

Xeon Phi Offload

HongsukYi
(hsyi@kisti.re.kr)

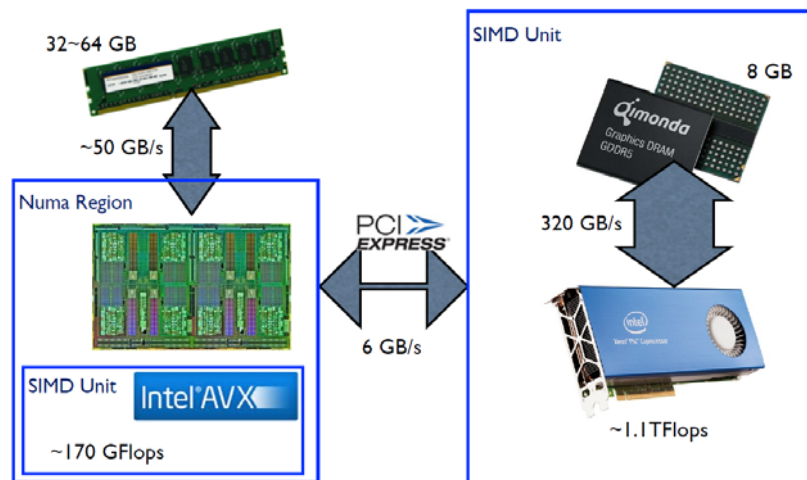
1

Contents

- Heterogeneous Offload model
 - ✓ Basics Offload
 - ✓ Two offload models
- Using pragma/directive offload
 - ✓ Directives (Code Blocks – Targets)
 - ✓ Preparation and Offload Process Steps (mechanism)
 - ✓ Data Transfers
 - ✓ Declaration for Functions and Global, Pointer Data
 - ✓ Persistent Data
 - ✓ Asynchronous Offloading

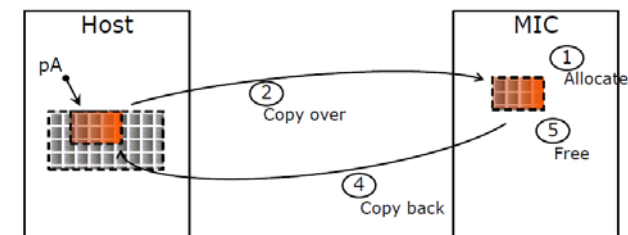
2

Basics: Offloading



3

Offload using explicit copies



- Default treatment of in/out variables in a #pragma offload statement

- At the start of an offload:

- o Space is allocated on the coprocessor
- o in variables are transferred to the coprocessor

```
#pragma offload inout(pA:length(n))
{...} 3
```

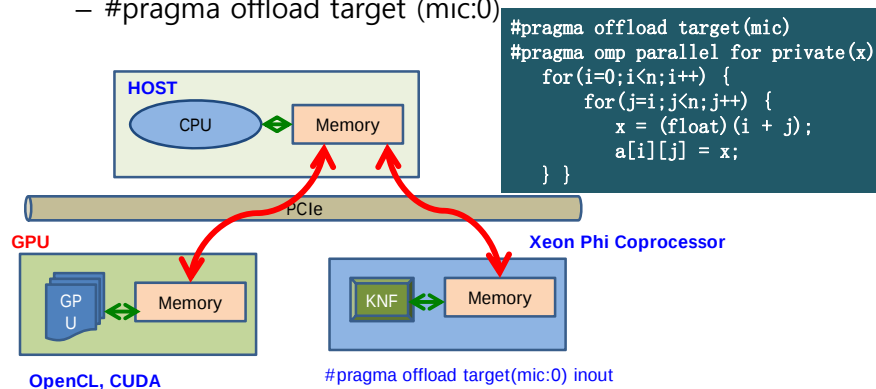
- At the end of an offload:

- o out variables are transferred from the coprocessor
- o Space for both types (as well as inout) is deallocated on the coprocessor

4

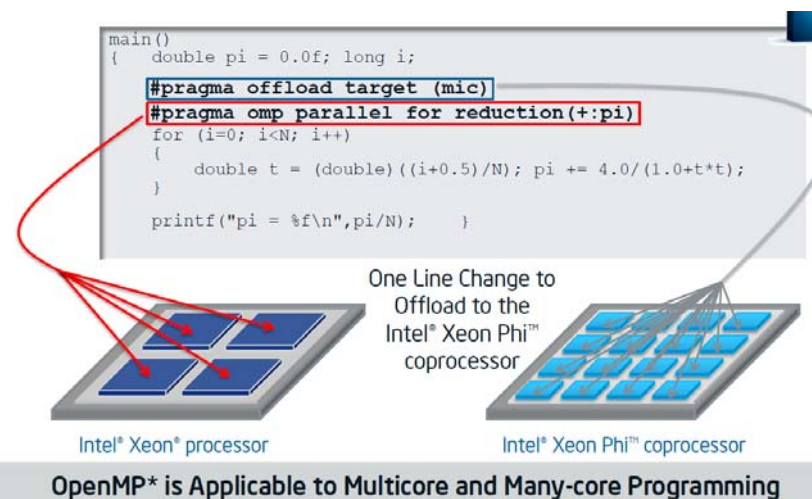
Data transfer for offload

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



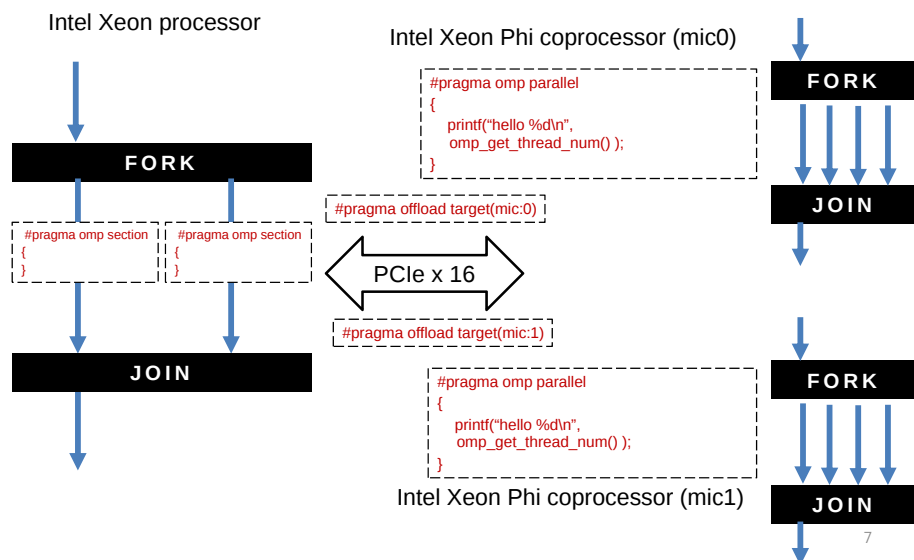
5

Go Parallel with OpenMP



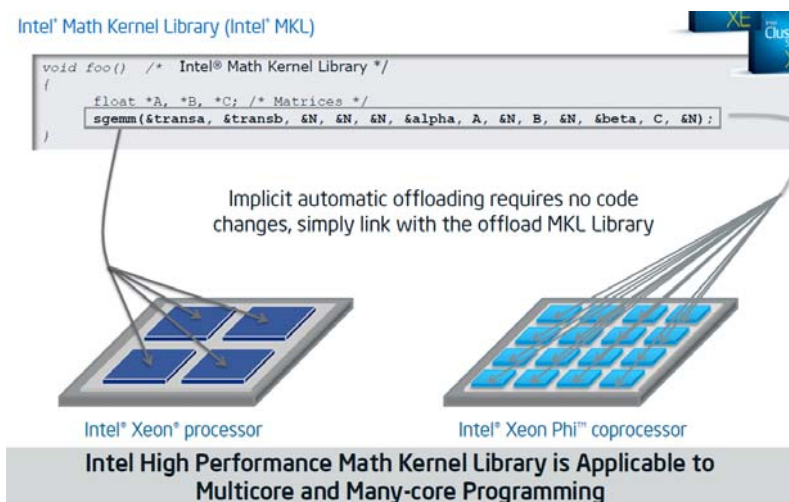
6

OpenMP Host with OpenMP Offload



7

Automatic Offload with MKL



8

Offload

```
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;  }

    #pragma offload target(mic) in(a)
    saxpyCPU(n, a, x, y);
    free(x); free(y);
}
```

9

Offloading Strategy

- Offload directive
 - ✓ #pragma offload target (mic:0)
- Great time to venture into manycores
 - ✓ Try offloading compute intensive section
 - ✓ Optimize data transfers
 - ✓ Use asynchronous transfer
- Good for
 - ✓ Computation time is substantially higher than the data transfer time

10

SAXPY: CUDA

```
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;  }

    float *dx; float *dy;
    cudaMalloc (&dx, n*sizeof(float));
    cudaMalloc(&dy, n*sizeof(float));

    cudaMemcpy(dx, x, n * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(dy, y, n * sizeof(float), cudaMemcpyHostToDevice);

    dim3 threadsPerBlock(128);
    dim3 blocksPerGrid(n/threadsPerBlock.x);
    saxpy_GPU<<<blocksPerGrid, threadsPerBlock>>>(n, 2.0, dx, dy);

    cudaMemcpy(h_y, d_y, n * sizeof(float), cudaMemcpyDeviceToHost);

    free(x); free(y);
    cudaFree(dx);    cudaFree(dy);
    return 0;
}
```

11

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;
}
```

12

SAXPY: OpenACC

```
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 acc data copyin(x[0:n])
    {
        #pragma acc parallel copy(y[0:n])
        #pragma acc loop
        for (int i=0; i< n; i++)
            y[i]=a*x[i]+y[i]
    }
    free(x); free(y); return 0;
}
```

13

SAXPY: OpenMP 4.0 (GPGPU)

```
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 omp target data map(to:x)
    {
        #pragma omp target map (tofrom:y)
        #pragma omp teams
        #pragma omp distribute
        #pragma omp parallel for
        for (int i=0; i< n; i++)
            y[i]=a*x[i]+y[i]
    }
    free(x); free(y); return 0;
}
```

14

Role of a single offload pragma

- #pragma offload target(mic) inout(a:length(10))
 - ① Coprocessor memory allocation
 - ② Input data copy from the host to MIC memory
 - ③ Offloaded execution on the coprocessor
 - ④ Copying of results back to the host memory
 - ⑤ Deallocation of memory allocated on the MIC

15

Simple example

- Insert Offload Directive:
 - ✓ #pragma offload target(mic:1)
 - ✓ !dir\$ offload target(mic)
- Compile with Intel Compiler:
 - ✓ icc prog.c
 - ✓ ifort prog.f90
- How to turn off offloading:
 - ✓ use `-no-offload` option
- OpenMP regions can be offloaded directly.
 - ✓ OpenMP parallel regions can exist in offloaded code blocks or functions.

16

Compile and run

- Compile on CPU interactively
 - ✓ `icc -o run.x -openmp -O3 myprog.c`
 - ✓ `ifort -o run.x -openmp -O3 myprog.f90`
- Run on compute node
 - ✓ `export OMP_NUM_THREADS=16`
 - ✓ `export MIC_OMP_NUM_THREADS=120 (240)`
 - ✓ `export MIC_PREFIX=MIC`
 - ✓ `./run.x`
- Use `KMP_AFFINITY`

17

L3-0: Count 3s

```
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;
}
```

```
$ icc count3s_off0.c
$ ./a.out
There are 3332344 elements
in the array[10000000]
r= 33.32(%)
```

```
int main( int argc, const char* argv[] ){
    int i; int NSIZE = 10000000;
    int* array;
    array = (int*) malloc(sizeof(int)*NSIZE);
    for ( i = 0 ; i < NSIZE ; i++) {
        array[i] = (rand() % 3 == 0 ? 3 : 1);
    }
    int num_c3s = count3s(NSIZE, array);
    printf("There are %d elements in the array[%d] r=%6.2lf(%) \n",
        num_c3s, NSIZE, (1.0E+2*num_c3s)/NSIZE);
    return 0;
}
```

18

L3-1: Count3s (Offload)

```
int main( int argc, const char* argv[] ){
    int i; int NSIZE = 10000000;
    int* array = (int*) malloc(sizeof(int)*NSIZE);
    for ( i = 0; i < NSIZE ; i++) {
        array[i] = (rand() % 3 == 0 ? 3 : 1);
    }
    printf("Hello World from CPU!\n");
    for (i=0; i < 10; i++){
        #pragma offload target(mic)
        printf("Hello World from MIC! i=%d\n", i);
        fflush(0);
    }
    int num_c3s = count3s(NSIZE, array);
    printf("num_c3s=%d in the array[%d] ratio=%6.2lf(%) \n",
        num_c3s, NSIZE, (1.0E+2*num_c3s)/NSIZE);
}
```

```
#pragma offload target(mic) ~
executed on the Xeon Phi.
```

19

L3-1: Count3s (Logging stdout)

```
./run.mic.2
Hello World from CPU!
I have 32 CPU cores.
I have 32 MIC cores.
I have 32 MIC cores.
I have 32 MIC cores.
I have 32 MIC cores.
I have 32 MIC cores.
num_c3s=3332344 in the
array[10000000] ratio= 33.32(%)
Elapsed times 0.6050(sec)
Hello World from MIC! i=0
Hello World from MIC! i=1
Hello World from MIC! i=2
Hello World from MIC! i=3
Hello World from MIC! i=4
[localhost.localdomain@root:/
```

```
With fflush(0)
./run.mic.3
Hello World from CPU!
I have 32 CPU cores.
Hello World from MIC! i=0
I have 228 MIC cores.
Hello World from MIC! i=1
I have 228 MIC cores.
Hello World from MIC! i=2
I have 228 MIC cores.
Hello World from MIC! i=3
I have 228 MIC cores.
Hello World from MIC! i=4
I have 228 MIC cores.
num_c3s=3332344 in the
array[10000000] ratio= 33.32(%)
Elapsed times 0.6081(sec)
```

20

LAB1: Review Native Mode

- which icc
 - ✓ `$ /opt/intel/composer_xe_2013.2.146/bin/intel64/icc`
- setup intel64 compiler
 - ✓ `$ source /opt/intel/composerxe/bin/compilervars.sh intel64`
 - add to the `.profile` or `.bash_profile`
- Compile with `–mmic`
 - ✓ `icc –o a.mic –mmic count3s.c`
- IP-addressable PCIe device with an μ OS
 - ✓ `scp a.mic mic0:~/.`
 - ✓ `scp /opt/intel/composer_xe_2013/lib/mic/libiomp5.so mic0:~/.`

21

LAB1: SINK_LD_LIBRARY_PATH

```
$ export SINK_LD_LIBRARY_PATH=/opt/intel/composerxe/lib/mic
$ micnativeloadex ./a.out -l
Dependency information for ./a.out
Full path was resolved as
/home/hsyi/wrk/summer13/L03.Offload/lab3Sol/c3s/simple/./a.out
Binary was built for Intel(R) Xeon Phi(TM) Coprocessor
(codename: Knights Corner) architecture
SINK_LD_LIBRARY_PATH = /opt/intel/composerxe/lib/mic
```

```
$micnativeloadex ./a.out
Hello world!
I have 228 cores.
num_c3s=3332344 in the array[10000000] ratio= 33.32(%)
```

22

Automatic Offload

- Offloads some MKL routines automatically
 - ✓ No coding change and no recompiling
- Makes sense with BLAS-3 type routines
 - ✓ Minimal Data $O(n^2)$, Maximal Compute $O(n^3)$
- Supported Routines (more to come)
- Compile as usual, use new `–mkl`
 - ✓ Works with serial, OpenMP and MPI codes.
- Enable with `MKL_MIC_ENABLE` variable

23

Compiler Assisted Offload

- A code block is assigned to execute on a coprocessor by a directive:
 - ✓ `#pragma offload target(mic[:dev_id]) ... C/C++`
 - ✓ `!dir offload target(mic[:dev_id]) ... FORTRAN`
- code block:
 - ✓ single or multiple program statements
 - C/C++: `{...}`
 - F90: `$dir offload begin...$dir end offload`
 - `target(mic[:dev_id]):`
 - ✓ `target(mic):`
 - offload on MIC if available, use cpu if MIC fails
 - ✓ `target(mic[:dev_id]):`
 - only offload, on MIC identified as `dev_id` (0, 1,...)

24

Data transfers

➤ Basic Idea

✓ Minimize unnecessary transfers with in/out/inout data specifiers.

- #pragma offload target(mic0) data_specifier(identifier_list) //syntax
- #pragma offload target(mic:0) in(b,c) // Only copy b and c into MIC
- #pragma offload target(mic:0) out(a) // Only return a
- #pragma offload target(mic) inout(d) // Default, copy into and out of

25

Setting Environment on the CPU

➤ set OMP_NUM_THREADS

- ✓ MIC_ENV_PREFIX=MIC
- ✓ OMP_NUM_THREADS=8
- ✓ MIC_OMP_NUM_THREADS=124.

➤ Multiple Environment Variables on All MICs

- ✓ sets X=x and Y=y on all coprocessors.
- ✓ MIC_ENV_PREFIX=MIC MIC_VAR X=x|Y=y

➤ on a Particular Coprocessor

- ✓ MIC_ENV_PREFIX=MIC
- ✓ OMP_NUM_THREADS=8
- ✓ MIC_4_OMP_NUM_THREADS=124

➤ Multiple Environment on a Particular Coprocessor

- ✓ sets X=x and Y=y on the fifth coprocessor installed.
- ✓ MIC_ENV_PREFIX=MIC
- ✓ MIC_4_VAR X=x|Y=y

26

Language Extensions for Offload

➤ Offload pragma/directives

✓ Provides offload capability with pragma/directive

✓ #pragma offload C/C++

✓ !dir\$ offload Fortran

➤ Mark functions and variables for availability on MIC

✓ __declspec(target(mic))

✓ __attribute__((target(mic)))

✓ #pragma offload_attribute(target(mic))

27

L3-2: Offloading functions

```
#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(%)
```

28

L3-4: OFFLOAD_REPORT

```
$ gcc c3s_05.c
$ ./a.out
Hello World from CPU!
Hello World from MIC!
Counting 3s from MIC!
num_c3s=3332344
in the array[10000000] ratio= 33.32(%)
$ export OFFLOAD_REPORT=0
$ ./a.out
Hello World from CPU!
offload warning: ignoring OFFLOAD_REPORT setting; use
a value in range 0-3
Hello World from MIC!
Counting 3s from MIC!
num_c3s=3332344 in the array[10000000] ratio= 33.32(%)
$ export OFFLOAD_REPORT=1
./a.out
Hello World from CPU!
[Offload] [MIC 0] [File]      off_c3s_05.c
[Offload] [MIC 0] [Line]      38
[Offload] [MIC 0] [Tag]       Tag0
Hello World from MIC!
Counting 3s from MIC!
[Offload] [MIC 0] [CPU Time]    0.000000 (seconds)
[Offload] [MIC 0] [CPU->MIC Data] 40000004 (bytes)
[Offload] [MIC 0] [MIC Time]    0.056709 (seconds)
[Offload] [MIC 0] [MIC->CPU Data] 4 (bytes)
num_c3s=3332344 in the array[10000000] ratio= 33.32(%)
```

```
$ export OFFLOAD_REPORT=2
$ ./a.out
Hello World from CPU!
[Offload] [MIC 0] [File]      c3s_05.c
[Offload] [MIC 0] [Line]      38
[Offload] [MIC 0] [Tag]       Tag0
Hello World from MIC!
Counting 3s from MIC!
[Offload] [MIC 0] [CPU Time]    0.000000 (seconds)
[Offload] [MIC 0] [CPU->MIC Data] 40000004 (bytes)
[Offload] [MIC 0] [MIC Time]    0.056709 (seconds)
[Offload] [MIC 0] [MIC->CPU Data] 4 (bytes)
num_c3s=3332344 in the array[10000000] ratio= 33.32(%)
$ export OFFLOAD_REPORT=3 (?)
```

Timing and Data in Offload Regions
 -> OFFLOAD_REPORT
 environment variable.
 -> __Offload_report API
 Exit() ~When calling exit() from within an
 offload region, the application reports an
 offload error stating the process on the device
 exited unexpectedly.

29

L3-5: MIC_ENV_PREFIX

```
printf("Hello World from CPU!\n");
#pragma offload target(mic) in(array:length(NSIZE))
{
    char* myvalue = getenv("MYFIRST");
    if(myvalue)
    {
        printf("MYFIRST=%s on the coprocessor \n",myvalue);
    } else {
        printf("MYFIRST is not defined on the coprocessor\n");
    }
    output();
    num_c3s = count3s(NSIZE, array);
}
```

```
$ export MYFIRST=myvalue | ./a.out
MYFIRST=myvalue on the coprocessor

$export MIC_ENV_PREFIX=MIC | ./a.out
MYFIRST is not defined on the coprocessor

$export MIC_MYFIRST=myvalue |./a.out
MYFIRST=myvalue on the coprocessor
```

30

Data transfer (1)

```
__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 main( int argc, const char* argv[] ){
    int i;
    int num_c3s ;
    int NSIZE = 10000;
    int array[NSIZE];
    for ( i = 0; i < NSIZE ; i++ ) {
        array[i] = (rand() % 3 == 0 ? 3 : 1);
    }
    #pragma offload target(mic)
    num_c3s = count3s(NSIZE, array);
    printf("num_c3s=%d in the array[%d] ratio=%6.2f(%) \n",
    num_c3s,NSIZE,(1.0E+2*num_c3s)/NSIZE);
    return 0;
}
```

- ✓ Automatically Offload
- ✓ Offload of local scalars and arrays of known size occurs automatically
 - Local scalars
 - Arrays of known size

```
./a.out
Hello World from CPU!
num_c3s=26 in the array[10000] ratio= 0.26(%)
Hello World from MIC! i=0
```

31

Offloading pointer-based arrays

```
// printf("Hello World from CPU!\n");
#pragma offload target(mic) in(array:length(NSIZE))
{
    char* myvalue = getenv("MYFIRST");
    if(myvalue)
    {
        printf("MYFIRST=%s on the coprocessor \n",myvalue);
    } else {
        printf("MYFIRST is not defined on the coprocessor\n");
    }
    output();
    num_c3s = count3s(NSIZE, array);
}
```

- The array size is unknown at compile time. We must indicate the array length in a clause of #pragma offload
- Length is in array elements not bytes

32

Persistent Data

Allocation Operation	Deallocation (Free) Operation	Operations Performed (Use Case)
alloc_if(true)	free_if(true)	This is the default when no storage operations are specified. Allocate space at beginning, free at end.
alloc_if(true)	free_if(false)	Allocate space, don't free (make space available on device, and retain for future use).
alloc_if(false)	free_if(true)	don't allocate, but free (reuse device storage, but will not need later)
alloc_if(false)	free_if(false)	don't allocate, don't free (reuse device storage, and leave for future use)

Table 2. Truth table for alloc_if and free_if modifier options for the data specifier in/out/inout/nocopy(...) clause.

33

L3-7: Data persistence ex(1/2)

```
int main( int argc, char* argv[] ){
    int i;  int NSIZE = 1000000;  int *array, *isum;  int NITER = 10;  int newSize;
    array = (int*) malloc(sizeof(int)*NSIZE*NITER);
    isum = (int*) malloc(sizeof(int)*NITER);

    int num_c3s=0;
    newSize = NSIZE*NITER;
    setInit(newSize, array);

    #pragma offload_transfer target(mic) in (array:length(newSize) alloc_if(1) free_if(0))
    printf("Hello World from CPU!\n");

    for (i=0; i< NITER; i++)
    {
        #pragma offload target(mic) nocopy (array:length(newSize) alloc_if(0) free_if(i==NITER-1)) \
        out (isum:length(NITER) alloc_if(i==0) free_if(i==NITER-1))
        count3s(i, NSIZE, array, isum);
        num_c3s += isum[i];
        printf("num_c3s=%d in the array[%d] ratio=%6.4lf(%%) \n",
            num_c3s,newSize,(1.0E+2*num_c3s)/((i+1)*NSIZE));
    }
}
```

34

Data persistence ex(2/2)

```
#pragma offload_attribute(push, target(mic))
// The target attribute is set for all functions
int count3s(int index, int N, int* data, int* isum){
    int icount=0;
    int i;
    for ( i = index*N ; i < N*(index+1) ; i++ ){
        if ( data[i] == 3 ) icount++;
    }
    isum[index] = icount;
    return icount;
}

int setInit(int newSize, int* data)
{
    int i;
    for ( i = 0; i < newSize ; i++ ) {
        data[i] = (rand() % 3 == 0 ? 3 : 1);
    }
    return 0;
}
```

```
$ gcc off_c3s_09.c
$ ./a.out
Hello World from CPU!
num_c3s=333346 in ratio=33.3346(%)
num_c3s=665810 in ratio=33.2905(%)
num_c3s=999549 in ratio=33.3183(%)
num_c3s=1332831 in ratio=33.3208(%)
num_c3s=1666019 in ratio=33.3204(%)
num_c3s=1999486 in ratio=33.3248(%)
num_c3s=2332089 in ratio=33.3156(%)
num_c3s=2664790 in ratio=33.3099(%)
num_c3s=2999338 in ratio=33.3260(%)
num_c3s=3332344 in ratio=33.3234(%)
```

35

L3-8: Asynchronous Offloading

```
int *array = (int*) malloc(sizeof(int)*NSIZE*NITER);
int *isum_target = (int*) malloc(sizeof(int)*NITER);
int *isum_host = (int*) malloc(sizeof(int)*NITER);
setInit(newSize, array);
#pragma offload_transfer target(mic) \
in (array:length(newSize) alloc_if(1) free_if(0))

for (i=0; i< NITER; i++) {
    #pragma offload target(mic:0) \
    nocopy (array:length(newSize) alloc_if(0) free_if(i==NITER-1)) \
    out (isum_target:length(NITER) alloc_if(i==0) free_if(i==NITER-1)) \
    signal (array)
    count3s(i, NSIZE, array, isum_target);
    count3s(i, NSIZE, array, isum_host);
    #pragma offload_transfer target(mic:0) wait(array)
    num_c3s += isum_target[i];
    num_c3s_host += isum_host[i];
    r_target = (1.0E+2*num_c3s)/((i+1)*NSIZE);
    r_host = (1.0E+2*num_c3s_host)/((i+1)*NSIZE);
    // printf("%d %lf %lf \n",i,r_target,r_host);
    printf("num_c3s=%d in the array[%d] r_diff=%6.4lf(%%) \n",
        num_c3s,newSize,r_target - r_host);
}
```

```
num_c3s=333346 in r_diff=0.0000(%)
num_c3s=665810 in r_diff=0.0000(%)
num_c3s=999549 in r_diff=0.0000(%)
num_c3s=1332831 in r_diff=0.0000(%)
num_c3s=1666019 in r_diff=0.0000(%)
num_c3s=1999486 in r_diff=0.0000(%)
num_c3s=2332089 in r_diff=0.0000(%)
num_c3s=2664790 in r_diff=0.0000(%)
num_c3s=2999338 in r_diff=0.0000(%)
num_c3s=3332344 in r_diff=0.0000(%)
```

36

L3-20: Consideration for LJ

➤ Affinity is VERY important on manycore systems

- ✓ Add to the `mtime()` routine to measure elapsed time

➤ OpenMP code

- ✓ Elapsed times on CPU

- `export KMP_AFFINITY = scatter`
- `export KMP_AFFINITY = compact`

- ✓ OpenMP runs on MIC

- ✓ Native Mode

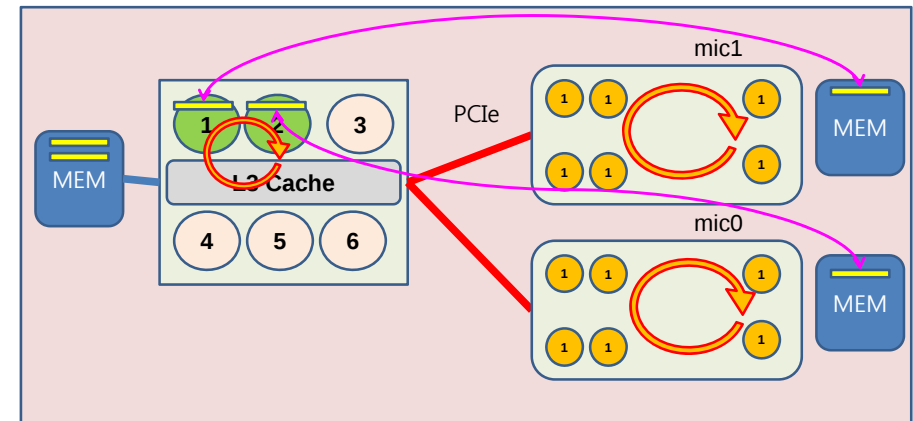
- `export MIC_ENV_PREFIX=MIC`
- `export MIC_KMP_AFFINITY = scatter, granularity=fine, verbose`
- `export MIC_KMP_AFFINITY = compact`

- ✓ Offload Mode

37

Offload MPI with OpenMP

MPI Only: `mpirun -n 2 a.offload.x`

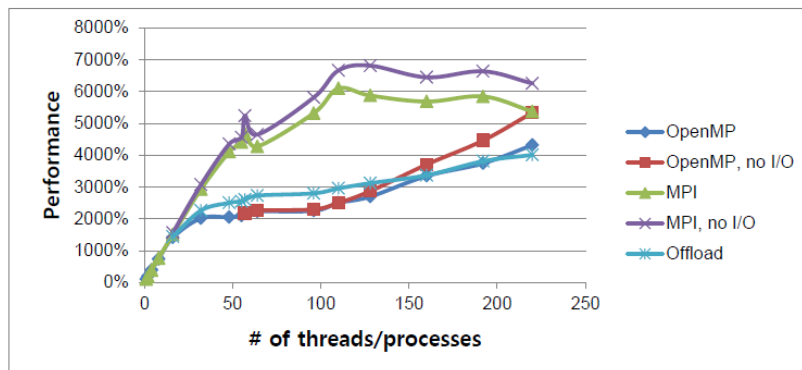


38

Benchmark

• LJ code

– Native mode, $N_2=500$, File I/O



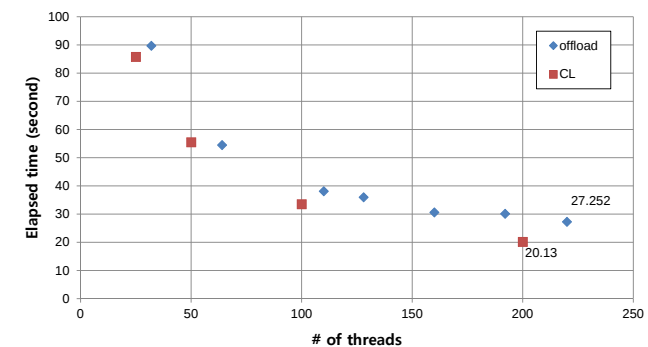
39

Performance comparison

• Lennard-Jones MD

- # of molecules = 3200
- 100 time steps
- Work group size = 16 or 32

- OpenCL shows better performance.
- Codes are in the same optimization degree.



40

MPI PROGRAMMING MODEL ON INTEL XEON PHI

41

Agenda

- MPI Programming Models
 - Host-only, Coprocessor-only, Symmetric
- Hybrid Programming
 - MPI + OpenMP, MPI + Offload
- Process Affinity

42

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

43

Host Only Model

Fortran	C
<pre>program hello implicit none include 'mpif.h' integer nprocs, rank, ierr call MPI_Init(ierr) call MPI_Comm_size(MPI_COMM_WORLD, nprocs, ierr) call MPI_Comm_rank(MPI_COMM_WORLD, rank, ierr) write (*,*) 'Hello World process ',rank,' of ', nprocs call MPI_Finalize(ierr) end</pre>	<pre>#include <stdio.h> #include <mpi.h> int main(int argc, char *argv[]) { int nprocs, rank; MPI_Init(&argc, &argv); MPI_Comm_size(MPI_COMM_WORLD, &nprocs); MPI_Comm_rank(MPI_COMM_WORLD, &rank); printf("Hello World process %d of %d\n", rank, nprocs); MPI_Finalize(); }</pre>
\$ mpiicc/mpiifort -o hello.x hello.c/f90	
\$ mpirun -np 4 -host localhost ./hello.x (in the host or coprocessor)	

44

Coprocessor Only Model

Environment

```
$ scp /opt/intel/impi/4.1.0/mic/lib/* mic0:/lib64
$ scp /opt/intel/impi/4.1.0/mic/bin/* mic0:/bin
$ I_MPI_MIC = on/enable/yes/1
```

Compilation

```
$ mpiicc/mpiif90 -mmic -o hello.x.mic hello.c/f90
```

Running

```
$ scp hello.x.mic mic0:~
$ mpirun -n 2 -host mic0 ~/hello.x.mic
$ ssh mic0
$ mpirun -n 2 -host mic0 ~/hello.x.mic
$ scp hello.x.mic mic1:~
$ mpirun -n 2 -host mic0 ~/hello.x.mic : -n 2 -host mic1 ~/hello.x.mic
```

45

Symmetric Model

Environment

```
$ scp /opt/intel/impi/4.1.0/mic/lib/* mic0:/lib64
$ scp /opt/intel/impi/4.1.0/mic/bin/* mic0:/bin
$ I_MPI_MIC = on/enable/yes/1
```

Compilation

```
$ mpiicc/mpiif90 -o hello.x hello.c/f90
$ mpiicc/mpiif90 -mmic -o hello.x.mic hello.c/f90
```

Running

```
$ scp hello.x.mic mic0:~
$ mpirun -n 2 -host localhost ./hello.x : -n 2 -host mic0 ./hello.x.mic
$ ssh mic0
$ mpirun -n 2 -host localhost ./hello.x : -n 2 -host mic0 ./hello.x.mic
$ export I_MPI_MIC_POSTFIX=.mic
$ mpirun -n 4 -machinefile hostfile ./hello.x
```

46

Hybrid Programming (MPI + OpenMP)

Fortran	C
<pre>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) !\$OMP PARALLEL PRIVATE(tid) tid = omp_get_thread_num() print *, 'tid = ', tid, ' rank = ', rank, ' nprocs = ', nprocs !\$OMP END PARALLEL CALL MPI_Finalize(ierr) End</pre>	<pre>#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 omp parallel private(tid) { tid = omp_get_thread_num(); printf ("tid = %d rank = %d nprocs = %d\n", tid, rank, nprocs); } MPI_Finalize(); }</pre>
<pre>\$ mpiicc/mpiifort -openmp -o hello.x hello.c/f90 \$ mpiicc/mpiifort -mmic -openmp -o hello.x.mic hello.c/f90 \$ scp hello.x.mic mic0:~ \$ export I_MPI_POSTFIX=.mic \$ mpirun -machinefile lists -np 4 ./hello.x \$ mpirun -host localhost -n 2 ./hello.x : -host mic0 -n 2 -env OMP_NUM_THREADS=4 ./hello.x.mic</pre>	

47

Hybrid Programming (MPI + Offload)

Fortran	C
<pre>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) !DIRS OFFLOAD BEGIN TARGET(mic) !\$OMP PARALLEL PRIVATE(tid) tid = omp_get_thread_num() print *, 'tid = ', tid, ' rank = ', rank, ' nprocs = ', nprocs !\$OMP END PARALLEL !DIRS END OFFLOAD call MPI_Finalize(ierr) End</pre>	<pre>#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(); }</pre>
<pre>\$ 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</pre>	

48

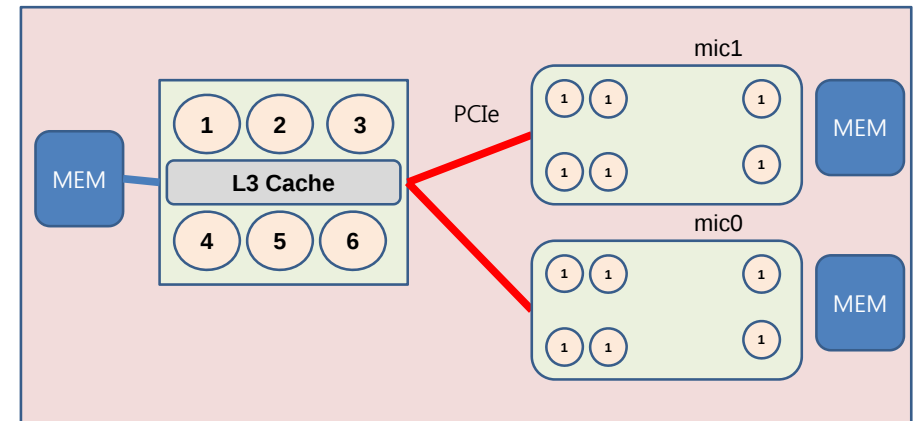
Process Affinity

- Bindings (OpenMPI)
 - -bind-to-core
 - -npernode
 - -pernode : equivalent to -npernode 1
 - -bynode
 - -report-bindings
- Intel MPI
 - I_MPI_PIN
 - I_MPI_PIN_MODE
 - I_MPI_PIN_DOMAIN
 - I_MPI_PIN_PROCESSOR_LIST

49

Heterogeneous Computing

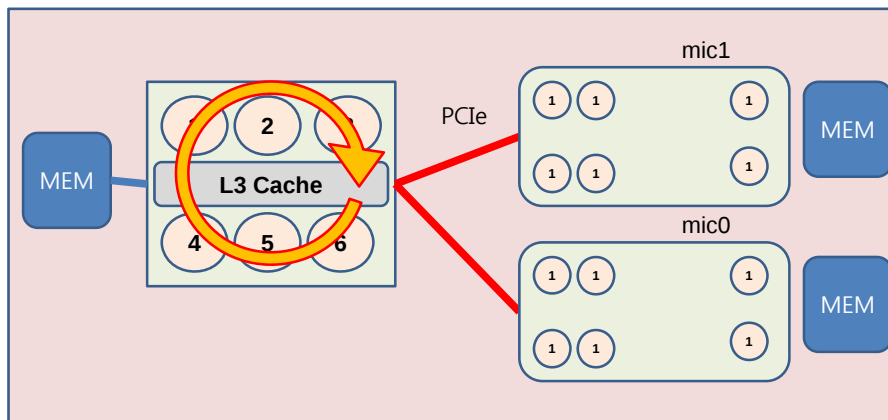
1 Node



50

Native MPI

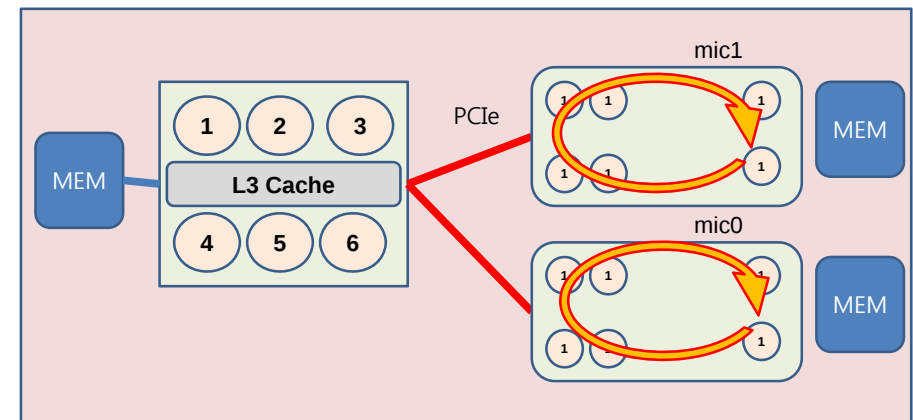
1) Host Only



51

Native MPI

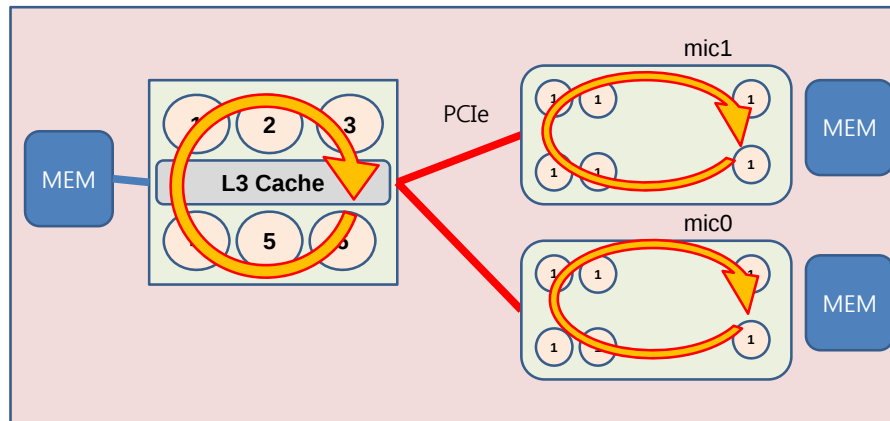
`mpiirun -mmic -o a.mic a.c ; mpiirun -n 120 -host mic0 a.mic: -host mic1 a.mic -n 120`



52

Native MPI

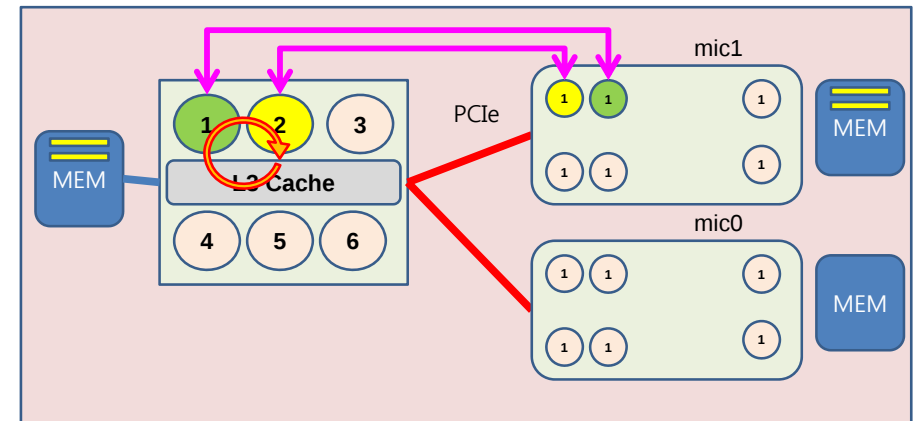
mpirun -host localhost -n 6 a.cpu : -host mic0 -n 120 a.mic : -host mic0 -n 120 a.mic



53

Offload MPI without OpenMP

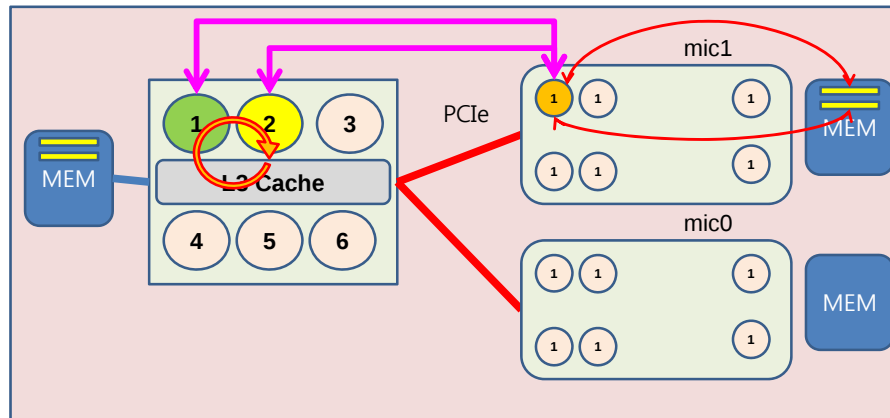
MPI Only: mpirun -n 2 a.offload.x



54

Offload MPI

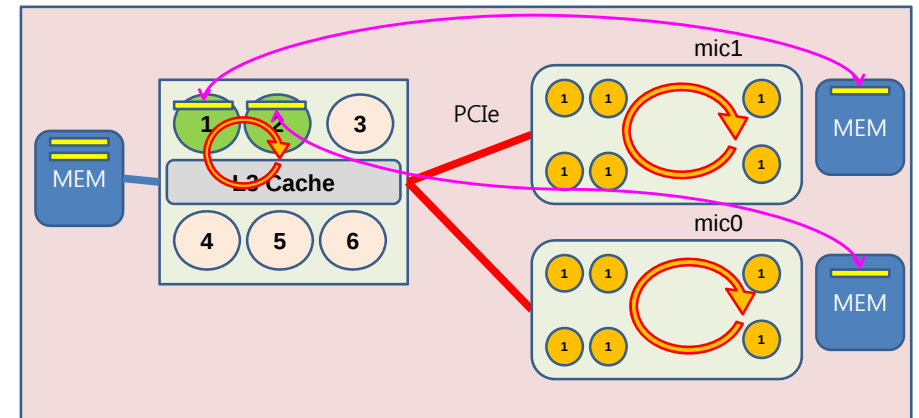
MPI Only: mpirun -n 2 a.offload.x



55

Offload MPI + OpenMP

MPI Only: mpirun -n 2 a.offload.x



56

MPI Optimization Strategies

- Work scheduling to balance the workload
 - Heterogeneous Computing
 - Xeon CPU vs Xeon Phi coprocessor
- A large amount of MPI communication
 - The total number of MPI processors exceed 240+ on MIC
 - MPI+OpenMP hybrid parallel programming model