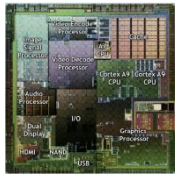
众核编程基础
MIC

薛巍

2012.06.05

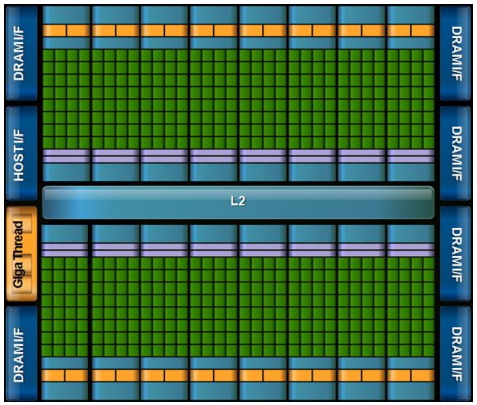
The Challenge of Parallelism

- Programming parallel processors is one of the challenges of our era



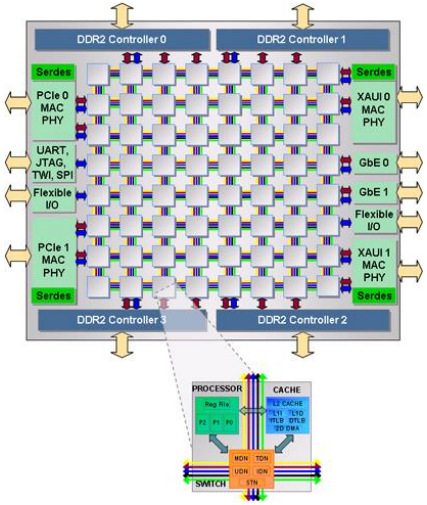
NVIDIA Tegra 2 system on a chip (SoC)

- **Dual-core ARM Cortex A9.**
- **Integrated GPU. Lots of DSP.**
- **1 GHz.**
- **2 single-precision GFLOPs peak (CPUs only)**



Nvidia Fermi

- **16 cores, 48-way multithreaded,**
- **4-wide Superscalar, dual-issue, 3**
- **2-wide SIMD (half-pumped)**
- **2 MB (16 x 128 KB) Registers, 1**
- **MB (16 x 64 KB) L1 cache, 0.75 MB L2 Cache**



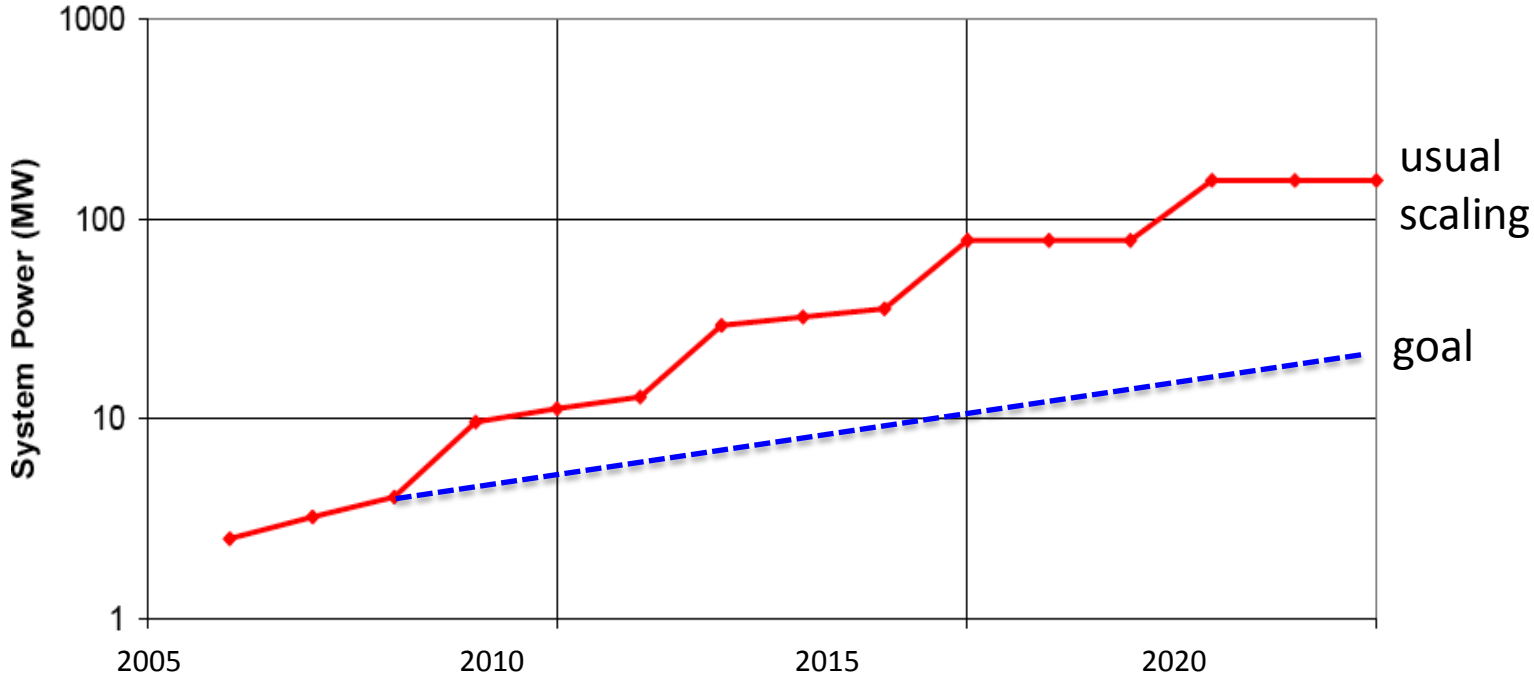
Tilera Tile64

- **64 processors**
- **Each tile has L1, L2, can run OS**
- **443 billion operations/sec.**
- **500-833 MHz**
- **50 Gbytes/sec memory bandwidth**

Energy Efficient Computing is Key to Performance Growth

At $1M per MW, energy costs are substantial

- 1 petaflop in 2010 used 3 MW
- 1 exaflop in 2018 would use 100+ MW with "Moore's Law" scaling

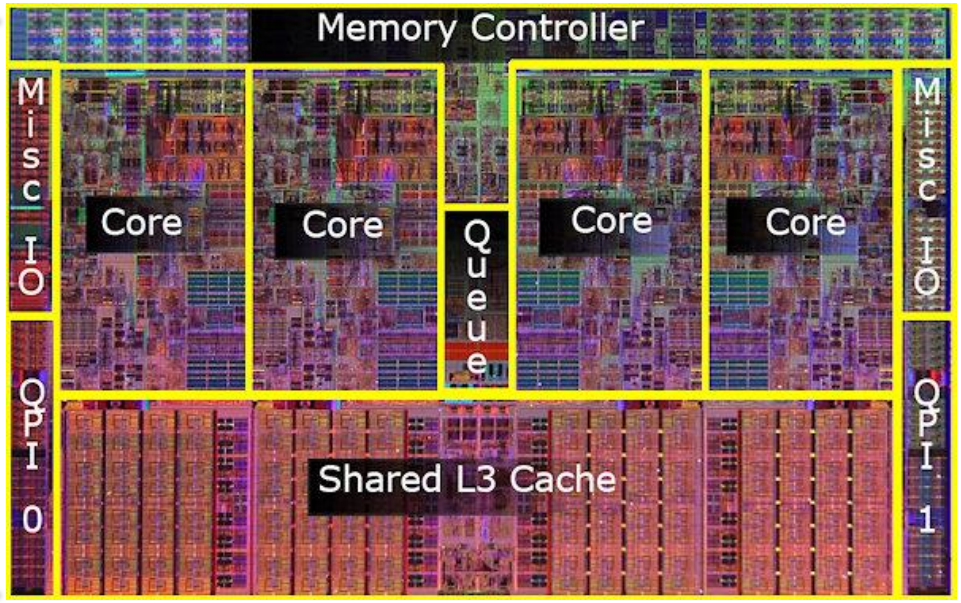


This problem doesn't change if we were to build 1000 1-Petaflop machines instead of 1 Exasflop machine. It affects every university department cluster and cloud data center.

Challenge: New Processor Designs are Needed to Save Energy

Cell phone processor (0.1 Watt, 4 Gflop/s)

Server processor (100 Watts, 50 Gflop/s)



- Server processors have been designed for performance, not energy
  - Graphics processors are 10-100x more efficient
  - Embedded processors are 100-1000x
  - Need manycore chips with thousands of cores

What is MIC

- Intel Many Integrated Core Architecture
  - Targeted at highly parallel HPC workloads
  - Power efficient cores, support for parallelism
    - Cores: less speculation, threads, wider SIMD
    - Scalability: high BW on die interconnect and memory
  - General Purpose Programming Environment
    - Linux
    - Fortran, C/C++
    - x86 memory model, IEEE 754

Knights Corner Coprocessor

Intel® Xeon® Processor

TCP/IP

PCIe x16

PCIe x16

System Memory

KNC Card

KNC Card

GDDR5 Channel ... GDDR5 Channel

GDDR5 Channel

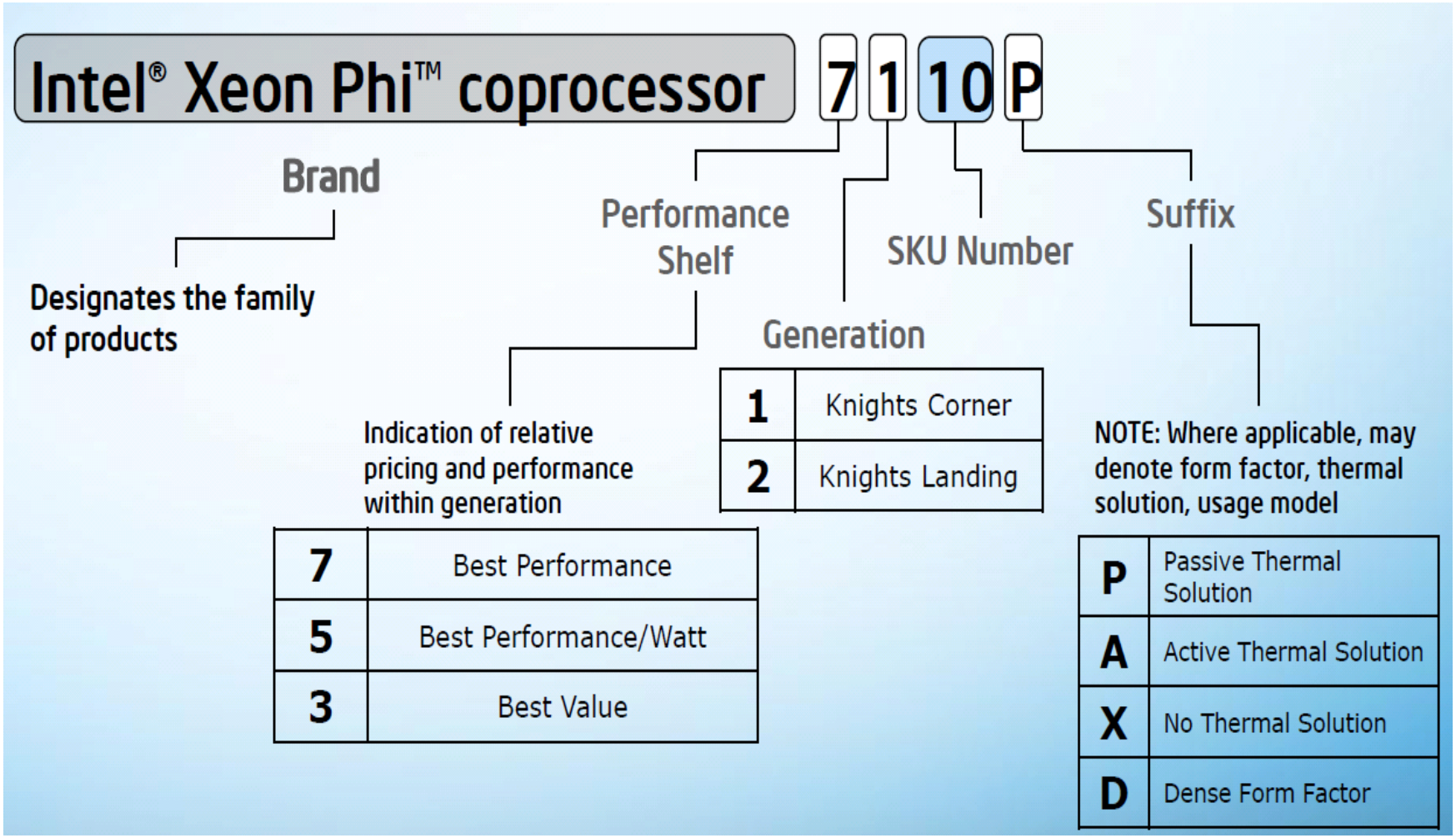
...

GDDR5 Channel

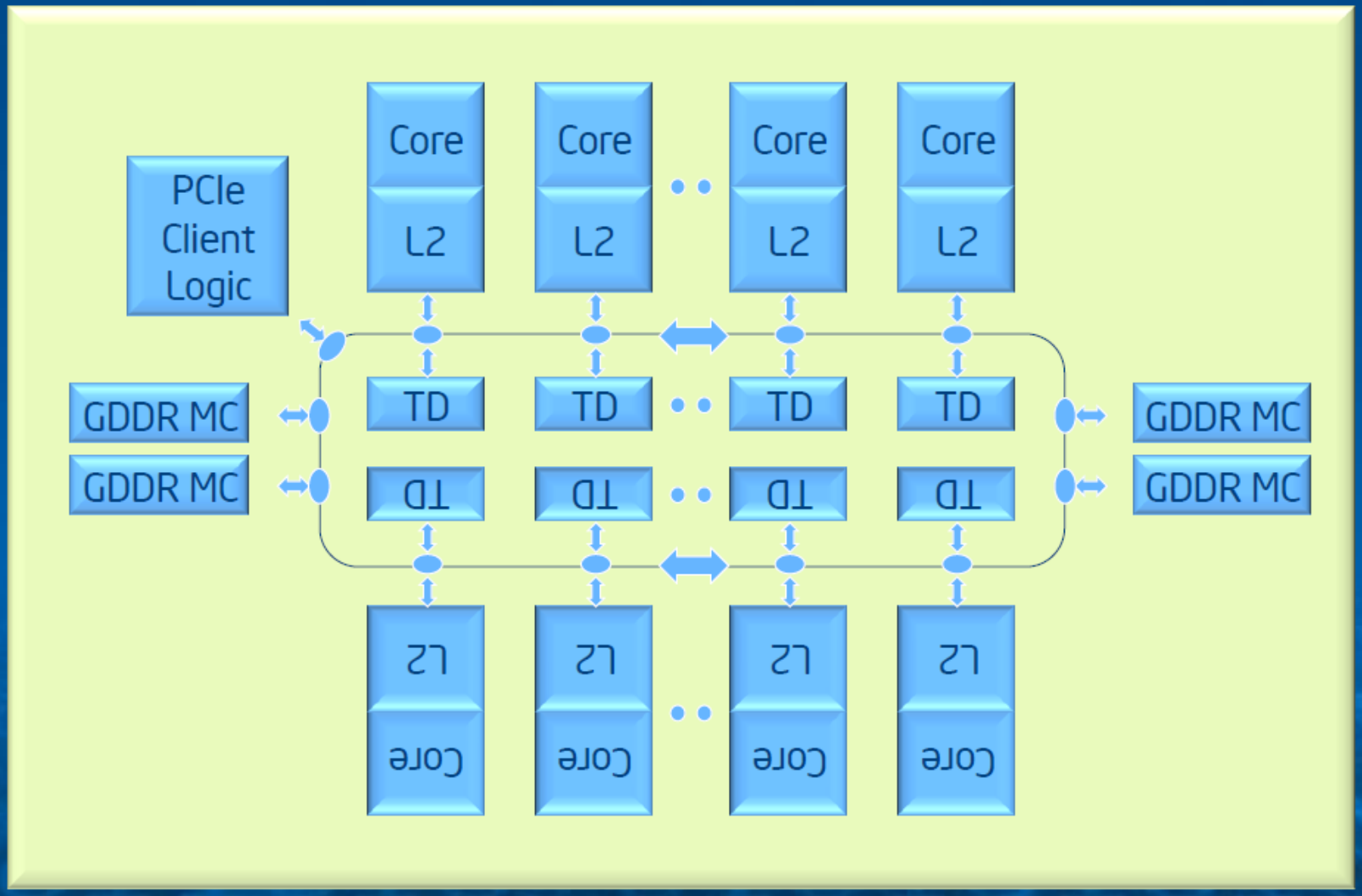
GDDR5 Channel ... GDDR5 Channel

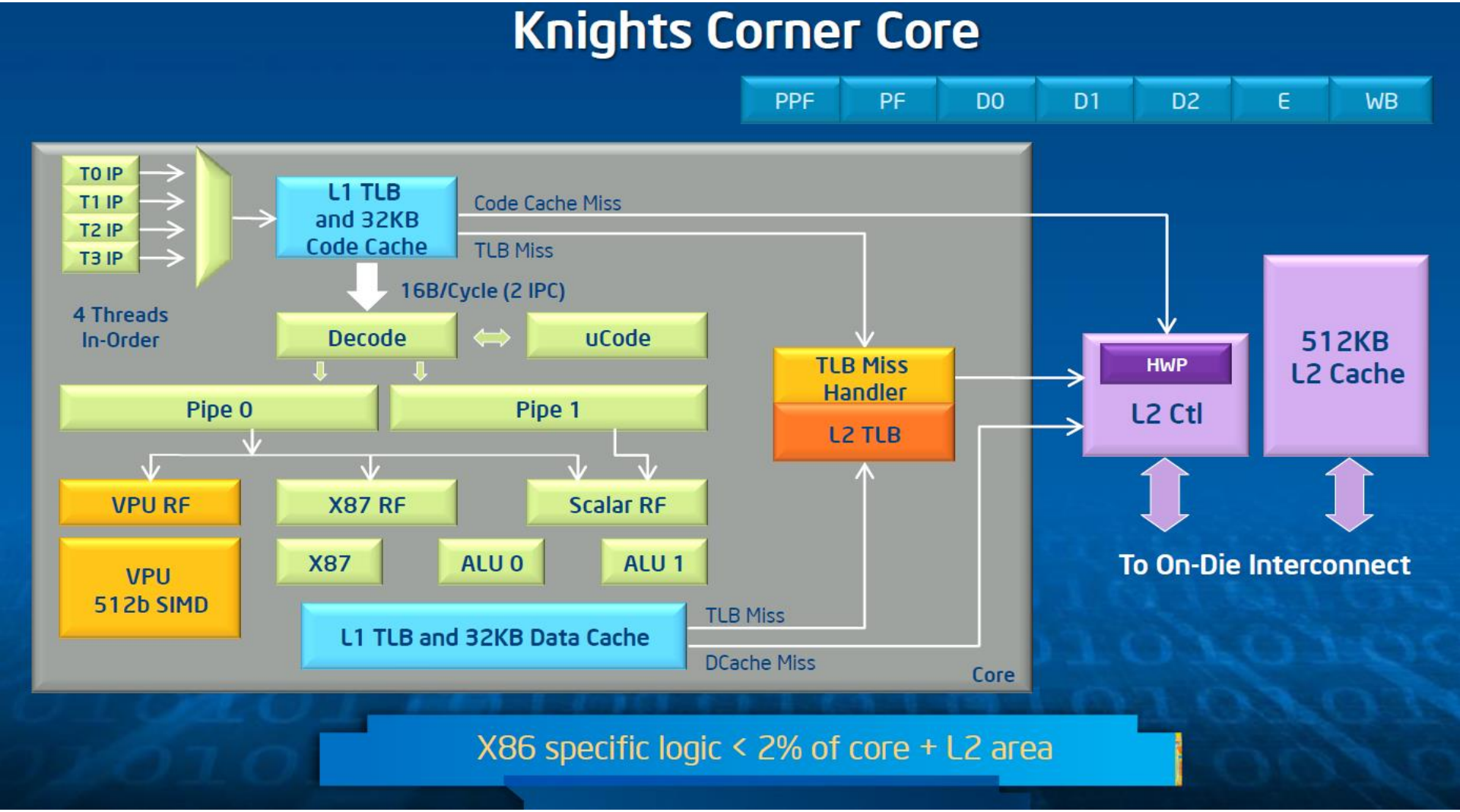
>= 8GB GDDR5 memory

Knight Corner Numbering



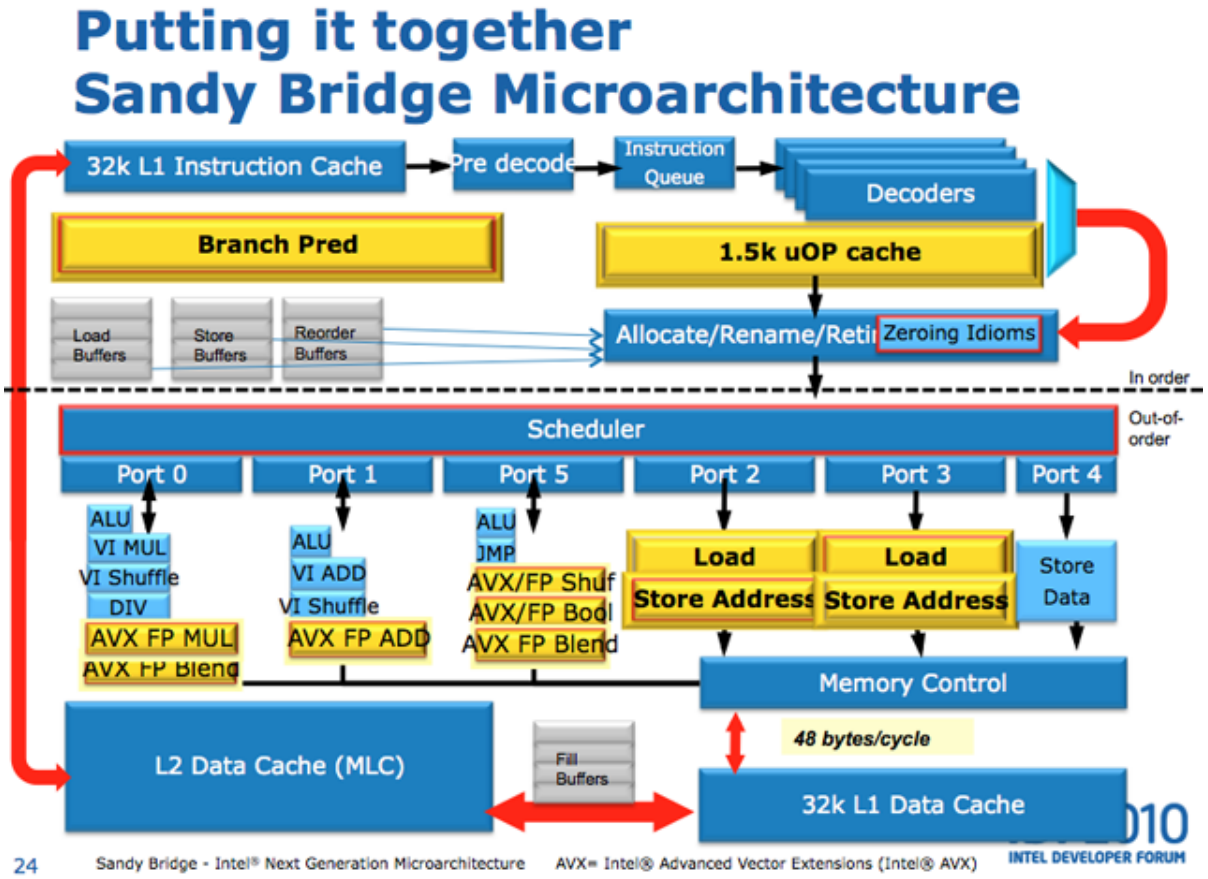
Knights Corner Micro-architecture

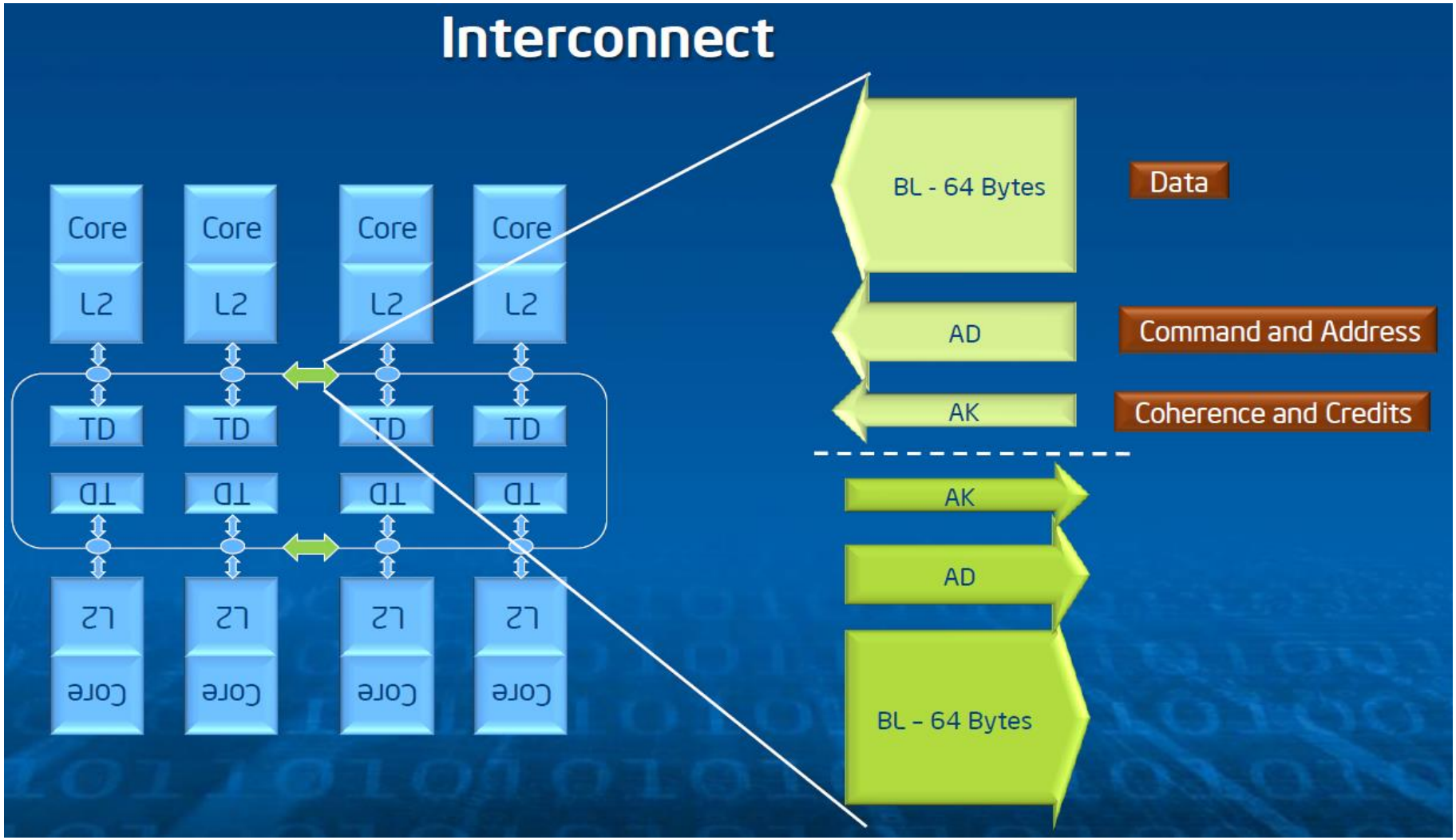




同一线程的指令不能连续发射

Comparing Xeon Phi Core with Xeon Core




Interconnect
Core
L2
TD
BL - 64 Bytes
Data
AD
Command and Address
AK
Coherence and Credits
AK
AD
BL – 64 Bytes

Intel Xeon Phi的编程模式

- Native Programming Model
  - entire application running independently on the MIC card
  - evaluate whether your application fits into the card
- Offload Programming Model
  - application decomposed into the host part and the kernel part
  - offload the heavy kernels to the acceleration card
  - similar to the GPU and the FPGA programming model

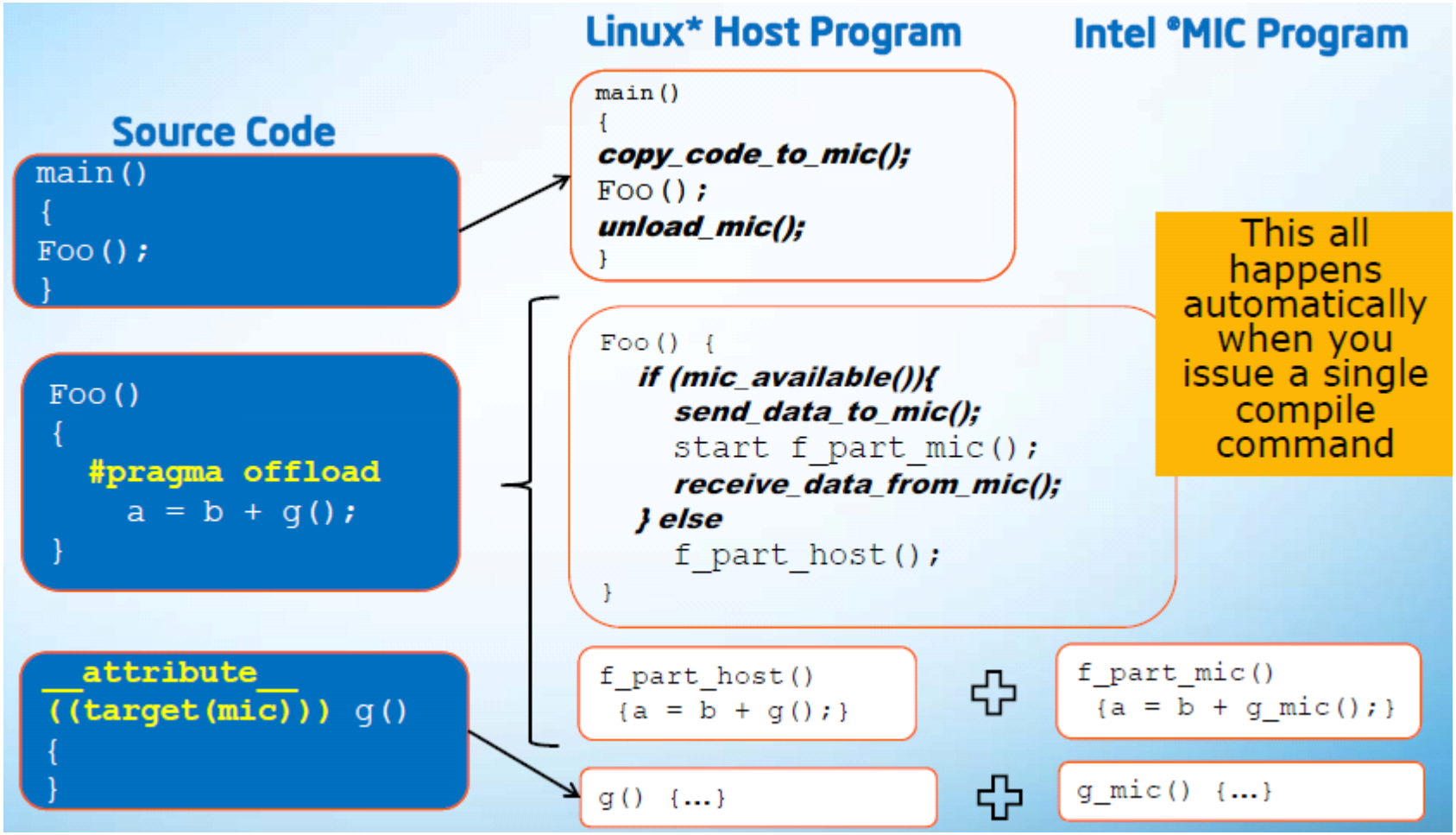
Native Programming Model

- The MIC card is running as an independent node (a simplified Linux)
- write the same code as on CPU
- different compilation and execution methods
  - specific compiling flags: -mmic
  - copy the executable and libraries to the MIC card
- generally provide better performance than the offload mode (20-30%)
- provide the option to use the CPU as the coprocessor

Offload Programming Model

- Add pragmas or keywords to specify the sections that would run on MIC
  - OpenMP:
    #pragma offload target(mic)
  - Intel Cilk+: _Cilk_offload_to
- Automated compilation of different code sections on different architectures
- Running of the code

Compiler Automation for offload code



Offload Data Transfer

- Explicit data transfer:
  - programmer specifies the variables that need to copied between the host and the card
  - #pragma offload target(mic) in(data:length(size))
- Implicit data transfer:
  - programmer marks the variables that need to be shared between the host and the card
  - _Cilk_shared double foo;

Example of offload code

- C/C++ Offload Pragma

```
#pragma offload target (mic)
#pragma omp parallel for reduction(+:pi)
 for (i=0; i<count; i++)  {
       float t = (float)((i+0.5)/count);
       pi += 4.0/(1.0+t*t);
 }
 pi /= count;
```

- Function Offload Example

```
#pragma offload target(mic)
      in(transa, transb, N, alpha, beta) \
      in(A:length(matrix_elements)) \
      in(B:length(matrix_elements)) \
      inout(C:length(matrix_elements))
        sgemm(&transa, &transb, &N, &N, &N,
&alpha, A, &N, B, &N, &beta, C, &N);
```

- Fortran Offload Directive

```
!dir$ omp offload target(mic)
!$omp parallel do
    do i=1,10
            A(i) = B(i) * C(i)
    enddo
```

- C/C++ Language Extension

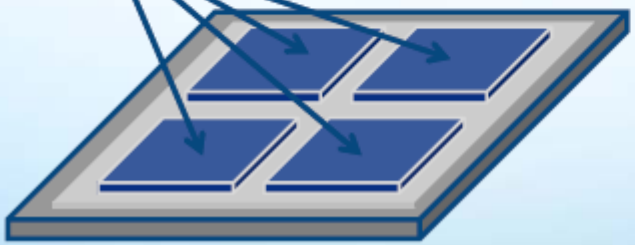
```
class _Shared common  {
   int data1;
   char *data2;
   class common *next;
   void process();
};
_Shared class common  obj1, obj2;
_Cilk _spawn  _Offload obj1.process();
_Cilk_spawn                obj2.process();
```

Example of automatic offload

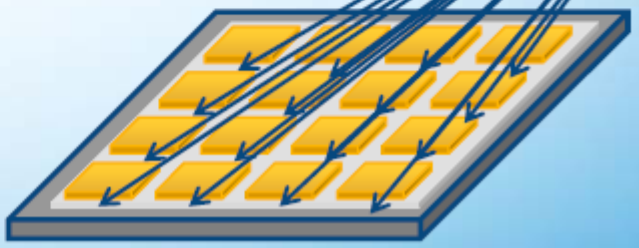
- Possible with advanced Intel Libraries – e.g. MKL

```
void foo()  /*  Intel® Math Kernel Library */
{
      float *A, *B, *C; /* Matrices */
      sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N, &beta, C, &N);
}
```

Implicit automatic offloading requires no code changes, simply link with the offload MKL Library



Intel® Xeon® processor



Intel® Xeon Phi™ coprocessor

Heterogeneous Compiler – What is generated

Note that for both techniques, the compiler generates two binaries:

- The host version
  - **includes all functions/variables** in the source code, whether marked `#pragma offload`, `__attribute__((target(mic)))`, `_Cilk_shared`, `_Cilk_offload`, or not
- The coprocessor version
  - **includes only functions/variables** marked `#pragma offload`, `__attribute__((target(mic)))`, `_Cilk_offload`, or `_Cilk_shared` in the source code

Heterogeneous Compiler – Command-line option

- "`-openmp`" is automatically set in offload

- Most command line arguments set for the host are set for the coprocessor build
  - Unless overridden by `-offload-option,mic,…`

- Setting up the compiler build environment

  CSH: `source /opt/intel/composerxe_mic/bin/compilervars.csh intel64`

  SH: `source /opt/intel/composerxe_mic/bin/compilervars.sh intel64`

Heterogeneous Compiler – Command-line option

Offload-specific arguments to the Intel® Compiler:

- Ignore language constructs for offloading (by default the compiler recognizes language constructs for offloading if they are specified):

  `-no-offload`

- Produce a report of offload data transfers at compile time (not runtime)

  `-opt-report-phase:offload`

- Specify options to be used for Intel® MIC Architecture

  `-offload-option,mic,`***`tool`***`,"`***`option-list`***`"`

Example:

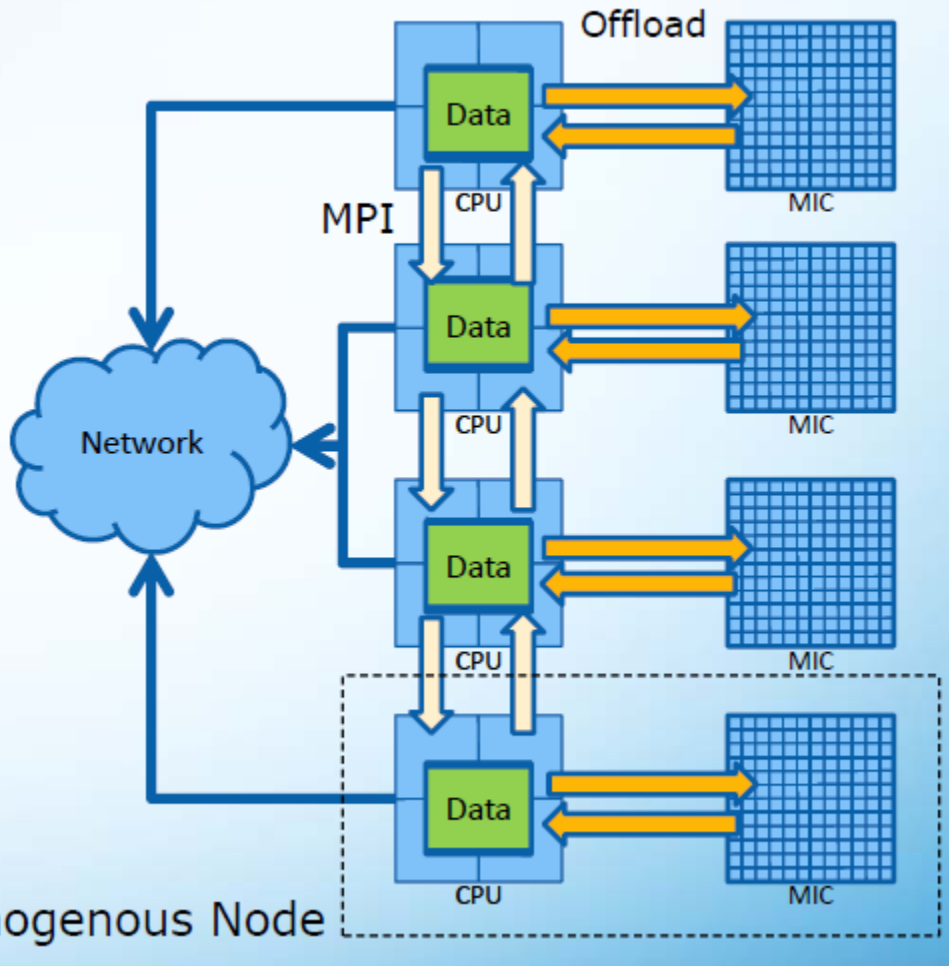
```
Icc -I/my_dir/include -DMY_DEFINE=10 -offload-option,mic,compiler, "-
  I/my_dir/mic/include -DMY_DEFINE=20" -offload-option,mic,ld, "-
  L/my_dir/mic/lib" hello.c
```

Passes `"-I/my_dir/mic/include -I/my_dir/include -DMY_DEFINE=10 -DMY_DEFINE=20"` to the offload compiler

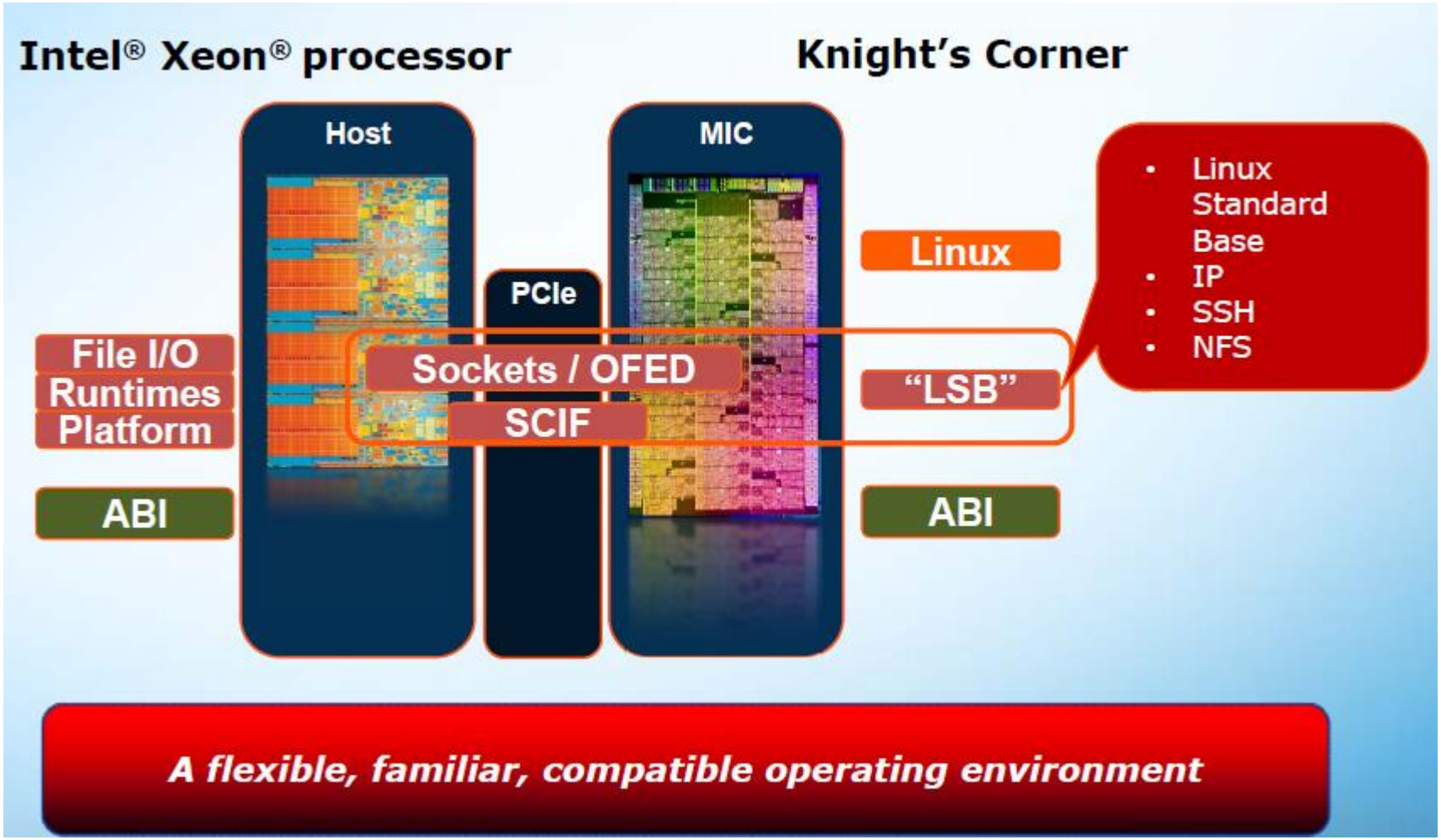
Passes `"-L/my dir/mic/lib -L/my dir/lib"` to the offload ld

Offload + MPI

- MPI ranks on Intel® Xeon® processors (only)
- All messages into/out of Xeon processors
- Offload models used to accelerate MPI ranks
- TBB, OpenMP, Cilk Plus, Pthreads within Intel® MIC
- Homogenous network of hybrid nodes



Operating Environment



Open**F**abrics **E**nterprise **D**istribution

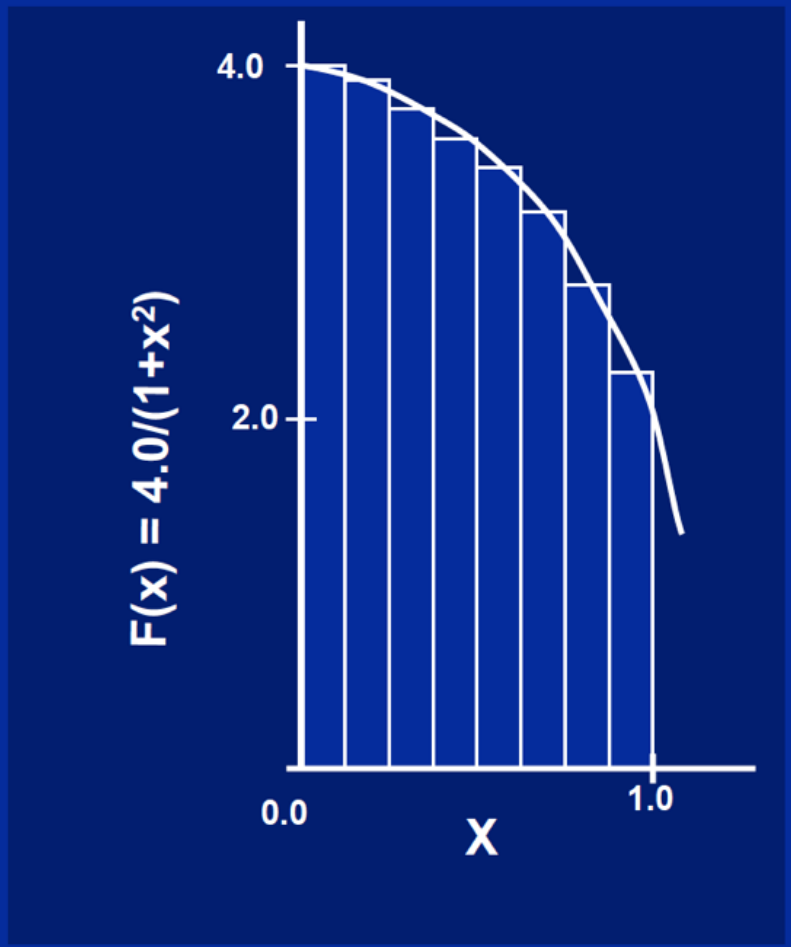
Symmetric **C**ommunications **I**nter**f**ace

Key points for MIC

- Choose the right Xeon-centric or MIC-centric model for your application
- Vectorize your application
  - Use Intel's vectorizing compiler
- Parallelize your application
  - With MPI (or other multiprocess model)
  - With threads (via Pthreads, TBB, Cilk Plus, OpenMP, etc.)
- Consider standard optimization techniques such as tiling, overlapping computation and communication, etc.

Computation of PI

Numerical Integration



Mathematically, we know that:

$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

We can approximate the integral as a sum of rectangles:

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Where each rectangle has width Δx and height $F(x_i)$ at the middle of interval i.

Serial Version

```
static long num_steps = 100000;
double step;
void main ()
{       int i;   double x, pi, sum = 0.0;

        step = 1.0/(double) num_steps;

        for (i=0;i< num_steps; i++){
                x = (i+0.5)*step;
                sum = sum + 4.0/(1.0+x*x);
        }
        pi = step * sum;
}
```

Experiments

- Pi
  - Serial version
  - OpenMP CPU version
  - OpenMP MIC version (Native and Offloading)

- 期权定价问题的step5

Guide for experiment

- Please login166.111.68.173 with putty
  - user:test
  - passwd:test@tsinghua

- Go to your dir in home dir
  - cd xuew

- Copy cases to your dir
  - cp –pr /tmp/lec11/MonteCarlo/ .
  - cp –pr ~/pi.cpp .

课程回顾

- 本课程的目的
  - 偏实践类课程
  - 实实在在体验并行软件开发过程
    - **并行编程** —实现
    - **并行程序性能分析与优化** —分析优化
    - 并行算法/程序设计 —设计
    - 并行计算工具 —组合
  - 一些对于计算机的感性认识

- 预期效果
  - 在今后采用并行计算方法提升程序性能
  - 能编写不复杂的并行程序
  - 知道使用并行工具提高并行程序实现效率
  - 初步了解并行计算程序设计和优化的过程

- 进一步深入学习可以首先参考：
  http://www.cs.berkeley.edu/~demmel/cs267

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

计算机系并行计算相关课程

计算机系统性能测试与分析

高级编译与优化技术

大规模科学计算

计算科学与工程中的并行编程技术

网络存储技术

网格计算

并行计算

高等计算机体系结构

高性能计算前沿技术探讨

高性能计算导论

并行计算基础

高性能计算平台与工具

Using Language Extensions for Intel® MIC

Simple Offload Extensions with the Intel® Compilers

	C/C++ Syntax	Semantics
New offload pragma	#pragma offload (clauses)	Execute next statement on target (which could be an OpenMP* parallel construct)
Place function on target	__declspec (target (x))	Compile function for host and target
Place on data target	__declspec (target(MIC)) float array [8000];	Two arrays are created, one on the host and one on the Intel® Xeon Phi™ coprocessor

	Fortran Syntax	Semantics
New offload directive	!dir$ omp offload <clauses>	Execute next OpenMP* parallel construct on target
Place function on target	!dir$ attributes offload:<x> :: <rtn-name>	Compile function for host and target

Using Language Extensions (contd.)

Variables restricted to scalars, arrays and pointers to scalars/arrays, structs (which can be bit-wise copied)

What	Syntax	Semantics
Target specification	target (name)	Where to run construct
Inputs	in (var-list modifiers$_{opt}$)	Copy CPU to target
Outputs	out (var-list modifiers$_{opt}$)	Copy target to CPU
Inputs & outputs	inout (var-list modifiers$_{opt}$)	Copy both ways
Non-copied data	nocopy (var-list modifiers$_{opt}$)	Data is local to target
Modifiers		
Specify pointer length	length (element-count-expr)	Copy that many pointer elements
Control pointer memory allocation	alloc_if (condition)	Allocate new block of memory for pointer if condition is TRUE
Control freeing of pointer memory	free_if (condition)	Free memory used for pointer if condition is TRUE