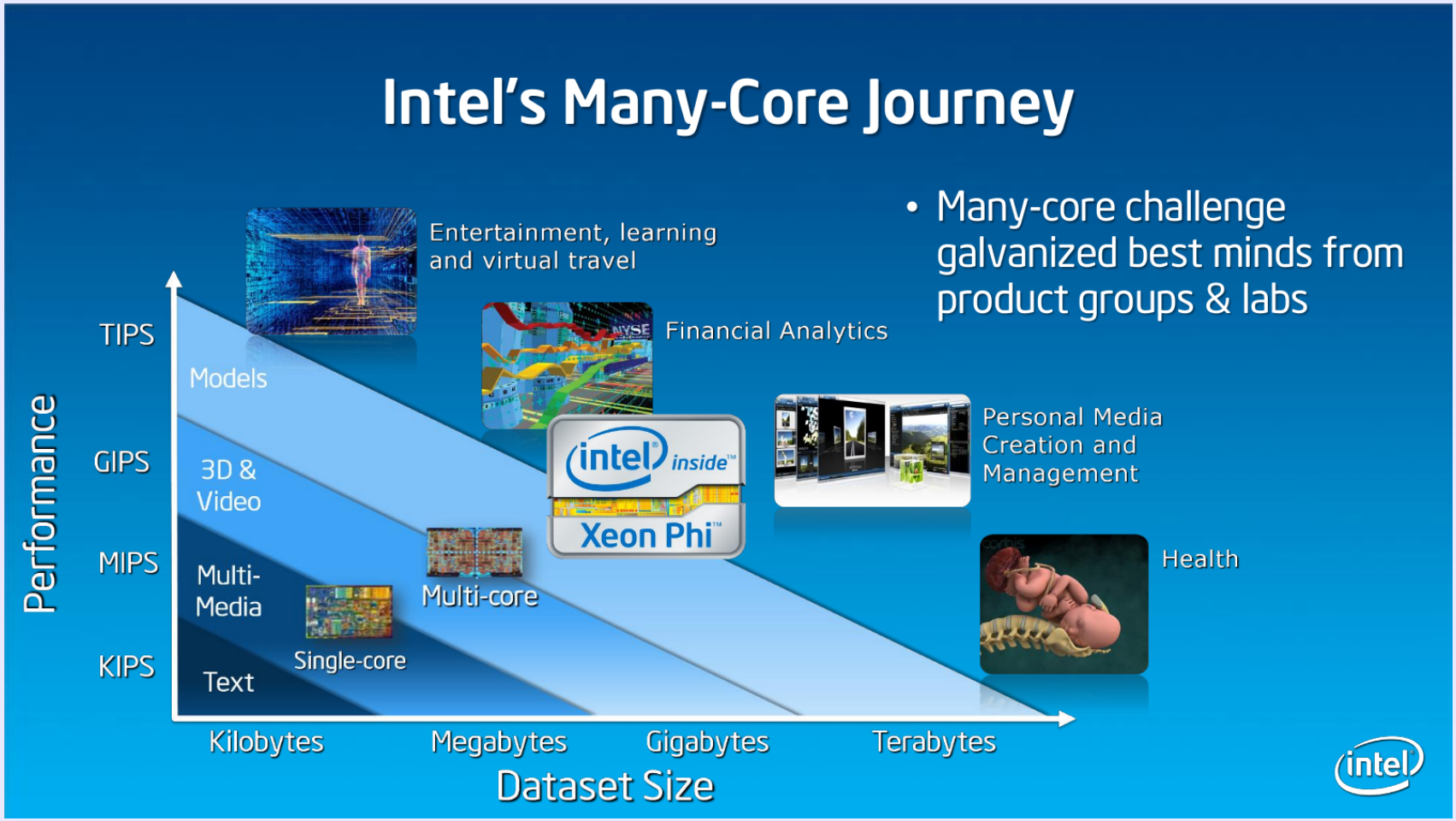
MIC编程介绍

崔　涛

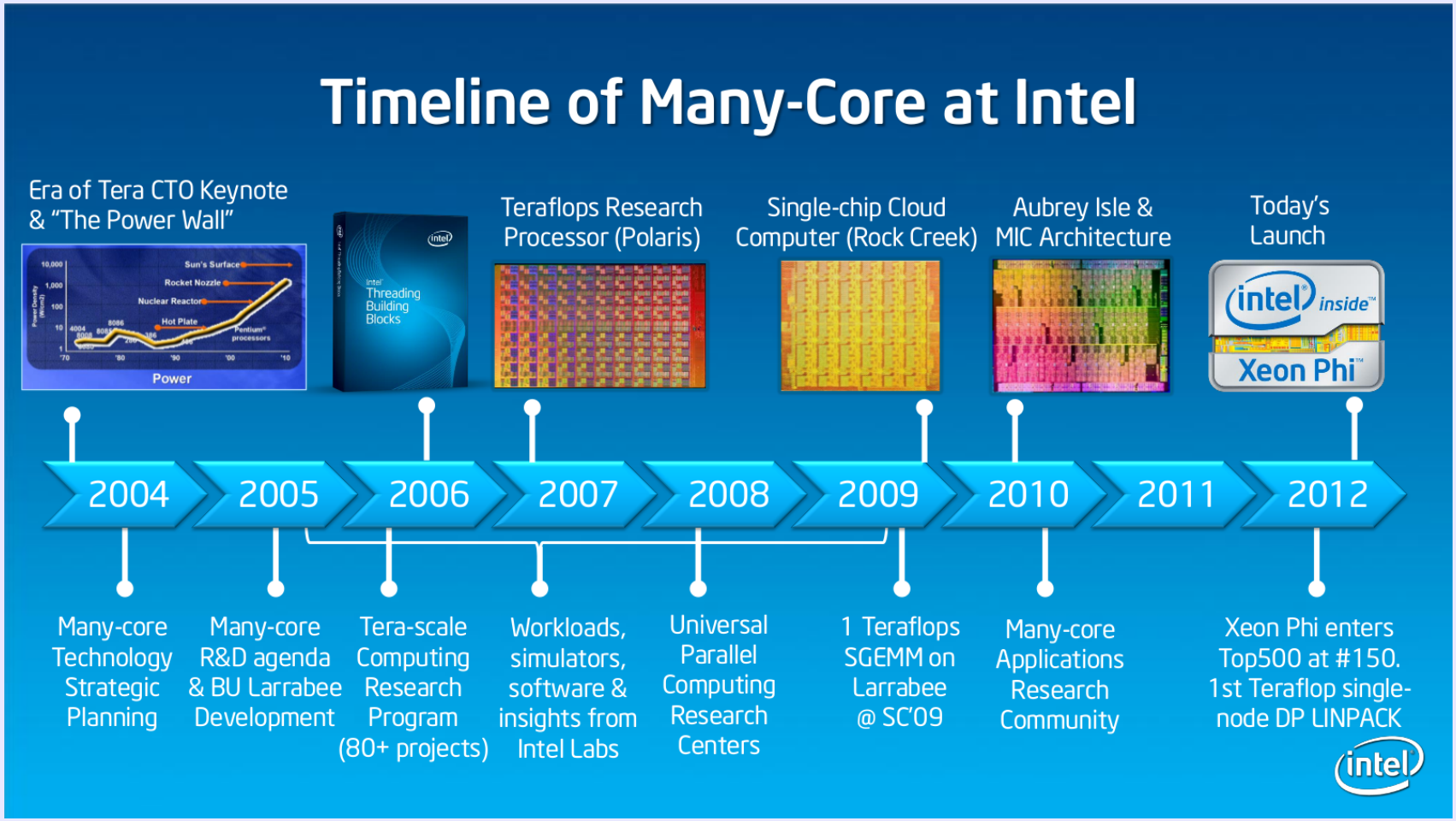
tcui@lsec.cc.ac.cn

中国科学院数学与系统科学研究院

Intel's Many Core



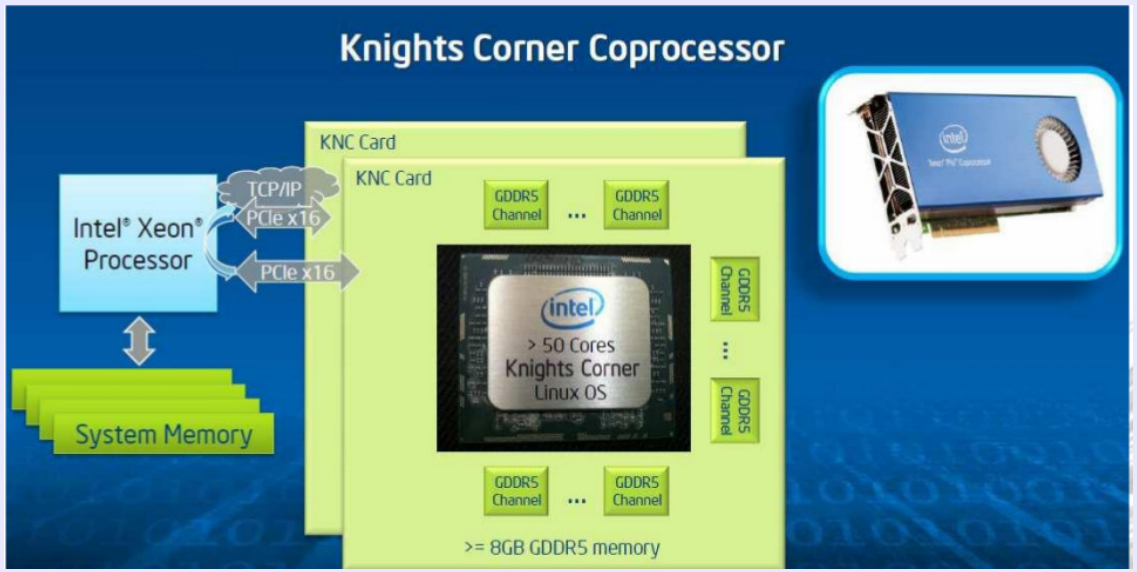
Intel's Many Core



什么是MIC

英特尔至强融核Xeon Phi，采用MIC（Intel Many Integerated Core）架构，用于高性能并行计算。MIC基于X86架构，支持多种并行模型，OpenMP、pThread、OpenCL、MPI等并行编程语言，采用C、C++和Fortran三种语言进行软件移植开发，特点以编程简单（引语方式）著称，工具链丰富。

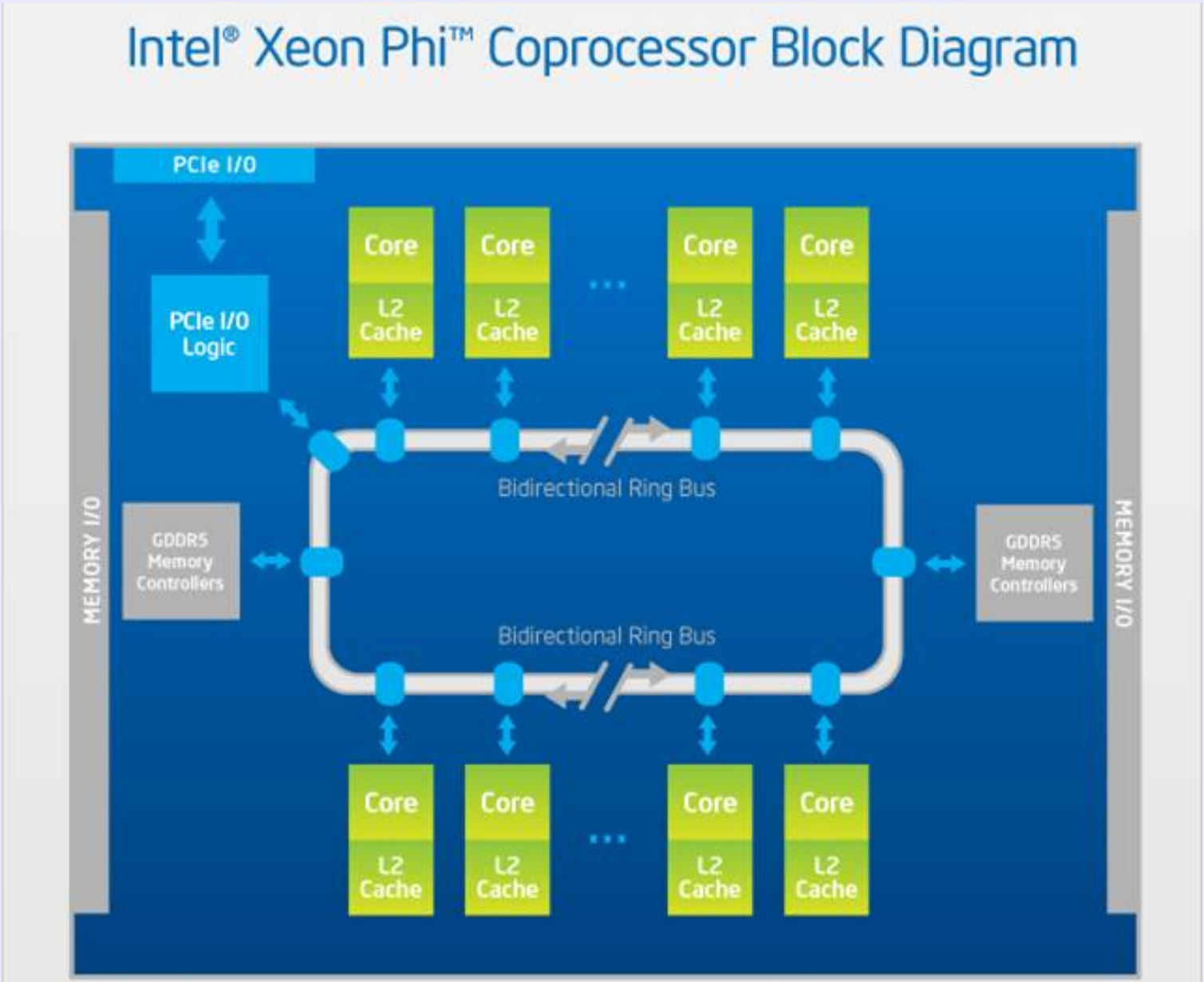
Xeon Phi的第一代产品的架构代号为“Knights Corner”。Knights Corner采用Intel 2012开始使用的3D“三门”晶体管技术，使用22nm工艺制造，MIC卡含有50个以上的核，每个核可以支持4个线程，双精性能超过1TFlops，含有512bit的向量宽度，支持8个双通道GDDR内存控制器，内存大小为6GB或8GB。



术语解析 I

MIC	Intel Many Integrated Core
Inte Xeon Phi	MIC产品家族品牌名称，英特尔至强融核。
KNC	Knights Corner.第一代至强融核协处理器。
Host	主机。安装了MIC协处理器的Inter至强处理器平台。
Target	与Host对应，指协处理器及相应的Host端运行时环境。
μOS	运行于MIC协处理器上的基于GNU Linux构建的操作系统及工具。
ISA	Instruction Set Architecture,指令级架构。
VPU	Vector Processing Unit,向量处理单元。
SPU	Scalar Processing Unit,向量处理单元。
Ring	片上双向环形高速互联总线。
CRI	The Core Ring Interface, 内核环形总线接口，是内核与片上环形总线的接口。
offload Compiler	可编译生成同时运行在Host和Target端的二进制文件的异构编译器。

Kight Corner协处理器微架构



查看MIC信息 I

- 查看是否按安装MIC卡:

```
[xxx@mic]$lspci|grep -i co-processor
```

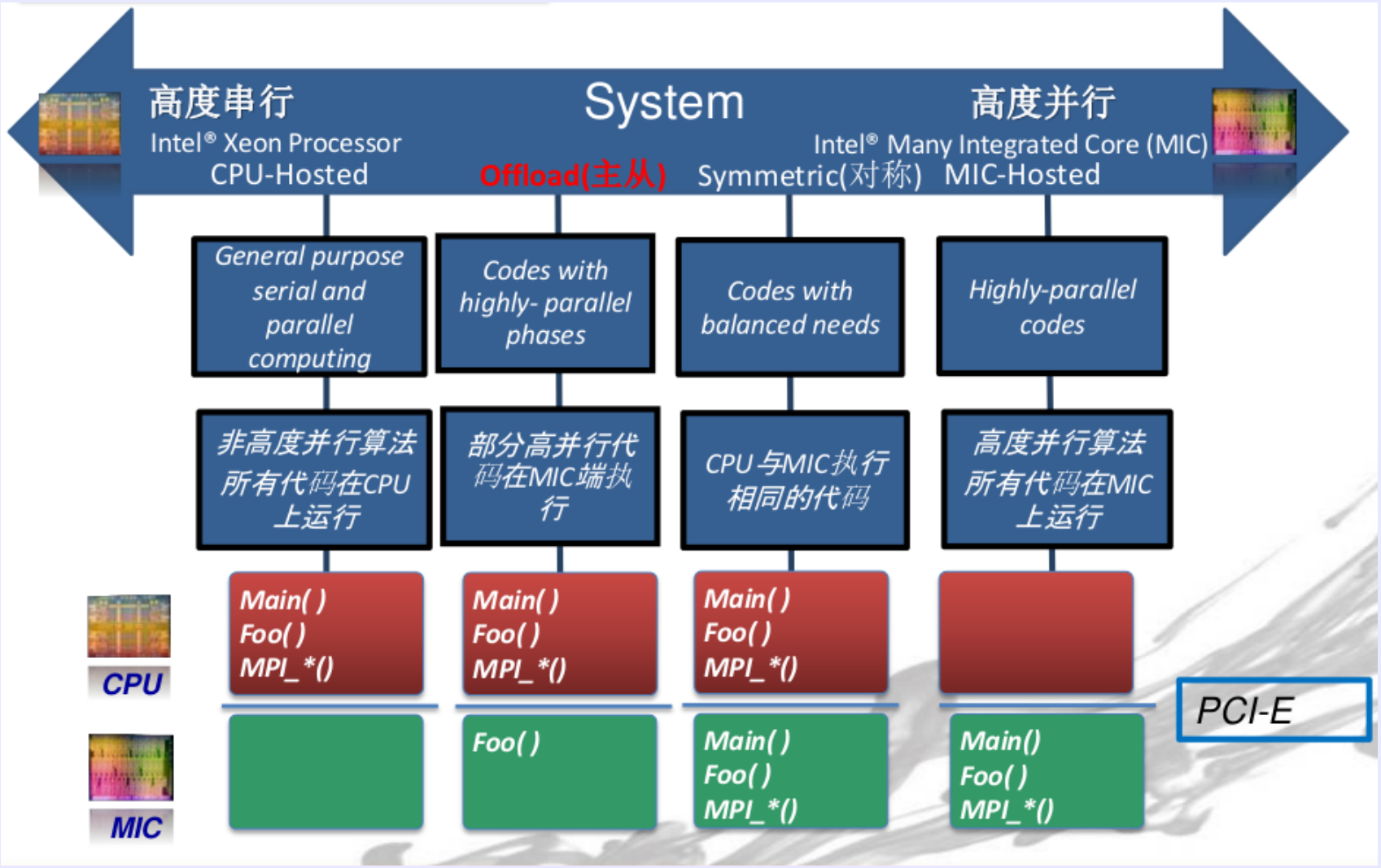
- 查看MIC卡相关信息micinfo,例如

```
[xxx@mic]$micinfo | grep Active | head -1 | awk '{ print $7 }'
```

- 查看MIC卡状态micsmc,例如

```
[xxx@mic]$micsmc -m | grep 'Total Memory' | head -1 | awk '{ print $
```

应用模式



[1]

[1] From: MIC众核技术解析及应用案例分析,吴庆,浪潮,2013

MIC常用编程模式

- native模式（MIC原生模式）
- offload模式(CPU为主MIC为辅模式)
- 对称模式（MPI模式）
  - 将MIC视为一个计算结点。
  - CPU和MIC同时参与计算。
  - CPU和MIC之间通过MPI交互。
  - 编译同native模式。

Native模式

MIC卡拥有自己的操作系统和IP地址，因此MIC支持卡上运行模式，即将程序和数据手工传输到MIC卡上，并直接在MIC卡上运行。在卡上运行的程序。该模式通常用于程序的整体算法是并行或部分代码可以CPU运行，但传输、同步开销过大的情况。

- OpenMP并行，不用改变任何代码。
- 在Host上的编译选项：`-mmic`
- 将程序和数据从Host上传Target：scp或NFS共享
- 登录MIC卡，设置`LD_LIBRARY_PATH`等变量。
- 直接运行程序。

offload模式运行过程

Offload模式适用于串行程序中还有部分高并行度的程序，类似OpenMP编程。程序运行经历如下过程：

- 程序启动，在CPU端运行主函数。
- 程序运行到"offload"代码段。
- 监测MIC是否存在，若在则用MIC版本代码，若部在则调用CPU版本。
- 如果第一次运行MIC程序，则驱动程序唤醒MIC卡。
- 在MIC卡上加载MIC版本代码。
- 驱动程序将数据从CPU内存拷贝到MIC端内存。
- CPU端程序暂停，MIC端程序开始执行。
- MIC端的程序运行完成后，将数据从MIC端拷贝到CPU端。
- MIC卡恢复低功耗状态，CPU端程序恢复运行。
- 程序结束。

Offload语法 I

MIC编程中最基本的语法就是offload语句，offload语句的作用为将程序和数据由CPU端传递到MIC端，并在MIC上执行，offload语法为：

- C/C++:

```
#pragma offload target(mic: id) in(...) out(...)
```

- Fortran:

```
!dec$ OFFLOAD  target(mic: id) in(...) out(...)
```

或：

```
!DIR$ OFFLOAD BEGIN  target(mic: id) in(...) out(...)
    ...
!DIR$ END OFFLOAD
```

target指定在哪块卡上运行，id代表设备编号，如果id等于-1，系统将自动选择一块计算设备（现在来说只有MIC卡），如果没有符合要求的设备（如只有CPU），程序将退出并报错；如果id大于等于0，程序将offload到相应的id号MIC上。关键字in()、out()、inout()、nocopy()的语法：

- 关键字可以没有或多个。有多个时，可以连续书写，之间用逗号或空格隔开。

Offload语法 II

- 括号内参数为变量名，可以是数组名、指针或普通变量，变量之间用逗号隔开。
- 变量为指针时，需要在变量后加上`:length(n)`,`n`为动态数组元素个数。
- 指针变量除加`length`关键字，还可以添加：

alloc_if()	如果条件为真则为变量开辟内存空间。
free_if()	如果条件为真则释放变量内存空间。
align()	参数为正整数,必须为2的正整数次幂，使得在设备端开辟的前述变量以该正整数长度对齐。
alloc()	参数是数组名，其含义为创建指定的部分内存空间。
into()	将数组从主机端拷贝到设备端的另外一个数组。只能在相同维度的数组间传递数据。

- 传输数组的一部分:

```
#pragma offload target(mic) in(p[5:20]:into(p2[10:25]))
```

变量或函数声明 I

利用offload模式在MIC卡上编程时，offload内调用的函数必须进行特别声明：

- C/C++:

```
__attribute__((target(mic))) 函数或变量声明
```

或

```
__declspec(target(mic)) 函数或变量声明
```

- Fortran:

```
!DIR$ attributes offload:target-name::routine名或变量名
```

编译运行

- 编译：
  - 编译器: Intel Compiler(icc、ifort)
  - CPU+MIC程序编译选项: DEFAULT 或`-offload-build`
  - CPU only程序编译选项：`-no-offload`
- 运行
  - 直接运行: `./prog`
  - 运行时查看信息：`env H_TRACE=1 ./prog`
  - 查看MIC运行时间：`env H_TIME=2 ./prog`

参考书

参考文献

1. 王恩东等,《MIC高性能计算编程指南》，中国水利水电出版社，2012
2. Intel Xeon Phi Coprocessor System Software Developers Guide