Introduction to the Xeon Phi programming model

Fabio AFFINITO, CINECA

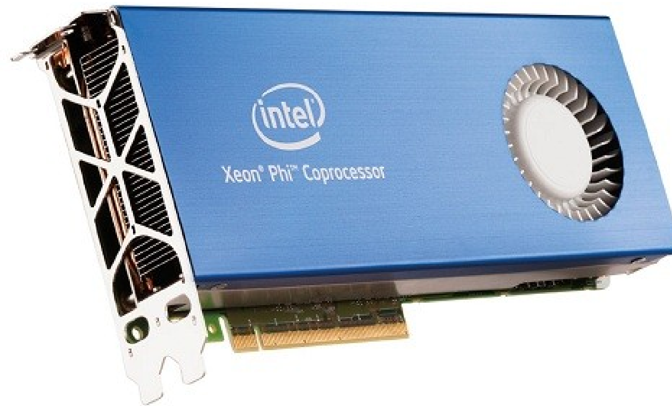
What is a Xeon Phi?

- MIC = Many Integrated Core architecture by Intel
- Other names: KNF, KNC, Xeon Phi...
- Not a CPU (but somewhat similar to the A2 chip..)
- Not an accelerator (but with the philosophy of a GPU...

.... It’s a “coprocessor”

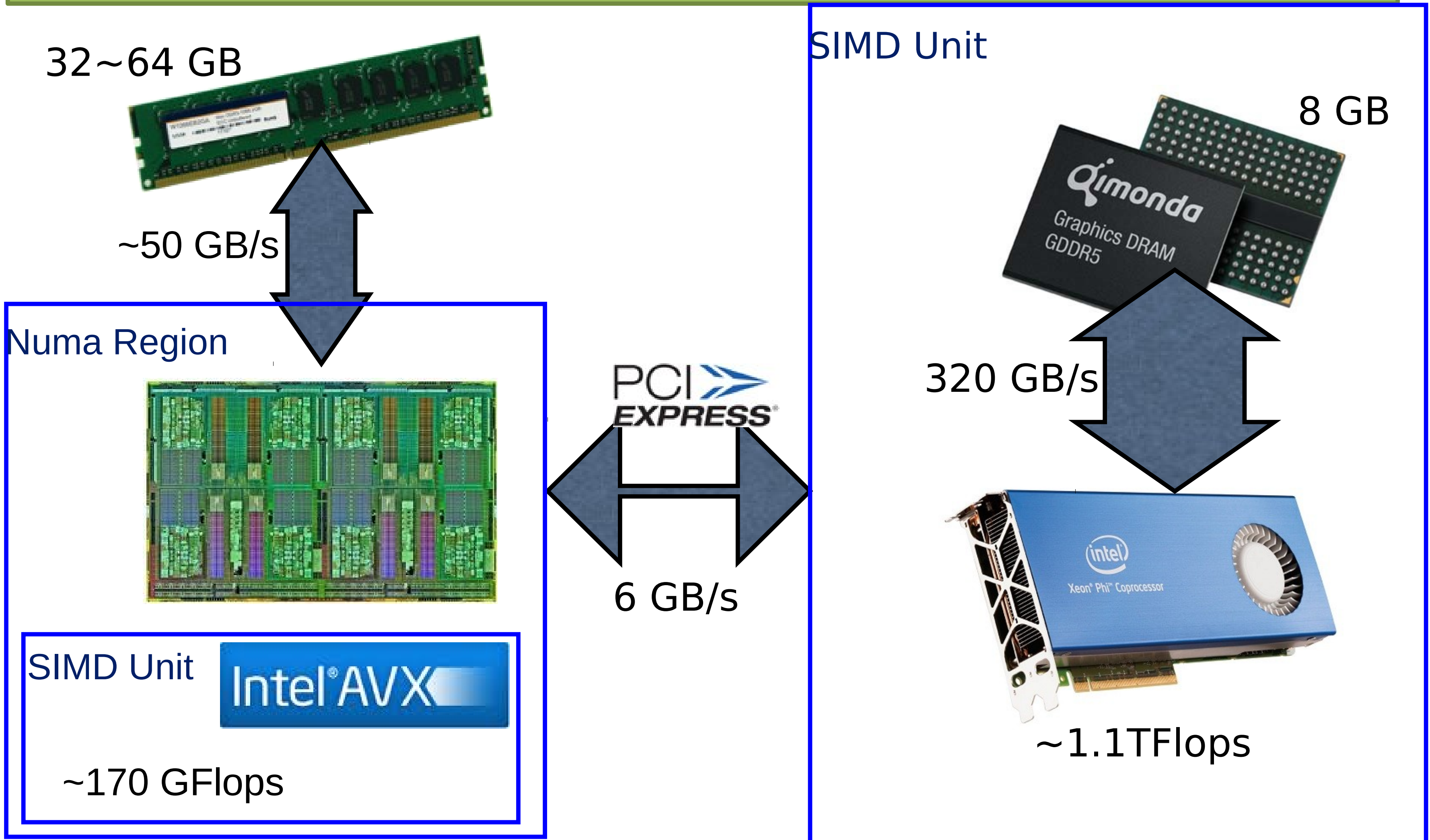
What is a Xeon Phi?

- It’s a coprocessor
  - Based on a x86 architecture (Pentium III)
  - Designed to reach the 1-Tflop/s peak power



“Now you can think reuse rather than recode with x86 compatibility” (Intel website)

What is a Xeon Phi?



Xeon processor vs coprocessor

- The Xeon Phi has:
  - Very large number of cores
  - Large bw to the memory
  - Small memory
- The Xeon processor has:
  - Higher clock frequency
  - Larger memory
  - Narrow bw to the memory

In any case, they are very far... PCIe 6GB/s

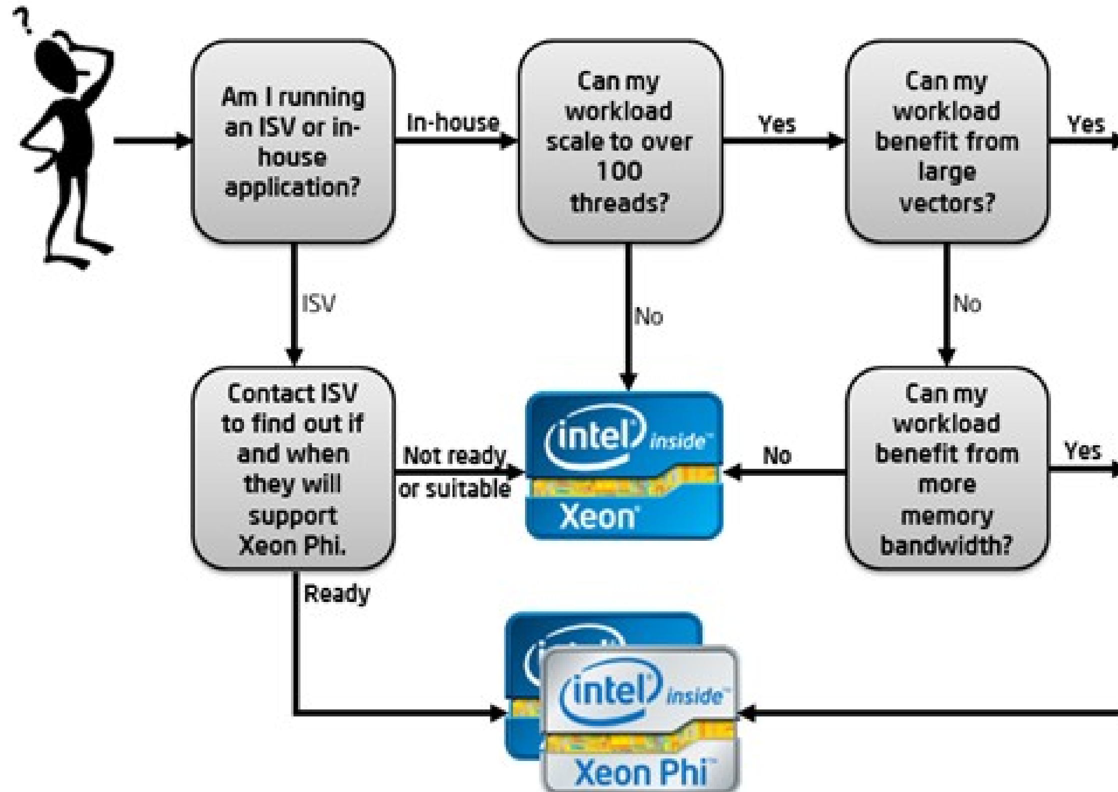
Xeon Phi vs GPU

- Similarities
  - They require a host
  - They communicate through a PCIe
  - They need “very parallel code”
  - They allow to offload part of code
- Differences
  - Xeon Phi is a x86 architecture
  - Xeon Phi has a fully coherent cache
  - Xeon Phi supports OpenMP multithreading

All the truth about Xeon Phi

- Technicalities
  - 61 x86 based cores, but not fully x86 compatible (you need to cross compilation)
  - Coherent cache but evidences show cache behavior similar to GPU
  - FPU is powerful (32 8-fold DP registers, FMA), nice masked instructions but some serious limitations (permutations, broadcast in the 256-bit lanes only…)
  - You can run in native mode, but performance can be limited by:
  - Memory
  - Amdahl law
  - FPU is powerful (32 8-fold DP registers, FMA), nice masked instructions but some serious limitations (permutations, broadcast in the 256-bit lanes only…)
- Bottom line: porting is (almost) for free. Getting performance is not.

Is the Xeon Phi right for you?



Xeon Phi programming models

- There are three options:
  - Symmetric: using both the MIC and the host in an independent way, communicating through MPI
  - Native: running directly code compiled for MIC, using the coprocessor as a many-core SMP node
  - Offload: running on the host and offloading high parallel kernels on the MIC, using pragmas or through accelerated libraries (for example MKL AO)

- Click to edit Master text styles
  - Second level
    - Third level
      - Fourth level
        - Fifth level

▪All products, computer systems, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Xeon®-Centric

MIC-Centric

Xeon-hosted

Offload

Symmetric

Reverse Offload

MIC-hosted

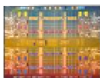
General purpose serial and parallel computing

Codes with highly parallel phases

Codes with balanced needs

Highly parallel codes with scalar phases

Highly parallel codes

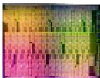


Main( )
Foo( )
MPI_*()

Main( )
Foo( )
MPI_*()

Main( )
Foo( )
MPI_*()

Foo( )



Foo( )

Main( )
Foo( )
MPI_*()

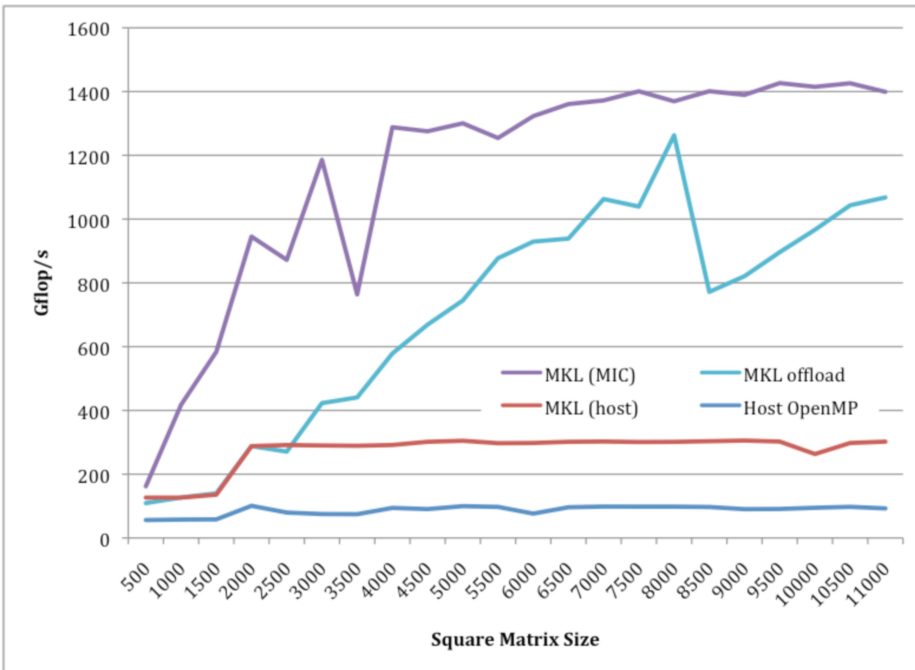
Main( )
Foo( )
MPI_*()

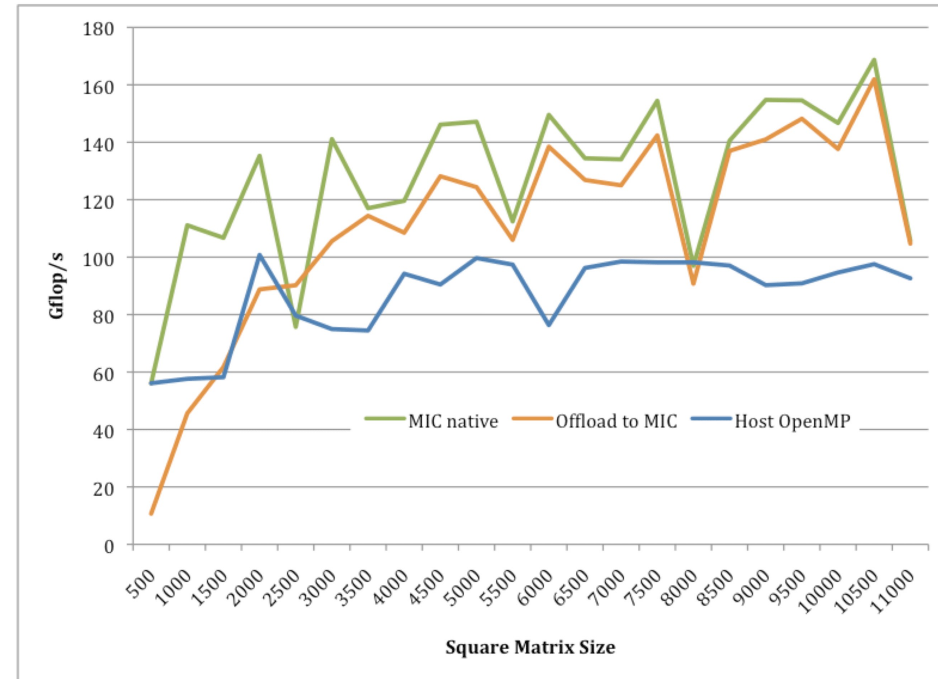
Main()
Foo( )
MPI_*()

Comparing execution modes

- Use matrix-matrix multiplication as a test case
  - OpenMP sgemm on HOST
  - OpenMP sgemm on MIC native
  - Offload mode

Comparing execution modes





Test-case: offloading dgemm

```
__declspec(target(mic))
void gemm(char transa, char transb,
        int M, int N, int K,
        double alpha, double* A, int lda,
        double* B, int ldb, double beta,
        double* C, int ldc)
{
      DGEMM(&transa, &transb, &M, &N, &K,
            &alpha, A, &lda, B, &ldb, &beta, C, &ldc);
}
```

```
#pragma offload target(mic)
        in(A[0:M*K]:align(Align))
        in(B[0:K*N]:align(Align))
        inout(Cg[0:M*N]:align(Align))
        inout(otime)
{
   otime -= mysecond();
   gemm(transa, transb, M, N, K,
        alpha, A, lda, B, ldb,
        beta, Cg, ldc);
   otime += mysecond();
}
```

Benchmarking offload...

- 60 cores (+1) x 4 threads per core...

```
$ export OMP_NUM_THREADS=240
```

```
$ ./dgemm_offload

MNK = 10000    10000    10000    Gflops Max: 285.3
```

Let's calculate the expected performance ...

peak = (# cores)*(vector size)*(ops/cycle)*(frequency)
peak = 60*8*2*1.052 = 1011 Gflop/s

285 << 1011 Gflop/s

Affinity

- In many cores architectures affinity and core-binding are crucial. Setting the KMP_AFFINITY

```
$ export MIC_ENV_PREFIX=MIC; export MIC_KMP_AFFINITY=scatter; ./dgemm_offload

MNK= 10000   10000      10000      Gflops Max:374.741736
```

```
$ export MIC_KMP_AFFINITY=compact; ./dgemm_offload

MNK= 10000   10000      10000      Gflops Max:641.1
```

```
$ export MIC_KMP_AFFINITY=balanced; ./dgemm_offload

MNK= 10000 10000      10000      Gflops Max:641.63
```

We are still very far from the expected performance….

This behavior holds for DGEMM algorithm… Maybe not for others (for FFT “scatter” is best choice…)

Alignment

- As a general rule, we need to align arrays to the vector size. For example, 16-byte alignement for SSE processors, 32-byte alignement for AVX processors, 64-byte for Xeon Phi. Unfortunately, in offload mode the compiler will consider that arrays alignment is the same on both the host and the device. We need to change the alignement on the host to match

```
double *A = (double*)_mm_malloc(sizeof(double)*size_A,Alignment);
```

Alignment = 16

```
$./t3_offload
M N K = 10000 10000 10000 Gflops Max: 641.63
```

Alignment = 64

```
$ ./t3_offload
M N K = 10000 10000 10000 Gflops Max: 724.22
```

285 -> 641 -> 724 … Can we do something more?

Huge pages

- We recall that Xeon Phi has a large bandwith access to memory. Huge pages then might help.

```
$ export MIC_USE_2MB_BUFFERS=100M
```

- This means that for any array allocation larger than 100MB, uses huge pages

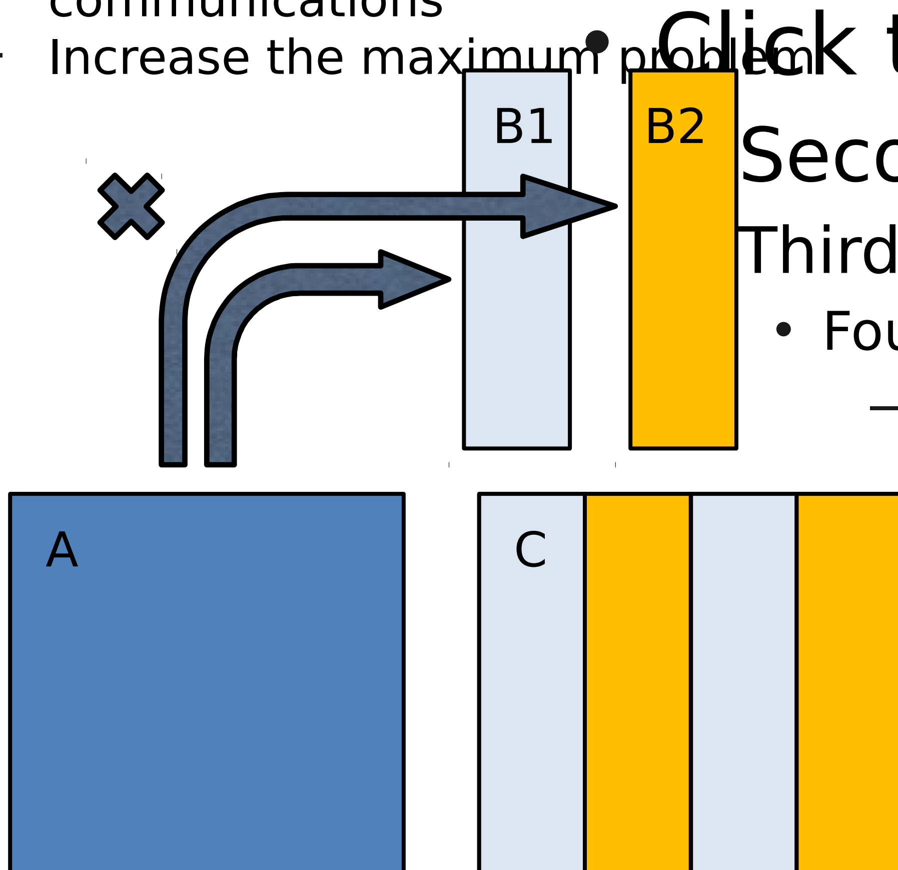
```
$ ./t3_offload

PEs = 1, M N K = 10000 10000 10000 Gflops Max: 806.9
```

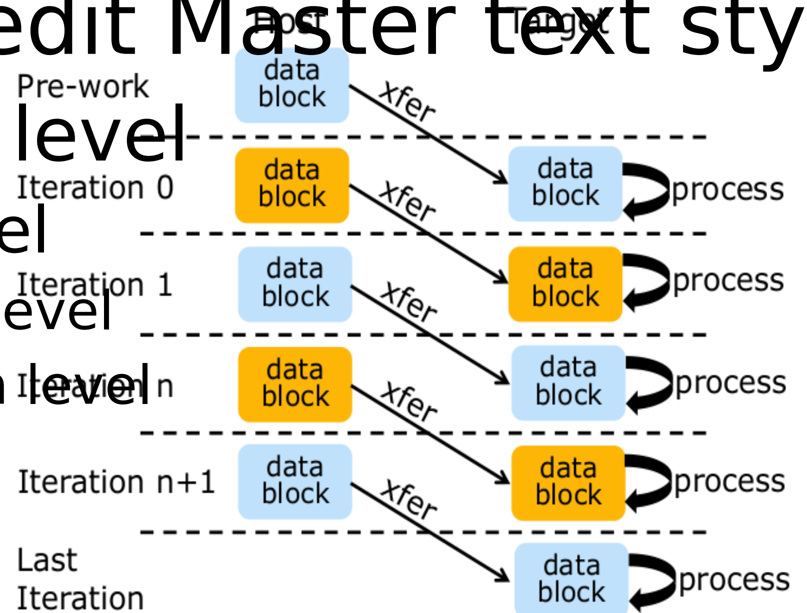
80% of the peak performance

Overlapping offload

Double buffering:
- Overlap computations and communications
- Increase the maximum problem

- Click to edit Master text styles
  - Second level
    - Third level
      - Fourth level
        - Fifth level





Overlapping offload

- Asynchronous movement of data
- Hide the bottleneck of data transfer with computation
- Exploits the data persistence on MIC
- It permits to work on larger matrices

Normal dgemm, maximum size = 14800x14800:

NBLOCKS	Gflops w/o data transfer	Gflops w/ data transfer
1	750	334

Split dgemm, size = 20000x20000:

NBLOCKS	Gflops w/o data transfer	Gflops w/ data transfer
4	730	218
8	675	244
16	649	300
25	634	289
32	614	297

Split dgemm, size = 26624x26624:

NBLOCKS	Gflops w/o data transfer	Gflops w/ data transfer
64	750	334

Courtesy of CSCS

Xeon Phi in the world

EURORA
@CINECA

STAMPEDE
@TACC

Xeon Phi Online

The Intel Software development website has a guide for developing software on Xeon Phi (KNC). The guide has a very good summary of vectorization techniques with the Intel compiler that is equally valid for Sandy Bridge:

http://software.intel.com/mic-developer

- Intel have only recently publically unveiled Xeon Phi, and the first commercially available cards are being delivered. You can find more information about developing for Xeon Phi as it comes available on the Intel developer site:

Online Documentation: http://software.intel.com/en-us/articles/programming-and-compiling-for-intel-many-integrated-core-architecture