Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

Intel MIC Programming Workshop: Offloading & MPI

Dr. Volker Weinberg (LRZ)

February, 7-8, 2017, IT4Innovations, Ostrava, Czech Republic

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

Supported Offloading Models

- Intel Language Extensions for Offload (LEO)
- OpenMP 4.x Offloading
- “Mine Yours Ours” (MYO) virtual shared memory model

Intel Xeon Phi Programming Models: Intel Language Extensions for Offload (LEO) I

Intel Offload Directives

- Syntax:
  - **C:**

    #pragma offload target(mic) <clauses>

    <statement block>
  - **Fortran:**
    - *!DIR$ offload target(mic) <clauses>*
      <statement>
    - *!DIR$ omp offload target(mic) <clauses>*
      <OpenMP construct>

Intel Offload Directive

- C:
  - Pragma can be before any statement, including a compound statement or an OpenMP parallel pragma
- Fortran:
  - If OMP is specified: the next line, other than a comment, must be an OpenMP PARALLEL, PARALLEL SECTIONS, or PARALLEL DO directive.
  - If OMP is not specified, next line must :
    - An OpenMP* PARALLEL, PARALLEL SECTIONS, or PARALLEL DO directive
    - A CALL statement
    - An assignment statement where the right side only calls a function

Intel Offload

- Implements the following steps:

  1. Memory allocation on the MIC
  2. Data transfer from the host to the MIC
  3. Execution on the MIC
  4. Data transfer from the MIC to the host
  5. Memory deallocation on MIC

Intel Offload: Hello World in C

```
#include <stdio.h>
int main (int argc, char* argv[]) {

#pragma offload target(mic)
 {
   printf("MIC: Hello world from MIC.\n");
 }
  printf( "Host: Hello world from host.\n");
}
```

Must be in a new line!

Intel Offload: Hello World in Fortran

```
PROGRAM HelloWorld

!DIR$ offload begin target(MIC)
PRINT *,'MIC: Hello world from MIC'
!DIR$ end offload

PRINT *,'Host: Hello world from host'
END
```

Intel Offload: Hello World in C

```
lu65fok@login12:~/tests> icpc offload1.c -o offload1

lu65fok@login12:~/tests> ./offload1
offload error: cannot offload to MIC - device is not available
```

```
lu65fok@i01r13c01:~/tests> ./offload1
Host: Hello world from host.
MIC: Hello world from MIC.
```

Intel Offload: Hello World in Fortran

```
lu65fok@login12:~/tests> ifort offload1.f90 -o offload1

lu65fok@login12:~/tests> ./offload1
offload error: cannot offload to MIC - device is not available
```

```
lu65fok@i01r13c01:~/tests> ./offload1
 Host: Hello world from host.
 MIC: Hello world from MIC.
```

Intel Offload: Hello World with Hostnames

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

int main (int argc, char* argv[]) {
  char hostname[100];
  gethostname(hostname,sizeof(hostname));

  #pragma offload target(mic)
  {
    char michostname[100];
    gethostname(michostname, sizeof(michostname));
    printf("MIC: Hello world from MIC. I am %s and I have %ld logical cores. I was
called from host: %s \n", michostname,  sysconf(_SC_NPROCESSORS_ONLN),
hostname);
  }
```

Intel Offload: Hello World with Hostnames

```
lu65fok@login12:~/tests> icpc offload.c -o offload
```

```
lu65fok@i01r13c01:~/tests> ./offload
Host: Hello world from host. I am i01r13c01 and I have 32 logical cores.
MIC: Hello world from MIC. I am i01r13c01-mic0 and I have 240 logical cores. I was called from host: i01r13c01
```

Intel Offload: -offload=optional / mandatory

```
lu65fok@login12:~/tests> icpc -offload=optional offload.c -o offload

lu65fok@login12:~/tests> ./offload
MIC: Hello world from MIC. I am login12 and I have 16 logical cores. I was called from host: login12
Host: Hello world from host. I am login12 and I have 16 logical cores.

lu65fok@login12:~/tests> icpc -offload=mandatory offload.c -o offload
lu65fok@login12:~/tests> ./offload
offload error: cannot offload to MIC - device is not available
```

Intel Offload: -none

```
lu65fok@login12:~/tests> icpc -offload=none offload.c -o offload
offload.c(13): warning #161: unrecognized #pragma
   #pragma offload target(mic)
         ^
lu65fok@login12:~/tests>

lu65fok@i01r13c01:~/tests> ./offload
MIC: Hello world from MIC. I am i01r13c01 and I have 32 logical cores.
I was called from host: i01r13c01
Host: Hello world from host. I am i01r13c01 and I have 32 logical
cores.
```

Intel Offload

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

int main(){
#pragma offload target (mic)
  {
    system(“command");
  }
}
```

Intel Offload: system(“set”)

```
lu65fok@i01r13c01:~/tests> ./system
BASH=/bin/sh
BASHOPTS=cmdhist:extquote:force_fignore:hostcomplete:interactive_comments:progcomp:prom
ptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_EXECUTION_STRING=set
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="4" [1]="2" [2]="10" [3]="1" [4]="release" [5]="k1om-mpss-linux-gnu")
BASH_VERSION='4.2.10(1)-release‘
COI_LOG_PORT=65535
COI_SCIF_SOURCE_NODE=0
DIRSTACK=()
ENV_PREFIX=MIC
EUID=400
GROUPS=()
```

Intel Offload: system(“set”)

```
HOSTNAME=i01r13c01-mic0
HOSTTYPE=k1om
IFS='
'
LIBRARY_PATH=/lrz/sys/intel/compiler140_144/composer_xe_2013_sp1.2.144/tbb/lib/mic:/lrz/sys/intel/compiler140_144/composer_xe_2013_sp1.2.144/tbb/lib/mic:/lrz/sys/intel/compiler140_144/composer_xe_2013_sp1.2.144/tbb/lib/mic:/lrz/sys/intel/compiler140_144/composer_xe_2013_sp1.2.144/tbb/lib/mic
MACHTYPE=k1om-mpss-linux-gnu
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/bin:/bin
POSIXLY_CORRECT=y
PPID=37141
PS4='+ '
PWD=/var/volatile/tmp/coi_procs/1/37141
SHELL=/bin/false
SHELLOPTS=braceexpand:hashall:interactive-comments:posix
SHLVL=1
TERM=dumb
UID=400
_=sh
```

Intel Offload: system(command)

```
#pragma offload target (mic)
  {
    system("hostname");
    system("uname -a");
    system("whoami");
     system(“id”);
  }
```

```
lu65fok@i01r13c01:~/tests> ./system
i01r13c01-mic0
Linux i01r13c01-mic0 2.6.38.8+mpss3.1.2 #1 SMP Wed Dec 18 19:09:36 PST 2013 k1om GNU/Linux
micuser
uid=400(micuser) gid=400(micuser)
```

Offload: Using several MIC Coprocessors

- To query the number of coprocessors:
  int nmics = __Offload_number_of_devices()

- To specify which coprocessor n< nmics should do the computation:
  #pragma offload target(mic:n)

- *If (*n > nmics*)* then coprocessor (n % nmics) is used

- *Important for:*
  - *Asynchronous offloads*
  - *Coprocessor-Persistent data*

Offloading OpenMP Computations

- C/C++ & OpenMP:

```
#pragma offload target(mic)
#pragma omp parallel for
for (int i=0;i<n;i++) {
      a[i]=c*b[i]+d;
}
```

- Fortran & OpenMP

```
!DIR$ offload target(mic)
!$OMP PARALLEL DO
  do i = 1, n
    a(i) = c*b(i) + d
  end do
!$omp END PARALLEL DO
```

Functions and Variables on the MIC

- C:
  - __attribute__((target(mic))) variables / function
  - __declspec (target(mic)) variables / function

  - #pragma offload_attribute(push, target(mic))
    … multiple lines with variables / functions
    #pragma offload_attribute(pop)
- Fortran:
  !DIR$ attributes offload:mic:: variables / function

Functions and Variables on the MIC

```
#pragma offload_attribute(push,target(mic))
const int n=100;
int a[n], b[n],c,d;
void myfunction(int* a, int*b, int c, int d){
  for (int i=0;i<n;i++) {
    a[i]=c*b[i]+d;
  }
}
#pragma offload_attribute(pop)

int main (int argc, char* argv[]) {
#pragma offload target(mic)
  {
        myfunction(a,b,c,d);
  }
```

Intel Offload Clauses

Clauses	Syntax	Semantics
Multiple coprocessors	**target(mic[:unit])**	Select specific coprocessors
Conditional offload	**if (condition) / manadatory**	Select coprocessor or host compute
Inputs	**in(var-list modifiers$_{opt}$)**	Copy from host to coprocessor
Outputs	**out(var-list modifiers$_{opt}$)**	Copy from coprocessor to host
Inputs & outputs	**inout(var-list modifiers$_{opt}$)**	Copy host to coprocessor and back when offload completes
Non-copied data	**nocopy(var-list modifiers$_{opt}$)**	Data is local to target
Async. Offload	**signal(signal-slot)**	Trigger asynchronous Offload
Async. Offload	**wait(signal-slot)**	Wait for completion

Intel Offload Modifier Options

Modifiers	Syntax	Semantics
Specify copy length	**length(N)**	Copy N elements of pointer's type
Coprocessor memory allocation	**alloc_if (bool)**	Allocate coprocessor space on this offload (default: TRUE)
Coprocessor memory release	**free_if (bool)**	Free coprocessor space at the end of this offload (default: TRUE)
Array partial allocation & variable relocation	**alloc (array-slice) in (var-expr)**	Enables partial array allocation and data copy into other vars & ranges

Intel Offload: Data Movement

- #pragma offload target(mic) in(in1,in2,…) out(out1,out2,…) inout(inout1,inout2,…)

- **At Offload start:**
  - Allocate Memory Space on MIC for all variables
  - Transfer in/inout variables from Host to MIC
- **At Offload end:**
  - Transfer out/inout variables from MIC to Host
  - Deallocate Memory Space on MIC for all variables

Intel Offload: Data Movement

- data = (double*)malloc(n*sizeof(double));
- #pragma offload target(mic) in(data:length(n))

- Copies n doubles to the coprocessor,
  not n * sizeof(double) Bytes
- ditto for out() and inout()

An example for Offloading: Offloading Code

```
#pragma offload target(mic) in(a,b:length(n*n)) inout(c:length(n*n))
{
  #pragma omp parallel for
   for( i = 0; i < n; i++ ) {
        for( k = 0; k < n; k++ ) {
             #pragma vector aligned
             #pragma ivdep
             for( j = 0; j < n; j++ ) {
                //c[i][j] = c[i][j] + a[i][k]*b[k][j];
                c[i*n+j] = c[i*n+j] + a[i*n+k]*b[k*n+j];
             }
         }
    }
}
```

Vectorisation Diagnostics

```
lu65fok@login12:~/tests> icc -vec-report2 -openmp offloadmul.c -ooffloadmul
offloadmul.c(35): (col. 5) remark: LOOP WAS VECTORIZED
offloadmul.c(32): (col. 3) remark: loop was not vectorized: not inner loop
offloadmul.c(57): (col. 2) remark: LOOP WAS VECTORIZED
offloadmul.c(54): (col. 7) remark: loop was not vectorized: not inner loop
offloadmul.c(53): (col. 5) remark: loop was not vectorized: not inner loop
offloadmul.c(8): (col. 9) remark: loop was not vectorized: existence of
vector dependence
offloadmul.c(7): (col. 5) remark: loop was not vectorized: not inner loop
offloadmul.c(57): (col. 2) remark: *MIC* LOOP WAS VECTORIZED
offloadmul.c(54): (col. 7) remark: *MIC* loop was not vectorized: not inner
loop
offloadmul.c(53): (col. 5) remark: *MIC* loop was not vectorized: not inner
loop
```

`-vec-report2` deprecated in icc >15.0. Use `-qopt-report=n -qopt-report-phase=vec` and view *.optrpt files.

Intel Offload: Example

```
__attribute__((target(mic))) void mxm( int n, double * restrict a, double * restrict b,
                                                                double *restrict c ){
    int i,j,k;
    for( i = 0; i < n; i++ ) {
     ...}
}

main(){
...
    #pragma offload target(mic) in(a,b:length(n*n)) inout(c:length(n*n))
    {
        mxm(n,a,b,c);
    }
}
```

Offload Diagnostics

```
u65fok@i01r13c06:~/tests> export OFFLOAD_REPORT=2

lu65fok@i01r13c06:~/tests> ./offloadmul
[Offload] [MIC 0] [File]            offloadmul.c
[Offload] [MIC 0] [Line]            50
[Offload] [MIC 0] [Tag]             Tag 0
[Offload] [HOST]  [Tag 0] [CPU Time]        51.927456(seconds)
[Offload] [MIC 0] [Tag 0] [CPU->MIC Data]   24000016 (bytes)
[Offload] [MIC 0] [Tag 0] [MIC Time]        50.835065(seconds)
[Offload] [MIC 0] [Tag 0] [MIC->CPU Data]   8000016 (bytes)
```

Offload Diagnostics

```
lu65fok@i01r13c06:~/tests> export H_TRACE=1

lu65fok@i01r13c06:~/tests> ./offloadmul
HOST:  Offload function
__offload_entry_offloadmul_c_50mainicc638762473Jnx4JU,
is_empty=0, #varDescs=7, #waits=0, signal=none
HOST:  Total pointer data sent to target: [24000000] bytes
HOST:  Total copyin data sent to target: [16] bytes
HOST:  Total pointer data received from target: [8000000] bytes
MIC0:  Total copyin data received from host: [16] bytes
MIC0:  Total copyout data sent to host: [16] bytes
HOST:  Total copyout data received from target: [16] bytes
lu65fok@i01r13c06:~/tests>
```

Offload Diagnostics

```
lu65fok@i01r13c06:~/tests> export H_TIME=1

lu65fok@i01r13c06:~/tests> ./offloadmul
[Offload] [MIC 0] [File]          offloadmul.c
[Offload] [MIC 0] [Line]           50
[Offload] [MIC 0] [Tag]            Tag 0
[Offload] [HOST]  [Tag 0] [CPU Time]       51.920016(seconds)
[Offload] [MIC 0] [Tag 0] [MIC Time]       50.831497(seconds)
*************************************************************
                        timer data       (sec)
*************************************************************
lu65fok@i01r13c06:~/tests>
```

Environment Variables

- Host environment variables are automatically forwarded to the coprocessor when offload mode is used.
- To avoid names collisions:
  - Set MIC_ENVIRONMENT_PREFIX=MIC on the host
  - Then only names with prefix MIC_ are forwarded to the coprocessor with prefix stripped
  - Exception: MIC_LD_LIBRARY_PATH is never passed to the coprocessor.
  - Value of LD_LIBRARY_PATH cannot be changed via forwarding of environment variables.

Environment Variables on the MIC

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

int main(){
#pragma offload target (mic)
 {
   char* varmic = getenv("VAR");
   if (varmic) {
     printf("VAR=%s on MIC.\n", varmic);
   } else {
     printf("VAR is not defined on MIC.\n");
   }
 }
 char* varhost = getenv("VAR");
 if (varhost) {
   printf("VAR=%s on host.\n", varhost);
 } else {
   printf("VAR is not defined on host.\n");
 }
}
```

Environment Variables on the MIC

```
lu65fok@i01r13c01:~/tests> ./env
VAR is not defined on host.
VAR is not defined on MIC.
lu65fok@i01r13c01:~/tests> export VAR=299792458
lu65fok@i01r13c01:~/tests> ./env
VAR=299792458 on host.
VAR=299792458 on MIC.
lu65fok@i01r13c01:~/tests> export MIC_ENV_PREFIX=MIC
lu65fok@i01r13c01:~/tests> ./env
VAR=299792458 on host.
VAR is not defined on MIC.
lu65fok@i01r13c01:~/tests> export MIC_VAR=3.141592653
lu65fok@i01r13c01:~/tests> ./env
VAR=299792458 on host.
VAR=3.141592653 on MIC.
```

The Preprocessor Macro __MIC__

- The macro __MIC__ is only defined in code version for MIC, not in the fallback version for the host
- Allows to check where the code is running.
- Allows to write multiversioned code.
- __MIC__ also defined in native mode.

The Preprocessor Macro __MIC__

```
#pragma offload target(mic)
{
    #ifdef __MIC__
        printf("Hello from MIC (offload succeeded).\n");
    #else
        printf("Hello from host (offload to MIC failed!).\n");
    #endif
}
```

```
lu65fok@login12:~/tests> icpc -offload=optional offload-mic.c
lu65fok@login12:~/tests> ./a.out
Hello from host (offload to MIC failed!).
lu65fok@i01r13c06:~/tests> ./a.out
Hello from MIC (offload succeeded).
```

Lab: Offload Mode I

Intel Xeon Phi Programming Models: Intel Language Extensions for Offload (LEO) II

Data Traffic without Computation

- 2 possibilities:
  - Blank body of #pragma offload, i.e.
    #pragma offload target(mic) in (data: length(n))
    {}
  - Use a special pragma offload_transfer, i.e.
    #pragma offload_transfer target(mic) in(data: length(n))

Asynchronous Offload

- Asynchronous Data Transfer helps to:
  - Overlap computations on host and MIC(s).
  - Work can be distributed to multiple coprocessors.
  - Data transfer time can be masked.

Asynchronous Offload

Any pointer type variable can serve as a signal!

- To allow asynchronous data transfer, the specifiers *signal()* and *wait()* can be used, i.e.

```
#pragma offload_transfer target(mic:0) in(data : length(n))
signal(data)
// work on other data concurrent to data transfer …
#pragma offload target(mic:0) wait(data) \
nocopy(data : length(N)) out(result : length(N))
{
    ….
   result[i]=data[i] + …;
}
```

Device number must be specified!

Asynchronous Offload

- Alternative to the wait() clause, a new pragma can be used:
  #pragma offload_wait target(mic:0) wait(data)

- Useful if no other offload or data transfer is necessary at the synchronisation point.

Asynchronous Offload to Multiple Coprocessors

```
char* offload0;
char* offload1;
#pragma offload target(mic:0) signal(offload0) \
                in(data0 : length(N)) out(result0 : length(N))
{
    Calculate(data0, result0);
}
#pragma offload target(mic:1) signal(offload1) \
                in(data1 : length(N)) out(result1 : length(N))
{
      Calculate(data1, result1);
}
#pragma offload_wait target(mic:0) wait(offload0)
#pragma offload_wait target(mic:1) wait(offload1)
```

Explicit Worksharing

```
#pragma omp parallel
{
   #pragma omp sections
   {
       #pragma omp section
       {
       //section running on the coprocessor
       #pragma offload target(mic) in(a,b:length(n*n)) inout(c:length(n*n))
          {
           mxm(n,a,b,c);
          }
       }
       #pragma omp section
       {
       //section running on the host
       mxm(n,d,e,f);
       }
    }
}
```

Persistent Data

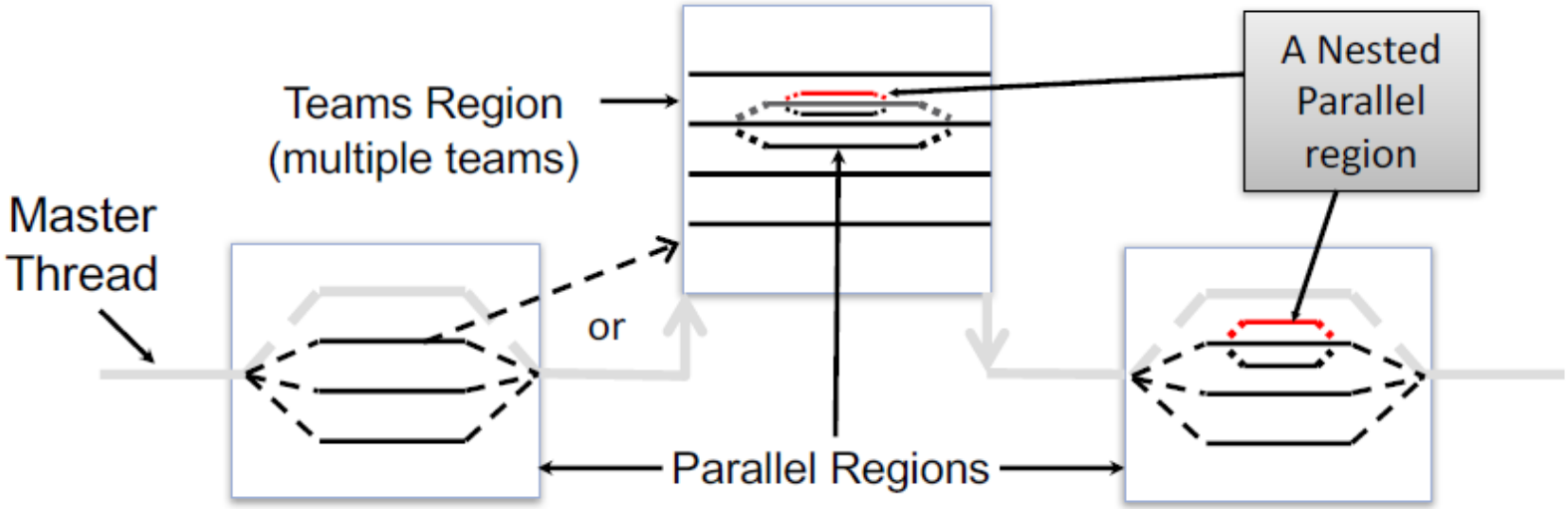
- #define ALLOC alloc_if(1)
  #define FREE free_if(1)
  #define RETAIN free_if(0)
  #define REUSE alloc_if(0)

- To allocate data and keep it for the next offload:
  #pragma offload target(mic) in (p:length(l) ALLOC RETAIN)
- To reuse the data and still keep it on the coprocessor:
  #pragma offload target(mic) in (p:length(l) REUSE RETAIN)
- To reuse the data again and free the memory. (FREE is the default, and does not need to be explicitly specified):
  #pragma offload target(mic) in (p:length(l) REUSE FREE)

Intel Xeon Phi Programming Models: OpenMP 4.x Offload Mode

OpenMP 4.x Execution Model

- Create and destroy threads,
- create and destroy leagues of thread teams,
- assign / distribute work (tasks) to threads and devices,
- specify which data is shared and which is private,
- specify which data must be available to the device,
- coordinate thread access to shared data.



OpenMP 4.x Device Constructs

- Execute code on a target device
  - **omp target** *[clause[[,] clause],…]*
    structured-block
- Manage the device data environment
  - **map** *([map-type:] list) // map clause*
    map-type := ***alloc*** | ***tofrom*** | ***to*** | ***from***
  - **omp target data** *[clause[[,] clause],…]*
    structured-block
  - **omp target update** *[clause[[,] clause],…]*
  - **omp declare target**
    [variable-definitions-or-declarations]
  - **omp target enter / exit data** *[clause[[,] clause],…]* (new: OpenMP 4.5)
- Workshare for acceleration
  - **omp teams** *[clause[[,] clause],…]*
    structured-block
  - **omp distribute** *[clause[[,] clause],…]*
    for-loops

OpenMP 4.x Offloading Computation

- Use **target** construct to
  - Transfer control from the host to the target device
  - Map variables between the host and target device data environments
- Host thread waits until offloaded region completed
- Use **nowait** for asynchronous execution

-
```
#pragma omp target map(to:b,c,d) map(from:a)
 {
#pragma omp parallel for
      for (i=0; i<count; i++) {
      a[i] = b[i] * c + d;
      }
 }
```

OpenMP 4.x Target Construct

- Map variables to a device data environment and execute the construct on that device.
- **#pragma omp target** *[clause[[,] clause] ...] new-line structured-block*
- where *clause* is one of the following:
  - **if(***[* **target :***]* *scalar-expression***)**
  - **device(***integer-expression***)**
  - **private(***list***)**
  - **firstprivate(***list***)**
  - **map(***[[map-type-modifier[,]] map-type***:** *] list***)**
  - **is_device_ptr(***list***)**
  - **defaultmap(tofrom:scalar)**
  - **nowait**
  - **depend(***dependence-type***:** *list***)**

map Clause

```
extern void init(float*, float*, int);
extern void output(float*, int);

void vec_mult(float *p, float *v1, float *v2, int N)
{
   int i;
   init(v1, v2, N);

   #pragma omp target map(to:v1[0:N],v2[:N]) map(from:p[0:N])
   #pragma omp parallel for
   for (i=0; i<N; i++)
     p[i] = v1[i] * v2[i];

   output(p, N);
}
```

```
module mults
contains
subroutine vec_mult(p,v1,v2,N)
   real,dimension(*) :: p, v1, v2
   integer :: N,i
   call init(v1, v2, N)
   !$omp target map(to: v1(1:N), v2(:N)) map(from:p(1:N))
   !$omp parallel do
   do i=1,N
      p(i) = v1(i) * v2(i)
   end do
   !omp end target
   call output(p, N)
end subroutine
end module
```

- On entry to the target region:
  - Allocate corresponding variables v1, v2, and p in the device data environment.
  - Assign the corresponding variables v1 and v2 the value of their respective original variables.
  - The corresponding variable p is undefined.
- On exit from the target region:
  - Assign the original variable p the value of its corresponding variable.
  - The original variables v1 and v2 are undefined.
  - Remove the corresponding variables v1, v2, and p from the device data environment

OpenMP 4.x Data mapping

- Map variables to a device data environment for the extent of the region:
  - **#pragma omp target data** *clause[[[,] clause] ...] new-line structured-block*

- Alternatively use 2 standalone directives
  - **#pragma omp target enter data** *[clause[[,] clause]...] new-line*
  - **...**
  - **#pragma omp target exit data** *[clause[[,] clause]...] new-line*

- Standalone directive to synchronize data
  - **#pragma omp target update clause[[[,] clause] ...] new-line**

OpenMP 4.x Data mapping

```
#pragma omp target data map(alloc:tmp[:N]) map(to:input[:N]) map(from:res)
  {
#pragma omp target
#pragma omp parallel for
    for (i=0; i<N; i++)
      tmp[i] = some_computation(input[i], i);

    update_input_array_on_the_host(input);

#pragma omp target update to(input[:N])

#pragma omp target
#pragma omp parallel for reduction(+:res)
    for (i=0; i<N; i++)
      res += final_computation(input[i], tmp[i], i)
  }
```

host
target
host
target
host

OpenMP 4.x Teams construct

- The **teams** construct creates a league of thread teams and the master thread of each team executes the region.
- **#pragma omp teams** *[clause[[,] clause] ...] new-line*
  structured-block
- where *clause* is one of the following:
  - **num_teams(***integer-expression***)**
  - **thread_limit(***integer-expression***)**
  - **default(shared | none)**
  - **private(***list***)**
  - **firstprivate(***list***)**
  - **shared(***list***)**
  - **reduction(***reduction-identifier* **:** *list***)**
- The **teams** construct creates a *league* of thread teams
  - The master thread of each team executes the **teams** region
  - The (max.) number of teams is specified by the **num_teams** clause
  - Each team executes with (max.) **thread_limit** threads
  - Threads in different teams cannot synchronize with each other

OpenMP 4.x Distribute Construct

- The **distribute** construct specifies that the iterations of one or more loops will be executed by the thread teams in the context of their implicit tasks. The iterations are distributed across the master threads of all teams that execute the **teams** region to which the **distribute** region binds.
- **#pragma omp distribute** *[clause[[,] clause] ...] new-line for-loops*
- Where *clause* is one of the following:
  - **private(***list***)**
  - **firstprivate(***list***)**
  - **lastprivate(***list***)**
  - **collapse(***n***)**
  - **dist_schedule(***kind[***, *chunk_size]***)**

Composite constructs and shortcuts in OpenMP 4.5

- 2.10.9 omp distribute simd
- 2.10.10 omp distribute parallel for
- 2.10.11 omp distribute parallel for simd
- 2.11.5 omp target parallel
- 2.11.6 omp target parallel for
- 2.11.7 omp target parallel for simd
- 2.11.8 omp target simd
- 2.11.9 omp target teams
- 2.11.10 omp teams distribute
- 2.11.11 omp teams distribute simd
- 2.11.12 omp target teams distribute
- 2.11.13 omp target teams distribute simd
- 2.11.14 omp teams distribute parallel for
- 2.11.15 omp target teams distribute parallel for
- 2.11.16 omp teams distribute parallel for simd
- 2.11.17 omp target teams distribute parallel for simd

OpenMP 4.x Composite constructs and shortcuts

- **omp distribute** — Iterations distributed across the master threads of all teams in a teams region
  - **omp distribute** simd — dito + executed concurrently using SIMD instructions
  - **omp distribute** parallel for — executed in parallel by multiple threads that are members of multiple teams
  - **omp distribute** parallel for simd — dito + executed concurrently using SIMD instructions

- **omp teams** — creates a league of thread teams and the master thread of each team executes the region
  - **omp teams** distribute
  - **omp teams** distribute simd
  - **omp teams** distribute parallel for
  - **omp teams** distribute parallel for simd

OpenMP 4.5

- **omp target** — map variables to a device data environment and execute the construct on that device
  - **omp target** simd
  - **omp target** parallel
  - **omp target** parallel for
  - **omp target** parallel for simd

- omp target teams
  - omp target teams distribute
  - omp target teams distribute simd
  - omp target teams distribute parallel for
  - omp target teams distribute parallel for simd

OpenMP 4.x SuperMIC Test

- **#pragma omp target**
- *u65fok@i01r13c06:~> ./a.out*
  Hello world from host: I have 32 cores omp_get_default_device=0
  omp_get_num_devices=2 omp_get_num_teams=1 omp_get_team_num=0
  omp_is_initial_device=1
  Hello world from MIC i01r13c06-mic0: I have 240 cores omp_get_num_threads=1
- **#pragma omp target teams**
  Hello world from MIC i01r13c06-mic0: I have 240 cores omp_get_num_threads=236
- **#pragma omp target teams num_teams(4)**
  Hello world from MIC i01r13c06-mic0: I have 240 cores omp_get_num_threads=59
  Hello world from MIC i01r13c06-mic0: I have 240 cores omp_get_num_threads=59
  Hello world from MIC i01r13c06-mic0: I have 240 cores omp_get_num_threads=59
  Hello world from MIC i01r13c06-mic0: I have 240 cores omp_get_num_threads=59
- **#pragma omp target teams num_teams(4) thread_limit(2)**
  Hello world from MIC i01r13c06-mic0: I have 240 cores omp_get_num_threads=2
  Hello world from MIC i01r13c06-mic0: I have 240 cores omp_get_num_threads=2
  Hello world from MIC i01r13c06-mic0: I have 240 cores omp_get_num_threads=2
  Hello world from MIC i01r13c06-mic0: I have 240 cores omp_get_num_threads=2

Intel Xeon Phi Programming Models: Intel “Mine Yours Ours” (MYO) virtual shared memory model

MYO

- “Mine Yours Ours” virtual shared memory model.
- Alternative to Offload approach. Only available in C++.
- Allows to share not bit-wise compatible complex data (like structures with pointer elements, C++ classes) without data marshalling. LEO Offload Model only allows offloading of bitwise-copyable data!
- Allocation of data at the same virtual addresses on the host and the coprocessor.
- Runtime automatically maintains coherence.
- Syntax based on the keywords __Cilk_shared and __Cilk_offload.

MYO: Example

```
#define N 10000
_Cilk_shared int a[N], b[N], c[N];

 _Cilk_shared void add() {
 for (int i = 0; i < N; i++)
     c[i] = a[i] + b[i];
}

 int main(int argc, char *argv[]) {
 …
 _Cilk_offload add(); // Function call on coprocessor:
 …
}
```

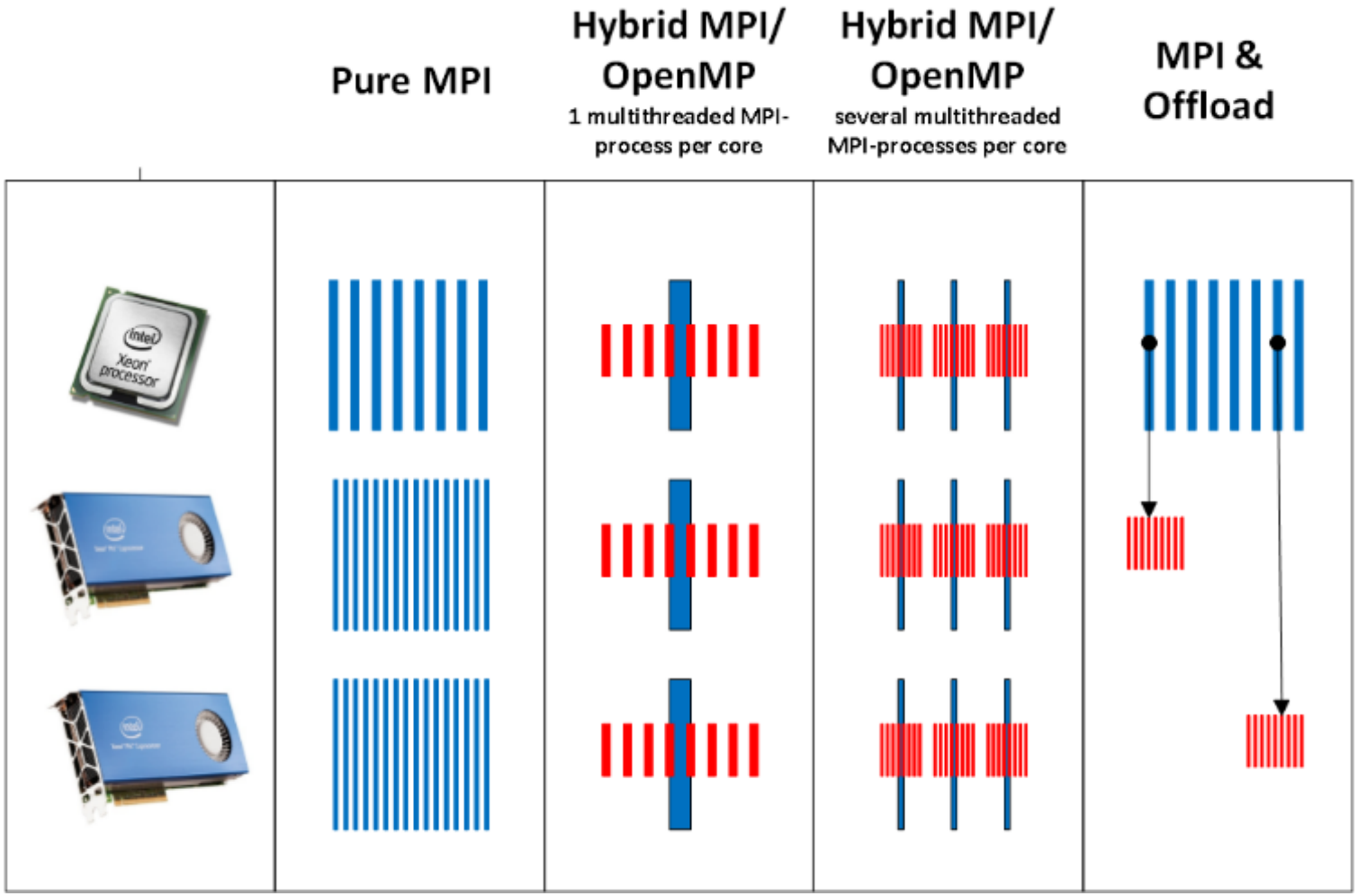
MYO Language Extensions

Entity	Syntax	Semantics
Function	int _Cilk_shared f(int x){…}	Executable code for both host and MIC; may be called from either side
Global variable	_Cilk_shared int x = 0	Visible on both sides
File/Function static	static _Cilk_shared int x	Visible on both sides, only to code within the file/function
Class	class _Cilk_shared x {...}	Class methods, members, and operators are available on both sides
Pointer to shared data	int _Cilk_shared *p	p is local (not shared), can point to shared data
A shared pointer	int *_Cilk_shared p	p is shared, should only point at shared data
Offloading a function call	x = _Cilk_offload func(y)	func executes on MIC if possible
	x = _Cilk_offload_to(n) func	func must be executed on specified (n-th) MIC
Offloading asynchronously	_Cilk_spawn _Cilk_offload func(y)	Non-blocking offload
Offload a parallel for-loop	_Cilk_offload _Cilk_for(i=0; i<N; i++) {…}	Loop executes in parallel on MIC

Lab: Offload Mode II

Intel Xeon Phi Programming Models: MPI

MPI on Hosts & MICs



Important MPI environment variables

- Important Paths are already set by intel module, otherwise use:
  - . $ICC_BASE/bin/compilervars.sh intel64
  - . $MPI_BASE/bin64/mpivars.sh
- Recommended environment on Salomon:

```
module load intel
export I_MPI_HYDRA_BOOTSTRAP=ssh
export I_MPI_MIC=enable
export I_MPI_FABRICS=shm:dapl
export I_MPI_DAPL_PROVIDER_LIST=ofa-v2-mlx4_0-1u,ofa-
v2-scif0,ofa-v2-mcm-1
export MIC_LD_LIBRARY_PATH =
$MIC_LD_LIBRARY_PATH:/apps/all/impi/5.1.2.150-iccifort-
2016.1.150-GCC-4.9.3-2.25/mic/lib/ depending on version
```

Invocation of the Intel MPI compiler

Language	MPI Compiler	Compiler
C	mpiicc	icc
C++	mpiicpc	icpc
Fortran	mpiifort	ifort

I_MPI_FABRICS

- The following network fabrics are available for the Intel Xeon Phi coprocessor:

shm	Shared-memory
tcp	TCP/IP-capable network fabrics, such as Ethernet and InfiniBand (through IPoIB)
ofa	OFA-capable network fabric including InfiniBand (through OFED verbs)
dapl	DAPL–capable network fabrics, such as InfiniBand, iWarp, Dolphin, and XPMEM (through DAPL)

I_MPI_FABRICS

- The default can be changed by setting the I_MPI_FABRICS environment variable to I_MPI_FABRICS=<fabric> or I_MPI_FABRICS= <intra-node fabric>:<inter-nodes fabric>

- Intranode: Shared Memory, Internode: DAPL (Default on SuperMIC/MUC)
  - export I_MPI_FABRICS=shm:dapl

- Intranode: Shared Memory, Internode: TCP (Can be used in case of Infiniband problems)
  - export I_MPI_FABRICS=shm:tcp

Sample MPI Program

```
lu65fok@login12:~/tests> cat testmpi.c
#include <stdio.h>
#include <mpi.h>

int main (int argc, char* argv[]) {
  char hostname[100];
  int rank, size;

  MPI_Init (&argc, &argv);      /* starts MPI */
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);       /* get current process id */
  MPI_Comm_size (MPI_COMM_WORLD, &size);       /* get number of processes */

  gethostname(hostname,100);
  printf( "Hello world from process %d of %d: host: %s\n", rank, size, hostname);
  MPI_Finalize();
  return 0;
}
```

MPI on hosts

- Compile for host using mpiicc / mpiifort:

  lu65fok@login12:~/tests> mpiicc testmpi.c -o testmpi-host

- Run 2 MPI tasks on host node i01r13a01

  lu65fok@login12:~/tests> mpiexec -n 2 -host i01r13a01 ./testmpi-host

  Hello world from process 0 of 2: host: i01r13a01

  Hello world from process 1 of 2: host: i01r13a01

MPI in native mode on 1 MIC

- Compile for MIC using mpiicc / mpiifort -mmic:

```
lu65fok@login12:~/tests> mpiicc  -mmic testmpi.c -o testmpi-mic
```

- Copy binary to MIC:

```
lu65fok@login12:~/tests> scp testmpi-mic  i01r13a01-mic0:
```

- Launch 2 MPI tasks from MIC node i01r13a01-mic0

```
lu65fok@i01r13a04:~/tests> ssh i01r13a01-mic0
[lu65fok@i01r13a01-mic0 ~]$ mpiexec -n 2 ./testmpi-mic
Hello world from process 1 of 2: host: i01r13a01-mic0
Hello world from process 0 of 2: host: i01r13a01-mic0
```

Do not mix up with mpicc and mpifort!!

```
lu65fok@login12:~/tests> mpicc  -mmic testmpi.c -o testmpi-mic
/usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../x86_64-suse-linux/bin/ld: skipping incompatible
/lrz/sys/intel/mpi_41_3_048/mic/lib/libmpigf.so when searching for -lmpigf
/usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../x86_64-suse-linux/bin/ld: skipping incompatible
/lrz/sys/intel/mpi_41_3_048/mic/lib/libmpigf.a when searching for -lmpigf
/usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../x86_64-suse-linux/bin/ld: cannot find -lmpigf
/usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../x86_64-suse-linux/bin/ld: skipping incompatible
/lrz/sys/intel/mpi_41_3_048/mic/lib/libmpi.so when searching for -lmpi
/usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../x86_64-suse-linux/bin/ld: skipping incompatible
/lrz/sys/intel/mpi_41_3_048/mic/lib/libmpi.a when searching for -lmpi
/usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../x86_64-suse-linux/bin/ld: cannot find -lmpi
/usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../x86_64-suse-linux/bin/ld: skipping incompatible
/lrz/sys/intel/mpi_41_3_048/mic/lib/libmpigi.a when searching for -lmpigi
/usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../x86_64-suse-linux/bin/ld: cannot find -lmpigi
collect2: ld returned 1 exit status
```

MPI on 1 MIC

- Compile for MIC using mpiicc / mpiifort -mmic:

  lu65fok@login12:~/tests> mpiicc -mmic testmpi.c -o testmpi-mic

- Copy binary to MIC

  (not necessary if home is mounted on MICs)

  lu65fok@login12:~/tests> scp testmpi-mic i01r13a01-mic0:

- Run 2 MPI tasks on MIC node i01r13a01-mic0

  lu65fok@i01r13a04:~/tests> mpiexec -n 2 -host i01r13a01-mic0 ./home/lu65fok/testmpi-mic

  Hello world from process 1 of 2: host: i01r13a01-mic0

  Hello world from process 0 of 2: host: i01r13a01-mic0

Full path needed!

MPI on 2 MICs

- Compile for MIC using mpiicc / mpiifort -mmic:

  lu65fok@login12:~/tests> mpiicc -mmic testmpi.c -o testmpi-mic

- Copy binary to MICs:

  (not necessary if home is mounted on MICs)

  lu65fok@login12:~/tests> scp testmpi-mic i01r13a01-mic0:

  lu65fok@login12:~/tests> scp testmpi-mic i01r13a01-mic1:

- Run 2 MPI tasks on MIC node i01r13a01-mic0

  lu65fok@login12:~/tests> mpiexec -n 2 -perhost 1 -host i01r13a01-mic0,i01r13a01-mic1 ./home/lu65fok/testmpi-mic

  Hello world from process 1 of 2: host: i01r13a01-mic1

  Hello world from process 0 of 2: host: i01r13a01-mic0

MPI on Host and 2 MICs attached to the host

lu65fok@login12:~/tests> mpirun -n 1 -host i01r13a01 ./testmpi-host : -n 1 -host i01r13a01-mic0 /home/lu65fok/testmpi-mic : -n 1 -host i01r13a01-mic1 /home/lu65fok/testmpi-mic

Hello world from process 0 of 3: host: i01r13a01

Hello world from process 2 of 3: host: i01r13a01-mic1

Hello world from process 1 of 3: host: i01r13a01-mic0

MPI on multiple Hosts & MICs

```
lu65fok@i01r13a01:~/tests> mpirun -n 1 -host i01r13a01 ./testmpi-host : -n 1 -host
i01r13a01-mic0 /home/lu65fok/testmpi-mic : -n 1 -host i01r13a01-mic1
/home/lu65fok/testmpi-mic : -n 1 -host i01r13a02 ./testmpi-host : -n 1 -host
i01r13a02-mic0 /home/lu65fok/testmpi-mic : -n 1 -host i01r13a02-mic1
/home/lu65fok/testmpi-mic
```

```
Hello world from process 3 of 6: host: i01r13a02
Hello world from process 0 of 6: host: i01r13a01
Hello world from process 2 of 6: host: i01r13a01-mic1
Hello world from process 5 of 6: host: i01r13a02-mic1
Hello world from process 1 of 6: host: i01r13a01-mic0
Hello world from process 4 of 6: host: i01r13a02-mic0
```

MPI Machine File

```
lu65fok@login12:~/tests> cat machinefile.txt
i01r13a01-mic0
i01r13a01-mic1
i01r13a02-mic0
i01r13a02-mic1

lu65fok@login12:~/tests> mpirun  -n 4 -machinefile machinefile.txt
/home/lu65fok/testmpi-mic
Hello world from process 3 of 4: host: i01r13a02-mic1
Hello world from process 2 of 4: host: i01r13a02-mic0
Hello world from process 1 of 4: host: i01r13a01-mic1
Hello world from process 0 of 4: host: i01r13a01-mic0
```

MPI Machine File

```
lu65fok@login12:~/tests> cat machinefile.txt
i01r13a01-mic0:2
i01r13a01-mic1
i01r13a02-mic0
i01r13a02-mic1

lu65fok@login12:~/tests> mpirun  -n 4 -machinefile machinefile.txt
/home/lu65fok/testmpi-mic
Hello world from process 3 of 4: host: i01r13a02-mic0
Hello world from process 0 of 4: host: i01r13a01-mic0
Hello world from process 2 of 4: host: i01r13a01-mic1
Hello world from process 1 of 4: host: i01r13a01-mic0
```

Offload from MPI Tasks

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

int main (int argc, char* argv[]) {
  char hostname[100];
  int rank, size;
  MPI_Init (&argc, &argv);      /* starts MPI */
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);        /* get current process id */
  MPI_Comm_size (MPI_COMM_WORLD, &size);        /* get number of processes */

  gethostname(hostname,100);

  #pragma offload target(mic)
{
    char michostname[50];
    gethostname(michostname, 50);
    printf("MIC: I am %s and I have %ld logical cores. I was called by  process %d of %d: host: %s \n", michostname,
        sysconf(_SC_NPROCESSORS_ONLN), rank, size, hostname);
  }
  printf( "Hello world from process %d of %d: host: %s\n", rank, size, hostname);
  MPI_Finalize();
  return 0;

}
```

Offload from MPI Tasks using 1 host

```
lu65fok@login12:~/tests> mpiicc testmpioffload.c -o testmpioffload
lu65fok@login12:~/tests> mpirun -n 4  -host i01r13a01 ./testmpioffload
Hello world from process 3 of 4: host: i01r13a01
Hello world from process 1 of 4: host: i01r13a01
Hello world from process 0 of 4: host: i01r13a01
Hello world from process 2 of 4: host: i01r13a01
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by process 3 of 4: host: i01r13a01
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by process 0 of 4: host: i01r13a01
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by process 1 of 4: host: i01r13a01
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by process 2 of 4: host: i01r13a01
```

Offload from MPI Tasks using multiple hosts

```
lu65fok@login12:~/tests> mpirun -n 4 -perhost 2   -host
i01r13a01,i01r13a02 ./testmpioffload
Hello world from process 2 of 4: host: i01r13a02
Hello world from process 0 of 4: host: i01r13a01
Hello world from process 3 of 4: host: i01r13a02
Hello world from process 1 of 4: host: i01r13a01
MIC: I am i01r13a02-mic0 and I have 240 logical cores. I was called by
process 2 of 4: host: i01r13a02
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by
process 1 of 4: host: i01r13a01
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by
process 0 of 4: host: i01r13a01
MIC: I am i01r13a02-mic0 and I have 240 logical cores. I was called by
process 3 of 4: host: i01r13a02
```

Offload from MPI Tasks: Using both MICs

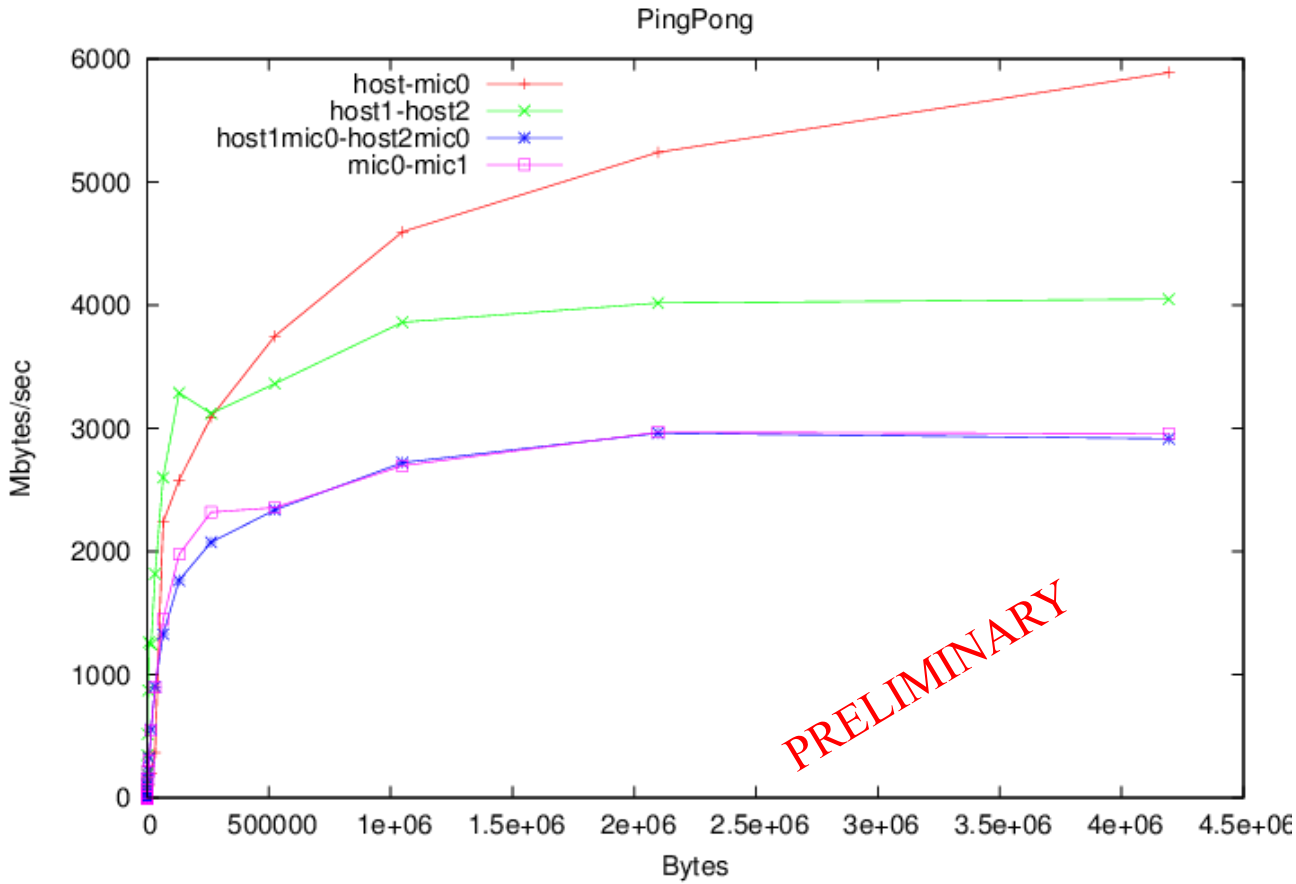
```
#pragma offload target(mic:rank%2)
 {
   char michostname[50];
   gethostname(michostname, sizeof(michostname));
   printf("MIC: I am %s and I have %ld logical cores. I was called by
              process %d of %d: host: %s \n", michostname,
              sysconf(_SC_NPROCESSORS_ONLN), rank, size,
hostname);
 }
```

Offload from MPI Tasks: Using both MICs

```
lu65fok@login12:~/tests> mpirun -n 4 -perhost 2   -host
i01r13a01,i01r13a02 ./testmpioffload
Hello world from process 0 of 4: host: i01r13a01
Hello world from process 2 of 4: host: i01r13a02
Hello world from process 3 of 4: host: i01r13a02
Hello world from process 1 of 4: host: i01r13a01
MIC: I am i01r13a02-mic1 and I have 240 logical cores. I was called
by  process 3 of 4: host: i01r13a02
MIC: I am i01r13a01-mic1 and I have 240 logical cores. I was called
by  process 1 of 4: host: i01r13a01
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called
by  process 0 of 4: host: i01r13a01
MIC: I am i01r13a02-mic0 and I have 240 logical cores. I was called
by  process 2 of 4: host: i01r13a02
```

Lab: MPI

Intel MPI Benchmarks: PingPong Bandwidth



SuperMIC: QPI agents of the CPU not involved
PCIe 2 host interface offers 6.2 GB/s
Important for host-mic: I_MPI_DAPL_PROVIDER_LIST=ofa-v2-mlx4_0-1,ofa-v2-scif0

Xeon Phi References

- Books:
  - *James Reinders, James Jeffers, Intel Xeon Phi Coprocessor High Performance Programming, Morgan Kaufman Publ. Inc., 2013 http://lotsofcores.com ; new KNL edition in July 2016*
  - *Rezaur Rahman: Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers, Apress 2013 .*
  - *Parallel Programming and Optimization with Intel Xeon Phi Coprocessors, Colfax 2013 http://www.colfaxintl.com/nd/xeonphi/book.aspx*
- Intel Xeon Phi Programming, Training material, CAPS
- Intel Training Material and Webinars
- V. Weinberg (Editor) et al., Best Practice Guide - Intel Xeon Phi, http://www.prace-ri.eu/best-practice-guide-intel-xeon-phi-january-2017/ and references therein.

Acknowledgements

- **IT4Innovation**, Ostrava.
- Partnership for Advanced Computing in Europe (**PRACE**)
- **Intel**
- **BMBF** (Federal Ministry of Education and Research)
- **Dr. Karl Fürlinger (LMU)**

Thank you for your participation!