
intel
SC16
FUEL YOUR INSIGHT

Intel® Compiler/Runtime Support for OpenMP Offloading

Rakesh Krishnaiyer

(Principal Engineer, Intel Compiler Lab)

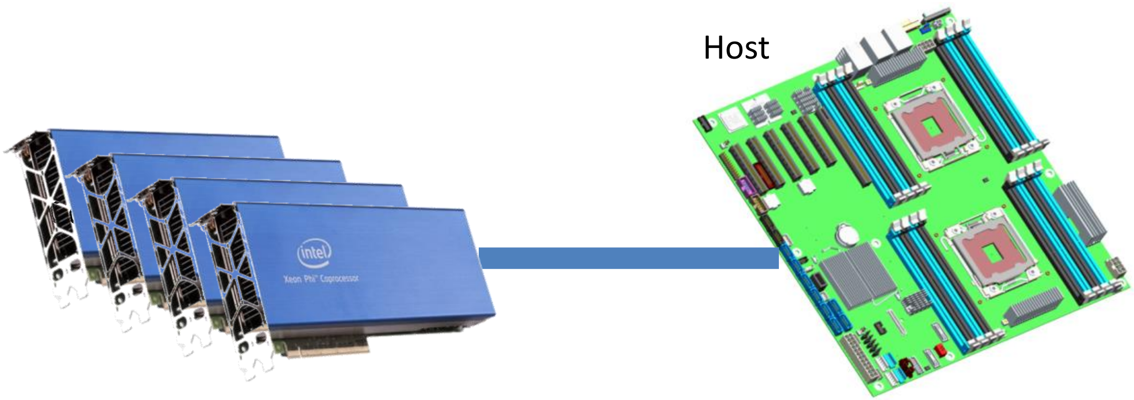
A Motivating OpenMP Example

```
#pragma omp declare target
#pragma omp declare simd simdlen(16)
uint32_t mandel(fcomplex c)
{   // Computes number of iterations(count variable) that it takes
    // for parameter c to be known to be outside mandelbrot set
    uint32_t count = 1; fcomplex z = c;
    for (int32_t i = 0; i < max_iter; i += 1) {
        z = z * z + c;
        int t = cabsf(z) < 2.0f;
        count += t;
        if (!t) { break;}
    }
    return count;
}
```

```
    #pragam omp target device(0) map(to:in_vals) map(from:count)
    #pragma omp parallel for schedule(guided)
    for (int32_t y = 0; y < ImageHeight; ++y) {
        #pragma omp simd safelen(16)
        for(int32_t x = 0; x < ImageWidth; ++x) {
            count[y][x] = mandel(in_vals[y][x]);
        }
    }
```

Device Model

- OpenMP supports accelerators and coprocessors
- Device model:
  - One host
  - Multiple accelerators/coprocessors of the same kind

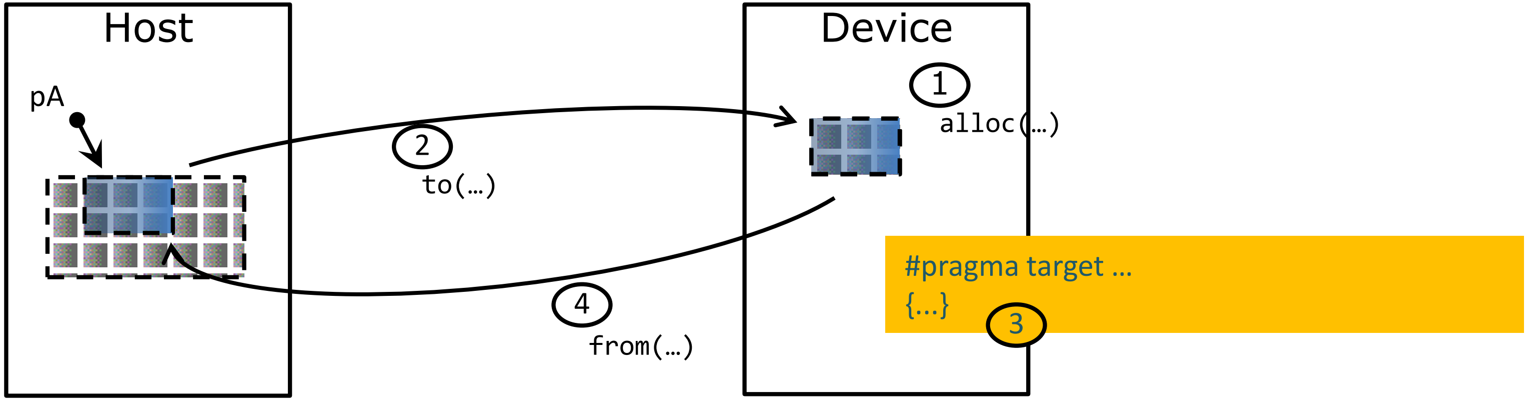


OpenMP Data Environment Examples

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

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

    do_some_other_stuff_on_host();

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

Execution Model



- The `target` construct transfers the control flow to the target device
  - The transfer clauses control direction of data flow
  - Array notation is used to describe array length
- The `target data` construct creates a *scoped* device data environment
  - The transfer clauses control direction of data flow
  - Device data environment is valid through the lifetime of the `target data` region
- Use `target update` to request data transfers from within a `target data` region

Offloading: RTM-Stencil Example

```
#pragma omp declare target
void do_row(float, float, int);
#pragma omp end declare target

void team_distribute_stencil(int sweeps) {
    for (int t=0; t<sweeps; ++t) {
        // Target region starts with 68 teams of threads, each team has 4 threads
    #pragma omp target device(0) map(tofrom: V[:])
    #pragma omp teams firstprivate(t) num_teams(68) thread_limit(4)
     {  // The k loop is blocked by hand, and each block handed to a team
         int p = omp_get_num_teams();
         int ib = (N-2+p-1)/p;
         #pragma omp distribute
        for (int k=1; k<N-1; k+=ib) {
                int u = std::min(k+ib,N-1);
                #pragma omp parallel for
               for( int i=k; i<u; ++i) do_row(V[1-(t&1)], V[t&1], i);
         }
     }
   }
}
```

Intel® Xeon Phi™ OMP 4.5 Offload in 17.0 Compiler

OpenMP 4.5 “if” clause changes –

if ([*directive-name-modifier* :] scalar-expression)

OpenMP 4.5 Device pointer -

- #pragma omp target data … use_device_ptr(list) ..
- #pragma omp target … is_device_ptr(list) ..

OpenMP 4.5 Deferred Map

#pragma omp declare target clause[[,] clause] …] new-line

where *clause* is one of the following:

[to] **(** *extended-list* **)**

link (*list* **)**

Streams additions - #pragma offload

OpenMP 4.5 Combined Constructs –

#pragma omp target parallel

#pragma omp target parallel for

#pragma omp target parallel for simd

#pragma omp target simd

OpenMP 4.5 Clause changes–

#pragma omp target

private (*list* **)**
firstprivate (*list* **)**
defaultmap (tofrom : scalar)
is_device_ptr (*list* **)**

#pragma omp target data

use_device_ptr (*list* **)**

Intel® Xeon Phi™ Processor Programming Models

Server (KNL)

1. Native programming
   - Intel® Xeon Phi™ server is a standalone machine

2. MPI + OpenMP
   - Intel® Xeon Phi™ server is a computing node

3. Compiler Offload
   - Migration path for compiler offload users
   - Uses fabric instead of PCIe as transportation (OOF)

Coprocessor (KNC and KNL)

1. Native programming
   – Intel® Xeon Phi™ coprocessor as a standalone machine

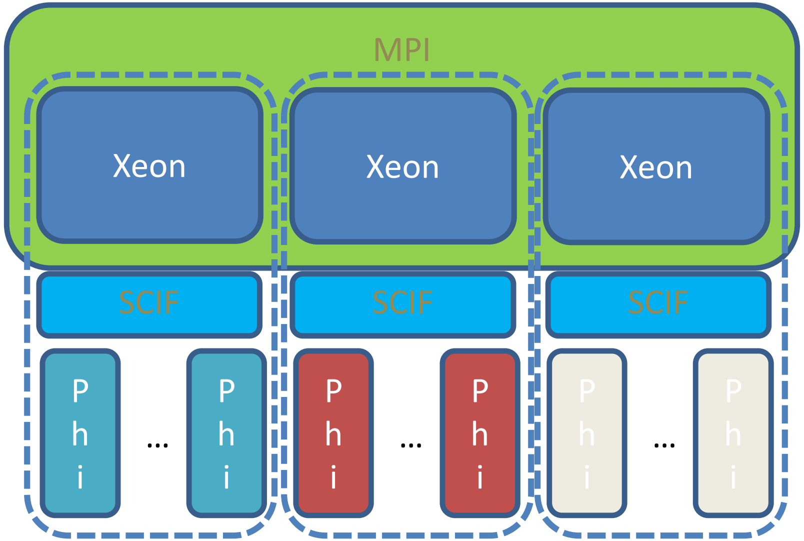
2. MPI + OpenMP
   – Intel® Xeon Phi™ coprocessor as computing node

3. Compiler Offload
   – Intel® Xeon Phi™ coprocessor as offload target

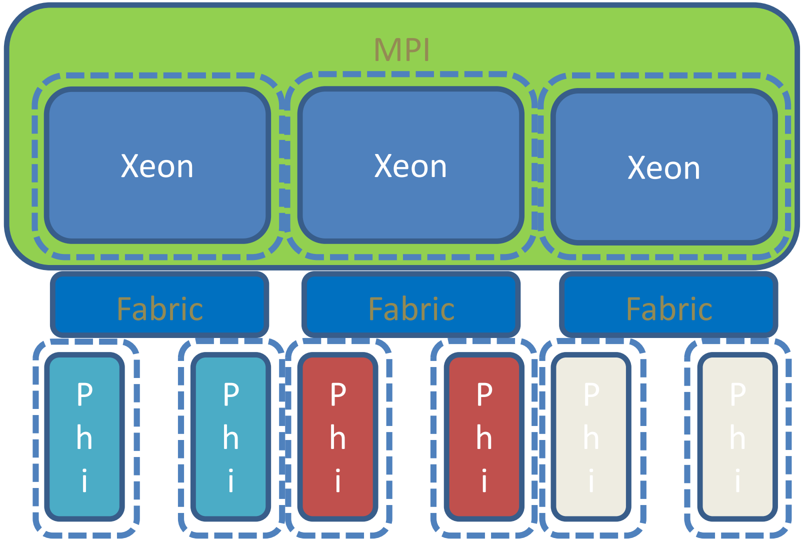
Offload Over Fabric - OOF

- host boundaries

Offloading to Intel® Xeon Phi™ coprocessors

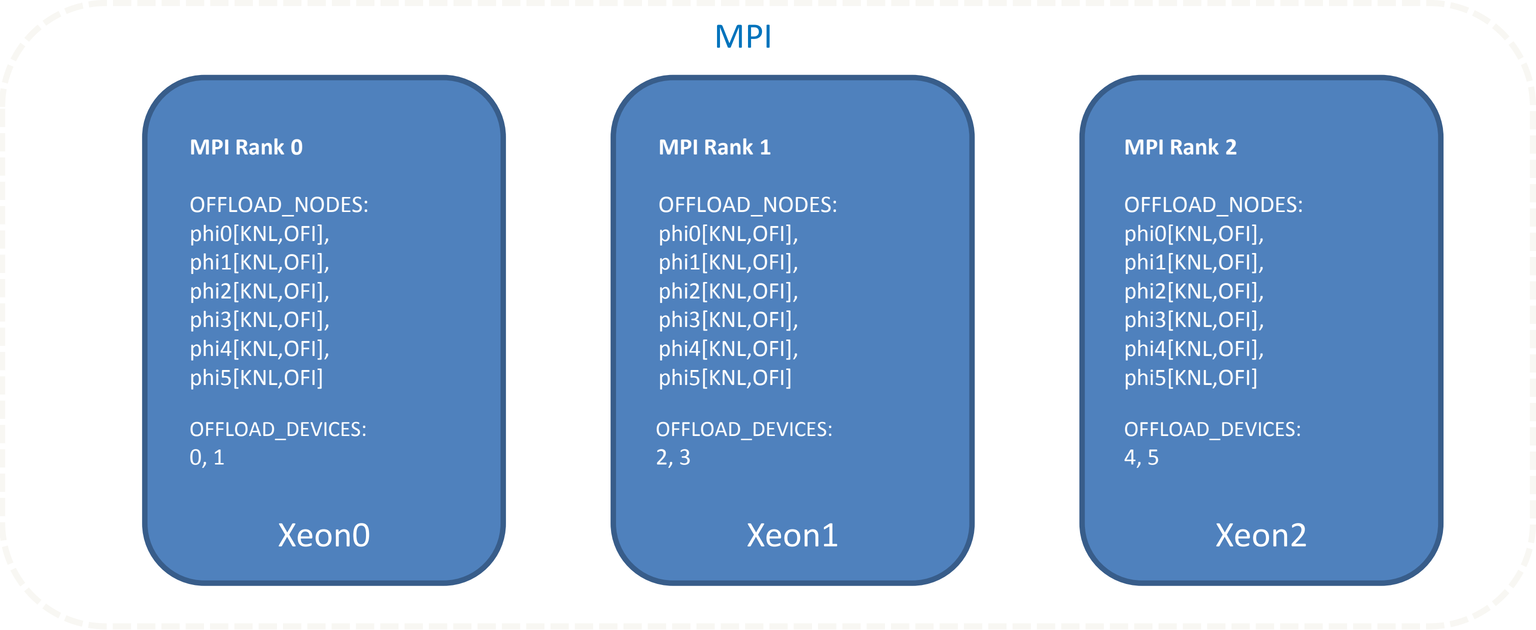


Offloading to Intel® Xeon Phi™ x200 (KNL) Server nodes



Common code for both cases. MPI ranks use pragmas to offload highly parallel tasks to Intel® Xeon Phi™ nodes
Configured by set of environmental variables (discussed on following slides)

OOF Application Configuration



- OFFLOAD_NODES have the same value on all Intel® Xeon™ servers
- OFFLOAD_DEVICES restricts the offload process to use only the “devices” specified as the value of the variable

Offload Compile/Run Using 17.0

- Compilation using 17.0 Compiler:
  - Offloading to Intel® Xeon Phi™ x100 (KNC):
    - icc –offload-arch=mic …
  - Offloading to Intel® Xeon Phi™ x200 (KNL):
    - icc –offload-arch=mic-avx512 …
    - For OOF, additionally specify the KNL devices using OFFLOAD_NODES:
    - export OFFLOAD_NODES="machine1, machine2, 10.2.100.5" OR
    - export OFFLOAD_NODES="machine3[x200,OFI]" OR …
- Programs written for KNL OOF and PCI-card offload will run unchanged
  - Offload program initiated from a Xeon node

OMP4.5: Unstructured Data Mapping Example

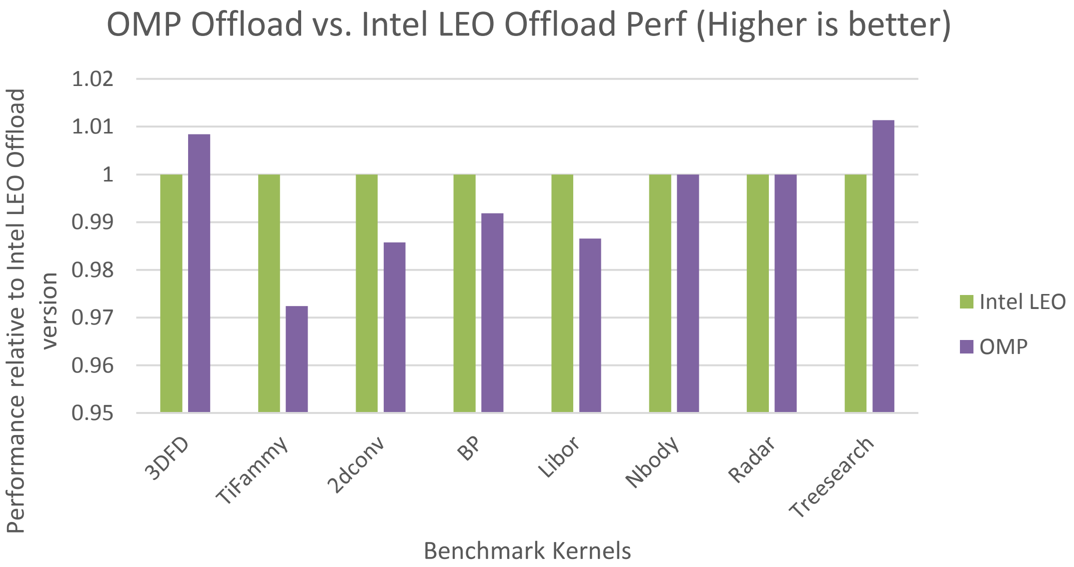
```
double A[N];
foo()
{
    …
    #pragma omp target enter data map(to : A)
        device_work1(A); // A is used here
}
bar()
{
    …
    #pragma omp target exit data map(from: A)
        device_work3(A); // A is updated here
}
try1()
{
    foo();
    #pragma omp target
        device_work2(A);
    bar();
}
```

OMP4.5: nowait clause on target

```
#define N 1000000 //N must be even
void init(int n, float *v1, float *v2);

int main() {
    int i, n=N;     int chunk=1000;    float v1[N],v2[N],vxv[N];
    init(n, v1,v2);

   #pragma omp parallel
  {
    #pragma omp master
    #pragma omp target teams distribute parallel for nowait \
        map(to: v1[0:n/2]) \
        map(to: v2[0:n/2]) \
        map(from: vxv[0:n/2])
    for(i=0; i<n/2; i++){ vxv[i] = v1[i]*v2[i]; } // Computation on target

    #pragma omp for schedule(dynamic,chunk)
    for(i=n/2; i<n; i++){ vxv[i] = v1[i]*v2[i]; }  // Simultaneous execution on host
    // Implicit barrier here ensures both host and target computations are done
   }
 printf(" vxv[0] vxv[n-1] %f %f\n", vxv[0], vxv[n-1]);
 return 0;
}
```

- Product of two vectors (arrays), *v1* and *v2*, is formed. One half of the operations is performed on the device, and the last half on the host, concurrently

OMP vs Intel-pragma-offload : Performance Comparison



- Performance within 2-3% of each other – very close
- Compiler implementation maps both to similar IR
- Same runtime used for both
- Data collected on Intel® Xeon Phi™ x100 (KNC)

KNC Used for Gordon Bell Submission

- **SeisSol Gordon Bell Nomination paper in SC14:** ***Petascale high order dynamic rupture earthquake simulations on heterogeneous supercomputers (Alexander Heinecke et. al.)***
  - Breakthrough research project that is furthering understanding of earthquakes by using numerical simulation of the propagation of seismic waves helps to understand complicated wave phenomena
- ***"Finally, we discuss how offloading computations to Xeon Phi is incorporated into SeisSol. Adding support for accelerators or coprocessors often requires major code changes. These changes are essential for performance reasons as functions (such as the wave propagation solvers) need to be rewritten matching specific languages or libraries for the targeted device. In contrast, we used the Intel Language Extensions for Offloading (LEO), which include pragma directives for (a)synchronous data transfers and program execution on the Xeon Phi device. Down to the matrix kernels covered in Sec. III-C, there is no need to re-implement the time, volume or flux kernel. Their CPU implementation, including an OpenMP parallelization, can be directly called inside the code section annotated for offload. Furthermore, we captured all interactions with the coprocessor and the launch of compute tasks shown in Fig. 4 in separated static functions. These functions are called from SeisSol's time marching loop ensuring a smooth integration into SeisSol's code base …"***
- **http://insidehpc.com/2014/11/seismic-code-modernization-yields-petascale-performance-gordon-bell-award-nomination/**

Legal Disclaimer & Optimization Notice

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Stop by at the Intel Booth to learn more about Intel® Xeon Phi™

intel®

@IntelHPC | intel.com/SC16

OMP4.5: Asynchronous target with Tasks

```
#pragma omp declare target
      extern void init(float *, float *, int);
#pragma omp end declare target

extern void foo();      extern void output(float *, int);

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

 #pragma omp task shared(v1, v2) depend(out: v1, v2)
 #pragma omp target device(dev) map(v1, v2)
 {
        // check whether on device dev
      if (omp_is_initial_device())  abort();    // initial_device means host

      v1 = malloc(N*sizeof(float));
      v2 = malloc(N*sizeof(float));
      init(v1, v2, N);
 }

 foo(); // execute other work asynchronously on host

 #pragma omp task shared(v1, v2, p) depend(in: v1, v2)
 #pragma omp target device(dev) map(to: v1, v2) map(from: p[0:N])
 {
      // check whether on device dev
      if (omp_is_initial_device())  abort();

     #pragma omp parallel for
     for (i=0; i<N; i++)
        p[i] = v1[i] * v2[i];

     free(v1);     free(v2);
}

#pragma omp taskwait

 output(p, N);
}
```

Terminology

- Device:
  an implementation-defined (logical) execution unit
- League:
  the set of threads teams created by a `teams` construct
- Contention group:
  threads of a team in a league and their descendant threads
- Device data environment:
  Data environment as defined by `target data` or `target` constructs
- Mapped variable:
  An original variable in a (host) data environment with a corresponding variable in a device data environment
- Mapable type:
  A type that is amenable for mapped variables.