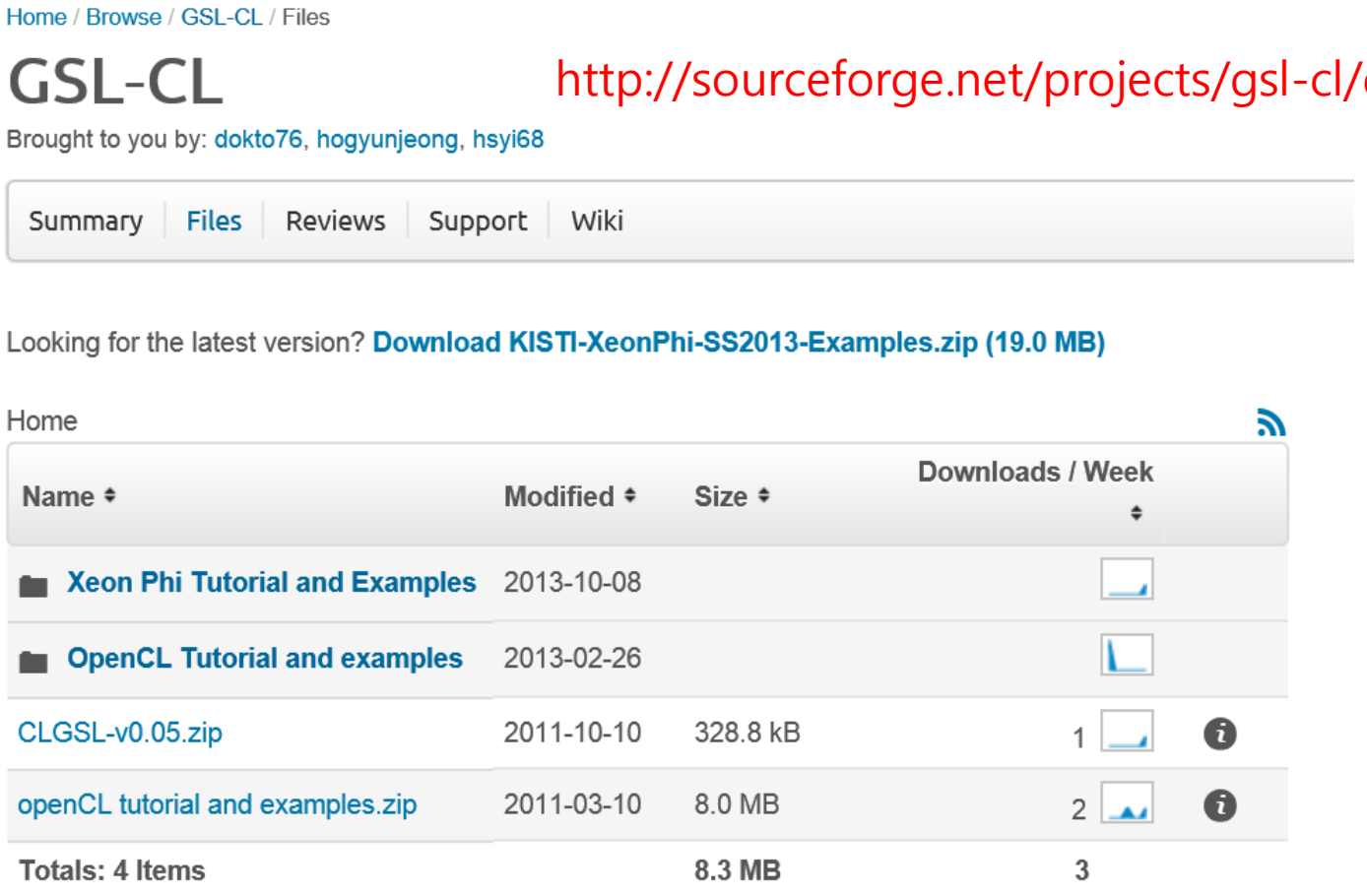
Many-core Processor Programming for beginners

Hongsuk Yi (李泓錫)
(hsyi@kisti.re.kr)
KISTI
(Korea Institute of Science and Technology Information)

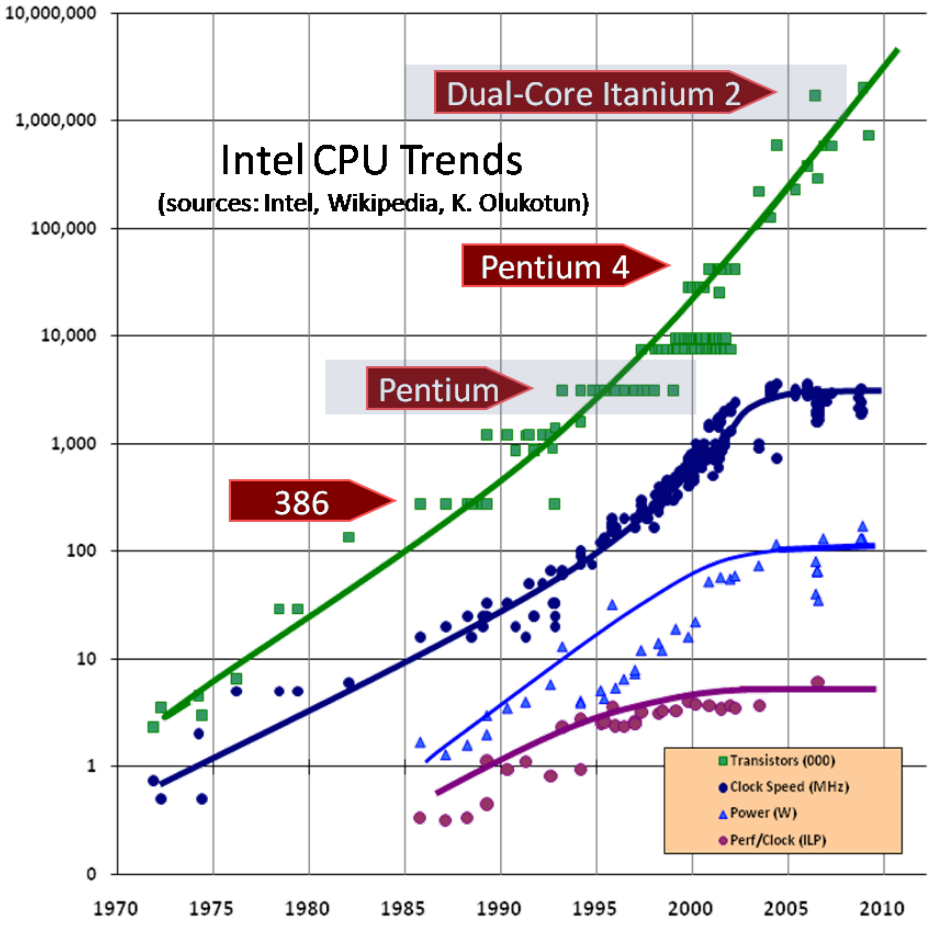
Contents

- Overview of the Heterogeneous Computing
- Introduction to Intel Xeon Phi Coprocessor
- Parallel Programming methods on Intel Xeon Phi
  - Native
  - Offload
  - Vectorization, OpenMP, MPI, OpenCL
- Summary

You can download the course materials



The free lunch is over



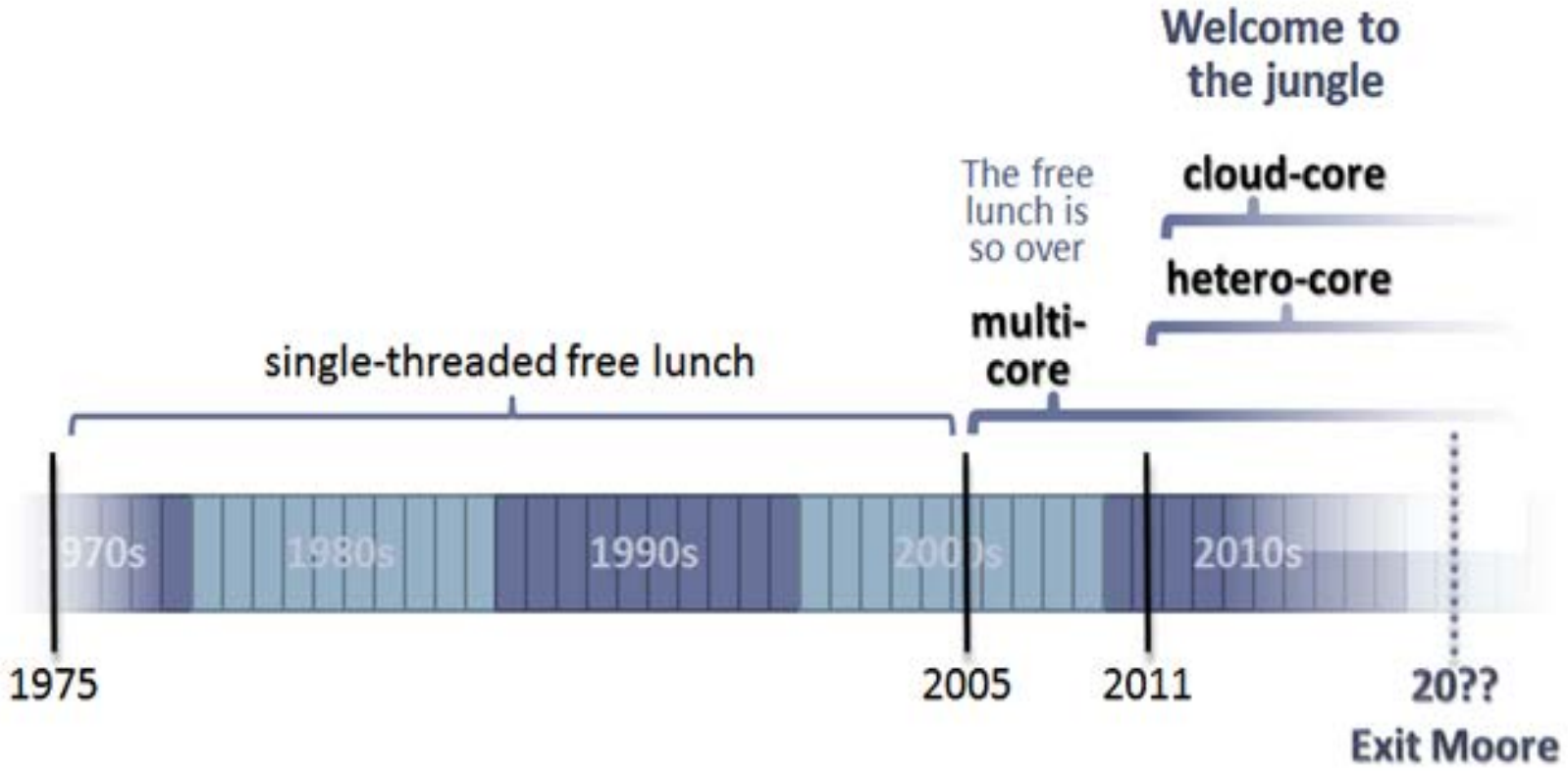
Intel CPU trends
Sources: Intel, Wiki

1965, G. Moore claimed #transiters on microchip doubles every 18 months

The near-term future performance grows in Hyperthreading Multicore

Over the past 30 years, CPU designers have achieved performance gains from Clock speed, execution optimization and cache.

Welcome to the heterogeneous computing world !!



Ref: http://herbsutter.com/welcome-to-the-jungle/

#1 Heterogeneous Supercomputer

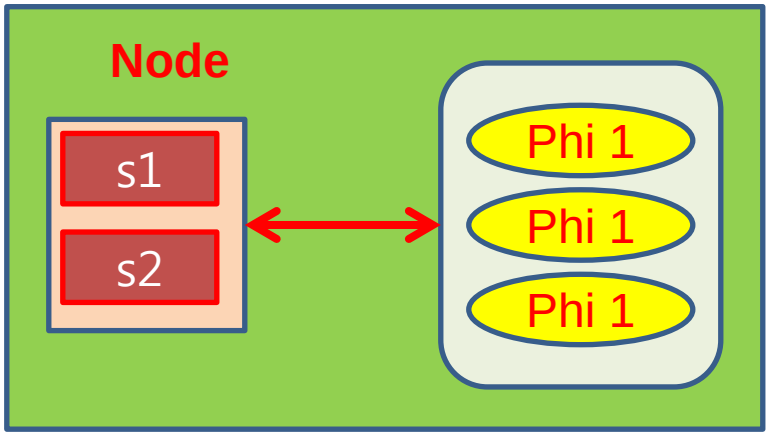
Tianhe-1 : The 5th 2009
- 0.56 Pflops
- 1024 x Xeon(E5540) + 5120 AMD GPU

Tianhe-1A : The 1st 2010
- 2.566 Pflops
- 14366 Xeon
- 7168 NVIDIA GPU

➢ Tianhe-2 (31 Pflops)

✓Tianhe-2 has been measured at speeds of 31 Pflops

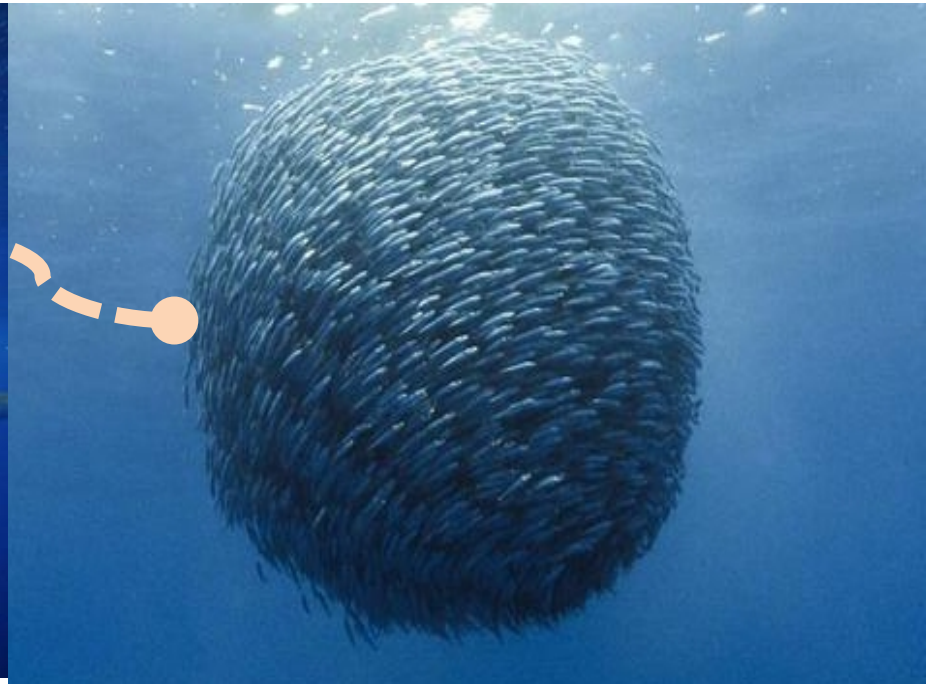
- built with Intel Ivy Bridge and Xeon Phi processors.
- There are 32,000 Intel Ivy Bridge Xeon sockets and 48,000 Xeon Phi boards for a total of 3,120,000 cores



What is Heterogeneous Computing?

Multi-cores

Many-cores, coprocessors, Accelerators

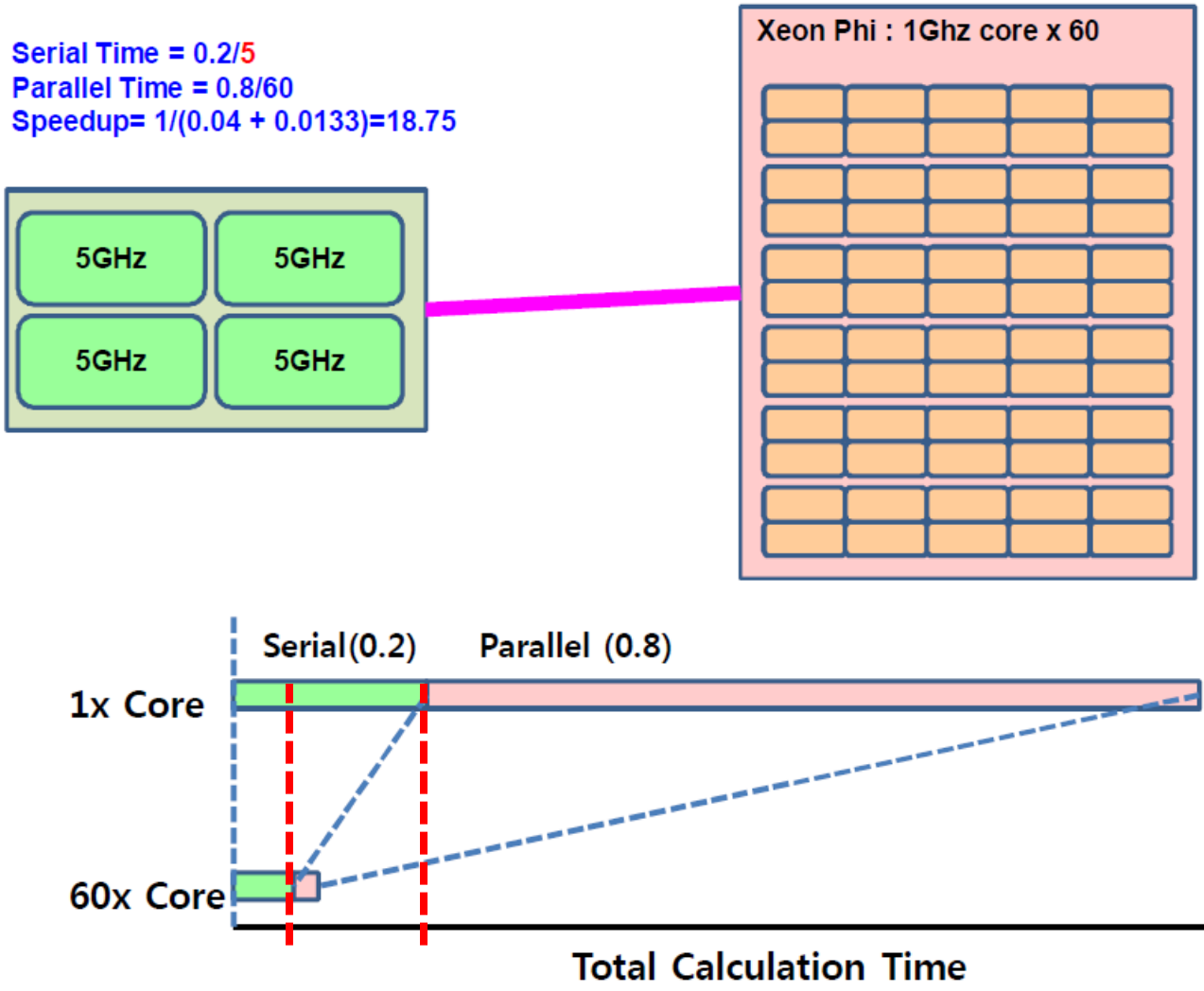


GPU
FPGA
MIC
DSP

Heterogeneous Programming Model

OpenACC
OpenHMPP
CUDA
OpenCL
OpenMP 4.0
SIMD
Offloading

Benefits of Heterogeneous Computing



Speed-up=1

Speed-up=5

Speed-up=19

Introduction to Intel Xeon Phi Coprocessor

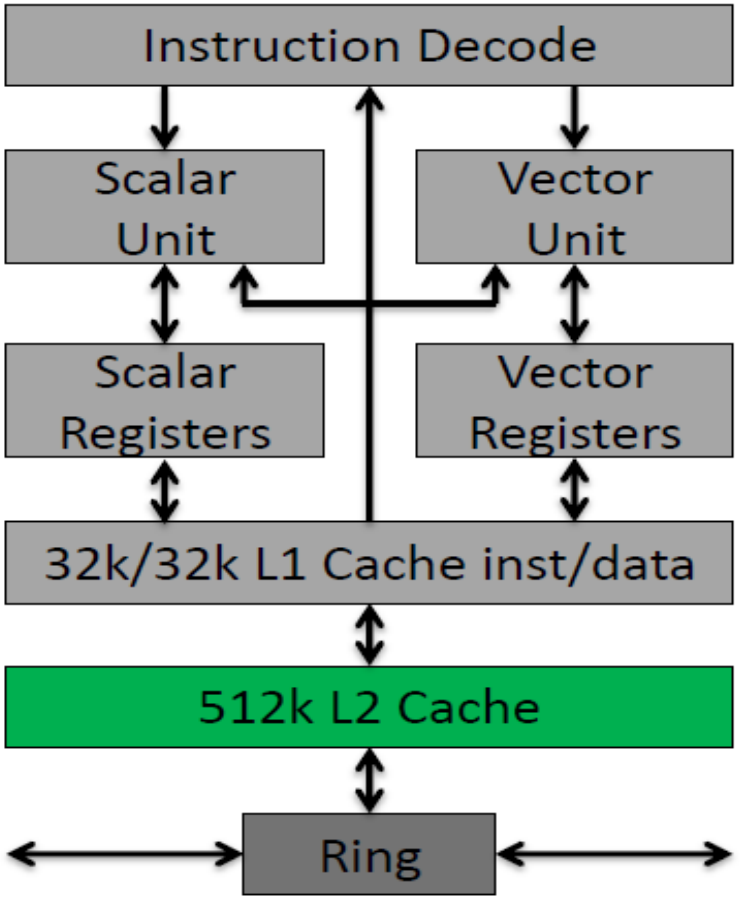
Chapter II

Xeon Phi : What is it?

- Lots of names
  - You might have heard it referred to as MIC (Many Integrated Core of KNF/KNC (Knights Cornor)
    - It is a recently released accelerator based on x86 architecture (P3)
    - Knights Corner (code name)
    - Xeon Phi Coprocessor (product name) is designed as a 1-Tflops
  - Different models have number designations
    - 5100P, 3120A, 7120 (16 GB in memory)
- Now you can think reuse rather than recode or rewrite with x86 compatibility
- Really?

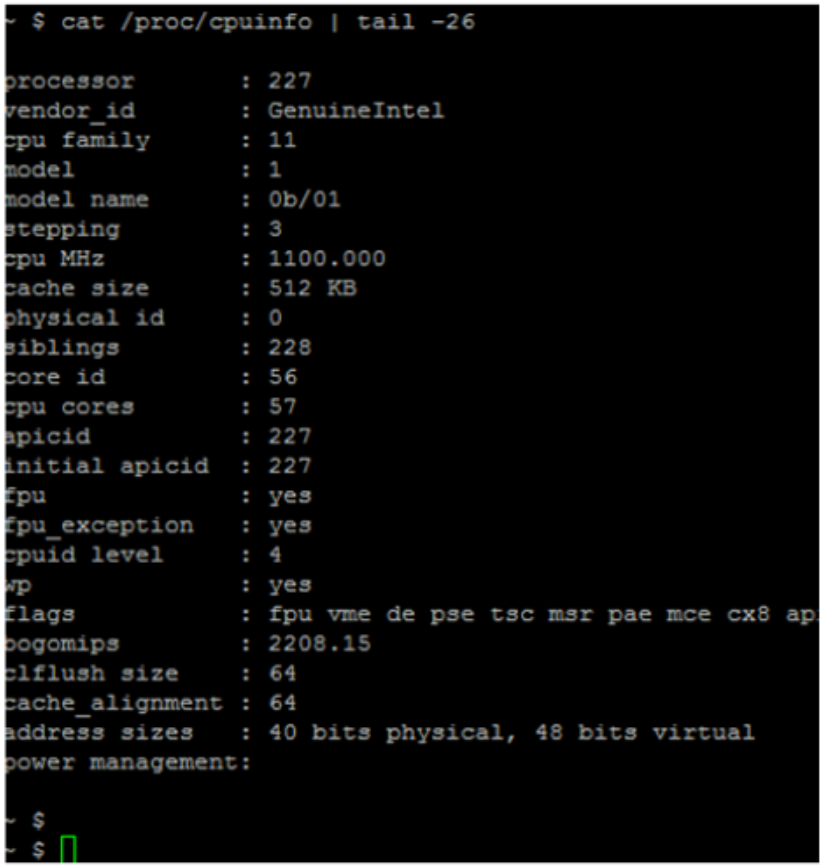
What Xeon Phi is ?

- Technical specifications:
  - Up to 61 P1 cores
  - 4-way hyper-threading
  - 16GB memory of GDDR5
  - 512-bit SIMD instructions
  - IP addressable
  - PCIe bus connection
  - Lightweight Linux OS
    - It's just Linux



Let's go "Hands-on" cat /proc/cpuinfo

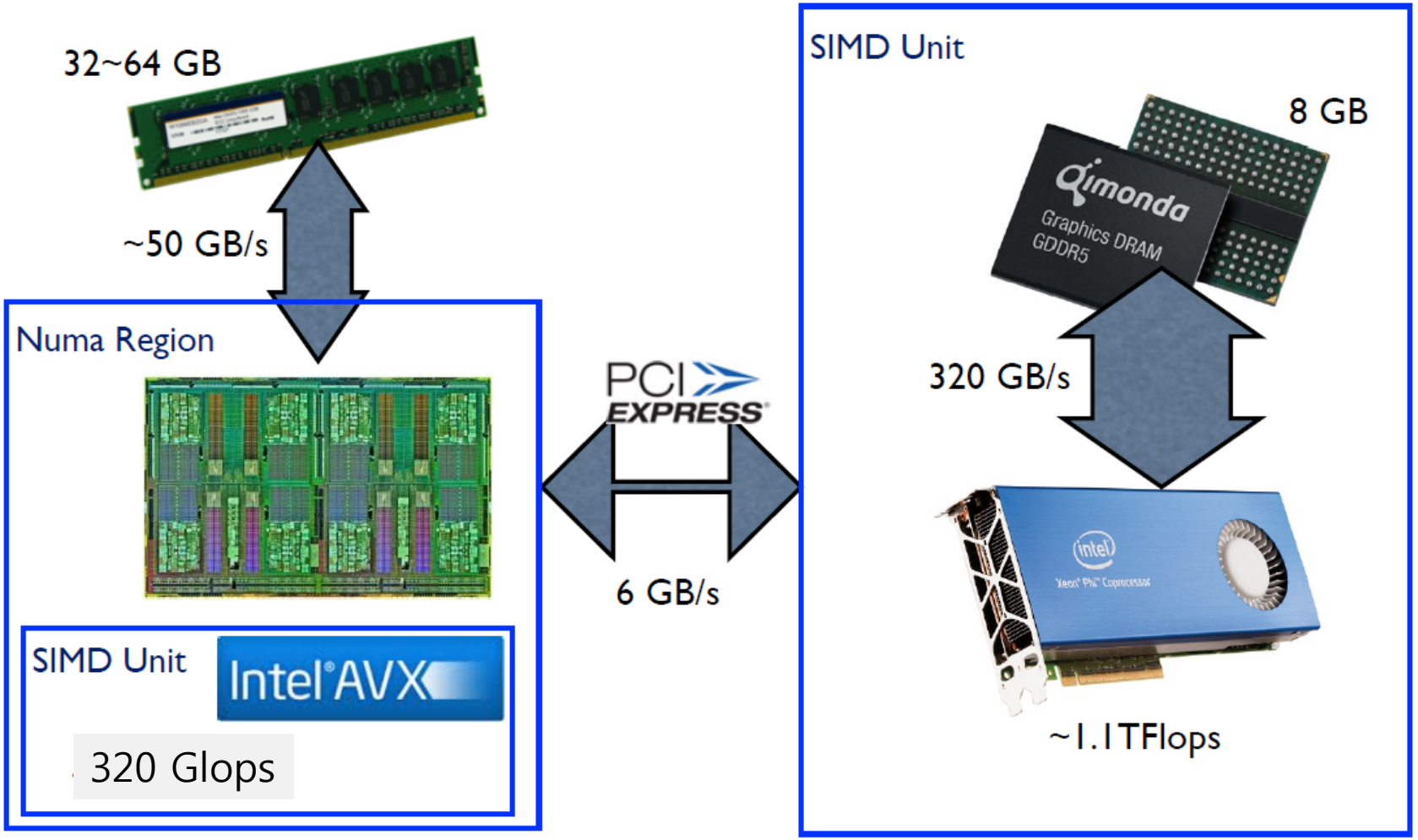
```
$ ssh mic0
$ cat /proc/cpuinfo | tail -26

processor       : 243
vendor_id       : GenuineIntel
cpu family      : 11
model           : 1
model name      : 0b/01
stepping        : 1
cpu MHz         : 1090.908
cache size      : 512 KB
physical id     : 0
siblings        : 244
core id         : 60
cpu cores       : 61
apicid          : 243
initial apicid  : 243
fpu             :   yes
fpu_exception   : yes
cpuid level     : 4
wp              :   yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic mtrr mca
                  pat fxsr ht syscall lm rep_good nopl lahf_lm
bogomips        : 2190.07
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:
$
```



What Xeon Phi is Not ?

- Technical specifications:
  - 60 x86 based cores but is not x86 compatible
    - Cross-compilation needed
  - FPU is powerful (FMA), nice masked instructions but some serious limitations (permutations..)
  - You can run in native mode, but performance is limited
    - Memory size
    - Single core performance is REALLY slow
  - Xeon standard tools (MPI, OpenMP, OpenCL, Intrinsics) are available but require specific code to match the architecture (SMT, FPU,..)

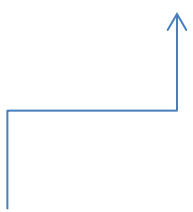
Xeon Phi Node



There is no single driving force

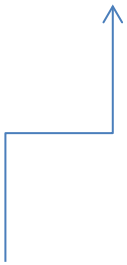
Source : George Hager
http://ircc.fiu.edu/sc13/PractitionersCookbook_Slides.pdf

Perf. = nCores * SIMD * Freq * FMA

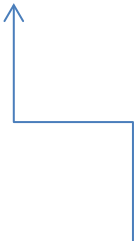


Parallelization
- OpenMP, Cilk

Heterogeneous
- MPI, OpenCL



Vectorization
- SIMD

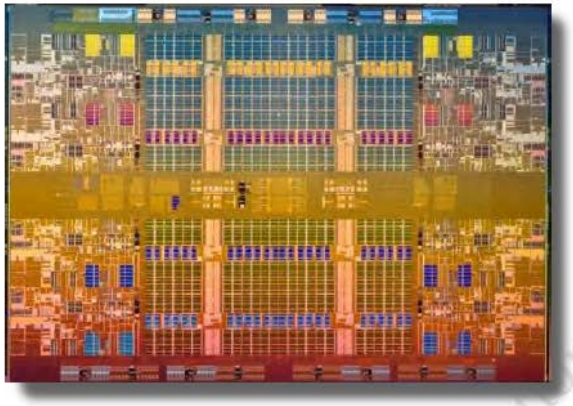


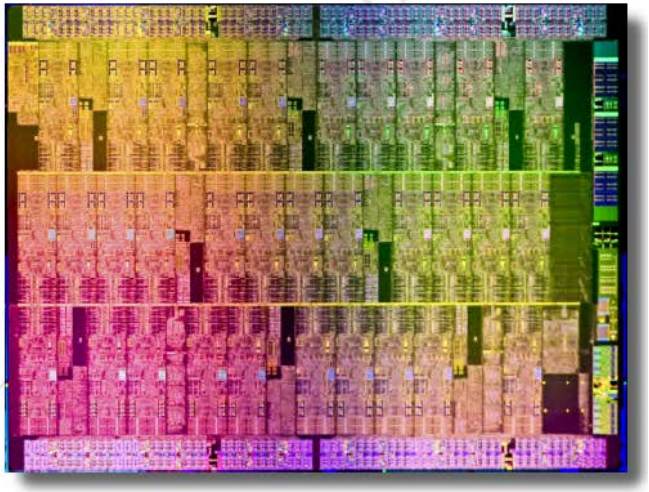
Optimization
- Prefetch, Loop
- Cache, Latency

HC here is here to stay : SIMD + OpenMP + MPI + OpenCL/CUDA

A Tale of Two Architectures

Xeon Processor





Xeon Phi Coprocessor

	Xeon Phi		Sandy bridge
nCore	57	61	16
SIMD (DP)	8	8	4
Freq	1.1	1.09	2.59
Execution Style	In-order		Out-of-order
Gflops	1003.2	1063.8	331.5
Bandwidth (GB/s)	320.0	350	102

Xeon Phi database

Model	Cores	Clock (GHz)	L2 Cache(MB)	GDDR5 (GHz)	Memory (GB)	Interface (bit)	Perf. Gflops	TDP (Watt)
SE10P/X	61	1.10	30.5	5.5	8	512	1073	300
5110P	60	1.05	30	5.0	8	512	1011	225
5120D	60	1.05	30	5.5	8	512	1011	245
7120P/X	61	1.25	30.5	5.5	8	512	1220	300
3120A/P	57	1.10	28.5	5.0	6	384	1003	300

Active Cooling (3210 A) – KISTI EDU LAB

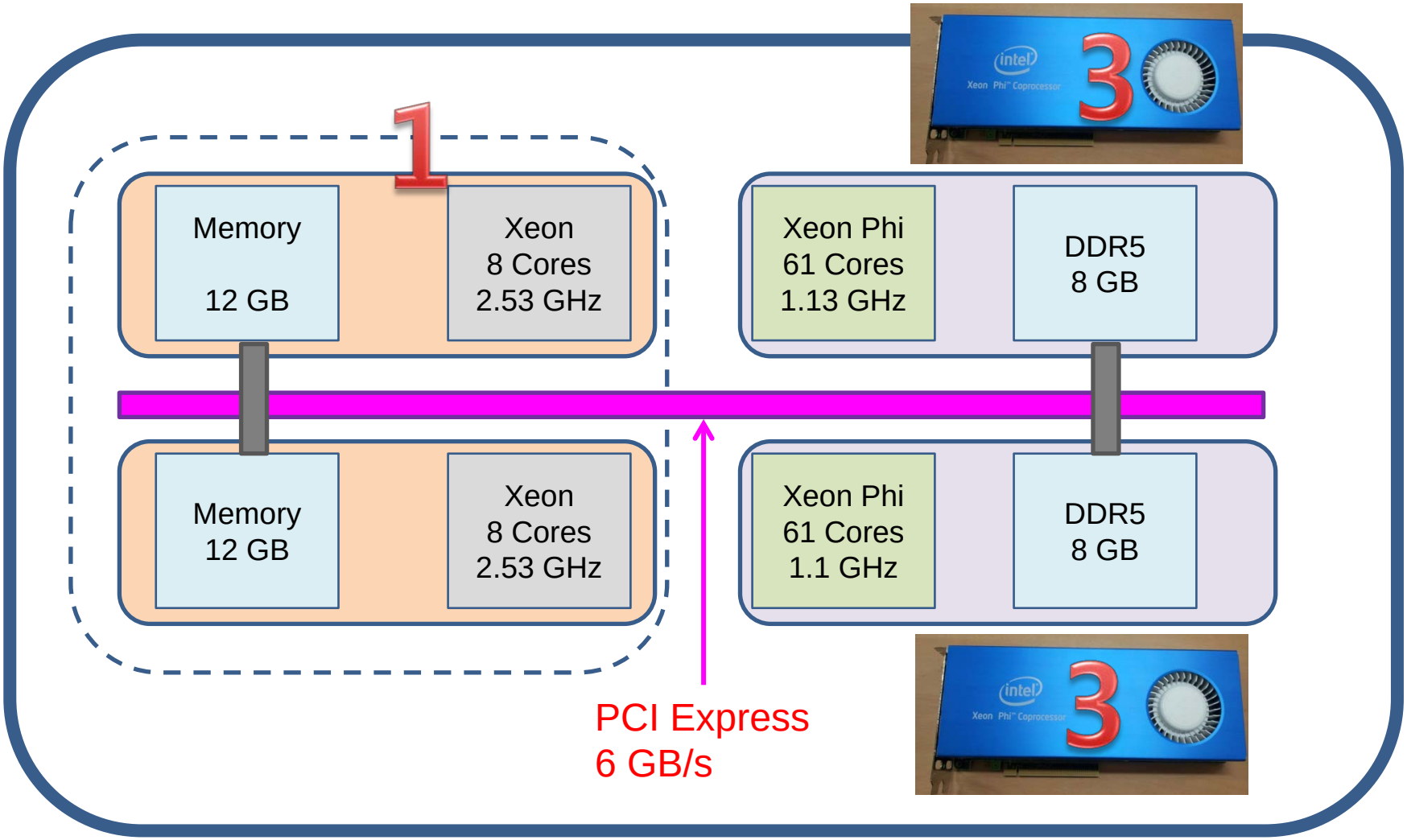
Any model with the extension "*P" in the name has the passively cooled cooler, While others have the active drum cooler.

CPU/GPU/MIC

	Number of Cores	Performance DP(GF)	Bandwidth (GB/s)	TDP (Watts)	Price (US $)
NVIDIA C2050	448	515.2	144	248	1,350
NVIDIA K20	2496	1170	208	225	3,100
NVIDIA K20X	2688	1310	250	235	4,500
AMD HD7970	2048	947	264	210	600
Intel Phi 5110P	61	1010	320	225	2700
Intel SB E5-2680	16	331.5	80	250	10,000

TDP : Thermal Design Power

Computing Node



Performance of CPU vs Phi

- 2 Sockets (8 cores) => 3 times
- 1 Sockets => 2 x 3 =6
- Only Use 1 Core => 8*6= 48
- No SIMD (but average) = 3(1~8) * 48 = 144
- No Optimization (O0) = 4 (1~10) * 144 = 576
  - Without Intel compiler, O3

Coprocessor Only (“native”)

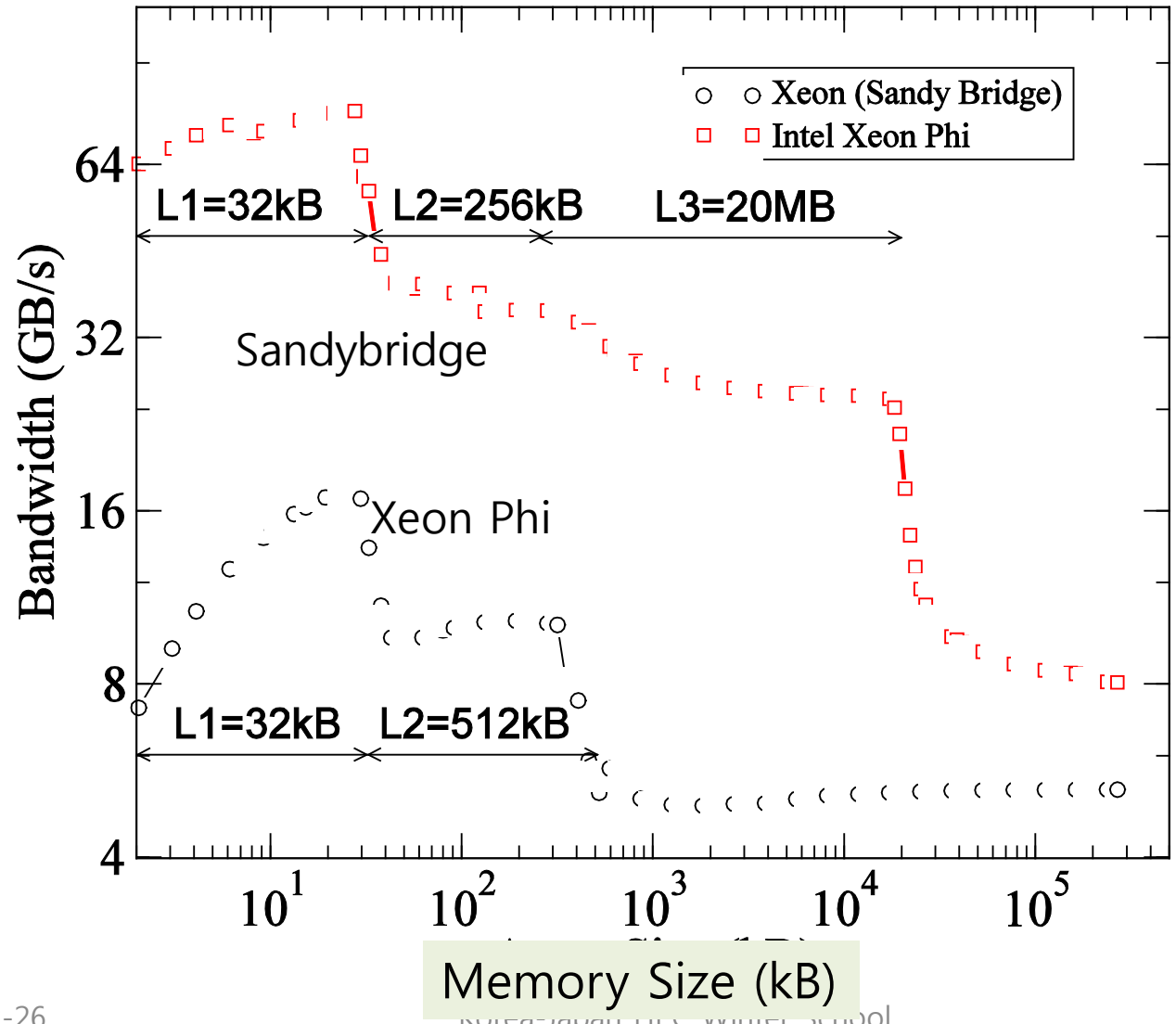
- Cross-compile for the coprocessor
  - easy (just add “-mmic”, login with ssh and execute)
  - OpenMP, posix threads, OpenCL, MPI usable
  - 240 Hardware threads on single chip
- However, in-order cores
  - Very small memory (8 GB), small caches (only 2 levels)
  - Limited hardware prefetching
  - Poor single thread performance ~ 1GHz
  - Host CPU will be bored
  - Only suitable for highly parallelized / scalable codes

SAXPY in native mode

```
void saxpyCPU(int n, float a, float *x, float *y) {
   for (int i = 0; i < n; ++i)
   y[i] = a*x[i] + y[i];
}

int main(int argc, const char* argv[]) {
   int n = 10240; float a = 2.0f;
   float* x; float* y;
   x = (float*) malloc(n * sizeof(float));
   y = (float*) malloc(n * sizeof(float));

for(int i=0; i<n; ++i){        x[i]=i;  y[i]=2.0*I + 1.0;    }

saxpyCPU(n, a, x, y);
   free(x); free(y);
}
```

```
$icc –mmic –o a.mic saxpy.c
```

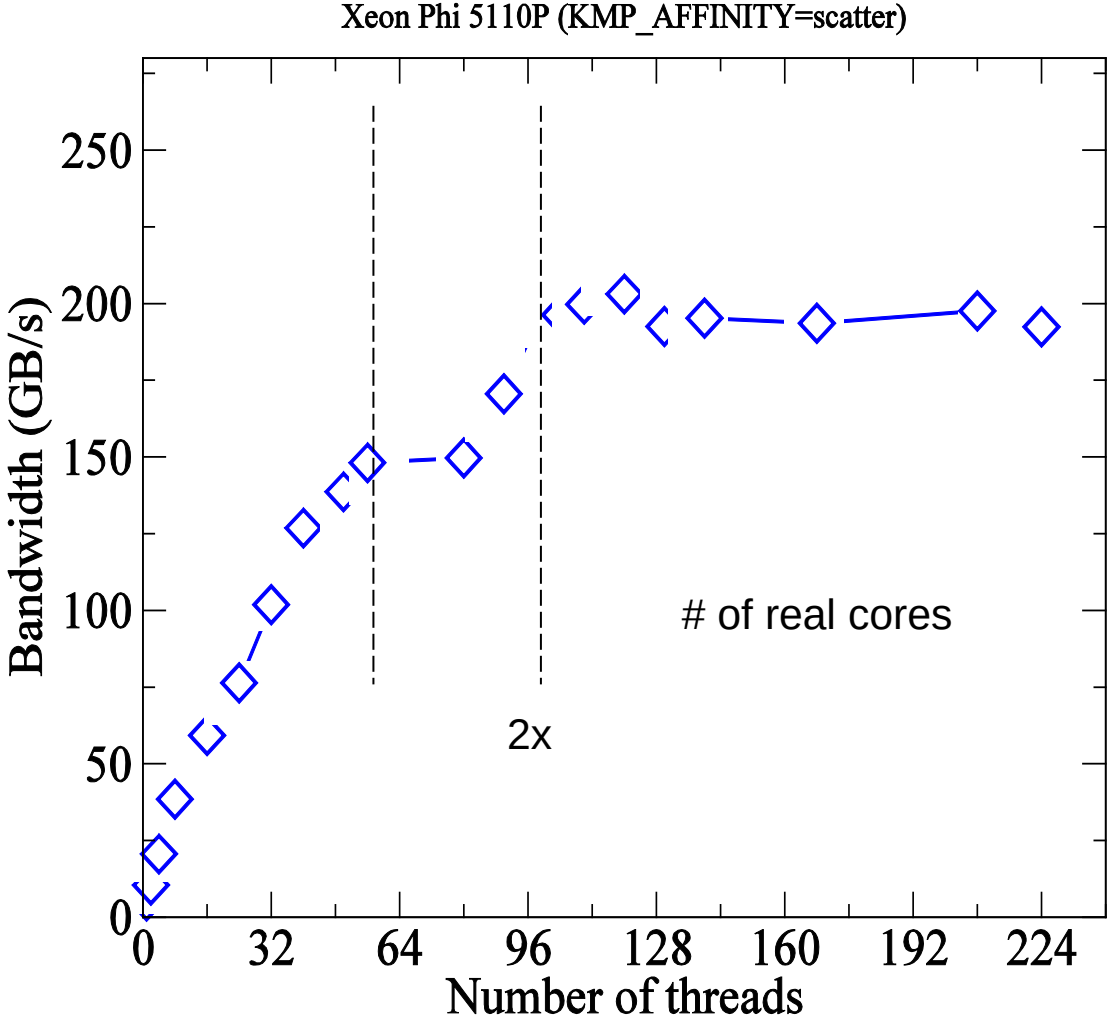
Bandwidth on single core



Example
Array Copy
x(:) = a*x(:)+y(:)

How does data travel from Mem. to CPU and back?

Copy bandwidth on Intel Phi



Setup

- Double-precision 1D array
- Memory alignment in 64 bytes
- TRIAD

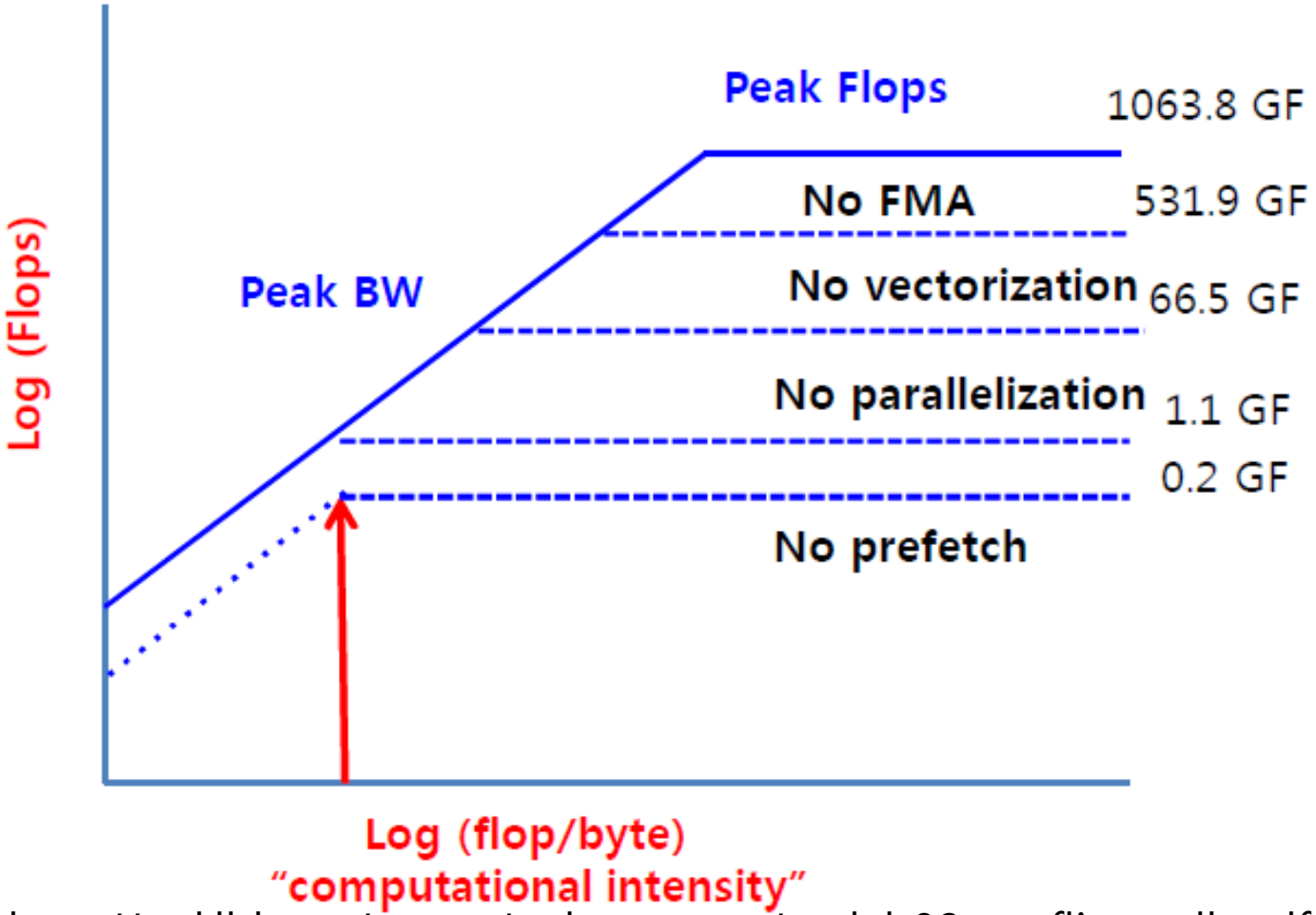
Xeon Phi 5110P

- Theoretical aggregate bandwidth = 352 GB/s

OpenMP with
Intel C/C++ compiler
2013 XE

KMP_AFFINITY=scatter

Roofline Model for Xeon Phi (DP)



Sources: http://crd.lbl.gov/assets/pubs_presos/parlab08-roofline-talk.pdf
Performace is upper bounded by both the peak flop rate, and the product of streaming bandwidth and the flop:byte ratio

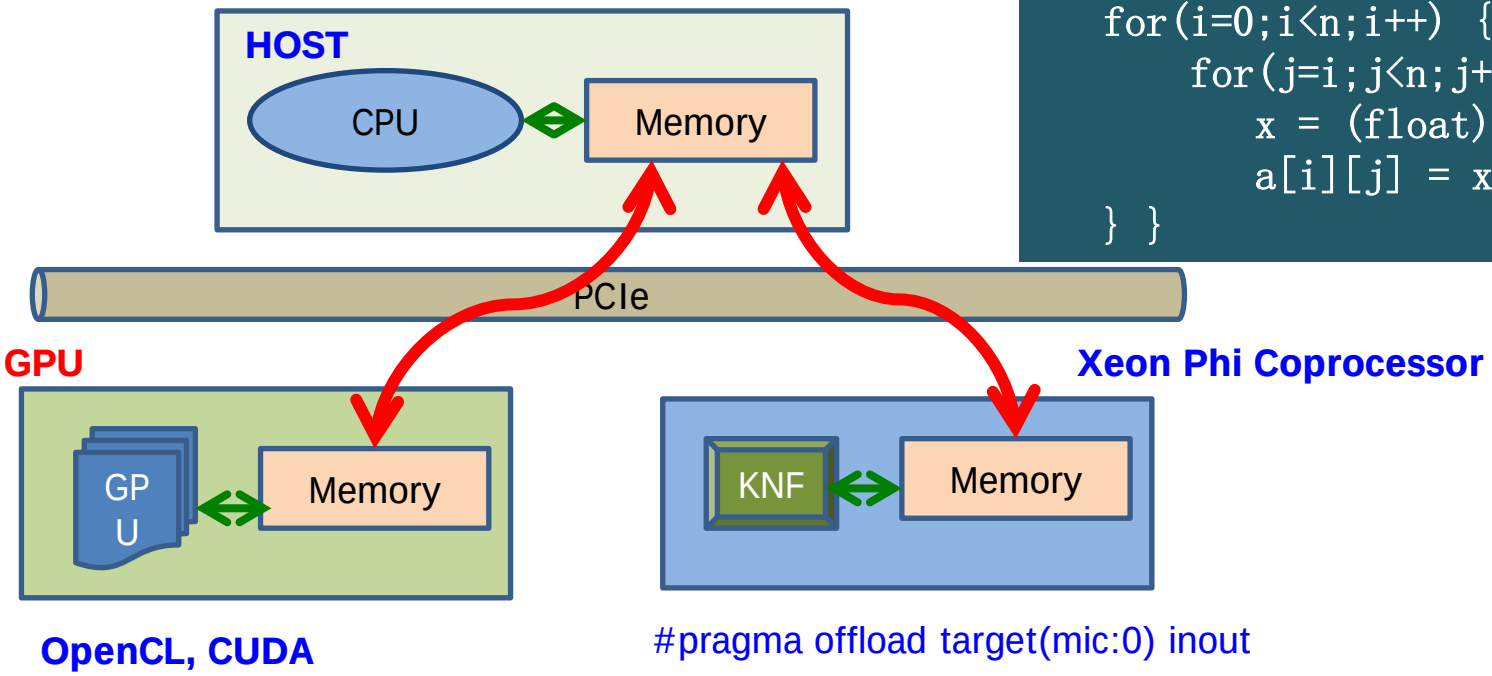
MIC Thread Affinity

- The optimal number of threads
  - Intel MIC maintains 4 hardware contexts per core
  - Round-robin execution policy,
- Use OpenMP application with NCORE cores
  - On host only for 2 x ncore
  - On MIC Native mode for 4 x ncore
  - Offload 4 x (ncore-1) because OpenMP runtime avoids the core OS runs
- Two ways to specify thread affinity
  - OMP_NUM_THREADS, KMP_AFFINITY
    - kmp_set_defaults("KMP_AFFINITY=compact")
      omp_set_num_threads(244);

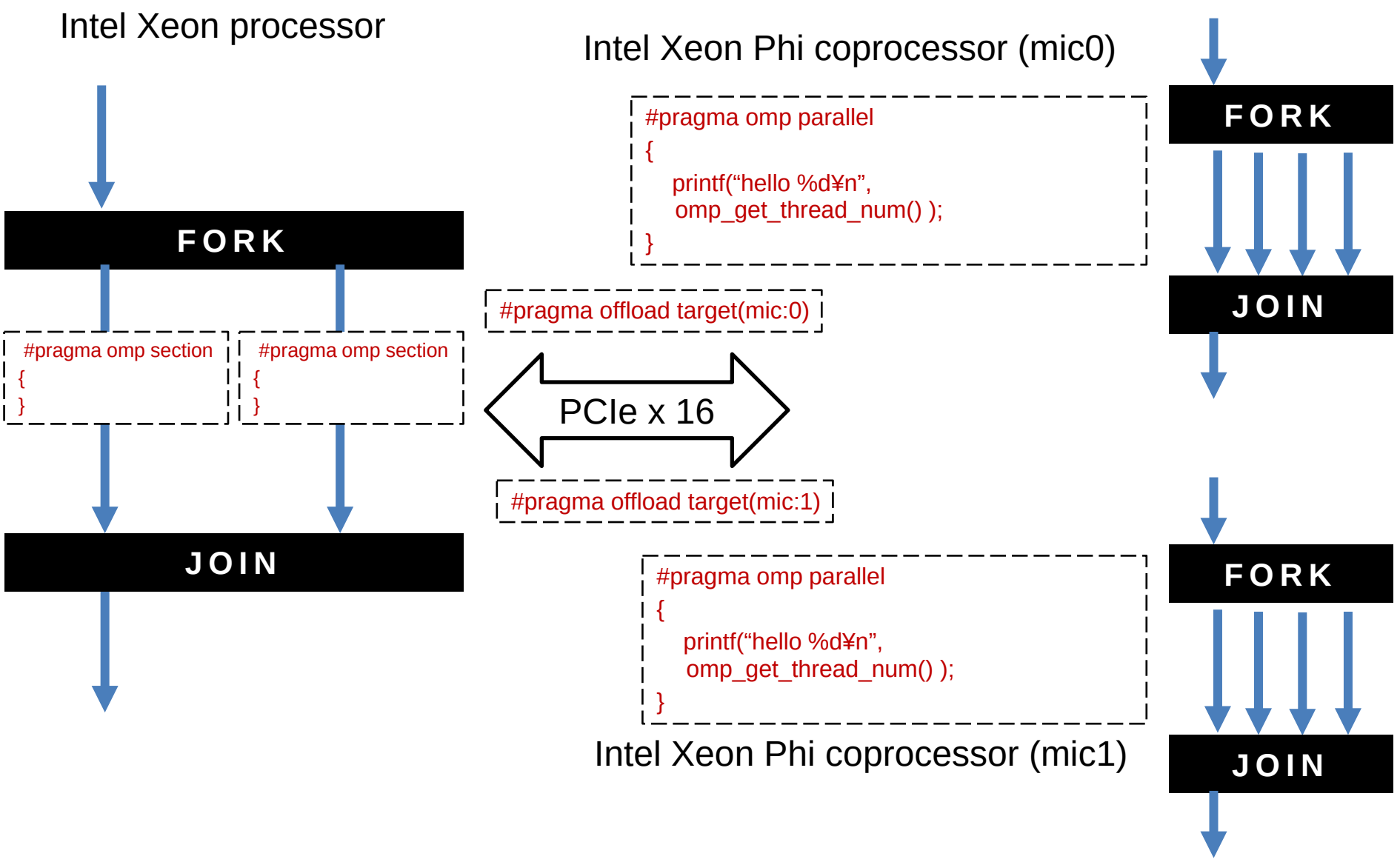
Data transfer for offload

- OpenMP programming model for data transfer
  - Host CPU and MIC do not share memory
  - Add pragmas similar to OpenMP
  - #pragma offload target (mic:0)

```
#pragma offload target(mic)
#pragma omp parallel for private(x)
   for(i=0;i<n;i++) {
       for(j=i;j<n;j++) {
          x = (float)(i + j);
          a[i][j] = x;
   } }
```



OpenMP Host with OpenMP Offload



Offloading Strategy

- Offload directive
  - #pragma offload target (mic:0 or mic:1)
- Great time to venture into manycores
  - Try offloading compute intensive section
  - Optimize data transfers
  - Split calculation
  - Use asynchronous transfer
- Good for
  - Code spends a lots of time doing computation without I/O
  - The data is relatively easy to encapsulate
  - Computation time is substantially higher than the data transfer time

SAXPY: Xeon Phi

```
int main(int argc, const char* argv[]) {
    int n = 10240;     float a = 2.0f;     float* x; float* y;
    x = (float*) malloc(n * sizeof(float));
    y = (float*) malloc(n * sizeof(float));
    for(int i=0; i<n; ++i){        x[i]=i;  y[i]=2.0*I + 1.0;    }

#pragma offload target(mic) in(x:length(n)) inout(y:length(n))
{
  #pragma parallel for
  for (int i=0; i< n; i++)
      y[i]=a*x[i]+y[i]
  }
   free(x); free(y);  return 0;
}
```

Count3s example with Offloading

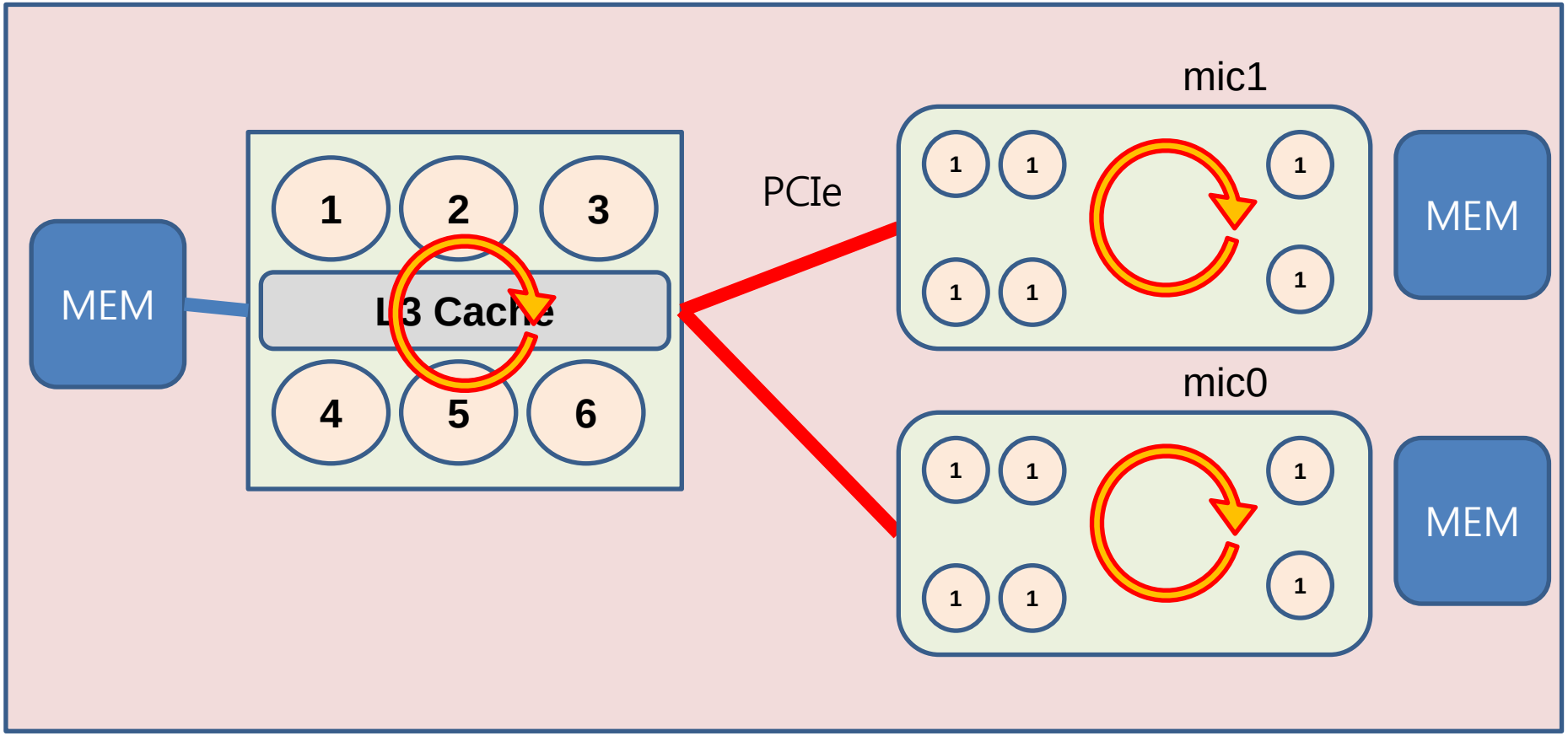
```
#include <stdlib.h>
__attribute__((target(mic))) int count3s(const int N,
const int* data){
    int icount=0;
    int i;
    for ( i = 0 ; i < N ; i++ ){
        if ( data[i] == 3 ) icount++;
    }
    return icount;
}

int num_c3s;
#pragma offload target(mic) in(array:length(NSIZE))
    {
        printf("Counting 3s from MIC! ¥n");
        fflush(0);
        num_c3s = count3s(NSIZE, array);
    }
}
```

```
./a.out
Hello World from CPU!
Hello World from MIC!
Counting 3s from MIC!
num_c3s=3332344
in the array[10000000] ratio= 33.32(%)
```

MPI Programming Models

- Host-only Model
  - All MPI ranks reside on the host
  - The coprocessors can be used by using offload pragmas
- Coprocessor-only Model
  - All MPI ranks reside on the coprocessors
- Symmetric Model
  - The MPI ranks reside on both the host and the coprocessor

“Native MPI” : Symmetric Mode



mpirun –host localhost –n 6 a.cpu : -host mic0 –n 120 a.mic : -host mic1 –n 120 a.mic

Hybrid Programming (MPI + Offload)

Fortran

```
program hybrid_program
  use omp_lib
  implicit none
  include 'mpif.h'
  INTEGER rank, nprocs, tid, ierr

  call MPI_Init(ierr)
  call MPI_Comm_size(MPI_COMM_WORLD, nprocs, ierr)
  call MPI_Comm_rank(MPI_COMM_WORLD, rank, ierr)

!DIR$ OFFLOAD BEGIN TARGET(mic)
!$OMP  PARALLEL  PRIVATE(tid)
    tid = omp_get_thread_num()
    print *, 'tid = ', tid, ' rank = ', rank, ' nprocs = ', nprocs
!$OMP END PARALLEL
!DIR$ END OFFLOAD
    call MPI_Finalize(ierr)
End
```

C

```
#include <stdio.h>
#include <mpi.h>
#include <omp.h>

int main(int argc, char *argv[])
{
   int rank, nprocs, tid;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
#pragma offload target(mic)
#pragma  omp  parallel  private(tid)
{
  tid = omp_get_thread_num();
  printf ("tid = %d rank = %d nprocs = %d ¥n",  tid,  rank,
    nprocs);
}
   MPI_Finalize();
}
```

$ mpiicc/mpiifort –openmp –o hello.x hello.c/f90

$ export MIC_ENV_PREFIX=MIC

$ export MIC_OMP_NUM_THREADS=4

$ mpirun –host localhost –n 2 ./hello.x

Vector register

Scalar: 64(80)-bit wide register, 1 double/ instruction
add [a] [b]

Intel SSE: 128-bit wide register(xmm), 2 doubles/ instruction
SSE2, SSE4.2

Intel AVX: 256-bit wide register(ymm), 4 doubles/ instruction
Shipped with Sandy bridge
Later, AMD will implement.

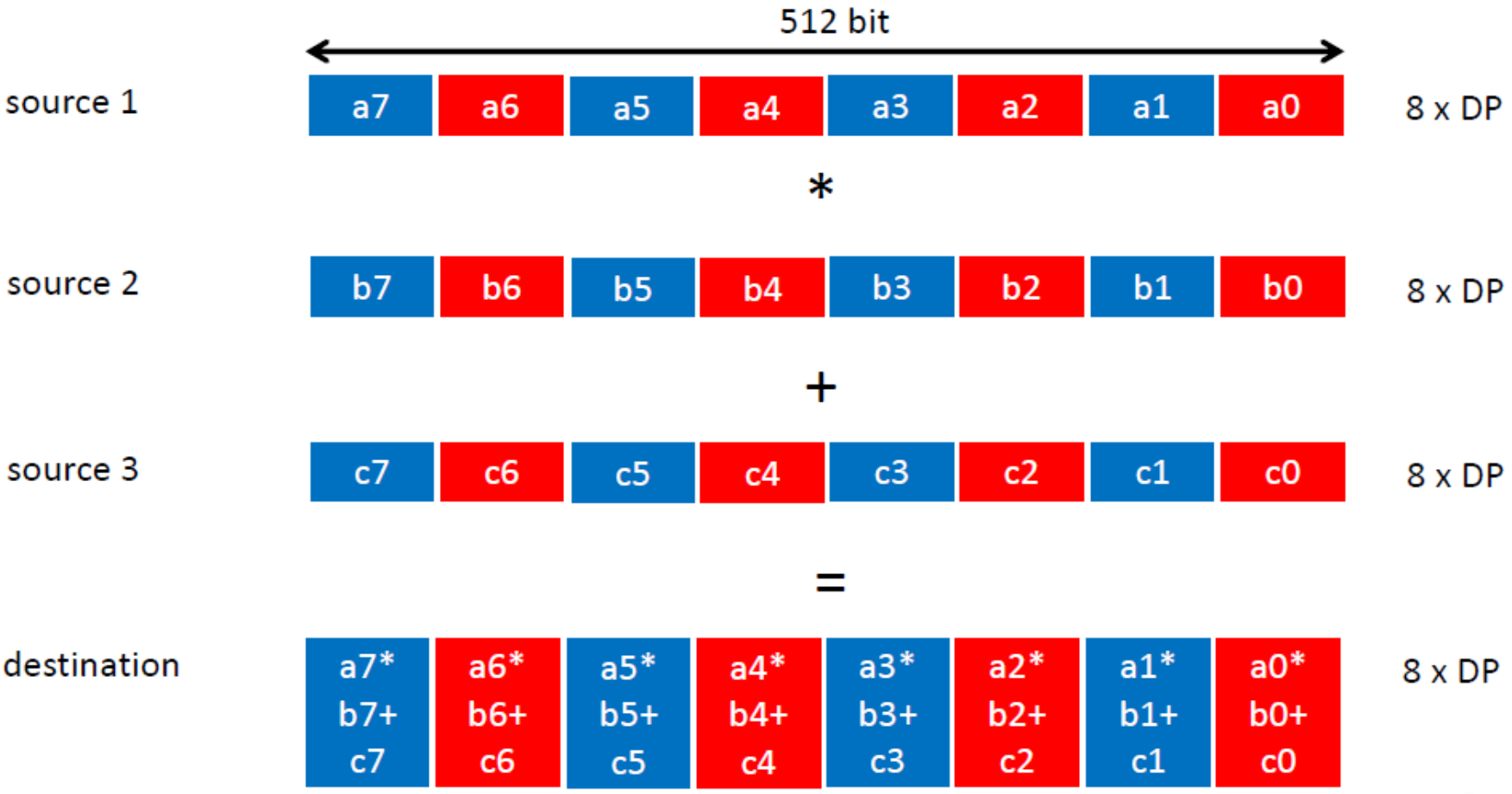
Intel Phi: 512-bit wide register(), 8 doubles/ instruction
Shipped with Knights Corner at 2012

Intrinsic: _mm{bit}_{operation}_{packed, scalar}{precision}()
ex. _mm128_add_pd(), _mm128_exp_pd() on icc

Compiler auto-vectorization, C Extension for Array Notation(Cilk+), ArBB(C++)

OpenCL: easy vectorization combined with multi-threading

SIMD Fused Multiply Add



SoA vs AoS

- Use Structure of Arrays (SoA) instead of Array of Structures (AoS)
  - Color structure


```
struct Colors{ //SoA
    float* r;
    float* g;
    float* b;
}
```

Summary

- You will benefit using Xeon Phi if you can
  - Take advantage of high memory bandwidth
  - Take advantage of wide vectors
  - Take advantage of the high thread count
  - If you can't, you're probably better off with Xeons
- If you want a good performance
  - you will need to optimize your code by extracting parallelism such as OpenMP, SIMD, OpenCL, MPI
  - This is at least the same amount of work you would put into porting to CUDA.