Introduction of Xeon Phi Programming

Wei Feinstein
Shaohao Chen

HPC User Services
LSU HPC/LONI

Louisiana State University
Overview

• Intel Xeon Phi architecture
• Programming models
  • Native mode
  • Offload execution
• Intel MKL (Math Kernel Library)
• Heterogeneous computing on MIC architecture
  • Inter-operation of offload, MPI and OpenMP
• Optimization, debugging and profiling
Multiple cores to many cores

10/21/2015 Introduction to Xeon Phi Programming
Multi-core vs. Many-core

**Intel Xeon**
- Multiple cores

**Intel Xeon Phi**
- Many Integrated Cores (MIC)
## Intel Xeon Phi processor vs coprocessor

### Table Comparison

<table>
<thead>
<tr>
<th></th>
<th>Xeon “Ivy Bridge” Processor</th>
<th>Xeon Phi “Knight’s Corner” Coprocessor</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Standard Linux</td>
<td>Special Linux distribution</td>
</tr>
<tr>
<td>Number of Cores</td>
<td>10</td>
<td>61</td>
</tr>
<tr>
<td>Single core frequency</td>
<td>2.8 GHz</td>
<td>1.2 GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>Up to 64 GB of DDR3</td>
<td>16 GB cached GDDR5</td>
</tr>
<tr>
<td>Hyper-threading</td>
<td>2-way</td>
<td>4-way</td>
</tr>
<tr>
<td>Vector length</td>
<td>256-bit AVX</td>
<td>512-bit SIMD</td>
</tr>
<tr>
<td>Peak processing power</td>
<td>0.224 TFLOPS</td>
<td>1.208 TFLOPS</td>
</tr>
</tbody>
</table>

2 Xeon ≈ 1 Xeon Phi
Xeon Processor

32~64 GB
~50 GB/s
Numa Region

32~64 GB
~50 GB/s
Numa Region

SIMD Unit
~220 GFlops

Phi Coprocessor

SIMD Unit

8 GB

320 GB/s

6 GB/s

~1.2 TFlops
Theoretical Maximums
(2S Intel® Xeon® processor E5-2670 & E5-2697v2 vs. Intel® Xeon Phi™ coprocessor)

**Single Precision**
(GF/s)
- E5-2670: 1,037
- E5-2697v2: 1,001
- 3120PA: 1,011
- 5110P: 1,011
- 51200: 1,053
- 7120PX/1D: 1,208

**Double Precision**
(GF/s)
- E5-2670: 518
- E5-2697v2: 518
- 3120PA: 1,053
- 5110P: 1,053
- 51200: 1,053
- 7120PX/1D: 1,238

**Memory Bandwidth**
(GB/s)
- E5-2670: 333
- E5-2697v2: 102
- 3120PA: 320
- 5110P: 352
- 51200: 352
- 7120PX/1D: 352

Updated

Source from Intel website

10/21/2015
Introduction to Xeon Phi Programming
SuperMIC at LSU

360 Compute Nodes
• Two 2.8GHz 10-Core Ivy Bridge-EP E5-2680 Xeon 64-bit Processors
• Two Intel Xeon Phi 7120P Coprocessors
• 64GB DDR3 1866MHz Ram (processors)
• 8 GB GDDR5 (coprocessors)
• 500GB HD
• 56 Gigabit/sec Infiniband network interface
• 1 Gigabit Ethernet network interface

Getting started …

Terminal 1 (run jobs)

• ssh username@smic.hpc.lsu.edu # login SuperMIC
• qsub -l -A allocation_name -l nodes=2:ppn=20,walltime=hh:mm:ss

Terminal 2/3 (monitor performance)

• ssh -X username@smic.hpc.lsu.edu # login SuperMIC with graphics
• ssh -X smic{number} # login the compute node with graphics
• micsmc & (or micsmc-gui & ) # open Xeon phi monitor from the host
• ssh mic0 # login mic0
• top # monitor processes on Xeon Phi
Programming models on MIC

**Native execution**

```c
main() {
    myFunction();
}
```

**Offload execution**

```c
main() {
    #pragma offload target(mic)
    myFunction();
}
```
Native execution

“Hello World” application:

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    printf("Hello world! I have %ld logical cores. \n", \n        sysconf(_SC_NPROCESSORS_ONLN ));
}
```

Compile and run on host:

```
user@host module load intel/14.0
user@host icpc hello.cc -o hello.cpu
user@host ./hello.cpu
Hello world! I have 32 logical cores.
```
Native execution

“Hello World” application:

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
    printf("Hello world! I have %ld logical cores. \n", \n        sysconf(_SC_NPROCESSORS_ONLN ));
}
```

Compile and run on the device:

```
user@host  icpc -mmic hello.cc -o hello.mic
user@host  micnativeloadex ./hello.mic
Hello world! I have 244 logical cores.

user@host  scp hello.mic mic0:~
user@host  ssh mic0 ./hello.mic
```
#include <stdio.h>
#include <omp.h>

int main( void ) {
    ...

    // Allocate memory aligned to a 64 byte boundary
    x = (double *)memalign(64,N*sizeof(double));
    y = (double *)memalign(64,N*sizeof(double));
    z = (double *)memalign(64,N*sizeof(double));
    int seed = 8719;
    srand(seed);
    for( i = 0; i < N; i++) {
        x[i] = rand()/(double)RAND_MAX;
        y[i] = rand()/(double)RAND_MAX;
        z[i] = rand()/(double)RAND_MAX;
    }
    #pragma omp parallel private(i) {
        Threads = omp_get_num_threads();
        for(i = 0; i < 10000; i++){
            #pragma omp for
            for( j = 0; j < N; j++){
                [j] = y[j] + z[j];
            }
        }
    }
...
Native execution with OpenMP

Compile and run on host:

```
user@host icc -openmp vector_omp.c -o vector_omp.cpu
user@host export OMP_NUM_THREADS=20 ./vector_omp.cpu
Affinity exercise completed with 20 threads.
Validation: x[N-1] = 0.88 and it should be 0.88
```

Compile and run on the device:

```
user@host icc -mmic vector_omp.c -o vector_omp.mic
user@host export SINK_LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH
user@host micnativeloadex vector_omp.mic -d 0 -e
    "OMP_NUM_THREADS=120"
Affinity exercise completed with 120 threads.
Validation: x[N-1] = 0.88 and it should be 0.88
```

```
user@host scp vector_omp.mic mic0:~
user@host ssh mic0
user@host export LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH
user@host export OMP_NUM_THREADS=120 ./vector_omp.mic
```
Native Execution - Vectorization

Compile and run on the device:

```bash
user@host icc -mmic -openmp -O3 -vec-report3 vector_omp.c \ -o vector_omp.vec
user@host icc -mmic -openmp -opt-report-phase=vec \ vector_omp.c -o vector_omp.vec

user@host icc -mmic -openmp -no-vec -vec-report3 vector_omp.c -o vector_omp.novec

user@host export SINK_LD_LIBRARY_PATH= \ $MIC_LD_LIBRARY_PATH
user@host micnativeloadex vector_omp.vec -d 0 -e \ "OMP_NUM_THREADS=120"
```
Summary for Native mode

- Add flag `-mmic` to create MIC binary files.
- `ssh` to MIC `/micnativeloadex` to execute MIC binary natively.
- Vectorization is critical.
- Monitor MIC performance with `micsmc`. 
Offload Execution

Compiler Assisted Offload

- Explicit
  - Programmer explicitly directs data movement and code execution

- Implicit
  - Programmer marks some data as “shared” in the virtual sense
  - Runtime automatically synchronizes values between host and MIC
Explicit offload

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

int main( void ) {

    int totalProcs;
    #pragma offload target(mic){
        totalProcs = omp_get_num_procs();
        printf( "procs: %d\n", totalProcs );
    }
    return 0;
}
```

```f90
program main

    use omp_lib

    integer :: nprocs

    !dir$ offload target(mic)
    nprocs = omp_get_num_procs()
    print*, "procs: ", nprocs
    !dir$ end offload

end program
```
Explicit offload

Compile and run from the host:

```bash
user@host icc -openmp off02block.c -o off02block-c
user@host export MIC_OMP_NUM_THREADS=20 OFFLOAD_REPORT=2
user@host ./off02block-c

user@host ifort -openmp off02block.f90 -o off02block-f
user@host MIC_OMP_NUM_THREADS=20 OFFLOAD_REPORT=2 \ 
    ./off02block-f
```
Explicit offload functions/variables

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

__attribute__((target(mic))) int __attribute__((target(mic))) successor(int m);
__attribute__((target(mic))) void increment(int* m);

int main(void)
{
    #pragma offload target(mic)
    {
        i = successor(123);
        increment(&i);
    }

    return 0;
}
```

```fortran
module utils
contains
    subroutine increment(m)
    ..
    end subroutine increment
end module utils

integer function successor(m)
...
end function successor

program main
...
end program main
```

```fortran
!dir$ attributes offload:mic :: successor
integer function successor(m)
...
end function successor
program main
!dir$ attributes offload:mic :: successor
!dir$ offload target(mic)
i = successor(123)

!dir$ offload target(mic)
call increment(i)
```
Explicit offload functions/variables

Compile and run from the host:

```
user@host icc -opt-report-phase=offload off04proc.c \
        -o off04proc-C
user@host export OFFLOAD_REPORT=2
user@host ./off04proc-C

user@host ifort off04proc.f90 -o off04proc-f
user@host OFFLOAD_REPORT=2 ./off04proc-f
```
Explicit offload OpenMP

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

int main( void ) {
    ...
    #pragma offload target(mic:0) {
        #pragma omp parallel for
        for ( i=0; i<500000; i++ )
            a[i] = (double)i;
    }
    printf( "\n\t last val = %f \n", \
            a[500000-1]);
    return 0;
    ...
}
```

---

```fortran
program off03omp.f90
main
    use omp_lib
    !dir$ offload target(mic)
    !$omp parallel do
        do i=1,N
            a(i) = real(i)
        end do
    !$omp end parallel do
    print*, "last val is ", a(N)
end program
```
Explicit offload OpenMP

Compile and run from the host:

```
user@host icc -openmp off3omp.c -o off3omp

user@host export MIC_OMP_NUM_THREADS=120 ./off03omp
```
Controlling the data transfer

Additional clauses, attributes, specifiers and keywords give the programmer a high degree of control over all steps in the process.
Explicit offload – static data transfer

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

int main( void ) {
    double a[100000], b[100000], c[100000],
           d[100000];

    #pragma offload target(mic:0) in(a) out(c,d)
    inout(b)

    #pragma omp parallel for
    for ( i=0; i<100000; i++){
        c[i] =  a[i] + b[i];
        d[i] =  a[i] - b[i];
        b[i] = -b[i];
    }

    return 0;
}
```

```fortran
program main

integer, parameter :: N = 100000
Constant real :: a(N), b(N), c(N), d(N) ! on stack

!dir$ offload target(mic) in(a),out(c,d), \ inout( b )

!$omp parallel do
    do i=1,N
        c(i) =  a(i) + b(i)
        d(i) =  a(i) - b(i)
        b(i) = -b(i)
    end do
!$omp end parallel do

end program
```
Explicit offload – dynamic data transfer

Compile and run from the host:

```
user@host icc -openmp off06stack.c -o off06stack
user@host OFFLOAD_REPORT=3 ./off06stack
```

- Data transfer efficiency is critical for offload execution
- Compare performance without data management
Explicit offload – dynamic data transfer

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

int main( void ) {
    a = (double*)memalign(64, N*sizeof(double));
    b = (double*)memalign(64, N*sizeof(double));
    #pragma offload target(mic) \\ in( a : length(N) alloc_if(1) \ free_if(1) ), \\ out( b: length(N) alloc_if(1) \ free_if(1) )
    #pragma omp parallel for
    for ( i=0; i<N; i++ ) {
        b[i] = 2.0 * a[i];
    }
    ...}

program main
real, allocatable :: a(:,), b(:,)
!dir$ attributes align:64 :: a
!dir$ attributes align:64 :: b
integer, parameter :: N = 5000000
integer :: i
allocate( a(N), b(N) )
!dir$ offload target(mic) &
    in( a : alloc_if(.true.) \ free_if(.true.) ), &
    out( b : alloc_if(.true.) \ free_if(.true.) )
!$omp parallel do
    do i=1,N
        b(i) = 2.0 * a(i)
    end do
!$omp end parallel do
end program
Explicit offload – Data-only offload

```c
#include <stdio.h>
#include <omp.h>
int main( void ){
  // allocate memory on mic
  #pragma offload_transfer target(mic:0) nocopy(a:length(N) \ 
    alloc_if(1) free_if(0)) nocopy( b : length(N) alloc_if(1)\ 
    free_if(0)) signal( &tag1 )
  for ( i=0; i<N; i++ ){
    a[i] = (double)(i);
  }
  // after tag1 is finished, copy a from host to mic, calculate
  //on mic, copy b from mic to host
  #pragma offload target(mic:0) wait(&tag1)in(a :length(N) \ 
    alloc_if(0) free_if(0) ) \ 
  out( b:length(N) alloc_if(0) free_if(0)) signal( &tag2 )
  #pragma omp parallel for
  for ( i=0; i<N; i++ ){
    b[i] = 2.0 * a[i];
  }
}...
```
Asynchronous offload

```c
main() {
    #pragma offload target(mic)
    CPU waits while MIC busy
}
```

Coprocessor:

```c
myFunction();
```
Asynchronous offload

```c
int n = 123;

#pragma offload target(mic:0) signal( &n )
    incrementSlowly( &n );
//CPU do something here while MIC is busy...
#pragma offload target(mic:0) wait( &n )
{
    printf( "\n\ntprocs: %d\n", omp_get_num_procs() );
    fflush(0);
}
printf( "\n\tn = %d \n",
```

```f90
integer :: n = 123
!
dir$ offload begin target(mic:0) signal( n )
    call incrementslowly( n )
!dir$ end offload
//CPU works here while MIC is busy...
!dir$ offload begin target(mic:0) wait( n )
    print *, " procs: ", omp_get_num_procs()
    call flush(0)
!dir$ end offload
print *, " n: ", n
```

10/21/2015
Introduction to Xeon Phi Programming
Asynchronous offload

Compile and run from the host:

```
user@host icc -openmp off08asynch.c -o off08asynch
user@host ./off08asynch
```
Offload Execution

Compiler Assisted Offload

- Explicit
  - Programmer explicitly directs data movement and code execution

- Implicit
  - Programmer marks some data as “shared” in the virtual shared memory using _Cilk_shared
  - Runtime automatically synchronizes values between host and MIC
```c
#include <stdio.h>
#include <omp.h>

_Cilk_shared double sum;
_Cilk_shared double b[n];
_Cilk_shared double* _Cilk_shared A;

_Cilk_shared void multiply_then_add() {
    ...
}

int main( void ) {
    A = (_Cilk_shared double*) _Offload_shared_malloc(sizeof(double)*n*m);
    const int numDevices = _Offload_number_of_devices();
    for(int i=0; i<numDevices; i++)
        _Cilk_spawn _Cilk_offload_to(i) multiply_then_add();
    _Cilk_sync;
    ...
}
```
Implicit offload – _Cilk_shared

Compile and run from the host:

user@host icpc matrix_cilk.cc
user@host OFFLOAD_REPORT=3 ./a.out
MKL on Intel® Xeon Phi™ Coprocessors

**Intel® MKL is industry’s leading math library**

<table>
<thead>
<tr>
<th>Linear Algebra</th>
<th>Fast Fourier Transforms</th>
<th>Vector Math</th>
<th>Vector Random Number Generators</th>
<th>Summary Statistics</th>
<th>Data Fitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS</td>
<td>Multidimensional (up to 7D)</td>
<td>Trigonometric</td>
<td>Congruential</td>
<td>Kurtosis</td>
<td>Splines</td>
</tr>
<tr>
<td>LAPACK</td>
<td>FFTW interfaces</td>
<td>Hyperbolic</td>
<td>Recursive</td>
<td>Variation coefficient</td>
<td>Interpolation</td>
</tr>
<tr>
<td>Sparse solvers</td>
<td>Cluster FFT</td>
<td>Exponential, Logarithmic</td>
<td>Wichmann-Hill</td>
<td>Quantiles, order</td>
<td>Cell search</td>
</tr>
<tr>
<td>ScaLAPACK</td>
<td></td>
<td>Power / Root</td>
<td>MersenneTwister</td>
<td>statistics</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rounding</td>
<td>Sobol</td>
<td>Min/max</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-deterministic</td>
<td>Neiderreiter</td>
<td>Variance-covariance</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* 2011 & 2012 Evans Data N. American developer surveys
Intel® Xeon Phi™ Coprocessor vs. 2S Intel® Xeon® processor (Intel® MKL)

Using Intel® MKL

Higher is Better

Intel® Xeon Phi™ Coprocessor 7120P
VS.
2 Socket Intel® Xeon® processor (E5-2697v2)

Native = Benchmark run 100% on coprocessor. AO = Automatic Offload Function = Xeon + Xeon Phi together

10/21/2015

Introduction to Xeon Phi Programming
MKL Models on Intel® Xeon Phi™ Coprocessors

Automatic Offload
- No code changes required
- Automatically uses both host and target
- Transparent data transfer and execution management

Compiler Assisted Offload
- Explicit controls of data transfer and remote execution using compiler offload pragmas/directives
- Can be used together with Automatic Offload

Native Execution
- Uses the coprocessors as independent nodes
- Input data and binaries are copied to targets in advance
MKL - Automatic Offload (AO)

- Offloading is automatic and transparent.
- Can take advantage of multiple coprocessors.
- By default, Intel MKL decides:
  - When to offload
  - Work division between host and targets
- Users enjoy host and target parallelism automatically.
- Users can still specify work division between host and target. (for BLAS only)
How to use automatic offload

• Using Automatic Offload is easy:

Call a function:  
mkl_mic_enable()

Set an env variable:  
MKL_MIC_ENABLE=1

• What if there doesn’t exist a coprocessor in the system?
  – Runs on the host as usual **without penalty**!

• The context of Automatic Offload is a single function

• MKL routine decides how to divide workload among host and devices
How to use automatic Offload

```c
#include <mkl.h>
#include <omp.h>
int main(){

#pragma omp parallel for
for (int i = 0; i < n*n; i++) {
    A[i]=(double)i;
    B[i] = -(double)I;  C[i] = 0.0;
}

for (int trial = 1; trial <= nTrials; trial++) {
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
        n, n, n, 1.0, A, n, B, n, 0.0, C, n);
}
```

```
user@host icpc -openmp -mkl dgemm.cc
user@host MKL_MIC_ENABLE=0  OFFLOAD_REPORT=2  ./a.out
user@host MKL_MIC_ENABLE=1  OFFLOAD_REPORT=2  ./a.out

[MKL] [MIC --] [AO Function] DGEMM
[MKL] [MIC 00] [AO DGEMM CPU Time] 2.723867 seconds
[MKL] [MIC 01] [AO DGEMM MIC->CPU Data] 25920000 bytes
```
Automatic Offload Enabled Functions

- A selected set of MKL functions are AO enabled.
  - Only functions with sufficient computation to offset data transfer overhead are subject to AO
  - Level-3 BLAS: GEMM, TRSM, TRMM, SYMM
  - LAPACK 3 amigos: LU, QR, Cholesky
- Offloading happens only when matrix sizes are right.
  (dimension sizes in numbers of elements)
  - GEMM: M, N > 2048, K > 256
  - SYMM: M, N > 2048
  - TRSM/TRMM: M, N > 3072
  - LU: M, N > 8192
MKL - Native Execution

Use the coprocessor as an independent compute node.

- Programs can be built to run only on the coprocessor by using the
  \texttt{--mmic} build option.

```
user@host icpc -openmp -mkl -mmic dgemm.cc -o dgemm.mic
user@host export SINK_LD_LIBRARY_PATH=$MIC_LD_LIBRARY_PATH
user@host micnativeloadex ./dgemm.mic
```
Compiler Assisted Offload (CAO)

Offloading is explicitly controlled by compiler pragmas or directives.

All MKL functions can be offloaded in CAO.
  • In comparison, only a subset of MKL is subject to AO.

Can leverage the full potential of compiler’s offloading facility.

Can offload multiple MKL functions using one offload region.

More flexibility in data transfer and remote execution management.
  • A big advantage is data persistence: Reusing transferred data for multiple operations.
How to Use Compiler Assisted Offload

• The same way you would offload any function call to the coprocessor.

• An example in C:

```c
#pragma offload target(mic) \
  in(transa, transb, N, alpha, beta) \
  in(A:length(matrix_elements)) \
  in(B:length(matrix_elements)) \
  in(C:length(matrix_elements)) \
  out(C:length(matrix_elements) alloc_if(0))
{
  sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N, 
        &beta, C, &N);
}
```
How to Use Compiler Assisted Offload

• An example in Fortran:

```fortran
!DEC$ ATTRIBUTES OFFLOAD : TARGET( MIC ) :: SGEMM
!DEC$ OMP OFFLOAD TARGET( MIC ) &
!DEC$ IN( TRANSA, TRANSB, M, N, K, ALPHA, BETA, LDA, LDB, LDC ), &
!DEC$ IN( A: LENGTH( NCOLA * LDA ) ), &
!DEC$ IN( B: LENGTH( NCOLB * LDB ) ), &
!DEC$ INOUT( C: N * LDC )
!$OMP LENGTH( PARALLEL
!$OMP SECTIONS SECTION
   CALL SGEMM( TRANSA, TRANSB, M, N, K, ALPHA, &
               A, LDA, B, LDB BETA, C, LDC )
!$OMP END PARALLEL SECTIONS
```
Linking flags using MKL

AO: The same way of building code on Xeon! \(-\text{mkl}\)

\[
i\text{cc } -O3 -\text{mkl} \text{ sgemm.c } -o \text{ sgemm.exe}
\]

Native: Using \(-\text{mmic}\)

\[
i\text{cc } -O3 -\text{mmic} -\text{mkl} \text{ sgemm.c } -o \text{ sgemm.exe}
\]

CAO: Using \(-\text{offload-option}\)

\[
i\text{cc } -O3 -\text{openmp} -\text{mkl} -\text{offload-option},\text{mic},\text{ld},\ \ "-L$\text{MKLROOT}/\text{lib/mic} -Wl,\
-\text{start-group} -\text{lmlk_in tel}_\text{lp64} -\text{lmlk_in tel}_\text{thread} \ \ -\text{lmlk}_\text{core} -Wl,--end-group" \text{ sgemm.c } -o \text{ sgemm.exe}
\]
Intel® MKL Link Line Advisor

A web tool to help users to choose correct link line options.


Also available offline in the MKL product package.
Heterogeneous Distributed Computing with Xeon Phi

MPI (Message Passing Interface) for Inter-node operations

- Symmetric pure MPI (native mode)
- Symmetric hybrid MPI+OpenMP
- MPI with OpenMP Offload
Heterogeneous Distributed Computing with Xeon Phi
- Symmetric pure MPI (native mode)

- MPI processes on hosts
- Native MPI processes on coprocessors
- No OpenMP
Heterogeneous Distributed Computing with Xeon Phi
- Symmetric hybrid MPI+OpenMP

- MPI processes on hosts
- Native MPI processes on coprocessors
- Multi-threading with OpenMP
Heterogeneous Distributed Computing with Xeon Phi
- MPI with OpenMP Offload

- MPI processes are multi-threaded using OpenMP
- MPI runs only on CPUs
- MPI processes offload to Phi
- OpenMP in offload regions.
#include <stdio.h>
#include <omp.h>
double do_some_integratin(long long ns, int myrank, int np) {
    ...
    #pragma omp parallel private(iam, x, i, np){
        #pragma omp for schedule(static), reduction(+:sum)
        for(i=start_int; i<=end_int; i++) {
            x = h * ((double)i - 0.5);
            sum = sum + (4./(1. + x*x));}
    }
    ...
    int main( void ) {
        MPI_Init( &argc, &argv );
        MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
        MPI_Comm_size( MPI_COMM_WORLD, &np );

        ... mypi = do_some_integratin(ns, myrank, np);
        ...
        MPI_Finalize();
    }
Symmetric hybrid MPI+OpenMP

Compile and run using one node

user@host  qsub -I -A hpc_train_2015 -l walltime=3:00:00
            -l nodes=1:ppn=20

user@host  module load impi/4.1.3.048/intel64

user@host  mpiicc -openmp pi_hybrid.c -o pi_hybrid.cpu

user@host  mpiicc -openmp -mmic pi_hybrid.c -o pi_hybrid.mic

user@host  mpiexec.hydra -host localhost -n 2 ./pi_hybrid.cpu

user@host  mpiexec.hydra -host mic0 -n 2 -env LD_LIBRARY_PATH
            $MIC_LD_LIBRARY_PATH ./pi_hybrid.mic

user@host  micrun.sym -c ./pi_hybrid.cpu -m ./pi_hybrid.mic
Symmetric hybrid MPI+OpenMP

Job submission using >1 nodes

#!/bin/bash
#PBS -q checkpt
#PBS -A hpc_train_2015
#PBS -l walltime=00:03:00
#PBS -l nodes=2:ppn=20
#PBS -o test.out2
#PBS -e test.err2
module load impi/4.1.3.048/intel64

# ====== input parameters ======
export TASKS_PER_HOST=2  # number of MPI tasks per host
export THREADS_HOST=10  # number of OpenMP threads spawned by each task on the host
export TASKS_PER_MIC=3  # number of MPI tasks per MIC
export THREADS_MIC=80  # number of OpenMP threads spawned by each task on the MIC

cd $PBS_O_WORKDIR

micrun.sym -c ./pi_hybrid.cpu -m ./pi_hybrid.mic
#include <stdio.h>
#include <omp.h>

double do_some_integratin(long long nsize, int myrank, int nprocs)
{
    #pragma offload target(mic:myrank)
in(start_int, end_int) out(sum)
    #pragma omp parallel private(iam, x, i, np)
    {
        ...
        #pragma omp parallel private(iam, x, i, np)
        {
            #pragma omp for schedule(static), reduction(+:sum)
            for (i=start_int; i<=end_int; i++) {
                x = h * ((double)i - 0.5);
                sum = sum + (4./(1. + x*x));}
        }
    }

    int main( void ) {
        MPI_Init( &argc, &argv );
        MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
        MPI_Comm_size( MPI_COMM_WORLD, &nprocs );
        ...
        mypi = do_some_integratin(nsize, myrank, nprocs);
        ...
        MPI_Finalize();
    }
MPI with OpenMP offload

Compile and run using one node

```
user@host  qsub  -I -A hpc_train_2015  -l walltime=3:00:00
          -l nodes=1:ppn=20

user@host  module load  impi/4.1.3.048/intel64

user@host  mpiicc  -openmp  pi_hybrid_off.c

user@host  OFFLOAD_REPORT=4  ./a.out

user@host  mpiexec.hydra  -host localhost  -n 2  ./a.out
```
Optimization and Tuning

- Single core optimization
  - Memory alignment
  - SIMD optimization
- OpenMP optimization
  - Thread affinity
  - False sharing
  - Nested parallelism
  - Load balancing
- Performance tools
  - Visual profiler, VTune Amplifier
Summary

• Programming models on MIC:
  • Native mode -mmic
  • Offloading
    • Explicit offload
    • Implicit offload

• Intel MKL
  • Automatic offload
  • Native mode
  • Compiler assistant offload

• Heterogeneous Distributed Computing on MIC architecture – MPI
  • Symmetric and hybrid offload