur_buffsize_m-1,cur_index_n:cur_index_n+cur_buffsize_n-1),(/cur_buffsize_m*cur_buffsize_n/))
   cbuff1=beta*cbuff1
   ! -- push the linear field containing the submatrix of C onto the card
   !DIR$ OFFLOAD_TRANSFER TARGET(MIC:offload_device) IN(cbuff1:length(cur_buffsize_m*cur_buffsize_n) ALIGN(64) ALLOC_IF(.false.) FREE_IF(.false.)) signal(signalcbuff1)
   !-- set the K index and K buffersize for the first submatrices of A and B
   cur_buffsize_k=buffersize(ks,bk,1)
   cur_index_k=1
   call shapelinear(transa, abuff1,a,cur_index_m,cur_index_m+cur_buffsize_m-1,cur_index_k,cur_index_k+cur_buffsize_k-1,cur_buffsize_m,cur_buffsize_k,lda)
   !DIR$ OFFLOAD_TRANSFER TARGET(MIC:offload_device) IN(abuff1:length(cur_buffsize_m*cur_buffsize_k) ALIGN(64) ALLOC_IF(.false.) FREE_IF(.false.)) signal(signalabuff1)
   call shapelinear(transb, bbuff1,b,cur_index_k,cur_index_k+cur_buffsize_k-1,cur_index_n,cur_index_n+cur_buffsize_n-1,cur_buffsize_k,cur_buffsize_n,ldb)
   !DIR$ OFFLOAD_TRANSFER TARGET(MIC:offload_device) IN(bbuff1:length(cur_buffsize_k*cur_buffsize_n) ALIGN(64) ALLOC_IF(.false.) FREE_IF(.false.)) signal(signalbbuff1)
   !DIR$ OFFLOAD_WAIT TARGET(MIC:offload_device) WAIT(signalcbuff1)
   ! -- computation of one (m,n) block starts here
   do k=1,nkb
      ! -- every other cycle another section of this if statement is executed. The first execution is the upper one.
      if(mod(k,2).eq.1) then
         cur_buffsize_k=buffersize(ks,bk,k)
         cur_index_k=(k-1)*bk+1
         !$OMP PARALLEL SECTIONS NUM_THREADS(2)
         !$OMP SECTION
         if(k.lt.nkb) then
            ! -- compute the start index and dimension of the next block (k->k+1). Copy ot to xbuffer1. Send xbuffer1 to the device.
            nxt_buffsize_k=buffersize(ks,bk,k+1)
            nxt_index_k=(k+1-1)*bk+1
            call shapelinear(transa,abuff2,a,cur_index_m,cur_index_m+cur_buffsize_m-1,nxt_index_k,nxt_index_k+nxt_buffsize_k-1,cur_buffsize_m,nxt_buffsize_k,lda)
            !DIR$ OFFLOAD_TRANSFER TARGET(MIC:offload_device) IN(abuff2:length(cur_buffsize_m*nxt_buffsize_k) ALIGN(64) ALLOC_IF(.false.) FREE_IF(.false.)) signal(signalabuff2)
            call shapelinear(transb,bbuff2,b,nxt_index_k,nxt_index_k+nxt_buffsize_k-1,cur_index_n,cur_index_n+cur_buffsize_n-1,nxt_buffsize_k,cur_buffsize_n,ldb)
            !DIR$ OFFLOAD_TRANSFER TARGET(MIC:offload_device) IN(bbuff2:length(nxt_buffsize_k*cur_buffsize_n) ALIGN(64) ALLOC_IF(.false.) FREE_IF(.false.)) signal(signalbbuff2)
         end if
         !$OMP SECTION
         !DIR$ OFFLOAD_WAIT TARGET(MIC:offload_device) WAIT(signalabuff1,signalbbuff1)
         if(lsame(transa,'n').and.lsame(transb,'n')) then
            !DIR$ OFFLOAD TARGET(MIC:offload_device) &
            !DIR$ & IN(abuff1:length(0) ALIGN(64) ALLOC_IF(.false.) FREE_IF(.false.)) &
            !DIR$ & IN(bbuff1:length(0) ALIGN(64) ALLOC_IF(.false.) FREE_IF(.false.)) &
            !DIR$ & IN(cbuff1:length(0) ALIGN(64) ALLOC_IF(.false.) FREE_IF(.false.)) IN(transa,transb,cur_buffsize_m,cur_buffsize_n,cur_buffsize_k,alpha,beta)
            call zgemm(transa, transb, cur_buffsize_m, cur_buffsize_n, cur_buffsize_k, alpha, abuff1, cur_buffsize_m, bbuff1, cur_buffsize_k, ONE, cbuff1, cur_buffsize_m)
```

## phiGEMM

computation is split between host and device, hiding the latency with computation on the host side

relying on CUBLAS

## xphizGEMM

computation is executed on the device; host is in charge of reshaping only. Latency is hidden with double-buffer technique

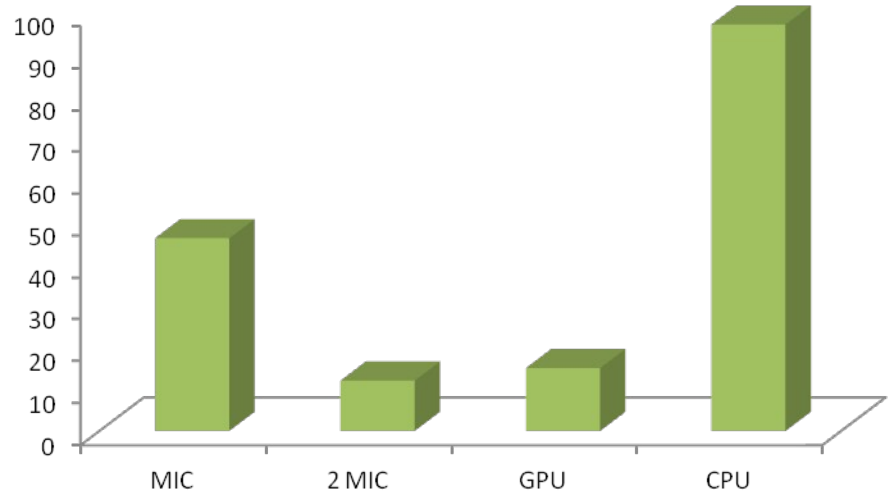
relying on MKL+OpenMP

Intrusivity and impact on the source code are very low.

# A simple benchmark: $SiO_2$

109 atoms

| DEVICE(S) | MIC | 2 MIC | GPU | CPU |
|---|---|---|---|---|
| MPI | 1 | 8 | 1 | 1 |
| THREADS | 8 | 2 | 8 | 8 |
| WTIME (min) | 46 | 12 | 15 | 97 |



Development on Intel Xeon Phi has just recently started…

… there is a long long road ahead

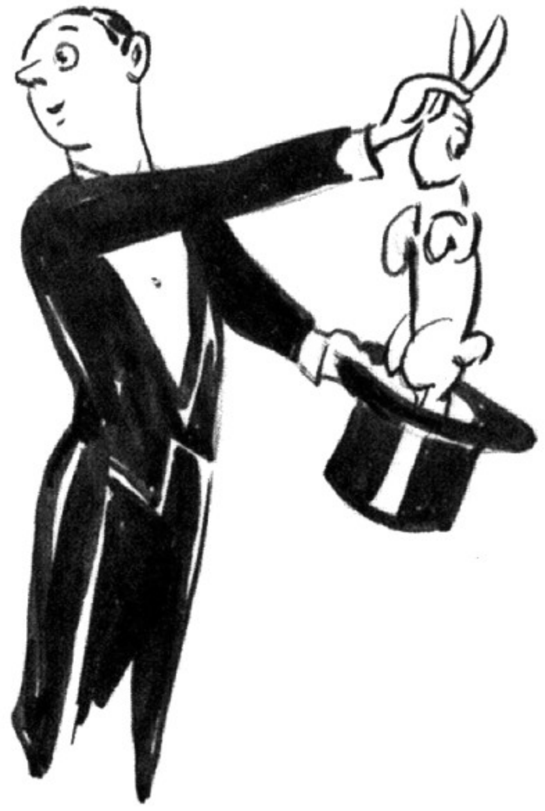
Open question: what can MICs learn from GPUs and viceversa?

It seems that GPUs and MICs are like two different cars that should run on the same path...

**BUT....**

... we still miss a **common** (consolidated) approach to drive...

Need for a **common** programming model???



?