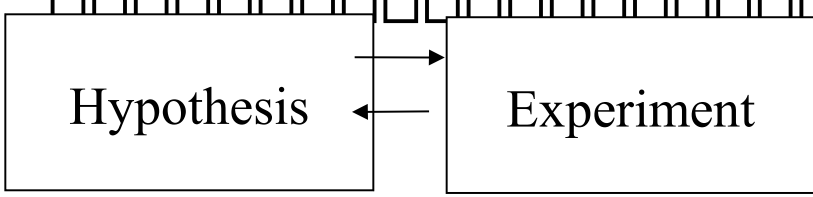
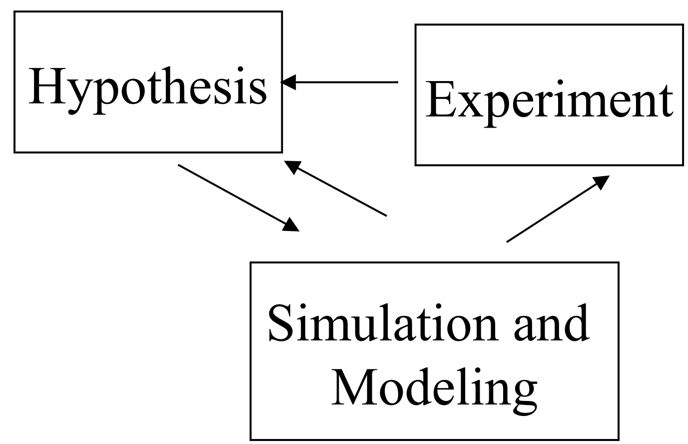
Session 1: Introduction to Intel Xeon Phi

[illegible]

- [illegible]
- [illegible]


Hypothesis
Experiment


Hypothesis
Experiment
Simulation and Modeling

Compute-Intensive Applications

- [illegible]
- [illegible]
- [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]

Grand Challenge Problem

- Drug design To develop drug that cures cancer or AIDS by blocking the action of HIV protease
- High-speed civil transport To develop supersonic jet through computational fluid dynamics running on super computer
- Fuel combustion Designing better engine model via chemical kinetics calculations to reveal fluid mechanical effects.
- Ocean modeling Large scale simulation of ocean activities and heat exchange with atmospherically flows

- Ozone depletion To study chemical and dynamical mechanisms controlling the ozone depletion process.
- Air pollution Simulate air quality model
- Protein structure design 3-D structural study of protein formation using MPP

Top500 Fastest Installed Computers

- www.top500.org
- [illegible] [illegible] top 500 supercomputer [illegible]
- [illegible] 1993 [illegible] 6 [illegible] [illegible] [illegible] [illegible]
- [illegible] Linpack Benchmark HPL
- [illegible] [illegible]

China's Tianhe-2 Supercomputer Takes No. 1 Ranking on 41st TOP500 List

2013-06-17 03:09:31+00:00

(0 comments)

MANNHEIM, Germany; BERKELEY, Calif.; and KNOXVILLE, Tenn.--Tianhe-2, a supercomputer developed by China's National University of Defense Technology, is the world's new No. 1 system with a performance of 33.86 petaflop/s on the Linpack benchma according to the 41stedition of the twice-yearlyTOP500 list of the world's most powerful supercomputers. The list was announced 17 during the opening session of the 2013 International Supercomputing Conference in Leipzig, Germany.

Tianhe-2, or Milky Way-2, will be deployed at the National Supercomputer Center in Guangzho, China, by the end of the year. The surprise appearance of Tianhe-2, two years ahead of the expected deployment, marks China's first return to the No. 1 position sinc November 2010, when Tianhe-1A was the top system. Tianhe-2 has 16,000 nodes, each with two Intel Xeon IvyBridge processors three Xeon Phi processors for a combined total of 3,120,000 computing cores.

Titan, a Cray XK7 system installed at the U.S. Department of Energy's (DOE) Oak Ridge National Laboratory and previously the N system, is now ranked No. 2. Titan achieved 17.59 petaflop/s on the Linpack benchmark using 261,632 of its NVIDIA K20x accelera

Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P

Site:	National University of Defense Technology
Manufacturer:	NUDT
Cores:	3,120,000
Linpack Performance (Rmax)	33,862.7 TFlop/s
Theoretical Peak (Rpeak)	54,902.4 TFlop/s
Power:	17,808.00 kW
Memory:	1,024,000 GB
Interconnect:	TH Express-2
Operating System:	Kylin Linux
Compiler:	icc
Math Library:	Intel MKL-11.0.0
MPI:	MPICH2 with a customized GLEX channel

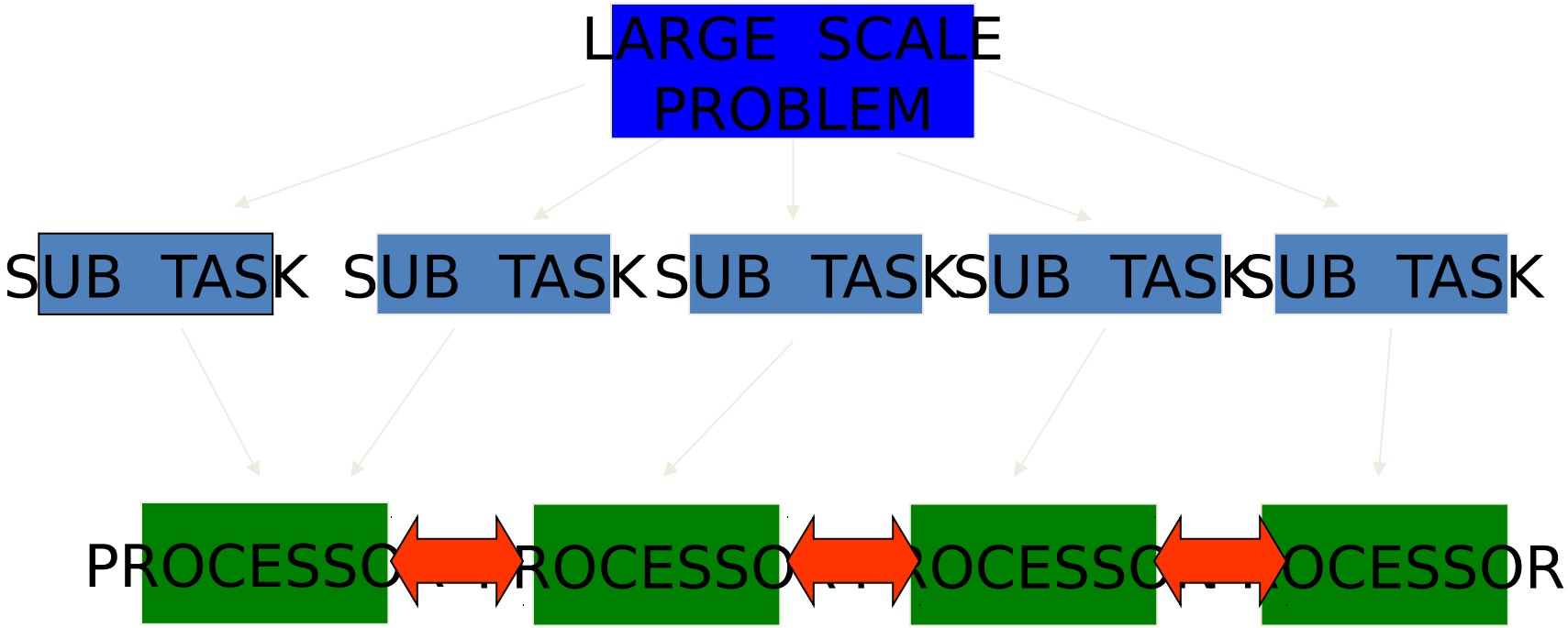
Ranking

List	Rank	System	Vendor	Total Cores	Rmax (TFlops)	Rpeak (TFlops)	Powe (kW)
06/2013	1	TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P	NUDT	3,120,000	33,862.7	54,902.4	17,808

Parallel Processing

- Solving large problem by breaking it in to a number of small problems, then solve them on multiple processors at the same time
- Real life example
  - building a house using many workers working on separate part
  - Assembly line in factory

Parallel Processing



Parallel Computing Model

- Functional Parallelism
  - Execute multiple function on many CPU
  - Limited parallelism but good for unstructured/dynamic problem

- Data Parallelism
  - Execute the same function on multiple CPU using different data
  - Highly parallel but limited to certain algorithm
  - GPU computing are based on Data Parallel Comcept

Parallel Computer

Parallel computer is a special computer that is designed for parallel processing

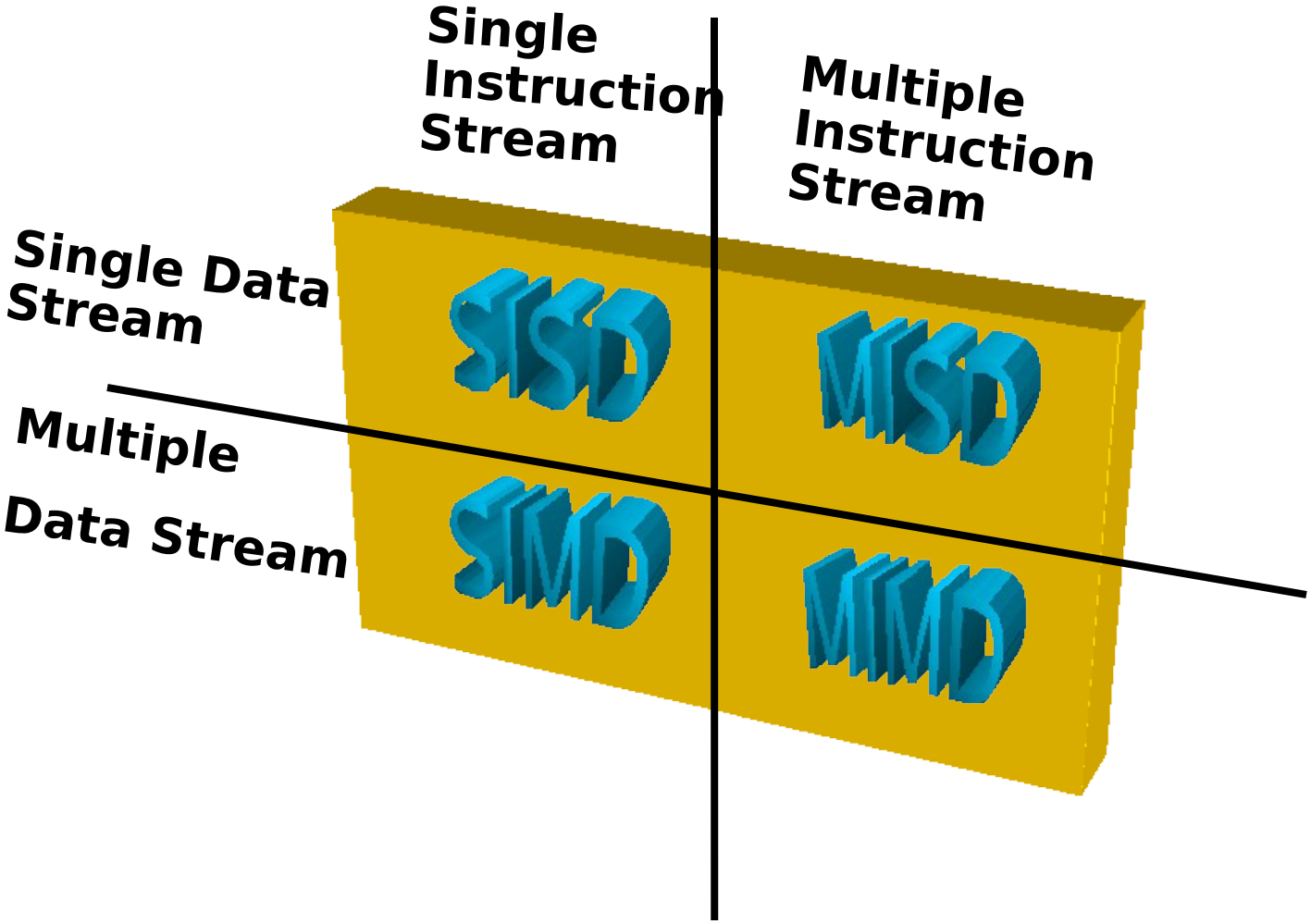
consists of

- Multiple processors
- High-speed Interconnection network that link these processors together
- Hardware and software that help coordinate computing tasks

Classification of Computer

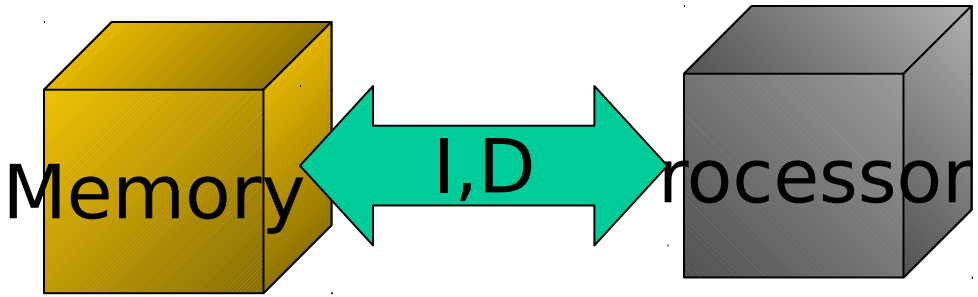
- Most widely used one is Flynn's Classification.
  - SISD (Single Instruction Stream, Single Data Stream)
  - SIMD (Single Instruction Stream, Multiple Data Stream)
  - MISD (Multiple Instruction Stream, Multiple Data Stream)
  - MIMD (Multiple Instruction Stream, Multiple Data Stream

Flynn's Classification



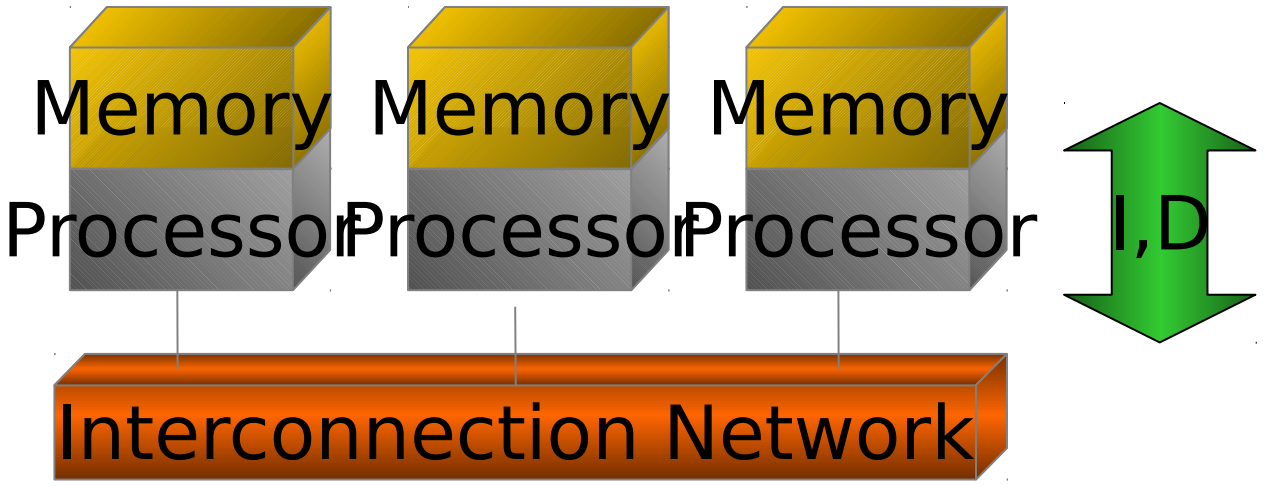
SISD

- Computer that consists of a single instruction stream and single data stream transfer from memory to processor
- Normal uni-processor system (Like Intel Pentium on PC)



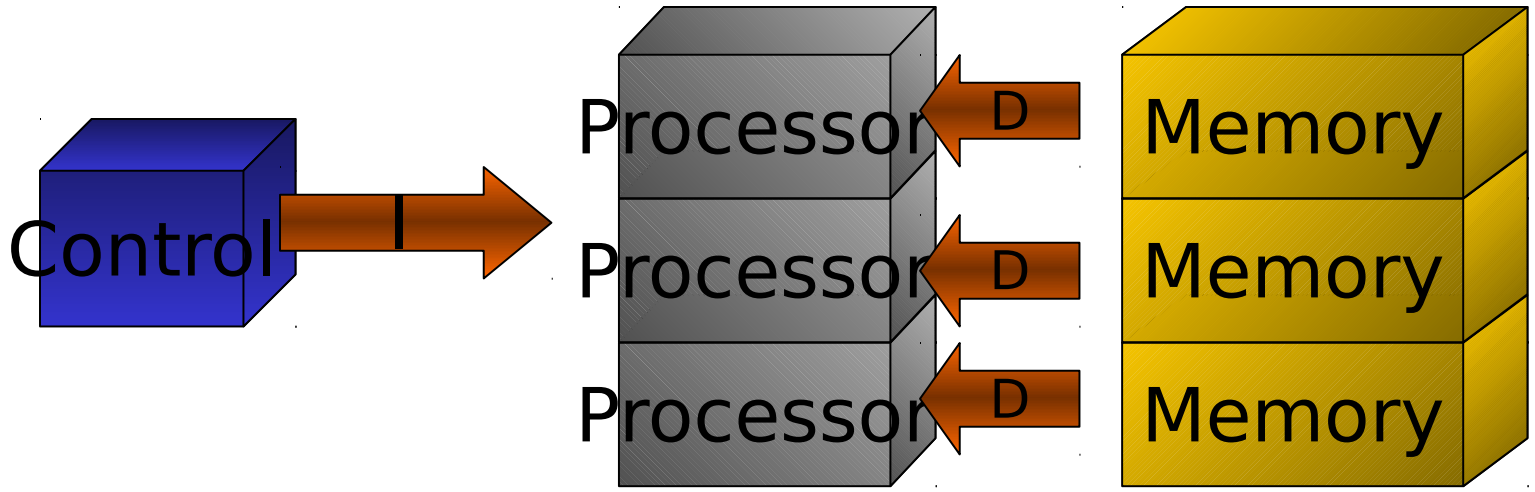
MIMD

- Multiple processor execute different program on different data
- Most commercial parallel supercomputer use this architecture (IBMSP, SGI Origin, HP CONVEX SPP)



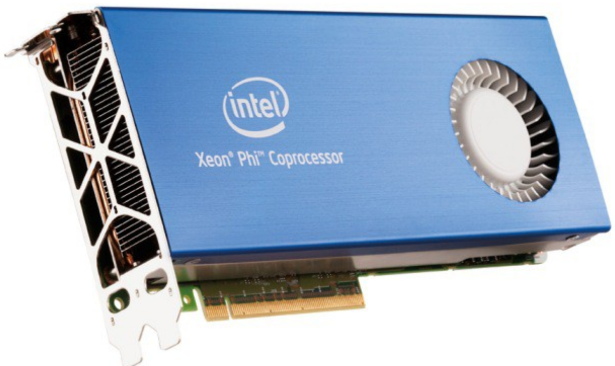
SIMD

- Multiple processing element
- Single control unit , execute same instruction on large data set
- Exploit data parallelism in code



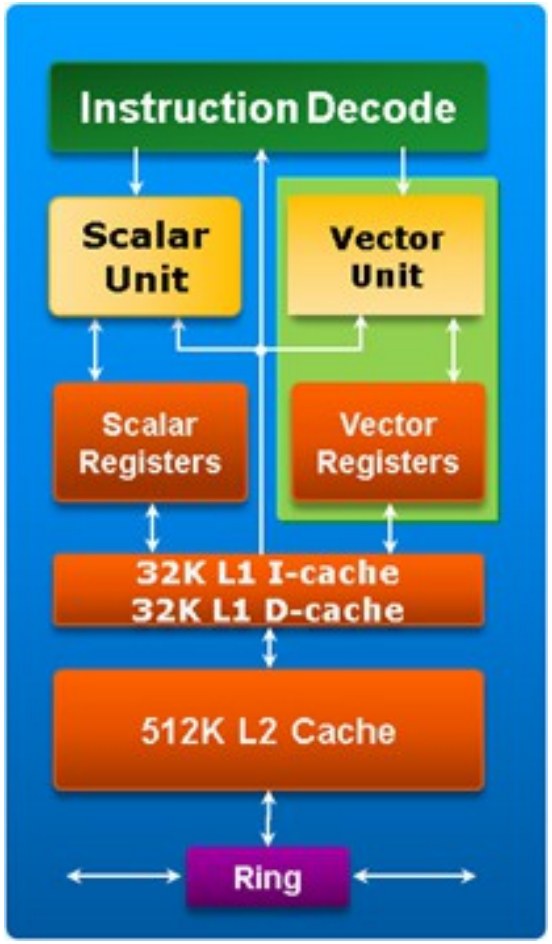
Intel Xeon Phi Overview

Intel Xeon Phi Coprocessor



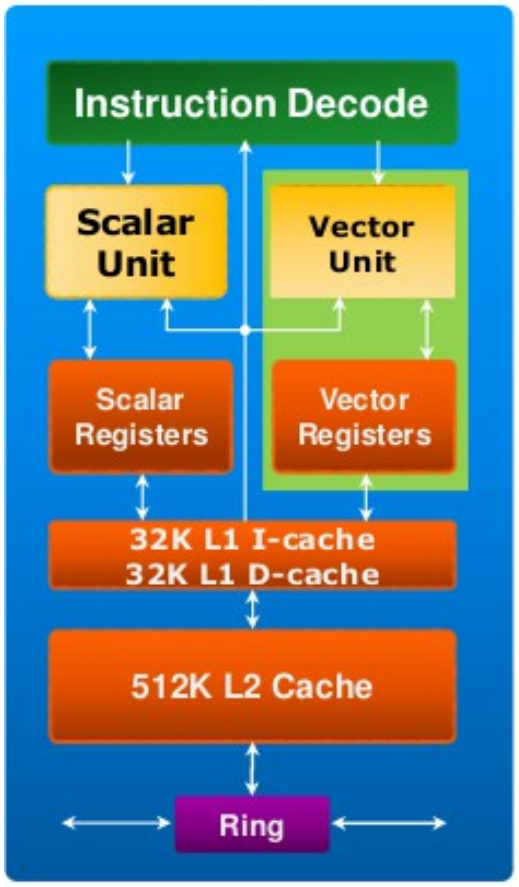
- Xeon Phi [illegible] [illegible] [illegible] [illegible] [illegible] Many core processors [illegible] CPU [illegible] [illegible] [illegible]
- Xeon Phi [illegible] x86 [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

Intel Xeon Phi Overview



- 61 cores, Ring interconnect
- 64-bit addressing
- Scalar Unit
- Vector Unit – 512-bit register

VPU : Vector Processing Unit



- Xeon Phi come with all new vector unit
  - 512-bit SIMD Instruction
    - Different from SSE, MMX, or AVX
  - 32 512-bit wide vector registers
    - Hold 16 singles or 8 doubles per register

Intel Xeon Phi Overview

- □□ uOS □□□□ Linux □□□□□□□□□□ □□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□ PCI Express
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
  - Native Mode
    - Login □□□□□□□□□□□□□□□ □□□□□□□□□□□□ □□□□□□□
  - Offload Mode

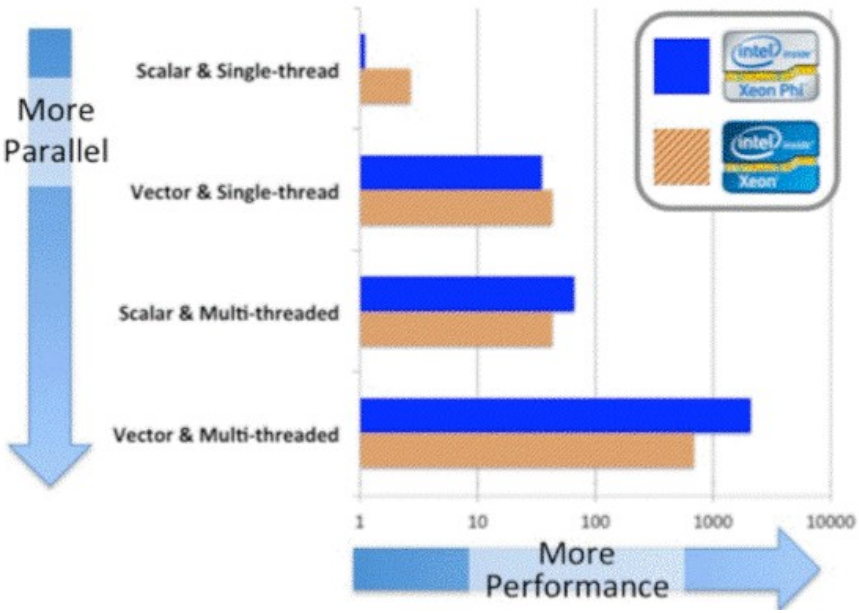
Table 1. Intel® Xeon Phi™ Coprocessor Family Overview

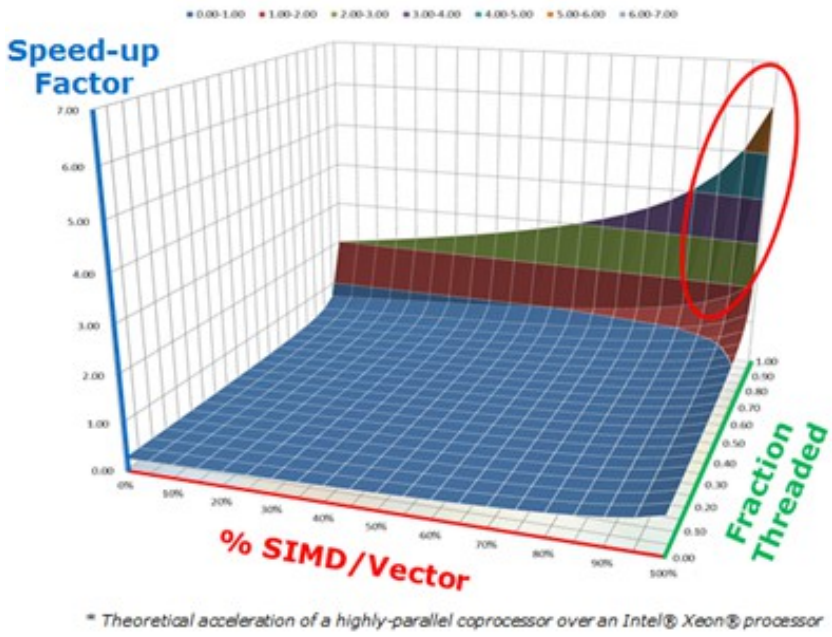
FEATURES	DETAILS	BENEFITS
Intel® Many Integrated Cores (MIC) architecture	▪ Up to 61 cores, 244 threads, and 16 GB of GDDR5 memory (352 GB/s bandwidth) per coprocessor ▪ Double-wide (256-bit) vector engines and 512-bit SIMD instructions ▪ Ideal for highly-parallel, vector-intensive, and memory-bound code	Up to 1.2 teraflops of double-precision performance per coprocessor[1]
Familiar Intel® architecture programming model	Developers can: ▪ Use familiar methods and tools, including the latest Intel® Software Development products ▪ Maintain a common code base for Intel® Xeon® processors and Intel® Xeon Phi™ coprocessors	Simplified development and dual-benefit for performance optimization
Linux* hosting capability	Run each coprocessor under the host operating system or as an independent server running Red Hat Enterprise Linux* 6.x or SuSE Linux* 12+	Exceptional execution flexibility
IP Addressable	Supports standard clustering models.	Simple scaling
Industry-leading silicon technology	▪ Intel 22 nm technology with 3D Tri-Gate transistors ▪ Power envelopes as low as 225 Watts per coprocessor	Exceptional compute density and energy efficiency
Flexible form factors	▪ Standard x16 PCIe* cards (with active, passive, or no thermal solution) and a unique dense form factor (DFF) for more customized integration ▪ Use up to 8 coprocessors per host server	Flexible integration and scalability
Flexible Execution Models	▪ Multicore only – MAIN() runs on host processor ▪ Multicore Hosted with Manycore Offload – MAIN() runs on host processor and select routines are executed on the coprocessor ▪ Symmetric execution – MAIN() runs symmetrically on processor and coprocessor ▪ Manycore only – Boot from host processor, MAIN() runs on coprocessor	Best flexibility for optimizing workload performance

Table 2. Intel® Xeon Phi™ Product Family Specifications

PRODUCT NUMBER	FORM FACTOR &, THERMAL SOLUTION[4]	BOARD TDP (WATTS)	NUMBER OF CORES	FREQUENCY (GHz)	PEAK DOUBLE PRECISION PERFORMANCE (GFLOP)	PEAK MEMORY BANDWIDTH (GB/s)	MEMORY CAPACITY (GB)	INTEL® TURBO BOOST TECHNOLOGY
3120P	PCIe, Passive	300	57	1.1	1003	240	6	N/A
3120A	PCIe, Active	300	57	1.1	1003	240	6	N/A
5110P	PCIe, Passive	225	60	1.053	1011	320	8	N/A
5120D	Dense form factor, None	245	60	1.053	1011	352	8	N/A
7110P	PCIe, Passive	300	61	1.238	1208	352	16	Peak turbo frequency: 1.33 GHz
7120X	PCIe, None	300	61	1.238	1208	352	16	

Application □□□□□□□□□





Programming Model

Programming Model

- [illegible] Model [illegible] Intel [illegible] [illegible] [illegible]
  - Intel Math Kernel Library (Intel MKL)
  - OpenMP
  - Intel Cilk Plus
  - Intel Threading Building Blocks (Intel TBB)
- [illegible]

Intel MKL

- Intel Math Kernel Library
- Library □□□□□□□□□□□□□□□□□□□□□□
- C/C++ □□□ Fortran
- □□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□ parallel □□□
- □□□□□□□□□□□□□ □□□□□□□□ optimize □□□□□□□□ device □□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□ □□□□□□ □□□□□□

```
alpha = 1.0, beta = 0.0;
sqemm(“N”, “N”, &N, &N, &N, &alpha, A, &N, B, &N,
   &beta, C, &N);
```

OpenMP

- Compiler directive in C/C++
- □□□□□□□□□□□□□□□□□□□□□□□□□□ Parallel □□□□□□□□ □□□ □□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□ pragma □□□□
- □□□□□□□□□□□□ task parallel □□□ data parallel

```
#pragma omp parallel for
for(i=0 ;i<N; ++i)
    for(j=0; j<N; ++j)
        for(k=0; k<N; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

Intel Cilk Plus

- Extension □□□ C/C++
- task parallelism
  - _Cilk_spawn , _Cilk_sync
- loop parallelism
  - _Cilk_for

```
_Cilk_for(i=0 ;i<N; ++i)
    for(j=0; j<N; ++j)
        for(k=0; k<N; ++k)
            C[i][j] += A[i][k] * B[k][j];
```

```
for(i=0 ;i<N; ++i)
    for(j=0; j<N; ++j)
        a[i][:] += b[i][k] * c[k][:];
```

Intel TBB

- Template library support C/C++
- □□ data-structure □□□ pattern □□□□□□□□□ □□ parallel □□□□□□

```
class Multiply {
public:
    void operator()(blocked_range<int> r) const {
        for (int i = r.begin(); i != r.end(); ++i)
            for (int j = 0; j < size; ++j)
                for (int k = 0; k < size; ++k)
                    c[i][j] += a[i][k] * b[k][j];
    }
};
…
parallel_for(blocked_range<int>(0,N), Multiply());
```

Building and Running Native Application

Coprocessor Native Execution

- Xeon Phi Coprocessor [illegible] [illegible] [illegible] Coprocessor [illegible]
- [illegible] Parallel Program [illegible] [illegible] Coprocessor
- [illegible] Coprocessor [illegible] CPU [illegible] [illegible]

Native Architecture

- Coprocessor [illegible] [illegible] Linux [illegible]
- [illegible] (CPU, Memory, Resources …)
- [illegible] PCI Express

Why Native Execution ?

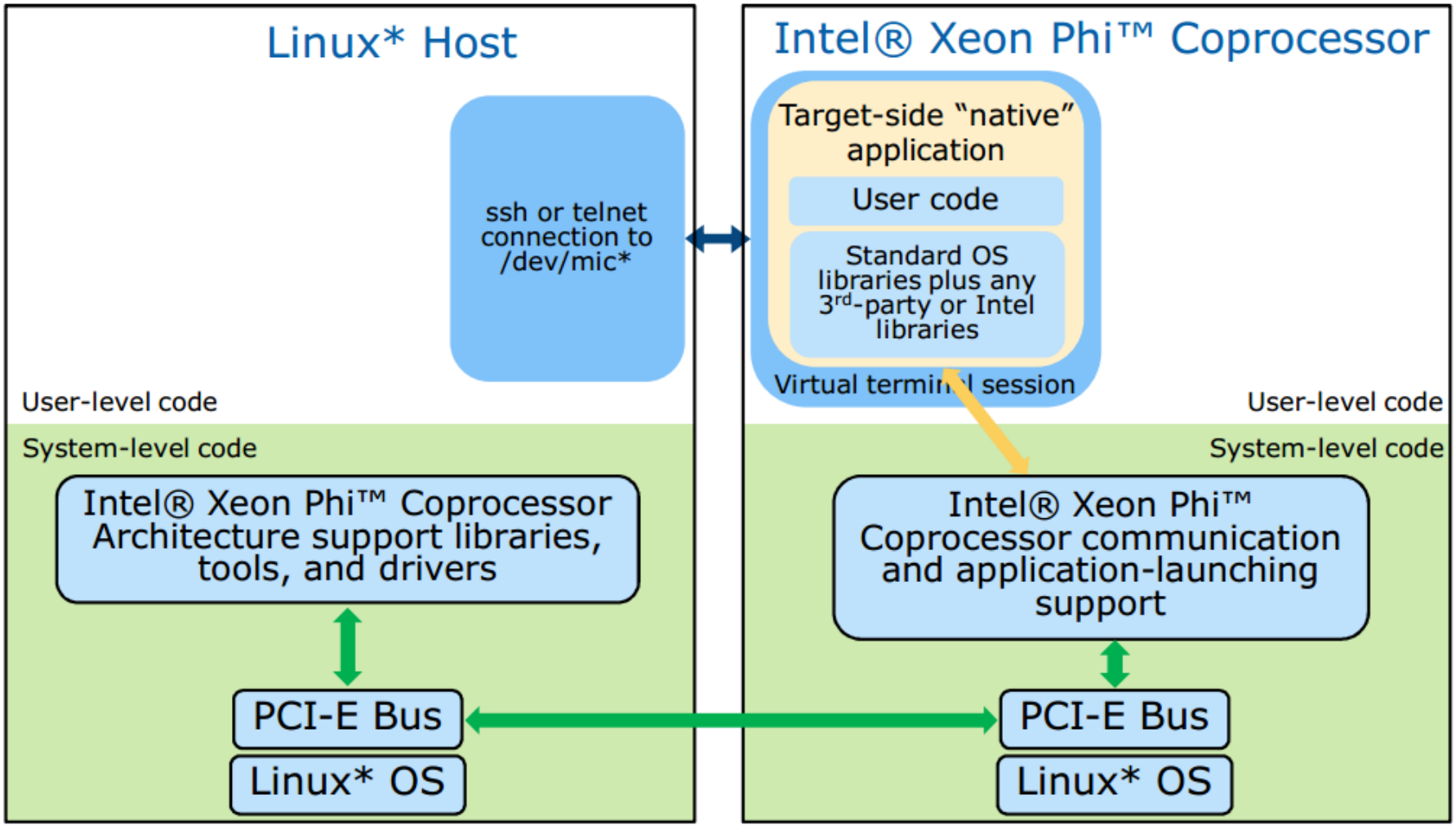
- [illegible] Parallel Program [illegible]
- [illegible] (OpenMP, Intel Cilk Plus, Intel Math Kernel Library, Intel TBB, MPI)
- [illegible] Coprocessor [illegible] Coprocessor
- [illegible] MPI [illegible] Coprocessor [illegible]

Why NOT Native Execution ?

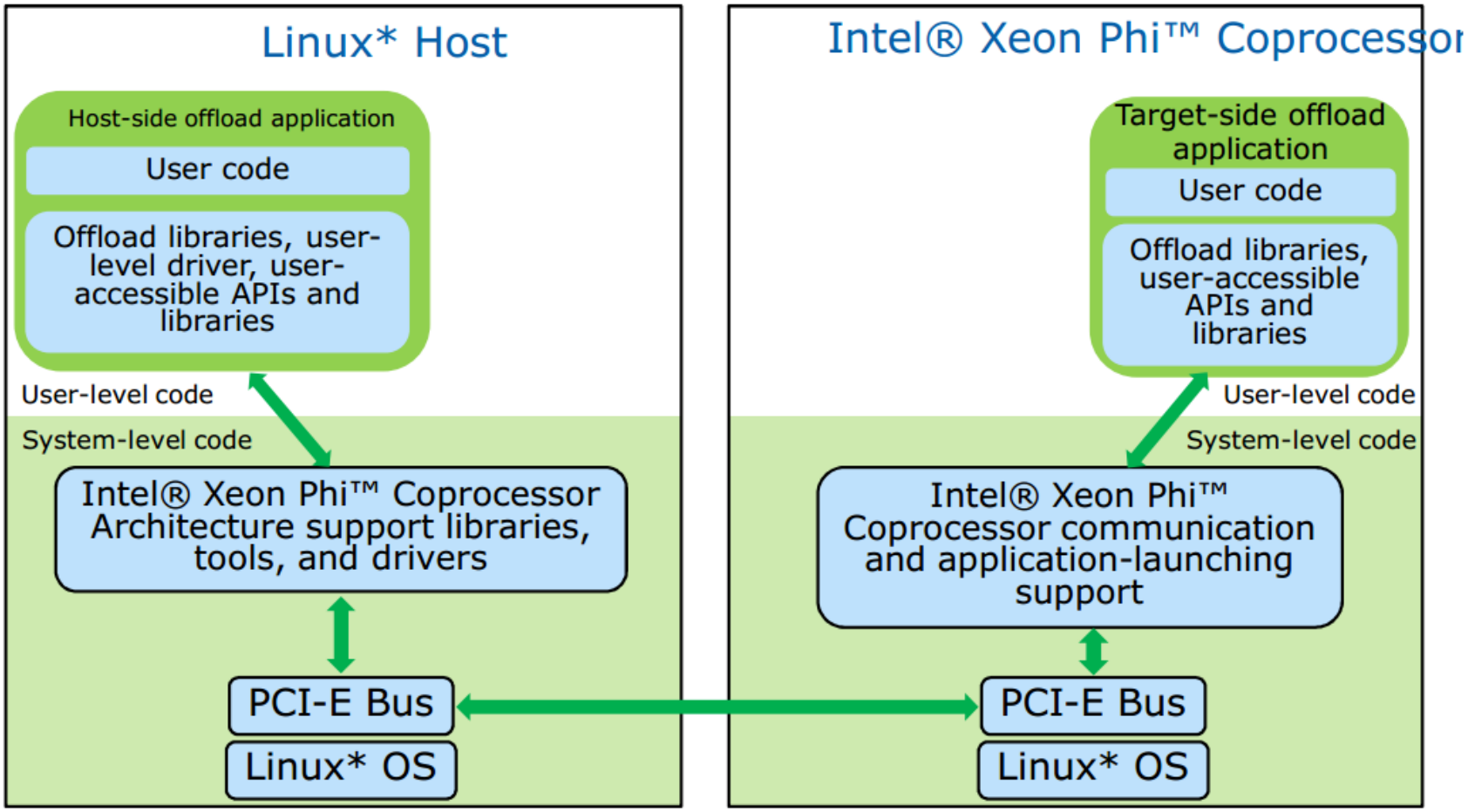
- [illegible]
  Coprocessor [illegible]
- Coprocessor [illegible] Single Thread Performance [illegible]
  [illegible]
- Coprocessor [illegible]

[illegible] Offload [illegible]
[illegible]

Coprocessor Architecture: Native



Coprocessor Architecture : Offload



Coprocessor Communication

- □□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□ Coprocessor □□□□□□□□□□□□□□□□ □□□□□□□□□□ Coprocessor □□□□□□□□□□ TCP/IP Bridge □□□□□□□ □□□□□□□□□□□□□□□□□□□□□□ IP Address □□□
- □□□□□□□□□□□□□□□□□□□□ Coprocessor □□□□ IP Address □□□□□□□□□□□ SSH □□□□ Telnet □□□□□□ □□□ OS □□□□□□□□□□□□
- □□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□ Linux □□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□ Coprocessor □□□

Coprocessor Configuration

- [illegible] [illegible] micctrl [illegible] Kernel Module [illegible] Coprocessor [illegible] IP Address [illegible] Coprocessor [illegible]
- [illegible] Coprocessor [illegible] File System [illegible]
- [illegible] Loader [illegible] Dynamic Link [illegible] Linux [illegible] Shared Library [illegible] LD_LIBRARY_PATH

Execution on Coprocessor

[illegible]

1. [illegible] binary [illegible] Coprocessor [illegible] scp [illegible] Secure Shell [illegible] Coprocessor [illegible]
2. [illegible] **micnativeloadex** [illegible] binary [illegible]

Code Compilation for Coprocessor

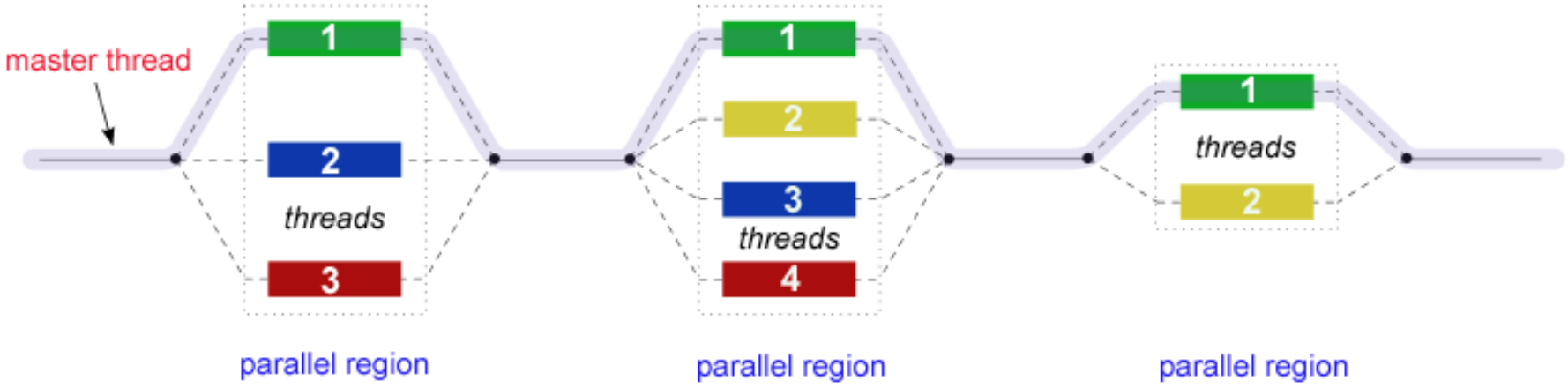
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Native Mode □□□□ □□□ Intel Compiler □□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□ Coprocessor
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C, C++, Fortran □□□□□□□□□□□□□□□□ OpenMP, Intel Cilk Plus, Intel TBB, Intel MKL □□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□ **–mmic** □□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□ Cross Compile □□□□□ Target Architecture □□□ Coprocessor

Introduction to OpenMP

What is OpenMP ?

- [illegible] shared memory [illegible] compiler [illegible] code [illegible]
- [illegible] compiler directive [illegible] compiler
- [illegible] loop [illegible]

Fork-join model



- fork □□□□□□□□□□□□ parallel region (□□□□□□□□ □□□□□□□□□□□□□□□ parallel directive)
- join □□□□□□□□□□□ parallel region (□□□□□□□□□□ □□□□□□□□□□□ parallel directive)

parallel directive

- □□□□□ thread □□□□□□ □□□□□□□□ thread □□□□□□□□□□□□□□□□□□□□□□□□ parallel directive

parallel construct syntax

```
#pragma omp parallel [clause[ [, ]clause] ...] new-line
structured-block
```

```
#pragma omp parallel
{
    cout << “Hello World\n”;
}
```

OpenMP's "Hello World“

Name this little "Hello World" program hello.c:

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

int main(int argc, char *argv[]) {
  int iam = 0, np = 1;

  #pragma omp parallel default(shared) private(iam, np)
  {
    #if defined (_OPENMP)
      np = omp_get_num_threads();
      iam = omp_get_thread_num();
    #endif
    printf("Hello from thread %d out of %d\n", iam, np);
  }
}
```

Reference: http://www.slac.stanford.edu/comp/unix/farm/openmp.html

Compiling and Linking OpenMP Programs

icc -openmp   -o hello hello.c

Running OpenMP Programs

export OMP_NUM_THREADS=4
Now you can start your program and it will execute with 4 parallel threads:

./hello
Hello from thread 0 out of 4
Hello from thread 1 out of 4
Hello from thread 2 out of 4
Hello from thread 3 out of 4

for directive

- [illegible] parallel directive [illegible]
- [illegible] thread [illegible]

for construct syntax

```
#pragma omp for [clause[[,] clause] ... ] new-line
for-loops
```

```
#pragma omp parallel
{
    #pragma omp for
    for(i = 0; i < 10000; i++) {
        res[i] = sin(vec_a[i]);
    }
}
```

sections & section directive

- section directive □□□□□□□□□□□□□□□ sections directive □□□ sections directive □□□□□□□□□□□□□□□ parallel directive
- □□□□□ section □□□□□□□□□□□□□□□□□□□□ thread □□□□□□□□□□□□

```
#pragma omp parallel
{
#pragma omp sections
{
    #pragma omp section
    f1(x);
    #pragma omp section
    f2(x);
}
}
```

sections & section directive

sections construct syntax

```
#pragma omp sections [clause[[,] clause] ...] new-line
{
    [#pragma omp section new-line]
        structured-block
    [#pragma omp section new-line
        structured-block ]
    ...
}
```

combine directive – parallel for

```
#pragma omp parallel
{
    #pragma omp for
    for(i = 0; i < 10000; i++) {
        res[i] = sin(vec_a[i]);
    }
}
```

```
#pragma omp parallel for
for(i = 0; i < 10000; i++) {
    res[i] = sin(vec_a[i]);
}
```

combine directive – parallel sections

```
#pragma omp parallel
{
#pragma omp sections
{
    #pragma omp section
    f1(x);
    #pragma omp section
    f2(x);
}
}
```

```
#pragma omp parallel sections
{
    #pragma omp section
    f1(x);
    #pragma omp section
    f2(x);
}
```

reduction clause

- reduction □□□□□□□□□□□□
- operator □□□□□□□□□□□ + , * , - , &, | , ^ , && , || (OpenMP 3.1 □□□□□ min □□□ max)

reduction clause syntax

```
reduction(operator:variable-list)
```

```
#pragma omp parallel for reduction(+:sum)
for(i = 0; i < 10000; i++) {
    sum += A[i] * B[i];
}
```

public and private clause

- default □□□□□□□□□□□□□□ (□□□□□□□□□□□□□□ □□□□□□□□□□□□□) □□□□ public

public clause syntax

```
public(variable-list)
```

private clause syntax

```
private(variable-list)
```

```
#pragma omp parallel for \
    reduction(+:sum) private(x)
for (i=1;i<= num_steps; i++) {
    x = (i-0.5)*step;
    sum = sum + 4.0/(1.0+x*x);
}
```

single and master directive

- □□□□□□□□□□□□□□□□□□□□□□□□□□□□ thread □□□□□□□□□□□□□□□□
- single - □□□□ thread □□□□□□□□□□□□□□□
- master - □□□□□□□□ master thread □□□□□□□□

single construct syntax

```
#pragma omp single[clause[[,] clause] ...] new-line
    structured-block
```

master construct syntax

```
#pragma omp master[clause[[,] clause] ...] new-line
    structured-block
```

single and master directive

```
#pragma omp parallel num_threads(4)
{
    #pragma omp for
    // initialize data

    #pragma omp single
    read_input_from_file();

    #pragma omp for
    // parallel work

    #pragma omp master
    print_output();
}
```

atomic and critical directive

- □□□□□□□□□□□□□□□□□□ parallel region □□□ □□□□□□□□□□□□□□□□□□□□ thread

atomic construct syntax

```
#pragma omp atomic new-line
    expression-stmt
```

critical construct syntax

```
#pragma omp critical [ (name) ] new-line
    structured-block
```

atomic and critical directive

```
#pragma omp parallel for
for (i = 1; i < SIZE; i++) {
    if (a[i] > max) {
        #pragma omp critical
        {
            if (a[i] > max) max = a[i];
        }
    }
}
```

Labs

- Lab 1: Introduction
- Lab 2: Compile & Execution
- Lab 3: Miscellaneous

Login to Phi Server

- ssh client putty linux ssh
- phi.cpe.ku.ac.th
-

```
login as: pu
pu@phi.cpe.ku.ac.th's password:
Last login: Mon Jul  1 16:41:56 2013 from
158.108.180.184
[pu@phi ~]$
```

Lab 1: Introduction

1. □□□□□□□□□ micinfo □□□□□□□□□□□□□□□□□□□□ Coprocessor

2. □□□□□□□□□ ping mic0 □□□ ping mic1 □□□□□□ IP Address □□□ □□□□□□□□□□□□□

3. □□□□□□□□□ ssh mic0 □□□□□□□□□□□ Coprocessor

4. □□□□□□□□□□□ Coprocessor □□□□ □□□□□□□□ hostname □□□ top □□□□□□□□□□□□□□□□□ Coprocessor

5. □□□□□□□□□ cat /proc/cpuinfo □□□□□□□□□□□□□□□□□□□ CPU

6. □□□□□□ Coprocessor □□□□□□□□□ exit

```
[pu@phi ~]$ ping mic0
PING XeonPhi-mic0 (172.31.1.1) 56(84) bytes of data.
64 bytes from XeonPhi-mic0 (172.31.1.1): icmp_seq=1 ttl=64
time=37.5 ms
64 bytes from XeonPhi-mic0 (172.31.1.1): icmp_seq=2 ttl=64
time=0.687 ms
64 bytes from XeonPhi-mic0 (172.31.1.1): icmp_seq=3 ttl=64
time=0.321 ms

--- XeonPhi-mic0 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2438ms
rtt min/avg/max/mdev = 0.321/12.839/37.511/17.446 ms

[pu@phi ~]$ ping mic1
PING XeonPhi-mic1 (172.31.2.1) 56(84) bytes of data.
64 bytes from XeonPhi-mic1 (172.31.2.1): icmp_seq=1 ttl=64
time=0.734 ms
64 bytes from XeonPhi-mic1 (172.31.2.1): icmp_seq=2 ttl=64
time=0.672 ms
64 bytes from XeonPhi-mic1 (172.31.2.1): icmp_seq=3 ttl=64
time=0.309 ms

--- XeonPhi-mic1 ping statistics ---
```

Lab 2: Compile and Execute

- [illegible] lab1/omp_offload_native.cpp [illegible] OpenMP [illegible]

- [illegible] intel compiler [illegible]

% icc –mmic –vec-report3 –openmp omp_offload_native.cpp

- [illegible] a.out

Lab 2 (continue)

- □□□□□□□□□□ a.out □□□□□□□□ Coprocessor □□□□ □□□□□□ scp a.out mic0:~
- Secure Shell □□□□□□□□ Coprocessor □□□□□□□□□ ssh mic0
- □□□□□□□□□□ a.out

  ./a.out 2048 64

  □□□□□□□□□□□□□□□□□□□□

□□□□□□□

```
[pu@phi Session 1 lab source codes]$ icc -mmic -vec-report3
-openmp omp_offload_native.cpp

[pu@phi Session 1 lab source codes]$ scp a.out mic0:~
a.out                                                    100%
36KB  35.7KB/s   00:00
[pu@phi Session 1 lab source codes]$
[pu@phi Session 1 lab source codes]$ ssh mic0
[pu@XeonPhi-mic0 pu]$
[pu@XeonPhi-mic0 pu]$ ./a.out 2048 64
matrix multiplication running with 244 threads
        avg time : 2.624381 s
[pu@XeonPhi-mic0 pu]$
```

Lab 3 : Miscellaneous

- □□□□□□□□□□ Threads □□□□□□□□□□□□□□□□ export OMP_NUM_THREADS=244
- □□□□□□□□□□□□□□□□□□□□ □□□□□□□□ NFS Server □□□ □□□□□□□□□□□□□□□□□□□□ share □□□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□ /share □□□□□□□□□□□□□□□□□□□□ mic
- □□□□□□□□□ Library □□□□□□□□□□□□□□□□□□□□ □□□ □□□□□□□□□□□□□□□□ binary □□□□□□□□ □□□□□□□□ □□□□□□□□□ LD_LIBRARY_PATH □□□□□□□□□□ Library □□□□□□□□□□□□□□□□

Introduction to Intel Cilk Plus

Intel Cilk Plus

- Cilk [illegible] MIT [illegible] parallel
- [illegible] Intel [illegible] Intel Cilk Plus [illegible] 2009
- work-stealing scheduler
- [illegible] Array Notation (A[0:n] = B[0:n] * C[0:n])

- [illegible] http://cilkplus.org/

Basic Intel Cilk Plus

- □□□□□□□□□□□□□□□□□□□ C/C++
- □□□□□□□□□□□□□□□□ 3 □□□□□□
  - _Cilk_for
  - _Cilk_spawn
  - _Cilk_sync

Data Parallelism

- _Cilk_for [illegible] loop [illegible] [illegible] [illegible] parallel [illegible]

```
_Cilk_for (int i = 0; i < 40000; i++) {
    A[i] = foo(B[i]);
}
```

Task Parallelism

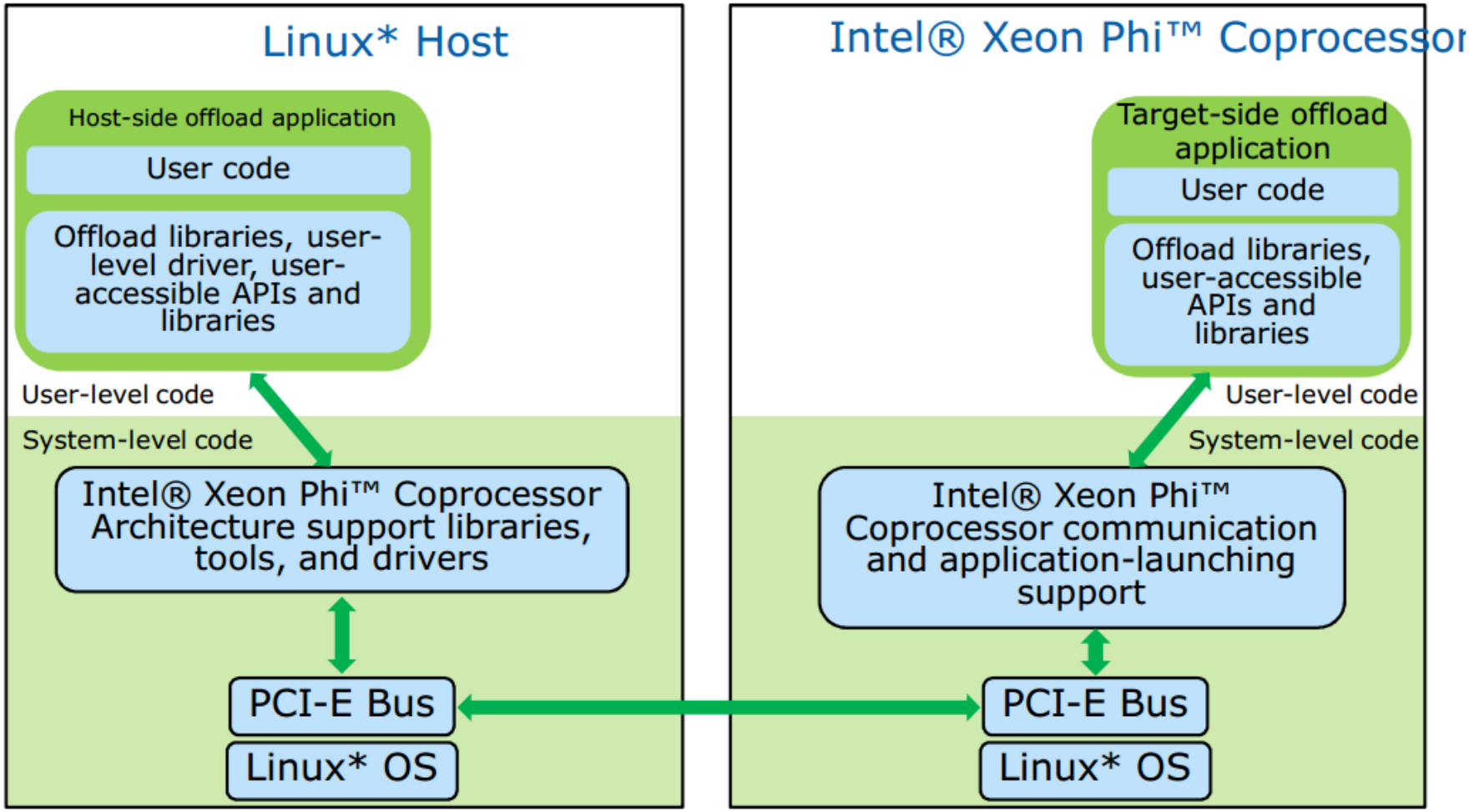
- _Cilk_spawn □□□□□□□□□□□ task □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- _Cilk_sync □□□□□□ synchronize □□□□□□□ □□□□□□□□□ spawn function □□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□□□□□□□

```
int fib(int n) {
    if(n <= 2) return n;
    else {
        int x, y;
        x = _Cilk_spawn fib(n-1);
        y = fib(n-2);
        _Cilk_sync;
        return x+y;
    }
}
```

Offload Concept

- [illegible] coprocessor
- [illegible] offload
- [illegible] compiler [illegible] runtime system [illegible]
- [illegible]
  1. Pragma offload
  2. Intel Cilk Plus

Offload Concept



Pragma offload example

```
#pragma offload target(mic) \
    in(b:length(count)) out(a:length(count))
{
     #pragma omp parallel for
     for(i=0; i<count; ++i)
          a[i] = b[i] * c + d;
}
```

- target(mic) – offload □□□□□ coprocessor
- in(b:length(count)) – □□□□□□□□□□ b □□□□ count □□□□□ mic
- out(a:length(count)) – □□□□□□□□□□ a □□□□ count □□□□□□□□□□ host

Pragma offload directives

C/C++ syntax	□□□□□□□□
#pragma offload <clause> <statement>	□□□□□□ <statement> □□□□□□ □□□□□□□□□□□□ offload □□□□□□ □□□□□ coprocessor □□□□□□ <clause> □□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□ offload
__attribute__((target(mic)))	□□□□□□□□□□□□□□□□□□□□□□□□□□□ definition □□□□□□□□□□ coprocessor
#pragma offload_attribute(push,target(mic)) … #pragma offload_attribute(pop)	□□□□□□□□□□□□□□□□□□□□□□□□□□□ #pragma offload □□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□

Pragma offload clauses

clause	□□□□□□□□
target(name[:card_numbe r])	□□□□□□□□□□□□□□□□□□□□□□□□□□□□ (card_number □□□□□□□□□□□ 0)
if(condition)	□□□□□□□□□□□□□□□□ (□□□□ boolean □□□ false □□□□□□□□□□□□□□□□□□□□ host)
in(var-list [:modifiers])	□□□□□□□□□□□□□□□□□□□□□□□□ host □□ □□□ coprocessor □□□□□□□ offload □□□□□□□□□
out(var-list [:modifiers])	□□□□□□□□□□□□□□□□□□□□□□□□ coprocessor □□□□□ host □□□□□□□□□□ offload □□□□□□□□□□□
inout(var-list [:modifiers])	□□□□□□□□□□□□□□□□□□□□□□□□ host □□ □□□ coprocessor □□□□□□□ offload □□□□□□□□□ □□□□□□□□□□ coprocessor □□□□□ host □□□□□□□□□□ offload □□□□□□□□□□□

Modifiers

modifiers	□□□□□□□□
length(element-count-expr)	□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
alloc_if(condition)	□□□ condition □□□□ true □□□□□□□□ allocate □□ coprocessor □□□□□□ □□□□□□□□□□
free_if(condition)	□□□ condition □□□□ true □□□□□□□□ deallocate □□ coprocessor □□□□□□ □□□□□□□□□□
align(expression)	□□□□□□□□□□□□ memory alignment □□□□□□□□□□□□□□□□ target

Offload with pragma offload

```
__attribute__((target(mic)))
int global_var = 45;

__attribute__((target(mic)))
int f(int x) { return (x*x+1); }

int main() {
    …
    #pragma offload target(mic) out(a:length(count))
    #pragma omp parallel for
    for(i=0; i<count; ++i)
        a[i] = f(i) + global_var;
    …
}
```

Conceptual Transformation

Source Code

```
main()
{
    f();
}
```

```
f()
{
   #pragma
offload
       a = b + g();
}
```

```
__attribute__((targ
et()mic)) g()
{
}
```

Host Program

```
main()
{
copy_code_to_coproc();
    f();
unload_coproc();
}
```

```
f() {
   if (coproc_available()) {
      send_data_to_coproc();
      start f_part_coproc();
      receive_data_from_coproc();
   } else
      f_part_host();
}
```

```
f_part_host()
{ a = b + g(); }
```

```
g() { ... }
```

Intel Xeon Phi Program

```
f_part_coproc()
{ a = b +
g_coproc(); }
```

```
g_coproc() { ... }
```

Pragma Offload Summary

- [illegible] offload
- [illegible] input [illegible] output [illegible]
- [illegible] function [illegible] global variable [illegible] offload code [illegible] __attribute__((target(mic)))

Offload with Intel Cilk Plus

- [illegible] _Cilk_offload [illegible] offload
- [illegible] _Cilk_shared [illegible] coprocessor

Offload with Intel Cilk Plus

```
_Cilk_shared int global_var = 45;

_Cilk_shared int f(int x) { return (x*x+1); }

int main() {
    int _Cilk_shared *a;
    a = _Offload_shared_malloc(count*sizeof(int));
    …
        _Cilk_offload
        {
        #pragma omp parallel for
        for(i=0; i<count; ++i)
            a[i] = f(i) + global_var;
        }
    …
    _Offload_shared_free(a);
}
```

_Cilk_offload syntax

Example	□□□□□□□□
x = _Cilk_offload func(y);	func □□□□□□□□ coprocessor □□□□□□□
x = _Cilk_offload_to (card_num) func(y)	func □□□□□□□□□□□□□ coprocessor □□□□□□□
_Cilk_offload _Cilk_for (i=0;i<N; ++i) { … }	offload parallel loop □□ □□□□□□□ coprocessor

Pragma offload vs. Cilk Plus

Pragma Offload	Cilk Plus
Explicit Copies – □□□□□□ □□□□□□□□ Copy □□□□□□ □□□	Implicit Copies – □□□□□□□ □□□□□□□□ Copy □□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□ coprocessor
□□□□□□□□□□□□□□□□□□□□□ scalar, arrays □□□ structure	□□□□□□□□□□□□□□□□□□□ □□□□□□ □□□□□□ classes (□□□□□□□□□□□□□□□□□□ block □□□□□ data □□□ □□□□□□□□□□□)
□□□□□□□□□□□□□ Fortran, C □□□ C++	□□□□□□□□□□□□□ C □□□ C++

Offload Compilation

- [illegible]
  - [illegible] pragma offload [illegible] [illegible] [illegible] offload [illegible]
- [illegible] offload [illegible]
  -no-offload
- [illegible] **-openmp** [illegible] OpenMP

Asynchronous offload and data transfer

- □□□ offload □□□□□□□□□□□□□□ blocking (□□□□□□ coprocessor □□□□□□□□□□□□□ □□□□□□□□□□□ host □□□)
- □□□□□□□□□□ host □□□ coprocessor □□□□□□□□□□□□□□□□□□□□□□□□□□
  - task,sections □□□ OpenMP
  - _Cilk_spawn □□□ Intel Cilk Plus

Asynchronous with OpenMP task

```
#pragma omp parallel
    #pragma omp single
    {
        #pragma omp task
        #pragma offload target(mic) …
        { /* Code to offload to mic */ }
        #pragma omp task
        { /* Code to run on host */ }
    }
```

Asynchronous with OpenMP section

```
#pragma omp parallel sections
{
    #pragma omp section
    #pragma offload target(mic) …
    { /* Code to offload to mic */ }
    #pragma omp section
    { /* Code to run on host */ }
}
```

Asynchronous with Cilk

```
y = _Cilk_spawn _Cilk_offload f(x);
/* Code to run on host */
_Cilk_sync;
```

End Session 2

Lab - convert code to offload

- □□□□□□□□ lab2/omp_offload.cpp
- □□□□□□□□□□ pragma offload □□□□□ □□□□□□□□□□□□□□□□□□ offload □□□□□□ □□□□
- □□□□□□□□□□□□ omp_offload.cpp □□□ omp_offload_solution.cpp □□□□□□□□ □□□□□□□

Lab – Compare host and offload performance

- Host (no offload)
  - □□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□ vectorization messages

```
$ icc -vec-report3 -openmp -no-offload omp_offload.cpp
$ ./a.out 2048
```

- Offload
  - □□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□ □□□ vectorization messages

```
$ icc -vec-report3 -openmp omp_offload.cpp
$ ./a.out 2048
```

Lab – Tracing offload program

- □□□□□□□□□□ environment variable □□□□□□□ H_TRACE □□□□□□□ 1 □□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□?

```
$ export H_TRACE=1
```

- □□□□□□□□□□□□ H_TRACE □□□□□□□□ H_TIME □□□□□□□ 1
  □□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□

```
$ unset H_TRACE
$ export H_TIME=1
```

- □□□□□□□□□□□□□□□□□□□□□□□ H_TIME □□□ H_TRACE

```
$ unset H_TRACE
$ unset H_TIME
```

Lab – Scaling offload program

- [illegible] thread [illegible] OMP_NUM_THREADS [illegible] 1, 2, 16, 32, 64, 93, 128 [illegible] 204 [illegible]

```
$ export OMP_NUM_THREADS=<number>
$ ./a.out 2048
```

Number of Threads	1	2	16	32	64	93	128	204
Time (s.)								

Advanced Offload Topics

Data Persistence

- □□ scope □□□ offload □□□□□□□□□□□□ □□□□□ allocate □□□ deallocate □□□□ □□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ alloc_if(cond) □□□ free_if(cond)

Data Persistence

```
#pragma offload target(mic) in(A:length(N) alloc_if(1) free_if(0))
{
   // Use “A” on coprocessor
}

// Host program
#pragma offload target(mic) nocopy(A:length(N) alloc_if(0) free_if(0))
{
   // Use “A” on coprocessor
}

// Host program
#pragma offload target(mic) out(A:length(N) alloc_if(0) free_if(1))
{
   // Use “A” on coprocessor
}
```